ADVANCES IN INFORMATION SECURITY

Insider Attack and Cyber Security

Beyond the Hacker

Edited by

Salvatore J. Stolfo
Steven M. Bellovin
Shlomo Hershkop
Angelos Keromytis
Sara Sinclair
Sean W. Smith

Springer

Insider Attack and Cyber Security

Beyond the Hacker

Advances in Information Security

Sushil Jajodia
Consulting Editor
Center for Secure Information Systems
George Mason University
Fairfax, VA 22030-4444
email: jajodia@gmu.edu

The goals of the Springer International Series on ADVANCES IN INFORMATION SECURITY are, one, to establish the state of the art of, and set the course for future research in information security and, two, to serve as a central reference source for advanced and timely topics in information security research and development. The scope of this series includes all aspects of computer and network security and related areas such as fault tolerance and software assurance.

ADVANCES IN INFORMATION SECURITY aims to publish thorough and cohesive overviews of specific topics in information security, as well as works that are larger in scope or that contain more detailed background information than can be accommodated in shorter survey articles. The series also serves as a forum for topics that may not have reached a level of maturity to warrant a comprehensive textbook treatment.

Researchers, as well as developers, are encouraged to contact Professor Sushil Jajodia with ideas for books under this series.

Additional titles in the series:

INTRUSION DETECTION SYSTEMS edited by Robert Di Pietro and Luigi V. Mancini; ISBN: 978-0-387-77265-3
VULNERABILITY ANALYSIS AND DEFENSE FOR THE INTERNET edited by Abhishek Singh; ISBN: 978-0-387-74389-9
BOTNET DETECTION: Countering the Largest Security Threat edited by Wenke Lee, Cliff Wang and David Dagon; ISBN: 978-0-387-68766-7
PRIVACY-RESPECTING INTRUSION DETECTION by Ulrich Flegel; ISBN: 978-0-387-68254-9
SYNCHRONIZING INTERNET PROTOCOL SECURITY (SIPSec) by Charles A. Shoniregun; ISBN: 978-0-387-32724-2
SECURE DATA MANAGEMENT IN DECENTRALIZED SYSTEMS edited by Ting Yu and Sushil Jajodia; ISBN: 978-0-387-27694-6
NETWORK SECURITY POLICIES AND PROCEDURES by Douglas W. Frye; ISBN: 0-387-30937-3
DATA WAREHOUSING AND DATA MINING TECHNIQUES FOR CYBER SECURITY by Anoop Singhal; ISBN: 978-0-387-26409-7
SECURE LOCALIZATION AND TIME SYNCHRONIZATION FOR WIRELESS SENSOR AND AD HOC NETWORKS edited by Radha Poovendran, Cliff Wang, and Sumit Roy; ISBN: 0-387-32721-5
PRESERVING PRIVACY IN ON-LINE ANALYTICAL PROCESSING (OLAP) by Lingyu Wang, Sushil Jajodia and Duminda Wijesekera; ISBN: 978-0-387-46273-8
SECURITY FOR WIRELESS SENSOR NETWORKS by Donggang Liu and Peng Ning; ISBN: 978-0-387-32723-5
MALWARE DETECTION edited by Somesh Jha, Cliff Wang, Mihai Christodorescu, Dawn Song, and Douglas Maughan; ISBN: 978-0-387-32720-4

Additional information about this series can be obtained from http://www.springer.com

Insider Attack and Cyber Security

Beyond the Hacker

by

Salvatore J. Stolfo
Steven M. Bellovin
Shlomo Hershkop
Angelos D. Keromytis
Columbia University, USA

and

Sara Sinclair
Sean W. Smith
Dartmouth College, USA

Springer

Editors:
Salvatore J. Stolfo
Steven M. Bellovin
Angelos D. Keromytis
Shlomo Hershkop
Columbia University
Department of Computer Science
1214 Amsterdam Avenue MC 0401
New York, NY 10027-7003 USA

Sean W. Smith
Sara Sinclair
Department of Computer Science
Dartmouth College
6211 Sudikoff Laboratory
Hanover, NH 03755-3510 USA

Series Editor:
Sushil Jajodia
George Mason University
Center for Secure Information Systems
4400 University Drive
Fairfax VA 22030-4444, USA
jajodia@gmu.edu

Library of Congress Control Number: 2008921346
ISBN-13: 978-0-387-77321-6
e-ISBN-13: 978-0-387-77322-3
Advances in Information Security series: Volume 39
Printed on acid-free paper.

The "Big Picture" of Insider IT Sabotage Across U.S. Critical Infrastructures by Andrew P. Moore, Dawn M. Cappelli, and Randall F. Trzeciak, Copyright 2007 Carnegie Mellon University is printed with special permission from the Software Engineering Institute.
CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

On behalf of the Organizing Committee, I am pleased to present to you the proceedings of the first Workshop on Insider Attack and Cyber Security held in Washington DC in June 2007. This book serves to educate all interested parties in academia, government and industry and that helps set an agenda for an ongoing research initiative to solve one of the most vexing problems encountered in securing our critical IT infrastructure, the insider threat. In some sense, the insider problem is the ultimate security problem. Insider threats, awareness and dealing with nefarious human activities in a manner that respects individual liberties, and privacy policies of organizations, while providing the best protection of critical resources and services that may be subjected to insider attack, is a very hard problem requiring a substantial effort by a large research community. We hope this book helps establish a community of researchers focused on addressing the insider problem.

The book contains a number of invited papers authored by attendees of the workshop. We believe the material that has been selected is of wide interest to the security research community. Papers have been invited that help define the nature and scope of the insider attack problem. Several papers provide an overview of technical solutions that have been proposed and discuss how they fail to solve the problem in its entirety. An essential theme of the workshop was to educate researchers as to the true nature of the problem in real-world settings. Papers are provided that describe the nature and scope of the insider problem as viewed by the financial industry. The book concludes with technical and legal challenges facing researchers who study and propose solutions to mitigate insider attacks.

We wish to thank Cliff Wang of the Army Research Office, Daniel Schutzer of the Financial Services Technology Consortium and Eric Goetz of the Institute for Information Infrastructure Protection for supporting our effort and sponsoring the Workshop, and Shari Pfleeger of Rand Corporation for providing the venue for our meeting and assistance in organizing the Workshop. We also thank the reviewers who served anonymously to vet the technical papers included here. Finally, we are especially grateful to Shlomo Hershkop and Sara Sinclair for their remarkable effort to organize and format the individual papers to produce a final cohesive manuscript.

January 2008 Salvatore J. Stolfo

Table of Contents

The Insider Attack Problem Nature and Scope

Steven M. Bellovin

Computer Science Department, Columbia University

1 Introduction

Hackers, especially "terrorist hackers" or "cyberwar hackers" get lots of press. They do indeed pose a serious problem. However, the threat they pose pales before that posed by those closest to us: the insiders.

The cyberthreat posed by insiders isn't new. Donn Parker's seminal 1978 book *Crime by Computer* estimated that 95% of computer attacks were committed by authorized users of the system. Admittedly, this was in the pre-Internet era, when very few non-insiders had any access at all; still, the underlying issue – that employees are not always trustable – remains. To be sure, this has always been true – thieving or otherwise corrupt workers have undoubtedly existed since commerce itself – but the power of computers (and our inability to secure them in the best of circumstances) makes the problem far worse today.

In June 2007, a workshop (sponsored by Cliff Wang of the Army Research Office) on the insider threat was held. Approximately 35 invitees attended, including security researchers, vendors, practitioners, and representatives of organizations that perceive a serious insider threat. The goal was to develop a research community on the insider threat. Of necessity, our first steps were to understand the scope of the problem, to develop a common vocabulary, and to start sketching a research agenda. This volume consists of papers contributed by some of those attendees.

2 Types of Attack

Fundamentally, there are three different types of attack: misuse of access, defense bypass, and access control failure. Each must be approached differently.

2.1 Misuse of Access

Misuse of legitimate access privileges is probably the hardest form of attack to detect or counter. In it, an insider uses his or her legitimate access rights for the wrong reason. In a university, for example, professors have the right to submit grade change requests after the semester is over. Typically, this is done to correct clerical errors or to deal with other unusual situations. The same action, if done in response to a bribe, would constitute insider misbehavior.

It is not possible to prevent or detect misuse by purely technical means, except in special situations. Generally speaking, the most that can be done is monitoring for unusual patterns or quantities of requests. Detailed logging can be useful if the person falls under suspicion for other reasons.

In some environments, such as the intelligence community, external data can be combined with technical analyses to detect abuse. For example, financial records, spending patterns, etc., can be examined to detect inappropriate sources of income. (Such data can also be missed. The CIA never noticed that Aldrich Ames drove a car that cost more than his annual salary.)

2.2 Defense Bypass

Insiders generally have a major inherent advantage over outsiders: they're already past some defense layers. For example, many companies rely on firewalls as part of their cybersecurity. More or less by definition, insiders are on the inside of the firewall; they are thus not blocked by it. Similarly, insiders generally have some sort of login access to an organization's computer systems; this permits local attacks, rather than only attacks against network services.

Again, it is hard to conceive of purely technical defenses. Insiders, by definition, *are* inside; they thus have more opportunities to commit mischief. Detection mechanisms can work well; in particular, they can look for either anomalous behavior or actual attacks on nominally-protected systems.

2.3 Access Control Failure

By contrast, access control failures represent a technical problem. Either an access control mechanism is buggy or the system has been configured improperly. Either way, the preferred solution is to correct the problem.

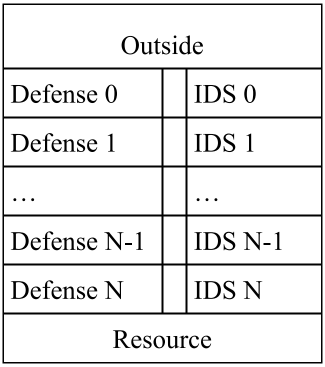
Ironically, detection is often more difficult, especially where a configuration error is involved, since by definition the system is not rejecting improper access requests. The best solutions involve looking for anomalous behavior by other applications.

3 Defend or Detect

There are two fundamentally different approaches to dealing with insider attacks: defend against them, or detect them after the fact. While defense is generally preferable, it isn't clear that it is always feasible. Only one of the possible attack types – access control failures – can be defended against in any strong sense.

It is tempting to say that the same is true for all attacks, by insiders or outsiders. Examination of the attack taxonomy shows that this assertion is false. By definition, insiders have more access; this is the essence of their status, their responsibilities – and their ability to launch attacks.

Another way to look at it is to consider system defenses schematically. Assume, as is generally the case, that the resource is protected by defense in depth. That is, there are *N* (more or less) independent defense layers. Further assume that each such layer consists of a combination of technical measures and an intrusion detection system tailored for that layer. The system then looks like this:



Outside		
Defense 0		IDS 0
Defense 1		IDS 1
…		…
Defense N-1		IDS N-1
Defense N		IDS N
Resource		

Fig. 1. Layering of system defenses.

An outsider must penetrate all *N* layers. Insiders, though, have fewer layers to penetrate. Their task is thus strictly easier than that of an outside attacker. This is, of course, the definition of our second class of attack. Second, authorized users, whether behaving properly or not, *of necessity* have access rights that let them penetrate all *N* layers.

It is clear, then, that technical defenses alone are insufficient. Even in principle, the only possible mechanism is intrusion detection. (Depending on the goals of the attackers, even IDS systems closer to the outside may be fruitful. In particular, if the goal is to exfiltrate sensitive data, this can be detected at any point between the inside and the outside.)

4 The Role of Process

In some circumstances, a combination of procedural and technical mechanisms can be employed as an additional mechanism to prevent or detect misuse of access attacks. Specifically, the ability to perform certain actions can be limited so that no one person can do them alone. Alternatively, manual audit checks can detect certain kinds of abuse after the fact.

Both of these ideas are rooted in old manual processes. Large checks have long required two signatures, certain cash register operations can only be done by supervisors, ordinary accounting audits can detect fraud, etc. The same can be done in computer systems: certain transactions can be blocked until requested by two different individuals.

Note that this does not contradict our assertion that there are no technical defenses against misuse of access attacks. If two person control is employed, a single individual does not have certain access rights. Furthermore, protection is provided by a combination of policy and technical defenses.

5 Conclusion

Defending against insider attacks is and will remain challenging. For the most part, traditional computer security defenses will not suffice. It will take a combination of things – technical defenses, intrusion detection systems, process, and more – to provide meaningful protection.

The remaining papers in the introductory section includes a detailed accounting of the workshop discussions provided by Charles Pfleeger and an industry perspective of the problem provided by Michael McCormick. The second section contains a number of invited technical papers (by Malek Ben Salem, Shlomo Hershkop, and Sal Stofo; Roy Maxion; Ravi Sahita; Ravishankar Iyer; Virgil Gligor; and Sara Sinclair) describing the state of the art in insider attack detection, including a proposal for hardware support for preventing insider attack and an overview of the state-of-the-art in masquerade attack detection, with a sobering view of the limits of anomaly detection techniques if poorly designed. The book concludes with a perspective on the legal and ethical issues (by Jeffrey Hunker; Daniel Schutzer; Angelos Keromytis) raised by technical approaches to detecting insider attack, as well as contributions that set an agenda for future research.

It is our hope and expectation that this book will be of interest to practitioners and researchers to develop an ongoing research community focused on the most vexing of computer security problems.

Reflections on the Insider Threat

Charles P. Pfleeger

Pfleeger Consulting Group

Abstract This paper reports on a workshop in June 2007 on the topic of the insider threat. Attendees represented academia and research institutions, consulting firms, industry—especially the financial services sector, and government. Most participants were from the United States. Conventional wisdom asserts that insiders account for roughly a third of the computer security loss. Unfortunately, there is currently no way to validate or refute that assertion, because data on the insider threat problem is meager at best. Part of the reason so little data exists on the insider threat problem is that the concepts of insider and insider threat are not consistently defined. Consequently, it is hard to compare even the few pieces of insider threat data that do exist. Monitoring is a means of addressing the insider threat, although it is more successful to verify a case of suspected insider attack than it is to identify insider attacks. Monitoring has (negative) implications for personal privacy. However, companies generally have wide leeway to monitor the activity of their employees. Psychological profiling of potential insider attackers is appealing but may be hard to accomplish. More productive may be using psychological tools to promote positive behavior on the part of employees.

1 Introduction

In June 2007 the U.S. Army Research Office, the Financial Services Technology Consortium (FSTC) and the Institute for Information Infrastructure Protection (I3P) sponsored a workshop on insider attack and cyber security. The two-day event featured participants from academia, research institutions, consulting firms, industry, and the government. The security researchers, practitioners and vendors in who attended shared insights and frustrations.

Reflecting on the presentations, discussions and comments, I am documenting in this paper some high level observations that came as a result of that meeting.

2 Who Is an Insider?

Who is an insider?

This question seems straightforward and easy to answer. But as with other fundamental terms in computer security (such as integrity, availability, or even security) the definition of insider is not well established. There are several possible uses of the term.

An insider can be:

- an employee, student, or other "member" of a host institution that operates a computer system to which the insider has legitimate access
- an associate, contractor, business partner, supplier, computer maintenance technician, guest, or someone else who has a formal or informal business relationship with the institution
- anyone authorized to perform certain activities, for example a bank's customer who uses the bank's system to access his or her account
- anyone properly identified and authenticated to the system including, perhaps, someone masquerading as a legitimate insider, or someone to whom an insider has given access (for example by sharing a password)
- someone duped or coerced by an outsider to perform actions on the outsider's behalf
- a former insider, now using previously conferred access credentials not revoked when the insider status ended or using access credentials secretly created while an insider to give access later

This rather broad range of interpretations of the term insider is by no means exhaustive. But it does point out the potential for confusion both inside and outside the computer security profession.

2.1 Motive

The motives for an insider attack are similarly diverse. In fact, the term "attack" may be overly harsh for certain types of insider actions:

- making an unintentional mistake
- trying to accomplish needed tasks—for example, in a case in which the system does not support a particular action or the insider is blocked from accessing certain data, the insider may try workarounds to accomplish the same thing
- trying to make the system do something for which it was not designed, as a form of innovation to make the system more useful or usable

- trying innocently to do something beyond the authorized limit, without knowing the action is unauthorized
- checking the system for weaknesses, vulnerabilities or errors, with the intention of reporting problems
- testing the limits of authorization; checking the system for weaknesses, vulnerabilities or errors, without the intention of reporting problems
- browsing, killing time by viewing data
- expressing boredom, revenge or disgruntlement
- perceiving a challenge: treating the system as a game to outwit
- acting with the intention of causing harm, for reasons such as fame, greed, capability, divided loyalty or delusion

We obviously react differently to these different motivations, sympathizing with the employee who wants to get work done in spite of the system, but deploring agents with malicious intent. Unintentional errors are usually seen as unfortunate but inevitable, and malicious behavior is usually seen as something heinous that should be prevented. But the area between these two ends is grey.

Unfortunately for research purposes different people include different ones of these cases in the definition of insider behavior. A given action may be classified as an insider attack in one study but not in another, which complicates assessing the severity and frequency of insider "attacks." Because different projects use different definitions, comparing results or statistics between projects can be difficult for analysts and misleading to the public.

As one participant pointed out during the workshop, two interesting cases arise: when bad things happen even though system privileges are not exceeded, and when good things happen even though system privileges are exceeded. We might initially say we want to prevent the former, but blocking acceptable behavior risks limiting a system's usability. We also tend to excuse the latter if the good dominates. These two cases show how difficult it is to separate acceptable insider behavior from unacceptable. With a murky policy definition, enforcement becomes problematic.

2.2 Effect

Another way to analyze the insider threat is to look at the effect insiders' actions have had. Here are some impacts of insider attacks:

- making available data or computer services to people who would otherwise not have had them—either because the people were not authorized or because the system failed to deliver as expected or intended
- receiving data for which the user was not authorized because such data fell outside the user's job requirements

- obtaining data or services for fraudulent purposes

The first impact here would sometimes be considered positive, and the last is usually negative. The middle impact can be mixed, depending on what use the user made of the data. The impact of a insider can thus range from positive to negative.

2.3 Defining the Insider Threat

Two major points stand out: First, we need standard definitions of insiders and insider behavior so studies and discussions can compare like entities. These definitions need to be used not just in the computer security research community but also by commercial security professionals (such as chief security officers and other management) and the press. (Convincing the press to use these terms precisely may be challenging.)

Second, we need to recognize that, unlike the "outsider" threat, insider behavior with the potential to do harm ranges from human nature (unintentional errors) through positive intentions (getting the job done in spite of an uncooperative system) and finally to all kinds of malice. Threat is the common term in computer security for an action with the potential to cause harm. But because the word "threat" has a negative connotation, some people would understandably not ordinarily use it to describe unintentional or non-malicious behavior. We must be especially careful when using the term "insider threat" to be sure our meaning is not misconstrued and insiders are not offended.

2.4 Context

Distinguishing acceptable from unacceptable insider behavior is difficult in part because of context. A disclosure may be acceptable only to certain people, at a certain time, in a certain location, in the presence (or absence) of certain other people, if certain other conditions obtain, for one time, and only if the recipient has not already obtained certain other data. Although such complex access control rules can be modeled and implemented, these rules go well beyond the subject–object–mode paradigm traditionally used for access control. These complex rules reflect the factors employed daily in personal data sharing decisions (between people, not involving computers), computer scientists do not even know the full set of parameters on which access control decisions are based outside of computers; thus it is premature to expect their implementation in most computing systems.

In fact, physical security recognizes a need for two kinds of systems: automated, mechanical systems that are unforgiving (such as gates and badge readers), and human overrides that can exercise judgment (such as dealing with the lost or forgotten badge or allowing emergency access by medical personnel). Acceptable behavior can be similarly rigidly determined by a system. But the working of some organizations is highly nuanced and sensitive data are communicated under subjective access control regimes.

These rich, context-based human access control decisions pose a problem for insiders: To share computerized data in those ways may require going outside or around the system, that is, violating system access controls. This is one of many examples in which insiders need to go outside or around the system's controls in order to accomplish needed goals.

3 Insider Threat Issues

Research on insider threats has several limitations. First, there is only meager data on inappropriate insider activity. Second, it would be very useful to probe the minds of insiders to determine what makes an insider good or bad. In part because of limited data, and in part because of limitations of current psychology, success in this avenue may be narrow. Third, the way to determine what insiders are doing is to monitor them, but monitoring of users or employees has privacy implications. Finally, technology is important in many areas of computer security, but the insider threat may be one for which the uses of current technology are somewhat incomplete.

3.1 Data

Research on the insider threat is hampered by the definitional problems described above. An even more serious limitation is the scarcity of data.

Scarcity of data seems puzzling in light of various comments at the workshop. One participant said the majority of insider attacks are undetected. That statement seems self-contradictory: If the majority of insider attacks are undetected, how can we know those attacks constitute a majority? Another researcher reported on a study to try to analyze behavioral intent using host-based sensors. The researcher acknowledged that the work had both false positives and false negatives. But here again, knowing or asserting that a system produces false positives and false negatives almost implies that we know the true positives and true negatives in order to be able to classify other events as false. Another participant noted that people use USB devices to transport data avoiding access controls. When another participant asked if there were studies to back up that assertion, the first replied that the report was merely anecdotal.

As a community we assert certain points, but in the realm of insider threat and insider behavior some of our assertions are hunches. Repeated enough times, hunches become accepted as fact.

Obtaining accurate data on the insider threat is difficult for several reasons, including

- imprecise definitions (as previously discussed)
- unclear policy of what constitutes welcomed or allowable insider behavior versus what constitutes behavior to be discouraged or prohibited
- massive amounts of data: assuming that the number of acceptable insider actions is far larger than the number of potentially negative insider threat actions, large amounts of uninteresting data will have to be filtered out
- reticence to share: because of laws, image, morale, and other factors, some organizations who collect data on insider activity are unwilling to share data with other organizations
- privacy concerns that limit data collection and sharing

The absence of good data limits researchers' ability to analyze, hypothesize and validate. One researcher went so far as to say that researchers need data to address problems; if organizations are not serious enough to supply researchers data, they (the organizations) aren't treating their problem as serious.

One source of data are police and court records. Cases are usually in the public record and details of the crime are reasonably complete. However, these records present a biased view of insider threat. First, police are involved only in crimes. As described earlier, insider behavior can sometimes be positive, so the police will not be involved. Even when the behavior is negative in some cases the companies will let the insider off with a warning or at most a dismissal. And some kinds of insider malicious activity are not criminal offenses. Second, some companies choose not to prosecute even in the case of a crime, fearing the negative publicity. Furthermore, faced with many crimes, district attorneys sometimes put computer crime cases low on their priority list, especially cases in which the loss is intangible and not huge, because of the complexity of prosecuting the case and the consequently low probability of winning. Finally, crime statistics typically cover only a single country (or other jurisdiction, such as a city or district), meaning that insider attacks against multinational companies may be hard to track. For all these reasons, criminal data must be viewed in context.

3.2 Psychology

In the workshop several speakers cited a need for a psychological component to insider threat study. There were basically two directions to the work involving psychology: profiling and motivating.

Some participants wanted a psychological profile of an insider who was likely to offend (and preferably before the offense). More than one person wanted to know how to identify potential insider attackers (before they attack, and ideally before they are hired).

For years the criminal justice system has unsuccessfully sought the profile of the criminal. Criminologists are not even close to identifying criminals reliably in advance. It seems as if criminals are varied in their motivation and psychological makeup. We may be able to identify some very antisocial personalities, but other criminals elude advance detection. The possibility of false positives hampers these efforts. If we have been unable to identify serious criminal intent or behavior, why should we expect to be able to identify insider threats?

Complicating psychological identification is that we send mixed signals to insiders. We praise creative individuals, ones who are adept at making a recalcitrant system work. Initiative, industriousness, and problem solving are positive traits on employee reviews. So we should not be surprised when an insider users these traits to be more productive.

We do not know if insiders expand their threat activity, first for nonmalicious purposes and then gradually to more malicious ends. Consequently we do not know if our rewarding unorthodox system use actually starts insiders on a path to malicious behavior. The situation is probably far more nuanced than this description.

Psychological screening would be ideal before an employee is hired. The typical job interview lasts no more than one day, and it involves both trying to get a sense of whether to hire the potential employee and at the same time convincing the employee to accept a job if offered. An intense psychological evaluation rigorous enough to identify potential inside attackers might be off-putting to non-attackers who should be hired. And time spent evaluating the candidate psychologically reduces the time to assess whether the person would be an asset to the organization. So, even if a psychological exam were available, its use might be counterproductive.

Prospects do not look good for developing psychological profiles. We have too little data (too few cases) with which to work, we do not have a good understanding of the norms of acceptable behavior, we are not sure where is the boundary between acceptable and unacceptable behavior, and we must be able to address many different motivations for unacceptable behavior. Perhaps when we understand general human behavior better we will be able to develop useful profiles.

The other major use for psychology is positive: developing ways of reinforcing good behavior. Some participants wanted to understand how to use psychology to keep insiders acting in positive ways. The prospects seem more promising for this use of psychology than for profiling.

The difference between profiling and motivating is that we want profiling to be precise, generating few false positives and false negatives (because the risk of a false positive is not hiring a potential good employee or holding back or dismissing someone who has not yet—and might never—exhibit harmful behavior, and the risk of a false negative is failing to prevent or detect an attack). If a motivating technique is largely effective, meaning that it serves it desired purpose on a significant enough proportion of people, it is deemed successful. We can afford to use several motivational techniques that work for different people.

3.3 Monitoring and Privacy

Privacy concerns significantly limit data collection and psychological modeling. Again, the definition of insider becomes important.

When the insider is an employee, privacy rights are subordinated to business rights. The courts have consistently upheld the right of a company to monitor employees' behavior, as long as there is a reasonable business purpose for the monitoring and the monitoring does not violate basic human and civil rights. Thus, companies can generally capture and analyze an employee's email and other communications that use company equipment, log all files and other resources an employee accesses, and retain copies of programs and data an employee creates under the company's auspices. A company is far more free in tracking its employees' system activities than would be law enforcement, for whom probable cause and a search warrant are needed.

But not all insiders are employees. Some definitions of insider include people such as account holders who access their bank accounts, patients who use an electronic system to communicate with medical professionals or view or manage their medical records, students at universities, customers who use online shopping systems, and similar users. Each of these users has certain authorized access to a system. Privacy for some classes of users is covered by laws, such as HIPAA for patients in the United States or the European Privacy Directive for many accesses by Europeans. In other cases, the privacy regulations allow monitoring, data collection and retention, and even data sharing if it is documented in a privacy policy (and sometimes even if not). In these cases, then, privacy rights vary.

Regardless of whether the company has the right to monitor its users' actions, some companies choose not to monitor because of possible negative public opinion.

Another type of insider is the business partner, consortium member, subcontractor, or the like. In these cases, privacy rights are even weaker than for the category of users. The contract between partners may spell out rights to track behavior, although not all such relationships are covered by a contract.

So, is monitoring of insiders' activity permissible? Perhaps and sometimes. Is it desirable for the organization to perform? Perhaps and sometimes. The other important question is whether the monitoring is effective.

3.4 Detecting Insider Attacks

Insider attacks are difficult to detect, either by human or technical means. One workshop participant observed that most insider attacks are detected only because of some reason to suspect: the insider may have talked (bragged) about the act, for example. In other kinds of crime police investigators sometimes profit from a perpetrator who does something to draw suspicion.

An insider attack recognition tool would be useful to flag attacks or suspicious behavior in time to limit severity. Clearly most insider activity is not malicious; otherwise organizations' computer systems would be constantly broken. Thus, the volume of nonmalicious insider activity far outweighs that of malicious activity. Such volume of data is hard to analyze.

A similar example is an intrusion detection system protecting a system from malicious network access: Most network traffic is benign. Intrusion detection technology is improving all the time. However, intrusion detection systems are best at finding specific examples of inappropriate access, either because the access fits a pattern of known malicious activity or because the access touches specific sensitive resources in unusual ways. The hardest attack for an intrusion detection system to recognize is one composed of pieces spread across a long period of time. For those attacks the intrusion detection system has to collect and correlate pieces of data over time, which implies a long window of data comparison.

Inside attackers presumably will perform both normal and malicious acts, which complicates the search for anomalous activity beyond that performed by an intrusion detection system.

One important question raised, then, about monitoring to identify inappropriate behavior is whether the monitoring is effective. Intrusion detection techniques may be of some value. But because there is so little published research on insider attacks, it is impossible to tell whether monitoring helps. Monitoring is useful to confirm a suspected case of insider attack. There is controversy as to whether monitoring serves as a deterrent; that is, if insiders know their activity is being monitored are they less likely to engage in inappropriate activity? The answer to that is unknown, although one workshop participant noted that monitoring is not effective to deter retail theft by employees. Another participant said that detection of a data leak is unlikely unless there is some trigger that makes the leak prominent.

3.5 Technology

What technology is available to detect, deter, or prevent insider attacks?

Most existing computer security technology is based on the concept of a perimeter defense. The attackers are outside the line, the defense blocks the attackers, and the sensitive resources inside are safe. Firewalls are the classic perimeter defense. With insider attacks, drawing the protection line is more difficult because the attacker and the sensitive resources are on the same side of the line.

Intrusion detection systems may be of some value in detecting insider attacks. As previously discussed, these systems need to analyze a large amount of data, correlate pieces of data potentially spread over a long period of time, and distinguish malicious from nonmalicious intent. These three factors exceed the current demands on intrusion detection systems.

Operating systems, access controls and audit logs offer little support for controlling insider threats, because the insider is acting within limits of authorized behavior, just doing inappropriate things with that allowed access.

But more to the point, technological approaches may be wrong for dealing with the insider threat. The basic element of the insider threat is human: a perpetrator has abused a position of trust. The insider is part of the organization and has some loyalty to it. Capitalizing on the human aspect includes determining what kinds of people are likely to abuse trust or creating an environment in which people would not want to abuse.

4 Conclusions

Little is known about the insider threat. Even computer security professionals use different definitions for insider, and so it is not surprising that the general computing field, as well as the general public, can offer little insight into the problem of insider attacks. Add to this organizational reticence to be embarrassed in public and it is not surprising that there is little valid measurement and reporting on insider attacks or the insider threat.

Several significant points evolved at the insider threat workshop:

- The term insider must be clearly defined. That definition must be communicated to security professionals, computer professionals outside of security, management, and the general public.
- The term insider attack must be clearly defined. That definition must be communicated to security professionals, computer professionals outside of security, management, and the general public.
- Data from reliable measurement of insider activity—malicious and not—must be gathered and shared within the security research community.
- Cooperation with industry is necessary: Industry has the insider attack cases, but security researchers have the tools and inclination to analyze the data. Each side needs the other.
- Developing a psychological profile of a likely attacker is an attractive goal. Because of variation among human motivations, and limitations in the knowledge of psychology, such a profile may prove elusive.

- Psychology may be more effective at finding positive controls: conditions that make it less likely that an insider will want to harm the organization.
- Technical controls to prevent, detect, or deter malicious insider behavior will be difficult to develop. The insider exploits legitimate access. Limiting such access may have a negative effect on nonmalicious employees' productivity.

Acknowledgments

I would like to thank the conference organizers for inviting me to attend this workshop and to write this overview paper for the proceedings.

Many of the basic ideas presented here come from the workshop participants, although not quoting from them directly, I have not identified them by name, because they have not had the opportunity to view or rebut anything I write about their ideas here. Conclusions and elaborations are my `own.

The “Big Picture” of Insider IT Sabotage Across U.S. Critical Infrastructures

Andrew P. Moore, Dawn M. Cappelli, Randall F. Trzeciak

CERT®[1], Software Engineering Institute and CyLab at Carnegie Mellon University

Abstract A study conducted by the U.S. Secret Service and the Carnegie Mellon University Software Engineering Institute CERT Program analyzed 150 insider cyber crimes across U.S. critical infrastructure sectors. Follow-up work by CERT involved detailed group modeling and analysis of 54 cases of insider IT sabotage out of the 150 total cases. Insider IT sabotage includes incidents in which the insider’s primary goal was to sabotage some aspect of the organization or direct specific harm toward an individual. This paper describes seven general observations about insider IT sabotage based on our empirical data and study findings. We describe a System Dynamics model of the insider IT sabotage problem that elaborates complex interactions in the domain and unintended consequences of organizational policies, practices, technology, and culture on insider behavior. We describe the structure of an education and awareness workshop on insider IT sabotage that incorporates the previously mentioned artifacts as well as an interactive instructional case.

[1] CERT and CERT Coordination Center are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

1 Introduction

Insiders, by virtue of legitimate access to their organizations’ information, systems, and networks, pose a significant risk to employers. Employees experiencing financial problems have found it easy to use the systems they use at work everyday to commit fraud. Other employees, motivated by financial problems, greed, revenge, or the wish to impress a new employer, have stolen confidential data, proprietary information, or intellectual property from their employers. Lastly, technical employees have used their technical ability to sabotage their employers’ systems or networks in revenge for negative work-related events.

In January 2002, the Carnegie Mellon University Software Engineering Institute’s CERT Program (CERT) and the United States Secret Service (USSS) National Threat Assessment Center (NTAC) started a joint project, the *Insider Threat Study*.[2] The study combined NTAC’s expertise in behavioral psychology with CERT’s technical security expertise to provide in-depth analysis of approximately 150 insider incidents that occurred in critical infrastructure sectors in the U.S. between 1996 and 2002. Analysis included perusal of case documentation and interviews of personnel involved in the incident.

Two reports have been published to date as part of the *Insider Threat Study*. One analyzed malicious insider incidents in the banking and finance sector (Randazzo 2004). The other analyzed insider attacks across all critical infrastructure sectors where the insider’s intent was to harm the organization, an individual, or the organization’s data, information system, or network [5]. Two additional reports will be published in the future: one pertaining to the information technology and telecommunications sector, and the other geared to the government sector.

The *Insider Threat Study* provided the first comprehensive analysis of the insider threat problem. CERT’s technical security expertise was augmented with expertise from several experts in the areas of psychology, sociology, insider threat, espionage, cyber crime, and specific domains like the financial industry. The results of the study show that to detect insider threats as early as possible or to prevent them altogether, members of management, IT, human resources, security officers, and others in the organization must understand the psychological, organizational, and technical aspects of the problem, as well as how to coordinate their actions over time.

The CERT project team felt that it was important to further utilize the wealth of empirical data from the *Insider Threat Study* to next concentrate on conveying the "big picture" of the insider threat problem - the complex interactions, relative degree of risk, and unintended consequences of policies, practices, technology, insider psychological issues, and organizational culture over time. Thus, the

[2] The Insider Threat Study was funded by the USSS, as well as the Department of Homeland Security, Office of Science and Technology, which provided financial support for the study in fiscal years 2003 and 2004.

MERIT project was initiated[3]. MERIT stands for the Management and Education of the Risk of Insider Threat. As part of MERIT, we are developing a series of models and associated tools that can be used to better communicate the risks of the insider threat.

This paper focuses on insider IT sabotage across the U.S. critical infrastructure sectors: insider incidents in which the insider's primary goal was to sabotage some aspect of the organization (e.g., business operations; information or data files; the system or network; organizational reputation) or to harm an individual. Section 2 describes key concepts for understanding the domain in the context of seven general observations about insider IT sabotage based on our empirical work. Section 3 presents the System Dynamics model of the insider IT sabotage problem, bringing into sharper focus the concepts previously described, with an emphasis on their dynamic interrelationship. Section 4 identifies leverage points for the possible mitigation of the insider IT sabotage problem. Section 5 illustrates the structure of a workshop about insider IT sabotage that incorporates the previously mentioned artifacts. Section 6 concludes with an assessment of the value of our modelling efforts and a summary of our ongoing and future work in the area. Additionally, appendices describe details of the System Dynamics approach we use, an instructional case used in our insider threat workshop, and an overview of the complete insider IT sabotage model.

2 General Observations About Insider IT Sabotage

The cases of insider IT sabotage were among the more technically sophisticated attacks examined in the *Insider Threat Study* and resulted in substantial harm to people and organizations. Forty-nine cases were studied, as described in [5]. Eighty-six percent of the insiders held technical positions. Ninety percent of them were granted system administrator or privileged system access when hired by the organization. In those cases, 81 percent of the organizations that were attacked experienced a negative financial impact as a result of insider activities. The losses ranged from a low of five hundred dollars to a high of "tens of millions of dollars." Seventy-five percent of the organizations experienced some impact on their business operations. Twenty-eight percent of the organizations experienced a negative impact to their reputations.

The *Insider Threat Study* focused on analysis of individual components of insider incidents, such as characteristics of the insider, technical details, planning and communication before the incident, detection and identification of the insider, and consequences of the attack. The purpose of the MERIT models is to analyze the cases in a different way. Rather than focusing on individual details of the

[3] The *MERIT* project is supported by the Army Research Office through grant number DAAD19-02-1-0389 ("Perpetually Available and Secure Information Systems") to Carnegie Mellon University's CyLab.

cases, MERIT attempts to identify common patterns in the evolution of the cases over time. Although 49 insider IT sabotage cases were examined for the *Insider Threat Study*, not all of the case files contained enough information for this modeling effort. In the end, 30 IT sabotage cases were selected for use in this project based on availability of pertinent information.

In performing the “big picture” analysis of insider IT sabotage, we identified seven general observations about the cases. We then validated those observations against the empirical data from the *Insider Threat Study*. We have used the comparative case study methodology [12], in our research. The findings from case study comparisons cannot be generalized with any degree of confidence to a larger universe of cases of the same class or category. What this method can provide, however, is an understanding of the contextual factors that surround and influence the event. We briefly describe each of those observations below, along with the percentage of cases that supports the observation. Band, et.al. [2] describes these observations in more detail, including their relevance to the problem of espionage.

Observation 1: Most insiders had personal predispositions that contributed to their risk of committing IT sabotage.

> Personal predisposition: a characteristic historically linked to a propensity to exhibit malicious insider behavior.

Personal predispositions explain why some insiders carry out malicious acts, while coworkers that are exposed to the same conditions do not act maliciously. Personal predispositions can be recognized by certain types of observable characteristics [2]:

- Serious mental health disorders – Sample observables from cases include alcohol and drug addiction, panic attacks, physical spouse abuse, and seizure disorders.
- Social skills and decision-making – Sample observables from cases include bullying and intimidation of coworkers, serious personality conflicts, unprofessional behavior, personal hygiene problems, and inability to conform to rules.
- A history of rule violations – Sample observables from cases include arrests, hacking, security violations, harassment complaints, and misuse of travel, time, and expenses.

All of the insiders in the MERIT cases who committed IT sabotage exhibited the influence of personal predispositions.

Observation 2**:** Most insiders who committed IT sabotage were disgruntled due to unmet expectations.

> Unmet expectation: An unsatisfied assumption by an individual that an organization action or event will (or will not) happen, or a condition will (or will not) exist.

All of the insiders in the MERIT cases who committed IT sabotage had unmet expectations. In the *Insider Threat Study* IT sabotage cases, 57 percent of the insiders were perceived as being disgruntled. Eighty-four percent were motivated by revenge, and 92 percent of all of the insiders attacked following a negative work-related event such as termination, dispute with a current or former employer, demotion, or transfer.

Unmet expectations observed in cases include insufficient salary/bonus, lack of promotion, restriction of online actions, limitations on use of company resources, violations of privacy, diminished authority/responsibilities, unfair work requirements, and poor coworker relations.

Observation 3**:** In most cases stressful events, including organizational sanctions, contributed to the likelihood of insider IT sabotage.

> *Stressful events:* those events that cause concerning behaviors in individuals predisposed to malicious acts.

Ninety seven percent of the insiders in the MERIT cases who committed IT sabotage experienced one or more stressful events, including sanctions and other negative work-related events, prior to their attack. The majority of insiders who committed IT sabotage in the *Insider Threat Study* cases attacked after termination or suspension from duties.

Stressful events observed in cases include poor performance evaluations, reprimands for unacceptable behavior, suspension for excessive absenteeism, demotion due to poor performance, restricted responsibilities and Internet access, disagreements about salary or bonuses, lack of severance package, new supervisor hired, divorce, and death in family.

Observation 4: Behavioral precursors were often observable in insider IT sabotage cases but ignored by the organization.

> *Behavioral precursor:* an individual action, event, or condition that involves personal or interpersonal behaviors and that precedes and is associated with malicious insider activity.

Ninety seven percent of the insiders in the MERIT cases who committed IT sabotage came to the attention of supervisors or coworkers for concerning behavior prior to the attack. Eighty percent of the insiders who committed IT sabotage in the *Insider Threat Study* exhibited concerning behavior prior to the attack, including tardiness, truancy, arguments with coworkers, and poor job performance.

Behavioral precursors observed in cases include drug use, conflicts with coworkers, aggressive or violent behavior, inappropriate purchases on company accounts, mood swings, poor performance, absence or tardiness, sexual harassment, deception about qualifications, violations of dress code, and poor hygiene. Many behavioral precursors were direct violations of explicit organizational policies and rules.

Observation 5: In many cases organizations failed to detect technical precursors.

> *Technical precursor:* an individual action, event, or condition that involves computer or electronic media and that precedes and is associated with malicious insider activity.

Eighty seven percent of the insiders in the MERIT cases of insider IT sabotage performed technical precursors prior to the attack that were undetected by the organization.

Technical precursors observed in cases include downloading and using hacker tools, failure to create backups, failure to document systems or software, unauthorized access of customers’ or coworkers’ systems, sharing passwords, demanding passwords from coworkers, system access after termination, inappropriate Internet access at work, and the setup and use of backdoor accounts.

Observation 6: Insiders created or used access paths unknown to management to set up their attack and conceal their identity or actions. The majority of insiders attacked after termination.

> *Access path*: a sequence of one or more access points that lead to a critical system.

Seventy five percent of the insiders in the MERIT cases who committed IT sabotage created unknown access paths. In the *Insider Threat Study* IT sabotage cases, 59 percent of the insiders were former employees, 57 percent did not have authorized system access at the time of the attack, and 64 percent used remote access.

Many insiders in the cases analyzed used privileged system access to take technical steps to set up the attack before termination. For example, insiders created backdoor accounts,[4] installed and ran password crackers,[5] installed remote network administration tools, installed modem access to organization systems, and took advantage of ineffective security controls in termination processes. Many of these steps created or allowed the use of access paths unknown to the organization.

Observation 7: Lack of physical and electronic access controls facilitated IT sabotage.

[4] A backdoor account is an unauthorized account created for gaining access to a system or network known only to the person who created it.

[5] A password cracker is a program used to identify passwords to a computer or network resource; used to obtain passwords for other employee accounts.

Electronic access controls: the rules and mechanisms that control electronic access to information systems
Physical access controls: the rules and mechanisms that control physical access to premises

Ninety three percent of the insiders in the MERIT IT sabotage cases exploited insufficient access controls. Access control vulnerabilities observed in cases include coworker's computers unattended while logged in, ability to create accounts unknown to organization, ability to release code into the production system without checking or knowledge of organization, an imbalance between physical and electronic access controls, and insufficient disabling of access at termination.

3 Model of the Insider IT Sabotage Problem

The *Insider Threat Study* investigated cases of actual insider attack. It therefore brought to light how the problem of malicious insider retribution arises and escalates within the organizational context. This section describes the key elements of the insider IT sabotage problem that we saw in a majority of cases. The patterns embodied by the model were not seen in all cases but in a sufficient number of cases to raise concern. In the next section we will describe the measures that an organization can take to prevent, detect, and respond to malicious insider actions based on our extended group's experience on the psychology of insiders as well as the managerial and technical aspects of organizational and information security. In the course of our study, we learned much more about what organizations should ***not*** do than what they should. Further research is needed to understand the effectiveness of various countermeasures for the insider threat problem.

After researching potential methods and tools that could be used for this purpose, System Dynamics was chosen for its strengths in modeling and simulation of complex problems [11]. This paper is written for readers who are not familiar with System Dynamics modeling. An explanation of System Dynamics is provided in Appendix A and will be helpful for understanding the following description. For those readers who are familiar with System Dynamics, we emphasize that we do not use the traditional causal loop diagramming notation in this paper. In our experience, the traditional notation using positive and negative signs can be confusing to audiences not familiar with System Dynamics; non-technical people generally have been intimidated by the notation and technical people often read too much into the signs. In the following presentation, we use a more subtle notation of dashed arrows for negative influence and solid arrows for positive influence.

3.1 Insider Expectation Escalation

Employee disgruntlement was a recurring factor in the insider IT sabotage cases, predominately due to some unmet expectation by the insider. For example:

1. The insider expected certain technical freedoms in his[6] use of the organization's computer and network systems, such as storing personal files, but was reprimanded by management for exercising those freedoms.
2. The insider expected to have control over the organization's computer and network system, but that control was revoked or never initially granted.
3. The insider expected a certain financial reward for his work, but bonuses were lower than expected due to the company financial status.

Fig. 1 represents the escalation of expectation that often leads to insider disgruntlement. As shown in the lower left side of the figure, the insider's *personal predisposition* could lead to heightened expectation. This predisposition differs from one person to the next, and influences the rate that expectations rise and fall.

The rise of expectations is influenced heavily by the *expectation fulfillment.* Policies and management controls are needed to keep employee expectations in check. As illustrated in reinforcing loop (R1), with lax management controls the *insider's expectation* grows commensurate with the *expectation fulfillment.* As expectation grows and is fulfilled, expectation grows even more.

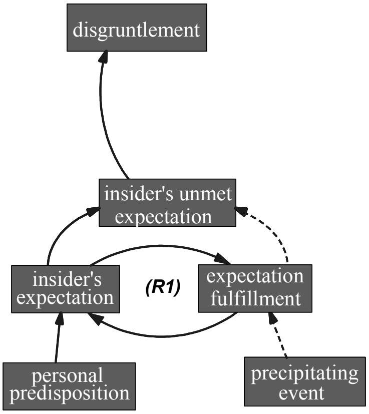


Fig. 1: Expectation Escalation

[6] Ninety-six percent of the insiders in the *Insider Threat Study* who committed IT sabotage were male. Therefore, male gender is used to describe the generic insider throughout this paper.

Lax management that permits continually increasing employee expectation can result in major problems later, especially if the insider is so predisposed. The trigger for those major problems, which we call the *precipitating event*, tends to be anything that removes or restricts the freedom or recognition to which the insider has become accustomed. For instance, the hiring of a new supervisor who suddenly enforces the organization's acceptable use policy can cause extreme disgruntlement in the employees. Other precipitating events include the insider being passed up for a promotion, sanctions by management, or termination of the insider.

3.2 Escalation of Disgruntlement

Often the first sign of disgruntlement is the onset of *behavioral precursors*, observable aspects of the insider's social (non-technical) behavior inside or outside the workplace that might be deemed inappropriate or disruptive in some way. Some examples of behavioral precursors in the MERIT cases were conflicts with coworkers; a sudden pattern of missing work, arriving late, or leaving early; or a sudden decline in job performance.

As shown in Fig. 2a, the degree of disgruntlement influences the insider's exhibition of behavioral precursors, which can be discovered provided that the organization has sufficient *behavioral monitoring* in place. An organization's punitive response to inappropriate behaviors in the form of sanctions can be technical, such as restricting system privileges or right to access the organization's systems from home, or non-technical, such as demotion or formal reprimand. The intended effect of sanctions, as shown in the balancing loop B1 of Fig. 2b, is to prevent additional behavioral precursors. Feedback loop R2, however, shows that sanctions can have unintended consequences such as escalation of disgruntlement. Whether sanctions curb behavioral precursor activity or spur the insider to greater disgruntlement and disruption depends largely on the personal predispositions of the insider.

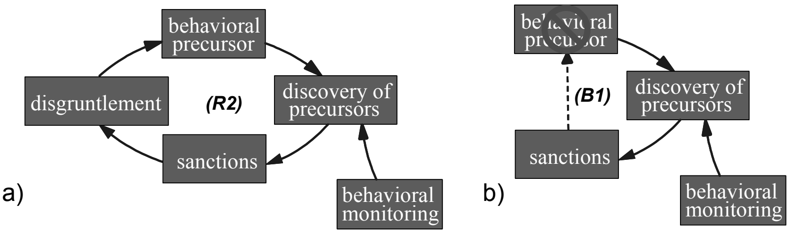


Fig. 2: a) Typical Escalation of Disgruntlement **b)** Intended Effect of Sanctions

3.3 Attack Setup and Concealment

Given an insider with personal predispositions, unmet expectations can lead to increasing disgruntlement which, if left unchecked, can spur not just behavioral precursors but technical disruptions and attacks on the organization’s computer and network systems. Prior to the actual attack, there are typically *technical precursors* - actions by the insider to either set up the attack (for example, installing malicious software programs) or to put in place mechanisms to facilitate a future attack (for example, creation of *backdoor accounts* - secret, unauthorized accounts to be used later for the attack). These technical precursors could serve as an indicator of a pending attack if detected by the organization.

Fig. 3 depicts the influence that insider disgruntlement can have on the occurrence of technical precursors that could indicate a pending attack. Some of these actions also contribute to the damage potential of the attack. Examples include sabotage of backups and decreases in the redundancy of critical services or software. As shown in loop R3, insiders may also acquire access paths unknown to the organization. This increases the insider’s ability to conceal their activity making it more difficult for the organization to discover the precursors. The feedback loop is reinforcing since the ability to hide their actions may embolden the risk-averse insider to continue, or even increase, their efforts to attack.

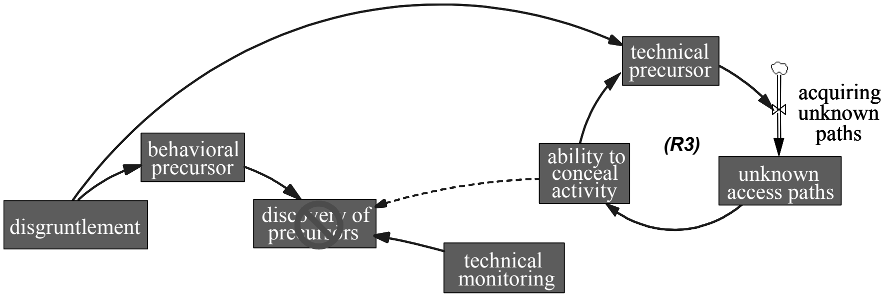


Fig. 3: Technical Precursors due to Disgruntlement

The extent to which insiders rely on unknown access paths to set up and execute their attack depends on their risk tolerance. Insiders who do not care whether they are caught, or insiders acting impulsively (often out of the passion of the moment), may use both known and unknown paths in their attack. Insiders who are particularly risk averse may only attack using access paths that are unknown to the organization. Of course, an insider may not know whether the organization is aware of a particular access path or not. Nevertheless, in either case, insiders generate technical precursors that suggest suspicious activity. Just as for behavioral precursors, the detection of technical precursors depends on having sufficient level of technical monitoring in place.

3.4 The Trust Trap

In addition to insider predispositions and behaviors, organizational predispositions and behaviors can also influence an organization's exposure to malicious insider acts. Fig. 4 depicts a trap in which organizations sometimes find themselves. We call this the Trust Trap and have described its role in previous models [1, 2,3].

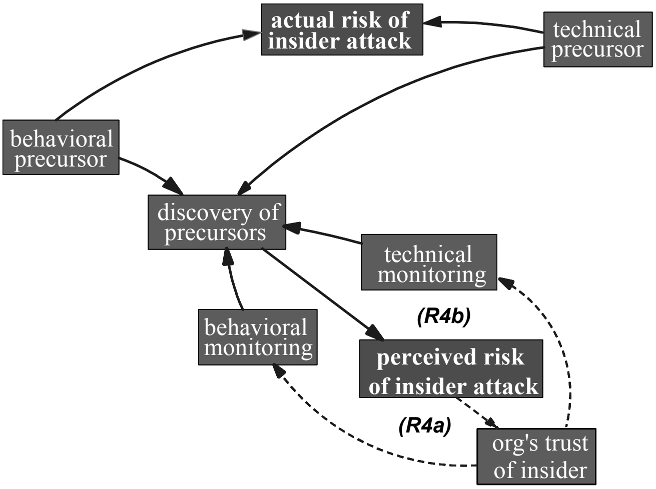


Fig. 4: Trust Trap

To understand the Trust Trap, we need to distinguish between the actual and perceived risk of insider attack. As shown in the top portion of Fig. 4, actual risk depends on the behavioral and technical precursors exhibited by the insider. The risk of insider attack is only perceived by the organization to the extent that they discover those precursors, however.

A key factor in the Trust Trap is the organization's trust of the insider, as shown in loops R4a and R4b. Clearly, there are good reasons why managers want to create a workplace in which individuals can trust each other and there is a good trust relationship between the organization and its employees, e.g., to increase morale and productivity. However, managers who strive to promote trusting workplace relationships sometimes shortcut essential behavioral and technical monitoring procedures, or let them erode over time due to competing pressures and priorities. Lower levels of monitoring lead to undiscovered precursors, resulting in an overall lower perceived risk of attack. This false sense of security reinforces managers' trust in the individuals working for them. The cycle continues, with the organization's monitoring capability steadily deteriorating until a major compromise becomes obvious to all involved.

4 Possible Leverage Points for Addressing the Problem

The intent of the MERIT project is to communicate the severity of the insider threat problem and describe it using System Dynamics models based upon empirical data. Although our research in CERT has focused on the insider threat problem, we would be remiss to leave participants with the impression that the organization is helpless to defend itself against someone from within. We can propose effective countermeasures based on our extended team’s expert opinions in behavioral psychology and information security.[7] All levels of management should recognize and acknowledge the threat posed by insiders and take appropriate steps to mitigate malicious attacks. While it may not be realistic to expect that every attempt at insider IT sabotage will be stopped before damage is inflicted, it is realistic to expect that organizations can build resiliency into their infrastructure and business processes to allow them to detect the attacks earlier, thereby minimizing the financial and operational impact.

This section of the report describes potential countermeasures that we believe could be effective in mitigating insider IT sabotage, based on expert opinions in our analysis of the problem.

4.1 Early Mitigation Through Expectation Setting

First of all, managers should recognize the personal predispositions of their employees and understand the impact they can have on insider threat risk. Second, organizations should attempt to manage the expectations of employees to minimize unmet expectations. This can be achieved through communication between managers and employees (especially in the form of regular employee reviews), taking action to address employee dissatisfaction when possible, and consistent enforcement of policies for all employees so that individual employees do not come to feel that they are above the rules or that the rules are unjustly applied.

[7] The effectiveness of the countermeasures proposed in this section is not supported in the case data since we were rarely able to obtain that kind of data during the coding process.

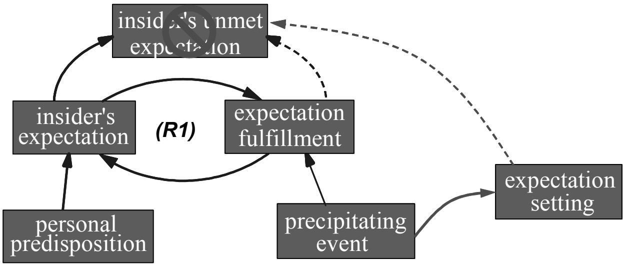


Fig. 5: Early Mitigation through Expectation Setting

Fig. 5 describes the influence *expectation setting* can have on the *insider's unmet expectations*. When the expectations of the insider are in line with the organization's practices and policies, unmet expectations are not an issue. However, if a precipitating event impacts expectation fulfillment, action by management to reset expectations might decrease the level of unmet expectations. If the organization fails to reset expectations, the level of unmet expectations may continue to rise, causing disgruntlement on the part of the insider.

For example, the organization can attempt to lower the level of unmet expectations regarding system use and job responsibilities by a number of proactive countermeasures:

- The organization institutes an acceptable use policy, describing the employee's roles and responsibilities when using the organization's information systems. The policy should be given to each employee as part of their orientation to the organization. As changes to the policy occur, employees need to be made aware of the changes and the impact to them. In addition, the policy should be consistently enforced for all employees so that no employees may feel that they are "above the rules."
- Managers, in conjunction with Human Resources, can clearly define job responsibilities for each employee in the organization. Processes such as performance reviews can be used to check and set expectations periodically.

4.2 Handling Disgruntlement Through Positive Intervention

As the organization discovers the behavioral precursors exhibited by the insider, they can employ positive intervention strategies to lower the disgruntlement of the insider. While the intent of employee sanctioning may be to reduce undesirable behaviors, it may backfire in some cases. Disgruntlement increases, leading to more disruptive behavior. Fig. 6 describes the influence *positive intervention* strategies might have on the *disgruntlement* of the insider. When positive intervention is used, the disgruntlement might be reduced, eliminating additional behavioral precursors, as well as the escalation to technical precursor behaviors (see Fig. 3).

One positive intervention strategy is an Employee Assistance Program (EAP). EAPs are sometimes offered by organizations as an employee benefit, to assist employees in dealing with personal or work-related issues that may affect job performance, health, and general well-being. EAPs can include counseling services for employees and/or their family members.

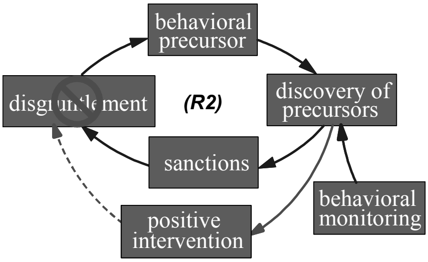


Fig. 6: Handling Disgruntlement through Positive Intervention

4.3 Targeted Monitoring

It is usually not practical for an organization to monitor every behavioral and technical action taken by each employee. However, a reasonable level of proactive logging of online activity across the organization's network provides data that can be monitored or audited for suspicious activity proactively, or targeted to monitor people who have raised the suspicions of their managers. Based on findings from the Insider Threat Study, for example, periodic account audits could be effective in detecting backdoor accounts that could be used for malicious insider activity.

Fig. 7 describes the relationship between the *perceived risk of an insider attack* and the amount of *technical* and *behavioral monitoring* organizations institute. As the perceived risk of an insider attack increases, due to detection of behavioral or technical precursors, the amount of technical and behavioral monitoring should also increase. Enhanced monitoring could lead to discovery of precursor activity, enabling the organization to identify individuals at a higher risk for malicious behavior and implement more targeted individual monitoring.

If a manager notices an employee progressing through the pattern of behavior described in the above, he might consider an audit of that employee's online activity, and, if the actions are extreme enough, perhaps escalate the level of logging of that employee's online activity. Note that policies should be in place in advance of such targeted monitoring; an organization should not perform these actions without consulting with their legal department in advance. Band et al. [2] identifies specific observable behaviors that should impact an organization's trust level.

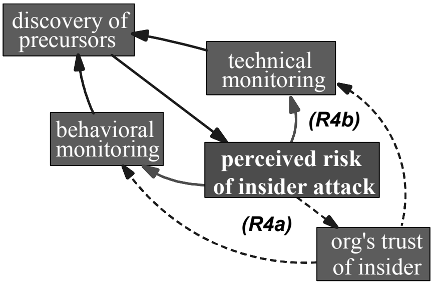


Fig. 7: Targeted Monitoring

4.4 Eliminating Unknown Access Paths

An organization's full awareness of access paths available to an insider is critical to being able to disable those access paths when needed. Fig. 8 reflects the relationship between two variables: *Insider access paths unknown to org* and *Insider access paths known to org*. Important relationships between the two variables include

- *forgetting paths:* Management or the IT staff may forget about known paths, making them unknown. The *forgetting paths* variable represents the rate at which access paths move from the known to the unknown category. For example, a manager might authorize a software developer's request for the system administrator password during a time of heavy development. Therefore, the system administrator password is an access path known to the organization at that point in time. If a formal list of employees with access to that password is not maintained, the manager could forget that decision over time. The manager may also simply resign from the organization, leaving no "organizational memory" of the decision to share the system administrator password. In either case, the system administrator password has now become an access path unknown to the organization.
- *discovering paths*: The *discovering paths* variable represents the rate at which management or the IT staff discover unknown paths, making them known. Access paths can be discovered by monitoring network traffic or by computer system account auditing, for example. Monitoring network traffic allows discovering suspicious network traffic for further investigation. Account auditing allows discovering unauthorized accounts directly.

Insiders can acquire new access paths unknown to the organization (through the *acquiring unknown paths* variable in the figure) by, for example, installing a backdoor account or stealing passwords. Finally, organizations can disable access paths that it knows about (through the *disabling known paths* variable in the figure) by, for example, removing backdoor accounts or changing shared passwords. The critical concept is that an organization may not know about all of the access paths each of their employees has to its critical systems.

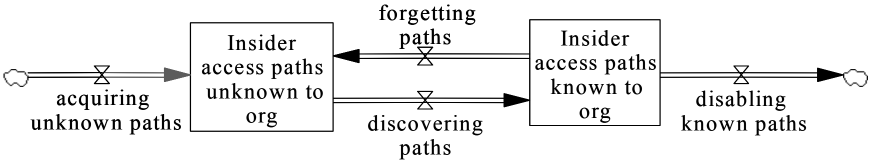


Fig. 8: Access Paths Available to Insider

Access paths unknown to the organization provide a mechanism that can be used by the insider to facilitate a future attack, even following termination. For example, organizations often did not know about (or did not think about) insiders’ access to shared accounts like system administrator or database administrator accounts; overlooking such accounts during an insider’s termination process often allowed an insider’s attack following termination. In addition, unknown access paths can make it more difficult for the organization to attribute the attack to the insider. If the organization is unaware of the paths that can be used by an insider for attacks, the task of protecting itself is significantly more complex.

Fig. 9 emphasizes the importance of diligent tracking and management of access paths into the organization’s system and networks. As tracking increases, the likelihood an organization will forget about the existence of specific access paths and who has access to them decreases. If precursor technical activity is detected, unknown access paths can be discovered and disabled, further reducing the number of unknown access paths available to the insider. As the number of unknown access paths decreases, the ability to conceal malicious activity by the insider decreases. As the *ability to conceal* decreases, the discovery of technical precursors increases. This makes it more and more difficult for the insider to conceal unauthorized or malicious online activity. Conversely however, if technical precursors are not discovered, the insider can accumulate unknown access paths, making it easier for him to conceal his actions.

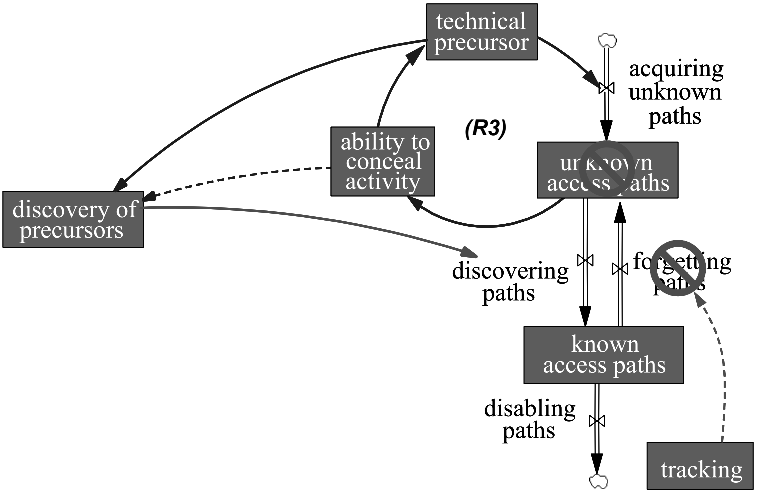


Fig. 9: Eliminating Unknown Access Paths

In the cases we examined, accounts that were secretly created by the insider or shared with other coworkers were access paths often used by the insider but unknown by management. In addition, lack of tracking led to unknown access paths for the insider that were overlooked in the termination process, and later used by the insider to attack. Therefore, an important practice for tracking access paths and reducing the occurrence of unknown access paths is ongoing and thorough account management. Account management is a complex task, encompassing verification of new accounts, changes to account authorization levels, tracking who has access to shared accounts, and decommissioning of old accounts. Unfortunately, it takes a significant amount of time and resources for an organization to recover from obsolete account management practices.

4.5 Measures Upon Demotion or Termination

Termination or demotion was the final precipitating event in many cases we examined. It is important that organizations recognize that such precipitating events may cause the insider to take technical actions to set up and carry out the attack possibly using previously acquired unknown access paths. A clearly-defined process for demotions and terminations in combination with proactive IT best practices for detecting unknown access paths and eliminating unauthorized access paths can reduce the insider's ability and/or desire to attack the organization.

Fig. 10 reflects the steps that the organization can take to mitigate the insider IT sabotage risk following demotion and termination. Prior to the demotion or termination, the organization should be certain about what access paths are available to the insider. If the insider is to be terminated, the organization must disable all access paths prior to notifying the insider of the action. It is important to understand that if the organization has been lax in tracking and managing access paths, it could be too late to confidently demote or terminate an employee without fear of retribution.

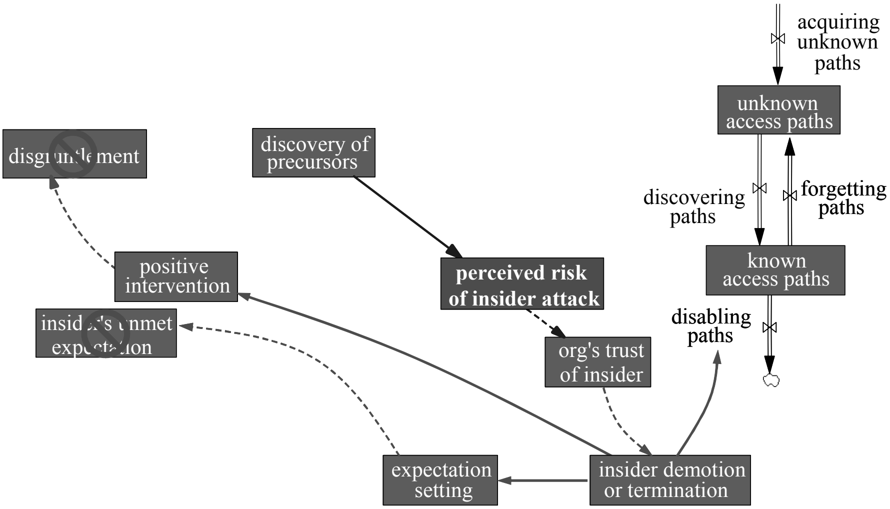


Fig. 10: Measures Upon Demotion or Termination

When a demotion occurs, the organization should analyze the roles and responsibilities of the new position and update authorization levels and access controls, including role-based access. Some organizations in the cases we analyzed overlooked the change in privileges, allowing the employee to retain privileges from their previous position, giving them access to information beyond that needed for their new position.

Expectation setting during a demotion or termination can be a deterrent against future attacks. The employee should be clearly told what the acceptable use policy is regarding their new position, what their roles and responsibilities are in their new role, what their performance improvement plan is (if one exists), and that future monitoring and auditing will be implemented to measure job performance against individual and organizational goals and objectives.

5 A Workshop on Insider IT Sabotage

Our insider IT sabotage workshop has the following structure:

- Insider Threat Study overview
- Interactive discussion of an instructional case of insider IT sabotage
- General observations from the insider IT sabotage cases

- System Dynamics model: problem, prevention, and mitigation
- Recommendations for countering the threat

We introduce domain concepts during the Insider Threat Study overview and in the interactive instructional case discussion, which is described below. By the time we introduce the model, participants are very familiar with the primitive domain concepts. The model then serves to bring into sharper focus the concepts previously described, with an emphasis on the dynamic interrelationship of those concepts. This approach helps ensure that workshop participants are not overwhelmed with too many new concepts, both modeling and domain, at the same time. The rest of this section provides an overview of our instructional case and general observations about insider IT sabotage. We present this material to illustrate how domain concepts are presented prior to presenting the model.[8]

5.1 The Instructional Case

A concrete case example helps to clearly illustrate the relationship between the aspects of the insider threat and the effectiveness of various measures to counter the threat. However, the sensitivity of actual *Insider Threat Study* case data precludes the use of actual cases for training. We, therefore, developed a fictional case scenario that is representative of a preponderance of actual cases of insider IT sabotage from the *Insider Threat Study*. The fictional organization is called iAssemble, Inc.[9] The full text of the iAssemble case example provided in Appendix B is a substantial revision of a previously published version [3] based on specific guidelines in the area [9]. Moore et al. [8] provides guidelines for using the instructional case in a classroom setting.

We believe that the iAssemble case provides a coherent and well-grounded basis for training on the important issues relevant to insider IT sabotage and is representative in character (but not necessarily detail) of many of the actual cases that we have seen. This fictional case deals with the events surrounding an insider IT sabotage case at iAssemble. Ian was hired during the company startup as a computer specialist and technical assistant to the original founders. With hard work and dedication, he became the sole system administrator at iAssemble, a position he held for four years. He was also responsible for building the software that ran the company's computer systems. With the increase in sales at iAssemble and its focus on making profits and meeting deadlines, iAssemble began hiring new people.

[8] Anyone interested in attending CERT's insider threat workshop can contact them at insider-threat-feedback@cert.org

[9] The iAssemble organization and case example are completely fictional; any resemblance to a real organization or insider threat case is unintentional.

After being passed over for the new lead system administrator position, Ian began acting out in the workplace. In the next few months, Ian’s disruptions to iAssemble operations grew to the point that iAssemble managers decided they had no choice but to let him go. On the day he was fired, Ian installed a malicious software program, which is generically referred to as a logic bomb, on iAssemble’s central server. The logic bomb, which detonated one week after Ian’s dismissal, deleted all of the programs supporting iAssemble’s mission critical processes that Ian himself had developed. The instructional case describes in more detail the motive behind the attack, the (non)-technical actions taken by the insider and the organization, and the impact of those actions on iAssemble.

The following four questions are used to focus the workshop participant’s attention on the issues and concepts central to understanding insider IT sabotage. They focus on

- identifying behavioral and technical precursors exhibited by the insider.
- understanding how Ian’s personal predispositions and unmet expectations caused an escalation of disgruntlement that was triggered by a precipitating event.
- technical discussions regarding the access paths into the organization’s systems that are available to the insider.

5.1.1 Question 1: Why did Ian attack iAssemble?

The concepts of unmet expectation and personal predisposition are critical to understanding why Ian attacked iAssemble. The root cause of Ian’s disgruntlement was his unmet expectation – his expectation for recognition and for control of the system. Ian enjoyed four years at iAssemble in which he had total control over the design and evolution of the company’s systems and networks. During that time, his expectation of continued control and prominence within the organization grew and became firmly entrenched. Ian’s personal predispositions exacerbated his sense of entitlement. Personal risk factors included Ian’s arrogant behavior in the workplace and his alcohol addiction problem. Ian was also under great personal stress due to family issues, which further amplified his disgruntlement at work.

5.1.2 Question 2: Why was Ian able to harm iAssemble after firing?

Discussion about this question typically focuses on how Ian accessed the system. Also relevant is the organization’s focus, prior to the attack, almost exclusively on the growth of the company with little or no recognition of the risks associated with that growth or with Ian’s actions in particular. Another key question is why iAssemble fired Ian before cutting off all access. This naturally leads to the definition of an access path, and the fact that Ian had access paths into the computer networks about which the organization did not know. For example, Ian was able to use his coworker's account to plant the logic bomb because they had shared passwords months earlier. In addition, the logic bomb itself can be viewed as an (unknown) access path in that it allows the insider to take action within the organization's network even when all of his direct connections have been severed.

5.1.3 Question 3: What could iAssemble have done to prevent the attack?

One focus of this question is to understand the importance of actions or events that occurred, or conditions that existed, prior to the insider's attack. Precursors may be both technical and behavioral in nature. For example, the sharing of passwords between Ian and a coworker facilitated the attack. The password sharing also opened an access path to the insider that the organization did not know about. In this case, the organization may have closed this avenue of attack by

- prohibiting password sharing by policy, reinforced through periodic security awareness training.
- instituting regular password changes, including administrator or other group accounts.
- requiring all employees to change their passwords when Ian was fired.

Another focus of this question is to discuss how the organization could have used the knowledge of precursors to prevent the insider attack or otherwise mitigate the risk of attack. While some precursors can be prevented with minimal cost, others are better detected and responded to on a case-by-case basis. Behavioral precursors are often one of the first indicators of employee disgruntlement. If an organization is successful in identifying these precursors and taking measures to address them in a timely fashion, they might be successful in preventing attacks. This depends on perceptive management and targeted behavioral monitoring.

Technical precursors to an attack are even more serious and usually follow but may come in parallel with behavioral precursors. They may, by themselves, cause disruption in the organization's systems. They often indicate steps taken to set up a future attack on the organization's systems, possibly unbeknownst to the organization, such as creation of malicious code. Other technical precursors simply enable the insider to conceal his malicious acts. For instance, insiders often create fictitious (backdoor) accounts for their surreptitious entry to the system at a later date. This is an example of an access path that is not known by management. The organization needs to have technical monitoring in place to be able to detect such precursors at an early stage and they must take appropriate actions. While behavioral precursors, by themselves, are indicative of insider threat risk, the combination of technical and behavioral precursors indicates an even greater risk of insider attack.

5.1.4 Question 4: What should iAssemble do in the future?

This question requires that participants take a step back from the details of the particular scenario to describe what iAssemble should do in the future to ensure that the risk of insider IT sabotage is acceptably mitigated. Effective risk mitigation strategies should focus as much on understanding and reducing the impact of possible attacks as it does on preventing them in the first place. Organizational focus should be on those malicious acts with the largest potential impact to the organization.

Of course, organizations cannot prevent all malicious acts. They cannot even always monitor for all precursors equally. Difficult decisions must be made, especially regarding monitoring for technical precursors, due to their associated costs. Resources used to audit and monitor technical accounts and activities may divert effort from project deliverables. Comprehensive monitoring of all employees is often not cost effective. Organizations can, however, implement proactive monitoring and logging of all staff for a key set of precursors. When circumstances dictate an increased risk, they should engage in more detailed, targeted monitoring.

There are also difficult questions regarding which measures should be used to mitigate risks. Should the organization use technical measures like restricting access to curtail the risk of insider attack? Should it use non-technical measures like a warning or reprimand for concerning behaviors? An organization needs to take into account the effect of these technical measures on morale and productivity as well as risk. The organization also needs to be aware of access paths available to the insider, including indirect access paths like malicious code.

If choosing non-technical measures to reduce risk, the organization needs to consider positive intervention, such as constructive dialogue with employees or Employee Assistance Program (EAP) referrals, in addition to punitive techniques, such as reprimands or sanctions. For certain insiders, punitive techniques may increase the insider’s disgruntlement to the point that they have little regard for future repercussions. Excessive or unreasonable monitoring within the workplace may make employees feel like they are being watched by “big brother.” Low levels of trust within the workplace can discourage employees, creating an environment of low morale and low productivity. Of course different workplace cultures accept different levels of monitoring. Organizational management has to find the right balance between providing a trusting workplace environment and managing risks associated with insider IT sabotage.

6 Conclusion

This paper describes how we have used the empirical data, in combination with our library of insider threat cases, to create materials for raising awareness on insider threat and risk mitigation. We describe our use of System Dynamics modeling to better communicate the nature of the insider IT sabotage problem and potential leverage points for its mitigation. The MERIT model effectively combines psychological and technical findings from the joint CERT/U.S. Secret Service *Insider Threat Study* with other insider threat expertise from participating researchers. MERIT has greatly enhanced our ability to lead facilitated training on the risk of insider IT sabotage. This section describes the value of System Dynamics modeling toward our better understanding of the insider IT sabotage problem, and the directions for our ongoing and future work.

6.1 Value of Modeling for Insight

We found that the System Dynamics approach helped to structure and focus the team's discussion. This was important since members of the team, by necessity, came from the different disciplines of psychology and information security.

By identifying the primary variables of interest, the influences between these variables, and the feedback loops that are so important for understanding complex behavior, the team found itself able to communicate much more effectively. The group modeling process enabled the team to step back and consider the "big picture" at times and focus on individual concepts at other times. The rigorous notation helped identify commonalities to simplify the models and prevent misunderstandings that could have hindered progress otherwise. In addition, it was immensely valuable for each team member to be able to come away with the models that we developed after our group sessions and devote individual thought to each. It not only documented our progress but helped us pick up from where we left off after a period of downtime and reflection on what we had accomplished. The models also provided a concrete target for validation through mapping to observables exhibited by the real-world cases.

Significant methodological and data challenges must be overcome before research on insider activity can be soundly prescriptive for mitigation policies, practices, and technology. However, we cannot overestimate the importance of looking at the total context of adverse insider behavior for understanding why these events happened and how they might be prevented in the future. By using the System Dynamics approach we attempt to assess the weight and interrelatedness of personal, organizational, social, and technical factors as well as the effectiveness of deterrent measures in the workplace. Prospective studies of these phenomena will always be challenging because of low base rates. In the meantime, System Dynamics modeling using available empirical data can bridge this methodological gap and translate the best available data into implications for policies, practices, and technologies to mitigate insider threat.

6.2 Related CERT Research

Ongoing and future research into insider threat at CERT includes three areas:

- a broader study of insider threat
- the development of an insider threat risk diagnostic
- the development of a training simulation for improved insider threat risk education, awareness, and mitigation

This section summarizes each of these areas of work.

6.2.1 Broader Study of Insider Threats

The library of assets produced by the MERIT project provides a collection of tools that have been very effective in transitioning our knowledge of insider IT sabotage to an international audience of security experts, IT practitioners, all levels of government and business managers, and law enforcement. Insider threat workshop participants appreciate the interactive nature of the initial discussions and the use of the model to interrelate important, but complex, insider IT sabotage domain concepts. They have one primary complaint – they need to understand insider fraud and theft of confidential or sensitive information in the same depth that the workshop provides for insider IT sabotage. Results of the Insider Threat Study show that insider fraud using IT is a significant problem in industry, especially in the banking and finance sector [Randazzo 04]. Likewise, theft of information using IT, including crimes like identity theft and corporate espionage, is a significant problem in today’s privacy-conscious and competitive corporate world. Case data collected on these two crimes bolsters this need by showing their significant difference as compared to IT sabotage crimes, especially in insider motivation, insider characteristics and the technical nature of the malicious activity [4].

The primary objective of our broader study is to extend MERIT to include a comprehensive pattern analysis and transition mechanism for *all* types of insider threat, including fraud, theft of confidential or sensitive information, and IT sabotage. Outputs of this project will include a complete package of empirically-based insider threat System Dynamics models, as well as a full day insider threat workshop that includes in-depth analysis and interactive discussion of the behavioral and technical aspects of insider fraud, theft of confidential or sensitive information, and IT sabotage. We expect that participation in the workshop will empower corporate and government personnel to develop comprehensive, efficient, and justifiable defenses to insider threats along with the organizational understanding and support needed to maintain a strong security posture over time.

In addition to looking in detail at insider fraud and theft of information cases, we are now collecting and analyzing insider compromises that have occurred since 2002. This extends the terms of analysis of the original *Insider Threat Study,* which analyzed insider compromises against U.S. critical infrastructure sectors occurring from 1996 to 2002. A focus of this broader analysis will be to determine how the insider threat is evolving as well as to generate a larger dataset on which to base findings. We plan to update our common sense guide to insider threat prevention and detection based on the results of this work [4].

6.2.2 Insider Threat Risk Diagnostic Instrument

The objective of this project is to build a comprehensive diagnostic instrument which is empirically based on all of our prior insider threat research that can be used by organizations to self-assess their insider threat risk, with the ultimate goal of improving the resiliency and survivability of the organization. The insider threat risk assessment diagnostic will enable organizations to gain a better understanding of current insider threat activity and an enhanced ability to assess and manage associated risks. It will merge technical, organizational, personnel, and business security and process issues into a single, actionable framework. As in our past projects, our project team includes psychological and technical expertise. The instrument will be structured to encompass all stakeholders in the fight against insider threat: management, information technology, human resources, and physical security.

We will build a pilot instrument based on over 200 insider threat cases in the CERT case library, and will continue to expand our library with recent cases for inclusion in this research. We welcome collaboration with external organizations on this project. Collaboration opportunities range from review of the instrument to confidential sharing of insider case and/or best practice information for inclusion in the instrument. In return for participation, we will offer those organizations opportunities to pilot the insider threat risk assessment diagnostic. Following each pilot, we will provide them with a confidential report on the findings of the pilot, and suggestions for their improvement. As with all of our insider threat research, all collaborations will remain confidential and no references will ever be made to any organizations and/or individuals.

6.2.3 Training Simulation for Insider Threat

A project that started in September 2006, called MERIT-*Interactive*, builds upon the MERIT foundation to develop a stand-alone tool that can be used for more effective widespread training on insider threat risk education, awareness, and mitigation. In collaboration with Carnegie Mellon's Entertainment Technology Center, we used state of the art multi-media technologies to develop a compelling training simulation (which we call MERIT-*Interactive*, or MERIT$_{IA}$ for short) that immerses players in a realistic business setting from which they first make decisions regarding how to prevent, detect, and respond to insider actions and then see the impacts of their decisions in terms of key performance metrics.

The MERIT-*Interactive* proof of concept provides

- a stand-alone, multi-media training simulation for interactive and independent hands-on analysis of the effects of decisions regarding policies, practices, and technology on malicious insider activity based on the MERIT model for IT sabotage.
- an effective means to communicate insider threat risks and tradeoffs, useful for both technical and non-technical personnel, from system administrators to corporate CEOs.
- the state of the practice information regarding insider threats and effective countermeasures.

We finished the proof of concept and now seek funding to develop a production version of the tool. We believe that MERIT$_{IA}$ will ultimately help decision-makers better understand risk from insider threat and the role their decisions play in promoting or mitigating that risk.

Acknowledgments

CERT would like to thank the Army Research Office and Carnegie Mellon University’s CyLab for funding this project.

CERT would also like to thank the following individuals for their collaboration on the MERIT model.

- Dr. Eric D. Shaw - Consulting & Clinical Psychology, Ltd., and a visiting scientist at CERT
- Dr. Stephen R. Band - Counterintelligence Field Activity – Behavioral Science Directorate
- Dr. Lynn F. Fischer – U.S. Department of Defense Personnel Security Research Center
- Dr. Elise A. Weaver – jointly as faculty at Worcester Polytechnic Institute and visiting scientist at CERT

Their expertise and experience in the psychological and social sciences areas have enabled a much richer treatment of the insider threat problem than would have otherwise been possible.

CERT also appreciates the work and dedication of the *Insider Threat Study* team members from CERT and the U.S. Secret Service, National Threat Assessment Center; without the study none of our follow-on insider threat research would have been possible.

Finally, Christopher Nguyen - a student at the Information Networking Institute of Carnegie Mellon University, Eric Hayes – our CERT technical editor, and the anonymous reviewers for the 2007 International Conference of the System Dynamics Society provided many comments which improved both the content and presentation of this paper.

References

[1] Anderson, D.F.; Cappelli, D.M.; Gonzalez, J.J.; Mojtahedzadeh, M.; Moore, A.P.; Rich, E.; Sarriegui, J.M.; Shimeall, T.J.; Stanton, J.M.; Weaver, E.; and Zagonel, A. 2004. Preliminary System Dynamics Maps of the Insider Cyber-Threat Problem. *Proceedings of the 22nd International Conference of the System Dynamics Society,* July 2004. Available at http://www.cert.org/archive/pdf/InsiderThreatSystemDynamics.pdf.

[2] Band, S.R.; Cappelli, D. M.; Fischer, L.F.; Moore, A. P.; Shaw, E.D.; and Trzeciak, R.F 2006. "Comparing Insider IT Sabotage and Espionage: A Model-Based Analysis" Software Engineering Institute Technical Report CMU/SEI-2006-TR-026, Carnegie Mellon University, December 2006. http://www.cert.org/archive/pdf/06tr026.pdf.

[3] Cappelli, D. M.; Desai, A. G.; Moore, A. P.; Shimeall, T. J.; Weaver, E. A.; and Willke, B. J. 2006a. "Management and Education of the Risk of Insider Threat (MERIT): Mitigating the Risk of Sabotage to Employers' Information, Systems, or Networks." Proceedings of the 24th International System Dynamics Conference. Nijmegen, Netherlands, July 2006. http://www.albany.edu/cpr/sds/conf2006/proceed/proceed.pdf.

[4] Cappelli, D.M.; Moore, A.P.; Shimeall, T.J.; and Trzeciak, R.J. 2006b. "Common Sense Guide to Prevention and Detection of Insider Threats: Version 2.1," *Report of Carnegie Mellon University, CyLab*, and the *Internet Security Alliance*, July 2006 (update of the April 2005 Version 1.0). http://www.cert.org/archive/pdf/CommonSenseInsiderThreatsV2.1-1-070118.pdf

[5] Keeney, M.M.; Kowalski, E.F.; Cappelli, D.M.; Moore, A.P.; Shimeall, T.J.; and Rogers, S.N. 2005. Insider Threat Study: Computer System Sabotage in Critical Infrastructure Sectors. *Joint SEI and U.S. Secret Service Report*, May 2005. Available at http://www.cert.org/archive/pdf/insidercross051105.pdf.

[6] Meadows, D. L.; Behrens, W. W.; Meadows D. H.; Naill, R. F.; Randers, J.; and Zahn, E. K. O. 1974. *Dynamics of Growth in a Finite World*. Cambridge, MA: Wright-Allen Press, Inc..

[7] Melara, C.; Sarriegui, J.M.; Gonzalez, J.J.; Sawicka, A.; and Cooke, D.L. 2003. A System Dynamics Model of an Insider Attack on an Information System. *Proceedings of the 21st International Conference of the System Dynamics Society* July 20-24, New York, NY, USA.

[8] Moore, A.P.; Joseph, H.G.; Trzeciak, R.F.; Cappelli, D.M. 2007. Instructional Case of Insider IT Sabotage: An Instructor's Manual, in preparation.

[9] Naumes, W.; and Naumes, M.J. 1999. *The Art & Craft of Case Writing*. Thousand Oaks, California: SAGE Publications.

[10] Rich, E.; Martinez-Moyano, I.J.; Conrad, S.; Cappelli, D.M.; Moore, A.P.; Shimeall, T.J.; Andersen, D.F.; Gonzalez, J.J.; Ellison, R.J.; Lipson, H.F.; Mundie, D.A.; Sarriegui, J.M.; Sawicka, A.; Stewart, T.R.; Torres, J.M.; Weaver, E.A.; and Wiik, J. 2005. Simulating Insider Cyber-Threat Risks: A Model-Based Case and a Case-Based Model. *Proceedings of the 23rd International Conference of the System Dynamics Society,* July 2005.

[11] Sterman, J.D. 2000. Business Dynamics: Systems Thinking and Modeling for a Complex World. New York, NY: McGraw-Hill.

[12] Yin, R.K. (2003). *Case Study Research.* (3 ed.) Thousand Oaks: Sage Publications.

Appendix A: System Dynamics Background

System Dynamics is a method for modeling and analyzing the holistic behavior of complex problems as they evolve over time. System Dynamics has been used to gain insight into some of the most challenging strategy questions facing businesses and government for several decades. System Dynamics provides particularly useful insight into difficult management situations in which the best efforts to solve a problem actually make it worse. Examples of these apparently paradoxical effects include the following [11].

- Low-nicotine cigarettes, supposedly introduced to the benefit of smokers’ health, that only result in people smoking more cigarettes and taking longer, deeper drags to meet their nicotine needs
- Levees and dams constructed to control floods that only produce more severe flooding by preventing the natural dissipation of excess water in flood plains

The *Insider Threat Study* found that intuitive solutions to problems with employees often reduce the problem in the short term but make it much worse in the long term. For example, employee termination might solve an immediate problem, but it may also lead to long-term problems for the organization if the insider has the technical means to attack the system following termination. System Dynamics is a valuable analysis tool for gaining insight into long-term solutions and for demonstrating their benefits.

A powerful tenet of System Dynamics is that the dynamic complexity of problematic behavior is captured by the underlying feedback structure of that behavior. We decompose the causal structure of the problematic behavior into its feedback loops to understand which loop is strongest (i.e., which loop’s influence on behavior dominates all others) at particular points through time. We can then thoroughly understand and communicate the nature of the problematic behavior and the benefits of alternative mitigations.

System Dynamics model boundaries are drawn so that all the enterprise elements necessary to generate and understand problematic behavior are contained within them. This approach encourages the inclusion of soft (as well as hard) factors in the model, such as policy-related, procedural, administrative, or cultural factors. The exclusion of soft factors in other modeling techniques essentially treats their influence as negligible, which is often not the case. This endogenous viewpoint helps show the benefits of mitigations to the problematic behavior that are often overlooked, partly due to a narrow focus in resolving problems.

In this project we rely on System Dynamics as a tool to help understand and communicate contributing factors to insider IT sabotage and espionage threats and implications for various mitigation strategies and tactics. It is tempting to use the simulation of the model to help predict the effect of mitigation strategies, but what is the nature of the types of predictions that System Dynamics facilitates? Dennis

Meadows offers a concise answer by categorizing outputs from models as follows [6].

1. Absolute and precise predictions (Exactly when and where will the next cyber attack take place?)
2. Conditional precise predictions (How much will it cost my organization if a cyber-attack occurs?)
3. Conditional imprecise projections of dynamic behavior modes (If a bank mandates background checks for all new employees, will its damages from insider fraud be less than they would have been otherwise?)
4. Current trends that may influence future behavior (What effect will current trends in espionage have on national security in five years?)
5. Philosophical explorations of the consequences of a set of assumptions, without regard for the real-world accuracy or usefulness of those assumptions (If a foreign country succeeds in human cloning, how would this affect the United State's risk of espionage?)

Our models and System Dynamics models, in general, provide information of the third sort. Meadows explains further that "this level of knowledge is less satisfactory than a perfect, precise prediction would be, but it is still a significant advance over the level of understanding permitted by current mental models."

As described in the main body of this paper, we have modified the System Dynamics causal loop diagram notation to be more suitable for the expected participants of our workshop. Arrows still represent the pair-wise influence of the variable at the source of the arrow on the variable at the target of the arrow, but their look indicates how they should be interpreted:

- Roughly, a *solid* arrow indicates a *positive* influence - that the value of the source and target variables moves in the *same* direction.[10]
- Roughly, a *dashed* arrow indicates a *negative* influence - that the value of the source and target variables moves in the *opposite* direction.[11]

As mentioned, dynamically complex problems can often be best understood in terms of the feedback loops underlying those problems. There are two types of feedback loops: *balancing* and *reinforcing*.

[10] More formally, a *solid* arrow indicates that if the value of the source variable increases, then the value of the target variable increases above what it would otherwise have been, all other things being equal. And, if the value of the source variable decreases, then the value of the target variable decreases below what it would otherwise have been, all other things being equal.

[11] More formally, a *dashed* arrow indicates that if the value of the source variable increases, then the value of the target variable decreases below what it would otherwise have been, all other things being equal. And, if the value of the source variable decreases, then the value of the target variable increases above what it would otherwise have been, all other things being equal.

- Balancing loops (labeled *B#* in the figures) describe the system aspects that oppose change, tending to drive organizational variables to some goal state. In other words, balancing loops tend to move the system to an equilibrium state even in the face of change. The behavior of a thermostat is an example of a balancing loop. It continually changes the air flow into a room based on the temperature of the room, with the goal of maintaining an equilibrium temperature.
- Reinforcing loops (labeled *R#* in the figures) describe the system aspects that tend to drive variable values consistently upward or consistently downward. In other words, reinforcing loops can "spiral out of control." A flu epidemic is an example of a reinforcing loop. It spirals out of control as more and more people contract the flu.

The type of a feedback loop is determined by counting the number of negative influences along the path of the loop. An odd number of negative influences indicates a balancing loop, and an even (or zero) number of negative influences indicates a reinforcing loop.

System Dynamics models are described as a sequence of feedback loops that characterize how the problem unfolds over time. Each feedback loop describes a single aspect of the problem. Multiple feedback loops interact to capture the complexities of the problem domain.

Appendix B: The Insider IT Sabotage Training Case[12]

1 Introduction

Chris, president of the computer systems sales company *iAssemble*, felt like he had just been hit by a Mack truck.

A partner in the company, Caroline, explained to him that something had just wiped out their system configuration and assembly programs. “And to top it off,” she continued, “the only backups were given to Ian before he was fired and we haven’t seen them since. Given the circumstances of Ian’s departure, we suspect that he might be responsible.”

“I just can’t believe that Ian would do something like that,” said Chris. “He’s been with the company since the beginning; he wrote most of those programs himself, for crying out loud!”

Chris paused and looked back at Caroline, “What the heck do we do now?”

1.1 Background

iAssemble sold computer systems directly to customers, building each system made-to-order and offering competitive prices. iAssemble had been doing extremely well and conducted an initial public offering (IPO) in 2001, after which its stock doubled.

Chris started the company in 1997 with his friend Caroline, who is now the Chief Technology Officer (CTO). The company has had success hiring experienced managers and employees since the beginning.

Ian was among the few employees who had been with iAssemble since its establishment. Ian started out as computer specialist and technical assistant to the two original founders, Chris and Caroline. When hired, Ian held certifications in personal computer (PC) hardware maintenance and operating system administration. Although he did not possess a college degree, with hard work and dedication, he became the sole system administrator at iAssemble, a position he held for four years. He also built the software that ran the computer system assembly machinery pretty much from the ground up. This software was the foundation for automating the PC assembly processes that allowed iAssemble’s rapid growth.

iAssemble grew at an increasing rate. Recognizing the need for qualified personnel, Chris and Caroline began to hire experienced system administrators who could also function as project managers. Lance was hired as lead system adminis-

[12] The iAssemble organization and case example are completely fictional, any unintentional resemblance to a real organization or insider threat case is unintentional.

trator because of his education and experience, and, James was hired as a junior system administrator to share Ian's growing systems administration workload and responsibilities.

Ian was assigned to mentor James and ensure his smooth assimilation within the company. Ian and James worked on several projects together. Ian respected James's abilities, finding him to be nearly as technically competent as himself. The two of them got along fine, but James began to notice that Ian was becoming increasingly short-tempered over a period of several months.

One day after Ian's moodiness started annoying him, James decided to find out what the problem was. "Hey, what's bugging you Ian?" inquired James.

As if he had been waiting for someone to ask that question, he dumped on James. "James, I can't take it any longer. That idiot Lance thinks he knows how to run these networks better than me. I know these systems inside and out. The changes that he is suggesting will bring the network to its knees. They've got me on these piddly projects while they are destroying the foundations that I laid for iAssemble."

"You should be managing these networks, Ian," suggested James. "Why didn't they make you lead admin?"

"I have no idea, but who wants to be pushing papers all day long, anyway" Ian interrupted. "Lance is perfectly suited to that, but he doesn't know the first thing about running these networks. They simply don't appreciate what I do around here and some day they may just regret it."

Ian's disgruntlement grew and it became obvious in his hostile dealings with coworkers. He even bottlenecked projects on purpose on several occasions, stalling his work to ensure Lance and the team missed project milestones. Ian received a written warning from Caroline after several coworkers formally complained. Enraged by this, he had a heated argument with a team member who quit the very next day, citing Ian as the reason for his resignation. Caroline sent Ian a letter of final warning that put him on probation for his conduct – any more such problems would result in his immediate termination from iAssemble.

This seemed to resolve the situation, at least for a while. During the subsequent months, iAssemble continued to thrive. With a whopping 68 percent growth in sales over the previous quarter, iAssemble was forced to hire additional people. The staff adopted a "do whatever it takes" attitude to their job in order to keep up with the demands placed on them due to the growth. One staffer described it as follows:

> "We were one lean coding machine in those days. We had to be to extend the systems to support the company's amazing growth. Ian thought of and implemented the idea to centralize the core software on a central server to coordinate updates more efficiently. We made vast improvements to the flexibility and sophistication of the assembly programs over a very short period of time. And the extensions worked well with very few glitches. Of course, we had to cut some corners, giving people access when and where they needed it to make things happen. If something did not contribute to extending the systems, it just did not seem worth doing. This was how we were able to accomplish as much as we did."

Unbeknownst to management, during these months Ian was busy developing and testing a logic bomb that would delete all of the files on the central server. He did the testing and some of the development on his office desktop machine to make sure that it would really work. He also planted a backdoor account, with administrator privileges, on the main machinery server that provided him with unconstrained access from home just in case he needed it.

Ian tried to get along with his coworkers, knowing that he would eventually get even. But he viewed most of his coworkers as incompetents; he just could not help "letting loose" on them every once in a while. He had already started looking for another job, one where his abilities were recognized and valued, so he felt that he would not be around iAssemble much longer.

1.2 The Final Weeks

Management decided that they needed to deal with Ian's lingering performance problems. In a meeting with Chris and Caroline, Lance complained, "Ian is an arrogant jerk. He harasses and bullies his coworkers, treating them like they are dirt under his feet. Larry, the new programmer that we hired a few months ago, suspects that Ian is messing with the code that he is developing to make him look bad. Most of the staff walks on eggshells around him. We've got to do something."

"How big a hole is it going to leave in development and operations if we fire him?" asked Chris.

"Virtually none," replied Caroline, "with all of our hiring and aggressive training programs over the past few months, the rest of the staff is well up to speed on how things run and the directions that we are going. Both James and I think that we'd be a whole lot better off without him."

"OK," replied Chris, "Caroline, you take care of this yourself. Make sure to coordinate with James to be darned sure you cut off his access before you let Ian know. I feel bad about this – Ian has been with us since the beginning, but he has brought this on himself. So let's make it happen. How soon should we do it?"

"The sooner the better in my book." said Caroline, "I'll schedule to meet with him this morning. Lance, you disable his access while I'm meeting with him, and I'll have security escort him from the building after the meeting."

Chris's comment about disabling Ian's access had left Lance with concerns. Security practices at iAssemble had been less than rigorous lately, with the push to get the new software out. But Lance decided not to voice them at the meeting and later that day on July 10, 2001, Ian was fired, his access was disabled, and he was escorted from the building just as they had planned. Passwords for all shared accounts, including the system administrator accounts, were changed while Caroline was meeting with Ian.

Unfortunately, iAssemble managers were not aware that James had shared his password with Ian months earlier in order to make the development process easier.

Ian went home the night he was fired and successfully logged into James's account. He then used his backdoor account on the machinery server to plant the logic bomb. He set it to go off one week later.

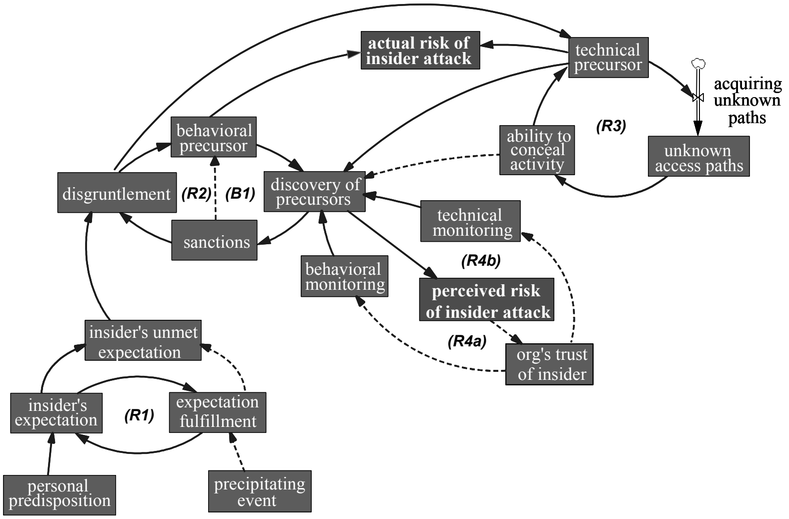
After the logic bomb detonated, Caroline was in Chris's office explaining that their critical software had been wiped out. Chris was puzzled as to how that was possible in light of the monitoring, policies, and security practices in place at iAssemble. After numerous hours of investigation, the system logs were used to trace the access of the machinery server to James's account. The evidence pointed to James as the saboteur. James claimed that he was not responsible for the deleted software and explained that he had given the password to his computer to Ian when they worked together. According to James, it had been such a long time ago that it had slipped his mind.

Management decided to call in law enforcement. Forensics analysis revealed that James's account was accessed remotely from Ian's home. Analysis of Ian's computer showed that he tested the logic bomb four times over a period of three months. When questioned, Ian continued to maintain his innocence, even though the evidence against him was substantial. Investigation into Ian's background revealed that his father had been suffering from lung cancer over the last year and that he had recently lost his driver's license due to a conviction for driving under the influence of alcohol.

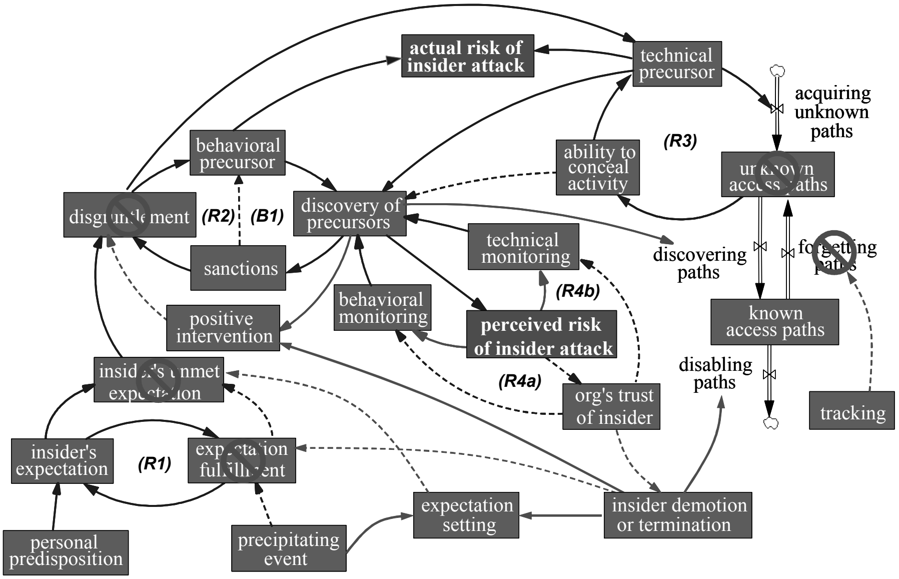
In a meeting with Caroline and Lance, Chris exploded, "We know who did it, but how do we recover from something like this? It will take months to recover operations even close to what we had. When this gets out stockholders are going to demand a detailed explanation."

Slamming his fist on his desk, Chris demanded, "We must not only understand how this happened, but why, and make sure it does not happen again!"

Appendix C: Model of the Insider IT Sabotage Problem



Appendix D: Insider Sabotage Mitigating Measures



Data Theft: A Prototypical Insider Threat

Michael McCormick, CISSP

Minneapolis, MN

Abstract The author is the lead information security architect at one of the United States' largest banks. In this paper he assesses the threat of confidential data leakage, focusing on its most virulent form -- insider data theft attacks. Technological and procedural controls typically found in enterprise environments are reviewed and found inadequate. Additional controls are proposed, and several areas for additional technical research are also suggested.

1 Introduction

1.1 Data Theft

Many potential insider threats lurk in a typical enterprise. Some pertain to the physical realm -- and hence fall largely outside the jurisdiction of information or "cyber" security -- such as merchandise theft, property destruction, or cash embezzlement. Those that do fall in the cyber realm generally involve the misuse of data or computational assets. Examples include theft or destruction of valuable data records, so-called logic bombs and software back doors, internal denial of service, and inappropriate consumption of computer or network resources.

Many 21st century enterprises find their most valuable assets shifting from physical materiel to cyber resources (atoms to electrons). The banking industry is on the forefront of this trend due to the accelerating digitalization of money. Lessons learned by banks combating insider data theft may serve as an early warning to other industries that are earlier in the asset virtualization process.

Of all insider perpetrated cyber attacks, data leakage has emerged in the last several years as one that's growing particularly fast [12], apparently causing the most damage (anecdotally), and certainly the one garnering the most sensational publicity. Such publicity in turn acts as a damage multiplier by triggering retaliatory behavior among frightened or angry customers, investors, regulators, and others who might not otherwise have known about a particular incident. Breach disclosure letters such as those required under California state law [14] act as a similar damage multiplier.

Data theft is a prototypical inside attack, in that it has much in common with other insider threats. The perpetrator profile is a common one (as we shall see in

2.3.1). Effective preventive or detective controls are difficult to define or implement. Since no single control is broadly effective, defense in depth is required, ideally spanning both technical and administrative controls in holistic fashion. Many of those controls turn out to have applicability to insider threats beyond data theft, as we shall see.

1.2 Data Leakage

Data theft is also interesting because it's part of a larger insider threat, *data leakage*, which includes accidental or unintentional data loss in addition to malicious theft. Inadvertent data loss is actually more common than theft. Many insider threats come in both malicious and non-malicious varieties, but security staff sometime has a natural tendency to focus on the former, and in doing so may miss opportunities to neutralize two birds with one stone.

A "data leak" generally refers to sensitive customer or corporate information electronically leaving the enterprise environment either inadvertently or deliberately. Hence data theft is a malicious subclass of data leak, and most controls that prove effective against data leaks can also mitigate data theft (although the converse isn't necessarily true).

Through incidents and observed trends over the past year or longer, it has become clear to many enterprises that they are increasingly vulnerable to a serious data leak that could damage them financially, legally, and reputationally. Confidential data is increasingly found in inappropriate places (both inside and outside the enterprise that owns it). Many private sector companies have embarked on enterprise data leak prevention (EDLP) programs to deal with this threat. A market of data loss prevention (DLP) technology vendors has sprung into being to profit from this development.

1.3 Risk

The level of damage inflicted by a data leak can be catastrophic, particularly a large-scale deliberate leak motivated by revenge. Damages come from customer attrition (e.g., closed accounts), subsequent fraud and identity theft committed with stolen data, shareholder sell-offs and retaliatory votes, lawsuits from investors or customers or partners, government investigations and penalties, and loss of reputation.

A typical enterprise is vulnerable to a wide variety of possible leaks, ranging from programmers or database analysts stealing customer data from production systems to sell on the black market, to managers taking sensitive data out of the office on a USB memory stick that gets stolen.

1.4 Recommendations

This paper analyzes current controls in place at a typical corporate enterprise, both technical and procedural, and finds them inadequate. We propose using current controls more effectively. We propose adoption of newer security technologies including DLP suites, data discovery/classification tools, content filters, database and application security measures, and removable media controls. We also propose a variety of procedural controls including policies and standards, clearance levels, employee and manager training, database best practices, and rigorous audits. We also propose several areas for long term research.

2 Status Quo

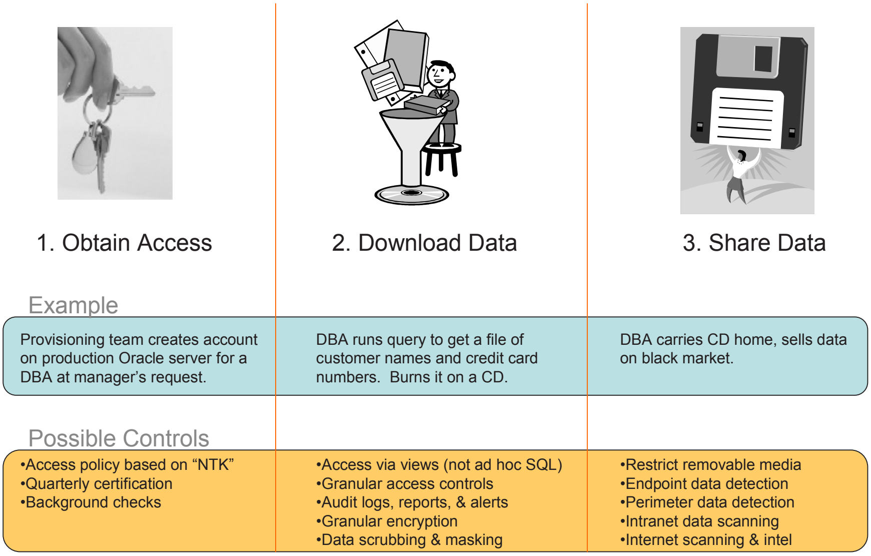
2.1 History

Several trends converged in 2007 leading to wide recognition that a typical organization needs a unified enterprise strategy to prevent data leaks:

- Secret Service [1] and CERT [6] both identified an increasing threat from inside attackers.
- PCI [15] auditors expressed concern about potential credit card information leak or loss.
- Regulators in some federally regulated industries expressed concern about handling of customer TIN and SSN data.
- Sensitive enterprise data was found on peer-to-peer public file sharing networks [17].
- Highly publicized data theft incidents caused enormous damage at CardSystems, TJX, Fidelity, and numerous other companies.
- In the financial services industry, many companies began implementing data leak prevention programs and systems, according to surveys conducted by FS-ISAC [10] and FSTC [11].

2.2 Risks & Controls

To understand both the risks and controls, one must consider the three major stages of a data theft and what controls are typically available at each stage:



	1. Obtain Access	2. Download Data	3. Share Data
Example	Provisioning team creates account on production Oracle server for a DBA at manager's request.	DBA runs query to get a file of customer names and credit card numbers. Burns it on a CD.	DBA carries CD home, sells data on black market.
Possible Controls	•Access policy based on "NTK" •Quarterly certification •Background checks	•Access via views (not ad hoc SQL) •Granular access controls •Audit logs, reports, & alerts •Granular encryption •Data scrubbing & masking	•Restrict removable media •Endpoint data detection •Perimeter data detection •Intranet data scanning •Internet scanning & intel

Figure 1- Stages of Data Theft

In Figure 1 we see the three stages of data leak or theft, along with an example involving a malicious Oracle DBA, plus some common security controls applied at each stage of the leak.

2.2.1 Attacker Profile

The fictitious crooked Oracle DBA of Figure 1 is just one possible example among many. However, studies of malicious leak perpetrators have shown a common profile. Typically the leaker is a trusted insider whose privileges and position of trust create tempting opportunities for data theft. Often the leaker is a disgruntled IT staffer. According to Carnegie Mellon CERT [7] the attack is often carried out after termination, typically using remote or physical access that hasn't been deprovisioned yet, or in some cases logic bombs left behind in the employer's environment.

Telltale behavior often precedes a malicious leak but coworkers either ignore it (until 20-20 hindsight after the fact) or don't report it. According to Carnegie Mellon, 60% of convicted inside attackers had clear behavioral "red flags" [7]. Usually some stressor in the perpetrator's work or personal life acts as a trigger. Typical workplace triggers include negative performance reviews, missed raises, and loss or perks.

The leaker's motive is almost always revenge and/or profit. The money motive leads to data theft alone, but the revenge motive can also lead to data destruction.

The leaker sometimes has an outside partner. The partner often has organized crime ties. The partner's role could be to finance or otherwise facilitate the attack.

In some cases the partner may simply be the buyer of stolen data after theft is complete.

It must be mentioned here again that not all data leakage is intentional; in fact the majority of it is accidental. Common examples include disposing of records, media, or equipment without following proper data destruction procedures. In such cases the leaker obviously does not fit the profile described here. In most such cases the leaker is simply unaware of correct procedures and policies, or may have forgotten that the records, media, or computers in question contain confidential information. A comprehensive EDLP strategy can mitigate inadvertent leakage as well as intentional data theft.

2.2.2 Damage Potential

The level of damage inflicted by a data leak can be catastrophic, particularly a large-scale deliberate leak motivated by revenge. Damages come from customer attrition (closed accounts), fraud committed using stolen data, identity theft, shareholder sell-offs and retaliatory votes, lawsuits from shareholders or customers or partners, government investigations and penalties, and loss of reputation.

If breach disclosure letters must be sent to affected customers, these can act as a damage multiplier due to negative PR, increased attrition (even customers not directly affected may close their accounts), cost of free credit reports, etc. Breach disclosure laws modeled after California SB1386 [14] are now going national. As an industry rule of thumb, a data breach costs about $250 per customer notified.

An example of the damage a large data theft can cause is TJX, which reportedly lost at least 45 million credit card numbers to unidentified attackers. In addition to a storm of bad publicity, TJX faced congressional hearings, FTC investigation, probes by several State Attorney Generals, lost customers, a class action lawsuit from customers, drop in stock price, lawsuits from shareholders, and lawsuits from banks.

2.2.3 Current Threat

The primary risk faced at a typical corporate enterprise today is that insiders & others with permission to read files or query databases containing customer data could copy this data. The most damaging misuse case would be large-scale intentional data theft. However the most common case would be copies made with good intentions (e.g., for offline use or backup) that fall into the wrong hands once outside the controlled environment.

Detection would only be likely if this inappropriate use is discovered serendipitously, such as a co-worker overhearing the leaker admit to copying customer data, or through law enforcement tying evidence of customer account data leakage back to the insider. Stolen customer data has been known to fall in the hands of criminals who use it to launch phishing attacks, commit identity theft, or commit other crimes. An enterprise may only become aware of data theft for the first time at this final stage of the attack, when a subsequent crime is committed outside the enterprise.

Example scenarios:

- DBA runs a query, export or backup which is then copied to removable media and taken off premises.
- Finance/Compliance analyst runs a report containing detailed customer listing which is then copied to removable media, printed, or encrypted and emailed.
- Fraud analyst runs a report containing detailed customer listing which is then copied to removable media, printed, or encrypted and emailed.
- LAN administrator takes copy of detailed customer listing from authorized individual's LAN or local storage.

Of course the threat is not limited to customer data. Employee, patient, or student records, information about suppliers or partners, business secrets (e.g., M&A plans), and other types of information assets or intellectual property are all valuable targets. However customer information is the most likely target in many private sector enterprises because it's available in large quantities and there is a ready-made black market on which it can be sold easily and anonymously.

2.2.4 Current Technical Controls

A variety of technologies are widely deployed which are intended to serve as preventive or detective controls to mitigate data leakage.

Data protection starts at the source. Database access controls are generally used on production tables with sensitive data, and ideally should restrict access to data based on need. In some cases database views or column based access controls are in place for more granular security. For some databases a provisioning process (possibly automated) ensures users are only granted access with proper approval. However due to complexities in access levels, approvers may not always understand what access they are approving. Certifications, attestations, or periodic audits should be performed to validate access.

Database access logging or audit trails are common on production systems with sensitive data, but the extent and quality of logging can be inconsistent. Different database platforms provide varying levels of log effectiveness natively. Even when logs are good, they often may not be reviewed or monitored adequately.

Database encryption is sometimes available (natively or through third-party solutions) but performance and operational concerns have limited its adoption. Also the effectiveness of database encryption to prevent misuse by authorized users is reduced if access is permitted via shared application service accounts.

Developer access to production databases in theory should be severely limited or non-existent. But in practice, segregation of duties within an application team between those who perform development and those who perform production support or other operational duties (sys admin, DBA, etc.) is sometimes inadequate. On a small application team one individual may be required to wear many hats. Even on a larger team it is sometimes necessary to grant production access permissions to a developer for troubleshooting purposes, if only temporarily.

Furthermore, sensitive data sometimes migrates from production environments into development or quality assurance environments because the quality and quantity of real production data is generally preferable to artificial data for testing purposes. Data scrubbing tools should be in place to permit programmers or testers to perform their work with sanitized data, eliminating the need to provide them live production data. However adoption and deployment of such tools is far from ubiquitous, and maturity of these solutions varies by platform.

Once data leaves a database, the main defense against theft or leaks is traditional access controls such as operating system file ACLs. For example, a file export or report may sit on a desktop hard drive or network file share, but should have an access control list (ACL) that prevents anyone besides its rightful owner from reading it. Unfortunately such an ACL usually allows privileged or “root” users with OS administrator rights to access the file.

2.2.5 Current Administrative Controls

Personnel (HR) rules and procedures for employees include “soft” administrative controls intended to prevent confidential data leakage. Examples include a corporate customer privacy policy or employee ethics training.

Background checks in theory could screen out employees pre-disposed toward or with a history of careless or malicious behavior. However, studies have shown that one third of convicted inside attackers had prior arrests [4], throwing the efficacy of background checks into doubt.

Corporate security or privacy policies may attempt to prescribe correct handling of sensitive information. However policies that aren’t supported by clear procedures, training, and tools are generally doomed to be ineffective or disregarded.

Regulations and laws require confidential customer data must be handled with care, thus making data leaks a compliance risk as well as a business risk. For example, PCI [15] requires credit and debit card information be tightly controlled, as described in data security standards (DSS) rules that are quite detailed and prescriptive. As of this writing, the State of Minnesota has legislated PCI DSS rules into law, and at least four other states including California are poised to follow suit. Other regulations governing customer data privacy include California SB1386 [14] and similar state laws governing breach disclosure, HIPAA, FERPA, and the Gramm-Leach-Bliley Act (GLBA).

2.2.6 Residual Risk

Administrative or “soft” controls are necessary, and to some extent do prevent accidental data leaks by educating the “good” employees about what behaviors are expected or disallowed with regard to confidential data handling. However such controls do little or nothing to deter intentionally wrongful behavior among the “bad” employees. Therefore their preventive value is inherently limited (not that they shouldn’t be improved where possible). Even among the “good employees” their value is limited. Many data leaks are not malicious, and were not even seen as “leaks” by the perpetrator but rather were intended to serve a business purpose and reflect only someone’s ignorance of policies, procedures, and risks. This highlights the inherent weakness of unenforced “soft” controls.

Let us turn to the prevailing “hard” controls, which tend to focus on database security. Their effectiveness is debatable. More importantly, no matter how strong they are, database centric controls largely miss the boat. Data compromise (both accidental and malicious) doesn’t generally occur on the system of origin. The leaked data is generally obtained from secondary sources. Data leaves a database in many ways – exports, extracts, replication, reports, and backups to name but a few. Once it leaves the controlled environment of the source system it is much more vulnerable to mishandling or theft.

There is no “silver bullet” that can eliminate the data leak risk. Indeed, any single control can likely be subverted:

- Logging controls could be circumvented by hiding access within legitimate access.
- Data discovery controls could be circumvented by manipulating the data to avoid detection (e.g. split a stolen SSN into multiple unrelated fields).
- Data leakage controls that inspect data at egress points could be circumvented by encrypting the data or otherwise manipulating it to avoid detection.
- External storage device controls could be circumvented by using other means of extracting data such as email or uploading the file to an Internet accessible web server via an SSL connection.

Nonetheless, a multi-pronged enterprise data leak prevention strategy could significantly lower the residual risk. Reducing the number of individuals with access to confidential data, reducing the number of data stores containing unmasked customer data, knowing where sensitive data is stored and transmitted, and implementing the additional controls proposed in the next section of this paper would reduce the risk of employee data theft by limiting the alternative means and providing detection deterrents.

3 Recommendations

3.1 Technical Controls

IT or information security staff should evaluate the following technologies, and adopt those that meet their requirements and work in the target environment.

3.1.1 DLP/ILP Suites

An emerging category of commercial security tools claim to prevent storage or transmission of confidential information in ways that violate security or privacy policies defined by the enterprise or by regulations (e.g., PCI DSS). Those which are full suites include modules for:

1. Data in motion – network perimeter taps, email gateway filters, etc.
2. Data at rest – hosts that scan databases, file shares, web sites, servers, etc.
3. Data in use – endpoint agents that monitor desktops and removable media

A full feature DLP suite can (configurably) take all of the following actions when relevant data is encountered:

1. Report on it
2. Trigger immediate alert
3. Destroy it
4. Relocate it
5. Block its transmission
6. Encrypt it

Considerable work has been done by Burton Group [2], Gartner [3] and Forrester [4] in categorizing and comparing commercial DLP/ILP vendors.

3.1.2 Discovery / Classification Tools

Even an enterprise deploying a robust DLP suite may need to supplement it with point solutions designed to discover and classify information in very specific niches. There are dark corners of an enterprise where the DLP tool may not be able to reach. For example, commercial products are on the market which specialize in filtering email, scanning web sites, or scanning network file shares.

All these data discovery and classification niche tools need to be carefully assessed and compared to any DLP suite the enterprise chooses. Some may prove redundant and should be retired. But others will complement (or add defense in depth to) the DLP suite and deserve a place in the enterprise toolkit.

3.1.3 Database Security

Much of the current technical control set is database centric, and sound data leak prevention should certainly start at the source. But as we saw in 2.2.4, there is much room for improvement. We can improve database security by activating or tightening many controls already available natively in many RDBMS platforms.

A typical enterprise may need to:

- Promote the use of database views more aggressively to further limit access based on need to know.
- Where possible, ensure consistent deployment of granular (column level if available) access controls in the database to reduce insiders' read access to sensitive customer attributes.
- Where granular access controls aren't available, keep sensitive data in separate tables with stronger access control.
- Expand use of granular database encryption (native or external) to reduce insiders' visibility to sensitive attributes.
- Extend data scrubbing to extracts, replication feeds, and file exports to minimize instances of sensitive data.
- Implement database security monitoring solutions or so-called "data firewalls".
- Prohibit or restrict data extracts with sensitive data.
- Use data masking to redact sensitive data values in extracts and reports.
- Where possible, configure audit logs to capture data access related to high risk users (i.e. individuals with direct access to data stores outside of applications that already log access).
- Keep logs on separate servers where administrators cannot access them.

One other thing to consider with regard to database audit logs is integration with an enterprise SIM correlation tool. This could allow more aggressive centralized monitoring of security events at the database layer, and correlation with incidents at other layers of the IT infrastructure such as the network.

3.1.4 Application Entitlements

By externalizing policy entitlement decisions and/or enforcement points ("PDP" and "PEP" in XACML nomenclature) outside of business applications into systems managed centrally (or at least separately), we enable separation of duties that can thwart data theft and other insider attacks perpetrated by programmers. While most programmers are completely trustworthy, there is always some temptation to include "back door" code in software that will grant the developer special privileges or access rights to production data and accounts. Pulling entitlement decision or policy enforcement logic out of applications removes the temptation.

Various commercial products are available, many based on the XACML industry standard, to perform this type of entitlement enforcement outside of applications [13]. Some can integrate with provisioning systems which will enable central (or delegated) administration of application permissions.

Of course rigorous code reviews, combined with sound source code control and configuration management, are another way to mitigate risks from back doors and logic bombs.

3.1.5 Removable Media

Some enterprises need to tighten control over removable media, as this is a primary vector for intentional data theft.

Physical security staff (facility guards) could subject laptops, thumb drives, etc. (especially those belonging to privileged individuals authorized to access sensitive data) to random searches. However this would likely prove costly and controversial. A technological control would be more pragmatic, even if less effective as a psychological deterrent.

Some DLP suites with endpoint agents offer the ability to enforce removable media policies on a targeted as well as enterprise wide basis. However a couple other lighter weight options are also available:

- Port lockdown products that disable USB/CD/floppy ports on selected desktops/laptops,
- Removable media encryption (RME) products that prevent access to encrypted thumb drives when plugged into foreign computers.

3.1.6 Defensive Search

The final (albeit forensic, not preventive) technical control available after a data theft is to find the stolen data in order to stop its further dissemination and/or identify the thieves or their partners. Many enterprises do this on a limited basis today through simple (manual) defensive use of Google or other search engines. For example, through "BIN googling" a financial institution can search for its bank identification number (BIN) on credit cards, debit cards, and employee purchase cards. This may turn up find leaked or stolen card numbers on web sites, ftp servers, or bulletin boards. Through other intelligence gathering methods one may also find traffickers sharing or discussing stolen data on IRC channels and other anonymous communication forums.

An enterprise must exercise caution in forensic Googling in order not to inadvertently leak the very information they're searching for. However it is possible to use the Google search API [16] to build tools that discreetly search for private data without inadvertently leaking it, in a way that automates the process so that more data (both quantity and types) can be searched for. Some regular expression matching capabilities are already available in the Google API to facilitate this.

3.2 Administrative Controls

Technology alone cannot prevent data leakage. As part of a comprehensive EDLP strategy, a typical enterprise needs to consider changes to policy, procedures, and training.

3.2.1 Policies

In too many enterprises, policies and procedures regarding sensitive data handling are piecemeal at best. The foundation of any such policy is a data classification scheme (e.g., public / confidential / secret / top secret) accompanied by clear guidelines and training on how to classify information. For people or systems authorized to handle sensitive data, procedures must spell out all requirements regarding encryption, transmission, backup, disposal, etc. for each level of data classification.

Policies should also create "Chinese walls" between internal organizations or roles to minimize insider data theft opportunities via segregation of duties. For example, a policy stating developers should not have permanent or unsupervised access to production environments.

3.2.2 Worker Clearances

Once data classifications are clarified (3.2.1) and a DLP suite is deployed (3.1.1), one encounters a new problem. When sensitive data is found on an insider's desktop, how do we know if it's allowed to be there? This exposes a larger underlying problem, which is most enterprises (outside the military sector) have no authoritative roster of insiders who are authorized to handle data that the general employee population may not.

This problem was solved centuries ago in the military, where a system of clearance levels is used to determine an individual's data access. For example, an officer with Top Secret clearance level can access Top Secret classified data. This simple but effective scheme enables security measures ranging from color-coded badges to sophisticated counter intelligence programs to be implemented.

The military clearance scheme is in effect a simple kind of role based access control (RBAC). A similar scheme could be adopted in a private sector enterprise – hopefully without sacrificing an often free-wheeling (and decidedly un-military) company culture. Such trade-offs between security and culture/morale always deserve careful consideration.

3.2.3 Employee Communication & Training

Every enterprise should periodically (e.g., annually) launch an internal campaign to educate workers and managers about protecting customer privacy and about handling other types of sensitive information. Such training also provides an opportunity to communicate new policies.

3.2.4 HR / Management Procedures & Training

Information security departments can work with human resource departments on procedures to recognize disgruntled employees based on red flag behaviors and common triggers such as negative performance reviews or pending termination. In such cases an organization usually has a legal right to scan their computer, search their belongings, ask questions in an exit interview, etc. Managers and HR specialists need to be trained about signs to watch for and actions to take.

3.2.5 Database Baselines & Best Practices

Every enterprise using information technology should define best practices for database (and application) design to ensure effective means to track authorized user data access, including:

- correct approach to data scrubbing
- correct use of faceless application IDs
- correct implementation of access controls
- more use of database views
- guidelines for encryption

3.2.6 System of Record Audits

Every enterprise using information technology should audit production databases containing sensitive data to verify access controls and certification processes are in place and that access is based on need-to-know. The auditors must be properly trained, and should report organizationally through a different chain of command than the IT staff.

3.3 Areas for Further Research

3.3.1 Honey Token Operations

So-called honey token or honeypot technologies could be adapted to catch insider data thieves in the act of accessing data inappropriately. In this case the honey data looks like valuable confidential information but is booby trapped to trigger a silent alarm if improperly accessed or copied. For example, a document beacon can be attached via a macro to a Microsoft Office document which alerts authorities if the document is opened or copied. A beaconing document containing phony credit card numbers could be placed on a server or sent over the network to see who tries to steal its contents.

Whether operations should be undertaken using such techniques within any given enterprise is a matter for debate. There are obvious ethical, personnel, and legal issues to consider. For now this remains an area of research to determine technical feasibility.

3.3.2 Applicant Screening Systems

One area of proposed research in the banking industry is a national "bad actor database" that could be used to better screen job applicants at US financial institutions. The database could include biometric fingerprint data so applicants can't misidentify themselves using an alias, and so the screening process can be integrated into existing background checks that also use fingerprints. (US banks generally take fingerprints from applicants or new hires as part of federally mandated background checks.) Any institution could start such a database unilaterally, but its full potential can only be realized if it's shared by multiple companies, all contributing data about criminals, fraudsters, fired employees, etc.

3.3.3 Behavior Tracking Systems

While controversial, systems intended to detect anomalous or suspicious employee behavior are an important area of research and development. Some monitor an office worker's actions on his/her computer, catching suspicious actions like large data downloads or unusual Intranet search strings. Some use Bluetooth or RFID to track an employee's physical movements around a building or branch. Some might someday even use lie detector style technology to detect when an employee is feeling stressed or anxious.

Some of this research will bear fruit and some won't. Those that lead to commercial technologies may or may not ever be used at a given enterprise. There are obvious privacy, personnel, ethical, and legal issues to consider.

4 Conclusions

Nothing less than a multi-pronged holistic EDLP program can effectively reduce the risk of a large or embarrassing "data spill" in most modern automated enterprises. There is no one "silver bullet" to prevent data leaks or theft. Only a defense in depth approach can succeed, assuming it addresses every dimension of the problem and every stage of the attack. In this paper we have proposed a number of ways to improve both technical and administrative controls in a typical enterprise, as well as several areas for technical research that could lead to new controls in the future.

Acknowledgments

We're grateful to the organizers of IACS07 [8] for providing a forum where academia, government, and industry can come together to discuss insider threats. In particular the author thanks Columbia University's Steve Bellovin and Sal Stolfo for their encouragement, and Sal Stolfo for sharing his work on document beacons. The author acknowledges his coworkers, especially the invaluable contributions of Michael Foster and Adam De Monaco. We also thank Financial Services Technology Consortium executive director Dan Schutzer for his advice and support.

Any opinions expressed herein are the authors, and not necessarily those of our employer.

References

[1] Randazzo M R, Keeney M, Kowalski E, Cappelli D, Moore A, Insider Threat Study: Illicit Cyber Activity in the Banking and Finance Sector, United States Secret Service, 2004.

[2] Henry T., Controlling Information with Network Content Filtering; Burton Group.

[3] Proctor P, Mogull R.. and Oullet E., Magic Quadrant for Content Monitoring and Filtering and Data Loss Prevention, 2Q07 Gartner Inc.

[4] Penn J. and Raschke T.., The Forrester Wave™: Information Leak Prevention, Q4 2006; Forrester Research.

[5] Yuhanna N. and Julian T., Securing Your Data from Insider Threats June 2007 seminar; Forrester Research.

[6] Cappelli D., D., Moore, A, and Shaw E., A Risk Mitigation Model: Lessons Learned From Actual Insider Sabotage, Carnegie Mellon CERT; *Proceedings CSI 2006*.

[7] Wilson B., Information Security: A Critical Competency, Carnegie Mellon CERT; *Proceedings FSTC 2007.*

[8] 2007 ARO/FSTC Workshop on Insider Attack and Cyber Security (IACS07), http://www.cs.dartmouth.edu/~insider/

[9] *Human Behavior, Insider Threat, and Awareness Project* Institute for Information Infrastructure Protection (I3P), http://www.thei3p.org/projects/insidthoverview.html

[10] FS-ISAC Information Leak Survey, Privately conducted survey, results distributed to members only. (Direct inquiries to the author or the consortium).

[11] FSTC Leak Prevention Survey, Privately conducted survey, results distributed to members only. (Direct inquiries to the author or the consortium).

[12] 2006 E-Crime Watch Survey *CSO Magazine* with the U.S. Secret Service, CERT® Coordination Center and Microsoft Corp.

[13] Externalization of Entitlements, Privately published white paper; direct inquiries to the author.

[14] California law SB1386 serves as the model for other state and federal breach disclosure legislation. http://www.oit.ucsb.edu/committees/itpg/sb1386.asp

[15] Plastic Card Industry Data Security Council, https://www.pcisecuritystandards.org/

[16] Google APIs for search (and other services) documented at http://code.google.com/apis/.

[17] Johnson, M.E. and Dynes, S., Dartmouth College; *Proceedings Workshop on the Economics of Information Security, 2007.*

A Survey of Insider Attack Detection Research

Malek Ben Salem, Shlomo Hershkop, and Salvatore J. Stolfo

Computer Science Department, Columbia University

Abstract This paper surveys proposed solutions for the problem of insider attack detection appearing in the computer security research literature. We distinguish between masqueraders and traitors as two distinct cases of insider attack. After describing the challenges of this problem and highlighting current approaches and techniques pursued by the research community for insider attack detection, we suggest directions for future research.

1 Introduction

Recent news articles have reported that the cell phones of prominent Greek legislators were found to be bugged [30]. Rogue software was injected into the operational systems of the Greek cell phone provider, Vodafone Greece, which controlled a tap for incoming and outgoing calls on selected phones. The phone used by the prime minister and other high ranking officials were apparently targeted. This act was eventually traced to a malicious insider who had hacked the Vodafone system sometime in 2004 and installed the equivalent of a rootkit on an internal Ericsson phone switch. The hack was accidentally discovered through a misconfiguration of a software update a considerable time after the tapping began. The rootkit update accidentally conflicted with other system processes and resulted in alarms being set off in the system. The complexity of the attack could only be attributed to someone with intimate knowledge of the Ericsson switch operating software, which was developed for the last 15 years in Greece.

External threats to the cyber-infrastructure of an organization are constantly evolving. The greatest threat, however, is the problem of insiders who misuse their privileges for malicious purposes. Insider attack has overtaken viruses and worm attacks as the most reported security incident according to a report from the US Computer Security Institute (CSI) [12]. The annual Computer Crime and Security Survey for 2007 surveyed 494 security personnel members from US corporations and government agencies, finding that insider incidents were cited by 59 percent of respondents, while only 52 percent said they had encountered a conventional virus in the previous year.

Much research in computer security has focused on the means of preventing unauthorized and illegitimate access to systems and information. Unfortunately, the most damaging malicious activity is the result of internal misuse within an organization, perhaps since far less attention has been focused inward. Despite classic internal operating system security mechanisms and the literature on formal specification of security and access control policies, including Bell-LaPadula and the Clark-Wilson models [1, 3], we still have an extensive insider attack problem. Indeed in many cases, formal security policies are incomplete and implicit or they are purposely ignored in order to get business goals accomplished. There seems to be little technology available to address the insider threat problem. The state-of-the-art seems to be still driven by forensics analysis after an attack, rather than technologies that prevent, detect, and deter insider attack.

The inside attacker has been defined in many different contexts with no standard definition agreed upon by the research community. How might one then think it is possible to make scientific progress if the problem itself is ill-defined? Nevertheless, there are many well known examples of insider attacks familiar to most people.

For our purposes in this paper, we define a malicious insider to be two classes of malfeasant users; *traitors* and *masqueraders*. A traitor is a legitimate user within an organization who has been granted access to systems and information resources, but whose actions are counter to policy, and whose goal is to negatively affect confidentially, integrity, or availability of some information asset [25, 40]. The traitor uses his/her legitimate credentials when perpetrating their malicious actions, such as in the Greek Vodafone case mentioned above.

The most familiar example of an insider is a masquerader; an attacker who succeeds in stealing a legitimate user's identity and impersonates another user for malicious purposes. Credit card fraudsters are perhaps the best example of masqueraders. Once a bank customer's commercial identity is stolen (e.g. their credit card or account information), a masquerader presents those credentials for the malicious purpose of using the victim's credit line to steal money.

We may distinguish traitors and masqueraders based upon the amount of knowledge each has. A traitor of course has full knowledge of the systems they routinely use and likely the security policies in force. The masquerader may have far less knowledge than the traitor. Furthermore, an insider attack may be due to an innocent mistake by a legitimate user. Hence, insider attack may also be distinguished by intent of the user's actions. Traitors and masqueraders are two sides of what we consider to be the insider threat. The distinction is not entirely satisfactory. After all, a disgruntled insider employee may act as a traitor and a masquerader after stealing the identity of a coworker. But for our present purposes, the distinction is clear enough to consider the general themes of past research in insider attack detection.

An extensive literature exists reporting on approaches that profile user behavior as a means of detecting insider attack, and identity theft in particular. A traitor is presumed to have full knowledge of the internal systems of an organization to which they belong. They use their own credentials and the access granted by those credentials to perform their malicious deeds. A traitor may exhibit normal behavior and still perpetrate malicious acts. Profiling user behavior in this case may seem less relevant except for identifying subtle but significant changes in a user’s normal behavior. A masquerader, on the other hand, has stolen someone’s credentials, and is unlikely to know the behavior of their victim. Thus, even though they control the victim’s credentials that grant access to whatever the victim is authorized to use, the masquerader is likely to perform actions inconsistent with the victim’s typical behavior.

Behavior is not something that can be easily stolen. Stealing someone’s credit card information does not reveal the amount and frequency of what the victim typically buys and from whom. Hence, if one profiles the typical buying patterns of a customer (and keeps this historical information secret) an identity thief, a masquerader, has a relatively low probability of misusing the stolen quarry in a manner consistent with the victim's behavior that will go unnoticed. Fraudulent transactions are thus fairly easy to detect even given proper credentials and credit availability. It is this observation that the credit card companies recognized a couple of decades ago when designing early fraud warning systems, and this idea has largely been the driving theme for much subsequent research on masquerade detection.

On the other hand, a traitor is presumably behaving normally and hence profiling a user to detect significant change as a means of detecting malicious actions may not be the best strategy for detecting this class of insider attack. The intelligence and military communities are challenged with detecting traitors and have devised a host of means of using decoys and trap-based defenses to entice and trick users into revealing their nefarious actions. Far less work has been reported in the computer security literature on developing decoy network defenses beyond early work on honeypots and general ideas on the use of honeytokens of various forms. The detection of traitors is an area ripe with challenges begging for new research.

In the following sections, we provide a general overview of the literature on the insider problem driven primarily by various methods of profiling user actions and the systems they use. Much of the work reports on studies describing various audit sources and algorithms to profile users that are tested on simulated masquerade attack data. Researchers have also distinguished between network-level and host-level detection systems. Most of this work is specific to masquerade attack detection, although some work is reported on trap-based defenses aimed to the traitor detection problem using honeypots and honeytokens. We conclude with a view of what we see as the state-of-the-art of the insider attack detection problem, and we provide recommendations on future research directions.

2 Insider Attacks

In order to understand how to detect malicious insider actions, we have to understand the many forms of attack that have been reported [29]. For example:

- Unauthorized extraction, duplication, or exfiltration of data
- Tampering with data (unauthorized changes of data or records)
- Destruction and deletion of critical assets
- Downloading from unauthorized sources or use of pirated software which might contain backdoors or malicious code
- Eavesdropping and packet sniffing
- Spoofing and impersonating other users
- Social engineering attacks
- Misuse of resources for non-business related or unauthorized activities
- Purposefully installing malicious software

Each of these actions can be considered malicious, but not every one of them may leave an audit trail which can be easily accessed. Several of these actions do leave some trail in some log file which can be linked to the actions of a user after the fact. Hence, when a malfeasance is detected, there is some hope forensics could lead to the perpetrator. Log analysis remains the state-of-the-art in insider attack detection, after a breach has been discovered. Naturally, sophisticated attackers may expend much effort trying to cover their tracks and attacking the logging or auditing sources to remain stealthy. If an organization is not actively monitoring their systems (and users) with sufficient controls preventing tampering with monitor logs, an inside attacker will undoubtedly rarely be detected.

In an insider threat study in the banking and finance sector, Randazzo et al. [31] list the characteristics of insider attacks. Their analysis of validated cases of insider attack indicated that:

- Most incidents required little technical sophistication
- Actions were planned
- Motivation was financial gain
- Acts were committed while on the job
- Incidents were usually detected by non-security personnel
- Incidents were usually detected through manual procedures

These observations should motivate any organization to field monitoring systems to have any hope of automatically and reliably detecting, and deterring, insider attack. We note from this study that most insider attacks on hosts seem to occur at the application level and not at the network-level and hence host-based monitoring is not a desiderata, it is a requirement.

When monitoring systems to mitigate the insider threat one can collect audit data at either host level activity, network level activity, and or a combination of the two. The main consideration is scalability versus coverage. Hosts sensors are hard to deploy, network sensors are relatively easy to install. Many of the insider problems do not even touch the network level. Schultz pointed out that not one approach will work but solutions need to be based on multiple sensors to be able to find any combination of features to detect insiders [30]. Models to detect insider threats will only be as good as the data collected.

3 Detecting Insider Attacks

3.1 Host-based User Profiling

Understanding the intent of some user action is important to mitigate the insider attack problem. Once an attack has taken place, an investigator needs to reconstruct the intent of the attacker from the audit source. This is a slow and manual process which cannot be easily generalized to pre-attack analysis. Rules might be able to be crafted to cover known attacks, but sophisticated attackers will find new ways and new attack methods to fly under the radar. In addition, the task of keeping rules or profiles updated to the latest threat is a significant challenge to using a host-based protection scheme.

One approach reported in the literature is to profile users by the commands they issue (among the first is [7]). In the general case of computer user profiling, the entire audit source can include information from a variety of sources:

- Command line calls issued by users
- System call monitoring for unusual application use/events
- Database/file access monitoring
- Organization policy management rules and compliance logs

The type of analysis used is primarily the modeling of statistical features, such as the frequency of events, the duration of events, the co-occurrence of multiple events combined through logical operators, and the sequence or transition of events. However, most of this work failed to reveal or clarify the user's intent when issuing commands. The focus is primarily on accurately detecting change or unusual command sequences. We begin with a survey of a collection of papers whose primary focus is command sequence analysis.

3.1.1 Modeling Unix Shell Commands

A hybrid high-order Markov chain model was introduced by Ju and Vardi [15]. A Markov chain is a discrete-time stochastic process. The goal of the work is to identify a "signature behavior" for a particular user based on the command sequences that the user executed. In order to overcome the high-dimensionality, inherent in high-order Markov chains, a "mixture transition distribution" (MTD) approach is used to model the transition probabilities. When the test data contains many commands unobserved in the training data, a Markov model is not usable. Here, a simple independence model with probabilities estimated from a contingency table of users versus commands may be more appropriate. The authors used a method that automatically toggled between a Markov model and an independence model generated from a multinomial random distribution as needed, depending on whether the test data were "usual" (i.e. the commands have been previously seen), or "unusual" (i.e. Never-Before-Seen Commands or NBSCs).

Schonlau *et al.* applied six masquerade detection methods to a data set of "truncated" UNIX shell commands for 70 users [7] collected using the UNIX *acct* auditing mechanism. Each user had 15,000 commands collected over a period of time ranging between a few days and several months. 50 users were randomly chosen to serve as intrusion targets. The other 20 users were used as simulated masqueraders. The first 5000 commands for each of the 50 users were left intact or "clean", the next 10,000 commands were randomly injected with 100-command blocks issued by the 20 masquerade users. When commands are grouped into blocks of 100 commands each, the block is either "clean", or "dirty", that is all 100 commands were originated by a masquerader. The complete data set and more information about it can be found at http://www.schonlau.net. The objective was to accurately detect the "dirty" blocks and classify them as masquerader blocks. This data set was widely used by several authors that investigated different detection methods and has served as the standard benchmark dataset for this line of research.

One detection method explored by Schonlau, called "uniqueness" relies on the fact that half of the commands in the training data are unique (i.e used by one user only), and many more are unpopular amongst the users ,(i.e used only by a few users). The second method investigated was the Bayes one-step Markov approach. It was based on one step transitions from one command to the next. The approach, due to DuMouchel [8], uses a Bayes factor statistic to test the null hypothesis that the observed one-step command transition probabilities were consistent with the historical transition matrix. The two hypotheses modeled were the null hypothesis, which assumed that the observed transitions probabilities stem from the historical transition matrix, and the alternative hypothesis which assumed that they were generated from a Dirichlet distribution.

A hybrid multi-step Markov method similar to the one introduced by Ju and Vardi [17] is also used. The fourth method used, called the compression method, was based on the premise that test data appended to historical training data compressed more readily when the test data stemmed from the very same user rather than from a masquerader, and was implemented through the UNIX tool "compress" which implements a modified Lempel-Ziv algorithm.

IPAM (Incremental Probabilistic Action Modeling), another method applied on the same dataset, and introduced by Davidson and Hirsch [7] was also based on one-step command transition probabilities estimated from the training data. The probabilities were continuously updated following an exponential decay scheme with the arrival of a new command.

The sequence-match approach was presented by Lane and Brodley [18]. For each new command, a similarity measure is computed between the 10 most recent commands and a user's historical profile. A user's profile consisted of command sequences of length 10 that the user had previously used. The similarity measure was a count of the number of matches in a command-by-command comparison of 2 command sequences with a greater weight assigned to adjacent matches. This similarity measure was computed for the test data sequence paired with each command sequence in the profile.

Maxion and Townsend applied a naïve Bayes classifier, which had been widely used in text classification tasks, to the same data set [22]. Maxion provided a thorough and detailed investigation of classification errors of the classifier in a separate paper [24], highlighting why some masquerade victims were more vulnerable than others, and why some masqueraders were more successful than others. Killourhy and Maxion also investigated a shortcoming of the naïve Bayes classifier when dealing with NBSCs [16].

The semi-global alignment method presented by Coull et al. [5] is a modification of the Smith-Waterman local alignment algorithm. It uses a scoring system that rewards the alignment of commands in a test segment, but does not necessarily penalize the misalignment of large portions of the signature of the user.

Another approach called a self-consistent naïve Bayes classifier is proposed by Yung [43] and applied on the same data set. This method was a combination of the naïve Bayes classifier and the EM-algorithm. The self-consistent naïve Bayes classifier is not forced to make a binary decision for each new block of commands, i.e. a decision whether the block is a masquerader block or not. Rather, it assigns a score that indicates the probability that the block is a masquerader block. Moreover, this classifier can change scores of earlier blocks as well as later blocks of commands.

Oka et al. had the intuition that the dynamic behavior of a user appearing in a sequence could be captured by correlating not only connected events, but also events that were not adjacent to each other, while appearing within a certain distance (non-connected events). With that intuition they developed the layered networks approach based on the Eigen Co-occurrence Matrix (ECM) [27, 28]. The ECM method extracts the causal relationships embedded in sequences of commands, where a co-occurrence means the relationship between every two commands within an interval of sequences of data. This type of relationship cannot be represented by frequency histograms nor through n-grams.

Table 1 presents the estimated accuracy of the classification methods which are all based on a two-class supervised training methodology whereby data is labeled as self or non-self. The Schonlau data used is a mixture of command sequences from different users. The classifiers produced in these studies essentially identify a specific user from a set of known users who provided training data. Furthermore, mixing data from multiple users to train classifiers to detect masqueraders is complicated and fraught with problems. Besides potential privacy threats, requiring the mixture of data from multiple users requires substantial retraining of classifiers as users join and leave an organization.

Method	False Alarms (%)	Missing Alarms (%)
Uniqueness	1.4	60.6
Bayes one-step Markov	6.7	30.7
Hybrid multi-step Markov	3.2	50.7
Compression	5.0	65.8
Sequence Match	3.7	63.2
IPAM	2.7	58.9
Naïve Bayes (Updating)	1.3	38.5
Naïve Bayes (No Updating)	4.6	33.8
Semi-Global Alignment	7.7	24.2
Eigen Co-occurrence Matrix	3.0	28.0
Naïve Bayes + EM	1.3	25.0

Table 1. Summary of accuracy performance of Two-Class Based AnomalyDetectors Using the Schonlau Data Set

In a real-word setting it is probably more appropriate to use a one-class, anomaly detection-based training approach. Wang and Stolfo experimented with one-class training methods in [24] using a naïve Bayes classifier and a Support Vector Machine (SVM) model of user commands to detect masqueraders. The authors have also investigated SVMs using binary features and frequency-based features. The one-class SVM algorithm using binary features was the best performing classifier among four one-class training algorithms that were analyzed. It also performed better than most of the two-class algorithms listed above, except the two-class multinomial Naïve Bayes algorithm with updating. In summary, Wang and Stolfo's experiment confirmed that, for masquerade detection, one-class training is as effective as two-class training.

Szymanski and Zhang [38] proposed recursively mining the sequence of commands by finding frequent patterns, encoding them with unique symbols, and rewriting the sequence using this new coding. A signature was then generated for each user using the first 5000 commands. The process stopped when no new dominant patterns in the transformed input could be discovered. They used a one-class SVM classifier for masquerade detection. Although they presented a weighting prediction scheme for author identification, we will limit our focus here to the masquerade detection application of their approach. The authors used an individual intrusion detection approach with 4 features (the number of dominant patterns in levels 1 and 2, and the number of distinct dominant patterns in levels 1 and 2), as well as a "communal" intrusion detection approach, where they added new features, such as the number of users sharing each dominant pattern in a block. Again, such an approach demands mixing user data and may not be ideal or easily implemented in a real-world setting.

Dash et al [6] created user profiles from groups of commands called sequences. 13 temporal features are used to check the consistency of patterns of commands within a given temporal sequence. Probabilities are calculated for movements of commands within a sequence in a predefined reordering between commands. They achieve high accuracy but also high false positive rates on their experiments.

Seo and Cha [33] experimented with combinations of SVM kernels with some success. They managed to increase the accuracy at the expense of somewhat higher false positives.

Tan and Maxion investigated which detector window size would enable the best detection results [39]. They uncovered that the best detector window size was dependent on the size of the minimal foreign sequence in test data, which is not determinable *a priori.* A foreign sequence is one that is not contained in the alphabet set of the training data, but each of its individual symbols is, whereas a minimal foreign sequence is a foreign sequence that contains within it no smaller foreign sequences.

It has been shown that the Schonlau data set was not appropriate for the masquerade detection task. Maxion lists several reasons [24]. First, the data was gathered over varied periods for different users (from several days to several months), and the number of login sessions varied by user. Second, the source of data is not clear. We do not know whether the users perform the same jobs or are widely spread across different job functions. Moreover, in **acct**, the audit mechanism used to collect the data, commands are not logged in the order in which they are typed, but rather when the application ends. Hence the methods applied that focus on strict sequence analysis may be faulty.

In order to alleviate some of the problems encountered with the Schonlau data set, Maxion applied naïve Bayes classifier to the Greenberg data set, a user command data set enriched with flags and arguments in [23]. He compared the performance of the classifier on the Greenberg data set by using enriched commands and truncated commands. The hit rate achieved using the enriched command data was more than 15% higher than with the truncated data. However, the false positives rate was approximately 21% higher as well. Nevertheless, when plotting the ROC curves for both data sets, the one for enriched data runs above the ROC curve for truncated data, showing that a better detection performance can be achieved using the user commands enriched with flags and arguments.

As noted, several types of attributes and statistical features can be used for modeling a user's actions. Ye et al. studied the attributes of data for intrusion detection [42]. The attributes studied included the occurrence of individual events (audit events, system calls, user commands), the frequency of individual events (e.g. number of consecutive password failures), the duration of individual events (CPU time of a command, duration of a connection), and combinations of events, as well as the frequency histograms or distributions of multiple events, and the sequence or transition of events. The goal was to find out whether the frequency property was sufficient for masquerader detection, and if so whether there was a single event at a given time sufficient for detecting a masquerader. Five probabilistic techniques were investigated on system call data: a decision tree, Hotelling's T^2 test, the chi-square test, the multivariate test, and the Markov chain. The data set used was made up of 250 auditable security-relevant events collected by the

Solaris Basic Security Module (BSM) and 15 simulated intrusions on the background of normal activities. The investigation confirmed the importance of both the frequency property, and the ordering property of events.

3.1.2 User Profiling in Windows Environments

Less research work has been applied to Windows environments compared to work directed for the Unix environment. Much of the difference lies in the auditing methods available on each platform. Linux apparently has cleaner auditing mechanisms (acct, BSM, etc.) whereas Windows has a plethora of system actions that can be captured by various monitoring subsystems.

Shavlik et al. presented a prototype anomaly detection system that creates statistical profiles of users running Windows 2000 [34]. Their algorithm measures more than two-hundred Windows 2000 properties every second, and creates about 1500 features from the measurements. The system assigns weights to the 1500 features in order to accurately characterize the particular behavior of each user – each user thus is assigned his or her own set of feature weights as their unique signature. Following training, each second all of the features "vote" as to whether or not it seems likely that an intrusion has occurred. The weighted votes "for" and "against" an intrusion are compared, and if there is enough evidence, an alarm is raised.

Nguyen, Reiher & Kuenning propose detecting insider threats by monitoring system call activity [26]. Instead of building profiles on system call traces, they analyze relationships between users and files, users and processes, and processes and files. They build user-oriented models as well as process-oriented models using file system and process-related system calls exploiting the regularity in the patterns of file accesses and process-calling by programs and users. They focus on building a Buffer-overflow Detection System (BDS), which is able to detect buffer overflows in many cases, but only if they occur in a set of programs that have a fixed list of children, i.e. only 92% of programs. The authors' approach, as they point out, was not suitable for detecting malicious insider activity on laptops, because the traces collected on laptops are very dynamic and users do not have a fixed pattern of working time which could be used to define an adequate time window for analysis.

Jha et al. present a statistical anomaly detection algorithm that has the potential of handling mixtures of traces from several users (this will occur when several users are colluding) by using mixtures of Markov chains. The technique which has an unobserved or hidden component can be compared to Hidden Markov Models (HMMs). The training algorithm for HMMs runs in time $O(n*m^2)$), where n is the number of states in the HMM and m is the size of the trace, whereas, the training time for Markov chains is O(m). So the authors' approach was less computationally-expensive than HMMs.

Li and Manikopoulos [20] explored modeling user profiles with SVMs using a audit data from a Windows environment gathered over a year. They model the sequence of windows and processes over time in a manner similar to what a process sensor would see. They simulate attack data by mixing data between legitimate user sessions. They reported some success at modeling the user profiles, but suffer with high false positive rates.

In most of the approaches surveyed above, either user command data or system calls data were used. User command data fail to capture window behavior and do not include commands executed inside a script, whereas system call data are not particularly human-readable, nor easily attributed to direct user action. On the other hand, process table data includes window behavior and anything running in a script, and can be easily interpreted when read by a human. Moreover, window tracing provides information at a level of granularity somewhere between the levels of a command line and a system call, while most of the system noise can be filtered out (a formidable challenge when tracing Windows), which makes it a good candidate for user profiling.

Goldring collected user data consisting of successive window titles with process information (from the process table) for a group of users over 2 years [11]. The combination of data sources allowed use of the process tree structure to filter out system noise. However it complicated the feature selection task. The system reduces the stream of data to a single feature vector that consists of a mixture of different feature types per session. A record is generated each time a new window is opened including information about the window title, and all contents in a window title's bar (a wealth of new information, e.g. subject lines of emails, names of web pages, files and directories). Besides that, the window's process and parent process ID's are saved. The window titles' data allows one to distinguish between the operating system's programs such as Control Panel and find Files, which would not be distinguishable from inspecting the process table alone. Goldring reported no performance results, but rather presented a proof-of-concept system. Even if detailed accuracy results were reported, the datasets used bear little resemblance to other data used by researchers. This highlights another important methodological weakness of this line of research where a paucity of data makes it difficult to know whether advances have been made.

3.1.3 User Profiling in Web Environments

There is a vast literature on data mining methods applied to web user "click" data for marketing analytics that goes well beyond the scope of this paper. However, some work has been done focusing on web profiling for security problems. Kim, Cho, Seo, Lee, and Cha studied the problem of masquerade detection in a web environment. They focused on "anomalous web requests generated by insiders who attempted to violate existing security policies given by the specific organization" [17]. They applied SVMs to web server logs and used two different kernels: TinySVM (an implementation of SVM for pattern recognition) and the Radial Basis Function (RBF) kernel. Only simple features were used, i.e. neither session features, nor temporal features were included. Simple features are those related to a single web sever request such as the IP address, the hour of the day, the HTTP method (get, post, put, delete, options, head, and trace), the requested page ID, the request status code, the number of transferred bytes, etc. The results showed that SVMs achieved near-perfect classification rates using simple features only. However, the method used did not handle concept drift well, and failed to generalize the model for two users due to changes in user behavior.

3.1.4 Program Profiling Approaches

Besides user issued commands, inside attackers may inject programs or infect host systems causing changes in underlying system configurations and program behaviors. Hence, approaches to profiling environments and program executions may have relevance to the insider attack detection problem. Much work in this area is devoted to detection of code injection attacks, too broad a topic to describe here. A few characteristic works are described in the following.

Forrest et al. proposed a real-time on-line anomaly detection system [8] that mimicked the mechanisms used by the natural immune systems. This is done by monitoring system calls of running privileged processes (profiles were built using normal runs of such programs). The modeling is limited to privileged root processes since they have more access to computer resources than user processes, and they have a limited range of behavior that is quite stable and predictable. A separate database of normal behavior is built for each privileged process. The database was specific to a particular architecture, software version and configuration, local administrative policies, and usage patterns, providing a unique definition of "self".

The underlying assumptions are that the sequences of system calls executed by a program are locally consistent during normal operation, and that if a security hole in a program is exploited, then abnormal sequences of system calls will occur. A number of experiments were performed using the normal traces of the *sendmail* and *lpr* processes as examples. The results obtained showed that the behavior of different processes was easily distinguishable using the sequence information alone for these two system programs. Several attacks on the *sendmail* process were tested, such as the sunsendmailcp script, the syslog attack, the lprcp attack script, the decode attack, and the lpr attack. Other sources of anomalous behavior tested included unsuccessful intrusion attempts, such as remote attack scripts, called sm565a and sm5x, and error conditions. The results have shown that short sequences of system calls could indeed define a unique and stable signature, which allows for the detection of common sources of anomalous behavior.

The method proposed is computationally efficient and has very low storage requirements. Many aspects of process behavior are ignored (e.g. parameter values passed to system calls, timing information, and instruction sequences between system calls). Although the approach could enable the detection of several scenarios, such as when a program moves to an unusual error state during an attempted break-in, when an intruder replaces code inside a running program, and when new processes are forked., it would not detect race conditions or masqueraders using another user's account. This work led to a number of derivative ideas explored by the computer security community.

Stolfo et al. [37] present the modeling of accesses to the Windows Registry by exploiting regularity in process accesses to the Windows registry. Malicious code often misuses Registry keys in various ways that are detectable as anomalous queries. They introduced a general purpose anomaly detection algorithm, the Probabilistic Anomaly Detection (PAD) algorithm, that assumes anomalies are statistical outliers and hence are a minority of the training data. PAD was applied to model Registry queries and was compared with the One-Class Support Vector Machine (OCSVM) algorithm using several different kernels. PAD showed better performance, both in accuracy, and in computational complexity, achieving a 100% detection rate of anomalies with a 5% false positives rate for the particular test sets available for the study.

3.2 Network-Based Sensors

3.2.1 Network Observable User Actions

ARDA (recently renamed IARPA) sponsored a Cyber Indications and Warning workshop dealing with the insider threat. One of the lessons learned was that in many cases insider threats have authorization to access information but may access information they do not have a "need to know". When an insider accesses information that they do not need to know, one may have good evidence of an insider attack. A system for detecting insiders who violate need-to-know, called ELICIT, was developed by Maloof and Stephens [21]. The focus of their work was on detecting activities, such as searching, browsing, downloading, and printing, by monitoring the use of sensitive search terms, printing to a non-local printer, anomalous browsing activity, and retrieving documents outside of one's social network. Five malicious insider scenarios were tested, that represented need-to-know violations. Contextual information about the user identity, past activity, and the activity of peers in the organization or in a social network were incorporated when building the models. HTTP, SMB, SMTP, and FTP traffic was collected from within a corporate intranet network for over 13 months, but no inbound or outbound traffic was gathered. In order to identify the information deemed outside the scope of an insider's duties, a social network was computed for each insider based on the people in their department, whom they e-mailed, and with whom they worked on projects. A Bayesian network for ranking the insider threats was developed using 76 detectors. Subject-matter experts defined the thresholds for these detectors, at which an alarm is set. A single threat score is computed for each user based on the alerts from these detectors.

Identifying specific users from observable network events consumed considerable effort. Event attribution proved to be a major challenge: 83% of events initially had no attribution, and 28.6% of them remained un-attributed, even after the use of two off-line methods to determine the originator of a particular event. The evaluation of the system used scenarios that were executed over a short period of time, less than one day. However, attacks by insiders who violate need-to-know usually occur over days, months, and even decades, such as in the case of Robert Hanssen. Therefore, it is important to evaluate the ELICIT system using other scenarios that occur over longer periods of time. In any event, although interesting, the focus of this system is limited to environments and organizations that have a formal policy restricting access to information on a need-to-know-basis. It is rare that such controls are easily discernible in most organizations.

3.2.2 Honeypots

Honeypots are information system resources designed to attract malicious users. Honeypots have been widely deployed in De-Militarized Zones (DMZ) to trap attempts by external attackers to penetrate an organization's network. Their typical use is for early warning and slowing down or stopping automated attacks from external sources, and for capturing new exploits and gathering information on new threats emerging from outside the organization. These trap-based defenses are also useful for the insider threat.

Spitzner presented several ways to adapt the use of honeypots to the insider attack detection problem [36]. Since insiders probably know what information they are after, and in many cases, where that information is to be found, and possibly how to access it, he recommends implanting honeytokens with perceived value in the network or in the intranet search engine. A honeytoken is "information that the user is not authorized to have or information that is inappropriate" [36]. This information can then direct the insider to the more advanced honeypot that can be used to discern whether the insider intention was malicious or not, a decision that may be determined by inspecting the insider's interaction with the honeypot. In order to reach such interaction that will be used to gather information, it is important to ensure that the honeypot looks realistic to the insider. Humans have a keen sense of suspicion, and hence the grand challenge for honeypots or any trap-based defense is believability, while preventing poisoning of operational systems.

Honeypots suffer from some shortcomings. First, the inside attacker may not ever use or interact with the honeypot or honeytoken, especially if their identity is known or discovered by the insider. Moreover, if an attacker discovers a honeypot, he/she can possibly inject bogus or false information to complicate detection.

3.3 Integrated Approaches

Among the first integrated systems devised for the malicious insider detection problem was the one presented by Maybury et al. in [25]. The integrated system used honeypots, network-level sensors for traffic profiling to monitor scanning, downloads, and inside connections, and Structured Analysis, a real-time and top-down structural analysis that uses the models of insiders and pre-attack indicators to infer the malicious intent of an insider. Moreover, several data sources were used in addition to auditing of cyber assets. Physical security logs, such as employee badge readers, were also integrated to keep track of the location of a user. The program funding this effort apparently ended prematurely. Insufficient test and evaluations were performed on an approach that seemed quite promising.

3.4 Summary

By way of summary, the papers surveyed report the use of heterogeneous audit sources. Most user profiling techniques designed for use in the Unix or Linux environment used the Schonlau data set, a data set made up of truncated sequences of user commands. We have surveyed all two-class based methods and the few one-class based methods applied to this data set. Other approaches using other data sets, such as the Greenberg data set that includes command flags and arguments, were presented. User commands in Unix and Linux environments are easily captured in and are directly observable user actions. The Schonlau datasets serve as a general benchmark dataset and hence most of the literature has been focused on masquerade detection using Unix commands.

In the Windows operating system environment, a variety of audit sources can be exploited. The range of data available inclues system calls, registry accesses [37] which occur when users execute applications, and a combination of process and windows data (window title, how long a window has been open, etc.).

On the network level, the observables are more distant from a distinct user. Attributing of a network level event to a distinct user is a hard. Detecting masqueraders from network level data alone remains a challenge. However network level events are valuable in detecting malicious or unusual activities such as massive downloading of information that the insider does not have a need to know, or the dissemination of information outside the organization's network.

In the reports appearing in the research literature it appears that the data used for training is real data acquired from real sources. However, for testing of proposed detection methods, most authors had to resort to simulated attacks. For instance, Maloof and Stephens asked a red team to perform some attacks based on pre-defined scenarios, and Schonlau used normal user data injected into a different user’s data set to serve as a simulated masquerade data. That is hardly a real masquerade attack.

The approaches used also depend on the type of insider problem tackled. For masquerade detection the approach of choice was host-based user profiling, whereas for traitor detection other approaches, such as host-based program profiling using systems calls or registry access data, were used to detect the malicious activity on a system. Network-level sensors were used for traitor detection by Maybury et al. and by Maloof and Stephens, whose approach seems promising for the detection of need-to-know violations. There have been a limited number of reports on trap-based, or honeypot-based, detection methods for the insider problem.

Of particular note is the difficulty of comparatively evaluating competing methods and approaches. This is primarily due to the lack of a uniform test data with known ground truth. Although, the Schonlau data set has been widely used by many authors, it has been shown that it is far from being suitable for an objective evaluation of the insider attack detection algorithms.

Table 2 represents a general summary of specific audit sources used by researchers to detect masqueraders or traitors gleaned from the surveyed research papers. Each cell of the table represents our opinion about how well a specific approach may be suitable as an audit source to detect masqueraders or traitors, expressed on a simple scale from Low to High. For example, researchers conjecture that a masquerader is more likely to trigger anomaly behavior models by executing commands that are unusual for the victim whose credentials they have stolen. Consequently, it is assumed that user command auditing has a high chance of successfully detecting masqueraders. That assumption has driven a considerable amount of research activity as described in section 3. Network-level audit sources are assumed helpful in detecting violations of “need to know” policies, such as exfiltration of data and hence have a high chance of successfully identifying traitors. Honeypots and related decoy technologies are proposed as suitable technologies for traitor detection, as well as masquerader detection. Alternatively, it is unclear how well an insider attack may be detected from Unix System Call anomalies, and hence we rate the utility of this audit source as low. We are unaware of any formal study of each audit source validating or refuting these assumptions. This table may serve as a guide for future research in monitoring technologies for insider attack detection.

	Masquerader	Internal Traitor
Two-Class Classifiers: Unix Command Sequences	High – Unfamiliar with local environment and user behavior	Medium – Can possibly mimic another normal user or train the classifier
One-Class: Unix Command sequences	High – Unfamiliar with local environment and user behavior	Medium – Can possibly mimic another normal user or train the classifier
Unix Audit Events	Medium – Given proper credentials and might not trigger alerts	Low – Application level malicious acts may not manifest as unusual events
Unix System Calls	Medium – Might not violate system call profile	Low – Application level malicious acts may not manifest as unusual events
Window Usage Events	Medium – Given proper credentials and might not trigger alerts	Low – Application level malicious acts may not manifest as unusual events
Windows Registry access	Medium – unless malicious programs access Registry	Medium – unless malicious programs access Registry
Network Activity Audit	Medium – If attack uses network and attribution is possible	High – If attack uses network and attribution is possible
Honeypots and Decoy Technologies	High – Unfamiliar with local information and likely to interact with honeypot	Medium – Unlikely to interact if aware of the location of honeypots

Table 2. Summary of Insider Approaches and Suitability of Audit Mechanisms.

4 Future Research Directions

User profiling as a means of identifying abnormal user behavior is well established as a primary methodology for masquerader attack detection. As we have noted, a masquerader impersonates another persona and it is unlikely the victim's behavior will be easily mimicked. Hence, abnormal behavior is a good indicator of a potential masquerade attack as a consequence of identity theft. User profiling may also be useful in detecting a traitor, if subtle but significant changes in a user's behavior indicate a malicious activity. We believe that it will be important to derive user *profile models that reveal user intent* in order to hone in on insider actions that are suspicious and likely malicious. It may not be enough to know of a malicious act merely from knowing that a user has issued an abnormal command sequence unless that sequence could violate a security policy. For example, we conjecture that modeling a user's search behavior may be one way of capturing a user's intent to seek information for malicious purposes, something that a masquerader, and possibly a traitor, is likely to do early in their attack behavior. Too much searching, or searching in abnormal directories or locations, seems more than odd, it may seem sinister in intent.

A major challenge of insider attack detection research is the lack of *real data* in order to study and measure general solutions and models. It is hard, if not impossible, to collect data from normal users in many different environments. It is especially hard to acquire real data from a masquerader or traitor while performing their malicious actions. It is hard to obtain real intrusions for ground truth test and evaluation for a number of reasons:

- Researchers generally do not have direct access to real attacks
- Attacks may be undetected and thus unavailable for study
- Organizations do not admit that they were attacked and hence shy away from cooperating with researchers
- Attacks might be mistaken for incompetence

Even if such data were available, it is more likely to be out of reach and controlled under the rules of evidence, rather than being a source of valuable information for research purposes. Because of the scarcity of real data, Chinchani et al. created RACOON [2], a system for generating user command data for anomaly detection from customizable templates representing particular user profiles. However, the system is likely to suffer from the same shortcomings of most simulated data. Even though noise is introduced into the simulated data, that noise still followed a predictable distribution and is unlikely to follow a real empirical distribution from a particular real world setting.

Given these challenges, devising *capture the flag* exercises to generate insider attack datasets that are realistic in nature may provide a means of advancing the state-of-the-art in understanding and solving the insider threat.

It is generally unknown what types of *audit sources are most discriminatory* to reliably detect insider malicious behavior. Moreover, it is not obvious what amount of data is needed for modeling, nor how long the training or data collection period should be.

We posit that malicious insider actions on computer systems are likely to occur at the *application level*. For instance, a customer service employee in a call center may access more customer records on one particular day than he/she typically accesses on other days, possibly to commit a crime to sell confidential information. Detecting such unusual events can only occur at the business application level, and application-level knowledge is needed to understand the user's intent and confirm whether the intent of user actions is possibly malicious. This may be detectable using host-based sensors and audit sources, and possibly through network-based sensors if the application is accessed remotely and the content flow on the network were exposed for analysis.

The most vexing problem for researchers is to devise detection methods that accurately distinguish between the cases where an insider attack is verified with high confidence versus cases where an insider attack is inferred from partial knowledge of possibly suspicious actions. Distinguishing false positives from true positives in the presence of uncertainty is particularly challenging when people's reputations are at stake. Hence, we also believe that any technologies developed to detect insider attack have to include strong *privacy-preserving guarantees* to avoid making false claims that could harm the reputation of individuals whenever errors occur.

Another important topic for research is the investigation of *alternative mitigation strategies*. For instance, how does a monitoring or detection system challenge a user when the system detects what it believes are malicious activities? How might a system alert a supervisor of a possible attack without disclosing an employee's true identity unless and until an attack has been validated?

Beyond the significant challenges in computing accurate user profiles, considerable effort is needed on developing techniques for *trapping traitor behaviors*. We believe a major challenge will be to develop and inject bogus data and information that is believable to sophisticated humans with full knowledge of an organization's internal systems without negatively impacting operations. How does one develop a trap for those who are aware that such technology is in use and do so without poisoning the legitimate operations of the organization's systems and functions?

5 Conclusion

Insider threat detection is a nascent research field ripe with opportunities for new approaches and new research methodologies. A plethora of machine learning and modeling algorithms are available as well as a wealth of audit sources that can be acquired effectively. However, building effective and highly accurate automated monitoring and analysis systems for detecting insider attacks remains an open challenge.

The lack of ground truth data limits the potential value of various proposed solutions since the accuracy of any proposed method is hard to measure and validate. Even so, much work has been published using "simulated" masquerade attack data. We surveyed the different machine learning and modeling algorithms applied to masquerader attack detection using host-based and network-based audit sources. There has been a modest amount of work in the area. However, the best audit sources and most discriminating features one might use in automated systems to detect masquerader are still unknown. The experimental methodology has been generally weak sue to the lack of suitable realistic data. Although, there have been many methods proposed, their utility is uncertain, and none of them is clearly superior to all others. Although one dataset, the Schonlau dataset, has been useful for a community of researchers to use in comparative evaluations, that dataset itself is insufficient to conduct realistic evaluations. The data set is limited in scope of information it provides, and does not contain true insider attack command sequences. At best, the dataset may be useful to compare computational performance between competing algorithms, but accuracy is not measurable in a meaningful way.

A number of other approaches have been studied only partially and remain the subject of considerable future research. Trap-based technologies and use of decoys and honeypots of various types have only been partially explored, and offer numerous challenges to be effective methods of detecting sophisticated human insider attacks.

By way of summary, new methods of detecting insider attack, whether by traitor or masquerader, remains an open and active area of research, and we expect it to be so for some time to come.

References

[1] Bell D E, LaPadula L J, Secure Computer Systems: Mathematical Foundations. MITRE Corporation, 1973.

[2] Chinchani R, Muthukrishnan A, Chandrasekaran M, Upadhyaya S, RACOON: Rapidly Generating User Command Data for Anomaly Detection from Customizable Templates. Computer Security Applications Conference, 2004. 20th Annual Volume, Issue, 6-10 Dec, 2004.

[3] Clark D, Wilson D R, A Comparison of Commercial and Military Computer Security Policies. IEEE Symposium on Security and Privacy, 1987.

[4] Costa P C G, Laskey K B, Revankar M, Mirza S, Alghamdi G, Barbara D, Shackelford T, Wright E J, DTB Project: A Behavioral Model for Detecting insider Threats. International Conference on Intelligence Analysis. McLean, VA, 2005.

[5] Coull S, Branch J, Szymanski B, Breimer E, Intrusion Detection: A Bioinformatics Approach. Proceedings of the 19th Annual Computer Security Applications Conference, 2003.

[6] Dash S K, Rawat S, Vijaya Kumari G, Pujari A K, Masquarade Detection Using IA Network. Computer Security Applications Conference, 2005.

[7] Davison B D, Hirsh H, Predicting Sequences of User Actions. AAAI-98/ICML-98 Workshop :5-12, 1998.

[8] DuMouchel W, Computer Intrusion Detection Based on Bayes Factors for Comparing Command Transition Probabilities. Technical Report TR91: National Institute of Statistical Sciences, 1999.

[9] Forrest S, Hofmeyer S A, Somayaji A, Longstaff T A, A Sense of Self for Unix Processes. IEEE Symposium on Research in Security and Privacy :120-128, 1996.

[10] Ghosh A K, Schwartzbard A, Schatz M, Learning Program Behavior Profiles for Intrusion Detection. USENIX Workshop on Intrusion Detection and Network Monitoring, 1999.

[11] Goldring T, User Profiling for Intrusion Detection in Windows NT. 35th Symposium on the Interface, 2003.

[12] Gordon L A, Loeb M P, Lucyshyn W, Richardson R, CSI/FBI Computer Crime and Security Survey, 2006.

[13] Jha S, Kruger L, Kurtz T, Lee Y, Smith A, A Filtering Approach To Anomaly and Masquerade Detection, 2004. http://www.people.fas.harvard.edu/~lee48/research/IDS.pdf

[14] Jones A K, Sielken R S, Computer System Intrusion Detection: A Survey, University of Virginia, Computer Science Technical Report, 2000.

[15] Ju W-H, Vardi Y, A Hybrid High-Order Markov Chain Model For Computer Intrusion Detection, Technical Report Number 92, National Institute of Statistical Sciences, 1999.

[16] Killourhy K, Maxion R, Investigating a Possible Flaw in a Masquerade Detection System, Technical Reports of the University Newcastle University, Number 869, 2004.

[17] Kim H S, Cho S, Lee Y, Cha S, Use of Support Vector Machine (SVM) In Detecting Anomalous Web Usage Patterns, Symposium on Information and Communications Technology, 2004.

[18] Lane T, Brodley C, Sequence Matching and Learning in Anomaly Detection for Computer Security. AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management :43-49, 1997

[19] Laskey K, Alghamdi G, Wang X, Barabara D, Shackelford T, Wright E, Fitgerald J, Detecting Threatening Behavior Using Bayesian Networks, Proceedings of the Conference on Behavioral Representation in Modeling and Simulation, 2004.

[20] Li L, Manikopoulos C N, Windows NT one-class masquerade detection. Information Assurance Workshop, Proceedings from the Fifth Annual IEEE SMC :82-87, 2004.

[21] Maloof M, Stephens G D, ELICIT: A System for Detecting Insiders Who Violate Need-to-know. Recent Advances in Intrusion Detection (RAID), 2007.

[22] Maxion R A, Townsend T N, Masquerade Detection Using Truncated Command Lines. International Conference on Dependable Systems and Networks :219-228, 2002.

[23] Maxion R A, Masquerade Detection Using Enriched Command Lines. International Conference on Dependable Systems & Networks, 2003.

[24] Maxion R A, Townsend T N, Masquerade Detection Augmented with Error Analysis. IEEE Transactions on Reliability 53, 2004.

[25] Maybury M, Chase P, Cheikes B, Brackney D, Matzner S, Hetheringston T, Wood, B, Sibley C, Martin J, Longstaff T, Spitzner L, Haile J, Copeland J, Lewandowski S, Analysis and Detection of Malicious Insiders, International Conference on Intelligence Analysis, 2005.

[26] Nguyen N T, Reiher P L, Kuenning G, Detecting Insider Threats by Monitoring System Call Activity. IEEE Workshop on Information Assurance :45-52, 2003.

[27] Oka M, Oyama Y, Kato K, Eigen Co-occurrence Matrix Method for Masquerade Detection, 2004 http://spa.jssst.or.jp/2004/pub/papers/04016.pdf.

[28] Oka M, Oyama Y, Abe H, Kato K, Anomaly Detection Using Layered Networks Based on Eigen Co-occurrence Matrix, RAID 2004, 223-237.

[29] Phyo A H, Furnell S M, A Detection-Oriented Classification of Insider IT Misuse. Proceedings of the 3rd Security Conference, 2004.

[30] Prevelakis V, Spinellis D, The Athens Affair. IEEE Spectrum, 44:7:26-33, 2007.

[31] Randazzo M R, Keeney M, Kowalski E, Cappelli D, Moore A, Insider Threat Study: Illicit Cyber Activity in the Banking and Finance Sector, 2004.

[32] Schonlau M, DuMouchel W, Ju W-H, Karr A F, Theus M, Vardi Y, Computer Intrusion: Detecting Masquerades. Statistical Science 16:1:58-74, 2001.

[33] Seo J, Cha S, Masquerade Detection based on SVM and sequence-based user commands profile. ACM Symposium On Information, Computer And Communications Security. :398-400, 2007.

[34] Shavlik J, Shavlik M, Selection, Combination, and Evaluation of Effective Software Sensors for Detecting Abnormal Computer Usage, Pentagon Reports, 2004.

[35] Schultz E E, A Framework For Understanding And Predicting Insider Attacks. Journal of Computers and Security 21:526-531, 2002.

[36] Spitzner L, Honeypots: Catching the Insider Threat. Computer Security Applications Conference, 2003.

[37] Stolfo S, Apap F, Eskin E, Heller K, Hershkop S, Honig A, Svore K, A Comparative Evaluation of Two Algorithms for Windows Registry Anomaly Detection. Journal of Compauter Security 13:4, 2005.

[38] Szymanski B K, Zhang Y, Recursive Data Mining for Masquerade Detection and Author Identification. Information Assurance Workshop :424-431,2004.

[39] Tan K, Maxion R A, “Why 6” Defining the Operational Limits of stide, and Anomaly-Based Intrusion Detector. IEEE Symposium on Security and Privacy, 2002.

[40] Tuglular T, Spafford E H, A Framework for Characterization of Insider Computer Misuse. Unpublished paper, Purdue University, 1997.

[41] Wang K, Stolfo S., One-class Training for Masquerade Detection. ICDM Workshop on Data Mining for Computer Security (DMSEC), 2003

[42] Ye N, Li X, Chen Q, Emran S M, Xu M, Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data. Systems, Man and Cybernetics, Part A 31:4:266-274, 2001.

[43] Yung K H, Using Self-Consistent Naïve-Bayes to Detect Masqueraders, Stanford Electrical Engineering and Computer Science Research Journal, 2004.

Naive Bayes as a Masquerade Detector: Addressing a Chronic Failure

Kevin S. Killourhy and Roy A. Maxion

Computer Science Department, Carnegie Mellon University

Abstract Masquerade detection undertakes to determine whether or not one computer user has impersonated another, typically by detecting significant anomalies in the victim's normal behavior, as represented by a user profile formed from system audit data, command histories, and other information characteristic of individual users. Among the many intrusion/masquerade-detection algorithms in use today is the naive Bayes classifier, which has been observed to perform imperfectly from time to time, as will any detector. This paper investigates the prospect of a naive Bayes flaw that prevents detection of attacks conducted by so-called "super-masqueraders" whose incursions are consistently undetected across an entire range of victims. It is shown in this paper, through controlled experimentation and a rigorous mathematical exposition, that a weakness in the detector causes it to miss attacks under certain conditions. Furthermore, meeting those conditions – and crafting an undetectable attack – is often entirely within the control of the attacker. This paper also demonstrates, however, that such attacks can be overcome by fortifying the algorithm with a diverse detection capability. The "fortified" detector improves detection and, more significantly, removes the threat of the super-masquerader, virtually eliminating the impact of the algorithm's defect.

1 Introduction

Colloquially, the masquerade problem can be illustrated with the following scenario [8]. A legitimate user takes a coffee break, leaving his/her terminal open and logged in. During the user's brief absence, an interloper assumes control of the keyboard, and enters commands, taking advantage of the legitimate user's privileges and access to programs and data. The interloper's commands may comprise read or write access to private data, acquisition of system privileges, installation of malicious software, etc. Because the interloper is impersonating a legitimate user, he or she is commonly known as a masquerader. The term may also be extended to encompass the case of abuse of legitimate privileges, that is, the case in which a user "masquerades" as him or herself to act maliciously.

The assumption underlying the common approach to detection of such illegitimate activity is that the masquerader's behavior (including the illegitimate behavior of a legitimate user) will deviate from the historical behavior of the legitimate user. Profiles of normal user activity can be constructed from system, accounting, or other log data that contain information such as time of login, physical location of login, duration of user session, cumulative CPU time, particular programs or commands executed, names of files accessed, and so forth [4]. Illegitimate activity will appear anomalous with respect to the user profile and, when detected, will indicate likely masquerade activity.

It is natural to assume that no masquerade detector will be perfect; some masquerade attacks will go undetected, and some alarms will be raised even when there is no masquerader present. Moreover, it would not be unexpected to observe that a particular masquerader is successful at compromising some users without being detected, and unsuccessful at compromising others. It *would* be surprising, however, to discover that some masqueraders are able to escape detection no matter who their victim is – and it is exactly this peculiarity that was observed in earlier work by our lab, a phenomenon here termed the "super-masquerader" [6].[1] Specifically, we define a super-masquerader to be a masquerader who is able to compromise any user's account without being detected. In the earlier work, two super-masqueraders were found, whose commands always evaded detection.

That work suggested [p.12] that this super-masquerader phenomenon might be due to the use of commands that had never before been seen by the detector, but the issue was not pursued (because potential confounds prevented the necessary causal analysis). It would be alarming to find that such commands are the cause of the super-masquerader phenomenon since an attacker could use uncommon commands to reduce the chance of detection. The present work takes that suggestion as a point of departure, explores the reason behind the super-masquerader phenomenon, and shows how it might be negated.

This work is a departure from most earlier work in masquerade detection in which little effort had gone into understanding why a detector failed, and instead, most effort was spent searching for newer, better detection algorithms. Our alternative strategy is to understand the shortcomings of a detector and, by fortifying it, to improve it.

2 Related Work

There have been several attempts to tackle the problem of detecting masqueraders. A nice collection of such work, in which a number of masquerade-detection techniques were applied to the same data set, was developed by Schonlau and his colleagues [12]. They compared the performance of six masquerade-detection algorithms (some new, others drawn from the computer science literature), targeting a false alarm rate of 1%. All methods had relatively low hit rates (39.4% - 69.3%) and high false alarm rates (1.4% - 6.7%). The results were compared using both

[1] In that work, columns 7 and 24 of Table 6 show no + or * symbols, either of which would indicate a detection; none of the masquerader attacks were detected.

cluster analysis and ROC curves, revealing that no single method completely dominated any other.

In terms of minimizing false alarms, the best result achieved in the Schonlau et al. work used a metric based on the uniqueness of commands to a user, obtaining a 39.4% hit rate with a corresponding 1.4% false alarm rate. In terms of detecting masqueraders, the best results reported in the Schonlau et al. work were for a Bayes one-step Markov model, with a 69.3% hit rate and a 6.7% false alarm rate.

In previous work [7], our lab reasoned that masquerade detection bore similarities to text classification, and experimented with a new detector based on the naive Bayes classifier (which will be described in detail in Sections 3, 5 and 6). The Bayes classifier had performed well in text classification tasks [9], and it was expected that it would perform well as a masquerade detector, too. This intuition was borne out when we applied the detector to the same dataset used by Schonlau et al.; the naive Bayes classifier achieved a 61.5% hit rate with a corresponding 1.3% false alarm rate, for an improvement of 55.78% over Schonlau's best detector (uniqueness) when the cost of misses and false alarms are assumed to be equal. Our lab later conducted an extensive study of masquerade detection accompanied with error analysis, providing insight into the causes of masquerader success and failure [8].

All of the above work used data in the form of truncated command lines, meaning that the data comprised user-level commands issued at the command line of a Unix shell, but with no command-line arguments, flags or grammar. In a subsequent approach, our lab [6] used the naive Bayes detector on a data set of enriched command lines (including arguments, flags, etc.), anticipating improvements due to the extra information on the command lines. A hit rate of 82.1% was achieved, accompanied by a false alarm rate of 5.7%, concomitantly reducing the cost of error by 30.02%. It was in this study that the super-masquerader phenomenon was discovered: super-masqueraders are able to escape detection no matter who their victim is.

Observation of a phenomenon like the super-masquerader leads one to wonder if there is some unanticipated interaction between the detection algorithm and the characteristics of the data on which it operates. Although other researchers seem not to have observed the super-masquerader, they have investigated some aspects of the relationship between data characteristics and naive Bayes performance. Mitchell [10], for example, provides a standard reference for the naive Bayes classifier. He discusses the mechanism – called an *m*-estimate or pseudocount – by which the classifier avoids the undue influence of certain features of the data on the decision procedure. As the current work will elucidate, the pseudocount plays a previously unanticipated role in the super-masquerader phenomenon.

Domingos and Pazzani [1] examined the degree to which the independence of features in a data set affect naive Bayes' performance, and whether violations of naive Bayes' "independence assumption" hurt performance. They showed that in many cases where this assumption is violated, naive Bayes can still perform well as a classifier. The probability estimations may be incorrect, but they still lead to the correct choice of class. Indeed, spam filters using naive Bayes are successful in practice. Rish et al. [11] studied the performance of naive Bayes when the independence assumption is grossly violated, e.g., when all other features are one-to-one functions of a particular feature. They found that naive Bayes performs very well under such conditions, and determined that the entropy of the probability distribution of a feature (conditioned on the class of the data point) is a better predictor of naive Bayes performance than measures of the independence of features (e.g., mutual information).

3 Background on Naive Bayes

Naive Bayes classifiers are simple, probabilistic classifiers known for their inherent robustness to noise and their fast learning time. The classifier works as follows when applied to the problem of user profiling at the Unix command line [7]. Sequences of commands from a potential victim (designated as the *self user*) and sequences of commands from all other legitimate users (designated as *nonself* users) are used to train the classifier. A set of *self* training data consisting of a sequence of commands generated by the *self* user is used to build a model of normal behavior (*self*). A set of *nonself* training data consisting of sequences of commands generated by *nonself* users is used to build a model of abnormal behavior (*nonself*).

The *self* model is used to estimate the probability that a new *test block* of commands was generated by the *self* user. This probability is called the *self* probability. Likewise, the *nonself* model is used to estimate the probability that the block was generated by a *nonself* user. This probability is called the *nonself* probability. With the *self* and *nonself* probabilities, naive Bayes computes an anomaly score for the test block. If the anomaly score is greater than a threshold, naive Bayes decides that a masquerader is present and raises an alarm. The threshold is calculated using cross-validation; the procedure used by naive Bayes to estimate probabilities and calculate the anomaly score is described and analyzed more fully in the sections below.

4 Objective and Approach

The objective of the present research is to understand the phenomenon of the super-masquerader and to determine whether or not there are measures available to counteract it.

Never-before-seen commands (NBSCs) are commands which are new to naive Bayes, having never appeared in the *self* or *nonself* training data. Starting with the aforementioned observation that these NBSCs might cause the super-masquerader phenomenon, it is hypothesized that NBSCs tend to lower the naive Bayes anomaly score such that it drops below the threshold. Such a low anomaly score causes the algorithm to inappropriately favor the victim instead of the masquerader, with the consequence that the detector cannot recognize the attack. It may seem surprising that NBSCs could cause the anomaly score to drop, but the fact that NBSCs are used by neither the *self* user nor the *nonself* users makes it less clear whether they raise or lower anomaly scores.

This paper explores the hypothesis in three stages. First, to ascertain whether the hypothesis is reasonable, an empirical investigation with synthetic data will measure the effects of NBSCs, controlling for confounding variables in the environment. Second, a rigorous mathematical analysis will reveal the algorithmic basis for the findings of the empirical investigation. Finally, with the adverse effect of NBSCs confirmed, a strategy to fortify naive Bayes against this phenomenon will be composed and evaluated.

5 Experiment With Synthetic Data

It was suggested previously that NBSCs caused the super-masquerader phenomenon after observing their prevalence in the test blocks of super-masqueraders [6]. However, since that study did not control for confounding variables (e.g., other features of the test blocks that might explain the phenomenon), nothing more than the observed association between NBSCs and super-masqueraders could be inferred.

In this section, the confounding variables are identified and controlled in order to ascertain whether the use of NBSCs could cause an attack to be missed. Confounding variables include the behavior of the victims as well as the behavior of the masquerader (beyond the use of NBSCs). These other variables of the data were controlled and manipulated in an experimental study in order to isolate the effect of NBSCs on naive Bayes.

In this experiment, naive Bayes is treated as a "black box" that calculates an anomaly score when provided with a data set consisting of *self* and *nonself* training data and a test block. A series of synthetic data sets was used to measure the effect of NBSCs on the naive Bayes detector across a wide range of conditions, including those that are common and those that are rare in practice. This section provides details of the methods and data used in the experiments as well as the experimental results. The following are addressed: (1) selection of confounding variables to control; (2) format of the data sets; (3) experiment control for the generation of data sets; and (4) procedure for running naive Bayes on each data set.

5.1 Variable Selection

The first step in conducting the experiment is to identify the independent variables of interest. Since the purpose of this experiment is to observe the effect that

NBSCs have on naive Bayes' anomaly score, the most important variable to control and manipulate is the number of NBSCs in the test block (denoted k). In this experiment, the test block will contain 10 commands, and so k can range from 0 to 10.

However, other variables of the data set affect the anomaly score, and may even alter the effect of NBSCs. Controlling for the effects of confounding variables allows their influence to be isolated, which in turn allows the effects of NBSCs to be measured independently.

There are infinitely many ways to characterize a test block and *self* and *nonself* training data, each of which could be considered a variable to control. However, not all of these potential variables influence the anomaly score. For instance, naive Bayes ignores the order of the commands in the test block; as a consequence, variables describing the order of commands in a block would have no effect on the anomaly score.

Three potential confounding variables were selected to be controlled because of their effects on the anomaly score. These three variables, according to the theory on which the naive Bayes detector is based, were judged to have the largest effect. They are described below, and the next section shows how they are controlled and used when constructing a data set.

The first confounding variable is the number of users in the *nonself* training data, denoted m. This variable can affect the "prior probabilities" that naive Bayes estimates from the training data. Since such estimates are inaccurate in the domain of masquerade detection, our lab previously [7] reduced the influence of this variable (by altering the way the naive Bayes detector calculates prior probabilities), but did not eliminate its influence completely. In this experiment, the effect of m over the range of 1 to 10 users will be considered.

The second confounding variable is the number of times the commands in the test block also appear in the *self* training data. For each command c_i in the test block, the relative frequency of that command in the *self* training data affects the way naive Bayes calculates the *self* probability. (The subscript i indicates the order of the command c_i in the test block.) The number of times command c_i appears in the *self* training data is denoted $s[c_i]$, and in this experiment $s[c_i]$ will range from 0 to 10 commands.

The third confounding variable is the number of times the commands in the test block appear in the *nonself* training data. The reason parallels that of the second variable, specifically, the effect on the *nonself* probability calculation. The *nonself* training data may include data from one or more users (depending on the value of the first confounding variable m). The number of times command c_i appears in any one user's data from the *nonself* training data is denoted $n[c_i]$, and in this experiment $n[c_i]$ will range from 0 to 10 commands.

Four variables have been selected for control: one variable of interest (k) and three potential confounding variables (m, $s[c_i]$, and $n[c_i]$). In the remainder of this section, these four variables will be arranged into a 4-element vector ($k, m, s[c_i], n[c_i]$). This vector will guide the generation of synthetic data and the evaluation of naive Bayes under the conditions set forth.

5.2 Synthetic Data

There are three types of data in each data set: the *self* training data (for building the model of normal behavior), the *nonself* training data (for building the model of abnormal behavior), and the test block (commands generated by the potential masquerader). The generation of each synthetic data set can be viewed as a procedure that takes as input a particular value for each of the four variables in the vector ($k, m, s[c_i], n[c_i]$), and constructs *self* and *nonself* training data sets, plus a test block. These data are constructed as follows.

***Self* training data** The *self* training data comprise 1000 commands. Since the experimental objective is to examine the effect of commands that are novel to the naive Bayes classifier (NBSCs), and these novel commands are always outside the scope of the *self* data, there is no need for the *self* data ever to change. Therefore, this data set remains static throughout the experiment; i.e., there is only one *self* data set. The length of 1000 was chosen to be consistent with previous work done in masquerade detection (e.g., [6, 7]), as well as its convenience as a multiple of 10.

The *self* training data was populated with commands[2] selected from the 11×11 matrix in Table 1 so that the distinct elements in the data set would range widely

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
1	01	02	03	04	05	06	07	08	09	10	11
2	12	13	14	15	16	17	18	19	20	21	22
3	23	24	25	26	27	28	29	30	31	32	33
4	34	35	36	37	38	39	40	41	42	43	44
5	45	46	47	48	49	50	51	51	53	54	55
6	56	57	58	59	60	61	62	63	64	65	66
7	67	68	69	70	71	72	73	74	75	76	77
8	78	79	80	81	82	83	84	85	86	87	88
9	89	90	91	92	93	94	95	96	97	98	99
10	100	101	102	103	104	105	106	107	108	109	110
11	111	112	113	114	115	116	117	118	119	120	121

Self-ness

Nonself-ness

Table 1. Matrix of commands used to populate the *self* and *nonself* training data with commands that range in relative frequency.

[2] The commands shown here are numeric for convenience, but are treated as being categorical in the detection experiments.

in relative frequency. Naive Bayes calculates a higher *self* or *nonself* probability for commands with higher relative frequencies in each respective training set. Consequently, it is possible to populate the *self* training data with commands that range in relative frequency (from .001 to .01), by taking commands with increasing repetition from each *row* of the command matrix. 605 of the 1000 commands in the *self* data are chosen from the cells of Table 1 (with replacement). Each of the 11 commands in row 1 of the table appears once in the data (11 commands); each of the 11 commands in row 2 appears twice (22 commands); each from row 3 appears three times (33 commands), and so on up to the 11 commands in row 10 which appear 10 times each (110 commands). The commands in row 11 do not appear in the *self* data. The commands from rows 1 through 10 of Table 1 are 605 of the 1000 commands in the data. The remaining commands comprise 395 repetitions of a 111th command, and are used to pad out the data set to 1000 commands ("122" is the symbol used for the 111th command, because 122 is outside the scope of the table).

Although the naive Bayes algorithm is not influenced by the order in which these commands are arranged in the data, the commands were distributed as evenly as possible across five blocks of 200. For example, the first 11 commands would be arranged so that two commands are placed in each of the five blocks (totaling 10) and the 11th either remaining in the last block, or overflowing to the first block. Of the second set of 11 commands, the first two would appear twice each in the first block, and so on through the 5th block, with similar treatment of overflow. This method of distributing the commands evenly throughout the 1000 commands ensured that the results of the 5-fold cross-validation step (in learning the decision threshold between *self* and *nonself*) would not be biased by a preponderance of one group of commands in any of the five blocks of 200 commands.

***Nonself* training data.** The *nonself* training data set comprises 1000 commands for each of *m* users (for a total of 1000*m* commands), where *m* is the second of the four variables that act as input parameters to the data synthesis procedure. Each of the *m* users' data sets are constructed similarly to the *self* training data, except that the columns of the command-matrix in Table 1 are used, not the rows. Commands in column 1 appear once in each *nonself* user's training data, commands in column 2 appear twice, and so forth, through column 10, which is used ten times. Commands in column 11 do not appear in the *nonself* training data.

Test block. The test block (10 commands) is populated with the appropriate proportion of seen-before commands (SBCs) and never-before-seen commands (NBSCs) as directed by the input parameters. Specifically, the input parameter *k*, which ranges from 0 to 10, specifies the number of NBSCs in the test block. Note that command "121" in row 11, column 11 of Table 1 appears in neither *self* nor *nonself* training data; it is the designated NBSC. This NBSC is replicated as many times as required by the parameter value *k*, and the remainder of the test block is padded out with SBCs, chosen as follows. The input parameter $s[c_i]$, which ranges from 0 to 10, is used to select a row from the command-matrix in Table 1. (If $s[c_i]$ is zero, denoting a command that never appears in the *self* training data, row 11 is selected.) Likewise, the input parameter $n[c_i]$ is used to select a column from the command-matrix. The command c_i in the selected row and column of the command-matrix is chosen as the SBC for the test block. The chosen SBC is replicated as many times as required to pad out the test block to 10 commands.

5.3 Experiment Control

The generation of data sets was controlled by stepping through the combinations of values for the variables in the 4-element vector: (k, m, $s[c_i]$, $n[c_i]$). The vector encodes particular values for each of the four parameters that describe features of the data set to be generated. For example, the parameter vector (6, 5, 3, 4) corresponds to a situation in which the test block has six never-before-seen commands (k = 6); there are five users in the *nonself* training data (m = 5); the seen-before command in the test block appears three times in the *self* training data ($s[c_i]$ = 3) and four times in each of the *nonself* user's training data ($n[c_i]$ = 4). Note that the particular command is "26" as drawn from row 3 and column 4 of Table 1. The data set described by this parameter vector can be constructed using the procedure outlined in the previous section.

The parameter vector's starting state is (0, 1, 1, 0). The second parameter (m) starts at 1 because the number of *nonself* users must be greater than zero. The third parameter ($s[c_i]$) starts at 1 to avoid selection of the NBSC as a seen-before command. The other two parameters begin at zero. Ten was chosen as an upper bound on the range of all the parameters, so the parameter vector's final state is $\langle 10,10,10,10 \rangle$.

There are 10 possible values for the first parameter, and 11 for each of the other parameters, giving a total of $10\times11\times11\times11=13310$ different combinations. However, the combinations where the NBSC would be selected as the seen-before command are excluded. When $s[c_i]$ and $n[c_i]$ are both zero, there are still 10 values for the first parameter and 11 values for the fourth parameter, which gives $10\times11=110$ combinations that must be excluded. Each of the $13310-110=13200$ distinct parameter vectors was used to generate 13200 data sets, and naive Bayes was run on each.

5.4 Procedure

Naive Bayes operates by learning a model of normal behavior (legitimate user) from *self* training data and a model of abnormal behavior (masquerader) from *nonself* training data. It uses these models to calculate an anomaly score on test data (in this case, a single block of test data), and uses that score to decide whether to raise an alarm. For each synthetic data set, the steps in the procedure for running naive Bayes were as follows.

1. **Configure naive Bayes.** Set the naive Bayes "pseudocount" parameter to 0.01 (for consistency with prior work [8]). Set the block size to 10, and the alphabet size to 122 (for consistency with the data set).
2. **Train on *self* and *nonself* data.** Train the naive Bayes classifier on *self* data; establish a model of normal behavior. Train on the *nonself* data to establish a model of abnormal behavior.
3. **Compute anomaly threshold.** Compute the anomaly threshold by 5-fold cross validation. Details of how cross validation works can be found in an earlier paper [8].
4. **Score the test block.** Run the detector on the test block, and observe the anomaly score that naive Bayes assigns to the block.
5. **Decision.** Decide whether or not the test block was detected by comparing the anomaly score with the threshold.

5.5 *Results and Analysis*

The principal result of this experiment is that NBSCs do cause naive Bayes to miss masquerade attacks. The effect of NBSCs can be seen in Figure 1, where the number of NBSCs in a test block is compared to the percentage of runs where the

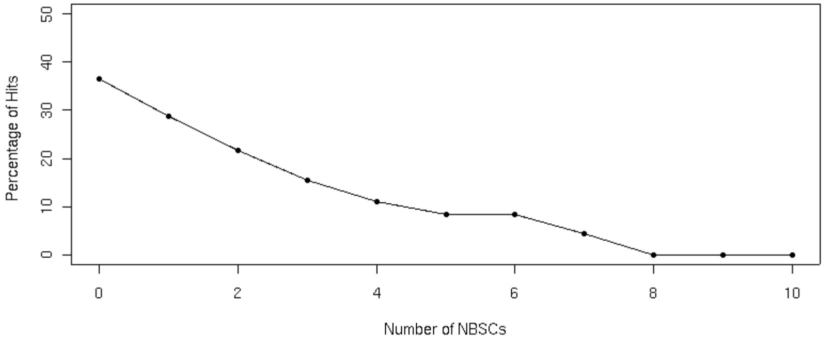


Fig. 1. Percentage of synthetic data sets in which the test block is detected (hit rate), broken down by the number of NBSCs in the block.

test block is detected as a masquerader, i.e., the hit rate. The number of NBSCs ranges from 0 to 10 (i.e., the range of the variable k) across the 13200 data sets. The hit rate for a specified number of NBSCs is calculated as the percentage of synthetic data sets with that number of NBSCs in which naive Bayes was able to detect the test block as a masquerader. (Note that 100% detection is not expected here since at least half of the test cases driven by the 4-element control vector in Section 5.3 were expected to have a higher *self* probability than *nonself* probability.) With no NBSCs, 36.28% of the test blocks are able to be detected. With only a single NBSC in the test block, that percentage drops to 28.59%. As the number of NBSCs increases, the hit rate falls. When the test block consists of eight or more NBSCs, the percentage of detected blocks drops to zero.[3] This result offers strong support for the hypothesis that NBSCs lower the anomaly score and cause naive Bayes to miss masqueraders, regardless of potential confounding factors.

6 Naive Bayes Mathematical Formulation

It is natural to wonder why NBSCs seem to have a strong influence on the results of naive Bayes This section formulates a mathematical description of naive Bayes to help answer this question. The naive Bayes decision procedure is expressed as an equation, and manipulated algebraically to pinpoint when and why NBSCs cause the anomaly score to decrease (thereby missing a masquerade activity).

6.1 Calculating the Anomaly Score

As described in Section 4, naive Bayes uses the *self* and *nonself* training data to build *self* and *nonself* models. It calculates a *self* and *nonself* probability for each test block, and then combines them into an anomaly score that it uses to decide whether or not to raise an alarm. When estimating probabilities, naive Bayes assumes that a test block was generated either by the *self* user (denoted S) or the *nonself* user (denoted N). Both users are assumed to generate a sequence of commands, with every command occurring with some fixed probability based on its relative frequency in the training data. The probability that a given command appears is assumed to be independent of the presence of other commands in the block. (This assumption is what makes this kind of Bayesian classifier "naive.").

[3] The range of values selected for each of the parameters in the data generation process (e.g., $s[c_i]$ and $n[c_i]$) did not take into account the non-linear nature of the anomaly score (which will be shown in the next section). Consequently, the scores fall in clumps. The flat spot in Figure 1, where the hit rate is unchanged between NBSC values 5 and 6, is an artifact of the range of values considered. Had this range been extended, or the values been chosen differently, the value at 6 would be lower and the downward trend would be maintained.

Consequently, the conditional probability of a command c_i generated by the *self* user is estimated as in Equation 1.

The parameter p is a pseudocount, and α is the number of distinct commands in the data (i.e., the alphabet). The pseudocount can be any real number larger than zero (0.01 in this study as in earlier ones [6, 7]), and is included in the calculation to ensure that every command (even those not appearing in the training data) has a non-zero probability estimate. In a Bayesian framework, the pseudocount can be interpreted as the prior probability that each command was generated by the user.

$$P(c_i \mid S) = \frac{s[c_i] + p}{s_t + \alpha p} \qquad (1)$$

c_i:	a command (e.g., c_i = 'ls')
S:	the *self* model
$s[c_i]$:	the number of times c_i is in the *self* training data
s_t:	the total size of the *self* training data
p:	naive Bayes pseudocount
α :	alphabet size

The conditional probability P (c_i | N) of a command c_i generated by a *nonself* user is estimated in a similar fashion, except that $s[c_i]$ is replaced by n_m $[c_i]$, the number of times c_i is in the *nonself* training data; and $\mathrm{s_t}$ replaced by $\mathrm{n_t}$, the number of commands in the *nonself* training data. Whereas $n[c_i]$ denotes the number of appearances of c_i in one *nonself* user's training data, n_m $[c_i]$ denotes the number of appearances across all of the *nonself* training data (comprising m users).

To calculate the conditional probability of a test block of b commands generated by the *self* user (where c_1, c_2, . . ., c_b denote the commands), naive Bayes uses two simplifying assumptions. The first is the independence assumption just described: the conditional probability of each command is considered independent of the conditional probabilities of the other commands in the test block. The second assumption is that a sequence of commands in the test block is just a "bag of words." That is, the order of the commands does not matter; only their frequency matters. Because of these assumptions, the conditional probability that a block of commands was generated by the legitimate user (in other words, the commands belong to the *self* set) is the product of the conditional probabilities that each individual command in the block was generated by that user. The conditional probability that the test block was generated by a masquerader is similarly calculated from the respective probabilities of its component words. For ease of computation, the logarithm of these products is calculated, rather than the products themselves.

In traditional Bayesian statistics, these conditional probabilities would be multiplied by "prior" probabilities (denoted $P(S)$ and $P(N)$) to calculate the "posterior" *self* and *nonself* probabilities using Bayes' theorem. However, in the domain of masquerade detection, estimating $P(S)$ and $P(N)$ is akin to estimating the probability of a masquerader attacking at any given moment—something hard to calculate at all, much less with precision. Consequently, prior work [7] skirted the issue and assumed uniform priors ($P(S) = P(N) = 1/2$), which cancel each other out in the calculation of the anomaly score.

The anomaly score of the test block is defined as the ratio of the *self* log-probability of the test block to the *nonself* log-probability of the test block, as shown in Equation 2. Logarithms are used to mitigate the errors in numerical accuracy that accumulate when running calculations with small numbers (e.g., probability estimates). The ratio of the logarithms is calculated in order to convert the

$$\text{score} = \frac{\sum_{i=1}^{b} \log \dfrac{s[c_i]+p}{s_t+\alpha p}}{\sum_{i=1}^{b} \log \dfrac{n_m[c_i]+p}{n_t+\alpha p}} \qquad (2)$$

b:	number of commands in the test block
c_i:	a particular command in the test block
$s[c_i]$:	number of c_is in *self* training data
$n_m[c_i]$:	number of c_is in *nonself* training data
s_t:	the total size of the *self* training data
n_t:	total size of *nonself* training data
p:	naive Bayes pseudocount
α :	alphabet size

self and *nonself* probabilities into a single value that can range from 0 to infinity, with higher values as indicators of masqueraders. If the anomaly score is greater than the configured threshold, the naive Bayes detector raises an alarm. The threshold value can be set by hand or can be automatically determined using the *self* and *nonself* training data in a cross-validation experiment [7].

6.2 Manipulating the Anomaly Score

Given the formula for calculating the anomaly score (shown in Equation 2), it can be algebraically manipulated to make the influence of NBSCs clear. In this section, the equation is thus rewritten to express the number of NBSCs as a parameter in the decision procedure, and to analyze their influence on the score.

In Equation 2, since both $\log(s_t + \alpha p)$ and $\log(n_t + \alpha p)$ are constant with respect to the summations, they can be factored out. Further, since naive Bayes trains on equal amounts of command data for each user, the amount of *nonself* training data (denoted n_t) is simply the amount of *self* training data (s_t) multiplied by the number of *nonself* users (i.e., those whose command data went into making the *nonself* training set). The number of *nonself* users is denoted m; thus, $n_t = ms_t$.

Next, a simplifying assumption is made that commands appear with the same frequency across all *nonself* users. (The naive Bayes detector pools the data of all the *nonself* users, so nothing is lost by making this assumption. It does make the forthcoming analysis clearer, however.) Since $n_m[c_i]$ is the number of times c_i appears in all the *nonself* training data, and c_i appears the same number of times in each *nonself* user's training data, if $n[c_i]$ denotes the number of appearances in one user's data, it follows that $n_m[c_i] = mn[c_i]$. By substituting ms_t for n_t and $mn[c_i]$ for $n_m[c_i]$ in Equation 2 and factoring out the denominators within each of the summations, the effect of the number of *nonself* users (m) can be separated from the summation.

Finally, the role NBSCs play in the calculation of the anomaly score can be formulated. Since the order of the commands in the test block does not matter, we assume, (without loss of generality) that any NBSCs in the test block occur in the first k commands. I.e., $c_1, \ldots, c_k$ are NBSCs while $c_{k+1}, \ldots, c_b$ have been seen before in at least one of the *self* or *nonself* sets of training data. More formally, if c_i is a NBSC (i.e., $i \le k$), then $s[c_i] = n[c_i] = 0$. Consequently, the effect of these commands can be factored out from the rest of the summation in both the numerator and the denominator. Equation 3 shows the final expression of the anomaly score after the denominators and NBSCs have been separated from the summations.

$$\text{score} = \frac{\overbrace{k \cdot \log(p)}^{\text{Part A}} - \overbrace{b \cdot \log(s_t + \alpha p)}^{\text{Part B}} + \overbrace{\sum_{i=k+1}^{b} \log(s([c_i] + p)}^{\text{Part C}}}{\underbrace{k \cdot \log(p)}_{\text{Part A}} - \underbrace{b \cdot \log(ms_t + \alpha p)}_{\text{Part B}} + \underbrace{\sum_{i=k+1}^{b} \log(mn[c_i] + p)}_{\text{Part C}}} \qquad (3)$$

b:	the number of commands in the test block
k:	number of NBSCs in the test block
c_i:	a particular command in the test block
$s[c_i]$:	number of c_is in *self* training data
$n[c_i]$:	number of c_i s in a *nonself* user's training data
s_t:	total size of *self* training data
m:	the number of *nonself* training users
p:	naive Bayes pseudocount
α:	alphabet size

The labeled parts of Equation 3 attempt to demarcate various influences on the anomaly score: Part A represents never-before-seen commands (NBSCs); Part B represents training data size; and, Part C represents the similarity of the seen-before commands in the test block to the *self* and *nonself* training data.

6.3 Effect of NBSCs

Both the numerator and the denominator in Equation 3 are negative numbers (since they are sums of the logarithms of probability estimates between zero and one). Consequently, if the numerator is more negative than the denominator, the score will be high, whereas if the denominator is more negative, the score will be low.

Both the numerator and the denominator of the anomaly score calculation can be effectively divided into three regions. The first region, labeled Part A, indicates the influence of the number of NBSCs. In both the numerator and the denominator, the Part-A region has the same value. This region is unbiased in that it has the same effect on both the numerator and the denominator and does not influence the score toward low *self*-like values or high *nonself*-like values.

The second region, labeled Part B, indicates the influence of the size of the *self* training data (s_t) and the number of users in the *nonself* training data (m). If the number of *nonself* users is 1 (i.e., $m = 1$), then Part B of the numerator and Part B of the denominator are equal, and this region has no effect on the anomaly score. However, if the number of *nonself* users is greater than 1 (i.e., $m > 1$), which has been the case in practice, then Part B of the numerator will be a small positive number and Part B of the denominator will be a large positive number. Since Part B is subtracted from the rest of the summation, the overall result will be to make the denominator more negative than the numerator, which results in a lower anomaly score (indicative of legitimate or *self*-like behavior).

The third region, labeled Part C, indicates the remaining influence of seen-before commands. This region can be unbiased, biased toward *self*, or biased toward *nonself*, depending on how often these seen-before commands appear in the *self* and *nonself* training data. It is important to note that of the three regions, this is the only one that can have a *nonself* bias. (Part A is unbiased, and Part B is either unbiased or biased toward *self*.) If a test block causes naive Bayes to compute a high anomaly score, it is because of a *nonself* bias from the seen-before commands in this region.

Notice that if an individual seen-before command were to become an NBSC, the overall influence of Part C would diminish, and the influence of Part A would increase. In other words, if the anomaly score were high (due only to the influence of seen-before commands), changing a seen-before command (with its *nonself* bias) to an NBSC (with its lack of bias) would diminish the anomaly score. Coupled with a *self* bias exerted by Part B (which is stronger if the number of *nonself* users is larger), the replacement of seen-before commands with NBSCs will clearly lower an initially-high anomaly score.

The cause of a bias in Part B is somewhat subtle, but it is rooted in the effect of the pseudocount parameter on the calculation. Again, this term slightly adjusts the probability of each command so that NBSCs are not assigned a zero conditional probability. However, the discrepancy between the numerator and denominator of Part B shows that this adjustment is larger for small training-data sets than for big training-data sets. Consequently, NBSCs are considered to be generated with higher probability from the *self* user (consisting of a smaller training-data set) than from the *nonself* user (a larger training-data set). This difference between the numerator and denominator produces a low anomaly score.

This analysis explains the mechanism by which NBSCs cause masqueraders to be missed. Specifically, in the data set where super-masqueraders were observed [6], there were 49 *nonself* users in the training data. The Part B region for such a data set would have a very strong bias toward *self*. NBSCs in the masquerade blocks do not directly lower the anomaly score, but they allow the underlying bias toward *self* (from the number of *nonself* users) to exert itself.

7 Exploiting NBSCs to Cloak Attacks

The experimental results and analysis of the previous sections have shown that NBSCs do cause naive Bayes to fail and to misclassify attacks. This section shows that such failures are not of purely academic interest, but actually represent a vulnerability of this classifier that can be practically exploited by a masquerader. A masquerader who tailors his or her attack so that the majority of commands typed are NBSCs stands a good chance of getting away undetected. In this domain, where the commands are typed at the Unix command line, there are several ways a masquerader can introduce NBSCs to cloak an attack.[4]

1. **Type nonsense.** If the detector looks at every command typed at the command line, regardless of whether it results in a program being run, a masquerader can easily introduce NBSCs with nonsense commands. For instance, if the masquerader types nonsense such as "`alksf`," the command-line interface will simply print an error message, but naive Bayes will treat the nonsense command as an NBSC.[5]
2. **Use aliases.** A masquerader can create never-before-seen aliases to commands that he or she wants to run. For instance, if a masquerader issues the command "`alias alksf 'rm -rf'`" (using C-shell syntax), it is then possible to remove files with the command "`alksf`."

[4] To cloak an attack is to modify it to take advantage of weak spots in a detector's coverage, and hence go undetected. See [14] for an example.

[5] There is an interesting correspondence between a masquerader injecting nonsense commands and a spammer padding email messages with unusual words to avoid detection by Bayesian filters. Recently, the unusual-word padding strategy seems to have been abandoned by spammers in favor of padding spam messages with very common words. Nonetheless, the correspondence suggests a potentially fruitful exchange of ideas between these fields (insider-threat and spam detection).

3. **Create symlinks.** Similarly, a masquerader could create a symlink to `rm` with the command "`ln -s /bin/rm alksf`" and then use `./alksf` to remove files. Unlike the use of aliases, this technique would work even if the detector is fed the name of the program executed rather than what was typed by the user on the command line.

8 Naive Bayes Fortification

Since masqueraders might use NBSCs to turn themselves into super-masqueraders, the aforementioned vulnerability in naive Bayes must be addressed. In this section, a strategy for mitigating the effect of NBSCs is proposed and evaluated. By fortifying the naive Bayes detector with an additional "NBSC detector," and using this second detector to compensate for a weakness in the first, a new detector is constructed that performs better than regular naive Bayes on a masquerader data set containing super-masqueraders.

8.1 The Fortified Detector

Fortification is simply a matter of identifying the weakness in naive Bayes, and building an auxiliary detector that is strong where naive Bayes is weak.[6] The previous sections have shown naive Bayes to be susceptible to evasion via NBSCs. However, NBSCs are nothing more than foreign symbols, i.e., commands that never appear in the training data. Any detector that can detect foreign symbols would act as suitable fortification. For instance, stide [3] has been shown able to detect foreign symbols [15], and the PHAD/PAD algorithm [5, 13] could also be adapted to serve this role. However, these detectors do not limit themselves to detecting only foreign symbols, and so they are more complex than what is needed here. A simple foreign-symbol detector was created, called the NBSC detector.

This novel, but simple, NBSC detector uses a standard look-up table to track NBSCs. There are two phases: training and testing. During training, the NBSC detector uses all of the data from all of the users (both *self* and *nonself*) to populate a look-up table. During testing, the NBSC detector counts how many NBSCs appear in a given test block, as follows. First, a counter (of NBSCs) is initialized to zero. Then, each command in the test block is sought in the look-up table. If the command cannot be found, the counter is incremented (since the command is an NBSC). Finally, the counter is compared to a threshold, and if the count exceeds the threshold, the NBSC detector generates an alarm. In our experiments, the

[6] Other fortification strategies exist, e.g., normalizing the pseudocount to scale with the size of the training data, or refining naive Bayes to estimate the conditional probability of NBSCs from the training data. The current strategy was chosen because it was simple and likely to work.

threshold is set so that an alarm is generated if the majority of the commands in the test block are NBSCs.

A fortified detector was created by combining the naive Bayes detector with the NBSC detector. During training, each block of *self* and *nonself* data is passed to both the naive Bayes and the NBSC detector. During testing, the test block is evaluated by both detectors. If either detector generates an alarm, then the fortified detector also raises an alarm.

While this simple form of fortification may appear brittle, evaluation shows that it is quite effective. In many environments, a burst of truly unique commands that have never been used before by any users of the system (i.e., NBSCs), should raise alarms, and such is the behavior of the fortified detector.

8.2 Evaluation Methodology

The fortified detector is evaluated using the masquerader data set with which the super-masquerader phenomenon was first observed, consisting of Unix command-line data enriched with arguments, flags, etc. The data set contains 2000 commands of *self* data for each of 50 users, and blocks of 10 commands from each of 30 different masqueraders. The first 1000 commands of each user's data constitute the training data. The remaining 1000 commands are divided into blocks of 10 and interspersed with the blocks of masquerader data to constitute the test data. Note that whereas our synthetic data sets contained a test block that was known to be a masquerader, this data set contains both the legitimate user's test blocks and masquerader test blocks (and so can be used to measure the tradeoff between detecting masqueraders who use NBSCs and an increase in false alarms caused by users who legitimately use NBSCs). The data set is described fully in an earlier publication [6].

Following the same procedure that was used to evaluate the original naive Bayes detector, a separate fortified detector was built for each of the 50 users. The fortified detector for a user was trained using the first 1000 commands of that user's data as *self* training data and the first 1000 commands of the other 49 users as *nonself* training data. The naive Bayes detector's anomaly-score threshold was computed using 5-fold cross-validation as in the original experiment. The NBSC detector's NBSC count threshold was set to 5. Since a block is only 10 commands long, the NBSC detector was configured to generate an alarm if the majority of commands in the block (i.e., more than 5) are NBSCs.

For each of the 50 users, the fortified detector was used to classify each of the 100 blocks of *self* test data and the 30 blocks of masquerader data. For each block of test data, the following observations are made: (1) the anomaly score calculated by naive Bayes, (2) the NBSC count calculated by the NBSC detector, and (3) whether or not the fortified detector raised an alarm.

8.3 Evaluation Results and Analysis

Table 2 compares the overall performance of the unfortified naive Bayes detector to that of the fortified detector. The upper portion of the table shows that naive Bayes detected 1232 of the 1500 masquerader test blocks, giving it a hit rate of 82.1%, and it alarmed 286 times on 5000 normal blocks, giving it a false alarm rate of 5.7%. The lower portion of the table shows that the fortified detector detects 1332 of the 1500 masqueraders, 100 more than regular naive Bayes, giving it a hit rate of 88.8%. At the same time, it alarms on 313 of the 500 normal blocks, giving a false alarm rate of 6.3%. Using a standard measure of cost, calculated by adding the miss rate and the false alarm rate, the unfortified naive Bayes detector has a cost of 0.236, while the fortified detector has a cost of 0.175, an improvement of 34.7%.

Original naive Bayes detector

	No Alarm	Alarm	Total
No Masq.	4714 (94.3%)	286 (5.7%)	5000
Masq.	268 (17.9%)	1232 (82.1%)	1500

Fortified naive Bayes detector

	No Alarm	Alarm	Total
No Masq.	4687 (93.7%)	313 (6.26%)	5000
Masq.	168 (11.2%)	1332 (88.8%)	1500

Table 2. A comparison of the performance of the original naive Bayes detector and the fortified naive Bayes detector. Each table displays the number of blocks broken down by whether the test block was a masquerader or not, and whether the detector alarmed or not. The numbers in parentheses indicate the corresponding percentage of the total (5000 *self* test blocks and 1500 masquerader blocks).

From the NBSC counts collected in this experiment, the number of NBSCs in each masquerade block can be calculated. Figure 2 shows how NBSCs are distributed across the masquerade blocks in the data set. There are thirty distinct masquerade blocks (because the same masquerader block was injected into every user's test data). Twelve of these use no NBSCs, seven use exactly one NBSC, and the remaining 11 masqueraders use between two and eight. The masquerade blocks that use the most NBSCs correspond to the two super-masqueraders. One uses 6 and the other uses 8. Both are detected by the NBSC detector with its threshold set to 5.

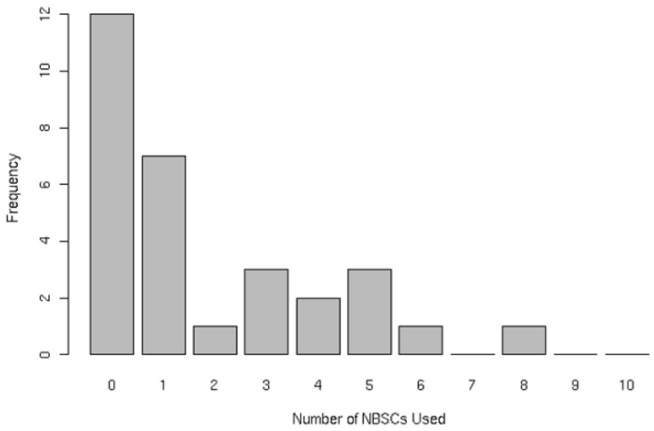


Fig. 2. Distribution (frequency) of NBSCs across the masquerade blocks. There are 30 distinct masquerade blocks; the number of NBSCs ranges from 0 to 10.

The super-masquerader phenomenon is not observed in the results from the fortified detector. Whereas the two super-masquerader blocks are missed 100% of the time by the unfortified naive Bayes detector, no masquerade block is missed more than 34% of the time by the fortified detector. This result indicates that no masquerader has better than a 34% chance of evading detection without specifically tailoring the attack to resemble a particular user's behavior (while no such user-specific tailoring is necessary to evade the naive Bayes detector with NBSCs).

9 Discussion

In the experiments with synthetic data, the super-masquerader phenomenon was found to be caused by NBSCs. In a mathematical analysis, the cause was traced to a subtle weakness in the naive Bayes classifier that was aggravated by a feature of the masquerade detection domain. Specifically, naive Bayes uses a pseudocount to adjust the probability of every command, because it would otherwise be unable to cope with previously unseen commands. However, this adjustment varies depending on the relative sizes of the *self* and *nonself* training data, and in masquerade detection, there is usually a much greater supply of *nonself* training data than *self* training data. As such, naive Bayes remains unable to cope with previously unseen commands (i.e., NBSCs), resulting in missed attacks. It was shown that masqueraders can take advantage of this weakness in the detector to intentionally modify an attack to evade detection. One significant lesson to be learned from this discovery is that weaknesses of a classifier that appear to be merely of theoretical interest can become vectors of attack when those classifiers are operating in adversarial environments such as computer security.

Once the cause of the super-masquerader phenomenon was discovered empirically, and verified mathematically, a simple fortification technique was able to remove the vulnerability. Between 1998 and 2001, Schonlau and others [2, 7, 12] had investigated a number of extraordinarily different techniques for detecting masqueraders. Analogous to the situation with naive Bayes, the alternative detection strategies yielded mixed results. Some masqueraders were detectable, and others were not. Some users' data generated many false alarms; others few. While considerable effort had been expended to find a detector that would detect masqueraders without failure, little effort had gone into determining how and why the existing detectors fail, and whether anything could be done to prevent those failures. An alternative strategy – the one pursued here – makes progress by identifying and addressing the problems with existing detectors. By specifically addressing the cause of failure, efforts to improve detectors can be more fruitful.

10 Conclusion

In this work, a chronic failure of the naive Bayes masquerade detector – detecting masqueraders who use NBSCs to evade detection – has been explored and mitigated. First, the injurious effect of NBSCs was confirmed in a controlled experiment with synthetic data. Then, the cause of this failure was identified through mathematical analysis. Finally, a strategy for fortifying the detector and preventing the failure was formed and evaluated. This paper shows how weaknesses of classifiers can themselves become vectors for cloaking attacks, and also demonstrates how a fault can be eliminated once its cause is known.

Acknowledgements

This work was supported by the National Science Foundation under grant number CNS-0430474. Thanks are due to Kymie Tan and Tahlia Townsend, as well as to several anonymous referees, for insightful comments. The views and conclusions contained in this chapter are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government, or any other entity.

References

[1] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, Vol. 29, No. 2-3, pages 103-130, November 1997.

[2] W. DuMouchel and M. Schonlau. A fast computer intrusion detection algorithm based on hypothesis testing of command transition probabilities. In R. Agrawal and P. Stolorz, (Eds.), *The Fourth International Conference of Knowledge Discovery and Data Mining (KDD-98)*, pages 189-193, 27-31 August 1998, New York, NY. AAAI Press: Menlo Park, CA.

[3] S. Forrest, S. A. Hofmeyr, A. Somayaji and T. A. Longstaff. A sense of self for Unix processes. In *IEEE Symposium on Security and Privacy*, pages 120-128, 6-8 May 1996, Oakland, CA. IEEE Computer Society Press, Los Alamitos, CA.

[4] T. F. Lunt. A survey of intrusion-detection techniques. *Computers & Security*, Vol. 12, No. 4, pages 405-418, June 1993.

[5] M. V. Mahoney and P. K. Chan. Detecting novel attacks by identifying anomalous network packet headers. Technical Report CS-2001-02, Florida Institute of Technology, Department of Computer Science, October 2001.

[6] R. A. Maxion. Masquerade detection using enriched command lines. In *International Conference on Dependable Systems and Networks (DSN-03)*, pages 5-14, 22-25 June 2003, San Francisco, CA. IEEE Computer Society Press, Los Alamitos, CA.

[7] R. A. Maxion and T. N. Townsend. Masquerade detection using truncated command lines. In *International Conference on Dependable Systems and Networks (DSN-02)*, pages 219-228, 23-26 June 2002, Washington, D.C. IEEE Computer Society Press, Los Alamitos, California.

[8] R. A. Maxion and T. N. Townsend. Masquerade detection augmented with error analysis. *IEEE Transactions on Reliability*, Vol. 53, No. 1, pages 124-147, March 2004.

[9] A. McCallum and K. Nigam. A comparison of event models for naive Bayes text classification. In *Learning for Text Categorization*, pages 41-48, 27 July 1998, Madison, WI, 1998. AAAI Press, Menlo Park, CA. (Papers from the 1998 AAAI Workshop, published as AAAI Technical Report WS-98-05.)

[10] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Boston, 1997.

[11] I. Rish, J. Hellerstein and J. Thathachar. An analysis of data characteristics that affect naive Bayes performance. Technical report RC21993, IBM T.J. Watson Research Center, 30 Saw Mill River Road, Hawthorne, NY 10532, 2001.

[12] M. Schonlau, W. DuMouchel, W.H. Ju, A. F. Karr, M. Theus and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, Vol. 16, No. 1, pages 58-74, February 2001.

[13] S. J. Stolfo, F. Apap, E. Eskin, K. Heller, S. Hershkop, A. Honig and K. Svore. A comparative evaluation of two algorithms for windows registry anomaly detection. Technical report, Columbia University, 23 February 2004.

[14] K.M.C Tan, K. S. Killourhy and R. A. Maxion. “Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits.” In Fifth International Symposium on Recent Advances in Intrusion Detection (RAID-2002), Andreas Wespi, Giovanni Vigna and Luca Deri (Eds.), 16-18 October 2002, Zurich, Switzerland, pp. 54-73. Lecture Notes in Computer Science #2516, Springer-Verlag, Berlin, 2002.

[15] K. M. C. Tan and R. A. Maxion. “Why 6?” Defining the operational limits of stide, an anomaly-based intrusion detector. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 188-201, 12-15 May 2002, Berkeley, CA. IEEE Computer Society Press, Los Alamitos, CA.

Towards a Virtualization-enabled Framework for Information Traceability (VFIT)

Ravi Sahita, Uday Savagaonkar

Communication Technology Lab, Intel Corporation[1]

Abstract Automated and targeted attacks to steal sensitive information from computers are increasing in frequency along with the stealthiness of these attacks. Tools for generating attacks on existing Information Technology infrastructure are readily available. These attacks can easily evade detection from today's countermeasures. Information theft is thus an important threat vector for networked communities where sensitive information is exchanged with partners in different administrative domains, with dissimilar security policies and configurations. The combination of disparately managed networks, ability to store information off-line, and remote access functionality complicate the enforcement of information security policies.

We tackle the issue of protecting sensitive information by applying a system-integrity and information-auditing perspective. We believe this is the first step towards mitigating insider abuse of data-use privileges. We present a Virtualization-enabled Framework for Information Traceability (VFIT) to prevent unauthorized handling of sensitive information. We show that this hardware platform on which information is created, transformed and stored is a key enforcement point to provide accountable information flow. We describe the application of our previous work on Virtualization-enabled Integrity Service (VIS) to implement VFIT. Our approach is data-centric and provides a mechanism that can deterministically audit use of information while it is in use in volatile or non-volatile memory. Using this mechanism, we describe how existing network security mechanisms and our proposed framework can be applied to applications to provide traceability for sensitive information in a distributed system.

[1] © Copyright Intel Corp. All rights reserved. Other Brands and Names are the property of their respective owners.

1. Introduction

The recent Computer Security Institute (CSI) report [1] has indicated an increase in targeted attacks and stealth methods attacks being used for stealing sensitive information. The top 3 causes of financial loss have been identified as the following: 1) financial fraud, 2) loss of information, and 3) insider abuse. In this paper, we focus on the audit (monitoring) and protection of information, with the eventual goal of mitigating insider abuse of information. To be able to successfully audit information handled by existing software, we need to be able to apply access-control policies on existing software. Our approach uses hardware virtualization to interpose memory access control on general purpose OSes and applications. Our goal is to minimize the changes imposed on the application and OS. The usage scenarios that VFIT is useful for are auditing information use on a computer platform and across a computer network.

Our contributions in this paper are the following:

1. 1. A hardware-enforced Information Traceability framework to efficiently enforce access-control for (and thus trace) memory containing sensitive data in a general purpose OS, even if the OS state is not trusted.
2. 2. A hardware-enforced approach to allow the traced information to securely propagate between applications within a virtual machine and in a network of such virtual machines.

We apply our previous work in Virtualization-enabled Integrity Service [2] as a building block in this paper.

The rest of the paper is structured as follows. Section 2 lists our design goals. In section 3, we provide some background for tools used in our paper. Section 4 of this paper covers our system architecture. Section 5 describes an implementation of our framework. Section 6 discusses performance overheads and threats mitigated. Section 7 describes related work in this space. Section 8 presents our conclusions and thoughts on future work.

2. Threat Model and Requirements

Trust means having confidence not just in the people and systems in your organization (insiders) that handle sensitive data, but in anyone you communicate with electronically. Business systems communicating across the Internet is analogous to trusting strangers at a distance. How can companies trust that a partner company is handling information in a safe manner? As transactions increasingly go online, knowing who and what you're interacting with becomes very important. These extended enterprises are insiders from an information perspective. There is no such thing as 100 percent security, but there are some steps companies can take or avoid, to increase the level of trust on information shared across systems – however manually enforcing these steps is not practical. This paper discusses mechanisms to answer the question - is the party I am sharing my data with, able to handle this data in a safe manner? We describe an approach to strengthen monitoring of information usage on a computer, with the ability to enforce information usage if the policy dictates. In essence, this system allows creation of policies regarding how information can and cannot be used in a computer network proposes technology mechanisms that helps enforce those policies.

The threats we want to address via this VFIT are as follows. The primary threat is access of data by malicious or unauthorized code. Unauthorized code is any code module that may not have the necessary credentials or permissions to access data on the same platform. This class of attacks includes automated attacks that can access data from memory or disk. This class of attacks also includes run-time attacks such as via stealth software like root-kits, or via inadvertent access due to software bugs. The second threat is tamper of critical data. This class of attacks attempts to modify data to the attackers liking. For example, malicious browser plug-ins can modify data submitted in a form [3]. The third threat we aim to address is malicious leakage of information. This class of attacks is a subset of unauthorized access, but is listed separately since we want to separately address covert channels and control flow based vectors for leaking information [4]. Specifically, we want to address malicious leakage of information via software attacks. Finally, the fourth threat we want to address is unauthorized transmittal of data. Most applications today are networked and transfer sensitive information, albeit over encrypted sessions. However, encryption is a two-sided blade since malicious entities can also use encryption thereby defeating interposed inspection. This class of attacks transmits data over encrypted or unencrypted channels to remote machines. For example, Haxdoor [5] sends harvested data to remote servers so that attackers can filter data offline.

The design goals for VFIT are the following. The first requirement is for the system to be applicable to off-the-shelf software. Our framework should be able to enforce information tracing on currently available operating systems, applications and hardware, with none to minor modifications of applications, but no operating system and hardware modifications, since these are not under the control of an application programmer. Our second requirement is that the system should impose a low overhead. We note that the overhead of typical information tracing mechanisms will be relative to the operational activity of the application and the workload on the overall system. We want to be able to characterize the CPU utilization of our mechanism and reduce this overhead. Our third requirement is to be able to protect sensitive information versus just detect information leakage. VFIT should go beyond detection, and should be able to protect against information leakage and inadvertent dissemination. In the inadvertent dissemination case, the insider is not malicious; however information misuse is one path by which information may leak. Our fourth requirement is for the system to be flexible enough so that data use is not hindered. The information tracing system should not prevent use and transformation of traced data in an authorized manner and should not require an upheaval of existing infrastructure. Allowing for information to be modified is a key difference of the information tracing problem which differentiates it from other related problems such as Digital Rights Management (DRM) in which the digital content created by the publisher is not allowed to be copied or modified after initial delivery. Our final requirement is to be able to enforce accountability with the information traceability system. The tracing system should ensure that traced data is used only on systems that have the capability to audit the information usage. Additionally, the system should be capable to allow the originator of the data make access decisions on the data. Finally, the system should be able to answer queries such as which user had access to the data.

3. Background

3.1. Models of Policy Enforcement

Existing work on policy enforcement on computer systems has been primarily in the following categories: encryption, access control, interposition, and language-based methods. End-to-end encryption-based methods rely on trust establishment between two parties (local or on a network), using cryptography-based protocols such as IPSec [6] and SSL [7]. These methods assume a secure execution environment for the communicating end-points so that the keys established for secure communication are not exposed. Interposition-based methods rely on inserting an independent inspection layer between two or more interacting entities. For example, most intrusion detection systems hook into the OS system calls thereby interposing their own heuristics engine to analyze system call usage. Such interposition layers can be used to apply rules on the information exchanged between two or more parties. In order for strict policy enforcement, the interposition layer must itself rely on a protected execution environment. Interposition-based methods can be instantiated using various approaches, for example, hooking privilege OS code; executing software on a hardware simulator or in a virtual machine, and operating in a physically separate appliance. All of these interposition methods rely on separation of privilege to protect the interposition layer. Access control enforces other models on files and other objects that an entity may access; however current methods of access-control are not useful against malicious software that is able to circumvent access checks or even worse tamper with access-control lists effectively rendering them insufficient. Language-based methods [8] specify language primitives or extensions to enable verification of information flow when handled by the programs using program annotation, compile-time checks and runtime type safety checks.

3.2. Hardware Virtualization

Virtualization refers to the technique of partitioning a machine into Virtual Machines (VMs). Although its history stretches back decades, virtualization has seen resurgence in interest recently---as exemplified by the VT-x technology from Intel. A hypervisor manages VMs by operating at the highest software privilege level (VMX-root mode or ring-0p in VT-x). A control transfer into the hypervisor is called a VMExit and transfer of control to a VM is called a VMEntry. A VM can explicitly cause a VMExit by using a VMCall instruction (a hypercall). A guest OS runs in VMX-non-root mode which ensures that critical OS operations that can violate the memory separation of the OS or the hypervisor cause a VMExit, which allows the hypervisor to enforce access-control rules. The hypervisor manages launch/shutdown of VMs, memory/device isolation, control register/MSR accesses, interrupts and instruction virtualization. Most importantly, the hypervisor supervises the use of physical memory. The hypervisor achieves this by managing shadow page tables for each VM running on the platform. In hardware VM, the OS is not aware of the hypervisor and hence maintains its own page tables, called the Guest Page Tables (GPTs). The shadow page tables are called the Active Page Tables (APTs) and are used by the processor for address translation. The hypervisor synchronizes APTs with GPTs using a family of algorithms called the Virtual Translation Look-aside Buffer (VTLB) algorithms which emulate the processor TLB. The algorithms leverage hardware virtualization to trap on page-faults and execution of certain instructions such as INVLPG, MOV CR3 that are used by an Operating System to manage virtual memory.

4. System Architecture

The components of VFIT are described in two sections below – Platform architecture and Network architecture. The Platform architecture describes the system to enforce information tracing on a platform, and the Network architecture describes the system to enforce information tracing as it is exchanged over the network. Fig. 1 shows a high-level system view of our architecture. We aim to enforce information traceability across components running within virtual machines, in a distributed system of virtual machines.

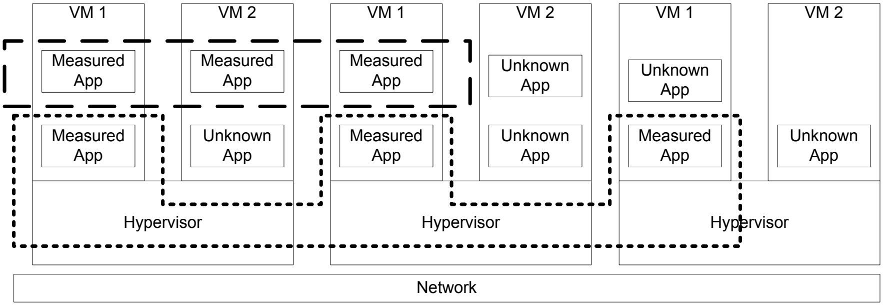


Fig. 1. Information tracing enforcement applied to a distributed system. (Dashed boxes show enforcement boundaries within which information use can be monitored or protected)

4.1. Platform Architecture

The platform architecture has three layers. The application layer is the privilege level at which the OS and applications execute. The intent of this work is to use as much existing software as possible in this layer. The next layer is the Information Traceability layer - it acts as a middleware that enforces information tracing by executing at an elevated privilege level (than the application layer). The platform creates and manages meta-data for the data that is being traced at this level. The hardware layer is the bottom layer of the platform and is used to enforce the access-control as setup by the Information Traceability layer. In this section, we describe the four components of the Information Traceability layer in more detail:

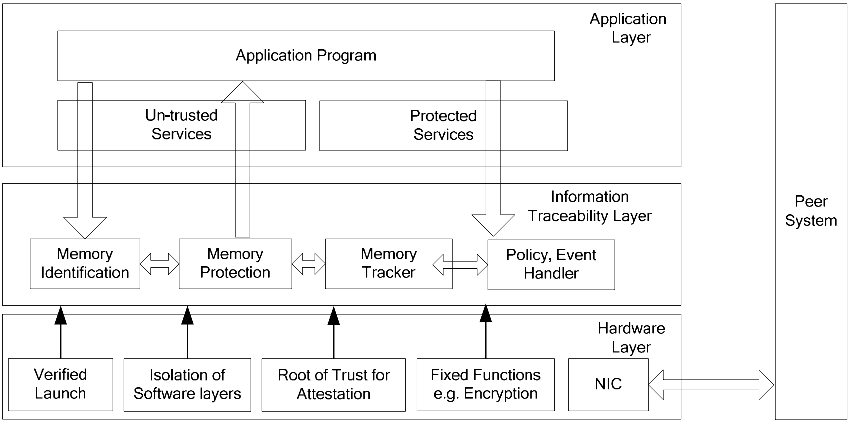


Fig. 2. Information Traceability Platform Architecture

The Memory Identification component is used to locate and verify data integrity in physical memory before data tracing is applied to this memory area. This component also performs sanity checking before admitting data memory into the tracing database---for example, checking that during initialization there is a single virtual memory mapping of the identified physical memory on the system. The Memory Protection component enforces the access control on the identified memory. The model we enforce is to limit access to the traced memory by programs that are identified based on their source and integrity. By source, we mean the software should be cryptographically associated with a software vendor and by Integrity, we mean the runtime memory image must not be tampered with. By virtue of using the program's image in memory as the only credential required at runtime, we can ensure the traced data memory is kept inaccessible to other unknown programs executing on that system. This separation of access is enforced by hardware and is event driven that reduces the interaction with the Memory Protection component only in cases when the policy setup is violated. The Event Handler component handles access events as they occur due to the protection setup by the Memory Protection component. This event handler identifies the context in which the access trap occurs, and performs the following activities. The Event Handler first redirects the access to a ghost memory area if the access is to be disallowed. If the access is to be allowed, the Event Handler drives the Memory protection component so that either the new page is also tracked from this point on or an existing traced page is removed from the tracking database. In this case, appropriate meta-data is updated in the tracking database. The Memory Tracker component ensures that protected data memory is appropriately associated with meta-data and on access events evaluates this meta-data on access events. This component is responsible to propagate meta-data to peer platforms with which the protected data is shared.

4.2. Network Architecture

For a networked application, data is typically moved from I/O channels into system buffers and then application memory. As the data is used it may also be copied into the application's sub-components. Any resultant/output data follows the reverse path into system service buffers and finally into I/O buffers before it is serviced. The goals of the networking subsystem of VFIT are the following. First, traced data should not be transmittable by unknown code on the platform. This is to protect against transmittal on I/O channels that are not under the purview of the Information Traceability Layer. Second, traced data transmitted (via the permitted I/O channels) from one system to another should be accessible on the remote system only if the data can be traced on that system. An example of valid transmittal events that are detected at the network I/O level is when a system library references protected data in memory to transfer contents to a packet buffer for the network driver. An example of an invalid transmittal event is a case where malicious software snoops data from the system I/O buffers to transmit over an un-monitored USB network driver.

We achieve our first goal using the platform architecture to ensure that traced data is accessible only to code that can prove its source and runtime image integrity. To achieve our second goal, the application communicating traced data to another application must associate the expected credentials with the data so that the credentials can be checked on the remote platform that receives the data. Hardware support for measured launch using the TPM provides the base to ensure that the platform can be attested by a trusted verifier to be running the right software. We use a hardware root of trust for measurement (RTM) and a hardware root of trust for reporting to encrypt data with an asymmetric key-pair accessible only to the Information Traceability Layer. We assume that the Information Traceability layer is provisioned with a certificate. The certificate is CA-signed public portion of a TPM-derived Aik. The TPM PCR values contain the measurement of the hypervisor performed by the CPU during launch [9]. TPM quotes can be used to cross-attest that the correct Information Traceability Layer is executing on the communicating platforms using TPM-based attestation protocols such as those documented by the Trusted Computing Group (TCG) [10].

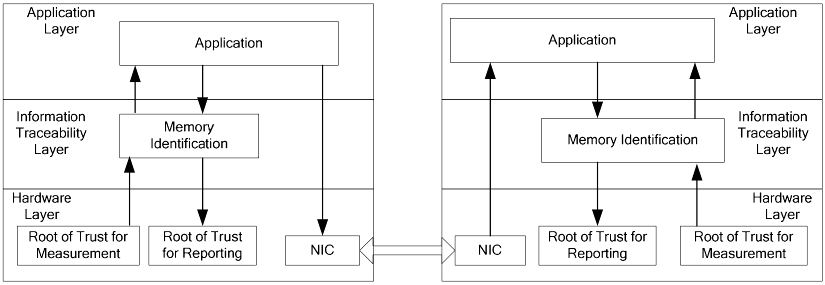

Application Layer
Application
Information Traceability Layer
Memory Identification
Hardware Layer
Root of Trust for Measurement
Root of Trust for Reporting
NIC
NIC
Application
Application Layer
Memory Identification
Information Traceability Layer
Hardware Layer
Root of Trust for Reporting
Root of Trust for Measurement

Fig. 3. Information Traceability Network Architecture

Once the two platforms have authenticated and attested that they are running the measured Information Traceability Layer, the communicating platforms can transfer the traced data over an encrypted channel. Note that the traced data exchanged is transferred with a cryptographic hash of the Integrity Manifest of the program that has the permission to access the traced data. If the program (identified by the Integrity Manifest) has been registered and protected on the destination platform, the Traceability layer will deposit the decrypted data in its memory and continue the trace process on that platform.

5. Implementation

We use hardware virtualization [11] to enforce memory access control onto software running in the Application layer (including OS in our definition). In our implementation, the Information Traceability layer is enforced by executing it in a hypervisor which runs at a higher privileged ring (ring 0p) than the OS it virtualizes. For completeness, we provide a brief overview of our previous work on Virtualization-enabled Integrity Services (VIS) [2]. We assume that the hypervisor is a small body of code that is measured on launch thereby reducing the attack surface. We further note that there is hardware support [9] to completely measure this layer of code during platform boot and record the measurements in tamper resistant hardware [10] for subsequent reporting.

5.1. Virtualization-enabled Information Tracing

We describe the implementation of our Information Traceability layer with reference to our previous work on Virtualization-enabled Integrity Services (VIS) [2]. VIS is used to protect software components from runtime tamper, and API circumvention attacks. VIS comprises of three components. The VIS Registration Module (VRM) is used to provide a hyper-call interface to software programs running in the guest OS. The Integrity Measurement Module (IMM) performs the Memory Identification function. A program's code/data is verified at runtime based on a signed Integrity Manifest which contains cryptographic hashes (measurements) of the program's expected state. The IMM also accounts for relocation that the OS loader performs to evaluate the expected state of the program in memory. The Memory Protection Module (MPM) component implements the Memory Protection function. It isolates a program's code and data from other programs. A program in the guest OS requests protection via a registration hypercall to the VRM, which uses the APT to identify the physical page locations for the program in memory and uses the IMM to verify the integrity of the contents of these pages. If the program passes integrity verification, the VRM uses the MPM to overlay access-control on the program memory using page-table partitioning. This process is described in more detail in a specific usage scenario below. Lastly, the Kernel Directory Service component is itself protected and is then used to enumerate the loaded OS services. This service identifies each OS service by name and also provides the address range in which the service is loaded. This service allows the Memory Tracker function implemented in the hypervisor to correlate access events with kernel services to make an access-control decision to allow or redirect access. Fig. 4 shows these components pictorially.

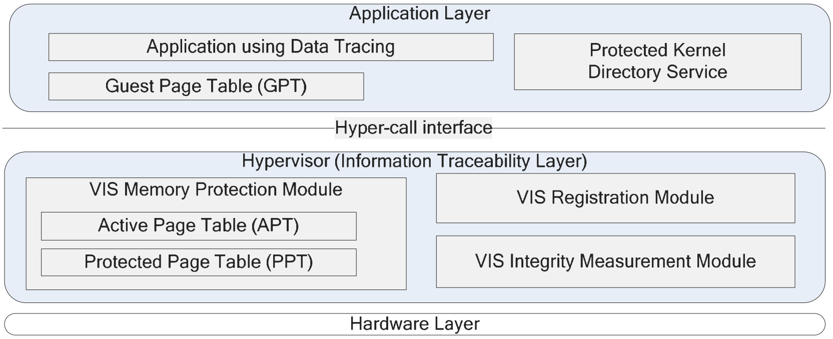

Application Layer
Application using Data Tracing
Guest Page Table (GPT)
Protected Kernel Directory Service
Hyper-call interface
Hypervisor (Information Traceability Layer)
VIS Memory Protection Module
Active Page Table (APT)
Protected Page Table (PPT)
VIS Registration Module
VIS Integrity Measurement Module
Hardware Layer

Fig. 4. VIS usage for Information Tracing

We apply VIS to implement the Information Tracking layer. We use the following application scenario to describe the operational aspects. We assume a networked application using Information Tracing is interacting with a remote peer application. For the purposes of this discussion, we assume that the information to be traced is explicitly identified by the application. This approach can also be applied to trace keyboard/network data.

Data to be traced is identified by the application or by system policy. This identification is simply provided via hyper-calls. Additionally to ensure the data pages are not shared with other applications, we require minimal modifications of the application to use pragma statements to align its data sections on page boundaries, and allocate extra memory for dynamic allocations. For applications that do not provide aligned memory, additional meta-data is captured in the shadow page table structures. An application that requests tracing for a data set must provide the virtual address for the data it requests tracing for. Additionally, the application must provide the manifests for the programs it expects to access this data. The application may optionally also provide an integrity measurement for the data area in memory. This is an optional step since the data area may be dynamically allocated memory and hence may not have a static integrity measurement associated with it. If integrity measurement is requested, the Memory Identification component verifies that the data being traced matches the signed cryptographic hash provided. The Memory Identification component locates the physical memory pages that contain the data to be traced.

Measured Data Pages are isolated from the application and operating system. The Memory Protection component marks the data pages in the shadowed Active Page Tables as not-present and references the physical pages from the Protected Page Table. The Memory Protection component also records the offset and length of the data areas to be protected in the protected physical pages in a Tracker Page Table. This setup is shown in Fig. 5.

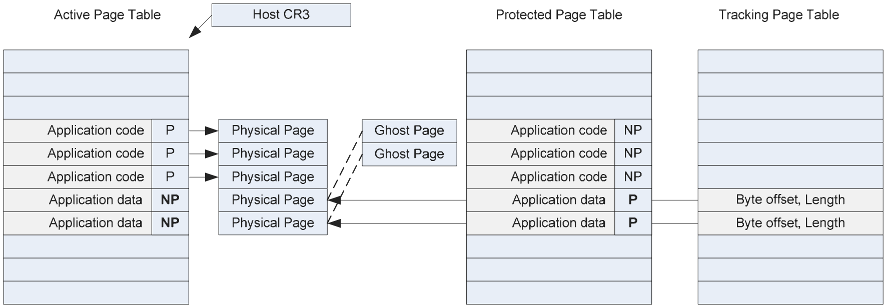


Fig. 5. Memory Protection setup for tracked data

As applications access the data, a page fault event transfers control via hardware into the hypervisor. The hypervisor uses the processor context and using this context queries the Kernel Directory Service to get additional information about the program/service that is accessing the traced data. A program may optionally itself provide an Integrity Manifest that identifies the program. The Integrity Manifest is signed by the vendor to ensure authenticity. The program image can optionally be used to generate an Integrity Manifest to measure the program in memory. After the program has been measured, program code pages can also be moved into the Protected Page Table to protect the integrity of the program and to ensure no future page-fault events occur. The hypervisor references the Protected Page table in the host CR3 and resumes execution in the measured application so it can access the traced data.

If the protected code tries to move traced data into un-measured pages of memory (non-present pages in the PPT), those accesses will also be visible as page-fault events to the hypervisor, and thus can protect against programming errors or inadvertent leaks. When the program completes operating on the traced data, the program's pages are removed from the PPT on unload from the APT as part of the VTLB (shadowing) algorithm. The sequence of events for enforcing or monitoring data tracing between applications is described in Fig. 6.

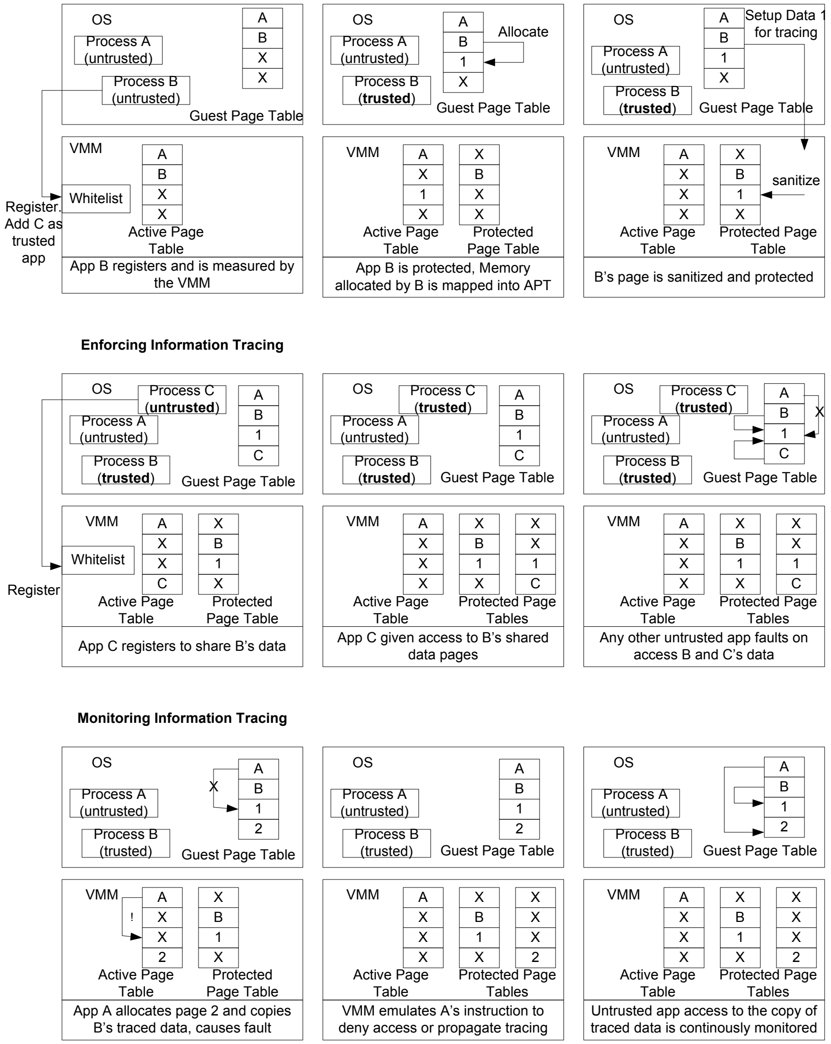

OS
Process A (untrusted)
Process B (untrusted)
Guest Page Table
A
B
X
X
VMM
Whitelist
Register. Add C as trusted app
Active Page Table
App B registers and is measured by the VMM
Process B (trusted)
Allocate
1
Protected Page Table
App B is protected, Memory allocated by B is mapped into APT
Setup Data 1 for tracing
sanitize
B's page is sanitized and protected
Enforcing Information Tracing
Process C (untrusted)
Process C (trusted)
C
Register
App C registers to share B's data
Protected Page Tables
App C given access to B's shared data pages
Any other untrusted app faults on access B and C's data
Monitoring Information Tracing
2
!
App A allocates page 2 and copies B's traced data, causes fault
VMM emulates A's instruction to deny access or propagate tracing
Untrusted app access to the copy of traced data is continously monitored

Fig. 6. Information Trace operation – Monitoring and Enforcement

6. Analysis

As described in the implementation, we describe how data tracing can be enforced on existing software with minimal modifications and general purpose Operating Systems using existing hardware. Our approach however introduces some limitations in the data usage as described in our model of memory protection. We have applied VFIT to an existing operating system (Windows XP with Service Pack 2) and several system applications and drivers (ring-0 intermediate network driver, ring-3 application and sub DLLs). The hypervisor is developed in-house and provides VIS functionality. We had to make minor modifications to the programs we used in our implementation, but we did not change the base Operating System or hardware. The expected performance overhead of our system is discussed below for the network driver. We are able to use VFIT to protect information that is handled by trusted code and prevent accesses by un-trusted or unknown code. The framework is flexible to allow applications to build a system where the restrictions can be adjusted as required.

6.1. Performance Discussion

The performance overhead of our system is relative to the activity of the application that requests tracing of data, and protection of code that accesses the traced data. Each initial access to the protected data results in a hypervisor round trip[2]. We qualify this access as initial, since depending on the requirement of the application, a page may be mapped into the protected page table after the initial access, thereby eliminating subsequent faults. A minimal hypervisor round trip on an Intel® Core™ 2 Duo for a page-fault VMExit, where a VM state read and write are involved, costs 2219 cycles [12]. These round trips must be minimized in order to reduce the performance overhead of the application that is using the proteced data. As the hypervisor changes CR3 to reference different shadow page tables, the TLB is destroyed on each transition. We used micro-benchmarks to quantify the effects due to TLB shoot-down effects observed when the page-table root to be 8 cycles/TLB entry. With the use of address space identifiers (or tagged TLBs), this impact will be reduced.

In our previous paper [2], we reported the measured performance overhead of protecting an intermediate network driver using VIS. We described two methods using software and hardware-assist to reduce the overhead of VIS for instruction page-faults to 4% CPU utilization. However, the expected CPU utilization when used as described in this paper for data tracing is expected to be higher, since in this case, we expect to perform emulation of some instructions which access traced data pages to add other data pages into the protected page tables. Using similar heuristics of sharing these traced data pages between verified components, we can reduce the impact of these additional faults. In our future work, we aim to analyze this performance overhead for commercial applications using VFIT.

[2] VMExit, VMEntry pair due to the page fault event

6.2. Threat Mitigation

We described a framework and implementation to associate protected data to measured code and disallow access to unmeasured code. This addresses automated attacks that can access data from memory. We also can associate integrity measurements with data once it is protected and traced thereby addressing the class of attacks that attempt to modify data to the attackers liking. We reduce the threat vectors for malicious leakage by disallowing access to un-trusted code. This does not address all methods of leaking information but addresses malicious leakage of information via software attacks. We disallow unauthorized transmittal of data unless the communicating parties can prove using trusted hardware that the required enforcement layers are active on the destination platform.

We show that with minimal program modifications we can address data-flow and control-flow oriented leakage by restricting access only to measured (and isolated) programs. Additionally, data handling by admitted programs is also contained in a hypervisor controlled memory view thereby preventing against misplacing data where un-trusted code can access it. A measured hypervisor can also provide additional services for secure handling of the traced data once it is protected, for example, where the hypervisor ensures clearing out protected memory pages when they are returned to the APT. Additionally, the hypervisor can leverage device virtualization hardware to protect against DMA attacks.

7. Related Work

Mandatory Access Control (MAC) was prescribed by the U.S. Department of Defense "orange book" [13] for systems handling classified information. According to this definition, the Trusted Computing Base (TCB) enforces a mandatory access control policy over all subjects based on sensitivity labels. The security level of the label is then used by the TCB to enforce operations on the objects. The TCB authenticates and checks authorization for the user. In our approach, the hardware platform, the hypervisor and the set of measured applications handling the data are in the TCB. The operating system is explicitly not part of the TCB given the number of attacks available against general operating systems.

The GIFT (General dynamic Information Flow Tracking) framework [14] for distributed applications describes language extensions for C programs to allow application developers to associate application-specific tags with input data and instrument the application to propagate tags to all the other data that are control or data-dependent on the GIFT-instrumented programs. Language oriented methods require programmers to create proxy code to handle information flow policies at run-time. In contrast, we are trying to address malicious information flow beyond a programs scope – such behavior is not specified at all and hence is not captured in the program. Additionally, we are focusing on existing applications which do not use these custom languages since they have not become as ubiquitous as C, C++ and Java. Java does provide type safety via byte verification, and confidentiality for the data by executing the program in a sandbox from other programs, however, it is not restrictive enough in a malicious environment. Finally, language based methods do not provide a system-wide capability to protect data once it leaves the language runtime.

AEGIS [15] is a single-chip processor system to address physical and software attacks. AEGIS assumes that all components external to the processor, such as memory, are un-trusted. This is in contrast with our approach where we cannot address physical attacks on memory with the current implementation. AEGIS provides a tamper-evident, authenticated environment for software, which is similar to our approach but with a larger TCB instead of processor changes, that includes the hypervisor. XOM [16] specifies architecture for execute-only memory, and uses a modified processor and a XOM Virtual Machine Monitor to create compartments where programs can execute in isolation. Contents of memory stored in external memory are encrypted as in AEGIS.

The ReVirt system [17] performs logging under the operating system by moving the logging functionality into UMLinux [18]. This allows ReVirt to replay the system's execution before, during, and after an intruder compromises the system. ReVirt logs enough information to replay a long-term execution of the virtual machine instruction-by-instruction. We utilize a similar approach, with the distinction that our approach is used to track machine behavior only when memory that is protected by our system is accessed by software that is un-protected in the virtual machine, thereby reducing the overhead of inspection.

D'Cunha's thesis report [19] proposes hardware enhancement to the processor memory management unit (MMU) thus enabling it to detect when memory containing trusted code or data is being maliciously modified at run-time. This approach selectively marks memory as immutable to harden the Trusted Platform Module (TPM) drivers. This work also implements a software prototype using the Xen hypervisor. Our work can be seen as an extension to this software implementation; we allow a programmer to request confidentiality for the data or code pages once they have been measured and verified by our system. This property is critical for protecting sensitive data from leak.

sHype [20] is a Xen-based Mandatory Access Control architecture, consisting of a policy manager that maintains the security policy, an access control module (ACM) implemented in the hypervisor that makes authorization decisions according to the policy; and mediation hooks controlling access of VMs to shared virtual resources based on decisions returned by the ACM. This is a class interposition model where the hypervisor is used to access-control resources it manages. Our work is similar to sHype, and it extends it by allowing access-control over regions of memory within the virtual machine in the context that the traced memory is being used by the virtual machine. This allows a richer set of policies that can be enforced by the hypervisor ACM. sHype was also applied for distributed MAC with Shamon [21] which allows creation of MAC VM coalitions. With our approach, we get similar benefits of supporting similar coalitions for access control, and allowing for finer granular enforcement of the MAC policy to a subset of components that can be executing within separate virtual machines across a distributed system.

Chow *et al.* [22] presents a framework for understanding data lifetime on a system by adding a taint framework to the BOCHS simulator. Simulator based approaches result on large overheads to the operation of the machine that runs on top of the simulator. Kong *et al.* [23] describe a Linux-based architecture that uses capability-based data access control to ensure that intermediaries that want to access sensitive information have the appropriate credentials to be able to access that information, creating a Protected Data Path (PDP) through a network of systems. . The fundamental limitation of this work is that it treats the entire Operating System to be a trusted component. However, considering the size and complexity of modern operating systems [24], such an assumption is not reasonable. Another difference in our approach is that we rely on the integrity of the executing program that is requesting access to the data instead of the possession of credentials to enforce data access. PDP also proposes the use of virtual machines as a means of separating the trusted domains as future work.

Ho *et al.* [25] demonstrate a demand emulation model using virtualization for taint-based protection using Xen. Our approach is a refinement of that approach in that it further reduces the amount of emulation that the hypervisor has to perform. In the demand emulation method, their shadow fault handler is called whenever a page fault is received from the protected VM, where the fault handler is responsible testing if the faulting page is marked as tainted, and if so, flags that the VM should be restarted in emulation mode when it returns from the fault. In our case, we do not require special handling of faults when the traced page (in our terminology) is being accessed by code pages that have been allowed by policy to access that data. We achieve this by shadow page table partitioning into protected page tables as described in our implementation. Another distinction is that unlike Ho *et al.*, we do not have to trust the initial state of the machine, and assume that the system could a priori contain malicious code that an attacker can exploit. We achieve this using a model for integrity verification and runtime protection of the code that handles the traced data. The other distinction in our method is the further reduction in the amount of time that the instructions need to be emulated.

8. Conclusion

We have demonstrated the feasibility of overlaying memory access control to protect memory in our previous work on Virtualization-enabled Integrity Services. In this report, we describe a framework to support Information Traceability in applications that handle sensitive data, without major modifications to the applications. We also discuss how this framework can be used for securely tracing information exchanged across a network of such systems. We have applied this system to sample applications on general purpose operating systems to be able to monitor information usage. CSO's can apply this technology to monitor and assuredly audit sensitive information and trace it across networks, without requiring changes on all of their existing applications and network infrastructure. In our future work, we plan to apply this framework to specific applications such as web servers and further quantify the performance of this system to protect against software-based threats causing information theft, thus mitigating a threat vector of the Insider attack on information. We also plan to work on the expression of policies for such information tracing systems to allow CSO's to be able to express policies over data classes handled within and across computer networks.

Acknowledgments

The authors acknowledge Michael Covington, David Durham and Prashant Dewan for their comments and feedback on various facets of this work; and Toby Kohlenberg for providing insight in this area of research. Thanks to the organizers of the 2007 ARO/FSTC Workshop on Insider Attack and Cyber Security. Also, sincere thanks go to the anonymous reviewers who provided constructive feedback that greatly improved the quality of this report.

References

[1] Computer Security Institute: Computer Crime and Security Survey 2007. http://www.gocsi.com/forms/csi_survey.jhtml

[2] Sahita, R., Savagaonkar, U.R., Dewan, P., Durham, D.: Mitigating the Lying-Endpoint Problem in Virtualized Network Access Frameworks. In: Lecture Notes in Computer Science, Virtualization of Networks and Services. Volume 4785. (2007) 135-146

[3] McAfee: Trojan – Formspy. http://vil.nai.com/vil/content/v_140256.htm

[4] Vachharajani, N., et al.: RIFLE: An Architectural Framework for User-Centric Information-Flow Security. In: 37th IEEE/ACM International Symposium on Microarchitecture. (2004)

[5] F-Secure: Virus Information Pages: Haxdoor. http://www.f-secure.com/v-descs/haxdoor.shtml

[6] Kent, S., Atkinson, R.: Security Architecture for the Internet Protocol. IETF RFC 2401 (November 1998)

[7] Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol. IETF RFC 4346 (April 2006)

[8] Sabelfeld, A., Myers, A.C.: Language-Based Information-Flow Security. In: IEEE Journal on Selected Areas in Communications. Volume 21. No. 1. (January 2003)

[9] Intel Corporation: Intel® Trusted Execution Technology Preliminary Architecture Specification. http://www.intel.com/technology/security/

[10] Trusted Computing Group: Trusted Platform Module (TPM) Specifications. https://www.trustedcomputinggroup.org/specs/TPM

[11] Intel Corporation: IA-32 Intel® Architecture Software Developers Manual. http://www.intel.com/products/processor/manuals/index.htm

[12] Adams, K., Agesen, O.: A Comparison of Software and Hardware Techniques for x86 Virtualization. In: Proc. 12th Architectural Support for Programming Languages and Operating Systems. (2006)

[13] U.S. Department of Defense: Trusted Computer System Evaluation Criteria. DoD 5200.28-STD. (December l985)

[14] Lam, L. C., Chiueh, T.: A General Dynamic Information Flow Tracking Framework for Security Applications. In: Proceedings of the 22nd Annual Computer Security Applications Conference on Annual Computer Security Applications Conference (ACSAC). IEEE Computer Society (December 2006)

[15] Suh, G. E., Clarke, D., Gassend, B., van Dijk, M., Devadas, S.: AEGIS: architecture for tamper-evident and tamper-resistant processing. In: Proceedings of the 17th Annual international Conference on Supercomputing (ICS '03). ACM. (June 2003)

[16] Boneh, D., Lie, D., Lincoln, P., Mitchell, J., Mitchell, M.: Hardware Support for Tamper-Resistant and Copy-Resistant Software. Technical Report. UMI Order Number: CS-TN-00-97., Stanford University. (2000)

[17] Dunlap, G. W., King, S. T., Cinar, S., Basrai, M. A., Chen, P. M.: ReVirt: enabling intrusion analysis through virtual-machine logging and replay. In: ACM SIGOPS Operating Systems Review. (Dec. 2002)

[18] Sieh, V. Buchacker, K.: Testing the Fault-Tolerance of Networked Systems. In: International Conference on Architecture of Computing Systems (ARCS). Workshop Proceedings (2002)

[19] D'Cunha, N.: Exploring the Integration of Memory Management and Trusted Computing. Dartmouth Computer Science Technical Report TR2007-594. (May 2007)

[20] Sailer, R., Valdez, E., Jaeger, T., Perez, R., van Doorn, L., Griffin, J. L., Berger, S.: sHype: Secure Hypervisor Approach to Trusted Virtualized Systems. IBM Research Report RC23511. (February 2005)

[21] McCune, J. M., Jaeger, T., Berger, S., Caceres, R., Sailer, R.: Shamon: A System for Distributed Mandatory Access Control. In: Proceedings of the 22nd Annual Computer Security

Applications Conference on Annual Computer Security Applications Conference. ACSAC. IEEE Computer Society (December 2006)

[22] Chow, J., Pfaff, B., Garfinkel, T., Christopher, K., Rosenblum, M.: Understanding Data Lifetime via Whole System Simulation. In: Proceedings of the 13th USENIX Security Symposium. (August 2004)

[23] Kong, J., Schwan, K., Widener, P.: Protected Data Paths: Delivering Sensitive Data via Untrusted Proxies. In: Proceedings of the 2006 International Conference on Privacy, Security and Trust (PST 2006). Ontario, Canada. (October 2006)

[24] Wikipedia: Source lines of code. http://en.wikipedia.org/wiki/Source_lines_of_code

[25] Ho, A., Fetterman, M., Clark, C., Warfield, A., Hand, S: Practical taint-based protection using demand emulation. In: Proceedings of the ACM Sigops/Eurosys European Conference on Computer Systems (EuroSys '06) ACM (April 2006)

Reconfigurable Tamper-resistant Hardware Support Against Insider Threats: The Trusted ILLIAC Approach

Ravishankar K. Iyer, Paul Dabrowski, Nithin Nakka, Zbigniew Kalbarczyk

Coordinated Science Laboratory, University of Illinois at Urbana-Champagne

Abstract "An insider attack, sometimes referred to as an inside job, is defined as a crime perpetrated by, or with the help of, a person working for or trusted by the victim" [1]. This one-sided relationship of trust makes the insider attacks particularly insidious and difficult to protect against. This article motivates the need for secure and tamper-resistant storage of the secret information that is impenetrable even by the operating system and efficient ways of meeting this need. It highlights innovative new work being developed in the context of the Trusted ILLIAC project at the University of Illinois. A progression of techniques is presented providing increasing levels of security starting from a purely software-based approach, to hardware/software partitioned and hardware-only mechanisms. This is to guard the system effectively against insiders having increasing levels of intrusive access from user-level, administrative up to even physical access to the system under threat of attack. Techniques covered include software- and hardware-based memory randomization, hardware for a threshold cryptography enabled mechanism to allow tamper-proof key management and support the software technique. Further, we describe an Information Flow Signatures based technique to provide runtime data integrity guarantees. Reconfigurable hardware is used to ensure the secure computation of critical data. In order to enable this trusted computing hardware we explore requirements for securely initializing it under the threat of an insider attack. The unique advantage of a hardware implemented mechanism is that the secret, either the key or the code that operates on security-critical data, cannot be revealed or modified even by the operating system.

1 Introduction

Since the very first IT survey on cyber-attacks, one fact has remained almost constant; a greater percentage of attacks appear to come from the inside (from "trusted" folks), than from the outside (the "untrusted" folks). In the past 3 years about 56% of organizations, on an average, had from 1 to 10 insider attacks [2]. When this type of security breach only impacted a specific firm it was acceptable to maintain silence. But with electronic commerce and the Internet recasting the way we do business the nature of the game has changed dramatically. Over the past year, millions of consumers have been exposed to potential identity theft in major breaches at a variety of organization, e.g., banks and insurance agencies. The rate at which confidential information is being exposed has forced governments all over the world to initiate regulations concerning the security of confidential information held by businesses, forcing organizations to reevaluate their "trusted" vs. "untrusted" environments. In this scenario it is essential that the security perimeter of any organization be protected not only from the outside but also from the inside.

In this article we motivate the need for secure and tamper-resistant storage of the secret information, such as critical data, code and cryptographic keys, inaccessible even through the operating system and efficient ways of meeting this need. A typical insider has at least user-level and possibly administrative level and physical access to the system under threat. To effectively foil insider attacks at all levels, critical data must be secured from unauthorized modifications even by an administrator, who has privileged operating system-level access. We consider insiders with varying degrees of access of increasing levels of intrusion into the system. In this article we summarize the techniques developed to protect against insider attacks in the context of the Trusted ILLIAC project at the University of Illinois [3, 18]. We discuss a novel approach to program information about security-critical data and the mechanisms for runtime protection of this data into a field-programmable gate array (FPGA). The FPGA is configured at application initialization, making it almost impossible even for the operating system to modify it in an unauthorized manner. The set of techniques discussed in this chapter are summarized in Table 1.

We start with a description of a software approach to randomize memory layouts of a process, called Transparent Runtime Randomization, to foil attacks from insiders with user-level access. This is improved by performing the randomization completely in hardware so as to protect the key from even an administrator-level insider. Following this we present a hardware mechanism to support threshold cryptography through a secure and tamper-resistant key store, sharing keys across multiple nodes. To provide the user the ability to delineate specific data structures of his application as security-critical and protect them against any malicious attacks we present an Information Flow Signatures for data integrity. This technique utilizes hardware support to enforce the "trusted"ness of code executing on behalf of the application and we explore requirements for securely initializing it under the threat of an insider attack.

Insider Attack	Possible Consequence	Countermeasure
User-level attack using input parameters to application	Attacker may gain privileged access to the system	Transparent runtime randomization (TRR) in software. (Section 2)
Information disclosure through operating system level access	Attacker can access application level randomization keys to foil TRR	Tamper-resistant hardware key-store to protect disclosure of secret key information. (Section 3)
Access to application specific security-critical data	Attacker can modify application-specific critical data such as password authentication etc. to gain unauthorized access.	Information flow signature checking to protect security-critical data. Criticality information (e.g., location of passwords, authentication code etc.) stored within application binary. (Section 4)
Modifying application binary to tamper security-critical data	Attacker can make security critical code as non-critical or mark his own malicious code as trusted	FPGA-based tamper-resistant storage of application-specific security critical data (Section 5).

Table 1. Summary of proposed protection techniques against insider attacks

2 Software-based Transparent Runtime Randomization

An insider with user-level can craft inputs to programs to corrupt memory locations in a way that transfers control to his malicious code. This section discusses *Transparent Runtime Randomization* (TRR) a technique to foil and detect such attacks, generally termed as Unauthorized Control Information Tampering (UCIT) attacks. TRR is a generalized approach that dynamically and randomly relocates a program's stack, heap, shared libraries, and parts of its runtime control data structures inside the application memory address space.

Making a program's memory layout different each time it runs foils the attacker's assumptions about the memory layout of the vulnerable program and makes the determination of critical address values difficult if not impossible. An incorrect address value for a critical memory element causes the target application to crash. At a software-level, TRR is implemented by modifying the dynamic program loader, therefore, it is *transparent* to the application programs, i.e., existing applications run without any modification or recompilation [1].

TRR has been implemented on Linux/x86 platforms. It is shown, using published attacks, to be effective, not only against well-studied attacks such as stack buffer overflow and format string, but also against attacks such as `malloc`-based heap overflow, integer overflow, and double-free.

Principle of Operation. The locations of critical program elements that an attacker needs to determine for launching a successful attack usually reside in well defined memory regions in the process address space. We distinguish two types of memory regions: *position independent* and *position dependent*. A memory region is position independent if it can be freely placed in virtual memory at application startup time without breaking[1] the application, i.e., there are no inherent complex relationships with other parts of a program with respect to positioning. A memory region is position dependent if relocating it at application startup time could cause a chain of broken references from either the program code or data. A process's stack, heap, and shared libraries are position independent, while the global offset table is position dependent. The following explains the position dependency nature of these basic data regions.

- User stack: Before an application process begins to execute, the operating system kernel sets up the user stack and stores information such as environmental variables and command line arguments on the stack. The kernel then sets up the stack pointer (on Linux/x86, it is the *esp* register). The application program accesses data on the user stack through the stack pointer plus an offset. The program works correctly as long as the stack pointer is correctly initialized; hence the stack is position independent.
- Shared libraries: Shared libraries, also known as dynamically linked libraries, are compiled as Position Independent Code (PIC). The library functions are invoked by the program using base register plus offset and can be loaded anywhere in a program's address space as long as the base register is set up correctly.
- User heap: Heap is managed by dynamic memory management functions such as `malloc`() and `free`(). At runtime, `malloc`() determines the beginning of the heap via the `brk`() system call. A program accesses the heap using pointers to memory regions allocated by `malloc`(), hence the program does not make any assumptions on the runtime location of the heap.
- Global offset table (GOT): Once a program is compiled, the GOT is fixed at a location inside the program's static data segment. Any uncoordinated relocation of GOT will break the program, since part of the program code (the procedural linkage table or PLT) directly references GOT. As a result, GOT is position dependent, and relocation of it requires corresponding changes in the referencing code in the PLT as well.

TRR randomly relocates both position independent and position dependent regions by modifying the dynamic program loader. While relocation of position independent regions is relative simple, relocation of position dependent regions poses a challenge to TRR. The potential hardware architecture for TRR is outlined in [1]. In a recent study it was shown that it is possible to extract the ran-

[1] *Breaking* a program means to either cause it to crash or to output incorrect results.

domization key of TRR by a brute force attack that tries to enumerate all possible randomizations for the stack locations and engineer an attack. This may be practical in a 32-bit processing system in which only the lower 16 bits of the address (referred to as contributing to the entropy in the address) vary and hence within a short time it is possible to cycle through all address values. The task becomes much more difficult in a 64-bit system with 40 bits of entropy. A point to note is that on every unsuccessful attack TRR crashes the system under attack and when the application is re-launched the memory segments are re-randomized. So, the attacker must cycle through all the 2^{16} possibilities again. A system that is being constantly attacked (and hence, crashed and rebooted) by such a brute force attack would very soon come to the attention of an administrator [6].

The insider with a user-level access cannot determine the key used for the randomization of the different data segments. With a root-level privilege however careful runtime monitoring can reveal the key. The key can be used to modify the attack parameters making the application as vulnerable as without TRR. This motivates the need for a securely storing the randomization key. The next section addresses this challenge in the context of a Digital Certification System for storing cryptographic keys, although the approach can be generalized to securely storing other kinds of secret information.

3 Tamper-resistant Key-store Support for Threshold Cryptography

Cryptography-based security applications are faced with a similar problem of the secret, the encryption and/or decryption key(s), being potentially exposed to malicious insiders. This section presents a mechanism to store the key in a secure and tamper-resistant key-store in hardware. In addition, threshold cryptography is used to increase the difficulty of intrusion from breaking into one single node to breaking into a majority of participating nodes. The operation of the tamper-resistant key-store is demonstrated on a multithreaded Attribute Authority, a digital certification system that handles the issue, revocation and query of Attribute Certificates [5].

The hardware crypto-engine integrates, in a single chip, a large number of RSA Processors to accelerate computationally expensive RSA operations, and a *tamper-resistant KeyStore* to preserve (shares of) secret keys. The integration is done seamlessly with threshold cryptography support by using Shoup's algorithm [9]. The implementation is based on FPGA technology. The crypto-engine architecture is general and can serve a broader range of security applications (e.g., SSL connection establishment, elliptic curve operations). Indeed, the internal design implements generic functions that any cryptographic coprocessor is likely to require (e.g., dispatching operation requests to multiple functional units, loading secret keys in the device).

3.1 Crypto-engine Architecture

The crypto-engine architecture (depicted in Fig. 1) is centered on a *Main Controller* component, which supervises the operation of the other crypto-engine components (e.g., RSA Processors and KeyStore) and the communication with the host system (i.e., the computer system hosting the crypto-engine).

RSA Processor. An RSA Processor decomposes modular exponentiation, involved in an RSA operation, into a series of modular multiplications and squares, which are efficiently computed employing the Montgomery algorithm [9]. The RSA Processor operates serially on words of *S* bits, where *S* is a design parameter that can be chosen from among the values 32, 64, 128, and 256. This approach (1) makes the processor design modular and scalable with the length of the RSA modulus and exponents (thus enabling the use of threshold cryptography) and (2) allows trading off performance versus area occupation, which can be used when dimensioning the crypto-engine. An early RSA Processor was introduced in [10]. Major extensions were necessary to integrate it into the full crypto-engine architecture and to manage long RSA exponents as required by the threshold cryptography algorithm adopted [8].

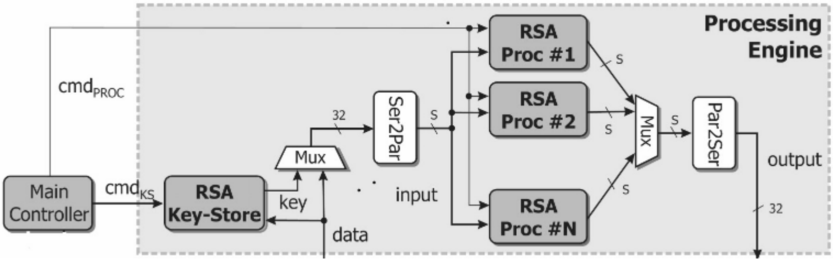


Fig. 1. SA Processor Engines and Key-Store

KeyStore. The KeyStore component provides tamper-resistant storage and fast access to a number of RSA keys used by the RSA Processors. Each KeyStore entry includes a modulus *N*, a generic exponent *Exp*, and a factor *W*. Exponent *Exp* can be the public exponent *E*, the private exponent *D* of a standard RSA key, or a secret key share S_i [8]. Factor *W* depends only on *N* and is used to convert the input data in the residue representation needed by the Montgomery algorithm [9]. The KeyStore can accommodate multiple key entries, and this feature can be used when signatures with multiple RSA keys are required (e.g., a single Attribute Authority can use different RSA key pairs for issuing attribute certificates with different policies [7]).

3.2 Security Analysis

The security analysis considers attackers whose ultimate goal is to forge Attribute Authority signatures. To accomplish this goal, attackers need to obtain the Attribute Authority's private key and/or to take control of the Certificate Engine system, without being detected. Threshold cryptography guarantees that the private key cannot be reconstructed if fewer than half of the private key shares are disclosed.

The remainder of this section focuses on how the crypto-engine approach makes the attack substantially more difficult than on a system implemented entirely in software. An attacker can succeed by hardware-level intrusion if he/she has physical access to the replica node.

Hardware-Level Intrusion. Our analysis is based on the attack categories identified in [11]: physical attack, read-back attack, and side-channel attack. A *physical attack* aims at uncovering the FPGA design by opening up the FPGA package and probing (undocumented) points inside the chip without damaging the device. Due to increasing FPGA complexity, this attack can be achieved only with advanced inspection methods (e.g., Focused Ion Beam), which are quite costly and are probably possible only for large organizations (e.g., intelligence services).

A *read-back attack* accesses/reads the FPGA configuration file from the FPGA chip (using the read-back functionality generally available on the FPGA device for debugging purposes), after which the attacker reverse-engineers the obtained bit-stream. To prevent the read-back attack, most manufacturers provide the option of disabling the read-back functionality. Moreover, even though theoretically possible to interpret and/or to modify the bit-stream of an FPGA, major vendors (e.g., Xilinx, Actel) maintain that it is virtually impossible. The irregular row and column pattern of the hierarchical interconnection network exacerbates the inherent complexity of the reverse-engineering process [12].

A *side-channel attack* exploits unintentional information leakage sources (e.g., power consumption, timing behavior, electromagnetic radiations) in the implementation. At present, little work has investigated the feasibility of such attacks against FPGAs. Nevertheless, attacks using power consumption and specific to RSA are known in the literature. For instance, Simple Power Analysis and Differential Power Analysis exploit the fact that a straightforward implementation of the Right-to-Left Binary Algorithm (widely used in RSA hardware circuits, including our RSA Processor) has power consumption that changes in time with the bit-sequence of the RSA key (thus, monitoring the FPGA power consumption allows discovering the RSA key). Simple countermeasures can be found in [13. In our case, power attacks are more difficult to launch, since multiple RSA Processors operate concurrently and asynchronously, effectively masking the information that can be revealed by the overall FPGA power consumption.

We note that hardware-implemented cryptographic co-processor engines limit the types of secure computations that the user can perform to only the implemented cryptographic routines. In the next section the Trusted Computing Base is extended to support execution and data integrity monitoring of programmer-identified security-critical portions of the code through Information Flow Signature checking.

4 Information Flow Signature Checking for Data Integrity

We now present a technique, information flow signature checking [29], to protect application critical data from unintended modifications, even by the operating system or an insider with root privileges, and describe its implementation. We define unintended modifications as those that represent a violation of the intended behavior of a program defined by its source code. For example, this includes the operating system or root user trying to write directly to a memory location that contains critical data, or malicious modification of a pointer in a program, causing it to modify an object other than the one intended by the source code.

The application developer wishing to employ this technique defines which data is critical, such as a database or structure containing login information in a server application. Using the precise pointer analysis of the IMPACT compiler [27], the automated technique extracts the backward computation tree [25] of instructions which are allowed by the source code of the application to directly or indirectly modify critical program data[2]. This backwards tree of dependencies is encoded in the form of a signature, referred to as the *Information-flow signature (IFS)*. The signature is enforced during application execution using a combination of software and programmable hardware. Since the analysis is based on the properties of the program according to its source code, hence there are no false-positives, and valid code is never rejected[3]. The proposed technique is based on the well-known observation that the possibility of unintended modifications to data arises from the 'gap' between an application's source code and how it is executed during runtime. *Information flow signature checks enforce the source-level semantics of memory accesses at runtime thereby closing this gap for critical data.*

Using hardware support during runtime, the technique ensures the integrity of the protected data even if other data in the application is compromised or controlled by the attacker. As a result, the IFS technique can be applied selectively to protect critical data in the application with significantly lower overheads compared to other techniques for memory safety [20, 22, 23] which take an 'all-or-nothing' approach and are unable to provide the same guarantees when applied selectively to the program. The IFS technique ensures that data compromised by unintended data modifications cannot propagate to critical data. This is crucial for fast recovery from attacks.

[2] We use the term data to refer to both variables as well as memory objects in the program.

[3] Some applications may write past the end of an object into another memory object during correct execution. We do not consider such programs.

4.1 Threat Model

The aim of the technique is to preserve data integrity rather than its confidentiality: hence, the technique does not address side-channel attacks [24]. The threat model assumes that the attacker can execute arbitrary code, as an insider might be able to, and overwrite program variables stored in memory and processor registers as long as the malicious memory accesses are performed through the processor. For example, malicious DMA transfers are not covered by the threat model for this technique. Specifically, an attacker could use an IEEE 1394 interface port to initiate transfers to main memory of a system which are not visible to the processor [26]. We also assume that the initialization of the technique is performed correctly. *Later in this paper we explore the required mechanisms to ensure both a trusted initialization path and the mitigation physical attacks and attacks which bypass the processor of the system.* If initialized properly, the technique is immune to attacks on the operating system after program loading is completed.

Examples of attacks covered in the threat model include:

- Classical memory corruption attacks such as buffer overflow, format string, and heap corruption attacks that overwrite non-control data in the application. These attacks violate the source-level semantics of the program and are caught by the technique. We assume that other techniques such as control-flow integrity [21] or program shepherding [19] have been deployed to protect control-data in the application.
- Software-based insider attacks, in which the attacker attempts to alter (at runtime) part of the program to gain control over the critical data. An example of this class of attacks is a malicious plug-in for a web-browser that tries to modify sensitive data in the browser in violation of its interface with the browser. The attack will be detected even if the plug-in's code is unavailable at compile-time as the checks are in the browser's source code.

4.2 Approach

Attacks on data integrity may be performed by causing an instruction to write to an address outside the bounds of the instruction's valid destination object, or in the case of an attack by an insider with root privileges, may simply be writing over critical data or data which will be used indirectly to compute critical data. Therefore, they can be prevented by checking the bounds of every write to memory, and determining if the instruction is allowed (as defined by the source code) to write to the object in a given memory location. However, performing bounds checks on every write is prohibitively expensive - more so in the case of non type-safe[4] languages such as C/C++, where it is very hard to statically determine the targets of memory reads and writes. We observe that checking every write is excessive when a user is only interested in protecting certain critical data. The goal of our technique is to check a minimal number of instructions while ensuring that the critical data is not corrupted.

Since we are applying protection selectively, it is insufficient to check only the direct writes to critical data. This is because while protecting direct writes makes it difficult for the attacker, a smart attacker can still influence the value of a critical data indirectly as shown below:

Suppose program variable *c* is calculated by adding two other variables, *a* and *b*. Assume that *c* is part of the critical data in this application. An attacker can influence the value of *c* indirectly by corrupting either *a* or *b*. Thus, in order to protect critical data from corruption, we ensure that all data and instructions that the critical data is dependent upon are also protected. This constitutes the backward dependence tree [25] of the critical data. In general, the backward tree contains the set of instructions and data objects that both directly and indirectly influence the critical data.

The protection scheme consists of two phases: A) a compile-time phase to extract the backward tree of critical data in the program, and B) a runtime-phase to check if the critical data is influenced in violation of the statically derived backward tree. The runtime phase is implemented using a combination of software and hardware.

A. Compile-time Phase, a compiler-based static program analysis determines the following:

1. The instructions that can influence the *critical* data (according to program semantics)
2. For each instruction in (1), the set of objects (data) that the instruction is allowed to write to

The compiler marks each instruction in (1) and each object in (2) as *trusted*. This *trusted* property is propagated at runtime according to propagation rules which have been mathematically proven to ensure that any instruction which may have been compromise is marked as un-trusted, and thus can never write to critical data.

B. Runtime Phase, The following invariants are enforced by a combination of hardware and software at runtime.

1. **Level 1:** *Critical* data is modified only by *trusted* instructions and objects (*every instruction* running on the processor is checked and this invariant is enforced in hardware), which includes instructions from

[4] Even in type-safe languages, one still needs to perform checking at runtime as attacks are performed by distorting the behavior of instructions at runtime.

applications other than the one being protected (i.e., an insider with root privileges is not allowed to write directly to critical data, regardless of which application is used to write to this data)

2. **Level 2:** Each *trusted* instruction writes only to its statically allowed objects (enforced in software)

If either property is violated, an interrupt is triggered which halts execution of the program and raises a security alert before *critical* data is corrupted. This is vital to ensure fast application recovery, from a checkpoint for example (we do not consider recovery here).

4.3 Implementation

The technique is implemented using a combination of hardware and software. The hardware implementation requires the addition of a special CHK instruction, which is used to initialize the hardware checks, to the instruction set of the processor, but no other direct modifications to the pipeline of the processor. The processor's caches are not modified. Instead a content addressable memory outside of the processor's pipeline, which we have named the Critical Data Trusted Instruction (CDTI) store, is used to store the information flow signature for critical data. All of the hardware checks are performed by the IFS checking module, implemented outside of the processor's pipeline. This module has a read-only interface to internal signals of the pipeline including: 1) register file control, 2) current instruction and its pointer, 3) pipeline stall, flush, and 4) cache control. A prototype has been demonstrated using a modified Leon 3 processor [28] and the IFS checking module synthesized to an FPGA board. Overheads of the checking module are approximately 4.4% in gate count, and 1% in clock frequency in comparison to the un-modified processor.

Initialization and System Assumptions: High-level communication used to initialize the checking module with information about which instructions are trusted and which data is critical occurs through CHK instructions, an extension to the instruction set architecture to enable such communication between the application and the hardware. The backward dependence tree and information about critical data is loaded from an instrumented program at load time by using these CHK instructions, which we assume for now to be un-corrupted by an attacker. *We explore mechanisms to ensure that CHK instructions and the program binary have not been tampered with in Section 5.* Once the hardware checker has been initialized, no writes to critical data will be allowed unless they are through a valid path in the protected application. This prevents data corruption from an insider with root privileges and the operating system. |

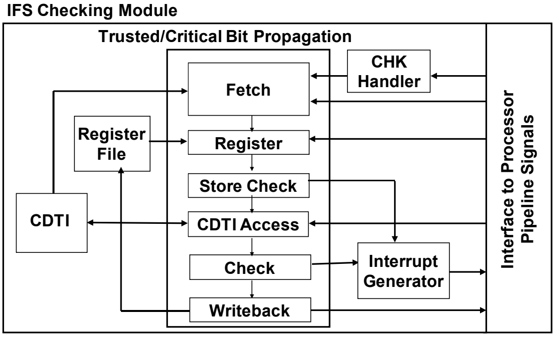


Fig. 2. Schematic of hardware checking module

Example of Operation: The operation of the checking module in Figure 2 is as follows:

1. During program initialization, CHK instructions read from the main processor pipeline enter the *CHK handler* within the module and are used to initialize the CDTI.
2. During runtime, the *fetch stage* looks up each instruction's "trustedness" within the CDTI based on the program counter value read from the main processor pipeline.
3. Trusted instructions have their operands looked up in the *register stage* of the module.
4. The *store check* stage of module enforces Level 1 checking rules for store instructions, before they enter the memory stage of the processor. (e.g. if a trusted store instruction uses non-critical operands, the checking module raises an alarm before the memory operation occurs)
5. *CDTI access* looks up the information flow signature in the CDTI using cache control signals from the processor.
6. In the *check* stage, trusted instruction operands are checked and the destination of un-trusted instructions is checked to make sure they do not write to critical data.
7. In the *writeback stage*, trusted bit information is propagated back to the register file.

5 System Architecture Including the Trusted Computing Engine

We consider a system architecture for the trusted initialization of hardware-based security checks, such as those based on Information Flow Signatures, and the runtime guarantees required to be resistant to insider attacks. This architecture ensures that an application instrumented with initialization data for the IFS checks cannot be corrupted without detection. Furthermore, we ensure that regardless of the privilege level of a user, the secure coprocessor equipped with Information Flow Signature checks, for example, will not run IFS-protected binaries unless they are the product of a trusted application developer. Further guarantees which ensure the trusted initialization and resistance to insider attacks are listed in the following subsections.

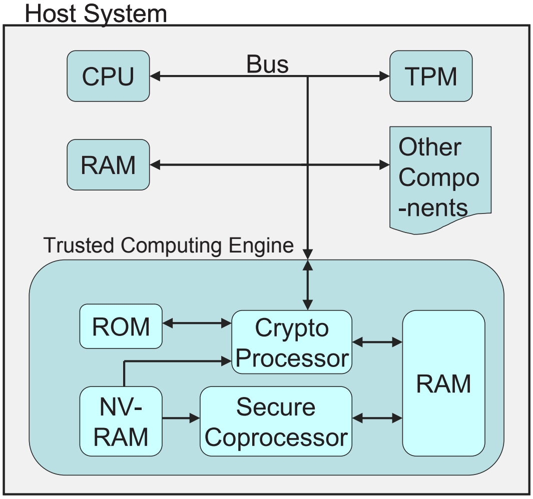


Fig. 3. Host System with the Trusted Computing Engine (TCE)

The basic system architecture is depicted in **Error! Reference source not found.** and includes a "host system" which is made up of typical, off-the-shelf components and a Trusted Computing Engine (TCE), which is a physically tamper-resistant expansion card on a high-speed bus (such as PCI-Express) that will run security-critical applications. Such tamper-resistant enclosures can be manufactured to be highly resistant to physical attacks [15].

The basic components of the TCE include a RAM, a tamper-resistant ROM containing a public RSA key, and Non-Volatile memory used to store FPGA configurations. The secure coprocessor, an FPGA programmed to contain a soft-processor augmented with Information Flow Signatures security checking hardware, and the crypto processor, an FPGA programmed with cryptographic functionality, perform the computational work of the TCE. The functionality of the TCE system is drastically different than that of the Trusted Platform Module (TPM). The TPM does not execute applications and is limited to the roles of attestation, key management, and basic cryptographic capabilities. On the other hand, the TCE provides secure initialization of applications, and employs runtime security techniques to protect applications running within the secure coprocessor.

During initialization of the system, the hardware within the TCE is automatically programmed by the NVRAM. During runtime, the secure coprocessor awaits for a security-critical application to be verified and loaded into the TCE RAM by the crypto processor. The crypto processor also facilitates any transactions between the secure coprocessor and the host system. When a security-critical application is run on the TCE, the local TCE RAM (not accessible directly by the host system CPU) is used by the application. In case of size limitations on the internal RAM, the crypto processor may be used to encrypt and swap memory out to the host system RAM.

5.1 Protecting Against Insider Attack With User-level Privileges: Runtime Guarantees

Critical data integrity is maintained by the Information Flow Signatures-enabled secure coprocessor as long as the Hardware-Based Security Checks (HBSC) are (i) initialized as intended, (ii) not re-configured during runtime, and (iii) are not subverted to gain access to critical data. Further, applications should have the ability to attest to the existence of the TCE during runtime. For example, this allows a client application to require a server application to be executing on a TCE before connecting, thus ensuring that an un-tampered binary with data integrity guarantees is running on the server.

In this scenario we assume that: (i) the user has only remote access to the system through remote login and (ii) the HBSC are correctly initialized. Based on these assumptions we focus only on providing guarantees during runtime operation. The following section delves into the issues associated with trusted initialization of the HBSC. We now describe how the TCE with an IFS-enabled secure coprocessor enforces runtime guarantees.

- Subverting the Security Technique and Hardware. The Information Flow Signatures do not have direct read or write access to critical data. Hence, it is not possible to subvert the hardware mechanism to overwrite or retrieve critical data. Techniques other than Information Flow Signatures may have to consider this threat.
- HBSC Re-Configuration. In order to protect against updating and modification of the security hardware configuration during runtime, it is possible to use a ratchet lock similar to that in the IBM 4758 secure coprocessor [15] to prevent access to configuration data after the initial configuration of the hardware. Essentially, a special command can be issued to the hardware security module that prevents further configuration.
- Security Hardware Attestation. The Trusted Computing Group's (TCG) Trusted Platform Module (TPM) is designed with the ability to perform hardware platform measurements [16]. The TPM's abilities can be leveraged by the security hardware in order to provide remote attestation capabilities to the hardware.

5.2 Protecting Against Insider Attack with Administrative Privileges: Initialization and Runtime Guarantees

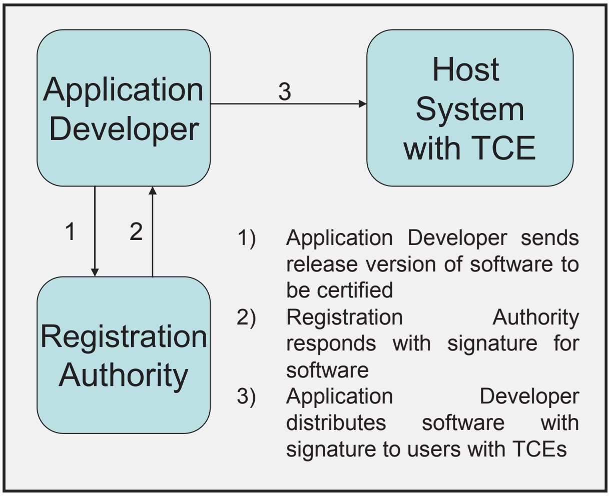
Protection against attacks by those with administrative privileges on a computer system is a difficult undertaking, especially since an administrator may have access to the physical hardware of the computer system, and may be able to modify the operating system and other binaries before execution. This section explores a system architecture to give the required capabilities to remedy such a threat with the goal of critical data *integrity* in mind, though this system can also be applicable to enforcing critical data *confidentiality*. The architecture focuses on providing the following mechanisms for trusted initialization of HBSC:

- Verify the authenticity of the HBSC initialization sequence
- Ensure that correct HBSC initialization data is loaded when an application requires it
- If HBSC initialization does not occur, proper action is taken to maintain data integrity

If these objectives for the trusted initialization of the HBSC are met, the previously defined criteria for runtime protection of the security technique can be used to ensure the proper runtime operation of security checks. However, because of the nature of the new threat model, the system should provide method to verify to a 3rd party that the HBSC initialization sequence was executed properly. Thus, further requirements for hardware attestation are made.

Authenticity of HBSC Initialization Data. To verify that the initialization data for security-checking hardware has not been corrupted or tampered with, the cryptographic RSA signing operation is used on the data contained within the initialization portion of the application to be protected by the security technique. In order for this to be a valid method of verifying the authenticity of the initialization data, we must consider the following: who is doing the signing, and how is the public key of this entity known in order to verify the signature?

The signing is performed by a centralized registration authority (RA), which could be the manufacturer of the security hardware. *Note: A registration authority, in the context of this chapter, has a specific and different meaning than in the typical Public Key Infrastructure.* It is possible to have the public key of the RA stored in a read-only, tamper-proof memory programmed by the manufacturer. The crypto-processor uses this key to verify the signature of the initialization data. This requires every application which utilizes the security technique to be signed by the RA. The secure coprocessor contains hardware, similar to the RSA Processor described in [14], to perform verification of the security check initialization data before loading it.



1) Application Developer sends release version of software to be certified
2) Registration Authority responds with signature for software
3) Application Developer distributes software with signature to users with TCEs

Fig. 4. Software Distribution Process

Correct HBSC Initialization Data. In order to verify that the correct initialization data for a given program has been given to the secure coprocessor, the RA signs a hash of the entire software application, including the initialization portion. This has the downside of being a significant computation burden to verify, giving further justification for the crypto-processor.

5.2.1 Maintain Data Integrity in Case of Initialization Failure

In the case that an application binary has been tampered with, the crypto-processor does not load the application into the memory space of the secure coprocessor, and the critical data to be used by the application is not initialized in the TCE.

Attestation. Similar to the previous attestation scenario, the TPM may be leveraged to provide measurements. In this case, the crypto-processor within the TCE initiates an integrity measurement which reflects the initialized status of the TCE.

5.2.2 Utilizing an Architecture Designed to Prevent Insider Attacks

Now that we have a notion of what must be done in order to guarantee the initialization and correct runtime operation of a hardware-based security technique, we may consider how a system might be utilized to take advantage of the capabilities.

The applications which are run on the secure coprocessor have their critical data protected against unintended critical data modifications and other attacks which involve exploiting program semantics that are normally not enforced during runtime [18]. However, this technique does not cover the possibility of using a valid program path to influence critical data – hence it may still be possible for the malicious administrator to corrupt program input, for example network traffic. The applications themselves must be constructed in such a way that these threats do not expose the potential for insider attacks through valid program paths.

Applications running on the secure coprocessor that use network access to gain access to data which is to be protected from insider attack must implement authentication mechanisms which take advantage of the given hardware attestation features. Such authentication mechanisms can ensure that "counterfeit" applications (which have the same functionality as the original program but may have been modified by a malicious attacker) cannot gain access to critical resources which are being protected from insider attack (see [17] for example protocols).

6 Conclusions and Future Directions

This article describes a secure and tamper-resistant mechanism for protecting secret information. This is to guard the system effectively against insiders regardless of their access level, user, administrative or physical. We first present a purely software-based mechanism to randomize memory the layout of a process leading to a hardware implementation of the same. A threshold cryptography-based approach that uses a tamper-resistant key-store is described to ensure secure access to the secret key. In order to extend the trusted computing base to execute security critical portions of the code, as designated by the programmer, we describe a system architecture which uses low-power, low-cost reprogrammable hardware to provide support for the initialization of security techniques. This hardware approach prevents any unauthorized modifications of security critical data or code, even by the operating system, thus providing effective protection against all levels of insider attacks. The architecture is based on specific considerations about the protections required for proper initialization and runtime execution of the Information Flow Signatures hardware-based checks, and provides new avenues for research. Possibilities for future work include leveraging multi-core architectures to design a TCE and using the reprogrammable nature of FPGAs to allow for secure hardware upgrades to the TCE.

References

[1] N. Einwechter. “The Enemy Inside the Gates: Preventing and Detecting Insider Attacks,” http://www.securityfocus.com/infocus/1546

[2] L. A. Gordon, M. P. Loeb, W. Lucyshyn and R. Richardson. “2005 CSI/FBI Computer Crime and Security Survey.” http://americas.utimaco.com/encryption/fbi_csi_2005_p1.html.

[3] Trusted ILLIAC: a large, demonstrably trusted cluster computing platform. http://www.iti.uiuc.edu/t-illiac/index.html

[4] J. Xu, Z. Kalbarczyk, and R. K. Iyer. “Transparent runtime randomization for security,” In *22nd International Symposium on Reliable Distributed Systems (SRDS'03)*, pages 260--269, Florence, Italy, Oct. 2003. IEEE Computer Society, IEEE Press.

[5] Saggese, G. P., Basile, C., Romano, L., Kalbarczyk, Z., and Iyer, R. K. 2004. “Hardware Support for High Performance, Intrusion- and Fault-Tolerant Systems,” In *Proceedings of the 23rd IEEE international Symposium on Reliable Distributed Systems (Srds'04)* - Volume 00 (October 18 - 20, 2004). SRDS. IEEE Computer Society, Washington, DC, 195-204.

[6] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, Dan Boneh. “On the effectiveness of address-space randomization,” *ACM Conference on Computer and Communications Security* 2004: 298-307

[7] S. Farrell and R. Housley, “An internet attribute certificate profile for authorization,” IETF — RFC 3281, 2002.

[8] V. Shoup, “Practical threshold signatures,” *LNCS*, vol. 1807, pp. 207– 218, 2000.

[9] P. L. Montgomery, “Modular multiplication without trial division,” *Math. of Computation*, vol. 44, no. 170, pp. 519–521, 1985

[10] Mazzeo, et al., “An FPGA-based implementation of the RSA algorithm,” in Proc. *of DATE*, 2003.

[11] T. Wollinger, J. Guajardo, and C. Paar, “Cryptography on FPGAs: State of the art implementations and attacks,” *ACM Trans. on Embedded Computing Systems*, 2003.

[12] Xilinx, “Configuration issues: Power-up, volatility, security, battery back-up,” Application Note XAPP 092, 1997.

[13] M. Joye, “Recovering lost efficiency of exponentiation algorithms on smart cards,” *Electronics Letters*, vol. 38, no. 19, pp. 1095–1097, 2002.

[14] Ross Anderson, Mike Bond, Jolyon Clulow and Sergei Skorobogatov. “Cryptographic Processors – A Survey.” University of Cambridge Technical Report: ISSN 1476-2986. 2005.

[15] Joan G. Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendertvan Doorn, Sean W. Smith, Steve Weingart. “Building the IBM 4758 Secure Coprocessor,” *IEEE Computer*, October 2001.

[16] “TPM Main Part 1 Design Principles Specification Version 1.2.” Level 2 Revision 103. 9 July 2007

[17] Sean W. Smith. “Outbound Authentication for Programmable Secure Coprocessors,” *Proceedings of the 7th European Symposium on Research in Computer Security*, 2001.

[18] R. K. Iyer, Z. Kalbarczyk, K. Pattabiraman, W. Healey, W. M. Hwu, P. Klemperer, R. Farivar. ”Toward Application-Aware Security and Reliability,” *IEEE Security & Privacy*, Volume 5 Number 1, Jan 2007.

[19] Kiriansky, V., Bruening, D., and Amarasinghe, S. “Secure execution via program shepherding,” In *Proc. of the 11th USENIX Security Symposium* (Aug. 2002).

[20] Miguel Castro, Manuel Costa, and Tim Harris. “Securing software by enforcing data-flow integrity,” In *Symposium on Operating System Design and Implementation (OSDI)*, Seattle, WA, Nov. 2006.

[21] Abadi, M., Budiu, M., Erlingsson, U., and Ligatti, J. “Control-flow Integrity: Principles, implementations, and applications,” In *Proc. ACM Computer and Communications Security*, Nov. 2005.

[22] Necula, G. C., McPeak, S., and Weimer, W. 2002. “CCured: type-safe retrofitting of legacy code,” In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Pro-*

gramming Languages (Portland, Oregon, January 16 - 18, 2002). POPL '02. ACM Press, New York, NY, 128-139.

[23] Dhurjati, D., Kowshik, S., and Adve, V., "SAFECode: enforcing alias analysis for weakly typed languages," In *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '06)*. ACM Press, PP. 144-157.

[24] Boneh, D., DeMillo, R. A., & Lipton, R. J. "On the Importance of Eliminating Errors in Cryptographic Computations," Journal of Cryptology: *The Journal of the International Association for Cryptologic Research*, vol. 14, pp. 101-119, 2001.

[25] Mark Weiser. "Program slicing," In Proceedings of the 5th International Conference on Software Engineering, pages 439-449. IEEE Computer Society Press, 1981.

[26] Adam Boileau. "Hit by a Bus: Physical Access Attacks with Firewire," Presented at *Ruxcon 2k6*, 2006.

[27] UIUC Open-IMPACT Effort. "The OpenIMPACT IA-64 Compiler." http://gelato.uiuc.edu

[28] Jiri Gaisler, Gaisler Research. "Leon 3 Synthesizable Processor." http://www.gaisler.com

[29] W. Healey, K. Pattabiraman, S. Ryoo, P. Dabrowski, WM. Hwu, Z. Kalbarczyk, R. K. Iyer. "Ensuring Critical Data Integrity via Information Flow Signatures," University of Illinois Technical Report. UILU-ENG-07-2216. CRHC-07-09.

Surviving Insider Attacks: A Call for System Experiments

Virgil D. Gligor[1] and C. Sekar Chandersekaran[2]

[1]Carnegie Mellon University, [2]Institute for Defense Analyses

Abstract The handling of insider attacks is a significant technical challenge as little assurance theory and design practice exists to guide the design of effective, credible countermeasures for large systems and applications. Much of the relevant theory has focused on insider attacks on individual security protocols and small-scale applications. In this position paper, we suggest that confidence in a system's resilience to insider attacks can emerge by the application of well-accepted survivability principles and design methods. We caution, however, that different trade-offs emerge in applying these principles to practical designs, thereby requiring a careful balance among the costs of countering insider attacks, recovery from attack, and attack deterrence, and between the fine granularity of access permissions and ability to administer these permissions is a safe manner. In view of the dearth of practical solutions for surviving insider attacks in any significant-size system, we suggest that experiments in applying well-accepted principles and design methods to critical subsystems (e.g., user authentication, DNS) are necessary to provide effective and quantifiable assurances.

1 Introduction

Insiders are generally understood to be trusted individuals or entities authorized to access and manage system and application resources. By virtue of their largely unrestricted access, insiders can launch extremely potent attacks against information systems. These attacks take advantage of the insider's authority and knowledge to configure and administer systems and the ability to exploit known, and perhaps deliberately induced, vulnerabilities in a largely unconstrained networked environment; e.g., in the Internet.

Anecdotal evidence over the past twenty years identified insiders as the most potent source of attacks in computer systems and networks in terms of the loss incurred as a consequence of these attacks. For example, in the summer of 1987, in a presentation given to a National Research Council panel studying the information security posture of the Department of Energy, Bob Courtney[1] suggested that,

[1] At the time of his presentation, the late Robert Courtney was an industry security consultant and a former manager for data security in IBM's Systems Develpment Division. In 1993, he re-

in non-government domains (e.g., business, industry, commerce), losses caused by insider attacks amounted to 19% of all losses, where 6% was attributable to disgruntled employees and 13% to dishonest ones. These losses trailed only those caused by administrative error (65%), and far exceeded those caused by infrastructure damage due to floods, fires, and earthquakes (8%), outsider attacks (2%), and unspecified other events (6%). Whether all Courtney's statistics are relevant today can be debated: the Internet has enabled outsiders' attacks on unprotected or poorly protected systems, causing substantial more losses than those of the centralized systems of the 70s and 80s. However, two facts are not debatable: (1) the handling of insider attacks in large-scale systems and enterprises remains a significant technical challenge, and (2) little assurance theory and design practice exists to provide guidance on designing effective, credible countermeasures for such systems. Much of the relevant theory has focused on insider attacks on individual protocols (e.g., applications of threshold cryptography [3, 4, 12] and password-based authenticated key exchange protocols [10, 11], and various corruption models [8]). Hence, the design of system-level countermeasures are likely to rely, at least for the foreseeable future, on the application of well-understood survivability principles to specific system designs to derive architectural structures that satisfy operational, cost, and usability constraints. These constraints require tradeoffs among different interpretations of survivability principles and help optimize these interpretations. Of course, the benefits of using survivability principles would have to be illustrated in realistic examples of specific subsystems and/or applications that survive insider attacks in a demonstrable way.

A useful application of survivability principles, which would blend good theory with design practice, would be a user authentication subsystem. Such a subsystem should be resilient to attacks by systems administrators and operators who are free to act simultaneously as insiders and outsiders. We suggest this application area for three reasons.

First, authentication subsystems are critical to the security of any system, yet they are extremely vulnerable to simple insider attacks. For example, if an administrator simply makes a file storing one-way encrypted passwords readable, no significant harm is supposed to be possible, and yet this practice would enable off-line dictionary attacks and lead to wholesale theft of passwords and corresponding identities.

Second, authentication subsystems offer a good case study on how operational, cost, and usability constraints enable tradeoffs among competing interpretations of survivability principles. For example, partitioning sensitive data among several diverse, redundant servers managed by administrators whose duties are separated to a very fine granularity may be an implementation of sound interpretation of survivability principles. However, they may impose substantial recurrent opera-

ceived the National Computer Systems Security Award given by the National Institute for Standards and Technology and the National Security Agency.

tional costs (e.g., personnel costs) and usability costs (e.g., physical separation and security may require significant and costly space upgrades).

Third, but by no means least significant, is the fact that cryptographic protocol theory provides resiliency techniques; viz., protocols for password-based authenticated key exchange [10,11]. Hence, one need not be sidetracked by the need to develop new theory, and focus exclusively on architectural design of the authentication subsystem. Other equally compelling examples of critical infrastructure subsystems that could benefit from the application of survivability principles include the Domain Name System and distributed directory services and protocols (e.g., Microsoft's Active Directory, or LDAP).

In this position paper, we suggest that confidence in a system's resilience to insider attacks can emerge by the application of well-accepted principles and design methods. In our view, developing new theories for the design of countermeasures to insider attacks in significant-size systems is both premature and possibly counterproductive. Only after the application of well-known and understood design principles and methods that withstood the test of time becomes manifestly inadequate, should one attempt to develop new theory. Lack of experimentation in this area indicates that development of new theory is premature at this time. Further, the transfer of useful research results to practice would be delayed by the natural resistance to the use of untried and untested theories in practice. Instead, designs of countermeasures to insider attacks should rely on well accepted methods and tools in system security and reliability, as well as in cryptography, to illustrate how resilient architectural structures emerge from the application of known survivability principles. We believe that, if the aim of such designs is to show how confidence in system resilience to insider attacks can emerge, then only well accepted principles, design methods, and tools that stood the test of time can be used.

2 Principles for Survivability

A significant number of basic security and dependability principles have been enunciated in the past; e.g., the work of Saltzer and Schroeder [17], Avizienis, Laprie, Randell, Landwehr [1,2], and Neumann [15]. Of these, we present five that (1) are applicable to the system architecture level, and (2) are supported by practical design methods and tools.

The security and dependability principles we have selected for discussion are:

1. Avoidance of a single point of failure
2. Independence of Failure Modes and Attack Vulnerabilities
3. Fast Recovery from Failure and Attacks
4. Attack Deterence
5. Least Privilege Authorization

Other important principles, such as defense-in-depth, are important, but their application to countering insider attacks is less compelling. Our thesis is that only by applying selected survivability principles together with well-accepted security, reliability and cryptographic techniques can confidence in system-architecture resiliency to insider threats emerge.

2.1 Avoidance of a Single Point of Failure

The common characteristic of single points of failure is that an attack by either an insider or an outsider that exploits a single vulnerability can lead to a system compromise. Typical architecture features used in fault tolerance, namely spatial separation and replication of system components, *offer insufficient protection against insider attacks*. For example, system administrators or operators still have unencumbered access to all functions that could shut down system operation or, worst yet, that could corrupt critical functions in an undetectable manner over a long period of time.

2.1.1 Separation of duty

Separation of duty [5, 6, 18] requires that certain system functions and components are accessible only by different insiders, thereby assuring that the erroneous or deliberately malicious actions of *a single insider cannot affect all critical functions at once*. By definition, separation of duty requires that insider roles be defined with separate access permissions, and that permission separation be supported by the underlying system. A typical example of separation of duty that has been used in business enterprises for many years is the separation and assignment of different employees to the accounts payable and purchasing departments. Violations of this type of separation would allow an employee who invents a fictitious company to issue purchase orders to that company as well as pay invoices received from that company, and effectively defraud his/her own employer. A direct application of this principle to system security administration has already been made in high-assurance systems where the duties of security administrators and operators are separated from other administrative functions [9, 13].

Another typical example of separation of duty has been the “n-person rule, ” where n >1, in the execution of critical transactions (e.g., missile launch, high-value purchases and/or payments). This rule requires that such transactions can only be completed by the separate actions of n individuals.

2.1.2 Critical function and data partitioning

Although separation of duty divides a set of critical functions of an application (e.g., an accounts payable, command and control, security administration) among multiple insiders*, a single* insider can still compromise *a critical function* that might lead to overall system failure. While simple separation of duty is necessary, it is not always sufficient to assure the integrity of *individual* critical functions accessible to insiders after duties are separated. To counter the possible misuse of a *single* critical system function by an insider, that function and its data must be partitioned and distributed across multiple system components, such that some or most components remain *inaccessible to any single insider*. The system should be designed in such a way that only if the number of critical-function partitions that are compromised by attacks exceeds some non-tivial threshold, the whole critical function is compromised.

For example, in the case of user authentication function and data, the ability to access an encrypted-password file cannot be easily denied to a system administrator; e.g., physical access by administrative personnel to such files is always possible. Yet, an unscrupulous administrator who can merely reads such a file can launch off-line dictionary attacks against one-way encrypted passwords. Cryptographic protocols that partition encrypted passwords and place them in separately protected files such that no single encrypted password is readable by any *single* administrator counter such attacks [10, 11].

2.1.3 Replication

The partitioning of a critical function and its data for the purpose of denying an insider access complete access to that function can lead to a single point of failure, which is precisely what we are trying to avoid. Failure of any function partition – possibly induced by an outsider's, not just and insider's, attack – would cause the failure of the entire function. To avoid such failures without giving up the advantage of function and data partitioning, would require the replication of each partition and the implementation of multiple-replica update protocols. Protocols for multiple replica updates have been in use for more than two decades and their design properties are well-understood.

2.2 Independence of Failure Modes and Attack Vulnerabilities

Avoidance of a single point of failure and its related design methods discussed above *cannot counter multiple non-independent (related) failures or attacks* against separated replicas of a system, duties (e.g., roles), or against critical-function partitions. For example, generic design flaws may affect *all* spatially separated replicas of a critical function using the same design, and could make that function susceptible to compromise. The same generic flaw could lead to identical (and hence non-independent) failures or penetrations of *all* critical component replicas. Similarly, separated duties (e.g., roles) *would not prevent system compromise if two or more insiders collude* to violate system integrity by joint malicious actions.

2.2.1 Design and implementation diversity

Design diversity ensures that generic flaws cannot arise that would affect multiple separated replicas of a critical component. For example, practice shows that different operating system families (e.g., MS Windows family and Unix/Linux family) are unlikely to be plagued by identical flaws. For any one of the common flaw types (e.g., buffer overflow, failure to validate input parameters or to enforce resource bounds), it is usually not possible to craft a single exploit that would work across diverse platforms. Hence, critical component replicas running different operating systems are more likely to survive an attack than if all replicas run the same operating system. Different forms of diversity have been used in practical designs [1,2].

2.2.2 Diversity of insider interests and skills

Insider personnel diversity seeks to provide similar benefits as those of design diversity. First, different individuals or entities must be assigned to different duties (e.g., roles) and critical functions. Second, different individuals or entities that have different interests (e.g., financial, corporate reporting lines) are less likely to collude. Third, different individuals must have the required skills to operate the diverse component platforms and applications (e.g., even if a Windows system administrator may have the skills to administer a Unix-based system, s/he may not have the skills to administer a specific applications on either system).

In summary, the key property provided by diversity *is failure mode/attack independence among component replicas*. This property enables the design of robust systems that can anticipate single and sometimes multiple failures and attacks and still continue to operate. However, design, implementation, and operational diversity is typically expensive and thus most designs aim at minimizing diversity without compromising survivability. This aim motivates, in part, the importance of fast detection of and recovery from failures and attacks presented below.

2.3 Fast Recovery from Failure and Attack

Even architectures that use spatially separated replicas of critical components must have a contingency plan that assures fast recovery of replicas from unexpected failures and attacks. In short, *one must expect to recover from unexpected events*. Why is replica recovery speed relevant to surviving insider attacks? Whenever fast recovery of replicas can be assured, the cost of diversity in survivable architectures can be lowered since only few failure-independent replicas become necessary. At the limit, *only two replicas* of a component are necessary to ensure survivability of that component with high probability for a long time; e.g., tens of years. For instance, in 1981, Gray [7] showed that, if repair/replacement of disks that fail at a rate of once per year is performed within short interval of time (e.g., an hour or less), disk mirroring ensures that the disk subsystem would fail with high probability only once in three thousand years. The cost advantages of "fast" recovery from failure were clearly understood by hardware designers, and this is why most reliable hardware designs in commercial use simply duplicate components that are assured to have independent failure modes.

2.3.1 Fast failure detection

While detection (and recovery) speed generally depends on the specific application in terms of desired up times, it is important that detection and recovery take place *before a second failure or attack can occur* in a component replica. Otherwise, failures and attacks may cascade and disruptions resulting from successful attacks may encourage additional attacks in rapid succession. This could enable the rapid propagation of an attack against multiple system replicas.

2.3.2 Fast repair

Rapid repair is also important to ensure that *only single failures or attacks can take place in reasonably small time epochs*. This helps reduce the otherwise necessary and costly diversity requirements by increasing the chances that a failed or penetrated critical function becomes operational again *before* another replica fails or is penetrated.

2.4 Attack Deterrence

All of the above principles of survivable system design assumed the certainty of failures and attacks. While this is a safe assumption in case of both physical and human failures, cost of the countermeasurs can be high. For example, the *fine-grained* separation duties will undoubtedly increase the number of administrative personnel. This would lead to an unavoidable recurrent cost. The alternative of coarse-gained or no separation of duty would incur the cost of recovery from an attack, which can also be high. An alternative to incurring such costs would be to decrease the probability of insider attacks by *deterrence.* The principle of deterrence says that *it is better to deter an attack than to have to recover from one*. This principle is grounded in the key observation that a rational attacker (1) will always avoid system areas where the chances of being detected are high or even non-negligible, and (2) will always seek the weakest link of a system. Hence a rational insider is likely to be deterred by attack-detection measures as his/her unencumbered access may no longer be the weakest link of the system.

The importance of this principle cannot be overestimated, particularly in the case of insider attacks. While diversity of insiders' interests makes insider collusion unlikely, *it cannot prevent it*. Hence, the ability to detect attacks would deter rational insiders, even if the detection occurs after the fact. Thus, deterrence becomes indispensable in system areas where the costs of countermeasures to, or recovery from, insider attack are prohibitively high. It typically takes place in the form of auditing [14].

Auditing of critical function execution deters an insider from execution an attack only if the auditing function is protected. The protection of the auditing function can be implemented in many systems by traditional access control mechanisms and application of the separation of duty principle. Hence any invocation of a critical function by an insider could be audited and hence deter an insider from launching an attack by misusing his/her privileged access permissions. To be effective, however, auditing must be conducted fairly frequently by separate personnel [9, 13].

2.5 Least Privilege Authorization

When applied to insider access, this principle [17] suggests that insiders have access only to the critical system functions and permissions necessary for carrying out their tasks. Further, it suggests that an insider's authorization to a critical function be restricted to include only the permissions or privileges necessary to carry out that function. This requires that permissions (1) assigned to an insider to perform a duty need be tailored to fit tightly the needs of that duty, and (2) assigned to a critical function be kept to the fewest necessary for the function's operation. This principle *complements* separation of duty in that permissions assigned to roles are kept to the minimum necessary for that role.

However, the application of the least privilege principle at process and object granularity also provides an illustration of unintended and undesirable consequences. Fine granularity of permissions, by definition, requires that multiple permissions must be initialized, reset and checked during system operation and, hence, their interaction must be well understood by system administrators. In view of the generally accepted fact that the largest source of security problems is "administrator error" and that, among all administrator errors, configuration errors are the main caused of such problems, the application of the least privilege principle appears to be a less than useful approach in this context. Some evidence of user's difficulties in specifying fine-grained access control policies in current systems is also provided by a recent studies [16].

3 Cost Factors

Architectures that employ the survivability principles discussed above incur extra costs, including fixed, non-recurrent, development, and equipment costs, as well as recurrent system administrative and maintenance costs. The fixed costs (e.g., hardware replicas) are typically marginal compared to the recurrent costs (e.g. personnel salaries and benefits) over a systems' lifetime. Hence, it becomes important to come up with designs that minimize and tradeoff recurrent costs against fixed costs.

Spatial separation, replication, and partitioning of critical functions requires substantial additional equipment and development costs, but *only marginally more recurrent* maintenance and administrative costs. In contrast, separation of duty, design diversity, and attack deterrence imposes *substantial recurrent* administrative costs not just in terms of personnel but also in terms of desirable administrative skills. Separation of duty and diversity imposes additional recurrent costs since it requires different individuals in different roles to carry out the fine-grain administrative functions. Deterrence requires increased and frequent use of administrative tasks, such as audit-trail review and audit-event correlation and analysis,, and requires special skills and analytical tools.

4 Conclusion: A Call for Research and Development Experiments

To date there have been very few compelling experiments reported whereby a significant-size subsystem can survive an insider attacks. Some systems have separated system administrative functions to a relatively fine granularity [9]. Yet experiments to indicate the effectiveness of such separation have not been conducted to date. The basic question that we ask is how can one use accepted design principles of security and reliability, as well as established techniques, to design systems that survive insider attacks.

We believe that no single principle, design method or implementation technique will be sufficient to accomplish this task. In referring to the (naïve) use of cryptography as the single, predominant technique for security, the late Roger Needham once said: "*he who thinks cryptography is going to solve his security problem understands neither cryptography nor his security problem*." When it comes to handling insider attacks, one could safely paraphrase Needham substituting all *individual* security and reliability techniques for cryptography. Indeed, piecemeal component engineering is unlikely to produce effective protection against insider attacks in large systems.

In view of the dearth of practical solutions for surviving insider attacks in significant-size systems, we suggest that experiments in applying well-accepted principles and design methods to critical subsystems (e.g., user authentication, DNS) are necessary to provide effective and quantifiable assurances. We caution, however, that different tradeoffs emerge in applying survivability principles to practical designs, thereby requiring a careful balance among the costs of countering insider attacks, recovery from attack, and attack deterrence, and between the fine granularity of access permissions and ability to administer these permissions is a safe manner.

References

[1] A. Avizienis and J.-C. Laprie, “Dependable Computing: From Concepts to Design Diversity,” *Proceedings of the IEEE*, vol. 74, no. 5, May 1986.

[2] A. Avizienis, J.-C. Laprie, B. Rendell, and C. Landwehr, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” *IEEE Transactions on Dependable and Secure Computing*, vol.1, no. 1, Jan-Mar 2004

[3] R. Bobba, S.I. Gavrila, V.D. Gligor, H. Khurana, and R. Koleva, “Administering Access Control in Dynamic Coalitions,” *Proc. of the 19th USENIX Large Installation System Administration Conference (LISA)*, San Diego, CA, December 2005.

[4] D. Boneh and M. Franklin, “Efficinet Generation of Shared RSA Keys,” *Journal of the ACM (JACM)*, Vol. 48, Issue 4, July 2001

[5] D. D. Clark and D.R. Wilson, “Evolution of a Model for Computer Security,” in *Report of the Invitational Workshop on Data Integrity*, Z. Ruthberg and W.T. Polk (eds.) NIST Special Publication 500=168, Appendix A, September 1989.

[6] V.D. Gligor, S. I. Gavrila and D. Ferraiolo, “On the Formal Definition of Separation-of-Duty Policies and their Composition,” *IEEE Symposium on Security and Privacy*, Oakland, California, May 1998, pp. 172-185.

[7] J. Gray, “The Transaction Concept: Virtues and Limitations,” *Proceedings of the VLDB*, Cannes, France, 1981.

[8] P. Gupta, V. Shmatikov. Key Confirmation and Adaptive Corruptions in the Protocol Security Logic,” *Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis* Seattle, August 15 - 16, 2006

[9] M.S. Hecht, M.E. Carson, C.S. Chandersekaran, R.S. Chapman, L.J. Dotterer, V.D. Gligor, W.D. Jiang, A. Johri, G. L. Luckenbaugh, and N. Vasudevan, “Unix without the Superuser,” Proc. *of the USENIX Conference*, Phoenix, Arizona, June 1987, pp. 243-256.

[10] J. Katz, R. Ostrovsky and M. Yung, “Efficient Password-Based Authenticated Key Exchange Using Human-Memorable Passwords,” Advanced in Cryptography - *Eurocrypt 2001, Innsbruck, Austria, May 2001.*

[11] "Two-Server Password-Only Authenticated Key Exchange," J. Katz, P. MacKenzie, G.Taban, and V. Gligor, in Proccedings of *Applied Cryptography and Network Security (ACNS), N.Y. 2005*

[12] H. Khurana, V.D. Gligor, and J. Linn, “Reasoning about Joint Administration of Access Policies for Coalition Resources,” Proc. of the IEEE International Conference for Distributed Computer Systems - ICDCS, Vienna, Austria, July 2002.

[13] National Security Agency, “*A Guide to Understanding Trusted Facility Management,*” National Computer Security Center, NCSC-TG-015, Version 1, 1989.

[14] National Security Agency, “*A Guide to Understanding Audit in Trusted Systems*,” National Computer Security Center, NCSC-TG-001, Version 1, 1988

[15] P. G. Neumann, “Principled Assuredly Trustworthy Compusable Architectures,” DARPA Final Report, SRI Project P11459, December 28, 2004.

[16] R. Reeder and R. Maxion, “User Interface Dependability through Goal-Error Prevention,” *International Conference on Dependable Systems and Networks*, Yokohama, Japan, June 2005.

[17] J. H. Saltzer and M.D. Schroeder, “The Protection of Information in Computer Systems, “ *Proccedings of the IEEE*, vol. 63, no. 9, Sept. 1975.

[18] R. T. Simon and M.E. Zurko, “Separation of Duty in Role-Based Environments,” *Proceedings of the Computer Security Foundations Workshop*, Rockport, Mass. June 1997.

Preventative Directions For Insider Threat Mitigation Via Access Control

Sara Sinclair and Sean W. Smith

Department of Computer Science, Dartmouth College

Abstract Much research on mitigating threat posed by insiders focuses on *detection*. In this chapter, we consider the *prevention* of attacks using access control While recent work and development in this space are promising, our studies of technologists in financial, health care, and other enterprise environments reveal a disconnect between what "real world" practitioners desire and what the research and vendor communities can offer. Basing our arguments on this ethnographic research (which targets both technology and the human business systems that drive and constrain it), we present the theoretical underpinnings of modern access control, discuss requirements of successful solutions for corporate environments today, and offer a survey of current technology that addresses these requirements. The paper concludes by exploring areas of future development in access control that offer particular promise in the struggle to prevent insider attack.

1 Introduction

Threat mitigation can be reactionary or preventative When it comes to insider attack, much current work falls in the former camp: how can we *detect* it? In this paper, we pursue the latter angle: how do we *prevent* insider attack? Other chapters in this book address prevention by targeting insiders' incentives and motivation. In this chapter, we target insiders' opportunity and technical capability to execute attacks.

In particular, we focus on the electronic environment in which insiders operate. Each employee of an enterprise needs access to certain internal electronic resources (databases, file servers, programs, etc.) in order to perform her job within the context of the organization. Computer security researchers often approach the problem of insider threat assuming that an organization implements a correct access control policy; this policy simultaneously grants the user sufficient *privileges* to perform necessary tasks, yet also appropriately *constrains* her access according to the principle of least privilege (and other primitives, as discussed in Section 3). This notion implies several other assumptions about the nature of policies and the human systems they are supposed to govern:

1. For any given organization, there exists an access control policy that simultaneously grants and constrains access in a manner that is correct for that organization's goals.
2. At one point, the organization correctly identified and implemented one such policy.
3. The correctness of the policy and its implementation are maintained over time, even as the resources, users, and organization's goals change.

However, we have heard over and over---from both information security professionals and end users in industries at risk of insider attack---that these assumptions do not hold in practice. According to these reports from the trenches, currently available techniques and technology do not seem to achieve the ideals promised by access control principles and theories.

For example, organizations have shared difficulties identifying correct policies (or even determining whether they exist), as these two examples demonstrate:

- The first phase of many authorization deployment schemes requires an initial identification period in which technologists, principal managers, and users are gathered to chart out all required access and constraints. A senior information security colleague in a highly relevant enterprise regards this approach as ludicrous—business users don't have the time, and even if they did, he regards it as impossible for such a group to codify all the nuances of the enterprise's real-world operations.
- One financial services colleague laments the "access control hygiene" problem [Donner, 2001]. The need to quickly grant access leads to shared passwords and "spaghetti" access controls. No one has any idea who has access to what, or why, yet off-the-shelf access control solutions dot appear to offer sufficient agility to replace current ad-hoc mechanisms.

Enterprise partners also describe scenarios in which implemented policies do not align with enterprise goals; in these cases, users are forced to violate the policy in order to meet their job requirements; the following is a sample of the anecdotes we have documented:

- A colleague in the oil industry discussed how the security rules required a password to enter the refinery control room. However, that password is written clearly on the control room door, because practicality requires that anyone be allowed to enter; in case of a fire emergency, someone has to turn things off.
- A practitioner in the medical industry talks about having to cut-and-paste medical images from the approved image application into PowerPoint (a violation of policy), then emailing the document to an external colleague in order to receive a second a opinion in difficult cases. The policy prohibits moving images in this way because it shifts data outside the system's ability to monitor its movements, yet

the practitioner has no other way to efficiently receive the information she needs.

- An information security manager in the finance industry now insists approved data applications remain flexible and attractive---because otherwise his users move the data into convenient third-party spreadsheets and Web-based tools. When a policy interferes with getting their job done, the users move the data beyond the reach of that policy.
- A colleague reported that the Chief Information Security Officer (CISO) of a large US corporations spent the first part of each day figuring out how to "work around" new security policies---which his own group had put in place---in order to get his job done.
- A doctor serving on his enterprise's IT committee, when hearing we worked in computer security, challenged us: was our goal to build better "IT security police," or to help improve the lives of patients? It was clear that he was not interested in helping us achieve the former end---and that his previous experience with computer security made him suspect it had nothing to do with the latter.

(Understandably, gathering attributable anecdotes in this space---let alone solid data---is tricky, as admitting to breaking IT policy can have repercussions for both individuals and organizations.)

We have also encountered enterprises that have essentially given up on *a priori* access control altogether, as in the following examples; in these cases, implemented policies fail to provide desired constraints, but at least meet minimal privileging requirements:

- Multiple medical institutions' policies allow every authenticated clinician to see any patient's data; in their experience, limiting access to "need to know" is too complex, and erring on the side of excessive restriction can directly result in loss of patients' lives.
- A professional in the power grid talks about how any person in the control room can do anything---because in the case of emergency, it would take too long to carry out the authentication process (or scramble to gain sufficient authorization if the party in question didn't already have it).

These stories indicate that real-world enterprises have a hard time not only identifying and implementing correct access control policies, but also determining if such policies are even practically possible for their organizations. When combating insider threat, if we assume that all enterprises have correct, effective policies already, we ignore an area of research that many practicing professionals are eager to see pursued. With the belief that work in this space will improve organizations' ability to prevent insider attack, it is this mismatch between the theory and practice of access control that we target here.

The next section of this paper identifies the types of insiders and attacks against which new research in preventative mechanisms could be useful. Bearing this threat model in mind, Section 3 provides an overview of principles and primitives on which modern access control technology is formed. Section 4 draws on our collaboration with technologists and policymakers from the financial, healthcare, and other industries to characterize requirements that drive and constrain solutions in these environments. We survey in Section 5 current access control technologies, and evaluate those solutions with respect to the requirements. Finally, Section 6 synthesizes from the survey a number of important insufficiencies of current technology, and offers ideas on how new systems or business practices could improve the state of the art. Throughout the paper we continue to share anecdotes and observations gleaned from professionals in a variety of industries; these stories help us understand better how to design our research solutions so they translate well into the real world.

2 Definitions and Threat Model

Choices of words and models allow us to highlight different aspects of a problem. Here we define a number of terms to help us narrow in on the parts of the insider problem we are targeting in this paper. We also identify specifically what types of threats we aim to mitigate.

2.1 The Insider

We define an *insider* of an organization as any person who has some legitimate privileged access to internal digital resources, i.e., anyone who is allowed to see or change the organization's computer settings, data, or programs in a way that arbitrary members of the public may not. This includes full-time employees, but may also include temporary workers, volunteers, and contractors, depending on the nature of the business. In some cases an insider may also be the child or spouse of an employee; one medical institution reported to us serious concern about doctors' families accessing medical systems through company laptops.

Note that in many cases the permission an individual has to access internal resources is not the *explicit* permission afforded to him by the organization, but the *effective* permission: a hospital may have an official rule stating that doctors may not share their laptops with their children, but also have a de facto rule of looking the other way. Organizations can implement penalties, incentives, and technology to enforce official rules and limit the set of insiders, but must also be pragmatic in accounting for effective insiders outside the formally approved set.

2.2 Types of Insiders

For our analysis, we divide the insiders who perpetrate “insider attacks” into three broad categories:

1. Those intent on malicious action,
2. Those willing to act for their personal benefit over that of the organization when the opportunity presents itself, and
3. Those insiders who inadvertently use their privileged access to do harm.

As in many areas of security, protecting against a determined and resourceful individual in class (1) is very hard. We are not asserting that better access control systems can prevent all attacks by those insiders in this category. We are, however, asserting that access control systems may make it harder for these people to do wrong, as well as reduce the opportunity and probability of harmful action by insiders in the other two categories---and that reports from the trenches support this view.

2.3 Damage of Insider Attacks

The damage to an organization by insider attacks against its electronic resources may take one or more of the following forms.

- The attacks may be destructive to systems or their availability, such as data corruption or denial-of-service attacks.
- The organization may suffer financially from the attack (either in direct costs or in lost productivity).
- They may consist of actions prohibited by law, such as insider trading or disclosure of patient health information, and thus result in regulatory fines or punishments.
- The attacks may also violate corporate rules or less formal customs, such as cultural expectations of privacy (for example, a bank employee who monitors his ex-girlfriend’s account balances, or hospital employees who read the medical records of celebrity patients).

In addition to the risk of lost business associated with destructive attacks, the cost of repair or theft, and the penalties incurred by legal violations, enterprises increasingly worry about the *reputation risk* that publicized insider attacks pose. One senior security professional of a large investment bank described to us the horror he feels at the thought of his firm receiving negative press in a major newspaper; any breach---even just a perceived breach---can dramatically impact stock prices and market shares. These complex costs associated with reputation risk are an important incentive for effective preventative measures.

2.4 Threat Model

Given these definitions of insiders and insider attacks, we will now discuss the type of scenario we will focus on for the rest of the paper; specifically, we address the concern of improper privileging.

We say that an access control policy P is *correct* for an organization O if P provides users access to and constrains users' access of electronic resources according to O's goals. These goals include business objectives, corporate policies, and regulatory requirements. A policy P that is correct for O is also *practically correct* when adopted by O if it meets the following key requirements, as discussed earlier: first, the correct policy must be logically possible (not self-contradictory or otherwise unrealistic); second, the policy must have been correctly implemented in the organization at one time; third, that correctness must have been maintained from the time of implementation to the present, across various changes in the organization and its goal; and fourth, this policy must actually match what happens in practice within the organization.

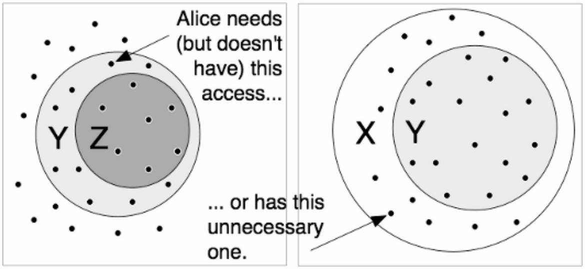

Alice needs (but doesn't have) this access...
Y
Z
... or has this unnecessary one.
X
Y

Fig. 1. Under-privileged (left) and over-privileged (right) access. The correct privileges for Alice's job are represented by set Y, and the actual privileges by Z or X.

Policy implementations that do not conform to these requirements will result in users having improper access privileges. For example, assume that the correct policy P_c provides Alice with the privileges of set Y in Figure 1. If P_c is not implemented correctly, Alice may only receive the set Z; in this case, Alice is under-privileged, and may not be able to complete the tasks her job requires. Similarly, if P_c was correctly implemented but did not adapt when Alice changed departments and started her current job, she may have access to set X, which constitutes over-privileging. Under-privileging can lead Alice and her coworkers to take matters into their own hands (with shared passwords, copy-and-pasting, and the like) in order to get their jobs done---thus leading to a *de facto* policy that not only allows over-privileging, but also moves the policy outside the realm of what's manageable.

From the anecdotes related in Section 1, and from many other conversations with practicing professionals about deploying and managing access control systems, we believe that over-privileging is a common occurrence in today's enterprise environments. Furthermore, we believe that over-privileging represents a significant source of insider threat for these organizations. The rest of this paper works to understand this issue of improper privileging, its relation to the problem of insider attack, and potential solutions.

3 Background and Primitives

The task of limiting access to electronic resources has been around almost as long as the electronic resources themselves. Scheduling algorithms in the first time-sharing systems provided multiple users with a (hopefully fair) share of CPU time, disk access, and network connectivity. In this section, we will consider the theoretical underpinnings necessary to use access control to prevent insider attack in distributed systems.

3.1 Authentication and Authorization

The computer security world defines *access control* as providing or limiting access to electronic resources (we can also say granting or limiting trust) based on some set of credentials. Access control typically consists of two components: *authentication* and *authorization*. Authentication is showing who (or what) you are; i.e., demonstrating possession of certain credentials. Authorization is the system determining if your credentials are sufficient to provide you with a requested type of access.

In many cases, we think of authentication in terms of *identity*: is that really Alice on the other side of the keyboard? In reality, there are lots of different types of credentials and properties other than identity that we can authenticate and use in making authorization decisions. For example, any valid "student" ID with Alice's photo will permit Alice to watch a Dartmouth hockey game; Bob may trust the person with the nametag at the appliance store to help him choose between features, no matter what the person's name is; however, Carlo may not trust the "valet" with the baseball cap at the hotel to take his car.

In focusing on the authentication decision---making sure that it's really Alice---it's easy to forget the subsequent mirror goal: how can we recognize when Alice is no longer there? Some real-world enterprises call this the *de-authentication* problem.

The mirror problem to authorization is *de-authorization* or *revocation*: if Bob does something bad (or simply changes jobs), how do we make sure he does not keep his now-inappropriate privileges on the system?

3.2 Access Control Principles

We quickly review the basics of access control and authorization, as they relate to an enterprise regulating how insiders access resources. (For a more thorough discussion of this material, readers should consult an introductory book, such as Smith and Marchesini, 2007.)

In the basic picture, we usually start by thinking about a matrix. Each column represents an *object:* an electronic resource. Each row represents a *subject*: an actor, such as an employee, who can take actions. Each box then lists the *privileges* that subject has to that object.

This basic model (e.g., Lampson 1974) lets us start thinking about initial principles.

The principle of *least privilege* teaches keeping each box as sparse as possible; the fewer actions one is allowed to take, the less the chance that, by accident or malice, one can cause damage. For example, the authors of this chapter often remove their own "write" permission from critical program or text files, in order to lessen the chance of accidentally modifying them while examining them with an editor.

The principle of *escalation* allows a subject to add back certain rights to an object. (Essentially, the rights-box for that object *itself* becomes an object in the matrix.) For example, when we really need to change one of those critical files, we can do so---after first adding our privilege back. Some enterprises make this process more heavyweight: e.g., by requiring the employee to explicitly request the privilege from a manager, or to explicitly acknowledge that the elevation is significant and will be audited (the latter is mechanism is sometimes called *break-glass*, used as a noun: "there was a break-glass on that record"). Some researchers have even formalized this notion as *optimistic security* [Povey, 1999].

The principle of *separation of duty* takes least privilege into another dimension: we decompose a critical action into separate pieces, and require that different subjects take these actions.

3.3 MAC, DAC, and Intermediate Schemes

Initially, we might try to use this basic matrix model to actually reason about and manage access control. Management at this extreme, raw level has its own name: *discretionary access control (DAC).* The owner of each object has full discretion on setting permissions.

However, with this approach, reasoning about or ensuring any high-level properties of what can happen in the system can quickly get out of hand. Consequently, approaches emerge to start imposing some structure and order on what happens. *Mandatory access control (MAC)* imposes strict limits on the permissions that can be granted, no matter what an object's owner likes. Usually, MAC is used in conjunction with *multilevel security (MLS),* where subjects and objects are organized into a lattice, and permissions ensure that information only flows in correct directions. (The MLS lattice was developed in the context of U.S. defense computing. Think of *clearance levels*, *compartments*, and *need to know*: an uncleared subject shouldn't read a top-secret document; a subject cleared for top-secret but only with a need to know about X shouldn't be able read a document associated with Y.) This seminal work resulted in a set of practices and principles (usually referred to as the *Orange Book*, the nickname of one of the resulting standards [DoD, 1985]) to ensure that users and data of all sorts of sensitivity levels can exist securely on the same system, even if some want to cheat---i.e., even if some users want to execute an early form of insider attack.

Many other formal models of security have been proposed, in order to bring some order to the chaos. The *Chinese Wall* is one that is particularly relevant to data-oriented enterprises. Here, a subject may have access to any object in some set---but once the subject exercises that right for one of these objects, she loses access to all the others. If Alice is standing on the Great Wall of China, she has the ability to jump to either side; however, once she jumps to one side, she can't jump over the wall to the other side. If Alice is a broker in an investment bank, she might have the ability to look at the records for client X or client Y; however, once she looks at X, then looking at Y might be a conflict of interest (depending on their relationship).

3.4 Users and Groups

In Section 2.2 above, we introduced the basic access control matrix model. In Section 2.3 above, in order to make it more manageable, we introduced some refinements and restrictions to the access rules. However, another approach to making it more manageable is to start putting more structure on the left-hand side of the matrix: how "users" map to rows.

One basic approach is to label the rows with a new construct, *domains*, and then think about users map to domains. E.g., in a UNIX-style OS, typing "sudo" to elevate privilege would correspond to changing domains to "superuser."

Another is to organize a set of users into a *group* and then use group membership to decide permissions. Traditional UNIX file permissions operate this way, although this group approach raises some annoying corner cases. What if a user belongs to two different groups, and their permissions differ? What if a user's personal permissions differ from the group's? Students learning UNIX file permissions for the first time usually get rather confused about such details, and even advanced programmers have trouble. Steve Crocker, formerly of ARPA and USC, reports that he regularly challenges system administration trainees to develop a UNIX file permission policy that matches a very natural and simple scenario from business organizational structure. Each year, each student comes with with a policy; and each year, upon closer examination, none of them actually meet the desired goal.

3.5 Roles and Role Engineering

The concept of *role-based access control (RBAC)* takes the indirection even further. As with the concept of domains, the rows in the matrix are labeled with *roles*. Depending on the design of the RBAC scheme in question, users may be assigned one or multiple roles; similarly, users in the latter set may have all their roles *active* all the time, or the scheme may constrain users to one or more active roles from their assigned set. (This capability helps provide, among other things, separation of duties.) The roles themselves may be organized into a rich *hierarchy*, with inheritance and other properties.

In the field, some practitioners use the term "role-based access control" loosely, just to refer to deciding access based on a user's job rather than their name. (RBAC standards and literature [NIST; Ferrailio et al., 1992 and 2007] present a more formal vision of what it takes for an access control system to be truly "role-based."

Existing research also offers guidance on *role engineering*, the process of identifying and managing the roles used in an RBAC scheme. Two basic approaches to role engineering, top-down and bottom-up, present different strengths; the latter tends to be quick and easy to roll out, but difficult to maintain, whereas the former approach is more time-consuming at the beginning, but offers advantages during role maintenance and management. The work in the top-down space aligns largely with requirements engineering principles, and includes scenario-driven, goals-driven, and hybrid approaches. Suggested bottom-up techniques include role clustering and discovery algorithms.

Some researchers use the term *attribute-based access control* for the somewhat similar concept of deciding access on something other than one's name.

3.6 Public Key Cryptography

Springing from the intersection of computer science and mathematics, *public key cryptography* is a tool that allows someone to take an action with their secret *private key* that can be verified by anyone who knows the matching *public key*---but knowledge of the latter does not (yet) enable one to calculate the former. Public key techniques can enable authentication directly---the user cryptographically proves knowledge of her private key. Public key can also provide glue to make other aspects of authentication work---e.g., it can enable a manager Alice to digitally sign an assertion that employee Bob should have access to the records for client C. Because requires neither shared secrets nor direct knowledge of the users (e.g., Alice's signed statement about Bob doesn't require that the record server have heard about Bob beforehand), public key techniques are attractive when authentication and authorization needs to cross boundaries within or between enterprises.

4 Requirements

Armed with the principles from the previous section, we now characterize the settings in which we hope to mitigate insider threat. We focus particularly on large corporate environments; although smaller organizations and those in other domains (particularly government) face legitimate insider threats, we believe that large corporations offer particularly fascinating challenges and opportunities for new development. Furthermore, we hope that improved access control solutions will generalize to simpler environments.

In this section we consider the functional requirements of access control in large corporations, as well as characteristics of these environments that drive and constrain the solutions actually deployed, including domain-dependent factors. We move on in Section 5 to evaluate current tools and techniques in reference to the requirements characterized here.

4.1 Functionality

The way an organization implements basic access control faculties (authentication, authorization, constraints, etc.) depends on its goals; large corporate environments' requirements of their access control systems often include the following.

Distributed authentication: resources managed by different entities throughout the enterprise should be accessible via a common authentication scheme. This means that an individual user should need a small number of credentials to authenticate throughout the organization, and not an extensive personal library of identifying information.

De-authentication: the system should recognize when a user is no longer using her authenticated session, and prevent other users from using that session illicitly.

Distributed authorization: as with authentication, resources managed by different corporate units should share a common infrastructure for presenting and evaluating authorization requests. Information used in authorization decisions may flow to distributed resources from a central source, or may be gathered in a distributed manner from a variety of authorities. (As a simplified example, consider residents of a condominium development who wish to add a deck to their unit. The community bylaws might require residents to go to the central association meeting to ask permission, or may require them to also query each of their neighbors individually before performing the renovation).

Distributed privilege assignment: The duties of issuing authorization privileges should not be concentrated exclusively in the IT department, but should generally rest with business users who are qualified make issuance decisions. This capability may be distributed to the point that average end users can assign privileges to each other; this is useful in scenarios where *delegation* of one's own tasks (and thus privileges) to a peer or subordinate is desirable.

De-authorization or privilege revocation: Corporations wishing to reduce the risk of attack by recently fired employees must be rigorous in de-authorizing users in a timely manner. Privilege revocation is also essential

Expressive Constraints: Increasing regulatory requirements combined with enterprise risk reduction strategies leads many corporations to seek fine-grained (often rule-based) constraints on access to follow principles like least privilege. Constraints may be either hard-wired (Avi can't access resource X) or dependent on additional context (if Xia has already purchased so much in supplies, she cannot put in another requisition order).

Privilege Auditing: Many corporations also aim to reduce risk by regularly auditing the privilege sets of their users. Access control systems should allow qualified managers to evaluate which users have which privileges, and potentially additional information, such as when the last time a user used a certain privilege, or how many of the user's peers share it.

Post-facto Auditing: Enterprises also need to be able to audit access control transactions that have occurred in the past, both to evaluate the success of their policies and to gather information in case of a security failure.

Many professional colleagues consider these capabilities as essential to access control systems, and vendors have accordingly attempted to integrate them in current products. However, organizations' deployment of these solutions are challenged other business requirements, which must also drive technology in this space.

4.2 Usability and Cost

Cost has long been a critical consideration for corporations deploying new technology. When thinking about cost, it's important to consider not just the monetary outlay for the obtaining, installing and maintaining the technology---it's also important to consider the impact the technology has on the enterprise's business processes. Can employees figure out how to use it? Does the technology make it easier for employees to get their jobs done, or does it slow them down? *Usability* and cost are thus closely linked in access control solutions that require interaction with users, be they non-expert employees or experienced system administrators. The motivations for usable authorization solutions are similar to those that drive quality user interface design (the usual connotation of "usability"), although the usability issues in this space extend far beyond the interface.

Usability and cost find a clear connection in the notion of productivity. Systems that are easy to use and facilitate users' doing their jobs (instead of getting in users' way) enable those users to complete work at a more rapid pace. We can expect that the introduction of new systems will reduce productivity temporarily, but highly usable solutions will improve performance after an initial period of user training and adjustment. New systems whose usability does not improve with time can result in direct reduction of productivity.

Systems that contain usability barriers also present "hassle cost," which can pose more serious concerns than reduced productivity. Basic cases include quiet boycotts of new solutions, such as nurses who continue to record patient data in paper-based notebooks instead of using laptops with too-small keyboards. Having some data in notebooks can make it more difficult for the next nurse on shift to complete his job, causing tensions in the department. In contrast, louder boycotts can involve entire departments or classes of users, and often result in tech teams being forced by upper management to change or remove new solutions. (The flow of communication between technologists and users in such scenarios can be a factor when these large-scale boycotts occur; direct feedback during early testing of a new technology often allows for usability refinement and prevents wholesale revolt.) High degrees of hassle cost engender animosity against the technology group, and can impact the success of future deployments.

Our external collaborators have also related surprising stories of users' ingenuity in circumventing systems that posed too great a hassle cost. One organization deployed proximity detectors to de-authenticate users. After an initial period of push-back, it seemed as if end users had come to accept the solution; however, the statistics regarding session length betrayed that the users were not actually being de-authenticated as desired. Further inquiry revealed that the proximity detectors had no minimum distance limit, and that a user could take advantage of this by covering a detector with a Styrofoam cup. After several iterations of refinement, the organization eventually had to roll back its deployment of proximity detectors because it could not satisfy the usability requirements of its users.

In the Styrofoam cup example, the company discovered rather quickly that users were manipulating the control technology. In other cases users have gone months or years quietly circumventing solutions that meet security requirements in order to complete their jobs in a usable way. Such practices present tremendous opportunity for insider attack.

Unusable systems will be circumvented or detested, with repercussions for employee morale (and future inclination to respect policy).

4.3 Scale and Complexity

A large corporation's scale and structure may challenge technologists' ability to deploy effective access control mechanisms. In particular, the number of employees, the geographic area over which those employees are distributed, the strength of centralized management or recognizable hierarchies within the company, and the amount and speed of change the company experiences in size and structure can all impact the feasibility of deploying practically correct policies. Table 1 provides a summary of these characteristics and the range of values that they can take on in our target settings. Below are also three small case studies to illuminate these issues; these fictional scenarios are based on observations and anecdotal evidence from collaborators in a variety of fields.

Characteristic	Range of Potential Values	
Number of employees	~10,000	~100,000+
Technology support	Fully centralized	Some or all provided by individual business units
Organizational change	Stable, low turnover	Undergoing corporate merger
Management structure	Hierarchical, single project supervisor	Matrixed, multiple dynamic project assignments, no single supervisor
Location	Single campus	Hundreds of locations world wide

Table 1 . A summary of scale and complexity issues that challenge corporations in using access control to prevent insider attacks.

Corporation Alpha has about twenty thousand employees on a single campus location. These employees are divided into specialized departments, each of which maintains a high degree of management autonomy. There exists a single technology support unit at the center of the company, but individual departments often require specialized information systems to operate effectively, and sometimes bring on additional internal support people to manage them. Despite their divergent specialties, different departments are highly dependent on each other for accurate information, and thus require a high degree of interoperability among their computer systems. Interdependence combined with autonomy creates a chicken-and-the-egg problem; individual groups cannot change solutions one department at a time without impacting others, nor is it easy to convince all departments to buy into a new solution at the same time.

Corporation Beta has about a hundred thousand employees who are spread throughout the eastern United States. The company is growing rapidly; it just completed a merger with one competitor, and another acquisition is under negotiation. The employee base, the number of campuses, the types of electronic resources, and even the sectors in which the company does business are all changing from one month to the next. The company must pilot access control technology during this period of quick evolution, yet faces tremendous difficulty in planning and deploying solutions that are sure to be sufficient over the next few years.

Corporation Gamma has about one hundred thousand employees spread throughout the world. Its overall size is stable, although much of the company is organized to morph fluidly in reaction to new needs or trends. A centralized supervisory hierarchy exists for performance review purposes, but employees may be assigned to a number of different projects and functional supervisors throughout the year. (These characteristics of Corporation Gamma match the description of "matrixed" organizations used in the business management field [Burns et al., 1993 and Anderson, 1994].)

The challenges that Corporations Alpha, Beta, and Gamma face are varied, and the solutions for each will be similarly diverse. However, they offer us an intuition for the types of scalability and complexity parameters that large organizations must consider in implementing practically correct access control policies.

4.4 Domain Considerations

From high-stakes service industries (such as hospitals) to moderately-paced retail enterprises (commercial banks) or aggressive corporate settings (investment firms), it is clear that challenges specific to an organization's domain can further confound the problem of choosing and deploying access control mechanisms. For example, a user's daily activities may vary little in one, while the other requires tremendous flexibility in performing tasks whose definition and scope change continuously. The issues presented above---usability, cost, scale, and organizational complexity--are factors in access control design across domains. However, the specific requirements that these factors produce can very with the mission and culture of an organization. We clearly cannot enumerate all the types of domain-dependent drivers and constraints, but present in this section a small set of enlightening examples.

User Expectations and Communication: The expectations of the user population can dictate many qualities of an access control system; many senior professionals (like lawyers, doctors, and business executives) are less willing to adapt to using new technologies, and many computer professionals are more inclined to use their advanced knowledge to circumvent security controls when they feel those controls prevent them from doing their jobs efficiently. (Partners in a number of industries say that security experts are usually the biggest violators of corporate security policies; they feel their knowledge of the rules entitles them to make exceptions for themselves. On the other hand, usability/security researchers report that knowing how to circumvent the rules is considered a badge of honor in IT circles---it indicates the wisdom of seniority.) Conversely, investment banks tell us that they are confident that (internal) users will let them know when they find usability issues; highly driven individuals recognize the benefits technology can have on the firm's profitability, and complain loudly when a technology makes their job harder (and often offer praise when it does the reverse).

Resource Dedication: Of course, the willingness of users to provide assertive feedback when they encounter problems is only useful when an organization has the resources and experience to make use of that feedback. The financial industry spends a large percentage of its operating budget on IT resources; large investment banks often have small armies of developers and technical support people to craft and manage information systems that will provide them a critical edge against their competitors. Efficient access to data correlates directly with profit, yet regulatory requirements and threats of insider attack also require correct access control to maintain that profitability. In this setting it is relatively easy for managers and users to see the link between successful security technology and achievement of the firm's fundamental mission: making money through ingenious manipulation of information.

Enterprise Mission: The mission of health care organizations fosters a dramatically different dynamic among technologists, managers, and users than that of investment banks. Much of the pressure in recent years for *Electronic Medical Records (EMR)* has been not from medical providers, but from medical insurers (including the federal government's Medicare program, which covers nearly 40 million Americans [ADA]). Digital information sharing presents huge benefits for medical insurers, and new laws similar to those in finance place regulatory constraints on information access. However, the role of technology and information is very different in healthcare than in finance, as is the budget supporting it. (One health care organization reports that only 3.5% of its operating budget goes to IT development and support.) The reasons behind these differences are complex and beyond the scope of the paper. It is worth noting, however, that where users in finance feel that new technology gives them a competitive edge on the market, users in healthcare often feel that new technology hinders their ability to complete their professional objectives. One doctor asked, "How can security technology help me make people more healthy?" His colleagues agreed: they were more likely to commit themselves to understanding and using access control mechanisms if they could clearly see a connection with their overarching mission as medical providers. This connection (or lack thereof) of security to mission combined with differing budgets and expertise among technical staff can constrain the number of acceptable access control solutions that will be deployable in some domains.

Urgency of Access: Another difference among domains that can drive system requirements is the urgency with which users must access information. In some settings, users who are underprivileged can be delayed in placing a retail purchase order or running a budget report---but this delay has no serious business repercussions. In other situations, users who remain underprivileged can be prevented from making a big deal by a certain deadline, which in turn results in lost profit. In other domains, insufficient access privileges can have more dire consequences; for example, not having critical medical information before a surgical operation or while handling an emergency room crisis or not being able to log in to an industrial computer during a factory malfunction can result in loss of lives, destruction of property---and legal and business consequences.

These examples are drawn from a large set of complex domain-dependent issues that can interact with an organization's ability to prevent insider threat via access control. In the following section we will survey current access control implementation and management tools; where we have found insufficiencies in these solutions, we will provide examples from the real world of how those tools could not effectively prevent insider attacks for a particular organization. We note, however, that the balance of usability, cost, scalability, and flexibility required by one organization or domain is not the same as by another. Our criticisms serve to drive future research and development, and should not be taken as a rejection of current access control technologies.

5 Tools

In this section we survey technology currently available to meet the access control requirements presented in Section 4. Some of the solutions in this section have been actively deployed in production environments for a long time, some are new and relatively untested, and some are promising but exist primarily as research results. After considering the capabilities of a variety of tools, we identify in Section 6 a number of research and development challenges that can drive the state-of-the-art in access control, and improve the ability of organizations to prevent insider attack.

5.1 Passwords: Knowledge-Based Authentication

Perhaps the first thing we think about when it comes time to implementing access control in an enterprise is authenticating the users. For many users, the most obvious approach is to authenticate via something one knows---and this is typically a *password* or longer *passphrase*. (Other knowledge-based schemes exist, but passwords are perhaps the most common in the workplace.) From the point of view of developers, password-based authentication has many nice properties. It's a well-understood technology with many mature software tools (ranging from simple OS utilities to Kerberos [Neuman, 1994] and beyond); it's easy to implement; and users understand it. However, it has many downsides as well.

One of the main ones is that strong passwords can be hard for users to remember. This leads to no end of security problems. Unless forced otherwise, users will pick weak passwords that are easy to remember. Users will re-use the same password for many accounts. Users will help themselves remember stronger passwords by writing them down on post-it notes or on the back of keyboards. (An information security manager at a large firm at risk of insider attack noted that, when strolling through a workplace, would nearly always find passwords written underneath keyboards.) Users will forget passwords they don't use often; enterprises forcing users to change passwords regularly incur increased help desk costs shortly after password-change day.

An aspect of passwords that is either bad or good (depending on one's point of view) is the fact that users can easily share them other users (and often do---for chocolate [BBC, 2004], for plastic dinosaurs, squirt guns, or just because someone somewhat official asks [Smith, 2004]). From a strict security perspective, this is bad: an access control policy doesn't do much good if an enterprise can never be sure exactly "who" a particular user really is. However, from a business perspective, this can be good: when an end user needs to delegate some privilege to a colleague, she can easily do so. (When users confess password-sharing to us, it's always to achieve a reasonable business goal; it's just that breaking infosec policy was the most efficient or perhaps the only way of doing it.) Of course, using passwords for delegation has security drawbacks---users give away everything, not just the privilege in question, and do so in a way that is not easily revocable or auditable.

Providing de-authentication in password-based systems is challenging, yet vitally important in environments of mobile users and shared workstations. (One senior medical colleague reports that, in the beginning weeks of their training, new interns find that much embarrassing email---as well as requests for particularly undesirable "on-call" hours---are generated from their email accounts if they fail to de-authenticate in some departments.) One approach to de-authentication might be to require Alice to type her password in every time she wishes to take an action; however, this would quickly become unusable in the majority of corporate environments. Another mechanism that is commonly used is a timeout, whereby the user's login session is terminated after a fixed period of inactivity. IT managers in some domains lament that no single *timeout* is correct across the organization; any value they choose will present an unacceptable hassle cost to some segment of the user population. It would seem that alternative de-authentication solutions, such as those in the biometrics and tokens discussions below, are necessary in domains where timeouts are insufficient.

5.2 Biometrics: Physiology-Based Authentication

In addition to identifying users based on "something they know," another approach is to authenticate users via "something they are." (Security textbooks also teach multifactor authentication: authenticating users by using more than one of these approaches to provide extra assurance.) Common techniques here include fingerprints, hand geometry, voice recognition, and even retina and iris imaging. In theory, biometrics have usability and security advantages. It's much harder for a user to forget a thumb than a password; it's also much harder to lend one's thumb to a colleague for a while. However, there are downsides as well. The effectiveness of the biometrics always seems to be in doubt; it seems vendors tend to claim stronger reliability than reality. Users also can find them intimidating or awkward to use (e.g., Sasse 2007). Another issue in many enterprise settings is whether a user's biometric will always be available or readable. How does one do fingerprint recognition or voice recognition on a masked and gloved medical technician in an operating room?

Biometric methods of de-authentication can detect when a human body is no longer present (and thus trigger the logout of the affiliated user). For example, pressure-sensitive mats, body heat sensors, and proximity detectors might all be useful solutions. Unfortunately, the same IT managers for whom session timeouts were not successful also experienced difficulties getting commercial proximity sensors to work effectively; the sensors either lead to false negatives (logging the person who temporarily stepped out of range) or false positives (leaving departed Alice logged in because her machine faces a busy corridor). This kind of technology might be more useful in domains where users did not need to step briefly away from the computer, or where computers were not surrounded by so much traffic.

5.3 Tokens: Possession-Based Authentication

In the standard security textbook mantra, the third main approach to authenticating users is via a *token*: something they possess. Token-based authentication is common in many workplace environments: employees carry and display badges, or carry identification cards in their wallets. These tokens often can directly interact with the enterprise's IT infrastructure: for example, a badge might have a machine-readable bar code, or use *radio frequency identification (RFID)* to identify itself without physical contact. (The RFID approach raises some interesting opportunities and privacy challenges, because the enterprise can easily interact with an employee's token without the employee even being aware. This can help the enterprise find that critical manager when they need her; however, a perceived loss of privacy can also negatively impact employee morale.) Tokens can also interact over direct electronic connections; *smart cards*---credit-card-sized cards with small integrated circuits---communicate over standard electronic contacts, whereas USB devices utilize the common device interface to connect to computers. Newer technology such as *Bluetooth* can move this more involved interaction to radio.

Some tokens automatically enable de-authentication because of how they communicate. USB tokens must be plugged in to a computer for them to be used; once they are removed, the computer knows the user has finished his session. Similarly, a computer who performs authentication using RFID badges can recognize when a given badge is no longer within range. Of course, de-authentication in these examples requires the user to remember to take her token with her when she leaves; however, corporate users frequently require ID badges, keys, or other objects to do their jobs, so such a requirement seems reasonable for many environments.

5.4 PKI: Authentication via Digital Certificates

Discussion of enterprise authentication can also broach the topic of *public key infrastructure (PKI*). In order for an enterprise to use the public key techniques of Section 3, it needs supporting glue---public key *infrastructure* is the term used for this glue. Typically, PKI begins with a *certification authority (CA)* issuing a *certificate* that can tell Bob what Alice's public key is---that is, if Bob believes such statements from this CA. A tricky aspect of PKI is *revocation*: declaring that a particular certificate should no longer be considered as valid. Many standard revocation techniques exist and are deployed; the primary approaches are, before accepting a certificate, to check a published (but perhaps outdated) list of revoked certificates, or to check with an *online certificate status protocol (OSCP)* service. In the field, many IT managers still regard it as a not-completely-solved problem. In military and industry deployments, the bandwidth necessary for unexpectedly huge *certificate revocation lists (CRLs)* almost crippled networks. As for OCSP, why bother having public key certificates if one has to check with some backend *directory* every time to see if a given certificate is still valid? Revocation---as well as the problem of key mobility---is a significant hassle.

Some enterprises use PKI explicitly: setting up keys for their users and educating them about their use. (Since humans tend not to be good at cryptography, these keys hide within other devices, such as the USB tokens discussed above, or even exist protected within the user's computer in a software *keystore*.) Other enterprises use what we term "stealth PKI": using PKI foundations to enable other authentication techniques, such as smart cards or badges, but hiding the existence of the underlying PKI from the users.

Current PKI tools operate almost exclusively on *identity certificates* (usually in the *X.509* format [Housley 2002]), which bind user identities to public keys. X.509 permits *attribute certificates* binding other properties instead; X.509 can also permit an end user to create a special *proxy certificate*. However, common Internet tools do not support these alternatives gracefully; the inability of X.509 in practice to speak about things other than names, and to let end users do spontaneous delegation to each other, are significant obstacles to using PKI to solve the access control problem in enterprises.

5.5 Distributed Authentication and Identity Management

For much of the history of access control, users' identities have served double duty as both authentication and authorization credentials. As such, the community has developed extensive technology for *identity management* tasks, which include adding, changing, removing, and auditing user accounts and their associated access privileges. (Systems that decouple authorization from identity also manage additional information; we discuss distributed authorization in sections below.)

Centralized identity management allows an enterprise to streamline access control operations across distributed systems: instead of resources *A*, *B*, and *C* all having independent set of accounts, credentials, constraint policies, and administrators, they share a central database that contains up-to-date user information. This database often takes the form of a *directory*. Standards movements in the early history of the Internet yielded the X.500 directory services specification [Chadwick 1994]). The X.500 community envisioned a global "directory in the sky," which would act as an omniscient phonebook and include all the information needed to securely communicate with anyone, anywhere, including users' public key certificates.

Scalability and privacy issues prevented this global directory from becoming a reality, but the concept lives on in the form of *Lightweight Directory Access Protocol (LDAP)* directories. A number of software vendors (as well as open source software groups, such as OpenLDAP) offer LDAP products; in particular, Microsoft's Active Directory (AD) seems to have a large market share among corporate collaborators, largely because it thoroughly integrated with the Windows operating system.[1]

Solutions like AD can act as a repository for a variety of identity-based authentication systems. In theory, AD can integrate with Kerberos as well as PKI implementations. (In practice, PKI implementation in AD is still too immature to be sufficiently usable and reliable for most corporate environments.)

When deployed widely in an organization, both systems like Kerberos and those based on a PKI can offer *single sign-on (SSO)* capabilities to end users. For example, once a user has authenticated[2] to Kerberos via AD, the server issues her computer a Kerberos ticket; she can then use the ticket to authenticate to a number of resources within the company. A PKI user who keeps his credentials in a software keystore or hardware device only needs to unlock it once in order to use his private key multiple times on a single machine. Transparent SSO can boost productivity and lower hassle cost, but requires that the de-authentication problem be addressed: how do we prevent another user from coming along and using the

[1] We note also that much Active Directory functionality is outside the LDAP specification; it is billed as an LDAP-*compliant* general directory service.

[2] Of course, the same risks posed by passwords, biometrics, or smartcards apply when those credentials are used for widespread distributed authentication; credential compromise in such settings poses tremendous risk for the organization.

ticket or unlocked keystore? This is less of a concern with hardware devices, at least once users gain the habit of keeping the device on their person. Portable keystores also provide *credential mobility*, which can help improve usability in environments such as hospitals---again, as long as users succeed in keeping their credentials with them at all times. However, another aspect of this problem is beginning to receive much attention: the ability of a malicious Web site to quietly borrow credentials from a user's keystore, software or hardware.

5.6 Distributed Authorization

As noted before, traditional access control products have viewed *authorization* almost exclusively in terms of *authentication* of a user's identity by a resource owner. However, in large environments like enterprise IT the picture is often more complex. For one thing, the "resource owner" may not necessarily be in a position to decide whether the user in question should receive access; database administrators are rarely qualified to approve an investment banker's access to certain account data, although the banker's supervisor may be sufficiently informed. For another thing, scale and complexity requirements often require additional layers of abstraction beyond a user's identity, such as roles, to manage the company's compliance with regulatory access constrains. Furthermore, the maintenance that goes into keeping such an infrastructure running is often complex beyond an individual human's understanding.

Trust Management Systems

Researchers encountering issues associated with distributed authorization have proposed extensive *trust management* infrastructures, such as PolicyMaker system [Blaze et al., 1996] and its successor KeyNote [Blaze et al., 1999]. Essentially, these systems develop formal ways to express access control policies and formally evaluate whether the requester's credentials merit authorization.

Privilege Management Infrastructures (PMIs)

The PKI community has also developed tools to adapt to the modern complexities of distributed authorization. As noted earlier, generic PKIs operate on identity certificates that bind public keys to identities; a *Privilege Management Infrastructure (PMI)* instead has *attribute authorities* (instead of CAs) issue attribute certificates that bind public keys to other attributes, such as "Employee," "Student in CS 101," "The Dean's Assistant," or even "Bob says is permitted to see record X." PERMIS---an academic project that has been piloted in some European civic applications and in GRID distributed computing---is a good example [Chadwick, 2002].

Distributed Policy Decision and Enforcement

Consequently, we see architectures for distributed authorization emerge, with *policy enforcement points (PEPs)* consulting *policy decision points (PDPs)* about whether to grant a request, and with policy languages---such as the *eXtensible Access Control Markup Language (XACML)* to express these policies

Active Directory

Many commercial tools have evolved to include features to address these complexities. For example, Active Directory supports a wide variety of capabilities beyond that of a traditional directory of users. Administrators can create *Organizational Units (OUs)*, which are composed of users, security groups, computers, and other OUs. *Group Policy Objects (GPOs)* allow AD administrators to manage privileges of both users and computers assembled into logical groups. GPOs interact with the Windows operating system to enforce constraints on file system access, trust policies (for example, what PKI certification authorities to trust), and application usage. The recursive nature of OUs allow administrators to define hierarchies to facilitate privilege management throughout the enterprise; Active Directory also provides a scripting capability to streamline tedious tasks associated with user and group privileging. By specifying a user, computer, group, and/or GPO, administrators can both audit the effective privileges in place and model the effects certain modifications would have.

However, the flexibility offered by Active Directory seems difficult for enterprises to harness. Richards et al. discuss in their administrative guide [Richards et al., 2006] the business case for migrating to an AD-based system: “Will it reduce your Total Cost of Ownership (TCO)? It sure will, but only if you design it correctly. Design it the wrong way, and you'll increase costs.” The authors quantify the scalability of the OU/GPO infrastructure by noting that Microsoft recommends organizational hierarchies of depth at most 10, and by relating their personal observations of significant slowdown when more than 12 policies are applied. This latter estimate is supported by the experience of one of our partner companies, who laments that their single greatest source of hassle cost is the lag users experience between logging in and actually being able to use a workstation. (This hassle also feeds into users’ unwillingness to voluntarily de-authenticate from a computer they are likely to use again in a short time.)

Role Engineering and Management

Most modern access control solutions implement some form of RBAC (described in Section 3.5). Although support for more advanced capabilities---like role hierarchies---has been rare, many vendors are introducing new features to meet regulatory and risk-mitigation demands. With the addition of these much-desired features, however, has also come the need for new technical solutions to role engineering and management problems.

As outlined earlier, the process of defining roles for an organization can be top-down, bottom-up, or a hybrid solution of the two. The first option is traditionally process-based, light on technology but heavy on interviews and scenario building. Our collaborating practitioners as well as a number of distinguished RBAC experts [Ferrailio et al., 2007] find top-down definition to be both tedious and difficult---if not impossible---in some scenarios. Vendors such as Bridgestream (recently acquired by Oracle, http://www.bridgestream.com/) and Eurekify (http://www.eurikify.com) offer a suite of applications to facilitate continuing role management as well as initial bottom-up role engineering. Products in this space include solutions for *role mining* or *role discovery*, whereby a clustering algorithm generates a set of candidate roles from users' existing privileges. Blindly clustered sets run the risk of becoming quickly outdated as the organization changes over time, however, and vendors are refining their approaches to include additional sources of information, such as organizational hierarchies or users' job titles. One financial institution we partnered with has dedicated significant resources to studying the capabilities of products in this space, and while the institution reports that they are going in the right direction, it also laments that most solutions are too immature and unproven to be deployable in live organizations.

Federation
The above discussions implicitly assumed that, even if distributed, authorization decisions needed to be made within the scope of a single enterprise. In practice, users and resources may be distributed across multiple enterprises, which raises the issue of how to *federate* different authorization systems. In the academic space, *Shibboleth* (http://shibboleth.internet2.edu/) is well-known example, used to allow individual institutions to maintain their own legacy ways of authentication, but let their users access resource at remote institutions. The *Liberty Alliance* (http://www.projectliberty.org/) is mainly industrial consortium devoted to open standards for Federation.

6 Ongoing Challenges

Thus far, this paper has surveyed existing research and development in access control, with a particular focus on the applicability of this work in preventing insider attacks in large corporate environments. Section 2 defined the specific insider threat model against which we aim to defend; Section 3 presented background principles of access control to elucidate the theoretical capabilities of these systems. Section 4 presented the functional, cost, usability, scalability, and complexity requirements that the threat model demands, and Section 5 surveyed some current access control tools (both research projects and commercial products). Overall, both the theory behind access control and the systems that implement it seem to be well developed.

Nonetheless, even with these principles, we still have an insider threat problem; armed with these tools, our colleagues in the trenches still report an inability to have accurate IT policy in practice. The natural response is: Why? Do the tools fail to accommodate some critical aspect of the real-world requirements? Have the basic principles overlooked something? Is it just a matter of economic incentive for a vendor to bring the right technology to the right market? To use the terminology of Section 2, would a “practically correct” access control system even reduce the incidence of insider attack? Is such an access control system possible? If not, what are the limits that bind it, and how close to the ideal can we get?

6.1 A Snapshot of a Motion Picture

Accurate access control policy requires an accurate vision of what users, roles, and permissions should be. However, real-world enterprises report that the dynamic nature of the real world makes it hard to capture a clear vision.

As discussed earlier, organizations attempting to deploy role-based access control solutions can choose top-down, bottom-up, or hybrid approaches to role definition. Bottom-up clustering algorithms succeed in grouping users using existing privilege assignments and limited organizational information, but it is not yet clear whether this approach generates roles that will be useful throughout the various changes employees and enterprises undergo. Hybrid solutions integrate the strengths of the top-down approach: build on existing task- or requirement-oriented practices to define roles, which although tedious and expensive in personnel costs, results in role sets that will gracefully evolve with the organization. Unfortunately, many organizations (often those renowned for their agility and adaptability to new business climates) are founded on dynamicism and matrixed structure; any product of a top-down methodology will be out of date before the process is finished!

Technologists at companies who experience such dynamicism thus report being forced to choose between roles that are difficult to manage (and likely to become inaccurate quickly), and roles that are more likely to evolve with the firm, but start off being incorrect at their initial deployment. We thus wonder, what approach should these enterprises take to role engineering? What ways can solutions integrate top-down and bottom-up methods to generate roles that are both accurate and that will evolve with the organization? Finally, what further research and development is necessary before vendors can realize such integration in commercial products?

6.2 Privilege Issuance and Review

In addition to the roles or other structures necessary to manage its access control system, an organization must also define the ways in which users acquire privileges, and design methods to effectively audit users’ privilege sets for correctness. Some organizations report tremendous difficulty identifying which manager or administrator should be in charge of privilege issuance to a given set of users (maybe Anya is best qualified to decide whether or not a given doctor needs a certain privilege, but Sergey knows best when it comes to nurses). Defining rules for practices like delegation or temporary privilege assignment are even more challenging, yet some, like break glass, are vital to an organization's ability to meet its most fundamental goals. (Of course, additional flexibility in privilege issuance results in less supervised control over the process, which in turn can increase risk.) Access control tools often allow such features in theory, but actual attempts to distribute issuance to end users seem to often fall short of desired functionality.

Beyond the challenges of privilege issuance, colleagues report that many managers experienced unanticipated difficulty in verifying the correctness of a user's privilege set. Although Andy may be Liz's supervisor on a given project, that doesn't mean that he will be able to review a list of her privileges and be able to identify the subset necessary for her to complete her assigned project tasks. Indeed, one organization reported that users were not able to identify which of their privileges were essential to their *own* jobs. How can we improve privilege review technology to better enable these vital business practices?

6.3 Auditing and Visualization

Privilege review allows an organization to verify correctness of one aspect of their access control system. However, corporate partners lament the lack of tools to help them maintain a broader understanding of the system's operation, and of the subtle effects of different policies. Enterprise-wide access control systems, especially those that implement privilege constraints like separation of duty, can be more complex than any computer network or technical program; we do not ask administrators to verify network architecture correctness by watching traffic flows, so how can we expect them to perform similar tasks in access control? Desired functionality in this space includes high-level cost assessment (both monetary and hassle), risk evaluation, troubleshooting assistance in case of improper privileging, and capability modeling to understand the impact a given set of policy changes would have on the enterprise's ability to meet its goals. Effective solutions to these problems will likely involve significant work in interface design and usability testing.

6.4 Role Drift and Escalation

An access control policy must adapt to organizational changes to maintain correctness over time. Enterprises that deploy role-based systems must ensure that roles are properly assigned to users, but must also make sure that roles contain proper privileges as new resources become available and old ones are phased out. How can technologists identify when a role is "drifting" away from its original definition, when it is appropriate to split or merge roles, or even when a new round of role discovery and definition is warranted?

Some domains require users to have the ability to escalate their privileges in certain situations; for example, health care professionals talk extensively about “break-glass” features. How can an organization implement this escalation requirement while still limiting insider threat? How can it evaluate tradeoffs in balancing the need to get the job done (where failure can mean dire repercussions) with the risk of insider attack that uninhibited access poses?

6.5 Expressiveness and Need to Know

As noted, many researchers assume that, by definition, the appropriate policy exists. Others assume that “proper authorization” will always allow “unauthorized action,” and proceed to define insider attack that way: unauthorized action by authorized individuals. Both views suggest (conflicting) assumptions that should be questioned. We have already seen that “correct” polices tend to not to exist in the field. Are they even possible? Alternatively, why is it that “unauthorized action” cannot be restricted by better authorization? There seems to be an implicit assumption that the behavior exploited by insiders cannot be expressed within a policy language. Is that true? If not, how closely can feasible IT policies approximate the “true” policy?

6.6 Incentives

It’s probably true in general that experts in an area implicitly assume that the entire population shares their belief in that area’s value and importance. Computer security is no exception. Reports from real users in real-world enterprises show a diversity of opinions about whether techniques such as authentication, access control, and general security hygiene help or hinder users getting their jobs done. To the extent that users perceive a lack of alignment between security technology and their core mission, security technology will not be effective.

This line of thinking suggests that research is needed into why this perceived misalignment exists. Would it be solved by better user education? Or is the technology itself fundamentally misaligned in some way?

7 Conclusions

In the previous section we considered a number of specific challenges; we believe that these issues are representative of the types of difficulties our partners in industry have had in using access control to prevent insider threat. It is not clear at this time whether current solutions can meet these challenges as they stand, whether we need new development efforts based on current principles, or whether we need new lines of research altogether. In any case, additional work in this space will almost certainly improve the ability of large enterprises to defend against the insider threat; unfortunately, as the authors of other chapters remark, it is difficult for researchers to gather actual data from real-world partners to inform this development effort. Technologists and corporate policy makers are happy to share anecdotes and general trends, but hesitant to offer attributable facts or hard information that could pose reputation risk.

Despite the difficulty in obtaining hard data, work in this space continues; in looking forward, we wonder: what degree of mitigation can we eventually hope to achieve? Can all insider threat be prevented with well-designed access control mechanisms? We conjecture to the contrary that no access control mechanism alone can protect against attacks executed by trusted individuals *using only the privileges deemed necessary to get their job done*. Other measures that offer disincentives against abusing their privileges can mitigate the threat, but there are some scenarios in which insiders must be trusted to use their better judgment in their interactions with electronic resources.

The concepts, survey, and ideas presented in this paper deals with insider attack prevention, whereas much current work in this space (indeed, most of this book) focuses on detection. We intend our focus on prevention to complement, not replace, that the detection efforts. Better prevention can simplify the problem space that detection must address; we recall the early history of the theory of safe systems [Harrison et al., 1976], where "detection" was in fact not computable until the problem space was constrained. History offers hope.

Acknowledgments

This chapter describes work resulting from a research program in the Institute for Security Technology Studies at Dartmouth College, as well as under the auspices of the Institute for Information Infrastructure Protection (I3P), which is managed by Dartmouth College. Both programs are supported by the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security, the I3P, or Dartmouth College.

In addition to our grant sponsors, we would also like to thank our partners in the financial, health care, and other industries, who must unfortunately remain anonymous here. We also thank Christopher Masone for his willingness to perform a late-breaking review of this chapter.

References

[1] American Dental Association. “Insurance: Medicare and Medicaid,” *ADA Official Website*. http://www.ada.org/public/manage/insurance/medicare.asp.

[2] Anderson, R. E. “Matrix Redux,” *Business Horizons*, Nov.-Dec. 1994, 6-10.

[3] Blaze, M; Feigenbaum, J.; Ioannidis, J.; and Keromytis, A. “The Role of Trust Management in Distributed Systems.” *Secure Internet Programming*. Springer-Verlag LNCS 1603, pp 185-210. 1999.

[4] Blaze, M.; Feigenbaum, J.; and Lacy, J. “Decentralized Trust Management.” *Proceedings of the 1996 IEEE Symposium on Security and Privacy*. pp. 164-173.

[5] British Broadcasting Corporation. “Passwords Revealed by Sweet Deal.” *BBC News*, UK Edition, April 20, 2004. http://news.bbc.co.uk/1/hi/technology/3639679.stm.

[6] Burns, L. R. and Wholey, D. R. “Adoption and Abandonment of Matrix Management Programs: Effects on Organizational Characteristics and Inter-organizational Networks.” *Academy of Management Journal*, Vol. 36, 1, 106-139.

[7] Chadwick, D. “The PERMIS X.509 Role Based Privilege Management Infrastructure.” *7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*. 2002.

[8] Chadwick, D. 1994. *Understanding X.500: The Directory*. London: Chapman & Hall, Ltd.

[9] Chadwick, D.; Otenko, A.; and Ball, E. “Role-Based Access Control with X.509 Attribute Certificates.” *IEEE Internet Computing*. March-April2003.

[10] *Department of Defense Trusted Computer System Evaluation Criteria*. DoD 5200.28-STD. December 1985.

[11] Donner, M.; Nochin, D.; Shasha, D.; and Walasek, W. “Algorithms and Experience in Increasing the Intelligibility and Hygiene of Access Control in Large Organizations.” *Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security*. Kluwer, 2001

[12] Ferrailio, D.F. and Kuhn, D.R. “Role Based Access Control.” *15th National Computer Security Conference*. 1992.

[13] Ferrailio, D.F.; Kuhn, D.R.; and Chandramouli, R. 2007. *Role-Based Access Control*. Norwood, Massachusetts: Artech House Publishers.

[14] Harrison, M.A.; Ruzzo, W.L.; and Ullmann, J.D. “Protection in Operating Systems.” *Communications of the ACM*. 19(8): 461—470. 1976.

[15] Housley, R.; Polk, W.; Ford, W.; and Solo, D. 2002 *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. Internet RFC 3280.

[16] Lampson, B.W. “Protection.” *ACM Operating Systems Review*. 8(1): 18—24. January 1974.

[17] NIST. *Role Based Access Control*. http://csrc.nist.gov/rbac/

[18] Neuman, B. C. and Ts'o, T. “Kerberos: An Authentication Service for Computer Networks.” *IEEE Communications*,. 32(9):33-38. September 1994

[19] Povey, D. “Optimistic Security: A New Access Control Paradigm.” *Proceedings of the 1999 New Security Paradigms Workshop*. 40-45.

[20] Richards, J. ; Allen, R. ; and Lowe-Norris, A. G. *Active Directory*, Third Edition. O’Reilly Media, January 2006.

[21] Sandhu, R.S.; Coyne, E.J.; Feinstein, H.L; and Youman, C.E. “Role-Based Access Control Models.” *IEEE Computer*. 29(2): 38—47. 1996.

[22] Sasse, M.A. “Red-Eye Blink, Bendy Shuffle, and the Yuck Factor: A User Experience of Biometric Airport Systems.” *IEEE Security and Privacy*. 5(3): 78—81. May/June 2007.

[23] Smith, S. W. “Probing End-User IT Security Practices---via Homework.” *The Educause Quarterly*. 24 (4): 68—71. November 2004.

[24] Smith, S. W.; and Marchesini, J. 2008. *The Craft of System Security*. Indianapolis, Indiana: Addison Wesley Professional.

[25] Weeks, S. “Understanding Trust Management Systems.” *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. pp. 94-105.

Taking Stock and Looking Forward – An Outsider's Perspective on the Insider Threat

Jeffrey Hunker

Carnegie Mellon University

Abstract Despite considerable work over the last decade, the research community has made little overall progress in solving or even reducing the insider threat. This conclusion stems not from lack of research quality but rather from lack of a framework to understand exactly what problem we are trying to solve. In this chapter I suggest that there are some fundamental questions relating to the insider threat that either have not been posed, or have been posed without sufficient rigor to motivate work towards meaningful solutions. Among these questions are:

- What exactly is an 'insider threat'?
- How does the research community get the needed data to create solutions?
- In role do the incentives organizations face in the solution space, and how do we change these incentives?
- How do we better integrate technical solutions with social science perspectives?
- Should there be a response and recovery system for insider threats, and what form would it take?

This list of fundamental questions is by no means exhaustive, but the hope is that the reader will be motivated to pursue these and other key questions with the goal of providing greater rigor in our work – and thus make more progress in effectively addressing aspects of the insider threat problem.

1 Introduction

The insider threat problem appears to be serious, and certainly long studied. In the 2007 CSI Computer Crime and Security Survey, 59% of respondents reported that they had experienced insider abuse of network resources; 26% reported that over 40% of their total financial losses from cyber attack were due to insiders. [1] The damage done to the Nation by one notorious insider, the FBI agent Robert Hanssen, is incalculable. The close cooperation by senior executives in key sectors, like banking and finance, with researchers working on the insider threat problem in itself suggests that insider misuse remains a serious challenge to these sectors.

A great deal of research has been done to examine the nature of the insider threat, with the hope that eventually the problem can be reduced or even completely solved. Starting in 1999, RAND conducted a series of workshops to elucidate the necessary research agenda to address this problem. [2] In parallel, the Defense Department produced its own report[3], outlining both a set of policy changes and research directions aimed at addressing the insider threat. Since then, a rich literature studying various aspects of the insider threat problem has emerged.

I come to the 'insider threat problem' as an outsider, having been only recently introduced to the problem. Having reviewed the work of the past decade, I unfortunately must conclude that, despite the considerable research, the research community has made little overall progress in solving or even reducing the insider threat. Such a dismal conclusion is shared by a number of my colleagues who have longer histories in working the problem and even greater familiarity with various aspects of the field. This conclusion stems not from lack of research quality but rather from lack of a framework to understand exactly what problem we are trying to solve.

On stepping back to survey past work, I find that it falls into three categories. The first category views the problem from the practitioner's point of view. Here, the malicious or misguided insider is a serious threat, described by recent case examples whose characteristics suggest what systems administrators and senior ex-

[1] Computer Security Institute, 2007. Computer Crime and Security Survey. pp. 12-13.

[2] Anderson, R.H., 1999. Research and Development Initiatives Focused on Preventing, Detecting, and Responding to Insider Misuse of Critical Defense Information Systems: Results of a Three-Day Workshop. RAND CF-151-OSD.

This conference was followed by two others; see:

Anderson, Robert H., Bozek, Thomas et al., 2000. Research on Mitigating the Insider Threat to Information Systems #2. Proceedings of a Workshop Held August 2000. RAND CF-163-DARPA.

Brackney, Richard C. and Anderson, Robert H., 2004. Understanding the Insider Threat: Proceedings of a March 2004 Workshop. RAND CF-196-ARDA.

[3] DoD Insider Threat Mitigation. Final Report of the Insider Threat Integrated Process Team. April 24, 2000. US Department of Defense. Office of the Assistant Secretary of Defense (Command, Control, Commuications, and Intelligence).

ecutives need to know and do[4] [5]. The second category reflects the interests of the research community. A body of work examines the psychological aspects and motivation of the insider; that is, why and under what circumstances an insider becomes an insider threat.[6] For example, the CERT and Secret Service used access to incarcerated individuals to study the personal characteristics and circumstances of individual insider malefactors.[7] [8]This information laid the groundwork for an understanding of the interpersonal and organizational dynamics that eventually produce an insider threat. The third category involves research that has a strong base in computer science. It examines the properties and attributes of information technology systems, asking what observables within the system and what properties of the system can be used or altered to reduce the insider threat. The RAND 2000 and 2004 workshops [9] exemplify this focus. More generally, in computer science, a great deal of work has been directed at determining which aspects of systems policy, monitoring, and detection can be directed towards preventing, detecting, and ameliorating insider threats.

Yet my colleagues and I conclude that little real progress has been made in reducing the insider threat. Why is this? I suggest that there are fundamental questions relating to the insider threat which either have not been posed, or have been posed without sufficient rigor to motivate work towards meaningful progress. My goal in this paper is to probe more carefully into facets of some basic questions and observations, with the hope that this probing may help stimulate further progress in the field.

[4] See for example Wagner, Mitch 2006. Protecting Against Insider Threats. Information Week's Security Weblog. Dec. 11, 2006.

[5] See for example Cole, Eric 2006. Insider Threat: protecting the enterprise from sabotage, spying and theft. Syngress; distributed by O'Reilly Media.

[6] See for example Band, Stephen R., Cappelli, Dawn, et al. 2006. Comparing Insider IT Sabotage and Espionage: A Model Based Analysis. Carnegie Mellon University. Software Engineering Institute. Technical Report CMU/SEI-2006-TR-026. December 2006.

[7] Keneney, Michel, Cappelli, Dawn, et al. 2005. Insider Threat Study: Computer System Sabotage in Critical Infrastructure Sectors. Carnegie Mellon University. Software Engineering Institute. May 2005.

[8] Randazzo, Marisa Reddy, Cappelli, Dawn, et al. 2004. Insider Threat Study: Illicit Cyber Activity in the Banking and Finance Sector. Carnegie Mellon University. Software Engineering Institute. August 2004.

[9] Anderson (2000) and Brackney (2004)

2 What Is An "Insider Threat"?

An insider is defined as an individual with privileged access to an IT system. An insider threat is an individual with such privileges who misuses them or whose access results in misuse.[10]

On the surface, these definitions seem satisfactory. Indeed, in the halcyon era of mainframe computing, they may well have defined a problem of substantive interest to both practitioners and researchers. When machine access was relatively limited and the tasks performed on IT systems were well defined, delineating the scope of the insider threat problem was feasible.

But consider the range of insider threats in today's environment as described by the examples below, each of which is based on real incidents:

- Employees regularly use the organization's e-mail and web server for personal purposes, even though security policy forbids anything but business use of the system;
- A faculty member lends his password to a trusted friend so that she has access to the on-line university library resources in order to assist in his research;
- An investment banker exploits an accounting anomaly in the way in which profits on trades are recorded to report large profits; in reality no such profits were created through the trades. No system privileges were violated;
- A senior executive uses his position to override the e-mail system controls and monitoring mechanisms so that he can send and receive illicit messages without their being logged into the system's memory.

Each of these is an insider threat. All would agree that the last case cited is of concern – system privileges were abused, and bad results occurred. What about the other cases? In the case of the investment banker, the crime could have equally well have been committed in a paper based system. With the faculty member, the intent was to work around burdensome administrative requirements in order to perform the primary task (research). System privileges were abused, but the result was positive for the organization (better research). And how much effort is appropriate to stop employees from sending personal e-mails (fully recognizing the productivity losses that sometimes can result from such action)?

The point here is not to suggest that some of the above incidents are true 'insider threats' and others not. Rather, the point is that our definition of the insider threat does not capture the nuances of the insider threat problem. Like pornography, we know the insider threat when we see it.

The definition of the insider threat problem matters: if one cannot define a problem precisely, how does one approach a solution, let alone know when the

[10] This is the definition now being used by a RAND team, of which the author is a member, conducting research on insider threats.

problem is solved? Different definitions embody alternative assumptions and perspectives of trust relationships in people and systems, as well as on the possible suite of countermeasures available. If we do not rigorously define our terminology, we cannot determine what data we want to capture.

Precision in defining the insider threat is challenging. Supplementing the definition at the start of this section, Bishop [11] cites three definitions used for the insider threat used in different papers:

- 'malevolent (or possibly inadvertent) actions by an already trusted person with access to sensitive information and information systems;'
- 'someone with access, privileges, or knowledge of information systems or services;'
- 'Anyone operating inside the security perimeter.'

Each definition seems reasonable, but each embodies different interpretations of the 'insider threat' with implications on assumptions of trust in systems and people, as well as on the countermeasures to be taken. In the first definition, the insider is trusted in some way not to abuse the information or system. The second definition would broaden the insider definition to include anyone who knows something specific about the system, whether the individual has access or not. The third definition would include the building's janitor.

As Bishop notes, "the insider problem is not 'a' problem. Rather, it is a continuum of problems, ranging from the case of the rogue user with little to no privileges, to the case of an official with a large number of privileges."

Hence the appeal of more colloquial descriptions of the 'insider threat,' such as the following from Schneier [12]:

> A malicious insider is a dangerous and insidious adversary. He's already inside the system he wants to attack, so he can ignore any perimeter defenses around the system. He probably has a high level of access, and could be considered trusted by the system he is attacking.

This description of the problem, while not useful as a problem definition, does point towards several observations.

First is the importance of considering what is meant by insider. Consider three different insider threat formulations. First is that the insider is Jeffrey, a physical person, who has legitimate system access. Alternatively, the insider could be someone else masquerading as Jeffrey on the system, perhaps because Jeffrey left his workstation logged in allowing such access. A third formulation would have Jeffrey no longer part of the organization (hence no longer officially having system privileges) but someone forgot to turn off Jeffrey's system access. Each of these is clearly a threat and each is a very different problem to solve.

[11] Bishop, Matt 2005. The Insider Problem Revisited. Proceedings of the 2005 workshop on New Security Paradigms. ACM. 2005. Pp. 75-76. DOI= http://doi.acm.org/10.1145/1146269.1146287

[12] Schneier, Bruce 2000. Secrets and Lies: Digital Security in a Networked World. Wiley Publishing 2000. p. 47.

Second is the need to disentangle the distinctions between abuse or attack, misuse or accident, and good and bad intent. For example, consider two alternative cases:

1. The system quite legitimately allows access, but bad results occur: The example cited above of the investment banker exploiting accounting anomalies to commit massive financial fraud is based on the experience of Joseph Jett, a New York investment trader. He determined that in executing a complex series of Treasury bond trades the bank's accounting system would record an artificially inflated profit; later he exploited weaknesses in the audit and oversight process to record trades that never took place. No system privileges were exceeded, and the entire fraud arguably could have taken place under a paper based system.

Recent press articles have suggested that systems administrators commonly use their privileges to read files (e.g., salary information) motivated by salacious curiosity. That they should not do so is obvious, but no system access privileges were breached; one might argue about whether anything bad results.

2. System privileges are violated, but good things happen: During the Vietnam War, Daniel Ellsberg, a RAND employee, photocopied a number of sensitive documents and gave them to the New York Times (the Pentagon Papers). This example is not actually an insider threat in the context considered here, but it illustrates powerfully some challenges in definition. Clearly Ellsberg violated his access privileges, but was what he did right or wrong?

Making this distinction between right or wrong certainly occurs frequently in more mundane circumstances. According to a recent RSA study, about a third of workers consciously break enterprise security practices because they want to expedite their work or increase their own productivity. "They're just trying to get their work done… But the compromises that occur as a result can be just as damaging as if they didn't know the policy at all"[13] And emergency situations may require privileges to be exceeded. The key point is that security violations can be 'justified' depending on perspective and context.

Hence the importance of context. If the 'problem' of system workarounds beneficial to the organization is to be distinguished from policy-appropriate use of the system for malicious deeds, a richer problem definition is required. Context could be defined along a number of dimensions -- intent, degree of system access, nature of the damage caused, among others – to create a more rigorous problem definition.

The role of organizational security policy and implementation presents another set of issues in creating an effective definition of the insider threat. Stated simply, to what extent should the concern about insider threats focus on inadequate or poorly thought-through system policy or design? If the organizational system policy allows access, the law says nothing – the access was authorized. Yet many or-

[13] Wilson, Tim. End Users Flout Enterprise Security Policies. Dark Reading, December 10, 2007. http:www.darkreading.com/document.asp?doc_id=141002

ganizations have inadequate security policies, or official policies are not followed in general practice.[14] Distinguishing among these circumstances is again important in deciding what problem exactly is a solution aimed at addressing.

In summary, it is to me striking that relatively little work has been done on the most fundamental question of research in insider threats – what problem exactly are we trying to solve? This discussion is not, as noted, an effort at creating the necessary definition that would provide such clarity; in this discussion I have suggested some dimensions of what might an adequate definition would include. However, if we cannot rigorously define the problem we are seeking to solve, then how can we approach it, or even know when the problem has been solved?

3 How Does The Research Community Get Better Data?

Many reports, and almost all discussions, on the insider threat problem bring forward the lack of data available to study the problem.

The lack of solid data for research is characteristic of the overall field of cyber security threat research. There are good reasons why organizations and individuals are reluctant to report cyber security breaches of any sort. The potential harm to the reporting entity is large – loss of reputation and loss of confidence by customers, suppliers and peers can be significant. Legal requirements, notably Sarbanes-Oxley, requiring senior executives to affirm the validity of the reported financial results, appear to increase the incentive for secure IT systems – but also perversely may reduce the incentive to investigate, or at least report, possible incidents. The potential benefits in terms of informing the community at large of possible threats, and possibly leading towards better security solutions, accrue to the community, not directly to the reporting organization.

The disincentive to share incident data is even greater for insider threats than for external attacks. The prospect that a trusted insider could be malicious is disturbing (if Joe could do that, what about the rest of us…); equally disturbing is the case that the organization's systems were inadequate to prevent even accidental misuse leading to serious consequences. The data available on external attacks is scanty, but it is a rich trove compared to that available on insider threats, and much of that insider threat data is available only to select researchers under highly restrictive conditions.

Hence, anyone seeking to understand the insider threat faces the situation outlined in Figure 1.

[14] Ibid.

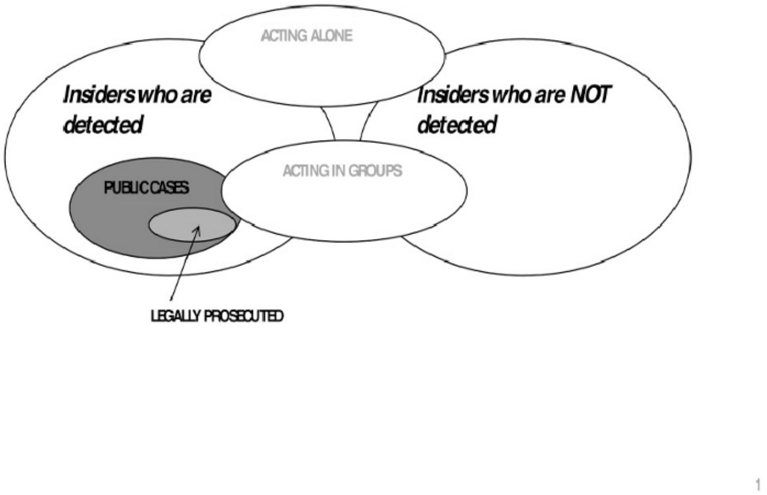


Fig. 1. Our Limited Public View of the Insider Threat

There is the universe of all insider threat incidents – however defined. Some insiders may be acting alone; others may be acting in groups (including outsiders, or all insiders). Of that space, unknown proportions are detected. Of the detected space, a presumably even smaller number (again unknown) are reported, either publicly or to select research projects. Of the publicly reported cases only in some cases are these incidents prosecuted. Only this last proportion (the number of publicly reported cases, versus the number entering the legal system) can be known with any precision, although this value has yet to be determined.

The result is that we have (publicly) two data points – survey data (e.g., the CSI survey, the CSO survey conducted with the CERT and Secret Service[15]) and case files of publicly accessible incidents. Neither is a reliable representation of the underlying population. The surveys are convenience surveys, which lack statistical validity, and may have reporting biases or distortions which we can only speculate about. Case data comes mostly from law enforcement, is collected in an *ad hoc* fashion, and inherently reflects a winnowing process of,(1) detection, (2)

[15] CSO Magazine, United States Secret Service and CERT Coordination Center 2006. 2006 eCrime Watch Survey. Framingham, MA: CXO Media.

seeking law enforcement redress or having the incident made public, and (3) decision to prosecute.

There are several perspectives on this lack of data. The first is to observe that in most cases even with data we may know the numerator, but not the denominator, of the characteristics under study. The CERT/Secret Service research examined in detail the personal characteristics of insiders whose actions lead to criminal prosecution[16]. The insights are illuminating, but there is no baseline of 'non-threatening' insiders against which to evaluate their findings. It might be that the characteristics common to the 'insider threat' individual are in fact common across the subject population. We simply do not know.

Even more fundamentally, if, as was suggested in the last section, we have not yet defined with any rigor exactly what insider threat problem we are seeking to study, then defining what we want to measure, let alone collecting the appropriate data, rests on shaky foundations. We need to get to a point where we can measure real things that actually address the problem we are trying to solve – and as a community we are not at that point yet.

Some speculation as to when insider threat incidents are made public may be useful. An insider threat incident may have a greater likelihood of being reported publicly when it:

- Takes place in a small organization lacking the resources to handle the incident internally;
- (a variant on the above point) When the incident is of such scope as to overwhelm the organization, regardless of size.
- When the object of the insider threat's actions is outside of the organization, such as when an insider misuses the IT system to attack a competitor;
- When the insider threat individual has a relatively low position within the organization;
- When the insider is socially or psychologically viewed by the organization as being outside of the 'old boys club.' This may help to explain the prevalence of insider threat cases involving economic espionage by non-US nationals that appear in a review of court records (although many other factors might contribute to such trends).

We know that organized crime is active in many aspects of external attacks, and there exists a vibrant cyber underworld engaged in the creation of attack tools, their deployment, and for handling the proceeds of attacks.[17] Identity theft (e.g., of credit card information) is a frequent target of outsider attacks. Little or none of this shows up in the reported insider threat data. We might expect that senior executives would misuse, perhaps maliciously, their system privileges. The gulf between what one might reasonably expect to see in the range of insider threats, and

[16] Keneney (2005), Randazzo (2004)

[17] Kursawe, Klaus and Katzenbeisser 2007. Computing Under Occupation. Proceedings of the 2007 workshop on New Security Paradigms (forthcoming). ACM 2007.

what appears in the data, may be large. It is possible that there are entire typologies of insider threats that simply do not appear in the data.

A consideration is to ask whether, as a community, researchers have thought too narrowly about how to obtain insider threat data. Typically researchers either arrange special access for organizational data on insider activity, with an accompanying set of restrictions on use, or else generate data through simulations (games, exercises). Otherwise, oftentimes researchers have to resort to formulating a reasonable problem, exploring solutions, but without a comprehensive recourse to good data.

The insider threat problem is serious, and the data to address it are not easily forthcoming. Perhaps more fundamental and structural solutions to the data problem are needed. The record of related fields points to at least three models for such fundamental and structured solutions. Changes in the law, making the reporting of insider threats mandatory, may present political challenges. Of note are the data breach notification laws (discussed later in this article) now enacted in over 30 states; the passage of these laws suggests that perhaps other mandatory disclosure requirements might, just might, be politically feasible.

A second model would be to expand the reach of the CERT network to more specifically focus on insider threats. The CERT network has built for itself a reputation as a trusted resource for reporting software and configuration vulnerabilities, in turn providing assistance to affected organizations while facilitating the creation and distribution of necessary patches. While the CERT does perform research on insider threats, its primary focus is on software vulnerabilities and providing incident response assistance; at least publicly, even for incident reports there is no distinction between insider and outsider attacks. As a model for obtaining information about insider threat incidents, the CERT could serve as a valuable resource by making more explicit in its activities the distinction between insider and outsider attacks. As will be discussed later, an explicit focus by the CERT on insider attack response might also be a valuable service to affected organizations.

The Bureau of Justice Statistics (BJS), part of the US Department of Justice, provides a third model for enhanced data collection. In the 1960's obtaining good data was also a problem facing research on physical crime. As concern about crime as a national, not just local, problem escalated, it became clear that no nationally accurate picture was available; in particular, there was no way to know what gap existed between the actual prevalence of different types of crime, and that being reported up through police departments to the FBI. This led to the creation of the BJS and a sister institution, the National Institute of Justice (NIJ). Today, researchers and policy makers have at least two major sources of (reasonably) reliable data on physical crime in the US – the FBI's Uniform Crime Reports, and the National Crime Victimization Survey, which checks the official picture with an independent, statistically valid survey of households. NIJ in addition funds a variety of research on issues of crime and the criminal justice system.

Each of these models has its drawbacks, and obviously would require fundamental institutional shifts. The point is, however, that it may be that such institution building will be required in order to solve, or at least begin to effectively address, the lack of data required to make real progress in solving the insider threat problem.

The fact that our definition of what we are looking for (the definition of the insider threat) is incomplete, and that we lack adequate data to examine incidents compared to the baseline population means that as researchers we can yet say only a very limited amount about the following:

- The characteristics (of the organization, insider, situation) that encourage malicious behavior;
- The linkage between what to monitor and what can be done (in reaction to a positive match) in order to prevent or mitigate the consequences of an insider attack;
- Understanding the different kinds of insiders and insider attacks;
- Balancing appropriately between restricting behavior to prevent inappropriate action, and enabling behavior to allow employees to be creative and productive.

3.1 Changing the Incentives that Organizations Face

An important part of insider threat research focuses the question of incentives relative to the insider; how might insider behavior be changed to decrease malicious or unintentional misuse. This is an important issue, but an equally important question is how the behavior of organizations might be changed. This relates to a fundamental delineation in the set of responses to insider threats: addressing the insider threat (as an individual, group) versus focusing on the organizational environment that allows or even inadvertently encourages insider threats.

The 2000 RAND workshop [18] noted that there are near-term solutions (6-12 months) to the insider threat, primarily through use of existing commercial and government off-the-shelf software and systems, including:

- Install vendor-supplied security patches;
- Review and monitor existing event logs;
- Use existing access control;
- Employ configuration management – the ability to map your network/hardware/software
- Filter malicious code at system choke points;
- Filter for future and unknown malicious code, and exercise mitigation and containment;
- Track data pedigree and integrity.

The report notes that while a number of the above recommendations can *technically* be implemented quickly, ‘the roadblock to implementation may be more

[18] Anderson (2000).

procedural and administrative than technical.' While this is undoubtably true, the challenges of scale and complexity in applying technology effectively are also daunting. Rather than being 'procedural and administrative,' the real roadblock is one of organizational incentives. This highlights the importance of considering what influences might be brought to bear on the organization for more effective response.

More effective organizational efforts to reduce the insider threat can take several forms. The RAND report focuses mostly on a variety of 'technical' avenues already extant for reducing insider threats, but also includes some aspects of organizational security policy. The challenge of getting organizations to adopt and use the best (or close to the best) available security practices and technologies is common across all cyber security problems, whatever the origin. The growing interest in the economics of cyber security reflects in part the question of the role that economic forces play in shaping the investment, and attention that organizations pay to improving security.

With few exceptions – such as the standards created by the Federal Information Security Management Act (FISMA) for US Federal agencies [19] – security standards are voluntary, and usually lacking in specificity. It is interesting to note that, to my knowledge, there has been no work on developing subsets of security policies targeted specifically at curbing the insider threat. There also seems to be little work on developing tailored packages of system configuration and design choices focused on reducing insider threats specifically.

There are other ways in which organizations could reduce insider threats. Changing the organization's culture and values – the generally shared expectation of what is and is not acceptable behavior – is an obvious approach. The question of how to change organizational culture is raised – and to some extent more or less answered – in the general management literature, as well as in more rigorous social science research, but never to my knowledge in the context of attacking the insider threat. There existis a number of case studies of organizations changing their culture, and shifting priorities – towards greater quality, or improved safety, or enhanced customer satisfaction. This suggests that reference to these histories might usefully inform a better understanding of changing organizational culture with reference to insider threat reduction.

Another observation on the types of organizational responses that might be developed to counter the insider threat lies in making the distinction between 'private action' taken when an insider threat is detected, and 'public action' including recourse to criminal prosecution. As just discussed, a reasonable hypothesis, supported by some anecdotal evidence, is that most cases of malicious insider threats are handled privately, that is, without recourse to either the courts, or the court of

[19] The Federal Information Security Act, part of the Electronic Government Act of 2002 is intended to develop a comprehensive framework to protect the government's information, operations, and assets related to information technology. See for example Office of Management and Budget, Federal Information Security Management Act (FISMA) 2004 Report to Congress. Office of Management and Budget, March 1, 2005.

public opinion (or censure). Insider threat individuals are fired, blacklisted, or reprimanded.

The question following is: does such a system of largely private action constitute a ‘better’ or more socially optimum approach than if, alternatively, all or most insider threats were publicly reported and prosecuted (at least those involving malicious intent)? I am unaware of any work on this question, or indeed that the question has been posed, at least in the context of the insider threat. Yet recent events point to this as a potentially important question. The California Data Breach Notification Law, and similar laws now in more than 30 other states, requires notification of individuals when some types of personal information have been compromised or exposed. These laws make no distinction between insider threats (accidental or otherwise) and outside attacks. The point is that, for a very select set of incidents, the boundary between private and public response by the organization has been shifted. This shift motivates two considerations for study. One it to consider whether requiring public disclosure of certain types of data breaches is, in some sense of social optimization, ‘better’ than not requiring such disclosure. Better understanding of this issue also has direct relevance to the range of possible solutions that might be brought to bear on the insider threat problem.

If the range of responses or actions that an organization has available, more or less in a form for immediate implementation, includes technical, security policy, and organizational cultural changes, then the question follows as to why these changes are not already in place. Incentives for taking action can operate along several dimensions. For each dimension a deeper understanding is required of how each can appropriately shape incentives:

- Awareness: Shifts in awareness might better be characterized as the realization by an organization’s leadership or its members generally, that ‘we’re just not going to take it anymore.’ Shifts in the social acceptability of drunk driving and cigarette smoking demonstrate that changes in how a problem is viewed by society do occur. The problem of creating greater awareness as a lever for changing organizational posture towards insider threats is, however, challenging in at least two ways. First, while its clear that organizations, and society, can shift their value system in fundamental ways, it is less clear how such changes can be engineered. For cyber security problems in general, greater awareness of the threat has been part of the solution arsenal for years, and it is unclear whether efforts directed towards that goal have really changed behavior much. People still pick easy-to-remember (and guess) passwords; access information is still written on sticky notes near computers (a conclusion based on my own observations). Second, since we have not defined with any rigor exactly what an insider threat is, defining what awareness, and consequently what behavior, is to be changed remains equally problematic.

- Economics: Some basic questions seem unanswered regarding the true cost of insider threats, how these costs are distributed, and whether these costs, once recognized, would be sufficient to motivate organizational response. It might be, for example, that most insider threats are undetected; alternatively, the business economic calculus of most organizations may conclude that the benefits of more effective response are outweighed by the costs.
- 'Law and quasi-legal requirements:A number of criminal laws relate to intentional unauthorized computer or device use, or to the consequential results of such actions (e.g., embezzlement, theft, extortion, sabotage). These laws are targeted at the individual; to my knowledge, none is targeted specifically at the insider threat. A more circumspect body of law is directed at the organization itself. Whether or not an employee is acting within the context of his employment is one part of this body of law; an 'insider threat' acting within the context of his employment creates legal exposure to the employer. Data breach disclosure has already been noted. For some sectors (national security, health care, banking and finance) sector-specific legal frameworks that are relevant to the insider threat exist.
- Law is a double-edged sword. On one hand the law might constrain certain actions that, for example, would violate personal privacy. On the other hand, law can create a motivation for more effective organizational response. The conceivable range of possible changes in the legal structure relevant to insider threats is broad. Political and other considerations may limit the realistic choices available, but it strikes me as an important question to consider whether the current legal approach, which focuses mostly on criminalizing the actions of the insider threat, strikes an appropriate balance with changing the incentives that organizations face in prevention and response.

The point is this: that there are a number of possible actions more or less immediately available to the organization that might reduce the insider threat. How much this suite of available solutions actually would reduce the problem is unknown, but it seems intuitively clear that some improvement would result. How to create the appropriate incentives, and what form those incentives would take, for organizations to adopt this suite of available solutions seems to be an important question that deserves more attention.

There is also an argument to be made about the current solutions being too expensive/complicated/non-scalable for use against a threat that is deemed harder to crack than the "outsider" threat (which is already very difficult to defend against, despite all the great technology we have, and the benefit of a more clear distinction between 'us' and 'them' than in the insider threat case).

But the point remains: if we aren't not using the tools available now to counter the insider threat, on what basis should there be confidence that new tools and approaches developed will in fact be used? To summarize: solutions that are not used do not work.

3.2 Integrating Technical Solutions with Social Science Perspectives

An unsurprising observation is that research on insider threats is dominated by computer scientists. That this is so is natural, given the primacy of the information system in considering the problem. The unfortunate consequence is that other perspectives which might usefully inform solutions have had less of a role in their development than is appropriate.

Consider one specific example: monitoring of insider activity. The Electronic Communications and Privacy Act makes clear that employees (insiders) generally do not have any rights to privacy in their electronic communications in the workplace. We can further identify a number of situations where regulatory or legal requirements obligate an enterprise to monitor employee behavior. The Federal government requires e-mail record retention. Banks are required to enforce internal firewalls and must monitor the behavior of employees to prevent specific behaviors such as insider trading. Hospitals and other health care enterprises face the challenge of authenticating the legitimacy of access to medical records. In general, it seems intuitively appealing that monitoring or audit processes to identify suspicious or abnormal behavior by insider individuals provides a promising approach in developing solutions to insider threats.

However, the problem of identifying suspicious behavior through monitoring can be extremely nuanced. Consider the following question: What is ‘abnormal’ behavior? Defining abnormal inherently assumes that a baseline of normal behavior is known. One approach has been to create individual baselines, along whatever criteria are chosen, as for example file access, or time of day usage, or other behavioral features, and to flag variances from this baseline. Baseline creep is an obvious challenge to this approach; a highly motivated malicious insider could, arguably, change behavior gradually over time so that inappropriate actions are no longer flagged by an automated monitoring approach.

More fundamentally, defining abnormal behavior also has inherent in it some assumption of a stable organizational environment, or an assumption that abnormal system behavior is inherently at deviance from the employee’s necessary job functions. There are numerous examples to suggest that neither of these assumptions is appropriate for many organizations. Investment banking takes place in a highly dynamic environment, where in the course of executing a particular transaction virtual teams may form and reform on short notice; resources that may never have been accessed before are called upon if they are appropriate to that particular deal, and individuals may be performing tasks outside of previously defined roles. This is inherent to the nature of the investment banking business, and typifies a number of professional environments where, because of financial or other implications, it is appropriate to worry about the insider threat.

Equally, an insider may be in a position that demands creativity or flexibility in order to adequately perform his job. An intelligence analyst may, based on a hunch or an insight, want to access files or otherwise use an IT system in ways that would, based on past behavior or defined job description, be considered out of variance.

Both these observations motivate a further question:

- Assuming that abnormal behavior is flagged, what is done in response? A simple, but in my opinion naïve answer would be something along the following: after abnormal behavior is flagged by some monitoring system, the organization would investigate the incident to determine whether the incident breached whatever definition of serious or inappropriate action has been set. I would suggest, however, that some thought needs to be given to the impacts of this approach to both the organization and the individual whose actions have been flagged. Creative employees may not appreciate having to continually explain and justify their actions to possibly unsympathetic IT security officers. Motivated insiders just trying to get the job done under conditions of stress and time pressures might be equally unappreciative of the benefits of an insider threat monitoring system. From the organization's perspective, there are possibly large administrative costs, potential legal exposure, and most likely the prospect of employee dissatisfaction from this course of action.
- Where do social expectations figure in? However clear the law may be about the lack of privacy in an employment context, there are instances where monitoring breaks an implicit, and perhaps poorly understood, social contract. From personal experience I can suggest that many medical doctors and senior business executives are unlikely to be accepting of knowing that their system actions are being monitored, whatever the overall benefits to the organization.
- How does the very act of monitoring change that which is being observed? With a nod towards quantum mechanics, monitoring might change some aspect of the IT systems' performance or functionality. Monitoring might also change organizational behavior, a consequence well known in social science research. These impacts are of course not mutually exclusive, and they may be unanticipated or even unobserved (unless one knows that to look for).

The intent in this discussion is not to opine specifically about the appropriateness of monitoring-based solutions. Monitoring and auditing to detect abnormal, and therefore perhaps threatening, insider activity is legal, and intuitively promising as an avenue towards solving the insider threat problem. The point here is not to suggest otherwise, but rather to observe that in pursuing this approach a number of issues arise that are in the realm of the social sciences rather than computer science. Not to consider these other perspectives runs the risk of creating solutions that are either unacceptable to the institution or its members (or both), or that in the process of solving the insider threat problem we generate other problems or costs that may be worse than the original problem (creative employees moving on to other companies or, more subtly, creative energies being restrained).

From the specific example above about monitoring flows a more general observation – that social science perspectives can and should play a greater role in addressing the insider threat. In considering the motivations behind insider threats this appeal to the social sciences has already been recognized, but the observation is more general. Technical solutions may create personal or organizational consequences that require a richer understanding that only the social sciences can provide. I would suggest that it would be useful to ask 'to what extent can technology bound the insider threat problem?' rather than 'how can technology solve the problem?'. By exploring the limits of what a technical approach can provide, a set of questions appropriate to the social sciences may then emerge, providing a more structured way of integrating the social sciences into the full consideration of solutions to the insider threat.

3.3 Creating a Response and Recovery System for Insider Threats

The consequences of an insider threat can be devastating to both organizations and individuals. The financial impact can be significant, as can be the disruption to the normal operations of the business. Less well considered is the impact on individuals. Since an insider threat by definition involves the misuse of trust, bonds of trust and understanding among the organizations members, and perhaps others, may be shattered. To my knowledge, little work has been done on the short- and long-term consequences of an insider threat incident to the organization or its members.

After an insider threat incident is detected, the organization faces a number of legal, institutional, and IT systems related challenges. How should the individual(s) responsible be dealt with? Should law enforcement and legal options be pursued? What changes in organizational security policy and practices should be made? What other changes in organizational practice (e.g., hiring, monitoring, review) are needed? What changes in system configuration and access control should be considered?

This set of questions begs an understanding of what constitutes effective response. Ideally, one would like to have a framework outlining the dimensions of effective response. As this discussion suggests, such a framework should include changes in security policy and system configuration, but also a consideration of how an organization heals itself after a trust relationship is breached.

Large organizations like multinational banks may or may not have the internal resources and infrastructure to address these issues themselves. Smaller organizations almost certainly do not.

Earlier, I suggested that more fundamental structural changes in the way researchers seek to obtain data on insider threats should be considered. In the spirit of (colloquially) killing two birds with one stone, it follows that creating a more formal response and recovery system for organizations that have faced insider threat incidents has both practical and research-motivated appeal. A central pool of resources, particularly of individuals who understand the insider threat problem, available to assist organizations having experienced an insider threat incident, could be of immense value. Certainly this would be the case for smaller organizations where arguably no one in the company or its set of advisors – technical, managerial, legal – has had prior direct experience in dealing with a serious insider threat incident. From a research perspective, a centralized response and recovery system could, properly structured, also provide a window into a richer set of data on insider threats.

The model referenced here is the CERT system. While providing and demonstrating through its history an ability to maintain absolute confidentiality of those organizations that report cyber attacks or vulnerabilities, the CERT in return provides those reporting with valuable services in both immediately responding to the incident (hence its name, Computer Emergency Response Team) and also by appropriately alerting the larger user community of the threat and working to generate solutions. The difference, however, is that an insider threat may involve dimensions beyond software and hardware vulnerabilities and exploit techniques. Ethical, legal, and social/organizational considerations may form an important part of an effective response. Experts with these differing perspectives should be an important part of an effective response team appropriate for response and recovery to insider threats. While the specific form of a central response service to insider threats needs to be defined, I would conclude that there is a need for such a system, that such a system would prove valuable to those organizations subject to insider threats, and in addition help address the data problem.

4 Conclusion

I observe that despite much good work, little progress has been made in solving the insider threat problem. Our lack of progress collectively may stem from the need to more rigorously consider a set of basic questions and issues about the insider threat problem.

I have posed five questions and issues in this chapter, about the definition of the insider threat, the lack of data to study the problem, the need to better understand incentives facing the organization as well as the individual insider, the challenge of integrating social sciences into the problem definitions and solutions, and the opportunity to create a response and recovery system for this unique set of problems. This list is by no means exhaustive; my hope is that those reading this article will be motivated to pursue these and other lines of thought with the goal of providing greater rigor and definition in our work. With greater rigor I hope that we may, as a community, make more progress in, if not solving, then effectively addressing aspects of the insider threat problem.

Research Challenges for Fighting Insider Threat in the Financial Services Industry

Dan Schutzer

Executive Director Financial Services Technology Consortium

1 Introduction

Although the problem of insider threat has always been of great concern in the Financial Services sector, its significance has increased in recent years. This increased concern can be attributed to three key factors:

Consolidation and Globalization: Consolidation and Globalization trends have led to larger institutions comprised of a significantly larger and more diverse workforce to manage, and greater employee turnover: the largest institutions can have hundreds of thousands of employees, not counting site contractors and outsourcers.

Rise of the Extended Enterprise: The dramatic advances in computer and communications technology, and the wide use of electronic commerce has made it possible and desirable to extend the traditional corporate boundaries, making its resources and information more accessible to outsourcers, third-party processors, vendor and customer partners.

Increased value of information: With the growth of electronic commerce, organized crime has found a more efficient and less risky way to commit acts of crime, including identity theft related fraud. It is now easier to recruit accomplices to commit crime more efficiently, and at a safe distance, often hiding from the law in foreign countries where it is more difficult to be caught and prosecuted. This has increased the value of information and the price people are willing to pay for it, including making it easier to collect and sell it without fear of being caught and brought to justice. This has increased the temptations for employees to turn bad and commit theft of sensitive personal information.

The first two factors have resulted in fewer employees remaining loyal to their company, and more likely to become disgruntled. It also makes it harder for the company to track and monitor suspicious activities. The last factor makes the motivation to commit insider crime stronger by making the theft of information both more rewarding and less risky. The third factor contributes to the growth of insider theft by increasing the motivation and making it harder to detect and protect against.

The challenge for research is to compensate for these negative trends by applying technology smartly and innovatively to assist in reducing the motivation of employees to commit or aid and abet insider theft. This can be accomplished by applying technology that can both reduce the motivation for employees to turn bad, improving employee morale, and increasing the risk and penalty of getting caught, while raising the bar of succeeding, by introducing more effective identity theft prevention measures.

In order to identify the most pressing research challenges needed to combat Insider Theft, it is helpful to discuss the overall problem and review the current approaches, shortcomings and gaps in the fight against Insider Theft. The following list is not meant to be exhaustive, but rather to stimulate the researcher into thinking about some of the key needs, problems and challenges in developing solutions to help fight Insider Theft.

An organization's defenses against Insider Theft can be organized along the following three categories:

1. Screening and selecting employees and other individuals who are granted access to sensitive company information and resources.
2. Controlling access to data and resources, and protecting data and resources against unauthorized access.
3. Monitoring and detecting attempts, patterns and incidents of Insider Theft, and acting in an effective and timely manner to prevent or mitigate loss.

The effectiveness with which these three types of defenses are used today, and corresponding research needs is discussed below.

2 Employee Screening And Selection

Studies have been shown that approximately 1/3 of all convicted insiders had prior arrests. This illustrates two current shortcomings:

1. the failure of background checks to detect known criminals, and
2. the ability to better screen the 2/3 of the convicted insiders who would have been hired even if more effective background checks had caught all the previously known criminals.

The first problem, inadequate background checks, could be improved through better information sharing amongst Financial Institutions. Although this is starting to take place, within the constraints imposed by legislature designed to prevent discrimination or violation of an individual's privacy rights. However, as this information-sharing improves, we are likely to see more effective evasive measures being used, such as falsifying social security numbers, names, and addresses. This reinforces the need for developing more sophisticated and robust means of identifying an individual, even one that is taking evasive actions not to be identified.

These may include the use of biometrics and advanced knowledge-based approaches (e.g., being able to sort through a large body of answers searching for clues or `slip ups' that could be used to uncover an applicant who is lying, and provide hints to their true identity). Here of course privacy concerns will have to be addressed.

The second problem, the ability to identity and screen out individuals with no prior known history of crime, who have a high likelihood of performing Insider Theft, is a much harder problem. It suggests a number of research topics; namely:

1. better psychological and biometric indicators that are able to distinguish between individuals more likely to succumb to temptation, from individuals with sufficient built-in moral ethics that they are unlikely to commit fraud, for money, or out of anger;.
2. the ability to find linkages to known criminals based on background checks and the information provided; and
3. the ability to perform this screening without violation of privacy rights or risk of litigation. The screening filter would have to be proven to not discriminate and to provide very accurate results, with very few false alarms (i.e., rejecting a good person).

3 Access Controls

Today's access controls are relatively rudimentary. It would be desirable to have more granular authentication and access controls that can more accurately uniquely identify people and devices and the information and resources they are eligible to access, where the access controls can be applied on arbitrarily small units of information, including unstructured data. An example of some of the desired capabilities includes being able to identify and allow only authorized portable USB storage devices to download sensitive files and then store them encrypted. Furthermore, these USB storage devices should not decrypt and allow data to be accessed by any machine other than an authenticated and authorized machine. And Furthermore, sensitive data should only be able to be accessible by individuals who can be uniquely be identified with high confidence (e.g., using two-factor authentication or better). Extremely sensitive data might be required to only be accessed when approved by two or more authenticated and authorized individuals – providing sufficient checks and balances to prevent collusion and/or credential stealing or guessing. This might also include more adaptive dynamic processes that can change the access controls and required checks and balances in real time, in response to changes in the environment, to monitoring and risk-scoring alerts, to changes in the nature of the risk of compromise of the data and resources. This would also involve being able to effectively classify information and resources in the first place. Adding all of these access control features on top of existing legacy financial service implementations is a challenge. Furthermore, all of this would have to be done in reasonable response times, imposing minimal processing and manual overhead.

4 Monitoring And Detection

It would be desirable to improve the ability to forecast when an employee is planning to commit insider crime before it actually occurs. Today, few have the ability to do this sort of forecasting reliably. Creating this capability might include developing the ability to detect changes in an individual over time that would indicate an increase in the likelihood that they were going to commit a crime. This could include measurable changes in their anxiety or anger, in their economic situation, in their workplace environment, or in their behavior (spotting anomalous or suspicious behavior, or behavior that fits a known criminal pattern), and combinations of all of these predictors. It also includes being able to take actions (e.g., adding restrictions, planting traces and traps, or just monitoring more closely with timely reporting), that would help prevent Identity crime without violating individual rights. This would also include more timely and meaningful reports. If forecasting fails, then at least developing tools to have in the early detection and rapid response, including loss mitigation and criminal capture and prosecution would be desired. Since many data breaches occur because of negligence or unintentional loss of sensitive data, such monitoring and detection research might also produce tools for these situations as well.

Hard Problems and Research Challenges Concluding Remarks

Angelos D. Keromytis

Computer Science Department, Columbia University

The purpose of the workshop whose proceedings you are holding was to define the nature and scope of the insider attack problem, describe the known state of the art in defenses, and to outline an agenda for future research.

One of the findings of the workshop, as discussed in Hunker's paper, *"Taking Stock and Looking Forward",* is that even the seemingly simple task of defining the problem boundaries is hard, not least because of the lack of appropriate definitions and contextual information, data for analysis, experimentation and, ultimately, validation of proposed solutions. This lack of data is driven by a variety of factors, the most prominent of which appears to be the sensitivity of the topic: organizations that have been the victims of insider attacks tend to handle such (known) incidents as quietly as possible. As described by both Pfleeger, in *"Reflections on the Insider Threat*, and Hunker, the lack of a good definitions relating to the problem confounds data collection and organization. Lacking good sources of both anecdotal and quantifiable information on insider attack incidents means that, in practice, we can only begin to outline the dimensions of the problem. The workshop made good progress in this space. Bellovin's introductory paper, *"The Insider Attack Problem Nature and Scope"*, helped us frame the problem, and to begin developing a shared vocabulary. Furthermore, as we saw in Moore and Cappelli's paper, "*The "Big Picture" of Insider IT Sabotage Across U.S. Cricital Infrastructures"*, advances in understanding the problem have been made in those cases where access to useful data was made possible. Thus, it is clear that a necessary component for successfully addressing the insider problem is the development of ways for sharing sensitive data pertaining to such attacks in a way that assuages the concerns of the organizations involved while providing useful material for study and experimentation to researchers. This is a large problem space that requires advances in research artifacts and public/organization policies and practices. For example, Hunker suggests that CERT (or a similar organization) provide a trusted first-response service to insider abuse incidents, and simultaneously act as a natural data aggregation point.

The second finding from the workshop is that further research and technical work is needed in a variety of areas that can be brought to bear on the problem. Although Ben Salem, et al.'s paper, *"A Survey of Insider Attack Detection Research"*, describes substantial promising work that has been done in the space of detection and monitoring (the research area that has been traditionally most involved in addressing the insider problem), many challenges remain. As shown by

Killourhy and Maxion in *"Naïve Bayes as a Masquerade Detector: Analysis of a Chronic Failure"*, the nature of the insider problem and the limitations of detecting misbehavior in complex environments mean that additional work is required if such mechanisms are to be employed with any confidence. Such limitations also point to the problem we identified earlier, the lack of good data for analysis, experimentation and validation. Furthermore, McCormick's paper, *"Data Theft: A prototypical Insider Threat"*, tells us that, despite the plethora of detection and mitigation techniques, current practices are inadequate for dealing effectively with one of the most prototypical insider attacks, theft of sensitive data from an organization. It is particularly revealing that this opinion is shared by many experienced security practitioners in the financial services industry, a sector that has both had considerable experience with malicious insiders (going back several centuries) and is a very aggressive adopter of new security technologies and practices. As Schutzer outlines in *"Research Challenges for fighting Insider Threat in the Financial Industry"*, the current vision of more effective defenses against insider attacks, focusing on insider theft, revolves around better screening and behavioral prediction (implying considerably more invasive information collection), combined with ever-so-fine-grained access controls. As Hunker reminds us, however, there are costs to such measures. These costs include financial expenditures, IT resources and personnel, and (if they are sufficiently invasive) a stifling of creativity and an erosion of trust between employer and employees.

It is worth remembering, however, a number of long-established principles that can guide us in the development of new systems, techniques and practices for addressing the insider problem more effectively. As Gligor and Chandersekaran reminds us in *"Surviving Insider Attacks"*, *separation of duty*, *critical function partitioning*, *system design and implementation diversity*, *fast failure detection and repair, attack deterrence*, and the application of *least privilege* are concepts that are particularly relevant to designing and implementing insider-resistant systems. The challenge, of course, is how to go from the general principles to specific solutions. Gligor and Chandersekaran suggest conducting experiments that are centered on building systems that solve specific sub-problems, an approach that can and should be pursued by interested stake-holders with access to data (e.g., financial sector) and research funding (e.g., government funding agencies).

A particular challenge in addressing the insider problem in modern enterprises is system scale and complexity: it is rarely the case that security practitioners, system administrators or, for that matter, any single person understands what sensitive data are handled by what part of the organization. Sahita and Savagaonkar's paper, *"Towards a Viurtualization-enabled Framework for Information Traceabilty"*, discusses an approach for addressing this problem and describes a mechanism for enforcing access control policies on sensitive information that leverage new virtualization capabilities in commodity processors. Iyer et al. discuss in *"Reconfigurable Tamper-resistant Hardware Support Against Insider Threats"* a suite of new approaches using reconfigurable hardware to help build computers that can defend themselves against other types of insider attack. While great first steps, further work is needed in developing algorithms and tools for allowing administrators to understand and operate on the wealth of information exposed by such techniques. In general, developing solutions that are practical and effective in large-scale systems remains an open challenge. Contrary to other areas of computer security, the insider problems requires that we take into consideration (and, perhaps, focus on) the human factor --- both on the malicious side (in trying to better understand, predict, deter, and handle malicious insiders) and on the part of the defenders (in providing them with appropriate tools). Sinclair and Smith's paper, *"Preventative Directions For Insider Threat Mitigation Via Access Control*", describes some first steps in this direction; much more work is needed in combining technology with an understanding of the human factor.

In closing, we briefly some specific issues that require further investigation, and some promising directions for further research:

- It is imperative that we acquire good data that can be used both to better understand the dimensions of the insider problem and to measure progress on detection and monitoring effectiveness. This will require bypassing strong organizational resistance in sharing sensitive data of this kind. Research focusing on anonymization/privacy preserving transformations of data traces and incident descriptions is an important component here. The establishment of appropriate repositories of such information for experimentation and validation should become a high priority task for industry and government stake-holders. The availability of data for measuring progress and for conducting comparisons between different approaches is a necessary component for establishing a sound scientific methodology in pursuing the insider threat problem.
- Part of the reason that existing detection and monitoring mechanisms are not deployed and used (or they are poorly received by end users) is that they they are by necessity intrusive, especially when suspected (but not proven) malfeasance is detected. Furthermore, such mechanisms can themselves be easily abused by their operators or by attackers that gain unauthorized access. Developing techniques that guarantee user privacy until/unless a confirmed violation occurs would greatly ease the use of existing and future protection mechanisms.
- Inferring user intent at the application layer to detect malfeasance is an area requiring further investigation. Most prior work has focused on collecting, analyzing and understanding low-level observables, such as network packet traces, system call sequences, file access patterns, *etc.* We believe that the semantic gap between low-level program behavior and high-level human operator intent many be too big to bridge. Ways of understanding application semantics and the impact of high-level operations appears to be a hard problem, but one that offers some hope in better associating human-level observable behavior and the underlying intent.

- Since there is limited/no access to good data that can help in the development and refinement of detection/monitoring mechanisms, we must also consider techniques that do not require such data. Preventive mechanisms, such as appropriate access controls and the application of known principles (as discussed previously), can help mitigate but not altogether eliminate the problem. Research in better configuring and tuning access control mechanisms has a role to play. A different, promising approach in detection involves the aggressive use of decoys, traps, and honeytokens in identifying and localizing insider incidents. These may vary in nature to fit the types of sensitive data that are in use, and can include trapped credit card numbers, unique username/password credentials (or other similar information) transmitted over specific network segments or stored in specific fileservers and/or user account files, interesting-looking emails pointing to honeypot servers, *etc.* Although such techniques have been used quite successfully in the past to address insider threats, and similar techniques (*e.g.,* network honeypots) are actively being used in different areas of computer security, there has been little systematic work to date on scalable decoy trap-based techniques. One particularly attractive aspect of such technologies is that they seem to impose minimal requirements with respect to access to data for development and validation. Naturally, such mechanisms will have to be designed such that they confound the sophisticated adversary who is aware that decoys are in use by an enterprise. At a minimum, these decoys will have to be indistinguishable from legitimate sensitive information. On the other hand, awareness of the use of decoys in an enterprise could act as a deterrent, or at least as a "speed bump" for attackers who would face the additional task (beyond penetration and data acquisition) of decoy/honeytoken identification and elimination from the captured data.
- Finally, it is important to remember that every action causes a reaction, especially in computer security. In the realm of insider threats, we must assume that any protection mechanism we employ will eventually become known to the adversary. This may be because the insider has legitimate access to this information, or because this information becomes known through a variety of means. Thereafter, the first task of a determined adversary will be to disable or "blind" such mechanisms. Thus, it is important to investigate self-protecting monitoring mechanisms, which can confound the attacker, or at least sufficiently complicate the attack process such that the chances of detection increase. The nature of such self-protection can be manifold: trusted hardware monitors, overlapping monitoring schemes, mutually-protecting monitors, *etc.* We believe that only a focused research effort in developing hardened protection mechanisms will result in adequately strong defenses.

Index