Designing Host and Network Sensors to Mitigate the Insider Threat

Brian M. Bowen Malek Ben Salem Shlomo Hershkop
Angelos D. Keromytis Salvatore J. Stolfo
Department of Computer Science, Columbia University

Abstract

We propose a design for insider threat detection that combines an array of complementary techniques that aims to detect evasive adversaries. We are motivated by real world incidents and our experience with building isolated detectors: such standalone mechanisms are often easily identified and avoided by malefactors. Our work-in-progress combines host-based user-event monitoring sensors with trap-based decoys and remote network detectors to track and correlate insider activity. We identify several challenges in scaling up, deploying, and validating our architecture in real environments.

1 Introduction

The annual Computer Crime and Security Survey for 2008 [1] surveyed 522 security employees from US corporations and government agencies, finding that insider incidents were cited by 44 percent of respondents, nearly as high as the 49 percent that encountered a conventional virus in the previous year. In general, there is an increasing recognition of the significance, scope and cost of the malicious insider problem. Some state-of-the-art defenses focus on forensics analysis and attribution after an attack has occurred using techniques such as sophisticated auditing [2] and screen capture [3]. Other commercially available systems are designed to prevent, detect, and deter insider attack. The ideal case is to devise systems that *prevent* insider attack. Policy-based mechanisms and access control systems have been the subject of study for quite some time but have not succeeded in solving the problem of preventing insider abuse. Monitoring, detection, and mitigation technologies are realistic necessities.

Detection systems have been designed to identify specific attack patterns or deviations from known, long-term user behavior. Such techniques are typically used as part of standalone mechanisms rather than in an integrated defense architecture. For that reason, the malicious behavior detection-based insider defenses suffer from several problems.

- Since behavior is a noisy approximate of user intent in the absence of sufficient contextual information about the user and the overall environment, such systems are typically tuned to minimize false alerts by being less stringent about what is considered malicious. While reducing the administrators' workload and the users' irritation factor, such tuning may allow some malicious behavior to go undetected. As with many probabilistic detection mechanisms. The tradeoffs inherent in tuning systems are poorly understood.
- Since the relationship between behavior and intent is hard to determine, and alarms may be false, it is difficult to confidently take some action (whether automated or at the human level) in response to an alert.
- By operating as standalone mechanisms, it is easy for an adversary with some knowledge of their existence to either evade or even *disable* them. In fact, we see an increasing number of malware attack that disable defenses such as antivirus software and host sensors prior to undertaking some malicious activity [4].

To address the malicious insider problem, we posit that systems must leverage multiple complementary and mutually supportive techniques to detect and deter intentionally malicious adversaries. We direct our efforts against inside attackers that have some, but perhaps not complete knowledge of the enterprise environment. In this article, we do not address the important problem of malicious system administrators who have control over all defensive systems. This remains a particularly interesting open problem.

The first component in our architecture is a decoy document generation system to deceive an insider by leveraging uncertainty of the authenticity of information that may be accessed in an unauthorized manner. Our system generates realistic-looking documents that contain both decoy credentials that are monitored for (mis)use, and stealthy embedded beacons that signal when the document is opened. Beacons are embedded in documents using methods of deception and obfuscation gleaned from studying malcode embedded in malicious documents as seen in the wild [5].

The network component integrates monitored network traps with the decoy document generation component, and allows our system to isolate the activity of malicious users. These traps allow us to follow "personalized" decoy credentials even after they leave the local environment. As with honeypots [10] the misuse of the network traps guarantees an insider has misappropriated internal information. The absconded information may even have been accessed in a way that has evaded detection (and even forensic analysis) altogether. Nonetheless, our system can determine the time and location of the leak through the use of decoys embedded in the content of the decoy document, allowing further forensic investigation and damage assessment.

The host-based sensors we designed in our architecture, collectively named "RUU", collect low-level audit data from which we identify specific user actions. RUU profiles user actions to form a baseline of normal behavior utilizing anomaly detection techniques to isolate behavior differences over time. Subsequent monitoring for abnormal behaviors that exhibit large deviations from this baseline signal a potential insider attack. We conjecture that the evidence provided by the host sensor combined with other detection techniques will indicate insider activity that violates policy. It is often noted that, on their own, anomaly detection systems have high levels of false positives. Combining multiple views of the same event can dramatically reduce the number of false positives associated with a malicious event [7].

In this article we:

- introduce and analyze a system that uses multiple sensors in a scalable fashion to leverage context for detecting malicious insider activity;
- describe new lightweight sensors that model user actions on the host level;
- present a design for decoys that combines a number of methods and monitors, both internal and external to an organization, to detect insider exploitation using ordinary-looking documents as bait.
- present a large-scale automated decoy creation and management system for deploying baited documents that allow us to detect the presence (and, in some cases, identity) of malicious insiders, or at least determine the existence of malicious insider activity. This provides a means for ordinary users to deploy decoys on their hard drives without having to deploy and configure sophisticated honeypots and sensors. Users are alerted by email when a decoy has been touched on their laptops and personal computers.
- present a preliminary analysis of the overhead and effectiveness of the system in a realistic environment.

2 Threat Model - Level of Sophistication of the Attacker

The architecture described in Section 3 relies on the use of decoys to deceive, confuse, and confound attackers, ultimately forcing them to expend far more effort to discern real information from bogus information. To understand the capability of the various decoys we introduce, it is necessary to first explore the various levels of attacker sophistication. We broadly define four monotonically increasing levels of insider sophistication and capability that may be used to break through the deception our decoys seek to induce. Some will have tools available to assist in deciding what is a decoy and what is real. Others will only have their own observations and insights.

Low: Direct observation is the only tool available. The adversary largely depends on what can be gleaned from a first glance. We strive to defeat this level of adversary with our beacon documents, even though decoys with embedded beacons may be distinguished with more advanced tools.

Medium: A more thorough investigation can be performed by the insider; decisions based on other information can be made. For example, if a decoy document contains a decoy account credential for a particular identity, an adversary may verify that the particular identity is real or not by querying an external system (such as `http://www.whitepages.com/`). Such adversaries will require stronger decoy information, possibly corroborated by other sources of evidence.

High: Access to the most sophisticated tools is available to the attacker (*e.g.,* super computers, other informed people who have organizational information). The notion of the "Perfect Decoy" described in the next section may be the only indiscernible decoy by an adversary of such caliber.

Highly Privileged: Probably the most dangerous of all is the privileged and highly sophisticated user. Such attackers will be fully aware that the system is baited and will employ sophisticated tools to try to analyze, disable, and avoid decoys entirely. As an example of how defeating this level of threat might be possible, consider the analogy with someone who knows encryption is used (and which encryption algorithm is used), but still cannot break the system because they do not have knowledge of an easy-to-change operational parameter (the key). Likewise, just because someone knows that decoys are used in the system does not mean they should be able to identify them all. Devising a hard-to-defeat scheme is the principle we explore in the next section.

We further define insider threats by differentiating between *Masqueraders* (attackers who impersonate another system user) and *Traitors* (attackers using their own legitimate system credentials) who each have varying levels of knowledge. The masquerader is presumed to have less knowledge of a system than the victim user whose credentials were stolen. The *innocent insider* who mistakenly violates policy is undoubtedly the largest population of insiders that we also target using trap-based decoys.

3 Architecture

The architecture combines host-based user-event monitoring sensors with trap-based decoys and remote network detectors as shown in Figure 1. The combination is designed to make it difficult for insiders to avoid detection with low likelihood of mis-attribution. The architectural components are described in detail in the sections that follow.

3.1 Decoy Document Distributor

One of the core components of the architecture is the Decoy Document Distributor (D^3) System, a web-based service for generating and distributing decoys. D^3 can be used by registered users to generate decoys for download, or as a decoy data source for the host and network components.

The primary goal of a decoy is to detect malfeasance. Since no system is foolproof, D^3 has been built so that multiple overlapping signals may be automatically embedded in decoy documents to increase the likelihood of detecting decoy misuse. Any alert generated by the decoy signals is an indicator that some insider activity has occurred. Since the attacker may have varying levels of sophistication (as discussed in Section 3.3), a combination of techniques is used in decoy documents to increase the likelihood that one will succeed in generating an alert. D^3 generates decoys with several means of detecting their misuse:

- embedded honeytokens, computer login accounts created that provide no access to valuable resources, and that are monitored when (mis)used;
- an embedded "beacon" that alerts a remote server at Columbia, that we call SONAR.
- an embedded marker, to enable detection by the host-level or network decoy sensor.

These features are explored in the following sections.

Our current deployment of D^3 is tailored for a university environment by both the type of documents and the bait within in them, but it can easily be adapted for other deployment environments (*e.g.,* an arbitrary commercial enterprise) . Complete details of D^3 including an evaluation of decoy documents can be found in [12]. The reader is encouraged to visit the Decoy Document Distribution (D^3) website to evaluate our technology developed to date at: `http://www.cs.columbia.edu/ids/RUU/Dcubed`.

3.2 SONAR

SONAR is the alert-management system. Its primary role is to collect alerts triggered by host and network monitors, and individual beacon signals generated by unauthorized opening of decoy documents downloaded by registered users. In response to signals it receives, it emits emails to the registered users associated with the particular decoys. Depending on the type of decoy, some signals are sent directly from a decoy itself (as is the case with beacons), while others require SONAR to poll other resources for information (*i.e.,* credential monitoring). SONAR currently polls a number of servers to monitor credential misuse including university authentication log servers and *mail.google.com* for Gmail account misuse. In the case of Gmail accounts, custom scripts access and parse the bait account pages to gather account activity information.

3.3 Decoys and Network Monitoring

The use of deception, or decoys, plays a valuable role in the protection of systems, networks, and information. The first use of decoys in the cyber domain has been credited to Stoll [8, 9] and detailed in the novel "The Cuckoos Egg." Stoll's methods included the use of bogus networks, systems, and documents to gather intelligence on the attackers, who were apparently seeking state

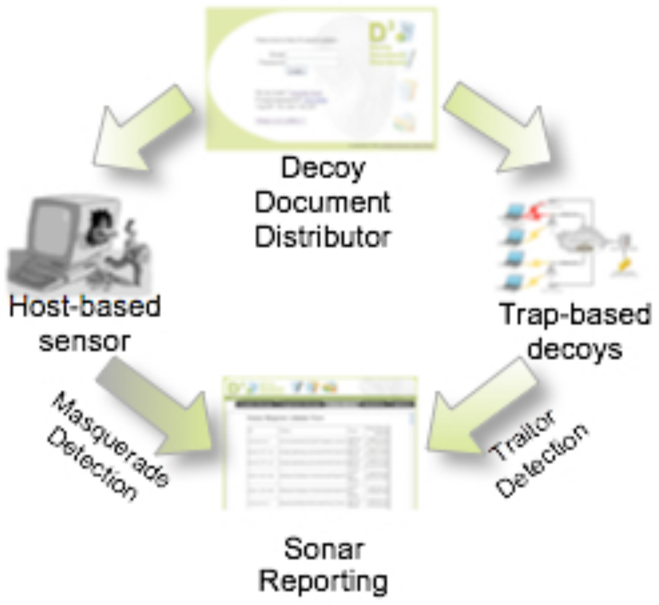

Figure 1: Architecture

secrets. Among the many techniques described, he crafted “bait” files, bogus classified documents that contained non-sensitive government information, and attached “alarms” to them so that he would know if anyone accessed them. Our decoy system builds on that notion increasing the scope, scale and automation of decoy generation and monitoring.

3.3.1 Perfectly Believable Decoys

In order to create decoys to bait insiders with various levels of knowledge and maximize the deception they induce, one must understand the core properties of a decoy. These properties guide the design of systems that automate the generation and placement of trap-based decoys. The properties include *conspicuousness, enticement, non-interference, variability, differentiable, detectability,* and *believability* [12]. In this article, we describe our efforts on maximizing the believability of decoys, one of the fundamental properties required to snare an inside attacker.

A good decoy should make it difficult for an adversary to discern whether they are looking at an authentic document from a legitimate source or if they are looking at a decoy. For concreteness, we build upon the definition of “perfect secrecy” proposed in the cryptographic community [13] and define a “perfect decoy” to be a decoy that is completely indistinguishable from one that is not. One approach we use in creating decoys relies on a document marking scheme in which all documents contain embedded markings such that decoys are tagged with HMACs (*i.e.,* a keyed cryptographic hash function) and non-decoys are tagged with indistinguishable randomness. Here, the challenge of distinguishing decoys reduces to the problem of distinguishing between pseudorandom and random numbers, a task proven to be computationally infeasible under certain assumptions about the pseudorandom generation process. Hence, we claim these to be examples of perfect decoys and the only attacker capable of distinguishing them is one with the key, perhaps the highly privileged insider.

As a prototype perfect decoy implementation, we built a component into D^3 for adding HMAC markers into PDF documents. Markers are added automatically using the iText API, and inserted into the OCProperties section of the document. The OCProperties section was chosen because it can be modified on any PDF without impact on how the document is rendered, and without introduction of visual artifacts. The HMAC value itself is created using a vector of words extracted from the content of the PDF. The HMAC key is kept secret and managed by D^3, where it is also associated with a particular registered host. Since the system depends on all documents being tagged, another component inserts random decoy markers in non-decoy documents, making them indistinguishable from decoys without knowledge of the secret key.

Decoys in Computer Security

Deception-based information resources that have no production value other than to attract and detect adversaries (like those used by Stoll described in Section 3.3) are commonly known as honeypots. Honeypots serve as effective tools to gather intelligence to understand how attackers operate. Honeypots are considered to have low false positive rates since they are designed to capture only malicious attackers, except for occasional mistakes by innocent users. Spitzner described how honeypots can be useful for detecting insider attack [10], in addition to the common external threats for which they are traditionally known. He discusses the use of honeytokens, which he defines as “a honeypot that is not a computer” [9], citing examples that include bogus medical records, credit card numbers, and credentials, with descriptions of how they can be used to detect malicious insiders [10, 9]. In current systems, the decoy/honeytoken creation is a laborious and manual process requiring administrator intervention. In contrast, we propose the seeding of decoy information (of various different types) throughout an operational system. Our work extends these basic ideas to an automated system of managing the creation and deployment of these honeytokens. Yuill *et al.* [8, 11] extend the notion of honeytokens with a “honeyfile system” to support the creation of bait files, or as they define them, “honeyfiles.” The honeyfile system is implemented as an enhancement to the Network File Server. The system allows for any file within user file space to become a honeyfile through the creation of a record associating a filename to userid. The honeyfile system monitors all file access on the server and alerts users when honeyfiles have been accessed. This work does not focus on the content or automatic creation of files, but does mention some of the challenges in creating deceptive files (with respect to names) that we address as well.

3.3.2 Trap-based Decoys

Our trap-based decoys have an inherent property of being detectable outside of a host, so they do not require host monitoring nor suffer the performance burden characteristic of decoys that require constant internal monitoring. This form of decoy is made up of “bait information” such as online banking logins provided by a collaborating financial institution [1], login accounts for online servers, and web based email accounts. In our current deployment we use Columbia University student accounts and Gmail email accounts as bait, but these can be customized to any set of monitored credentials. The trap-based decoys are managed by the D^3 web service, thereby

[1]By agreement, the institution requested that its name be withheld.

enabling programmatic access to them from all registered web-enabled clients. The automation of this service enables their distribution and deployment in large volume.

3.3.3 Beacon Decoys

Beacons are implemented to silently contact a centralized server when a document is opened, passing to the server a unique token that was embedded within the document at creation time. The token is used to uniquely identify the decoy document and its association to the IP address of the host accessing the decoy document. Additional data is collected, depending on the particular document type and rendering environment used to view the beacon decoy document. The first proof-of-concept beacons have been implemented in Word and PDF and are deployed through the D^3 website. The Word beacons rely on a stealthily embedded remote image that is rendered when the document is opened. The request for the remote image signals SONAR that the document has been opened. In the case of PDF beacons, the signaling mechanism relies on the execution of Javascript within the document-rendering application.

The D^3 web service generates many types of beacon decoys including receipts, tax documents, medical reports and other common form-based documents with decoy credentials, realistic names, addresses and logins, information that is familiar to all users. In contrast to the HMAC decoys, the believability of these documents lies in the realism of the content within them.

As noted earlier, the believability of decoys depends on how indistinguishable they are from normal documents. In the case of beacons, the network connection of the beacon may be used as a distinguishing feature. Hence, in their current form the utility of beacon decoys may be limited to ensnaring only the least sophisticated attacker. We are currently investigating environments in which it is possible to embed beacons in all documents, thereby making beacon decoys indistinguishable (modifying the document rendering application is a feasible option). Another potential problem for beacons is that it is possible for the signaling mechanisms to fail or be subverted; however, when combined with other mechanisms, we posit their use increases the likelihood of detection.

3.4 Host-based Sensors

One of the key techniques employed by the architecture involves host-level monitoring of user-initiated events. The host sensor serves two functions. The sensor is designed to profile a baseline for the normal *search* behavior of a user. Subsequent monitoring for abnormal file search behaviors that exhibit large deviations from this baseline signal a potential insider attack. The host sensor also detects when decoy documents containing embedded markers are read, copied, or exfiltrated. The goal of the host-level decoy sensor is to detect these malicious actions accurately and with negligible performance overhead. Abnormal user search events that culminate in decoy document access are a cause for concern. A challenge to the user, such as asking one of a number of personalized questions, may establish whether a masquerade attack is occurring. In Section 3.4.1, we present a preliminary evaluation of this sensor.

Our prototype sensor has been built for the Windows XP platform and relies on hooks placed in the Windows ServiceTable. This is a typical approach used by malicious rootkits; however, in contrast to the traditional rootkit objective of remaining undetected, the host-level decoy sensor does not require operational secrecy. Our threat model assumes attackers have knowledge that a system is being monitored, but they must not know the identities of the decoys or the key used by the sensor to differentiate them. Furthermore, the attacker will likely not know the victim user's behavior, information that is not readily stolen such a credential or a key. Given that adversaries may be aware of system monitoring, special care must be taken to prevent the sensor from being subverted or, equally important, to detect if it is subverted. We have ongoing work aimed at preventing and detecting subversion of the sensor. One strategy involves a means to "monitor the monitor" to detect if the host sensor is disabled use of tamper-resistant software techniques. One possible solution we are investigating relies on "out-of-the-box" monitoring [16], in which a virtual machine-based architecture is used to conduct host-based monitoring outside of the host from within a virtual machine monitor.

Data and Evaluation

Research in insider attack is made difficult due to the lack of readily available insider attackers or a complete set of realistic data they generate. For this reason, researchers must resort to generating their own data that simulates insider attacks. The Schonlau dataset [14] is the most widely used for academic study. It consists of sequences of 15,000 UNIX commands generated by 50 users with different job roles, but the data does not include command arguments or timestamps. The data has been used for comparative evaluations of different supervised machine learning algorithms. The Schonlau data is not a "true Masquerade" data set: the data gathered from different users were randomly mixed to simulate a masquerader attack, making the dataset perhaps more suitable for "author identification" studies. An alternative approach to acquire sufficient data for evaluating monitoring and detection techniques is to devise a process to acquire human user data under normal operation as well as simulated attack data where "red team" users are tasked to behave as inside attackers. This type of study is typically subject to Institutional Review Board approvals since human subjects are involved. The process is costly, in time and effort but is sensible and appropriate to protect personally identifiable data of individual volunteer subjects. This was the approach taken by Maloof *et al.* for evaluating ELICIT [6]. We as well gathered data from 34 users, all CS students at Columbia University, by distributing host sensors that upload system event data during normal system use. The population of student volunteers assures us the data they generate is derived from sources that have a common "role" in the organization, and hence variations in the user behavior and their data are not attributable to different job functions as is undoubtedly the case with the Schonlau dataset. We have also gathered data from 14 paid volunteers who emulated masquerade attacks on equipment provided in our lab. The dataset, which we call the RUU (Are You You?) data set, is over 8 GBytes and is available to legitimate researchers for download: `http://www1.cs.columbia.edu/ids/RUU/data/`. The data collected for each user averages about 5 days of normal system use, ranging in the extreme between 1 day and 59 days, and an average of more than 1 million records per user. Preliminary results using this data and the abnormal search benavior sensor described in the article show that the red team of masqueraders deviate substantially from ordinary system users. [15]

3.4.1 Detecting Anomalous User Search Actions

The sensor collects low-level data from file accesses, windows registry accesses, dynamic library loading, and window access events. This allows the sensor to accurately capture data about specific system and user behavior over time. For example, we posit that one method to check if an insider has infiltrated the system is to model "search" behavior as a baseline for normal behavior. We conjecture that each user searches their own file system in a unique manner. They may use only a few specific system functions to find what they are looking for. Furthermore, it is unlikely a masquerader will have full knowledge of the victim user's file system and hence may search wider and deeper and in a less targeted manner than would the victim user. Hence, we believe search behavior is a viable indicator for detecting malicious intentions. Specific sections of the windows registry, specific DLLs, and specific programs on the system are involved with system searching applications. For a given time period (10 seconds in our initial experiments), we model all search actions of a user. After a baseline model is computed, the sensor switches to detection mode and alerts if the current search behavior deviates from the user's baseline model. Deviation measurements are made by examining a combination of the volume and velocity of system events in association with other user activities that should add some context to the user search actions, such as the number of processes being created and destroyed. Presently, this sensor component is being integrated in the architecture to function with the host sensor described next that detects decoy document accesses.

To evaluate this model, we first gathered user-event data to compute the baseline normal models, as well as data that simulates masquerade attacks. The dataset, known as the RUU dataset is described in "Data and Evaluation" sidebar. For the former, we had 34 computer science students install a host sensor on their personal computers. The sensor monitored all registry-based activity, process creation and destruction, window GUI access, and DLL libraries activity. The data gathered consisted of the process name and ID, the process path, the parent of the process, the type of process action (e.g., type of registry access, process creation, process destruction, etc.), the process command arguments, action flags (success/failure), and registry activity results. A *timestamp* was also recorded for each action. The collected data was automatically uploaded to a server, after the students had the chance to filter any data that they were not willing to share.

To obtain masquerade attack data, we conducted a user study in which 14 students had unlimited access to the same file system for 15 minutes each. None of the users had prior access to this file system, which was designed to look very realistic and to include potentially interesting patent applications, personally identifiable information, as well as account credentials stored in various files. The students were provided a scenario where they were asked to perform a specific task, which consisted of finding any data on the file system that could be used for financial gain.

The features used for modeling were in essence volumetric statistics characterizing search volume and velocity, and describing the overall computer session in terms of number of processes running, particulalrly the number of editing applications. A one-class Support Vector Machine (ocSVM) model was then trained for each user using those features. The same features were extracted from test data after dividing them into 10-second epochs. The ocSVM models were tested against these features, and a threshold was used to determine whether the user activity during the 10-second epochs was normal or abnormal. If the user activity was performed by the normal user, but was classified as abnormal by the ocSVM model, a false positive is recorded. Our results using the collected data and the modeling approach described above show that, we can detect all masquerader activity with 100% accuracy, with a false positive rate of 0.1%.

Extensive prior work on masquerade attack detection has focused on the Schonlau dataset for evaluation [14]. The data set served as a common gold standard for researchers to conduct comparative evaluations of competing machine learning algorithms. The basic paradigm this work follows is a supervised training methodology where 5000 commands from each user serve as training data for the users normal behavior model. A classifier or model for each user is then tested against hold out data not used in training from the users command dataset but embedded in a random location with another randomly chosen users data. The performance results reported indicate the accuracy of the classifiers learned by a particular machine learning algorithm in identifying foreign commands, those blocks of commands deemed abnormal.

The model we chose to embed in the user search command sensor is different from these prior bag of command oriented models. Our current studies analyze user command events and the rates at which commands are issued using the RUU datasets described in the sidebar. Accuracy is estimated with respect to classification errors measured for each 10 second epoch of user events. Furthermore, whereas the Schonalu data consists of Unix commands, the RUU datasets contain user events created in a Windows environment.

In order to compare our results with these prior studies, we need to translate the false positive rates in classifying blocks of 100 commands with the error rate of classifying user commands issued within each standard duration epoch. Unfortunately, the Schonalu datasets are devoid of timestamps and a direct comparison of our modeling technique is not feasible. No one can accurately determine how long it takes each user in the Schonlau data to issue 100 commands. If we assume that it takes 20 seconds to issue one user command on average (a rough estimate from the RUU datasets for certain periods of time), our experiments show a detection rate of 100% can be achieved with a false positive rate of 1.4%. This is a 78% improvement in false positive rate over the best reported classifier in the prior Schonlau work. Indeed, none of the prior work reports a 100% detection rate at any reasonable false positive rate. If we assume it takes on average longer than 20 seconds to issue a user command, the results we achieved drops the false positive rate even further.

The comparison may not be entirely fair since the models and the data are quite different even though the data are generated by human users. The use of temporal statistical features from the RUU data set is crucial in modeling users behavior leading to far more accurate results than blocks of commands. Furthermore, in our work, we focus on user search events, limiting the amount of data analyzed and reducing the complexity of the learning task. Complete details of the volumetric and contextual features we used when modeling user commands and the results achieved are reported in [15] where results of reducing the data in the Schonlau experiments are also described. The RUU datasets were created and are available to serve as a more up to date and modern gold standard for other researchers to perform similar studies.

3.4.2 Detecting Perfectly Believable Decoys

The second host sensor also detects malicious activity by monitoring user actions directed at HMAC-embedded decoy documents. Any action directed toward a decoy is suggestive of malicious activity. When a decoy document is accessed by any application or process, the host sensor initiates a verification function. The verification function is responsible for differentiating between decoys and normal documents by computing a decoy HMAC (as described in Section 3.3.1) for the particular document in question and comparing it to the one embedded in the OCProperties section of the document. If there is a match, the document is deemed a decoy and an alert is triggered; otherwise, the document is deemed normal and no action is taken.

The host sensor performs tasks similar to antivirus programs. In evaluating the performance of the sensor, we use overhead comparisons of antivirus programs as a benchmark, since the task of comparing an HMAC code is not substantially different from testing for an embedded virus signature. Hence, accuracy performance is not revelant for this particular detector. However, there is a fundamental difference between the task of detecting malware and that of detecting decoy activity. Antivirus programs are designed to prevent the execution of and quarantine malicious software whenever any process is initiated. In decoy detection the objective is merely to trigger an alert when a decoy file is loaded into memory. Thus, the decoy detection need not serialize execution; for example, it may be executed asynchronously (and in parallel by running on multiple cores).

We have tested the decoy host sensor on a Windows XP machine. A total of 108 decoy PDF documents generated through D^3 were embedded in the local file system. Markers containing randomness in place of HMACs were embedded in another 2,000 normal PDF files on the local system. Any attempt to load a decoy file in memory was recorded by the sensor including content or metadata modification, as well as any attempt to print, zip, or unzip the file.

The sensor detects the loading of decoy files in memory with 100% accuracy by validating the HMAC value in the PDF files. However, as we discovered during our validation tests, decoy tests can be susceptible to non-negligible false positive rates. The problem encountered in our testing was created by antivirus scans of the filesystem! The file accesses of the scanning process that touched a large number of files, resulted in the generation of spurious decoy alerts. Although we are engineering a solution to this particular problem by ignoring automatic antivirus scans, our test does highlight the challenges faced by such monitoring systems. There are many applications on a system that access files indiscriminately for legitimate reasons. Care must be taken to ensure that only (illicit) human activity triggers alerts. As a future improvement to the sensor, file touches not triggered by user-initiated actions, but rather caused by routine processes, such as antivirus scanners or backup processes may be filtered. Nevertheless, this demonstrates a fundamental design challenge to architect a security system with potentially interfering competing monitors.

With regard to the resource consumption of the sensor, the components of the sensor used an average 20 KB of memory during our testing, a negligible amount. When performing tests such as the zipping or copying of 50 files, the file access time overhead averaged 1.3 sec on a series of 10 tests, using files with an average size of 33 KB. The additional access time introduced by the sensor is unnoticeable when opening or writing document files. Based on these numbers, we assert that our system has a negligible performance impact to the system and user experience.

4 Concluding Remarks and Future Work

We introduced an architecture that relies on a set of mutually supportive monitoring and auditing techniques. The approach is grounded on the security principle of defense-in-depth and aims to maximize the likelihood of detecting insider malfeasance by employing multiple detection methods. These mechanisms are designed to act synergistically with the goal of making it difficult for an adversary to avoid detection. The architecture combines host-based user-event monitoring sensors with trap-based decoys with the aim of maximizing detection of malicious insider behaviors. To aid in deployment of decoys we built D^3, a convenient system to automatically generate and distribute decoy documents. As part of the system, we introduced the concept of the perfectly believable decoy, and developed host-level sensors to demonstrate it with negligible performance overhead. A user search behavior sensor was also presented demonstrating impressive masquerade detection performance far better than previously published results. We posit the integration of all these sensors raises the bar against insider attackers. The risk of detection is likely far greater. Much work remains to be done, particularly on response strategies and system designs that gather and protect evidence to create a demonstrable sense of risk that an insider may be caught and punished for their malicious acts.

The spectrum of techniques we propose covers a broader range of potential attack scenarios than would any of the constituent components in isolation. To date, we have tested and evaluated the individual detectors in isolation, but have not created an integrated end-to-end solution. A fully integrated detection system proposed here cannot be adequately developed, deployed, and formally tested without a fully capable response component, a separate topic beyond the scope of this paper. We must carefully consider the responses to events that are detected by the detectors. For example, should the user be challenged with questions to ascertain whether they are a masquerader, or should a signal alert a system administrator to immediately revoke a credential that is being misused? These questions are context dependent (*e.g.,* determined by an organization's policies) and typically part of product design in a commercial setting. Testing each component detector poses its own challenges due to the lack of generally available insider attack data as discussed in the sidebar. The sidebar describes the 9 month effort to acquire simulated masquerader data for testing one of the sensors. Acquiring useful "traitor" data to test an integrated system poses challenges we have yet to overcome in a university environment. Even so, we posit that a true controlled study evaluation should be performed when the integrated system responds to insider events it has detected.

References

[1] Richardson R., “CSI Computer Crime and Security Survey”, 2008.

[2] Verdasys. `http://www.verdasys.com/`.

[3] Oakley Network’s Sureview. `http://www.oakleynetworks.com/products/sureview.php`.

[4] Llet, D., “Trojan attacks Microsoft’s anti-spyware”, CNET News, February 9, 2005.

[5] Li, W., Stolfo, S. J., Stavrou A., Androulaki, E., and Keromytis, A., “A Study of Malcode-Bearing Documents”. DIMVA, 2007.

[6] Maloof, M. and Stephens, G. D., “ELICIT: A System for Detecting Insiders Who Violate Need-to-know”. RAID, 2007.

[7] Lee, W., Fan, W., Miller, M., Stolfo, S., and Zadok, E. “Toward Cost-Sensitive Modeling for Intrusion Detection and Response,” Workshop on Intrusion Detection and Prevention, 7^{th} ACM Conference on Computer Security, November 2000.

[8] Yuill, J., Zappe M., Denning D., and Feer F.. “Honeyfiles: Deceptive Files for Intrusion Detection”, IEEE Workshop on Information Assurance, June 2004.

[9] Spitzner, L., “Honeytokens: The Other Honeypot”, Security Focus, 2003.

[10] Spitzner, L., “Honeypots: Catching the Insider Threat”, ACSAC, December 2003.

[11] Yuill, J., D. Denning, Feer, F., “Using Deception to Hide Things from Hackers : Processes, Principles, and Techniques”, Journal of Information Warfare, 5(3):26–40, November 2006.

[12] Bowen, B. M., Hershkop, S., Keroymytis, A. D., Stolfo, S. J., “Baiting Inside Attackers using Decoy Documents,” Columbia University Department of Computer Science Technical Report, CUCS-016-09, 2009.

[13] Katz, John and Yehuda L., Introduction to Modern Cryptography, Chapman and Hall CRC Press, 2007.

[14] Masquerading User Data. `http://www.schonlau.net/intrusion.html`.

[15] Ben Salem, M., Stolfo, S. J., “Masquerade Attack Detection using a Search-Behavior Modeling Approach”, Columbia University Department of Computer Science Technical Report, CUCS-027-09, 2009.

[16] Jiang, X., Wang, X., “Out-of-the-box Monitoring of VM-Based High-Interaction Honeypots”, RAID, pp. 198–218, 2007.