Designing Host and Network Sensors to Mitigate the Insider Threat

Insider attacks—that is, attacks by users with privileged knowledge about a system—are a growing problem for many organizations. To address this threat, the authors have designed an architecture for insider threat detection that combines an array of complementary monitoring and auditing techniques.

Brian M. Bowen, Malek Ben Salem, Shlomo Hershkop, Angelos D. Keromytis, and Salvatore J. Stolfo
Columbia University

In its survey of 522 security employees from US corporations and government agencies, the annual *CSI Computer Crime and Security Survey* for 2008[1] found that 44 percent of respondents cited insider incidents. This number is nearly as high as the 49 percent of respondents who encountered a conventional virus in the previous year. In general, organizations are increasingly recognizing the significance, scope, and cost of the malicious insider problem. Some state-of-the-art defenses focus on forensic analysis and attribution after an attack using techniques such as sophisticated auditing (see www.verdasys.com) and screen capture (see, for example, www.oakleynetworks.com/products/sureview.php). Other commercially available systems aim to prevent, detect, and deter insider attacks. Ideally, the system would prevent such attacks. Although researchers have studied policy-based mechanisms and access control systems for quite some time, no mechanism exists to prevent insider abuse. Monitoring, detection, and mitigation technologies are realistic necessities.

Detection systems aim to identify specific attack patterns or deviations from known, long-term user behavior. Such techniques are typically part of a standalone mechanism rather than an integrated defense architecture. Malicious behavior detection-based insider defenses thus suffer from several problems:

- Behavior is a noisy approximate of user intent in the absence of sufficient contextual information about the user and the overall environment. Consequently, such systems are often tuned to minimize false alerts by being less stringent about what they consider malicious. Although it reduces administrators' workload and users' irritation, such tuning can let some malicious behavior go undetected.
- Because the relationship between behavior and intent is difficult to determine and alarms can be false, it's difficult to confidently take some action (whether automated or manual) in response to an alert.
- Adversaries with some knowledge of these techniques' existence might evade or even disable them. In fact, an increasing number of malware attacks by first disabling defenses such as antivirus software and host sensors prior to undertaking some malicious activity.[2]

To address the malicious insider problem, 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. We don't address the important problem of malicious system administrators who have control over all defensive systems here. This remains a particularly interesting open problem.

Our architecture consists of three main components: a *decoy document-generation component* that leverages uncertainty of the authenticity of information that might be accessed in an unauthorized manner; a *network component* that integrates monitored network traps with the decoy document-generation component to isolate malicious users' activity; and *host-based*

COPUBLISHED BY THE IEEE COMPUTER AND RELIABILITY SOCIETIES ■ 1540-7993/09/$26.00 © 2009 IEEE ■ NOVEMBER/DECEMBER 2009

sensors, collectively named RUU (Are You You?), that collect low-level audit data from which we identify specific user actions.

Threat Model—The Attacker's Sophistication Level

Our architecture uses decoys to deceive, confuse, and confound attackers, ultimately forcing them to expend far more effort to discern real from bogus information. To understand the various decoys' capabilities, we first explore the various levels of attacker sophistication. We broadly define four monotonically increasing levels of insider sophistication and capability to break through the deception our decoys seek to induce. Some insiders will have tools available to assist in discerning decoys from real data. Others will have only their own observations and insights.

- *Low*. Direct observation is the only tool available to the adversary. We strive to defeat this level of adversary with our beacon documents, even though more advanced tools might distinguish decoys with embedded beacons.
- *Medium*. The insider can base decisions on information found through more thorough investigation. For example, if a decoy document contains a decoy account credential for a particular identity, an adversary can verify that the identity is real by querying an external system (such as www.whitepages.com). Such adversaries will require stronger decoy information, possibly corroborated by other evidence sources.
- *High*. The attacker has access to the most sophisticated tools (for example, super computers or other individuals with organizational information). Our notion of a perfect decoy, which we describe later, might 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 use sophisticated tools to try to analyze, disable, and avoid decoys entirely. Defeating this level of threat might still be possible. Consider, for example, someone who knows that encryption is used (and which encryption algorithm), but doesn't know of an easy-to-change operational parameter (the key). Likewise, just because someone knows that decoys are used in the system doesn't mean they can identify them all.

We further define insider threats by differentiating between *masqueraders* (attackers who impersonate another system user) and *traitors* (attackers using their own legitimate system credentials), each with varying levels of knowledge. The masquerader will likely 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.

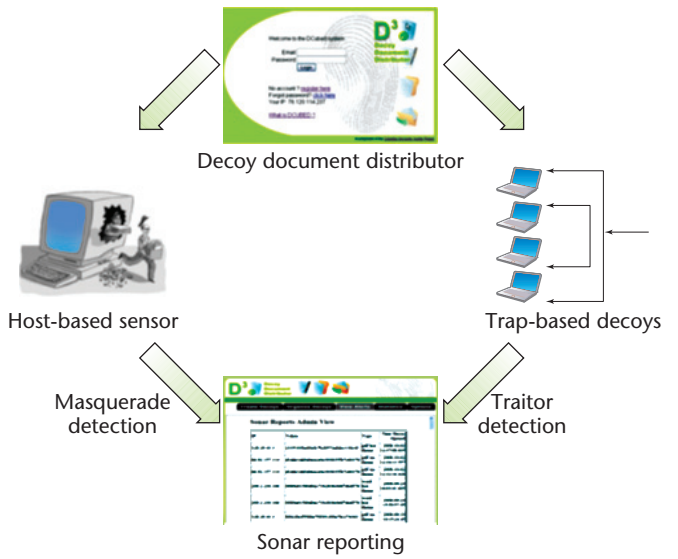

Figure 1. Architecture for monitoring and detecting insider attacks. Host-based sensors monitor user activity to detect malicious users masquerading as other system users. Trap-based decoys attempt to catch attackers who use their own legitimate credentials.

Architecture

Figure 1 shows the architecture, which combines host-based user-event-monitoring sensors with trap-based decoys and remote network detectors. This combination of components makes it difficult for insiders to avoid detection with a low likelihood of misattribution.

Decoy Document Distributor

One of the architecture's core components is the *decoy document distributor* (D^3) system, a Web-based service for generating and distributing decoys. Registered users can employ D^3 to generate decoys for download, or host and network components can use it as a decoy data source.

A decoy's primary goal is to detect malfeasance. Because no system is foolproof, we built D^3 to automatically embed multiple overlapping signals in decoy documents to increase the likelihood of detecting decoy misuse. Any alert generated by the decoy indicates insider activity. Because the attacker might have varying levels of sophistication, we use a combination of techniques to increase the likelihood that one will generate an alert:

- *embedded honeytokens*—computer login accounts that provide no access to valuable resources and

Decoys in Computer Security

Deception-based information resources that have no production value other than to attract and detect adversaries are commonly known as *honeypots*. Honeypots gather intelligence about how attackers operate. They're considered to have low false-positive rates because they're designed to capture only malicious attackers, except for innocent users in occasional mistakes.

Lance Spitzner described how honeypots can be useful for detecting insider attacks.[1] He discusses the use of *honeytokens*, which he defines as "a honeypot that is not a computer."[2] Spitzner cites examples such as bogus medical records, credit-card numbers, and credentials, and describes how they can be used to detect malicious insiders.[1,2]

In current systems, creating decoys or honeytokens is a laborious and manual process requiring administrator intervention. Our work extends these basic ideas to an automated system of managing the creation and deployment of these honeytokens.

Jim Yuill and his colleagues extend the notion of honeytokens with a system to support the creation of bait files, or, as they define them, *honeyfiles*.[3,4] They implement the honeyfile system as an enhancement to the network file server. The system allows the user to make any file within the user file space a honeyfile by creating a record associating a filename to a userid. The honeyfile system monitors all file accesses on the server and alerts users when honeyfiles are accessed. This work doesn't 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.

References

1. L. Spitzner, "Honeypots: Catching the Insider Threat," *Proc. Computer Security Applications Conf.* (ACSAC 03), IEEE CS Press, 2003, p. 170.
2. L. Spitzner, "Honeytokens: The Other Honeypot," *Security Focus*, 17 July 2003; www.securityfocus.com/infocus/1713.
3. J. Yuill et al., "Honeyfiles: Deceptive Files for Intrusion Detection," *Proc. IEEE Workshop on Information Assurance*, IEEE CS Press, 2004, pp. 116–122.
4. J. Yuill, D. Denning, and F. Feer, "Using Deception to Hide Things from Hackers : Processes, Principles, and Techniques," *J. Information Warfare*, vol. 5, no. 3, 2006, pp. 26–40.

are monitored when (mis)used (see the "Decoys in Computer Security" sidebar);
- *embedded beacons* that alert a remote server (called Sonar) at Columbia; and
- *embedded markers* to enable detection by the host-level or network decoy sensor.

Our current D^3 deployment is tailored for a university environment by both the type of documents and the bait within them, but it's easily adaptable for other deployment environments (for example, an arbitrary commercial enterprise). Complete details about D^3, including an evaluation of decoy documents, are available elsewhere.[3] You can evaluate our technology at the Decoy Document Distribution Web site (www.cs.columbia.edu/ids/RUU/Dcubed).

Sonar

The Sonar alert-management system's primary role is to collect alerts triggered by host and network monitors, and individual beacon signals generated by the unauthorized opening of decoy documents downloaded by registered users. When it receives a signal, Sonar sends an email to the registered users associated with the particular decoy. Depending on the type of decoy, some signals are sent directly from a decoy (as is the case with beacons), whereas others require Sonar to poll other resources for information (that is, credential monitoring). Sonar currently monitors several servers for credential misuse, including university authentication log servers and mail.google.com. In the case of Gmail accounts, custom scripts access and parse the bait account pages to gather account activity information.

Decoys and Network Monitoring

The use of deception, or decoys, plays a valuable role in the protection of systems, networks, and information. Clifford Stoll is generally considered the first to use decoys in the cyber domain[4,5] and detailed his experience in the novel, *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage* (Pocket Books, 2000). Stoll's methods included the use of bogus networks, systems, and documents to gather intelligence on attackers, who were apparently seeking state secrets. For example, he crafted bait files and bogus classified documents containing nonsensitive government information and attached alarms to them so 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.

Perfectly believable decoys. To create decoys to bait insiders with various levels of knowledge and maximize their deception, it's important to understand a decoy's core properties. These properties—such as conspicuousness, enticement, noninterference, variability, differentiable, detectability, and believability—guide the design of systems that automate the generation and placement of trap-based decoys.[3] Here, we describe our efforts to maximize decoys' believability.

A good decoy should be difficult to distinguish

from an authentic document from a legitimate source. For concreteness, we build upon the definition of "perfect secrecy" proposed in the cryptographic community[6] and define a "perfect decoy" to be a decoy that is completely indistinguishable from a nondecoy document. 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 keyed-hash message authentication codes (HMACs) and nondecoys 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, 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 to PDF documents. This component adds markers automatically using the iText API and inserts them into the document's OCProperties section. We chose this section because it can be modified on any PDF without impacting how the document is rendered or introducing visual artifacts. The D^3 component creates the HMAC value using a vector of words extracted from the PDF. The HMAC key is kept secret and is managed by D^3, where it's also associated with a particular registered host. Because the system depends on all documents being tagged, another component inserts random decoy markers in nondecoy documents, making them indistinguishable from decoys without knowledge of the secret key.

Trap-based decoys. Our trap-based decoys are detectable outside a host, so they don't require host monitoring nor do they suffer the performance burden characteristic of decoys that require constant internal monitoring. This form of decoy consists of bait information, such as online banking logins provided by a collaborating financial institution (the institution requested that we withhold its name), login accounts for online servers, and Web-based email accounts. Our current deployment uses Columbia University student accounts and Gmail accounts as bait, but we can customize these to any set of monitored credentials. The D^3 Web service manages the trap-based decoys, thereby enabling programmatic access to them from all registered Web-enabled clients. Automating this service enables their distribution and deployment in large volume.

Beacon decoys. Beacons are embedded in documents using methods of deception and obfuscation gleaned from studying malcode embedded in malicious documents as seen in the wild.[7] Beacons 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 uniquely identifies the decoy document and records the IP address of the host accessing that document. The server collects additional data, depending on the document type and rendering environment used to view the beacon decoy document. We implemented the first proof-of-concept beacons in Word and PDF and deployed them through the D^3 Web site. The Word beacons rely on a stealthily embedded remote image that's rendered when the document is opened. The request for the remote image signals to 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—familiar information to all users. In contrast to the HMAC decoys, these documents' believability lies in their content's realism.

As noted earlier, decoys' believability depends on how indistinguishable they are from normal documents. A beacon's network connection can serve as a distinguishing feature. Hence, in their current form, a beacon might be able to ensnare only the least sophisticated attacker. We're currently investigating environments in which we can 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 the signaling mechanisms can fail or be subverted; however, when combined with other mechanisms, their use should increase the likelihood of detection.

Host-Based Sensors

A key technique our architecture uses involves the host-level monitoring of user-initiated events. The host sensor is composed of two components, a *behavior modeler* and *document access sensor.* The first profiles user search actions to form a baseline of normal behavior using anomaly-detection techniques to isolate behavior differences over time. Large deviations from this baseline found in subsequent monitoring signal a potential insider attack. 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.[8]

Second, the host sensor detects when decoy documents containing embedded markers are read, copied,

or exfiltrated. The host-level decoy sensor aims 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 several personalized questions, might establish whether a masquerade attack is occurring.

Our prototype sensor runs on the Windows XP/Vista platforms and relies on hooks placed in the Windows ServiceTable. Malicious rootkits often use this approach; however, whereas traditional rootkits try to remain undetected, the host-level decoy sensor doesn't require operational secrecy. Our threat model assumes attackers know that a system is being monitored, but they don't know the identities of the decoys or the private key the sensor uses to differentiate them. Furthermore, the attacker likely won't know the victim user's behavior—information that isn't as easy to steal as a credential or a key. Given that adversaries might be aware of system monitoring, the system must take special care to prevent the sensor from being subverted or, equally important, to detect if it is subverted.

We have ongoing work that aims to prevent and detect subversion of the sensor. One strategy involves "monitoring the monitor" to detect if the host sensor is disabled using tamper-resistant software techniques. One possible solution relies on out-of-the-box monitoring,[9] in which a virtual machine-based architecture conducts host-based monitoring outside of the host from within a virtual machine monitor.

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 lets the sensor accurately capture data about specific system and user behavior over time. For example, we might check whether an insider has infiltrated the system by modeling search behavior and comparing it to the baseline of normal behavior. We conjecture that users search their own file systems in a unique manner, using only a few specific system functions to find what they're looking for. Furthermore, it's unlikely that a masquerader will have full knowledge of the victim user's file system, and thus might search wider and deeper and in a less targeted manner than would the victim user. Hence, search behavior is a viable indicator of malicious intentions.

Specific sections of the Windows registry, specific dynamic linked libraries (DLLs), and specific programs on the system are involved in system searching applications. For a given time period (10 seconds in our initial experiments), we model a user's search actions. After computing a baseline model, the sensor switches to detection mode and alerts the system if the current search behavior deviates from the user's baseline model. The sensor measures deviation 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's search actions, such as the number of processes being created and destroyed. Presently, we'll integrate this sensor component into the architecture to function with the host sensor that detects decoy document accesses.

To evaluate this model, we gathered user-event data to compute the baseline normal models, and data that simulates masquerade attacks (we describe this data set in the "Data and Evaluation" sidebar). For the user-event data, 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 process's parent, the type of process action (registry access, process creation, process destruction, and so on), the process command arguments, action flags (success/failure), and registry activity results. We also recorded a timestamp for each action. The collected data was automatically uploaded to a server after the students filtered any data they weren't 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. None of the users had prior access to this file system, which we designed to look realistic and include potentially interesting patent applications, personally identifiable information, and account credentials stored in various files. We provided the students with a scenario in which they were to find any data on the file system that could be used for financial gain.

The features used for modeling were essentially volumetric statistics characterizing search volume and velocity and describing the overall computer session in terms of the number of processes running, particularly the number of editing applications. We then trained a one-class support vector machine (ocSVM) model for each user using those features. Next, we extracted the same features from test data after dividing them into 10-second epochs. We tested the ocSVM models against these features, using a threshold to determine whether the user activity during the 10-second epochs was normal or abnormal. If the normal user performs the activity, but the ocSVM model classifies it as abnormal, the system records a false positive. Our results using the collected data and the modeling approach described show that we can detect all masquerader activity with 100 percent accuracy and a false-positive rate of 0.1 percent.

Extensive prior work on masquerade attack detection has focused on the Schonlau data set for evaluation (see the "Data and Evaluation" sidebar). This data

Data and Evaluation

Research in insider attacks is difficult due to the lack of readily available insider attackers or a complete set of realistic data that they generate. Researchers must therefore generate data that simulates insider attacks. The Schonlau data set (www.schonlau.net/intrusion.html) is the most widely used in academic studies. It consists of sequences of 15,000 Unix commands generated by 50 users with different job roles, but the data doesn't include command arguments or time stamps. Researchers have used the data for comparative evaluations of different supervised machine learning algorithms. The Schonlau data isn't a "true masquerade" data set. They approximate masqueraders by randomly mixing the data gathered from different users to simulate a masquerader attack, making the data set perhaps more suitable for author identification studies.

An alternative approach to acquiring 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 in which red-team users behave as inside attackers. Because they involve human volunteers, these studies are typically subject to institutional review board approvals. The process is costly in both time and effort but is sensible and appropriate to protect volunteers' personally identifiable data. Marcus Maloof and Gregory Stephens took this approach in evaluating Elicit.[1]

We gathered data from 34 users, all computer science students at Columbia University, by distributing host sensors that upload system event data during normal system use. The population of student volunteers assures us that the data they generate is derived from sources with a common role in the organization. Hence, variations in the user behavior and data aren't attributable to different job functions, as is undoubtedly the case with the Schonlau data set. We also gathered data from 14 paid volunteers who emulated masquerade attacks on equipment in our lab. The data set, which we call the RUU (Are You You?) data set, is more than 8 Gbytes and is available to legitimate researchers for download at http://www1.cs.columbia.edu/ids/RUU/data. The data collected for each user averages about five days of normal system use, ranging in the extreme between one and 59 days, and an average of more than 1 million records per user. Preliminary results using this data and the abnormal search behavior sensor described in the article show that the red team of masqueraders deviate substantially from ordinary system users.[2]

References

1. M. Maloof and G.D. Stephens, "Elicit: A System for Detecting Insiders Who Violate Need-to-Know," *Recent Advances in Intrusion Detection* (RAID), LNCS 4637, Springer, 2007, pp. 146–166.
2. M. Ben Salem and S.J. Stolfo, *Masquerade Attack Detection Using a Search-Behavior Modeling Approach*, tech. report CUCS-027-09, Dept. of Computer Science, Columbia Univ., 2009.

set has served as a common gold standard for researchers conducting comparative evaluations of competing machine learning algorithms. The basic paradigm this work follows is a supervised training methodology in which 5,000 commands from each user serve as training data for the user's normal behavior model. This model is evaluated against data not used in training from the user's command data set but embedded in a random location with another randomly chosen user's data. Performance results indicate the accuracy of the classifiers learned by a particular machine learning algorithm in identifying foreign commands—that is, those blocks of commands deemed abnormal.

The model we chose to embed in the user search command sensor differs from these prior bag-of-command based models. Our current studies analyze user command events and the rates at which commands are issued using the RUU data sets described in the "Data and Evaluation" sidebar. The models estimate accuracy with respect to classification errors measured for each 10-second epoch of user events. Furthermore, whereas the Schonlau data consists of Unix commands, the RUU data sets contain user events created in a Windows environment.

To compare our results with these prior studies, we 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 Schonlau data sets have no timestamps, so a direct comparison with our modeling technique isn't 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 data sets for certain periods of time), our experiments show that we can achieve a detection rate of 100 percent with a false-positive rate of 1.4 percent. This is a 78 percent improvement in false-positive rate over the best reported classifier in the prior Schonlau work. Indeed, none of the prior work reports a 100 percent detection rate with any reasonable false-positive rate. If we assume it takes on average longer than 20 seconds to issue a user command, the false-positive rate drops even further.

The comparison might not be entirely fair because the models and data are quite different even though human users generate the data. The use of temporal statistical features from the RUU data set is crucial in modeling user behavior, leading to far more accurate results than blocks of commands. Furthermore, we focus on user search events, limiting the amount of data

analyzed and reducing the learning task's complexity. Complete details of the volumetric and contextual features we used when modeling user commands and the results achieved are reported elsewhere.[10] This work also describes the results of reducing the data in the Schonlau experiments. The RUU data sets can serve as a modern gold standard for other researchers performing similar studies.

Detecting perfectly believable decoys. The host sensor also detects malicious activity by monitoring user actions directed at HMAC-embedded decoy documents. Any action directed toward a decoy suggests malicious activity. When an application or process accesses a decoy document, the host sensor initiates a verification function. This function differentiates between decoys and normal documents by computing a decoy HMAC (as described earlier) for the document in question and comparing it to the one embedded in the document's OCProperties section. If there's a match, the document is deemed a decoy, and the sensor triggers an alert; otherwise, the document is deemed normal, and no action is taken.

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

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

The sensor detects the loading of decoy files in memory with 100 percent accuracy by validating the HMAC value in the PDF files. However, as we discovered during our validation tests, decoy tests can be susceptible to nonnegligible false-positive rates. The problem encountered in our testing was created by antivirus scans of the file system. The scanning process's numerous file accesses generated spurious decoy alerts. Although we're engineering a solution to this particular problem by ignoring automatic antivirus scans, our test highlights the challenges such monitoring systems face. Many applications on a system legitimately access files indiscriminately. Care must be taken to ensure that only (illicit) human activity triggers alerts. Future versions of the sensor might filter file touches not triggered by user-initiated actions but rather by routine processes, such as antivirus scanners or backup processes. Nevertheless, this issue demonstrates a fundamental design challenge to architecting a security system with potential interference from competing monitors.

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

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've tested and evaluated the individual detectors in isolation but haven't created an integrated end-to-end solution. A fully integrated detection system as proposed here can't be adequately developed, deployed, and formally tested without a fully capable response component—a topic beyond this article's scope.

We must carefully consider how detectors respond to events. For example, should the detector challenge the user with questions to ascertain whether the user is a masquerader, or should a signal alert a system administrator to immediately revoke a credential that's being misused? These questions depend on the context (for example, an organization's policies might determine its response) and are typically part of product design in a commercial setting.

Testing each component detector also poses challenges due to the lack of generally available insider attack data, as discussed in the "Data and Evaluation" sidebar. 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 in which the integrated system responds to insider events. □

Acknowledgments

This material is based on work supported by the US Department of Homeland Security under grant award number 2006-CS-001-000001-02 and the Army Research Office

under grant ARO DA W911NF-06-10151. 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 US Department of Homeland Security or the Army Research Office.

References

1. R. Richardson, *CSI Computer Crime and Security Survey*, Computer Security Inst., 2008.
2. D. Llet, "Trojan Attacks Microsoft's Anti-Spyware," *CNET News*, 9 Feb. 2005.
3. B.M. Bowen et al., *Baiting Inside Attackers Using Decoy Documents*, tech. report CUCS-016-09, Dept. of Computer Science, Columbia Univ., 2009.
4. J. Yuill et al., "Honeyfiles: Deceptive Files for Intrusion Detection," *Proc. IEEE Workshop on Information Assurance*, IEEE CS Press, 2004, pp. 116–122.
5. L. Spitzner, "Honeytokens: The Other Honeypot," *Security Focus*, 17 July 2003; www.securityfocus.com/infocus/1713.
6. J. Katz and L. Yehuda, *Introduction to Modern Cryptography*, Chapman and Hall CRC Press, 2007.
7. W. Li et al., "A Study of Malcode-Bearing Documents," *Proc. Detection of Intrusions and Malware, and Vulnerability Assessment* (DIMVA 07), LNCS 4579, Springer, 2007, pp. 231–250.
8. W. Lee et al., "Toward Cost-Sensitive Modeling for Intrusion Detection and Response," *J. Computer Security*, vol. 10, nos. 1–2, 2002, pp. 5–22.
9. X. Jiang and X. Wang, "Out-of-the-Box Monitoring of VM-Based High-Interaction Honeypots," *Recent Advances in Intrusion Detection* (RAID), LNCS 4637, Springer, 2007, pp. 198–218.
10. M. Ben Salem and S.J. Stolfo, *Masquerade Attack Detection using a Search-Behavior Modeling Approach*, tech. report CUCS-027-09, Dept. of Computer Science, Columbia Univ., 2009.

Brian M. Bowen *is a PhD student in Columbia University's Department of Computer Science, where he's a member of the Network Security Lab and the Intrusion Detection Systems Lab. He's also a senior member of the technical staff at Sandia National Laboratories and is enrolled in the Sandia Doctorate Study Program. Bowen's research interests include network security, primarily trap-based defense using deception techniques. He has an MSc in computer science from Stony Brook University. Contact him at bmbowen@cs.columbia.edu.*

Malek Ben Salem *is a PhD student in computer science at Columbia University, where she works in the Intrusion Detection Systems Lab. Her research interests include developing novel data mining techniques and applying them to computer security in general, and to host intrusion detection in particular. Ben Salem has an MSc in computer science from Columbia University. Contact her at malek@cs.columbia.edu.*

Shlomo Hershkop *is the assistant director of the Computing Research Facilities and is an adjunct assistant professor in Columbia University's Department of Computer Science. His research interests include data mining security, anomaly detection, and email modeling and analysis. Hershkop has a PhD in computer science from Columbia University. He's a member of the IEEE, the ACM, and Usenix. Contact him at shlomo@cs.columbia.edu.*

Angelos D. Keromytis *is an associate professor in the Department of Computer Science and director of the Network Security Lab at Columbia University. His research interests include most aspects of security, in particular systems and software security, cryptography, and access control. Keromytis has a PhD in computer science from the University of Pennsylvania. He is a senior member of the ACM and the IEEE. Contact him at angelos@cs.columbia.edu.*

Salvatore J. Stolfo *is a professor in the Computer Science Department at Columbia University and the director of the Intrusion Detection Lab, which pioneered the use of data analysis and machine learning techniques for the adaptive generation of novel sensors and anomaly detectors for a variety of computer security tasks. His research interests include parallel computing, artificial intelligence, data mining, and computer security and intrusion detection systems. Stolfo has a PhD in computer science from New York University's Courant Institute. He is a board member for* IEEE Security & Privacy. *Contact him at sal@cs.columbia.edu.*