Active Authentication using File System Decoys and User Behavior Modeling: Results of a Large Scale Study

Jonathan Voris[b,d], Yingbo Song[b], Malek Ben Salem[a,b], Shlomo Hershkop[b], Salvatore Stolfo[b,c]

[a]*Accenture Technology Labs*
[b]*Allure Security Technology Inc.*
[c]*Columbia University*
[d]*New York Institute of Technology*

Abstract

We propose methods to enhance existing user authentication paradigms (such as passwords) with continuous *active* authentication. Our system adds additional levels of security without burdening the user with more credentials to manage. We utilize two complementary authentication modalities to validate user-identity: 1) behavior profiling for user-system interaction, and 2) baiting adversaries using automatically distributed file-decoy tripwires. We present the results from a 160-subject user study used to validate our system. Our results show that the presence of decoy documents on a system does not interfere with normal user activities, and that, with 95% accuracy, our system will detect an intrusion within 15 minutes with at most one false-positive for 40 hours of user activity.

Keywords:
intrusion detection, masquerader attacks, behavioral biometrics, active authentication, decoys, honeyfiles

1. Introduction

Data theft remains one of the main cyber risks incurred by organizations and individuals alike. Verizon's data breach investigation report [1] analyzed more than 2,500 data breaches, resulting in more than 1.1 billion compromised records over the nine-year range of the study. It reports that 92% of the investigated data breaches were perpetrated by external attackers. It also highlights the fact that 66% of these data theft incidents took months or longer to be discovered, while most of the attacks were not sophisticated. The report reveals that neither the existing access control technologies designed to prevent unauthorized access to data, nor the more sophisticated Data Loss Prevention technologies have been effective at preventing such attacks against enterprises.

In fact, existing access control mechanisms rely mostly on passwords, which have long been used as the first (and often only) step in authentication. While mostly reliable, their main downside is that passwords are often easily transferable, stolen, or guessed. Several studies have illustrated the limitations of using passwords for access control [2, 3, 4]. These reports show that passwords can be hard to remember and manage, as password policies now require longer and more complex passwords that have to be changed more frequently.

Passwords are also typically only used at the beginning of a session, and the user is implicitly trusted for the duration. In this paper, we propose to enhance existing authentication methods with continuous active authentication techniques based on modeling and validating user-system interaction and by using file system decoys. Our goal is to detect unauthorized misuse of a user's credentials by periodically and transparently validating the monitored behavior against known profiles for legitimate activity. "Behavior", in this paper, is a general term for how a user interacts with the computing environment, *e.g.* open files, processes, accessing the network *etc.*

We propose two complementary authentication modalities: 1) user behavior modeling and validation, and 2) baiting adversaries using file decoys. Although modeling users' system interactions has been studied in previous work [5], this work validates our approach with large-scale user studies, including a masquerader study where users are asked to mimic insiders who have gained illegitimate access. In this paper, a "masquerader" refers to someone who is accessing another person's computing environment without authorization, using that person's (the victim's) credentials. This project also introduced more advanced host sensors which were better able to capture user behavior with respect to their operating system. Additionally, this paper is the first to assess the interaction between behavior analytics and decoys.

Our approach is based on the hypothesis that masqueraders will be unfamiliar with the target environment, and will act differently than a legitimate user. We posit that it is possible to enumerate a set of system-level measurements suitable for detecting masqueraders whose actions are motivated by data theft. The owner of a system, who is intricately familiar with their own system, should access their own data in a targeted, purposeful, and verifiable manner, while a masquerader, who is attempting to steal information from an unfamiliar system, would interact with the system in a measurably abnormal manner. Prior work has shown that such differences in search behavior may be used to detect masquerade attacks motivated by data theft [6], a concept which is enhanced and refined using the broader dataset collected as part of this study.

Use of decoy files is an extension of tripwire-based detection paradigm, and have been shown to be effective in identifying malfeasance [7, 8, 9, 10]. It leverages the fact that masqueraders would not be able to recognize such traps in an unfamiliar environment. This paper provides details of a formal user study that corroborates this conjecture, and extends upon this with new decoy-based technologies. In our system, decoys are generated and distributed automatically, and after a number of decoy file touches, the system will notify the true owner via an out-of-band alert and execute a mitigation strategy. Monitoring access to decoy data can thus provide additional and immediate high-confidence detection in a practical and cost effective fashion.

Prior user studies in this area have been limited in scale [5]. This paper broadens and extends previous research with a more principled design for the behavior modeling algorithms. While prior use of decoys for security has been studied [10, 8, 11, 9], they lacked a large-scale evaluation for how often legitimate users would trigger on their own decoys (false positive rate); this is addressed in our study. Our experiment shows that users touch decoys frequently when first placed in the file system, but soon their curiosity decays and additional touches become rare, thus these files do not interfere with normal operations.

Our user study included 209 users who have installed our sensors on their desktop and laptop computers. The sensor software collected pertinent features as these participants utilized their systems as they typically work on a day-to-day basis. In addition, 22 users recruited to act as masqueraders who have gained access to a target system; these participants were instructed to steal information while their (malicious) behaviors were recorded [10]. This study improves upon prior classification performance by demonstrating that our latest behavior-based classifier achieves a baseline detection accuracy of 68% at 1% FP rate, and when used in continuous active authentication, has an expected 95% chance of detecting a malicious session within 15 minutes, under the constraint that only *one* false positive is generated per typical

40-hour work week.

This paper is organized as follows. Section 2 reviews pertinent previous work in the areas of continuous authentication and deception-based computer defenses. Next, Section 3 presents our active authentication approach and establishes the details of our threat model. Section 3.1 explains the implementation of the host sensors. Section 4 presents our experimental setup, while Section 5 discusses the details of our user behavior models. We evaluate the proposed active authentication approach in Section 6. Finally, we conclude the paper with Section 7.

2. Related Work

This journal paper is an extension of our prior workshop paper [12] on the same topic. The main contributions of this work include a significantly larger user study along, updated feature sets, and new performance results from this larger study. This section summarizes prior work related to the proposed detection modalities: continuous authentication and deception or "decoy" based detection.

2.1. Continuous Authentication

Behavior-based biometrics can be used to provide non-intrusive user authentication. Several such approaches have been proposed to authenticate users **at the beginning of a user session**. Most of these are based on modeling mouse movement patterns or typing patterns, typically referred to as keystroke dynamics [13, 14, 15, 16, 17]. Other similar works sought to attribute a user session to a given user **at the end of a user session**, *i.e.* once the session is over [18, 19, 20, 21]. However, little work focused on behavioral biometrics as active authentication mechanisms throughout the entire user session.

Some researchers proposed approaches relying on soft biometric traits for authentication, such as a user's skin color or the color of their clothes [22], although such security mechanisms are easily evaded. Furthermore, they continue to rely on the initial login as the main protection mechanism and do not provide any a method to augment password-based initial authentication solutions. Our approach focuses on continuous authentication through a user's session.

Keystroke and mouse dynamics have been the main continuous authentication method of the past, as they do not require any specialized equipment or additional hardware. Various modeling approaches have been proposed with varying accuracy results. Shen, Cai and Guan proposed a continuous authentication mechanism based solely on mouse usage patterns [23]. They distinguished between patterns, or frequent segments of mouse dynamics, and holistic behavior, the more intermittent pieces. They found that "patterns" are more descriptive of user behavior as they are stable features across user sessions. The same patterns emerged as discriminative features, *i.e.* descriptive of unique user behavior.

Pusara and Brodley built C5.0 decision trees on the basis of users' mouse movements within a time window of configurable size, and used the models to re-authenticate users [24]. The data was collected in a free environment, *i.e.* from the users' own computers. But the user sample was too limited to report generalizable results. The mouse movement-based models, which were trained using data from all 11 users, achieved an average false-acceptance rate (FAR) of 1.75% and an average false-rejection rate (FRR) of 0.43%, but the verification time took up to 15 minutes depending on the size of the time window.

Meng, Gupta, and Gao have recently investigated whether some people can change their typing pattern and mimic a targeted victim's keystroke dynamics when typing their password [25]. They developed a system that trained impostors to mimic the password typing behavior of a certain victim. The system, called *Mimesis,* provided feedback to the impostors, each time they typed the victim's password. Experiments were performed with two different passwords: an easy to type, all lower-case password, and a harder one, with a combination of lower and upper case letters, as well as special characters. The experiments demonstrated that the attackers could quickly learn and adopt the victim's typing behavior, particularly when typing the easy password, showing that keystroke biometrics might not be effective in mitigating weak passwords.

Although some promising results have been reported, authentication using mouse dynamics and free-text keystroke dynamics have not matured as a reliable technology. They have been tested within limited and pre-defined settings (*e.g.* while working on a specific task or interacting with one application), and therefore have not dealt with the intrinsic behavioral variability as users interacts with different applications.

In our study, we do not restrict our measurements to specific software applications or specific hardware devices. Instead, we provide an installable sensor and monitor user's behaviors as they interact with their own computing environments, at work and at home. Furthermore, our approach is less vulnerable to changes in user behavior due to physiological factors such as pain or injury, which can affect keyboard or mouse dynamics.

2.2. Decoys and Deception

The earliest known example of deception in computer security literature is a discussion by Hollingworth about how "entrapment modules" can be used to augment computer defenses [26]. Computers intended to serve as bait for malicious actions are typically referred to as "honeypots," while networks of such machines go by "honeynets." These systems are typically self-contained and deployed to appear as though they are useful components of a broader network. However, they are actually partitioned from valued network components and do not store any meaningful data themselves.

"Honeytokens" extend the concept of honeypots in a way that is more valuable against masquerade attacks and other insider threats [27]. Coined by Spitzner, honeytokens apply the concept of deception-based defense to computer systems at a finer granularity. Instead of entire phony machines or networks, honeytokens are discrete chunks of information which are designed to lure attackers with data that appears enticing, but is in fact spurious. A fabricated set of personally identifiable information is a representative example of a honeytoken. Files, which house such deception data, are referred to as "honeyfiles" [28]. Decoys described in this paper are variations of honeyfiles. They are used as tripwires in our system, but are also capable of acting as beacons to send signals back to a central server.

3. Active Authentication

We propose an active authentication modality to validate user identity. This is based on profiling users' unique behaviors as they interact with their personalized computing environments. Our sensor monitors file system access, process- and networking-related behaviors, as well as access to decoy files. Our objective is to capture a unique *cognitive fingerprint* for each user. Our behavior models are built on meta-data level measurements, and measures occurrences of distinct events without using data content. For example, we do not make use of information such as which websites a user visits or the contents of any particular file; rather we look at when such events occur, with respect to time, order, and volume (rate). Active authentication is performed by periodically and *transparently* validating a user's behavior against pre-trained models.

In our study, masqueraders are assumed to have bypassed the initial authentication (password login) phase. Masqueraders are expected to have limited knowledge of the system upon initial access – specifically, they would have limited knowledge of where valuable resources are located and where decoy files are placed. When such decoys are accessed ("touched"), the sensor would automatically respond via an additional identification challenge or act according to some other security policy.

With regard to behavior, the masquerader can not be expected to simultaneously explore an unknown environment and search for valuable information while accurately mimicking the victim's normal profile; this information would be unknown to the masquerader. Even if an attacker is able to successfully emulate recent user activity, via program histories, for example, our approach would still limit the duration and scope of such an attack, preventing large-scale exfiltration of data. Although collecting data to verify these assumptions fell outside of the scope of the present work, we plan to conduct studies which verify the implausibility of evading detection by observing and mimicking an authentic user's system usage behavior as future work.

We developed sensors for both Windows and OS X. They were designed to be lightweight and transparent, and monitor user-OS interaction continuously. Variations of these sensors were used to collect the dataset used in our experiments. For brevity, only the Windows sensor is described in this paper (all references are to this version); the OS X version differs only in implementation details and is functionally equivalent.

3.1. Modular Host Sensor

The sensor is designed in a modular fashion to allow easy hooking into native subsystems, and to provide the ability of add additional components as needed. These modules hook into low-level operating system libraries to minimize data collection overhead. The current list of implemented modules are given below:

Process Monitor: This module monitors process creation and termination events. It records the process ID, its executable path, its state, the time at which the process was created or terminated, and the parent process ID. It polls the system at a fixed interval to take a snapshot of all processes at the moment, comparing it with the last snapshot to determine which processes are newly spawned and which ones have terminated. The time between when a process appears and when it is removed determines its life span.

Window Touches Monitor: The windows monitor logs when a new graphical window is switched to the foreground. The monitor records foreground window information, including its title, the current thread ID and process ID responsible for the window. By noting the time when the user switches between windows the sensor can log how long a specific window was in focus.

File Touches Monitor: The file monitor audits the system for various file touch events. These notifications are raised when a file is created, renamed, deleted, or any part of it is changed, including its name, attributes, creation time, last access time, last write time, security settings, and size. In addition, this monitor will check the file touched to determine if the file accessed is a decoy. System files touches are filtered by the sensor for optimization purposes.

Port Monitor: The port monitor sensor queries the state of all the TCP connections on the system. It examines all the connections, filters out local connections. The connections are compared with those polled in the previous cycle to identify new and terminated connections. The port monitor logs the opening and the closing of ports. The particular port number and its duration is then logged.

Log Monitor: The log monitor uses the existing System and Event Log infrastructure in the local operating system to monitor log entries. OS level audits ensure that specific security events will trigger a system or security log event which the sensor logs.

Decoy deployment module: The behavioral sensor also features the Decoy Distributor Tool (DDT) described in Section 3.2. Integration with DDT enables decoy deployment automation and fast detection of decoy touches, which are logged by the host sensor.

One challenge in behavior profiling is how to isolate *user*-triggered eventes from *system*-triggered events. Opening a web-browser is a user event; caching a file is a system event. Intuitively, only user-triggered events are relevant for active authentication, and we want to ignore actions generated without direct user influence, such as via background processes: anti-virus, file back-up, version control software, *etc.* Within the scope of our study, we relied on white-listing known system background processes. This includes `rundll32.exe` and `csrss.exe`. Fine turning this measurment is the subject of our continued research.

3.2. Decoy Technology

In our system, decoys are normal document files such as PDFs and Word documents that exist within a file system. Without opening them, they are indistinguishable from other files of the same type. However, when they are opened, the system will immediately trigger an intrusion response. Decoy technology is founded on the idea that user workstations are personalized to them – users who are familiar with their own systems will learn (over time) to avoid their own tripwires, while intruders who are unfamiliar with the target environment would inadvertantly trigger them.

We developed a decoy generation system [11] to efficiently create, manage, and monitor decoys with minimal user involvement. Users can generate and download personalized decoys from our servers. These decoy files contain unique watermarks that our sensors recognize. Our sensor hooks into the underlying process that Windows uses to open files to scan for these watermarks for every opened file. This includes copying, moving, and deleting files. As long as our host sensor is running, a decoy file cannot be manipulated without being detected. Our system is accessible as a web site that offers several decoy related services to users [29].

To automate decoy document generation and distribution we developed the Decoy Distributor Tool (DDT). DDT is a tool designed to disseminate decoys throughout a file system with minimal user interactions. It does not require any prior knowledge of the organization or the content of the file system where decoys are to be placed. It reduces the complexity of decoy document distribution to simple choices of the amount of decoys desired and the portion of a file system on which they are to be placed, thus substantially lowering the time required to establish a system of insider threat sensors, while retaining all of the security benefits of manual decoy usage.

3.3. Decoy Document Placement

A study performed in [9] demonstrated that the placement of decoy documents greatly affects the probability of a user accessing these files. Locations in which decoys are placed on a file system should be carefully selected so that the decoys remain conspicuous to malicious insiders but do not impede a legitimate user's normal actions. The DDT scans from the root folder specified on the target machine and identifies two sets of folders: one set of 10 folders with the most recently accessed documents and another set of 10 folders containing the greatest number of files with common document extensions *.pdf*, *.doc*, *.docx*, *.ppt*, *.xls*, *.txt*, *.html*, and *.htm*. Selecting the most populated and most recently accessed folders increases the conspicuousness of decoys, since these are likely directories that would be the most probable targets for malicious insiders. The DDT first chooses folders from the intersection of these two sets, i.e. folders with both properties of most recently accessed and the greatest number of files with common document extensions in it. It then proceeds to populate the remaining 10 most crowded directories, followed by any of the top 10 directories showing the most activity that have not yet been included.

However, one might argue that placing the decoy documents in high-traffic locations would interfere with the user's normal activities and would drive the user to accidentally open these files occasionally. In an effort to evaluate decoy files' interference property, we administered a user study which we referred to as the RUU2 ("Are You You?") project [30]. Figure 1 depicts the amount of decoys that were touched by a RUU2 user on average as our sensor application proceeded to monitor their computer usage. Data was taken from 200 RUU2 clients. There is a large spike in decoy access immediately after observation commences. This represents both the initial deployment of decoys as well as initial user curiosity regarding the presence of these files on their system. After this bootstrapping procedure, decoys are less likely to be accessed.

In Figure 2, we focus on a single RUU2 participant to provide a more detailed look at decoy access behavior for a particular user. Despite the fact that the user's overall file system activity remains constantly high throughout our observation window, the only time this particular user accessed his or her decoy files was at the very beginning of the study, when half of the 40 decoys that were distributed were touched. This plot demonstrates the differences between decoy access rates and the number of times users touch their file system in general. Users are far less likely to access their decoys after their initial curiosity has died down.

When blending decoy documents into a folder with existing documents, the DDT chooses a date for the file depending on the naming convention selected. If the naming method appends either "-final" or "-updated" to a filename, the DDT will set the document's creation date to at most 48 hours after the most recently created document in the destination directory. The document's last modified date will then be set to up to two days after the resultant creation date. If an adversary decides to sort the contents of that folder by date, the decoy documents would appear at the top of the sorted list, making them prominent and likely more attractive targets.

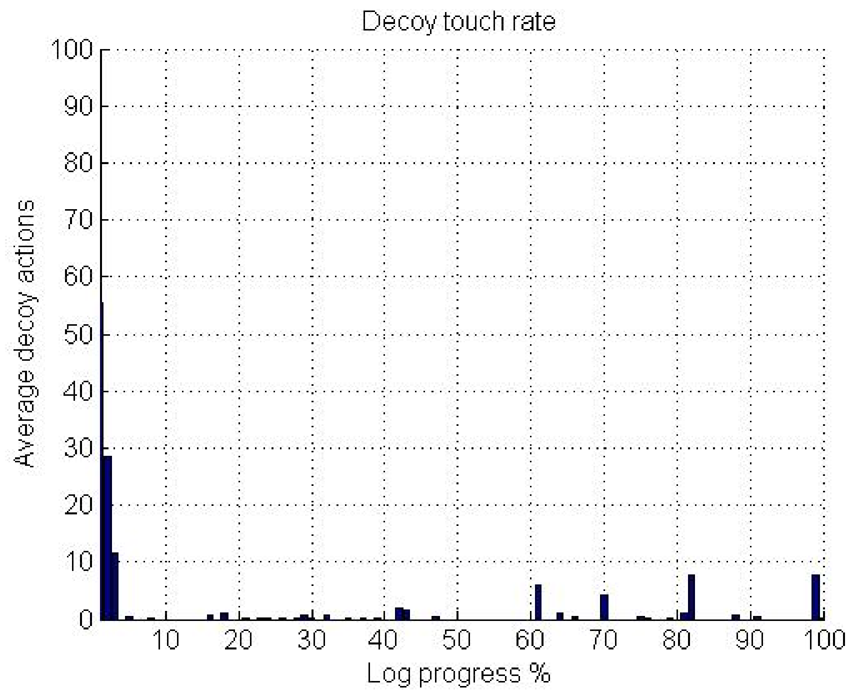

Figure 1: Average Decoy Touch Rate Across All RUU2 Users Aligned by Log Process Percentage

4. User Study Design and Setup

We conducted a multi-month IRB-approved user study with over 200 human subjects for the purpose of data-collection. The volunteers in this study installed a version of our sensor that was tuned to passively collect behavior measurements, and ran these sensors for varying lengths of time. These high-level measurements, described in the following section, are anonymized to remove personal information and uploaded to our central server. The data was subsequently used to test behavior models, measure decoy access rates, and evaluate detection and false-postive rates. All data is assumed to reflect normal (benign) user activity with no malicious (masquerader) instances. In a separate study, we recruiters volunteers to act as masqueraders. The data we gathered over the course of this study was treated as the malicious class of measurements and analyzed to measure the efficacy of our approach.

For the first phase of the study, users were required to use their own computers, so no experimental framework was required. We discuss the

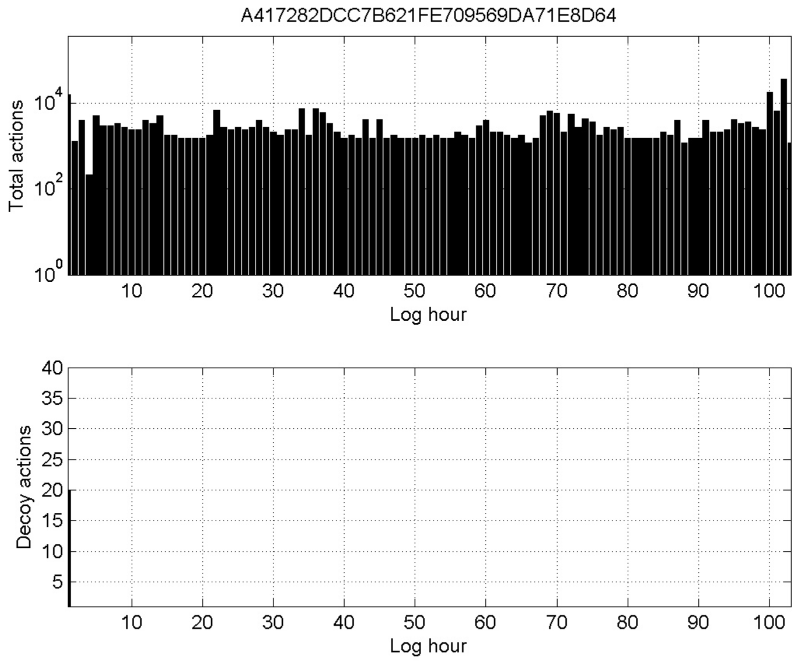

Figure 2: A Sample of a RUU2 User's Total Actions versus his or her Decoy Actions

design of this portion of our study in Section 4.1. The second part of this study required a controlled working environment in order to capture instances of masquerader behavior. We detail this study in Section 4.2.

4.1. Data collection and Decoy Analysis

The goal of this study was to collect data and measure the extent to which our authentication approach interfered with the everyday usage patterns for legitimate users. We recruited over 200 users to participate in a long-term (multi-month) study. We reached out primarily to students studying at the computer science department at a large university in the United States. This study was advertised by posting flyers around campus, via email solicitation, and posting on popular classified listings web sites.

Potential applicants were directed to a website that provided a summary of study details, such as what data would be recorded and our experiment objectives. This web site also included a download link to our sensor software. After installing the sensor on their systems, users were directed to a web based form which asked them to register their sensor's identification number with their contact information. Our pool of users were drawn primarily from the University community.

After installing and registering the sensor software, users were asked to run DDT, which was included as a component in the sensor. This step generated and distributed 40 decoy files throughout their home directories. A variety of decoy file types were generated, resembling typical files, such as fake tax return statements, credit card receipts, project-related files, to-do lists, *etc.*

Out of a total of 209 installations, 49 were either test sensors used for development purposes or were users who contributed insufficient amounts of data for modeling. This left us with a pool of 160 users who contributed sufficient data. Roughly 120 of these were Windows users, and the rest were OS X users. The capture period spanned one week on average; some were as short as a few minutes while others extended up to three months.

4.2. Detection Validation

This second, separate study was aimed at collecting instances of masquerader activity, and was much shorter in duration in order to reflect real intrusion instances. The goal is to record and measure masquerader activity in a controlled environment as they attempt to seek out data in an unfamiliar system. Thirty decoy files were distributed within the file system. This study included 22 participants, who were all compensated $10 for their assistance.

We created a target system for attacks by creating a virtual image of a file system which had been actively used in our lab as part of normal research for years. We used an existing machine that had seen real use instead of a fresh installation because it had gathered data and applications in a way that would be difficult to believably replicate. Using a refreshed virtual machine for each masquerader test, instead of a single computer whose state would change for each trial, each participant was assured to be working in exactly the same environment. The file system on the machine was a replica of a real file system with real user data copied from a student's laptop.

The DDT was then used to fetch 40 decoy files from our decoy creation site [29] and sow these files throughout our test file system. Forty files were selected to create an even comparison to the study performed in [9]. In addition to its other tasks including recording user interactions with the system, we used our sensor software to monitor access to these files.

As discussed in Section 3.3, the DDT places decoys in the most active directories according to file volume and recentness of activity. Table 1 lists the locations that our distribution prototype identified as being the most suitable for decoy deployment. Note that there are 17 target directories because we opted to use the 10 largest directories as well as the 10 directories with the most recent activity; the first three directories on the list were among the top ten on the file system for both recent activity as well as the amount of files they contained.

1	C:\Documents and Settings\username\Desktop
2	C:\Documents and Settings\username\My Documents
3	C:\
4	C:\Documents and Settings\username\My Documents\Downloads
5	C:\Documents and Settings\username\My Documents\WebServices
6	C:\Documents and Settings\username\Downloads
7	C:\Documents and Settings\username\My Documents\ BusinessAnalysis
8	C:\Documents and Settings\username\My Documents\Mortgage
9	C:\Documents and Settings\username\My Documents\Confidential\ Employee Evaluations
10	C:\Documents and Settings\username\My Documents\Taxes
11	C:\Documents and Settings\username\My Documents\Financial
12	C:\Documents and Settings\username\My Documents\Financial\ Bank Statements
13	C:\Documents and Settings\username\My Documents\Personal
14	C:\Documents and Settings\username\Desktop\Insurance
15	C:\Documents and Settings\username\My Documents\Architecture
16	C:\Documents and Settings\username\My Documents\CrazyBusy
17	C:\Documents and Settings\username\My Documents\work_files

Table 1: Decoy Destination Directories Selected by the DDT

Each masquerader was provided a scenario that asked participants to put themselves in the role of a person under financial duress. The scenario further described a situation in which the participant would have good reason to resent a coworker who may have (financially) lucrative data on his or her machine. Note that this is critical to manipulate the intention of the participant and to give them a motive for malicious action. In this paper this is referred to as the "Masquerader dataset."

We prepared two variants of the scenario description which was presented to the participants in the detection validation study. To avoid introducing unnecessary confounding factors, these study instruments differed in one small but important aspect. One scenario specifically informed participants of the presence of decoy documents in the test environment while the other did not. The effect of this variable on the behavior of our simulated masqueraders is explored in Section 6.4.

5. User Behavior Modeling

Our sensor records system-level events that include: process creation and deletion, window manipulation, network, and file-system actions. These events are stored locally as sensor logs on the host machine, where they are continuously analyzed by a statistical-pattern-recognition module contained within the sensor. This module applies a sliding window to the sensor logs and extracts statistical features from this data. These features are defined as the occurrence rates of different categories of monitored system-level events, and are used as our unit of measurement for a "behavior sample" at any particular time. This section describes our feature selection process, our modeling approach, and discussions on measurement window sizes and how it affects accuracy. Experiments show that a five-minute sliding window would work well in practice.

For each behavior sample, classification involves three distinct steps: 1) feature extraction from sensor logs, 2) feature enrichment/normalization, and 3) comparison against a known profile to decide whether the instance is considered normal or abnormal.

5.1. Feature Extraction

Sensor logs are formatted as a tuple consisting of time, action category, action type, and details related to that action, organized in the following form:

[time stamp, category, action type, detail]

so a sequence of measured events may look like:

```
...
00:00:00 01/01/2013, file, open,   c:\windows\somefile.txt
```

```
00:00:02 01/01/2013, file, delete, c:\windows\someotherfile.txt
00:00:14 01/01/2013, proc, create, c:\programs\prog1.exe
...
```

We segregate these measurable events into 20 canonical categories. For a particular time-window, feature extraction involves counting the number of occurrences for each of these events within that particular window. These categories were selected through an iterative process of tuning the sensor's activity collector and adjusting the statistical models to see what information was available from the underlying operating systems as well as what worked best given the selected environment. The goal was to produce a model that offered high performance with low latency and overhead. Other factors that influenced this choice include data-alignment.

Feature extraction is used to enhance the discriminative power of the classifier. Intuitively, not all feature measurements will be equally useful. In fact, inclusion of overlapping or superfluous features (identical measurements from both legitimate and illegitimate use) might make masqueraders appear more like a normal user than they would otherwise. Part of our study is to discover which classes of measurements are more useful in discriminating between different behavior patterns.

We approached this by using a discriminant analysis technique known as "Fisher's criteria" to score each scalar feature in the feature vector. Our implementation for Fisher's criteria, in this case the one-dimensional variation, evaluates the value of each feature independently via the ratio of the inter-class and intra-class variance. This step is defined as follows: let $m_1, m_2, ..., m_k$ represent the arithmetic means of a measured feature for k users, and define $m_G = 1/k \sum_{i=1}^{k} m_k$ to represent the global (grand) mean. Fisher's "score" is defined as:

$$s = \frac{\sigma_b}{\sum_{i=1}^{k} \sigma_i},$$

where σ_b is the between-class variance

$$\sigma_b = \sqrt{\sum_{i=1}^{k} (m_i - m_G)^2},$$

and σ_i is the within-class variance for a particular user. The one-dimensional variation is appropriate because the underlying measurements were designed to be mostly independent. For example, file-access is not directly related to window movements. Further, we note that we evaluated linear discriminant analysis (using a projection unto a lower-dimensional feature sub-space) and found that it yielded no performance benefits [1]. The above equation is maximized when a feature has both low within-class variance (very stable and predictable) and high between-class variance (appears very different for every class.)

Our 20 features, ranked by their Fisher scores, are listed in Table 2. These results were generated by measuring the data collected from the "normal users" group in our study, described in Section 4. Note that these results were tuned to discriminate amongst instances of normal user behavior – they are extracted using the samples collected by the normal user pool. No malicious samples were incorporated in this evaluation process, these were not used in order to avoid introducing bias towards the detection of any specific type of intrusion. We tuned our feature set to identify differences in instances of normal behavior (a 160-class problem), and it is our hypothesis that the same set of features would perform well for the easier problem of discriminating between legitimate and malicious behavior (a 2-class problem.)

We justify these assumptions with experiment results shown in the subsequent section. While we did not use malicious samples when tuning/training our models, we used these when we evaluated the masquerade detection accuracy of the system. These separate experiments were carried out with data collected via the second "masquerader" study, where users were asked to act as insider threats.

In our model, we categorize PDFs, Microsoft Word files, Excel files, RTF files, text files, and similar document-type files as the "document" class. The label "decoy" is self-describing, and we use the more general "file" label to represent any other file type. Since our features are based on access counts, the volume of activity across these different features may vary by wide ranges, and this may introduce model bias towards some features with high volume. Therefore, we use the standard approach of log-squashing this value to smooth the measurements; this removes bias and prevent any one feature from dominating the others. For each feature x, we generate the

[1]direct comparison with FLD variation of Fisher's method omitted due to space limitation

	Fisher Score	**Feature type**
1	0.027054	Number of processes deleted
2	0.015762	Number of processes created
3	0.014442	Number of file changes
4	0.008218	Number of window touches
5	0.005888	Number of ports opened
6	0.004987	Number of ports closed
7	0.001290	Number of decoys renamed
8	0.000660	Number of documents touched
9	0.000231	Number of system actions
10	0.000156	Number of files renamed
11	0.000000	Number of ACL related actions
12	0.000000	Number of files touched
13	0.000000	Number of decoys created
14	0.000000	Number of decoys deleted
15	0.000000	Number of decoys touched
16	0.000000	Number of documents created
17	0.000000	Number of documents deleted
18	0.000000	Number of documents renamed
19	0.000000	Number of files created
20	0.000000	Number of files deleted

Table 2: Fisher discriminant analysis results

normalized value x^*:

$$x^* = \log(x+1).$$

Examining the results, the scores for decoy-related features 13, 14, and 15 at first seem to show that decoys are not useful in discriminating behavior. However, this is not correct – the numbers actually show something more subtle.

First, recall that decoys are explicitly modeled in the ML component. Rather, decoys act as independent tripwires in an orthogonal strategy that plays a separate role in our sensor architecture.

Referring to the results seen in Table (2), we see that decoy-related actions yielded low discriminative scores. What this shows is that the users in our study exhibited *consistent behavior* with respect to decoy-access patterns; so much so that their behavior is statistically indiscernible from one another. In fact, all users in the normal control group acted on deployed decoys in the same way: there was some initial burst of activity when decoys were first created – possibly because users were curious as to what would happen if they opened them – then, as the users got used to the decoys' presence, touch rates dropped to zero and stayed that way throughout the course of the study. What the results in Table (2) show, therefore, is that decoy use *does not interfere* with normal system activity, and that this is consistent across all users. Later sections show more specific examples of how decoy touch rates drop.

5.2. Modeling

Since we did not want to pre-define a prototypical masquerader class, a generative-based machine learning model was more appropriate. Thus, we derive a model for normal behavior and alert on outliers as masqueraders (*i.e* anomaly detection.) We interpret the individual features within a time slice as independently and identically distributed (i.i.d.) multivariate feature vectors – assuming that each instance comes from some underlying distribution and each (x-minute) time-slice is statistically independent from the rest. In practice, some statistical dependence exists – for example, repeatedly opening a URL contained within a document – however, we chose to drop this assumption to simplify the model and avoid overfitting. We use a Gaussian Mixture Model (GMM) to fit this distribution; GMM is a well-known generative model that is widely used as a standard benchmark for learning algorithms. As our work in this area is relatively new, and there are few direct comparisons available; the use of GMMs establishes a reliable baseline. Anecdotally, if we had explicitly defined an attack pattern (attacker class) then more powerful discriminative-based methods such as support vector machines would be available. To maximize classification performance, we use only features with Fisher scores greater than zero (refer to Table 2.)

Let x represent the aforementioned multivariate feature vector (in column form), let μ represent the sample mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

and Σ represent the sample covariance matrix:

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)(x_i - \mu)^\top,$$

for training samples $x_1, x_2, ..., x_N$, the prototypical GMM definition utilizing full-ranked covariance matrices is defined as:

$$\begin{aligned} p(x|\theta) &= \frac{1}{(2\pi)^{d/2}\sqrt{|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)^\top\right) \\ q(x|\Theta) &= \sum_{i=1}^{k} \alpha_i p(x|\boldsymbol{\theta}_i) \end{aligned}$$

where $\theta_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$ and α_i is the mixing coefficient for the GMM. $q(\cdot)$ is the final GMM likelihood function that is used to evaluate a sample, and k is the number of Gaussians in the mixture. In practice, k is analogous to the k in the well known k-means algorithm, and is selected via experimental tuning (in fact, GMM is a more generalized version of k-means.) These parameters are optimized ("learned") via the *expectation maximization* (EM) algorithm. EM is standard parameter estimation technique. At a high level, it works by calculating the sum likelihood of the training data and finding the set of parameters that optimizes this function:

$$\arg\max_{\Theta}\left(q(X|\Theta) = \sum_{i=1}^{N} q(x_i|\Theta)\right)$$

Parameter estimates for Θ are updated by taking a step towards the gradient of this likelihood function. Essentially, the process is a gradient descent estimation algorithm. Since the mathematical procedure for deriving GMM-style parameter updates using EM are known, they were omitted for brevity. By using the greedy technique proposed by Verbeek et. al [31], a GMM can be efficiently trained in $O(k^2n+kmn)$ time, where n is the number of training samples, k is the mixture component count, and m is the number of variants to be checked for each component.

6. Results

First, we evaluate the model accuracy by measuring the discriminative power of the classifier within the normal users in the RUU dataset, by comparing their logs against each other's. This measures instances of normal behavior vs. other normal behavior, and represents the "harder" detection problem because it consists of 160 classes of normal behavior. Next we show the results of a user study where volunteers were recruited to act as masqueraders, where their actions were recorded as a second "masquerader dataset." The masquerader detection is an easier challenge because we detect anomalies on a per-user basis and therefore it is a two-class problem. We report the performance of our classifier on this dataset. Finally, we examine at the effects of decoy deployment in real world-tested environments by looking at false trigger rates in our user study to show that decoys use do not interfere with behavior modeling.

6.1. Behavior classification

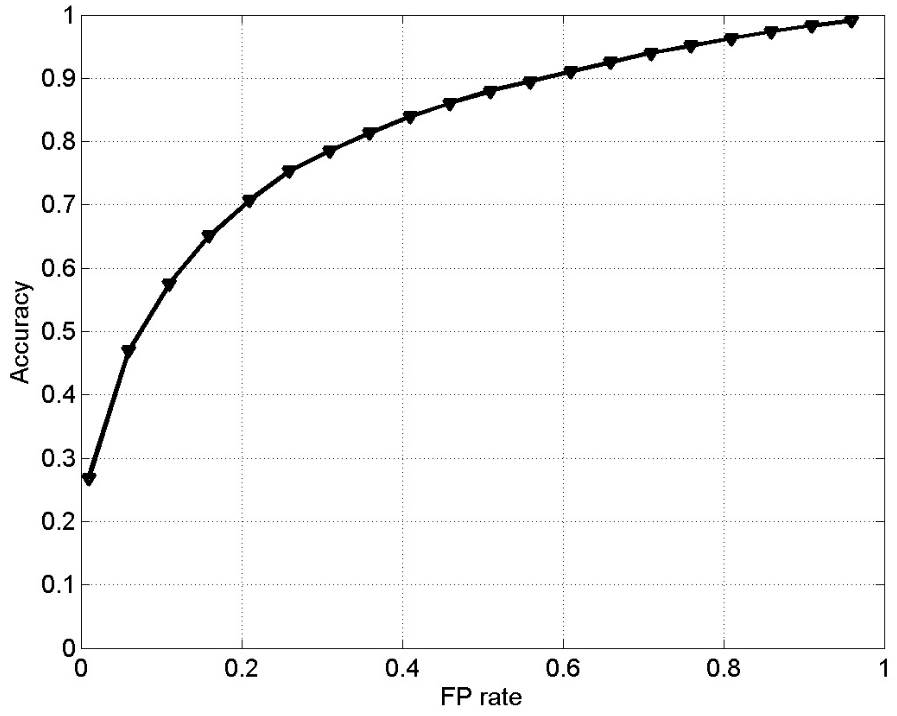

Figure 3: Multi-class classification. Average ROC for Gaussian Mixture Model with Fisher-features. 160 users from the RUU dataset.

Figure 3 shows the average Receiver-Operator-Characteristic (ROC) curve for the 160 users. This experiments evaluates our model's ability to discriminate between *different instances of legitimate activity* across 160 different users. This classification task is more challenging than detecting deviations from one particular user's behavior pattern. We use this challenge to tune our sensors and our models. Performance results against masquerader activity, which is the ultimate goal of the sensor architecture, are reported in the following subsection are shown to be much higher.

This ROC result shows that at 10% FP rate, the classifier can discriminate legitimate user actions from among 160 different users with roughly 57% accuracy. Evaluation was performed using one-vs-all, max-likelihood classification, where each test sample was evaluated against every user model and labeled with the user whose model yielded the highest likelihood score. The *random-guess* baseline for this experiment would yield $1/160 = 0.63\%$ accuracy. All report numbers in this paper are averages of five independent trials.

Figure 4 shows the Area-Under-The-Curve measurement, which is the integral of the ROC. This measurement gives us an aggregate score for each user and allows us to chart the classification accuracy across users, and shows the distribution of accuracies across all users.

6.2. Masquerader Detection

Classifying masqueraders is a two-class problem, making it "easier" than the 160-class problem that our sensor as tuned on. The results in this section provide evidence to support this. Our generative-based approach allowed us to formulate our models without training against any specific attack patterns to avoid the pitfall of potentially over-fitting to one specific masquerader profile.

Figure 5 shows the ROC results for the masquerader dataset. For detection evaluation, we train a model for each legitimate RUU user using that user's own traffic, and then measure the detection accuracy of the GMM model against all of the sensor logs generated by the masqueraders in this study. Figure 5 shows the average ROC across all RUU users in detecting these 22 masqueraders. As demonstrated by Axelsson in [32], a intrusion detection method is restricted more by its false positive rate than by its true detection accuracy. We therefore select a false positive threshold value and use this to determine our system's effective detection rate. From our ROC curve, we can see that our classifier achieves 68% detection at 1% FP rate, and 73% masquerader behavior detection accuracy at a 2% FP.

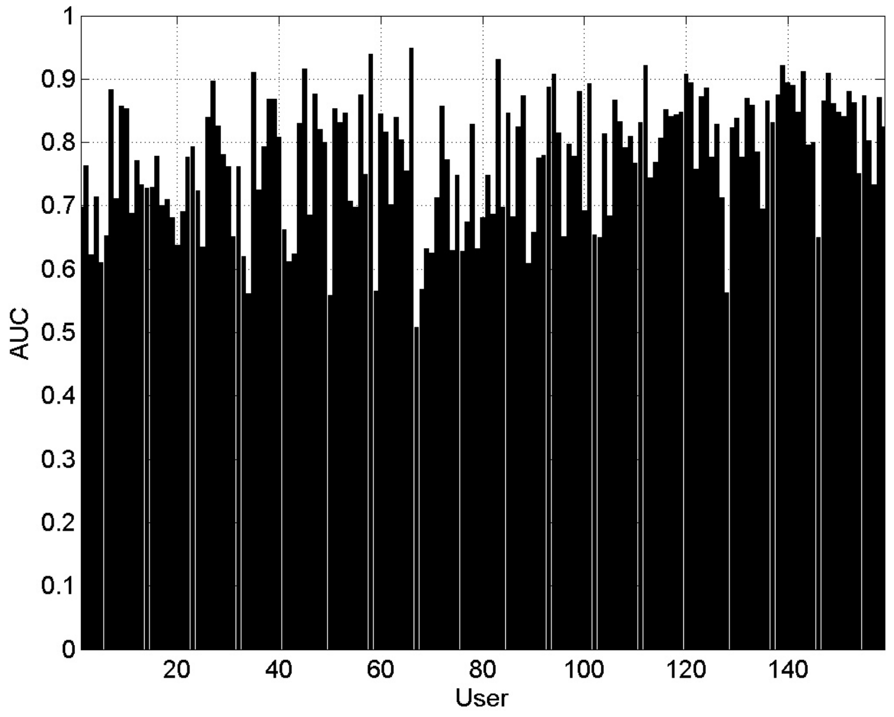

Figure 4: Multi-class *normal-behavior* classification ROC: differentiating between samples of normal behavior among 160 users. Area under the curve (AUC) results for Gaussian Mixture Model with Fisher-features. Average AUC = 0.7722.

Figure 6 shows the AUC chart for the masquerader set. As the results show, the performance is much higher on the real-world-measured intrusion dataset, even though the attacker class was not explicitly modeled in our method and no malicious behavior data was used in the training process. Classification accuracy is entirely dependent on the model's ability to recognize outliers from normal behavior.

While the results reflect aggregate detection rates across all instances of masquerade data, when the sensor is deployed in practice, however, the accuracy rate is determined by the continuous re-evaluation procedure. Continuous active-authentication means that the sensor would issue an identity challenge as soon as the *first* alert is detected. Therefore only one alert is needed before a successful interdiction, if we assume that the attacker cannot correctly answer the challenge.

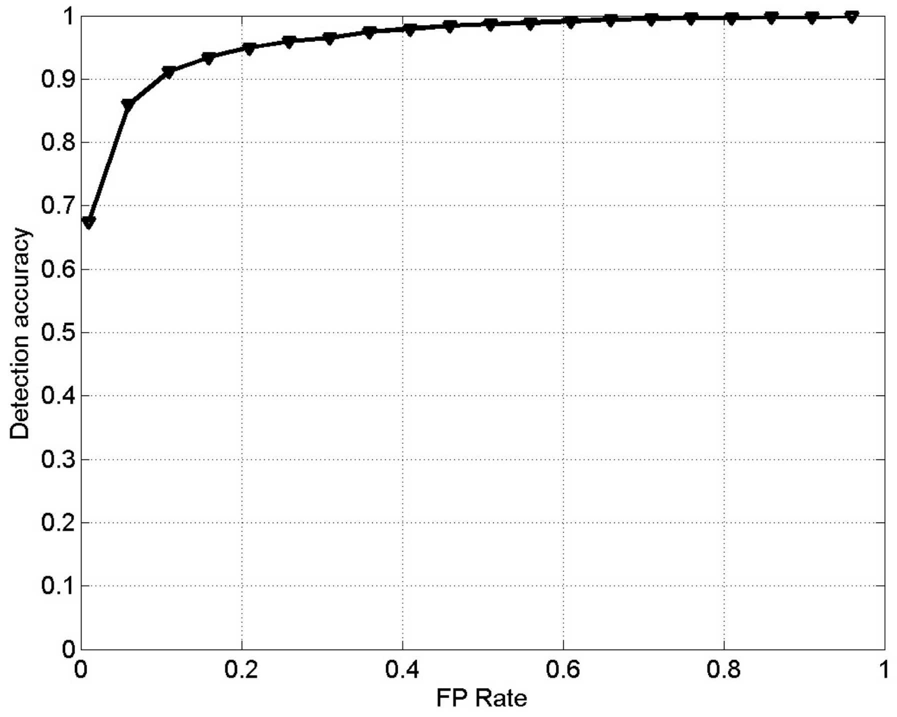

Figure 5: Masquerader detection ROC: discriminating between normal vs. masquerader data.

For example, setting the FP rate to be 2%, and let y represent the chance that the sensor fails to detect a legitimate user, $y = 1 - 0.73 = 0.27$, the number of behavior time slices evaluated before there is a less than 5% chance of continued evasion is:

$$\begin{aligned} 0.27^x &< 0.05 \\ x &< \log(0.05)/\log(y = 0.27) \\ x &\approx 2.29 \end{aligned}$$

thus we need 3 samples (roughly 15 minutes of activity) before the sensor can be expected to detect a masquerader with greater than $1 - 0.05 = 95\%$ probability. Ultimately, the factors that influence this performance curve is multifaceted and their relations non-linear. Performance depends on model

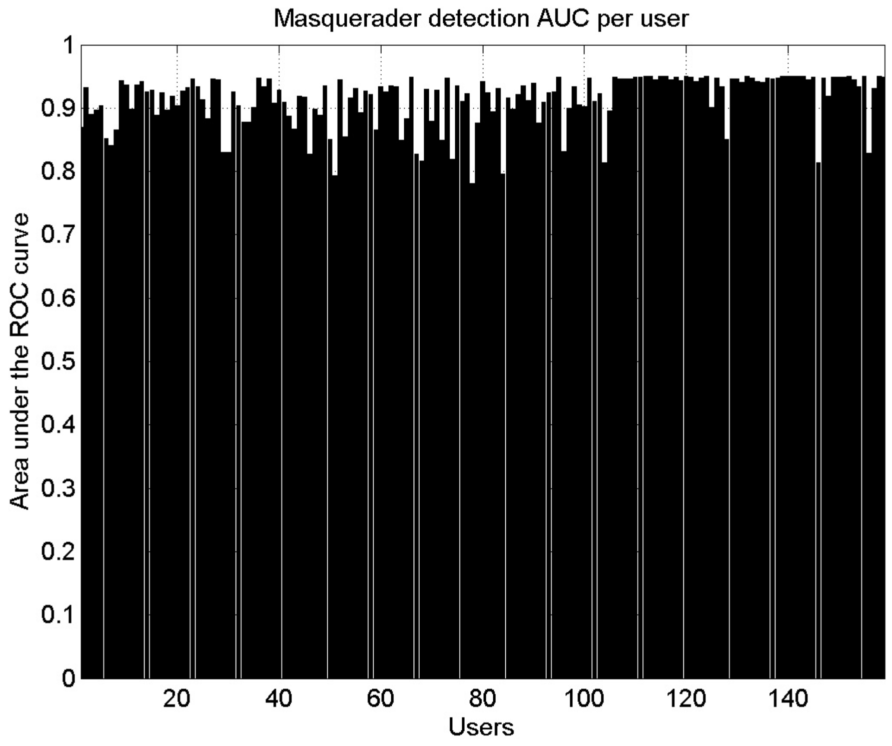

Figure 6: Masquerader detection AUC per user.

evaluation period as well as the user-configured desired FP rate. We tabulate some potential settings that we feel are realistic and extrapolate the expected sensor performance. These results are shown in Table (3).

For the technique to be considered practical in real-world environments, it must operate at extremely low FP rates. For a more stringent setting, figure 7 shows the accuracy range for FP rates within the 0-1% range. These results were obtained in the same manner as those shown in figure 5 – only the targeted FP range has changed. To provide some perspective, if our sensor were to be installed in a business workstation and used continuously throughout a 40-hour work week, and tuned to yield only one false-positive during this time, table 3 shows the expected time-to-detection (TTD) performance given evaluation frequency.

Consider one row in this table: with an evaluation frequency of once every 3 minutes, a 40-hour work week would trigger 800 evaluations. In order to

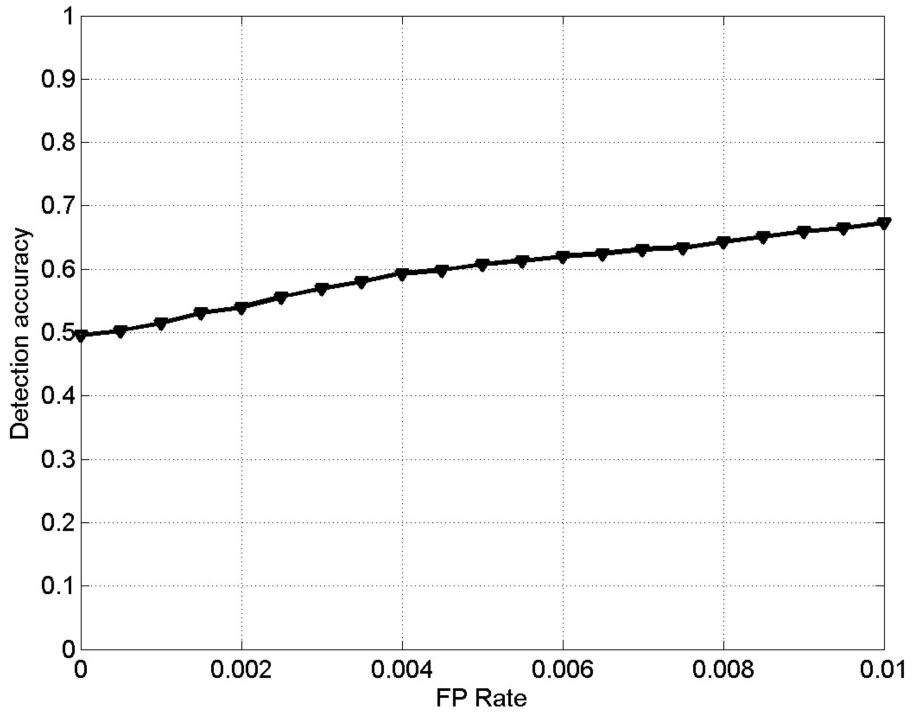

Figure 7: Masqueraders detection accuracy within the very restrictive 0-1% FP range.

Frequency	Total Samples	FP Req.	Acc.	Evals	TTD
1m	2400	0.042%	49.55%	5	5m
2m	1200	0.083%	50.29%	5	10m
3m	800	0.125%	51.46%	5	15m
4m	600	0.167%	53.11%	4	16m
5m	480	0.208%	54.00%	4	20m

Table 3: Time until detection (TTD) given evaluation frequency for a 40-hour work week.

maintain the at most one false positive per week, the sensor must be tuned so that it performs with a maximum 0.125% FP rate. At this FP rate, the sensor has a base accuracy of 51.46% according to the masquerader ROC (false negative is the complement 48.54%), which means five evaluations are necessary before there is a less than 5% chance that a masquerader remains undetected. This yields a TTD estimate of 15 minutes.

6.3. Non-Interference of Decoys

Since our technology deploys decoys to the user's file system, it follows that we should consider the potential for false positives, where a legitimate user triggers a defensive reaction by accessing these files. Our study includes an analysis of how the presence of decoy documents interferes with normal operations. We found that users quickly learned to avoid their own decoys after the initial deployment period.

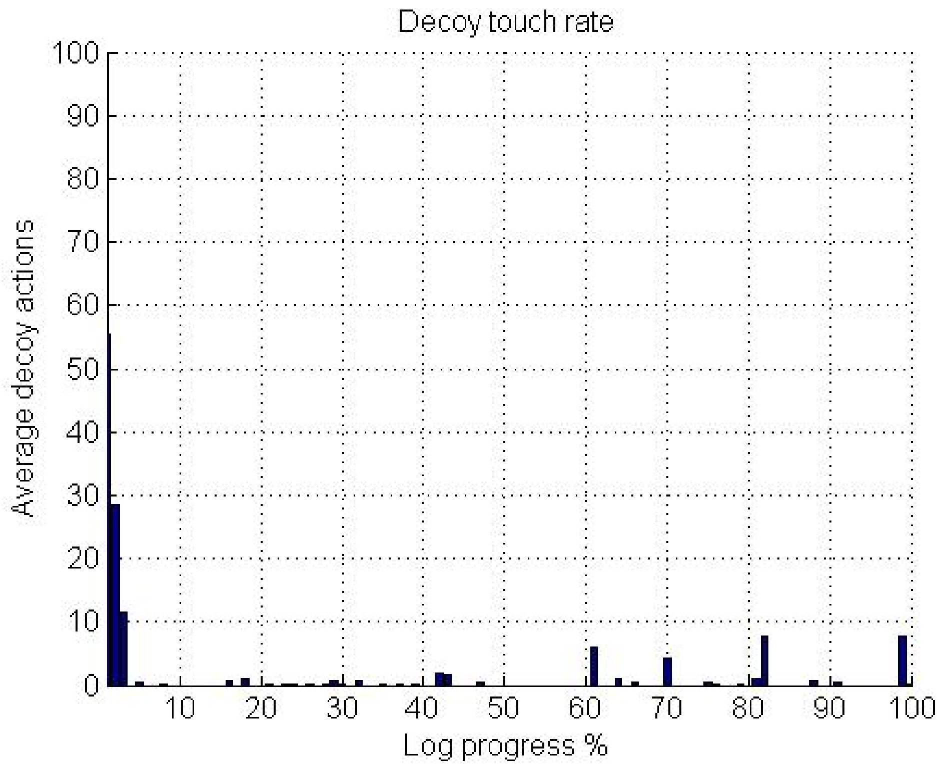

Figure 8: Average decoy touch rate across all 160 RUU users, aligned by progression % (beginning to end).

Figure 8 shows decoy access behavior averaged across 160 users in the RUU dataset. The x-axis represents the study's progression percentage: 0% indicates the beginning of the study and 100% indicates the conclusion of the study. There is a large spike in decoy access immediately after the experiment begins. This is expected behavior because the user is instructed to deploy decoys to their system, thus the sensor picks up some initial decoy-related actions. It also reflects initial user curiosity regarding the presence of these files on their system; users will attempt to open decoy documents to see what would happen. After this short bootstrapping phase, the figures show that decoys are infrequently accessed. These results show that decoys *do not interfere with normal activity.* Legitimate system users are not confused by decoys due to their ability to discern their spurious content from their own authentic data.

6.4. Influence of Knowledge Regarding Decoy Presence

As discussed in Section 4.2, an auxiliary goal of our detection validation study was to assess the impact of decoy knowledge on decoy efficacy. That is, we wished to determine whether users' awareness of the presence of decoy documents on a system made them better or worse at avoiding the detection of their masquerader activity. To this end we designed two variants of the written study scenario which was supplied to our participants. Test subjects in one group received a scenario which specifically identified decoy documents as a security mechanism. The alternative scenario, on the other hand, included a more generic statement regarding system surveillance. 10 participants were recruited to test each variant of these scenarios.

The difference between decoy access rates for the informed and uninformed pool of users is shown in Figure 9. As this figure illustrates, the amount of detail provided to our simulated masqueraders did not have a statistically significant effect on participants' rate of decoy access. The average and minimum amounts of decoy access events remained the same for users when decoys were mixed in with legitimate file locations.

Intuitively, our initial hypothesis was that informing masqueraders of the presence of decoy content on a system would cause them to hesitate to interact with the system when compared to users who do not have any knowledge that decoys are being employed as a security mechanism. The results of our detection validation do not offer evidence in support of this claim though. We were unable to identify any statistical difference between the group of users who were notified regarding decoy content and those who were left uninformed.

A surprising finding of our study was the result that under some circumstances informing masqueraders of the presence of decoys seems to have in fact increased rates of decoy activity. Though more research is needed to fully ascertain the underlying rationale behind this observation, we currently suspect that this had to do with the fact that although users were aware of

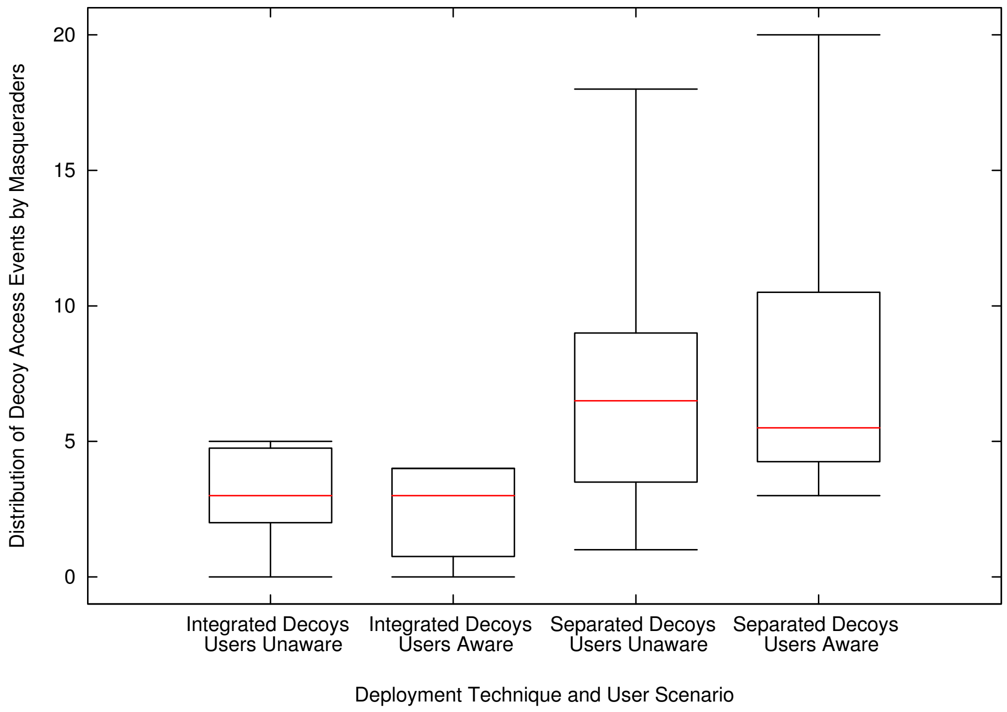

Figure 9: Comparison of Decoy Document Access Event Distributions for Different User Scenarios

the fact that decoys were deployed on a system, they had no prior experience working with a system in the presence of decoy material.

7. Conclusion

We introduced novel methods for real-time continuous active-authentication based on automatically learning and validating user-system interaction (behavior) patterns. Beyond behavior modeling, the sensor also deploys and monitors decoy trip-wires that may be programmmatically generated and distributed. This technology augments existing data protection measures by adding a new transparent layer of active authentication. We have experimentally demonstrated that our methods can achieve good performance under realistic ($< 0.01\%$) FP requirements. We have further shown that our generated decoys do not interfere with normal user activity. The results of large-scale user studies demonstrate that this approach is potentially very effective in protecting personal data, even in cases where attackers posses their target victims' credentials. The machine-learning-based classifier achieves masquerader detection accuracy of 95% with an expected time-to-detection (TTD) of 15 minutes when using a 3 minute reevaluation interval, while obeying the constraint that only one false positive is generated per 40-hours of activity.

Acknowledgment

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions, and/or findings contained in this article/presentation are those of the author(s) /presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

[1] W. Baker, A. Hutton, C. D. Hylender, J. Pamula, C. Porter, M. Spitler, 2013 Data Breach Investigations Report: A study conducted by the Verizon Business RISK team in cooperation with the U.S. Secret Service and the Dutch High Tech Crime Unit (2013).
URL `http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2011_en_xg.pdf`

[2] M. Bishop, D. V. Klein, Improving system security via proactive password checking, Computers & Security 14 (3) (1995) 233–249.

[3] D. Florencio, C. Herley, A large-scale study of web password habits, in: Proceedings of the 16th international conference on World Wide Web, WWW'07, ACM, New York, NY, USA, 2007, pp. 657–666. doi:10.1145/1242572.1242661.
URL `http://doi.acm.org/10.1145/1242572.1242661`

[4] M. Dell'Amico, P. Michiardi, Y. Roudier, Password strength: an empirical analysis, in: Proceedings of the 29th conference on Information communications, INFOCOM'10, IEEE Press, Piscataway, NJ, USA, 2010, pp. 983–991.
URL `http://dl.acm.org/citation.cfm?id=1833515.1833671`

[5] Y. Song, M. B. Salem, S. Hershkop, S. J. Stolfo, System level user behavior biometrics using fisher features and gaussian mixture models, 2013 IEEE Security and Privacy Workshops 0 (2013) 52–59. doi:http://doi.ieeecomputersociety.org/10.1109/SPW.2013.33.

[6] M. Ben-Salem, S. J. Stolfo, Modeling user search-behavior for masquerade detection, in: Proceedings of the 14th International Symposium on Recent Advances in Intrusion Detection, Springer, Heidelberg, 2011, pp. 181–200.

[7] B. M. Bowen, Design and analysis of decoy systems for computer security, Ph.D. thesis, Columbia University (2011).

[8] J. Voris and N. Boggs and S. Stolfo, Lost in Translation: Improving Decoy Documents via Automated Translation, in: Workshop on Research for Insider Threat, 2012.

[9] M. Ben Salem and S. Stolfo, Decoy Document Deployment for Effective Masquerade Attack Detection, in: Conference on Detection of Intrusions and Malware and Vulnerability Assessment, 2011.

[10] J. Voris, J. Jermyn, N. Boggs, S. Stolfo, Fox in the trap: Thwarting masqueraders via automated decoy document deployment, in: Proceedings of the Eighth European Workshop on System Security, EuroSec '15, ACM, New York, NY, USA, 2015, pp. 3:1–3:7. doi:10.1145/2751323.2751326.
URL `http://doi.acm.org/10.1145/2751323.2751326`

[11] B. Bowen and S. Hershkop and A. Keromytis and S. Stolfo, Baiting Inside Attackers Using Decoy Documents, in: Conference on Security and Privacy in Communication Networks, 2009.

[12] Y. Song, M. Ben Salem, S. Hershkop, S. J. Stolfo, System level user behavior biometrics using fisher features and gaussian mixture models, in: Security and Privacy Workshops (SPW), 2013 IEEE, IEEE, 2013, pp. 52–59.

[13] F. Monrose, M. K. Reiter, S. Wetzel, Password hardening based on keystroke dynamics, in: Proceedings of the 6th ACM conference on Computer and communications security, CCS '99, ACM, New York, NY, USA, 1999, pp. 73–82. doi:10.1145/319709.319720.
URL `http://doi.acm.org/10.1145/319709.319720`

[14] H. Gamboa, A. Fred, A behavioral biometric system based on human-computer interaction, SPIE 5404 - Biometric Technology for Human Identification (2004) 381–392.

[15] A. A. E. Ahmed, I. Traore, A new biometric technology based on mouse dynamics, IEEE Trans. Dependable Secur. Comput. 4 (3) (2007) 165–179. doi:10.1109/TDSC.2007.70207.
URL `http://dx.doi.org/10.1109/TDSC.2007.70207`

[16] C. Shen, Z. Cai, X. Guan, Y. Du, R. Maxion, User authentication through mouse dynamics, Information Forensics and Security, IEEE Transactions on 8 (1) (2013) 16 –30. doi:10.1109/TIFS.2012.2223677.

[17] A. Al-Khazzar, N. Savage, Graphical authentication based on user behaviour, in: Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on, 2010, pp. 1 –4.

[18] T. Goldring, Authenticating users by profiling behavior. http://www.galaxy.gmu.edu/interface/i03/i2003html/goldringtom/goldringtom.presentation.ppt, in: Proceedings of the ICDM Workshop on Data Mining for Computer Security, DMSEC '03, 2003.
URL `http://www.galaxy.gmu.edu/interface/I03/I2003HTML/GoldringTom/GoldringTom.presentation.ppt`

[19] M. Schonlau, W. Dumouchel, W.-H. Ju, A. F. Karr, M. Theus, Y. Vardi, Computer intrusion: Detecting masquerades, Statistical Science 16 (2001) 58–74.

[20] R. A. Maxion, T. N. Townsend, Masquerade detection augmented with error analysis, IEEE Transactions on Reliability 53 (1) (2004) 124–147.

[21] M. Oka, Y. Oyama, H. Abe, K. Kato, Anomaly detection using layered networks based on eigen co-occurrence matrix, in: Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection, RAID '04, 2004, pp. 223–237.

[22] K. Niinuma, U. Park, A. Jain, Soft biometric traits for continuous user authentication, Information Forensics and Security, IEEE Transactions on 5 (4) (2010) 771–780. doi:10.1109/TIFS.2010.2075927.

[23] C. Shen, Z. Cai, X. Guan, Continuous authentication for mouse dynamics: A pattern-growth approach, in: Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on, 2012, pp. 1–12. doi:10.1109/DSN.2012.6263955.

[24] M. Pusara, C. E. Brodley, User re-authentication via mouse movements, in: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, VizSEC/DMSEC '04, ACM, New York, NY, USA, 2004, pp. 1–8. doi:10.1145/1029208.1029210.
URL `http://doi.acm.org/10.1145/1029208.1029210`

[25] D. G. Tey Chee Meng, Payas Gupta, I can be You: Questioning the use of Keystroke Dynamics as Biometrics, in: Proceedings of the 20th Annual Network & Distributed System Security Symposium, NDSS'13, The Internet Society, 2013.

[26] D. Hollingsworth, Enhancing Computer System Security, in: The Rand Paper Series, 1973.

[27] L. Spitzner, Honeytokens: The Other Honeypot (2003).
URL `http://www.symantec.com/connect/articles/honeytokens-other-honeypot`

[28] J. Yuill and M. Zappe and D. Denning and F. Feer, Honeyfiles: Deceptive Files for Intrusion Detection, in: Workshop on Information Assurance, 2004.

[29] Columbia University Intrusion Detection Systems Lab, FOG Computing, Available at `http://ids.cs.columbia.edu/FOG/` (2012).

[30] The Columbia University IDS Lab, RUU2 Project, Available at `http://ids.cs.columbia.edu/content/ruu.html` (2013).

[31] J. J. Verbeek, N. Vlassis, B. Kröse, Efficient greedy learning of gaussian mixture models, Neural computation 15 (2) (2003) 469–485.

[32] S. Axelsson, The Base-Rate Fallacy and its Implications for the Difficulty of Intrusion Detection, in: ACM Transactions on Information and System Security, 2000.