Modeling User Search Behavior for Masquerade Detection

Malek Ben Salem and Salvatore J. Stolfo

Computer Science Department
Columbia University
New York, USA
{malek,sal}@cs.columbia.edu

Abstract. Masquerade attacks are a common security problem that is a consequence of identity theft. This paper extends prior work by modeling user search behavior to detect deviations indicating a masquerade attack. We hypothesize that each individual user knows their own file system well enough to search in a limited, targeted and unique fashion in order to find information germane to their current task. Masqueraders, on the other hand, will likely not know the file system and layout of another user's desktop, and would likely search more extensively and broadly in a manner that is different than the victim user being impersonated. We identify actions linked to search and information access activities, and use them to build user models. The experimental results show that modeling search behavior reliably detects all masqueraders with a very low false positive rate of 1.1%, far better than prior published results. The limited set of features used for search behavior modeling also results in large performance gains over the same modeling techniques that use larger sets of features. *

Keywords: masquerade detection, user profiling, search behavior, svm.

1 Introduction

The *masquerade attack* is a class of attacks, in which a user of a system illegitimately poses as, or assumes the identity of another legitimate user. Identity theft in financial transaction systems is perhaps the best known example of this type of attack. Masquerade attacks are extremely serious, especially in the case of an insider who can cause considerable damage to an organization. Their detection remains one of the more important research areas requiring new insights to mitigate against this threat.

A common approach to counter this type of attack, which has been the subject of prior research, is to apply machine learning (ML) algorithms that produce classifiers which can identify suspicious behaviors that may indicate

* Support for this work has been partially provided by a DARPA grant ADAMS No. W911NF-11-1-0140.

malfeasance of an impostor. We do not focus on whether an access by some user is authorized since we assume that the masquerader does not attempt to escalate the privileges of the stolen identity, rather the masquerader simply accesses whatever the victim can access. However, we conjecture that the masquerader is unlikely to know the victim's search behavior when using their own system which complicates their task to mimic the user. It is this key assumption that we rely upon in order to detect a masquerader. The conjecture is backed up with real user studies. Eighteen users were monitored for four days on average to produce more than 10 GBytes of data that we analyzed and modeled. The results show that indeed normal users display different search behavior, and that that behavior is an effective tool to detect masqueraders. After all, a user will search within an environment they have created. For example, a user searches for a file within a specific directory, or a programmer searches for a symbol within a specific source code file. We assume the attacker has little to no knowledge of that environment and that lack of knowledge will be revealed by the masquerader's abnormal search behavior. Thus, our focus in this paper is on monitoring a user's behavior in real time to determine whether current user actions are consistent with the user's historical behavior, primarily focused on their unique search behavior. The far more challenging problems of thwarting mimicry attacks and other obfuscation techniques are beyond the scope of this paper.

Masquerade attacks can occur in several different ways. In general terms, a masquerader may get access to a legitimate user's account either by stealing a victim's credentials, or through a break in and installation of a rootkit or key logger. In either case, the user's identity is illegitimately acquired. Another perhaps more common case is laziness and misplaced trust by a user, such as the case when a user leaves his or her terminal or client open and logged in allowing any nearby coworker to pose as a masquerader.

In this paper we extend prior work on modeling user command sequences for masquerade detection. Previous work has focused on auditing and modeling sequences of user commands including work on enriching command sequences with information about arguments of commands [15, 10, 18]. We propose an approach to profile a user's search behavior by auditing search-related applications and accesses to index files, such as the index file of the Google Desktop Search application. We conjecture that a masquerader is unlikely to have the depth of knowledge of the victim's machine (files, locations of important directories, available applications, etc.), and hence, a masquerader would likely first engage in information gathering and search activities before initiating specific actions. To this extent, we conduct a set of experiments using a home-gathered Windows data. We model search behavior in Windows and test our modeling approach using our own data, which we claim is more suitable for evaluating masquerade attack detection methods.

The contributions of this work are:

- A **small set of search-related features** used for effective masquerade attack detection: The limited number of features reduces the amount of sampling required to collect training data. Reducing the high-dimensional modeling space to a low-dimensional one allows for the improvement of

both accuracy and performance. We shall use standard machine learning techniques to evaluate the system composed of these features. Other work has evaluated alternative algorithms. Our focus in this work is on the features that are modeled. The best masquerade attack detection accuracy was achieved using a modern ML algorithm, Support Vector Machines (SVMs). SVM models are easy to update, providing an efficient deployable host monitoring system. We shall use one-class SVM (ocSVM) models in this work.

– A **publicly available Windows data set** [1] collected specifically **to study the masquerade attack detection problem** as opposed to the author identification problem: The data set consists of normal user data collected from a homogeneous user group of 18 individuals as well as simulated masquerader data from 40 different individuals. The data set is the first publicly available data set for masquerade attack detection since the Schonlau dataset [14].

In Section 2 of this paper, we briefly present the results of prior research work on masquerade detection. Section 3 expands on the objective and the approach taken in this work. In Section 4, we present our home-gathered dataset which we call the RUU dataset. Section 5 shows how the malicious intent of a masquerader, whose objective is to steal information, has a significant effect on their search behavior. In section 6, we discuss experiments conducted by modeling search behavior using the RUU dataset. In Section 7, we discuss potential limitations of our approach and how they could be overcome. Finally Section 8 concludes the paper by summarizing our results and contributions, and presenting directions for our future work.

2 Related Work

In the general case of computer user profiling, the entire audit source can include information from a variety of sources, such as user commands, system calls, database/file accesses, and the organization policy management rules and compliance logs. The type of analysis used is primarily the modeling of statistical features, such as the frequency of events, the duration of events, the co-occurrence of multiple events, and the sequence or transition of events. However, most of this work failed to **reveal** or clarify **the user's intent** when issuing commands or running processes. The focus is primarily on accurately detecting change or unusual command sequences. In this section, we review approaches reported in the literature that profile users by the commands they issue.

Schonlau et al. [15] applied six masquerade detection methods to a data set of 'truncated' UNIX commands for 70 users collected over a several month period. Truncated commands are simple commands with no arguments. Each user had 15,000 commands collected over a period of time ranging between a few days and several months [14]. Fifty users were randomly chosen to serve as intrusion targets. The other 20 users were used as masqueraders. The first 5000 commands for each of the 50 users were left intact or 'clean', while the next 10,000 commands were randomly injected with 100-command blocks issued by

the 20 masquerade users. The commands have been inserted at the beginning of a block, so that if a block is contaminated, all of its 100 commands are inserted from another user's list of executed commands. The objective was to accurately detect the 'dirty' blocks and classify them as masquerader blocks. It is important to note that this dataset does not constitute ground truth masquerade data, but rather simulates impersonation.

The first detection method applied by Schonlau et al. for this task, called 'uniqueness', relies on the fact that half of the commands in the training data are unique and many more are unpopular amongst the users. Another method investigated was the Bayes one-step Markov approach. It is based on one step transitions from one command to the next. The approach, due to DuMouchel (1999), uses a Bayes factor statistic to test the null hypothesis that the observed one-step command transition probabilities are consistent with the historical transition matrix.

A hybrid multi-step Markov method has also been applied to this dataset. When the test data contain many commands unobserved in the training data, a Markov model is not usable. Here, a simple independence model with probabilities estimated from a contingency table of users versus commands may be more appropriate. The method used automatically toggles between a Markov model and an independence model generated from a multinomial random distribution as needed, depending on whether the test data are 'usual', i.e. the commands have been previously seen, or 'unusual', i.e. Never-Before-Seen Commands (NBSCs).

IPAM (Incremental Probabilistic Action Modeling), another method applied on the same dataset, and used by Davidson and Hirsch to build an adaptive command line interface, is also based on one-step command transition probabilities estimated from the training data [6]. A compression method has also been tested based on the premise that test data appended to historical training data compress more readily when the test data stems indeed from the same user rather than from a masquerader. A sequence-match approach has been presented by Lane and Brodley [8]. For each new command, a similarity measure between the 10 most recent commands and a user's profile is computed.

A different approach, inspired by the Smith-Waterman local alignment algorithm, and known as semi-global alignment, was presented by Coull et al. [4]. The authors enhanced it and presented a sequence alignment method using a binary scoring and a signature updating scheme to cope with concept drift [5]. Oka et al. [12] noticed that the dynamic behavior of a user appearing in a sequence can be captured by correlating not only connected events, but also events that are not adjacent to each other while appearing within a certain distance (non-connected events). To that extent, they have developed the layered networks approach based on the Eigen Co-occurrence Matrix (ECM).

Maxion and Townsend [10] applied a naïve Bayes classifier and provided a detailed investigation of classification errors [11] highlighting why some masquerade victims are more vulnerable or more successful than others. Wang and Stolfo compared the performance of a naïve Bayes classifier and a SVM classifier

to detect masqueraders [18]. Their experiments confirmed, that for masquerade detection, one-class training is as effective as two class training.

These specific algorithms and the results achieved for the Schonlau dataset are summarized in Table 1 (with True Positive rates displayed rather than True Negatives). Performance is shown to range from 1.3% - 7.7% False Positive rates, with a False Negative rate ranging from 24.2% to 73.2% (alternatively, True Positive rates from 26.8% to 75.8%). Clearly, these results are far from ideal.

Table 1. Summary of Accuracy Performance of Anomaly Detectors Using the Schonlau Data Set

Method	**True Pos. (%)**	**False Pos. (%)**
Uniqueness [15]	39.4	1.4
Bayes one-step Markov [15]	69.3	6.7
Hybrid multi-step Markov [15]	49.3	3.2
Compression [15]	34.2	5.0
Sequence Match [8, 15]	26.8	3.7
IPAM [6, 15]	41.1	2.7
Naïve Bayes (w. Updating) [10]	61.5	1.3
Naïve Bayes (No Upd.) [10]	66.2	4.6
Semi-Global Alignment [4]	75.8	7.7
Sequence Alignment (w. Upd.) [5]	68.6	1.9
Eigen Co-occurrence Matrix [12]	72.3	2.5

Finally, Maloof and Stephens proposed a general system for detecting malicious insider activities by specifically focusing on violations of 'Need-to-Know' policy [9]. Although the work is not aimed directly at masquerade detection, such a system may reveal actions of a masquerader. They defined certain scenarios of bad behavior and combined evidence from 76 sensors to identify whether a user is malicious or not. Our approach is more generalizable and does not specify what bad behavior looks like. Instead, we only model normal behavior and detect deviations from that behavior.

3 Objective and Approach

When dealing with the masquerader attack detection problem, it is important to remember that the attacker has already obtained credentials to access a system. When presenting the stolen credentials, the attacker is then a legitimate user with the same access rights as the victim user. Ideally, monitoring a user's actions after being granted access is required in order to detect such attacks. Furthermore, if we can model the user's intent, we may better determine if the actions of a user are malicious or not. We have postulated that certain classes of user commands reveal user intent. For instance, search should be an interesting behavior to monitor since it indicates the user lacks information they are seeking. Although

user search behavior has been studied in the context of web usage mining, it has not been used in the context of intrusion detection.

We audit and model the volume and frequency of user activities related to search/information gathering and information access, assuming that the masquerader will exhibit different behavior from the legitimate user and this deviation will be easily noticed. Hence, this approach essentially tracks a user's behavior and measures any changes in that behavior. Any significant change will raise an alarm. User behavior naturally varies for each user. We believe there is no one model or one easily specified policy that can capture the inherent vagaries of human behavior. Instead, we aim to automatically learn a distinct user's behavior, much like a credit card customer's distinct buying patterns.

We use one-class support vector machines to develop user behavior models. SVMs are linear classifiers used for classification and regression. They are known as maximal margin classifiers rather than probabilistic classifiers. Schölkopf et al. [13] proposed a way to adapt SVMs to the one-class classification task. The one-class SVM algorithm uses examples from one class only for training. Just like in multi-class classification tasks, it maps input data into a high-dimensional feature space using a kernel function.

The origin is treated as the only example from other classes. The algorithm then finds the hyper-plane that provides the maximum margin separating the training data from the origin in an iterative manner. We note that SVMs are suitable for block-by-block incremental learning. As user behavior changes and new data is acquired, updating SVM models is straightforward and efficient. Prior data may be expunged and the support vectors computed from that data are retained and used to compute a new update model using the new data [17, 16]. Also the use of a one-class modeling approach means that we do not need to define a priori what masquerader behavior looks like. We only model normal user behavior. We can preserve the privacy of the user when building user models, as we do not need to intermix data from multiple user for building models of normal and attacker behavior.

4 Data Gathering and "Capture The Flag" Exercise

As we have noted, most prior masquerade attack detection techniques were tested using the Schonlau data set, where 'intrusions' are not really intrusions, but rather random excerpts from other users' shell histories. Such simulation of intrusions does not allow us to test our conjecture that the intent of a malicious attacker will be manifested in the attacker's search behavior. For this reason, we have collected our own dataset, which we will use for testing. However, for completeness, we test our detection approach as a baseline against the Schonalu dataset. The results will be reported in Section 6.3. In the following subsections, we describe our home-gathered dataset and the host sensor used to collect it.

4.1 Host Sensor

We have developed a host sensor for Windows platforms. The sensor monitors all registry-based activity, process creation and destruction, window GUI and file accesses, as well as DLL libraries' activity. The data gathered consisted of the process name and ID, the process path, the parent of the process, the type of process action (e.g., type of registry access, process creation, process destruction, window title change, etc.), the process command arguments, action flags (success or failure), and registry activity results. A time stamp was also recorded for each audit record. The Windows sensor uses a low-level system driver, DLL registration mechanisms, and a system table hook to monitor process activity.

4.2 RUU Dataset

In order to address one of the most significant shortcomings of the Schonlau dataset, namely the lack of 'real' intrusions, we gathered user data and *simulated* masquerader data by conducting a user study under IRB approval. [†] We refer to this data as the RUU (Are You You?) dataset.

Collecting Normal User Data Eighteen computer science students installed the Windows host sensor on their personal computers. The host sensor collected the data described in the previous subsection and uploaded it to a server, after the students had the chance to review the data and their upload. The students signed an agreement for sharing their data with the research community. This unique dataset with a size of more than 10 GBytes is available for download after signing a usage license agreement. The data collected for each student spanned 4 days on average. An average of more than 500,000 records per user were collected over this time.

Collecting Simulated Masquerader Data To obtain masquerader data, we developed a "capture the flag" exercise in our lab with user volunteers acting as masqueraders. In order to ensure the validity of the experiment and the resulting data, we have decided to control the 'user's intent'. We hypothesize that user search behavior is a behavioral feature that is impacted by the user's intent. If a masquerader is looking to steal information, their intent will be manifested in their search behavior through the volume of the search activities performed by the masquerader. To that extent, we have designed a user study experiment, where the intent to steal information is the independent variable that we control. A total of sixty computer science students were randomly selected to participate in the exercise from a list of volunteers. The participants were randomly assigned to one of three groups. Each group was given a different scenario to execute: a 'malicious attacker' scenario, a 'benign masquerader' scenario, or a 'neutral'

[†] Human user studies of insider threat and masquerade detection are approved at Columbia University under Human Subjects protocol IRB-AAAC4240.

scenario. Note that selecting users from a homogeneous group and using specific and detailed scenarios are known techniques in behavioral sciences used to reduce subject and experimental treatment variability [7].

Malicious Scenario: The participants in this first scenario were asked to perform a specific task, which consisted of finding any information that could be used for financial gain from a coworker's desktop computer. The participants had access to a coworker's computer for 15 minutes while he or she left out for lunch. An attacker can gain access to the victim's system in a different way, for instance remotely if they know the access credentials. Regardless of the access method, we are interested in the attacker's behavior after they access the victim's computer. Our scenario clearly described the financial difficulties that the user was going through and the personal problems they were having with the coworker. In particular, they believed that their coworker had been undeservingly promoted instead of them, making any PII information theft or the cause of financial damage to their colleague, that they might engage into, justifiable to them. The task was goal-specific in order to **capture the intent** of the users. The participants in this scenario knew that all of their actions were being monitored. While they were encouraged to find relevant information for their task, they were also urged to hide their masquerade activity. It is also important to mention that they **were not** specifically **asked to search** the file system in the scenario, i.e. some users could decide to look into the computer owner's e-mail, which constitutes another approach for information gathering, instead of searching files. After completing the task, the participants filled a questionnaire about their experience and behavior during the task, and how different it was from their own normal search behavior. Most participants indicated that they tried to leave the desktop in the same state that they accessed it, including the same list of recently opened documents. Many avoided using desktop search tools and opted for manually exploring the contents of the file systems by traversing directories of interest. Some students also cleared the browser's cache and history. Others interleaved search and information gathering actions with user normal actions that should not raise any suspicions, such as checking a news website.

Benign Scenario: Similar to the previous scenario, the participants in this scenario acted as employees in a certain organization. While their coworker left out for lunch and stayed logged in to their computer, their own computer experienced a hard drive failure. Because they had to finish working on an urgent team project, which they had to provide an update on during a meeting occurring later in the afternoon, they did not hesitate to use the coworker's computer to finish working on the project.

Neutral Scenario: The participants in this scenario too had a coworker leave for lunch while staying logged in to their desktop computer. However, the participants in this scenario had no compelling reason to access the coworker's computer. They were left to freely choose whether they wanted to access their coworker's desktop. We observed the behaviors of the participants, and whether they decided to access the coworker's desktop. In particular, we observed what

they did if they decided to access it. The participants had also to describe what they did and explain their decision after completing the experiment.

The participants in the user study in all three groups had unlimited access to the same file system for 15 minutes each. None of the users had access to this file system before, which was designed to look very realistic and to include potentially interesting patent applications, personally identifiable information, as well as account credentials. The file system had more than 100,000 files indexed by desktop search tools installed on the system, and totaling a size of more than 70 GBytes. The files were copied from a modern Windows XP machine routinely used in a university research environment. They were created over a period of 5 years. The file system included at least 80 files containing personal information that could be used for identity theft or financial gain. The contents of certain personal files were sanitized, so that no personal information was leaked. We also installed applications that typical computer science students would have on their own machines, such as programming APIs, media players, etc together with code projects, games, music, and video files. The goal was to make this machine look similar to the ones that the normal users in our experiment were using. Special care was taken to make sure that the desktop appeared in the same state to all participants in the experiment. While simulating masquerader attacks in the lab is not ideal, it was the best available option. None of the students who shared their normal usage data were willing to lend their computers in order to conduct masquerade attack experiments on them.

5 User Study Experiment

The objective of this experiment is to provide evidence for our conjecture that the masquerader's intent has a significant effect on their search behavior. We extracted three features from the data collected in the user study after experimenting with several features such as the frequencies of the different types of user actions and application events: (1) the number of files touched during an epoch of two minutes, (2) the number of automated search-related actions initiated by the masquerader, and (3) the percentage of manual search actions during the same epoch. Automated search actions are search actions launched using a desktop search tool such as *Google Desktop Search*. Manual search actions are file system navigation or exploration systems. We display the distribution of these features for the malicious, benign and neutral scenario participants of the user study in Figure 1. Our experiments show that the manipulation of the user intent has demonstrated a significant effect on user search behavior.

Figures 1(a), 1(b), and 1(c) are box-and-whisker plots of the three user search-related features. The horizontal line in the middle of each of the boxes in these plots corresponds to the median value of the displayed feature. Fifty per cent of the data falls within this box, while the top and bottom quartiles (25% of the data) of the data are represented by the whisker lines above and below this box. Data points whose value is above 1.5 times the upper quartile or lower than 1.5 times the lower quartiles are considered outliers and are represented as

small crosses. The short horizontal lines above and below the box represent the maximum and minimum data values excluding outliers.

Figure 1(a) depicts the distribution of the number of accesses to all files residing on the file system per a 2-minute epoch. It is clear that the average number of file touches amongst the malicious scenario group is much higher than in the other two groups.

Figure 1(b) shows the distribution of search-related queries to the registries and accesses to search-related DLLs and applications for the different groups of the user study. The average number of search-related records in the first group is comparable to that in the other groups, which might be unexpected. However, this could be explained by the fact that the participants in the first group were penalized if their illegal masquerade activities were discovered, which is why most of them decided to refrain from using desktop search tools to find relevant information. This choice is reflected in Figure 1(b) and in Figure 1(c).

In the latter figure, one can clearly notice that the percentage of user actions related to navigating through the file system in order to explore its contents is much more significant in the first group. The participants in the second group did not have the intention to steal any information, which is why they did not engage in file system exploration activities. While most of the participants in the third scenario decided not to access the coworker's computer, some of them did urged by their curiosity, as indicated in the participants' answers to our post-experiment questionnaire. Figure 1(c) shows that for this group, the users explored the directories in the file system in order to satisfy their curiosity.

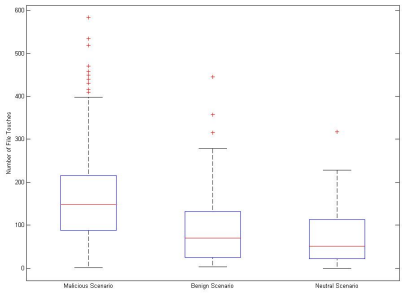
Finally, Figure 2 shows how the number of personal files accessed by masqueraders varies by user study scenario. The results of this user study provide evidence that search behavior is significantly affected by a masquerader's intent. The question that we attempt to answer next is: Can we model normal user search behavior and use it to detect malicious masqueraders?

6 RUU Experiment

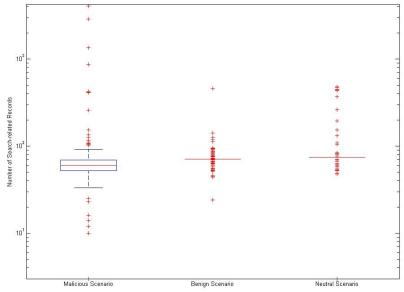
In order to evaluate our conjecture that search behavior modeling can provide a means for detecting malicious masqueraders, we use the normal user data to build user search behavior models. We then use the simulated masquerader data gathered for the participants in the 'malicious' scenario of our user study to test these user models. Here we describe our modeling approach, the experimental methodology, and the results achieved in this experiment.

6.1 Modeling

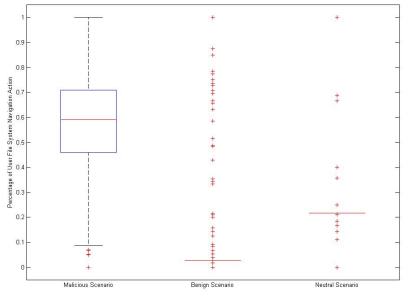
We devised a taxonomy of Windows applications and DLLs in order to identify and capture search and information gathering applications, as well as file system navigation user actions. The taxonomy can be used to identify other user behaviors that are interesting to monitor, such as networking-, communications-, or printing-related user activities. However, in the context of this paper, we only



(a) Distribution of File Touches across the three User Study Groups



(b) Distribution of Search-related Actions across the three User Study Groups



(c) Distribution of the Percentage of File System Navigation User Actions across the three User Study Groups

Fig. 1. Distribution of Search-related Features across the three User Study Groups

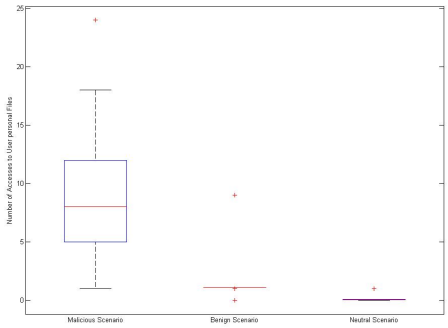


Fig. 2. The personal files accessed by masqueraders

use it to identify search- and file system navigation-related activities. Monitoring other user behaviors will be the subject of future work. The use of the taxonomy abstracts the user actions and helps reveal the user's intent.

We grouped the data into 2-minute quanta of user activity, and we counted all events corresponding to each type of activity within each of the 2 minute epochs. Eventually a total of three features were selected for each of those epochs. Each of the features is related to some aspect of the user's search or information gathering and information access behavior. These three features provided the best accuracy results in our experiments:

1. Number of **automated** search-related actions: Specific sections of the Windows registry, specific DLL's, access to specific index files, and specific programs, particularly desktop search tools, are correlated with system searching. For the 2 minute epoch, we model all search-related activity.
2. Number of file touches: Any file fetch, read, write, or copy action results into loading the file into memory. We count the number of times files are touched and loaded into memory by any process within each 2-minute epoch.
3. Percentage of file system navigation user actions: Not all search is performed using a desktop search tool. Navigating through the file system to explore its contents is also a form of user search. We model all **manual search** or file system navigation user activity occurring during the 2-minute epoch.

To identify the automated and manual search applications and user activities, we referred to our Windows applications taxonomy. The chosen features are simple search features that characterize search volume and velocity to test our hypothesis. While none of the features could be used to achieve high detection rates alone, the combination of the three features could be very effective. More complex search features that describe user search patterns could be extracted. Such features include, but are not limited to search terms and specific directory traversals. Evaluation of these features is the subject of our future work. for

more personalized and diversified user models that accurately model individual and unique user behavior.

6.2 Experimental Methodology

For each of the 18 normal users, the first 80% of their data were used for training a one-class SVM model. The user's test data and the masquerader data were kept separate. After the baseline models were computed, the same features used in the model were extracted for the test data after dividing them into 2-minute quanta of user activity. The models were tested against these features, and an empirically identified threshold was used to determine whether the user activity during the 2 minute-period was normal or abnormal. If the user activity was performed by the normal user, but was classified as abnormal by the ocSVM model, a false positive was recorded.

6.3 Detection Accuracy Evaluation

For evaluation purposes, we conducted two experiments. In the first one, we used one-class SVM models using the three features listed in Section 6.1. In the second experiment, we used the frequency of applications and processes within the 2 minute epoch as features for the ocSVM models. This is the modeling approach that achieved results comparable to those achieved by the naïve Bayes approach when applied to the Schonlau dataset [18], even though it is a one-class modeling approach, i.e. it uses less data for training the user models.

Accuracy Results Using the search-behavior modeling approach, 100% of the 2-minute quanta that included masquerader activity were detected as abnormal, while 1.1% of the ones with legitimate user activity were flagged as not confirming to the user's normal behavior. The results achieved are displayed in Table 2. The false positives (FP) rate is significantly reduced compared to the application frequency-based modeling approach, while a perfect detection rate is achieved. These results substantially outperform the results reported in the literature.

Table 2. Experimental results of ocSVM modeling approaches using search-behavior related features and application frequency features

Method	True Pos. (%)	False Pos. (%)
Search-behavior ocSVM	100	1.1
App.-freq. ocSVM	90.2	42.1

Monitoring file access and fetching patterns proved to be the most effective feature in these models. Consider the case where a user types *'Notepad'* in the search field in order to launch that application. Such frequent user searches are typically cached and do not require accessing many files on the system.

Note that if the attacker follows a different strategy to steal information, and decides to copy whole directories in the file system to a USB drive for later investigation, instead of identifying files of interest during one user session, then the 'file touches' feature will reflect that behavior.

Since each user has their own model with their own detection threshold, we cannot build a single Receiver Operating Curve (ROC) curve for each modeling approach. However we can compare the ROC curves for individual user models using the two modeling approaches investigated. One way to compare the ROC curves is to compare the Area Under Curve (AUC) scores. The higher the AUC score, the better the accuracy of the model. Figure 3 displays the AUC scores for all user models. The search-behavior modeling approach outperforms the application frequency based modeling approach for each user model. The average AUC score achieved for all ROC curves when modeling search behavior is 0.98, whereas the average AUC score for the application frequency-based models is 0.63. The bad performance of the application-frequency-based modeling approach can be explained by the high-dimensional feature vectors used in this modeling approach, which suggest that a lot more data may be needed for training.

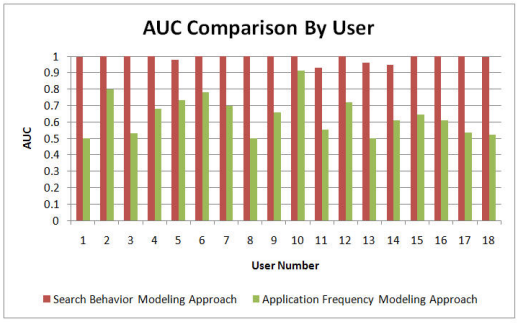


Fig. 3. AUC Scores By User for the Search Behavior and Application Frequency-Based Modeling Approaches using one-Class Support Vector Machines

Figure 4 depicts the number of ROC curves having AUC scores higher than a certain value for both modeling approaches. Note that for 12 user search behavior models, the AUC score is equal to 1 indicating the absence of any false positives.

The RUU data set consists of user data with varying amounts of data for different users. The amount of search behavior information varied from user to user. False positives were higher for users who contributed less data in general and less search-related data in particular than for those for whom we collected a large amounts of such data, such as users 11 and 14. For a 100% detection rate, the FP rate scored by these user models ranged between 11% and 15%, which

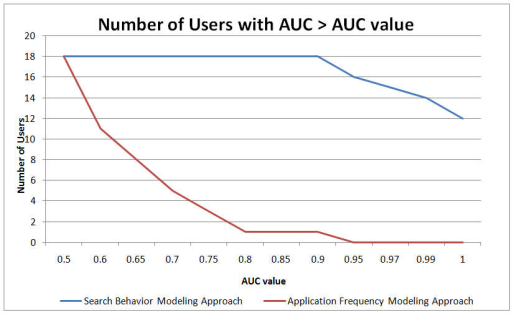


Fig. 4. The number of user models with AUC values greater than the value displayed on the x-axis for the search behavior and the application frequency modeling approaches using one-class SVMs. (The upper-left point shows 18 user models with AUC scores greater than 0.5)

proves the need for more training data for such users in order to improve the performance of the user models.

In summary, the significant accuracy improvement achieved can be explained by the fact that features used for modelign are good discriminators between normal user behavior and legitimate behavior. Despite the simplicity of the search features used, which only characterize search volume and velocity, we were able to reliably detect malicious masqueraders trying to steal information. We note that most masqueraders indicated in the post-experiment questionnaires that their strategy for finding relevant information started by quickly scanning the most recently opened documents, or the list of bookmarks. However, they still engaged in a wider search activity eventually when these sources proved fruitless.

Accuracy Results Discussion The results achieved using search behavior profiles require careful thought when considering the prior results using command sequences from the Schonlau dataset. Recall that the Schonlau dataset is not a 'true' masquerader dataset, since its 'intrusions' or 'masquerade' command blocks are just sequences of commands generated by randomly selected normal users. Search activities of the users may not be significant in this dataset. Furthermore, the Schonlau dataset does not include any timestamps, so temporal statistics cannot be extracted.

We introduce an alternative modeling technique focusing the analysis on specific types of user commands, namely information gathering or search commands. to accomplish the goal of accurately modeling user behavior we developed a taxonomy of Linux commands similar to the one we created for Windows applications and DLLs. We conducted an experiment where we followed the methodology described in prior work of Schonlau et al. [15] and Wang&Stolfo [18]. In this experiment, we measured the performance of one-class SVM models using fre-

quencies of simple commands per command block as features, and we compared the performance of ocSVM models using frequencies of command categories or specific behaviors (per the command taxonomy) as features. Table 3 shows the results achieved by the one-class SVM classifiers. The results confirm that the information that is lost by compressing the different user shell commands into a few categories does not affect the masquerader detection ability significantly. In section 6.4, we show how modeling search behavior by using the taxonomy of commands and applications reduces computational complexity, both for training and testing the classifier. This is possible thanks to the smaller number of features used for modeling, which reduces the amount of sampled data required for training, as the data becomes less sparse in the new feature space.

Table 3. ocSVM Schonlau Experimental Results

Method	**True Pos. (%)**	**False Pos. (%)**
ocSVM w/ simple cmds	98.7	66.47
ocSVM w/ taxonomy	94.8	60.68

In an operational monitoring system, one would be concerned with the error rate of a detector. The downside of a false positive is essentially annoyance by a legitimate user who may be alerted too frequently. An interesting problem to study is how to calibrate the modeling and detection frequency to balance the detector's false positive rate while ensuring its false negative rate is minimized. False negatives in this context, i.e. an undetected masquerader, are far more dangerous than an annoying false positive. A thorough evaluation of the right model checking and alerting frequency in light of average search times on a file system inter alia is the subject of ongoing research. Another focus of ongoing research is the correlation of search behavior anomaly detection with trap-based decoy files such as [2]. This should provide stronger evidence of malfeasance, and therefore improve the detector's accuracy. Not only would a masquerader not know the file system, they would also not know the detailed contents of that file system especially if there are well placed traps that they cannot avoid. We conjecture that detecting abnormal search operations performed prior to an unsuspecting user opening a decoy file will corroborate our suspicion that the user is indeed impersonating another victim user. Furthermore, an accidental opening of a decoy file by a legitimate user might be recognized as an accident if the search behavior is not deemed abnormal. In other words, detecting abnormal search and decoy traps together may make a very effective masquerade detection system. Ongoing work should establish evidence to corroborate this conjecture.

6.4 Performance Evaluation

Computational Complexity: Our experiment can be divided into four main steps: (1) identifying the features to be used for modeling, (2) extracting the

features to build the training and testing files, (3) building a ocSVM model for each normal user, and (4) testing each user model against the test data. We discuss the computational complexity of each of these steps for one user model.

Let o be the total number of raw observations in the input data. We use this data to compute and output the training vectors $x_i \in R^n, i = 1, ..., l$ and testing vectors $x_j \in R^n, j = 1, ..., m$ for each user u, where n is the number of features.

When using the application frequency features, this step requires reading all training data (about 0.8 of all observations o) in order to get the list of unique applications in the dataset. This step can be merged with the feature extraction step, but it would require more resources, as the feature vectors would have to remain in memory for updates and additions of more features. We chose to run this step in advance in order to simplify our program. This step is not required for the search behavior profiling approach, as all features are known in advance.

In the feature extraction step, we go through all input data once, grouping the observations that fall within the same epoch, and calculate and output n features for that epoch. This operation has a time complexity of $O(o + n \times (l + m))$.

Chang and Lin [3] show that the computational complexity of the training step for one user model is $O(n \times l) \times$#Iterations if most columns of Q are cached during the iterations required; Q is an $l \times l$ semi-definite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$; $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ is the kernel; each kernel evaluation is $O(n)$; and the iterations referred to here are the iterations needed by the ocSVM algorithm to determine the optimal supporting vectors.

The computational complexity of the testing step is $O(n \times m)$ as the kernel evaluation for each testing vector y_j is $O(n)$. We experimentally validate the complexity analysis in the next section to determine whether we have improved performance both in terms of accuracy and speed of detection.

Performance Results: We ran our experiments on a regular desktop with a 2.66GHz Intel Xeon Dual Core processor and 24GB of memory in a Windows 7 environment. We measure the average running time of each step of the experiment over ten runs. The results are recorded in table 4. As we pointed out in the previous subsection, the very first step is not executed in the our proposed search behavior modeling approach, but it takes more than 8 minutes when using the application frequency modeling approach. The running time of the feature extraction step shows that the number of raw observations in the raw data dominates the time complexity for this step. We point out that the RUU data set contains more than 10 million records of data.

The training and testing vectors are sparse, since only a limited number of the 1169 different applications could conceivably run simultaneously within a 2-minute epoch. This explains why the 389.7 ratio of features does not apply to the running time of the training and testing steps, even though these running times depend on the number of features n. While one might argue that, in an operational system, testing time is more important than training time, we remind the reader that a model update has the same computational complexity

as model training. For the latter, the use of a very small number of features as in our proposed approach clearly provides significant advantages.

All of these differences in running times culminate in a total performance gain of 74% when using the search behavior model versus the application frequency model typical of prior work. This computational performance gain coupled with improved accuracy could prove to be a critical advantage when deploying the sensor in an operational environment if a system design includes automated responses to limit damage caused by an insider attack.

Table 4. Performance comparison of ocSVM modeling approaches using search behavior-related features and application frequency features

Step	ocSVM app. freq.	ocSVM search-beh.
Identifying Features (min)	8.5	0
Extracting Features (min)	48.2	17.2
Training (min)	9.5	0.5
Testing (min)	3.1	0.5
Total (min) (Rounded)	**69**	**18**

7 Future Research

While the list of search applications and commands may have to be updated occasionally (just like an Anti-Virus needs periodic signature updates) for best detection results, most of the search-related activity would be manifested in accesses to search index files and regular user files on the system. An attacker could try to evade the monitoring system by renaming DLLs and applications so that they are assigned to a different category per our applications taxonomy, other than the search or information gathering category. Although we have not implemented a monitoring strategy to counter this evasive tactic, it is clear that a simple extension to the monitoring infrastructure can account for this case.

We assume that the attacker does not have knowledge about the victim's behavior. However, if the attacker has such prior knowledge, we propose combining user behavior profiling with monitoring access to well-placed decoys in the file system (as noted in Section 6.3) in order to limit the success of evasion. This should also help reduce false positives and present additional evidence od a masquerade attack, thus guiding the appropriate mitigation strategy.

A masquerader could choose to copy data to a USB drive for later examination. They may even choose to access the victim computer remotely and ex-filtrate data over the network. We could easily use the application taxonomy to monitor these specific behavior in case the attacker resorts to such strategies. As noted in section 6.3, the 'file touches' feature already captures some aspect of this behavior. The applications taxonomy could be used to extract 'Networking'-, 'Communications'- and I/O-related features to be included in the user model, so that such masquerader behavior gets detected easily.

8 Concluding Remarks

Masquerade attacks resulting in identity theft are a serious computer security problem. We conjecture that individual users have unique computer search behavior which can be profiled and used to detect masquerade attacks. The behavior captures the types of activities that a user performs on a computer and when they perform them.

The use of search behavior profiling for masquerade attack detection permits limiting the range and scope of the profiles we compute about a user, thus limiting potentially large sources of error in predicting user behavior that would be likely in a far more general setting. Prior work modeling user commands shows very high false positive rates with moderate true positive rates. User search behavior modeling produces far better accuracy.

We presented a modeling approach that aims to capture the intent of a **user** more accurately based on the insight that a masquerader is likely to perform untargeted and widespread search. Recall that we conjecture that user search behavior is a strong indicator of a user's true identity. We modeled search behavior of the legitimate user using three simple features, and detected anomalies that deviate from that normal search behavior. With the use of the RUU dataset, a more suitable dataset for the masquerade detection problem, we achieved the best results reported in literature to date: 100% masquerade detection rate with only 1.1% of false positives. Other researchers are encouraged to use the data set which is available for download after signing a data usage agreement [1].

In an operational monitoring system, the use of a small set of features limits the system resources needed by the detector, and allows for real-time masquerade attack detection. We note that the average model size is about 8 KB when the search-behavior modeling approach is used. That model size grows to more than 3 MB if an application and command frequency modeling approach is used. Furthermore, it can be easily deployed as profiling in a low-dimensional space reduces the amount of sampling required: An average of 4 days of training data was enough to train the models and build effective detectors.

In our ongoing work, we are exploring other features for modeling that could improve our results and extend them to other masquerade attack scenarios. The models can be refined by adding more features related to search, including search query contents, parameters used, directory traversals, etc. Other features to model include the use of bookmarks and recently opened documents which could also be used by masquerade attackers as a starting point for their search. The models reported here are primarily volumetric statistics characterizing search volume and velocity. We can also update the models in order to compensate for any user behavior changes. We will explore ways of improving the models so that they reflect a user's *unique* behavior that should be distinguishable from other legitimate users' behaviors, and not just from the behavior of masqueraders.

References

1. Ben-Salem, M. RUU dataset: http://www1.cs.columbia.edu/ids/RUU/data/.
2. Bowen, B. M., Hershkop, S., Keromytis, A. D., and Stolfo, S. J. Baiting inside attackers using decoy documents. In *SecureComm'09: Proceedings of the 5th International ICST Conference on Security and Privacy in Communication Networks* (2009).
3. Chang, C.-C., and Lin, C.-J. Libsvm: a library for support vector machines. http://www.csie.ntu.edu.tw/ cjlin/papers/libsvm.pdf, 2001.
4. Coull, S. E., Branch, J., Szymanski, B., and Breimer, E. Intrusion detection: A bioinformatics approach. In *Proceedings of the 19th Annual Computer Security Applications Conference* (2001), pp. 24–33.
5. Coull, S. E., and Szymanski, B. K. Sequence alignment for masquerade detection. *Computational Statistics and Data Analysis 52*, 8 (2008), 4116–4131.
6. Davison, B. D., and Hirsh, H. Predicting sequences of user actions. In *Working Notes of the Joint Workshop on Predicting the Future: AI Approaches to Time Series Analysis, 15th National Conference on Artificial Intelligence/15th International Conference on Machine Learning* (1998), AAAI Press, pp. 5–12.
7. Keppel, G. *Design and analysis : a researcher's handbook*. Pearson Prentice Hall, 2004.
8. Lane, T., and Brodley, C. E. Sequence matching and learning in anomaly detection for computer security. In *In AAAI Workshop: AI Approaches to Fraud Detection and Risk Management* (1997), AAAI Press, pp. 43–49.
9. Maloof, M. A., and Stephens, G. D. elicit: A system for detecting insiders who violate need-to-know. In *RAID* (2007), pp. 146–166.
10. Maxion, R. A., and Townsend, T. N. Masquerade detection using truncated command lines. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks* (2002), IEEE Computer Society, pp. 219–228.
11. Maxion, R. A., and Townsend, T. N. Masquerade detection augmented with error analysis. *IEEE Transactions on Reliability 53*, 1 (2004), 124–147.
12. Oka, M., Oyama, Y., Abe, H., and Kato, K. Anomaly detection using layered networks based on eigen co-occurrence matrix. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection* (2004).
13. Schölkopf, B., Platt, J. C., Shawe-taylor, J., Smola, A. J., and Williamson, R. C. Estimating the support of a high-dimensional distribution. *Neural Computation 13*, 7 (July 2001), 1443–1471.
14. Schonlau, M. Schonlau dataset: http://www.schonlau.net.
15. Schonlau, M., Dumouchel, W., Ju, W., Karr, A. F., Theus, M., and Vardi, Y. Computer intrusion: Detecting masquerades. *Statistical Science 16* (2001), 58–74.
16. Syed, N. A., Liu, H., Huan, S., Kah, L., and Sung, K. Handling concept drifts in incremental learning with support vector machines. In *In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-99* (New York, 1999), ACM Press, pp. 317–321.
17. Vapnik, V. N. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 1999.
18. Wang, K., and Stolfo, S. J. One-class training for masquerade detection. In *Proceedings of the 3rd IEEE Workshop on Data Mining for Computer Security* (2003).