Masquerade Attack Detection Using a Search-Behavior Modeling Approach

Malek Ben Salem and Salvatore J. Stolfo

Computer Science Department
Columbia University
New York, USA

{malek,sal}@cs.columbia.edu

Abstract

Masquerade attacks are unfortunately a familiar security problem that is a consequence of identity theft. Detecting masqueraders is very hard. Prior work has focused on user command modeling to identify abnormal behavior indicative of impersonation. This paper extends prior work by presenting one-class Hellinger distance-based and one-class SVM modeling techniques that use a set of novel features to reveal user intent. The specific objective is to model user search profiles and 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 extend prior research that uses UNIX command sequences issued by users as the audit source by relying upon an abstraction of commands. We devise taxonomies of UNIX commands and Windows applications that are used to abstract sequences of user commands and actions. We also gathered our own normal and masquerader data sets captured in a Windows environment for evaluation. The datasets are publicly available for other researchers who wish to study masquerade attack rather than author identification as in much of the prior reported work. The experimental results show that modeling search behavior reliably detects all masqueraders with a very low false positive rate of 0.1%, far better than prior published results. The limited set of features used for search behavior modeling also results in huge performance gains over the same modeling techniques that use larger sets of features.

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. The insider attack detection problem 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 develop novel algorithms that can effectively identify suspicious behaviors that may lead to the identification of imposters. 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 how the victim behaves when using a system. It is this key assumption that we rely upon in order to detect a masquerader. 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. 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 keylogger. 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 co-worker to pose as a masquerader. In the first two cases, the identity thief must log in with the victim's credentials and begin issuing commands within the bounds of one user session. We conjecture that legitimate users initiate the same repeated commands each time they log in to set their environment before using it, initiate some set of applications (read email, open a browser, or start a chat session) and similarly, clean up and shut down applications when they log off. Such repeated behaviors constitute a profile that can be modeled and used to check the authenticity of a user session early before significant damage is done. The case of hijacking a user's session is perhaps a bit more complicated. In either case, a monitoring system ought to detect any significant deviations from a user's typical profiled behaviors in order to detect a likely masquerade attack. Ideally, we seek to detect a possible masquerader at any time during a session.

In this paper we extend prior work on modeling user command sequences for masquerade detection. We use one-class support vector machines and introduce the use of the Hellinger Distance metric to compute a similarity measure between the most recently issued commands that a user types with a model of the user's command profile. Previous work has focused on auditing and modeling sequences of user commands including work on enriching command sequences with information about arguments of commands [1, 2, 3].

We propose an approach to profile a user's behavior based on a 'taxonomy' of UNIX commands and Windows applications. The taxonomy abstracts the audit data and enriches the meaning of a user's profile. Hence, commands or applications that perform similar types of actions are grouped together in one category making profiled sequences more abstract and meaningful. Furthermore, modeling sequences of commands is complicated whenever "Never-Before-Seen-Commands" are observed. A command taxonomy reduces this complexity, since any distinct command is replaced by its category, which is very likely to have been observed in the past. Commands are thus assigned a type, and the sequence of command types is modeled rather than individual commands.

One particular type of command is *information gathering* commands, i.e. *search* commands. 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 perform information gathering and search commands before initiating specific actions. To this extent, we conduct a second set of experiments using a Windows data set that we have gathered in our department. 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.

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, and presents the experiments conducted to evaluate whether a command taxonomy impacts the efficacy of user behavior models. In section 4, we present our home-gathered dataset which we call the RUU dataset. In section 5, we discuss experiments conducted by modeling search behavior using the RUU dataset. Section 6 discusses the results achieved, summarizes the contributions of the paper, and presents our ongoing work to improve and better evaluate our proposed modeling approach.

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 command line calls issued by users, system calls monitoring for unusual application use/events, 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 combined through logical operators, and the sequence or transition of events. However, most of this work failed to reveal or clarify the user's intent when issuing commands or running processes. The focus is primarily on accurately detecting change or unusual command sequences. In this section, we focus on the approaches reported in the literature that profile users by the commands they issue.

Schonlau et al. [1] applied six masquerade detection methods to a data set of "truncated" UNIX commands for 70 users collected over a several month period. Each user had 15,000 commands collected over a period of time ranging between a few days and several months [4]. 50 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", 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 complete data set and more information about it can be found at http://www.schonlau.net. The objective was to accurately detect the "dirty" blocks and classify them as masquerader blocks. 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). We note with interest that our taxonomy of commands reduces, if not entirely eliminates, the problem of modeling "Never-Before-Seen-Commands" since any command is likely to be categorized in one of the known classes specified in the taxonomy. Hence, although a specific command may never have been observed, members of its class probably were.

IPAM (Incremental Probabilistic Action Modeling), another method applied on the same dataset, and used by Davidson & Hirsch to build an adaptive command line interface, is also based on one-step command transition probabilities estimated from the training data [5, 6]. A compression

Table 1: Summary of accuracy performance of Two-Class Based Anomaly Detectors Using the Schonlau Data Set

Method	**True Pos. (%)**	**False Pos. (%)**
Uniqueness	39.4	1.4
Bayes one-step Markov	69.3	6.7
Hybrid multi-step Markov	49.3	3.2
Compression	34.2	5.0
Sequence Match	26.8	3.7
IPAM	41.1	2.7
Naïve Bayes (Updating)	61.5	1.3
Naïve Bayes (No Upd.)	66.2	4.6
Semi-Global Alignment	75.8	7.7
Eigen Co-occurrence Matrix	72.0	3.0
Naïve Bayes + EM	75.0	1.3

method has been also applied to the Schonlau data set 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, and was implemented through the UNIX tool compress which implements a modified Lempel-Ziv algorithm. A sequence-match approach has been presented by Lane & Brodley [7]. For each new command, a similarity measure between the most 10 recent commands and a user's profile is computed. A method, that is significantly different from other intrusion detection technologies, was presented by Coull et al. [8]. The method is known as semi-global alignment and is a modification of the Smith-Waterman local alignment algorithm. Oka et al. [9, 10] had the intuition 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). Based on that intuition they have developed the layered networks approach based on the Eigen Co-occurrence Matrix (ECM).

Maxion and Townsend [2] applied a naïve Bayes classifier, which has been widely used in text classification tasks, and they provided a thorough and detailed investigation of classification errors [11] highlighting why some masquerade victims are more vulnerable than others, and why some masqueraders are more successful than others. Maxion and Townsend also designed a new experiment, which they called the "1v49" experiment, in order to conduct this error analysis. Another approach called a self-consistent naïve Bayes classifier was proposed by Yung [12] and applied on the same data set. Wang and Stolfo used a naïve Bayes classifier and a Support Vector Machine (SVM) to detect masqueraders [3]. 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 datasets appear 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. The problem of effective and practical masquerade detection remains quite challenging.

Finally, Maloof and Stephens proposed a general system for detecting malicious insider activities by specifically violations of "Need-to-Know" policy [13]. Although the work is not aimed directly at masquerade detection, such a system may reveal actions of a masquerader. They define certain scenarios of bad behavior and combine evidence from 76 sensors to identify whether a user is malicious or not.

3 Objective and Approach

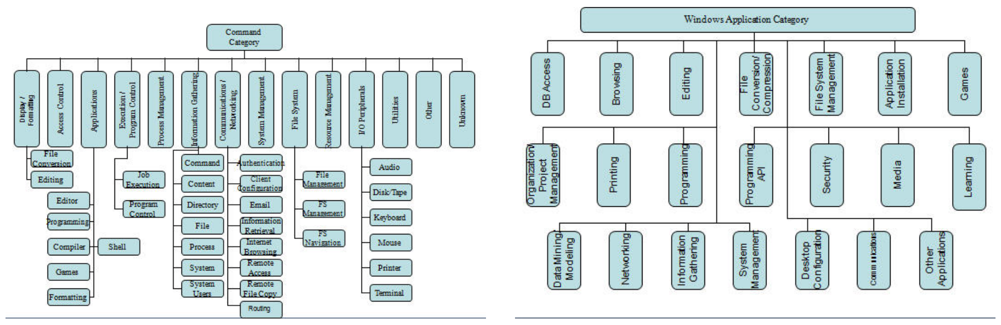
When dealing with the masquerader attack detection problem, it is important to remember that the attacker has already obtained credentials to access a system. When presenting the stolen credentials, the attacker is then a legitimate user with the same access rights as the victim user. Ideally, monitoring 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 [14, 15, 16], it has not been used in the context of intrusion detection. We define a taxonomy of commands to readily identify and model search behavior which appear using a variety of system-level and application-specific search functions. Another behavior that is interesting to monitor is remote access to other systems and the communication or egress of large amounts of data to remote systems, which may be an indication of illegal copying or distribution of sensitive information. Once again, the taxonomy defined allows a system to automatically audit and model a whole class of commands and application functions that represent the movement or copying of data. 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.

Our objective is to model the normal pattern of submitted commands of a certain user in a UNIX environment assuming that the masquerader will exhibit different behavior from the legitimate user and this deviation will be easily noticed. In order to detect the deviations, we compute the Hellinger distance between the frequencies of recent commands or command categories that show up in one block of commands of window size w and a second block of the same window size shifted by only one command. Hence, this approach essentially tracks a user's behavior and measures any changes in that behavior. Any significant change will raise an alarm. In the following we present the command taxonomy that we have developed as well as the Hellinger distance applied to blocks of issued commands.

3.1 User Command Taxonomy

We abstract the set of Linux/Unix commands and Windows applications into a taxonomy of command categories as presented in Figure 1(a). In particular, we are interested in identifying the specific set of commands that reveal the user's intent to search, to change access control privileges, and to copy or print information. Once these commands are identified, we can extract features representing such behavior while auditing the user's behavior.

The Unix taxonomy has 14 different categories: Access Control, Applications, Communications and Networking, Display and Formatting, Execution and Program Control, File System, I/O Peripherals, Information Gathering, Other, Process Management, System Management, Unknown, and Utilities. Most categories were further classified into sub-categories, however some did not require more granularity, such as the *Resource Management* category. The *Information Gathering* category includes commands such as **find** and **fgrep**. Examples of commands in the *Process Management* category include **kill**, **nohup**, and **renice**. **date**, **clock** and **cal** are examples of commands that fall in the *Utilities* category. The *Other* category includes commands that have



(a) Taxonomy of Linux and Unix user commands

(b) Taxonomy of Windows applications

Figure 1: Taxonomy of Linux and Unix Commands (a) and Windows applications (b)

been recognized but could not be classified under any other category. However, the *Unknown* category includes commands that were not identified or script names that are not recognizable. The Windows taxonomy is discussed in Section 5.

3.2 Hellinger Distance Experiment

3.2.1 Hellinger Distance

The Hellinger distance computes the change in two frequency tables. Each table is a histogram representing the frequency of some variable at some particular moment in time. In our work, we measure the frequency of command classes from the taxonomy (not the distinct command names) and the changes in their frequency. The Hellinger distance is defined as: $HD(f_p[], f_t[]) = \sum_{i=0}^{n-1}(\sqrt{f_p[i]} - \sqrt{f_t[i]})^2$ where $f_p[]$ is the array of normalized frequencies for the first set, $f_t[]$ the one for the second set, and n the number of possible commands/ command categories. This distance metric is applied whenever a user issues a command. A previous frequency table that modeled the previous commands is compared to a newly updated frequency table by modifying the frequency of the command types. Hence, each command creates a new Hellinger distance score that is subjected to threshold logic. Each bin of the frequency table is any chosen *category* of command extracted from the taxonomy. In the most general case all command categories would be tracked. The method is efficient to implement, but it remains to be seen how accurate it may be. Furthermore, the modeling of categories of commands may significantly reduce the information available when modeling sequences of commands.

3.2.2 Hellinger Distance Experiment Results

We use the Schonlau data set presented in Section 2, comprised of sequences of 15,000 commands for 50 users. We use the same experimental set up to be able to compare both results. For each user, there are between 0 and 24 masquerade blocks. In the first experiment, we apply the Hellinger distance to detect changes in user behavior. It is important to note that our approach is a one-class modeling approach. No positive examples of masquerade blocks were used for training, and no non-self models were computed for each user. In order to detect the changes in user behavior, we compute the Hellinger distance between two blocks of commands of size w=50. The window size was determined after an extensive study of each user's command blocks, and was found to achieve the best results. With each new command, we count the frequencies of the commands or command categories within the block of the last w commands. We shift the window of w commands by one command, and we count the same frequencies for the new block. Then we compute the Hellinger distance between the two sets of commands.

We also need to determine the threshold used to classify a block as a masquerader block or a normal user block. For this purpose we have computed the Hellinger distance for the first 5000 commands between consecutive blocks with a window of 50 commands, shifting the window by 1 command only. We thus determine the maximum Hellinger distance value for each user. In order to limit the false positive rate, we scaled the threshold value to 95% of the maximum value for each user model giving a distinct threshold to each model. Note that no inter-user data is mixed for this process. This constitutes the 'one-class' modeling approach over user command categories, and the method is relatively easy and efficient to implement. (A fixed size histogram would be maintained for each user at run-time in an operational setting.)

After computing one classifier for each user, the detection task includes computing the Hellinger distance in the same manner as above, but for the remaining 10,000 commands. The test of self-recognition is the same as in the Schonlau et al. paradigm, with between 76 and 100 blocks of self data presented to the detector for each user. Since the masquerader commands have been injected into blocks of 100 commands, we calculate one value out of all computed Hellinger distances for one block and we compare it with the classification threshold. Because the Hellinger distance scores are very noisy, we take the average score over the 100-command block as the overall score. If the average score is above the user threshold, the block is classified as a masquerader block. We do not use any blocks that have been classified as normal to update the user profile or recalculate the threshold. Significant improvement may be possible if models are updated.

The Area Under Curve (AUC), also known as the ROC score, which is a measure of the area under the ROC curve, reflects the performance of the detection method used. The higher the AUC is, the better the performance of the method. Figure 2 shows a user-by-user AUC comparison for all users whose files have been contaminated. Some users had no masquerader blocks injected, and therefore it was not possible to build a ROC curve for them. Comparing the average AUC for those scores, the taxonomy-based modeling method achieves more than a 1.8% improvement over the command-based one, with less information. The taxonomy-based approach outperforms the simple-commands-based approaches in most cases, and it only underperforms the other approach for very small AUC values.

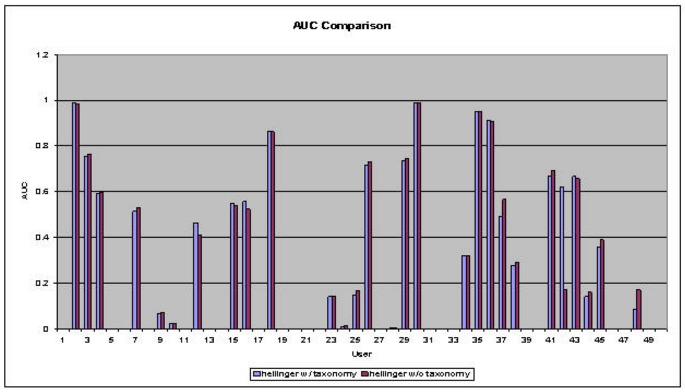


Figure 2: User-by-user comparison of ROC scores (AUCs)

3.3 One-Class Support Vector Machine Experiment

3.3.1 One-Class Support Vector Machines

Support Vector Machines (SVMs) are linear classifiers used for classification and regression. They are known as maximal margin classifiers rather than probabilistic classifiers. Schölkopf et. al [17] 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, such as the linear, polynomial, or Radial Basis Function (RBF) kernels. 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. The kernel function is defined as: $k(x, y) = (\Phi(x).\Phi(y))$,where $x, y \in X$, X is the training data set, and Φ is the feature mapping to a high-dimensional space $X \rightarrow F$.

3.3.2 SVM Experimental Set-Up

We used the LIBSVM package [18] to conduct our SVM experiments. It supports both multi-class classification and one-class classification. The one-class SVM function provided by this tool uses the RBF kernel. We have applied this kernel with the default settings to conduct the experiments. We have created a new version of the LIBSVM code, so that the one-class prediction models output the probability that a vector belongs to the "self" class, rather than output the classification value "self" or "non-self". We have used two different ways to represent features. The first is frequency-based where we count the number of times a simple command or a command category, retrieved using the command taxonomy, appears in the data set. The second approach is binary where we indicate whether the command or command category is present in the data set.

3.3.3 SVM Experiment Results

In this experiment we follow the methodology described in [1, 3], and we show that the performance of one-class SVMs (ocSVM) using command categories per our taxonomy is essentially the same as the performance of ocSVM that uses simple commands. We use the first 5000 commands of a user as positives examples for training the model. No negative examples are used for training. Then we test the classifier using the remaining 10,000 commands of the user, which may have injected command blocks from other users under a probability distribution described [4], which we will refer to as the "dirty" blocks.

Table 2: ocSVM Experimental Results

Method	**True Pos. (%)**	**False Pos. (%)**
ocSVM using simple commands (freq.-based model)	98.7	66.47
ocSVM using taxonomy (freq.-based model)	94.8	60.68
ocSVM using simple commands (binary model)	99.13	66.8
ocSVM using taxonomy (binary model)	86.58	56.9

Table 2 shows the results achieved by the one-class SVM classifiers. The ocSVMs that use the command taxonomy achieve better false alarm rates in both cases, when using the frequency-based model and when using the binary model. With the frequency-based model, the ocSVM using the command taxonomy also achieves comparable results with the ocSVM using simple commands. This is not true for the ocSVM using the binary feature. This can be explained by the difficulty to build representative models for the users based on a binary vector that only has 14 components, since the taxonomy is only made up of 14 different categories, particularly when the training data set includes 5,000 commands. With such a number of commands, the chances that all 14 categories of commands are represented are relatively high, and therefore the binary models built for the users will be very similar, i.e. the classifier will be unable to correctly decide whether a test command block is a self block or a masquerader block.

As mentioned, we modified the LIBSVM code so that the one-class prediction models output a probability that a vector belongs to the "self" class, rather than output the classification value "self" or "non-self". We have used these prediction values to build ROC curves for each model/user, and we show the corresponding AUC scores in Figure 3. The AUC scores show that, when using the frequency-based model to build the feature vectors, using the command taxonomy is comparable to modeling simple commands.

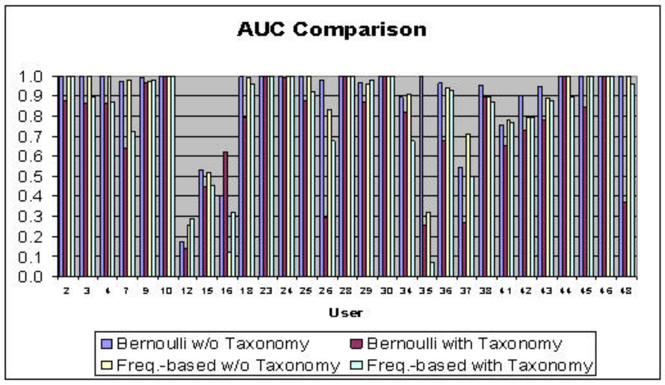


Figure 3: Comparison of AUC scores achieved using the 4 models in the SVM experiment

3.4 Discussion of the Schonlau Data Experiments

Unlike a modeling approach based on frequencies of simple commands, the taxonomy-based approach would not raise an alarm for a masquerader if, for instance, the same legitimate user starts running a different C compiler than what he or she normally uses. Both compilers used should be under the *Applications* category. So if the user behaves consistently, even if compilers change, the user model does not change if we use our taxonomy-based approach. However, using the simple commands approach might raise an alarm for a masquerade. Therefore, our approach is expected to limit the occurrences of false positives. Moreover, the taxonomy-based approach tends to reduce the problem of modeling "Never-Before-Seen-Commands" since any command is likely to be placed in a category with other similar commands, i.e., although a specific command may never have been observed, members of its class probably were.

The results shown above 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 order to further test this approach, we gathered *simulated* masquerader data by conducting a user study under IRB approval that will be described in the next section. This is a crucial step: The Schonlau datasets are not "true Masquerader" data sets. The data from different users were randomly mixed standing as a simulation of a masquerader attack. A willful act of malfeasance after identity theft is yet to be tested, albeit there is no generally available data set of this nature for scientific study. Hence, Schonlau resorted to simulating this malfeasance in as simple a fashion as possible, monitoring different users and mixing their data. It is fair to say, this mixture does NOT represent true malfeasance and willful intent.

In the next section, we describe the data that we have gathered, which we refer to as the RUU (Are You You?) dataset. The methodology and results described in the next section cannot be applied to the Schonlau datasets. The data captured lacks *timestamps* associated with the user commands. Hence, the modeling we propose that includes rates of emitted user events cannot be applied to the Schonlau datasets. As we shall see, the results achieved are far better with this new approach to modeling user behavior.

4 Data Gathering and "Capture The Flag" Exercise

In order to evaluate the search-behavior modeling approach, we needed to gather data, both normal user data and simulated masquerader data. To achieve this, we developed host sensors that could audit user activity and capture the data of interest.

4.1 Host Sensors

Two host sensors were developed: one for Windows and one for Linux. The Windows sensor monitored all registry-based activity, process creation and destruction, window GUI access, and DLL libraries activity. The data gathered consisted of the process name and ID, the process path, the parent of the process, the type of process action (e.g., type of registry access, process creation, process destruction, etc.), the process command arguments, action flags (success or failure), and registry activity results. A timestamp 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 activity. The Linux sensor used a kernel hook to audit all events on the host. It collected all process IDs, process names, and process command arguments in real time. The hooking mechanism used is the *auditd* daemon included in most modern Linux distributions.

4.2 RUU Dataset

Thirty four 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 filter any data that they were not willing to share. The students signed an agreement for sharing their data with the research community. This unique dataset with a size of more than 8GBytes is available for download. The data collected for each student was an average of 5 days worth of normal user data, ranging between 1 day and 59 days, and an average of more than 1 million records per user.

To obtain masquerader data, we developed a "capture the flag" exercise in our lab with user volunteers acting as masqueraders. 14 computer science students participated in the exercise, where they 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 students were handed a scenario where they were asked to perform a specific task, which consisted of finding any information that could be used for financial gain. The scenario clearly described the financial difficulties that the user was going through and the personal problems they were having with the computer's owner, a co-worker; In particular, they believed that their co-worker 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. It is also important to mention that the users 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 their 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 normal search behavior.

5 RUU Data Experiment

5.1 Modeling

A similar taxonomy to the user command taxonomy described in Section 3.1 was developed for Windows applications and DLLs. The taxonomy displayed in Figure 1 was particularly focused on categorizing search and information gathering applications, as well as document editing applications.

The data was grouped into 10 second quanta of user activity, and a total of 7 features were extracted for each of those epochs. Five of the features were only dependent on the events and data within the 10 second period of user activity:

1. Number of search actions: Specific sections of the Windows registry, specific DLL's, and specific programs on the system are correlated with system searching. For the 10 second time of user activity, we model all search actions of a user.
2. Number of non-search actions
3. Number of user-induced actions: e.g. manually starting or killing a process, opening a window, manually, searching for some file or some content, etc. These actions are distinguished from the ones that are automatically induced by running processes (e.g. a process performs a fork, etc.).
4. Number of window touches: e.g. bringing a window into the foreground, or closing a window, etc.
5. Number of new processes

The other 2 features qualified the general context that the user actions were performed in. These are as follows.

6. The total number of processes running on the system, and
7. The total number of document editing applications running on the system

5.2 Experimental Methodology

For each of the 34 normal users, the first 75% of their data was used for training a one-class SVM model. The data was first normalized to [0,1] using LIBSVM's scaling routine, which allows for saving the scale extracted from the training data to a file. That scale is then used for normalizing the test data of the same user as well as the masquerader data, by feeding the scale file to the scaling routine. Scaling prevents features with wide numeric ranges from dominating those with narrow ranges. The user's test data and the masquerader data were kept separate.

Two experiments were conducted to determine the value that may arise from using context features. Context features intend to represent information about prior user events. In the first experiment, only the first five features, measuring events for the current 10 second epoch of user activity are used for modeling, whereas in the second experiment, all seven features were used for modeling. The goal was to learn how important the overall context of the session was in detecting masquerade attacks.

After the baseline models were computed, the same features used in the model were extracted for the test data after dividing them into 10 second 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 10 second 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 is recorded.

5.3 Detection Accuracy Evaluation

86.24% of the 10-second quanta that included masquerader activity were detected as abnormal, while 12.58% 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 3. For evaluation purposes, we have also included the results of the ocSVM modeling approach which uses the frequency of applications and processes within the 10 second time quanta as features, and which achieved results comparable to those achieved by the naïveBayes approach when applied on the Schonlau dataset [3]. Our results show that the FP rate has been reduced by 67.8% while the detection rate only decreasted by 2.1%
.

Table 3: Experimental results of ocSVM modeling approaches using search-behavior related features and application frequency features

Method	True Pos. (%)	False Pos. (%)
Application-frequency based ocSVM without context features	88.1	39.1
Search-behavior-based ocSVM without context features	86.24	12.58
Search-behavior-based ocSVM with context features	100	0.1

Figure 4 depicts the number of ROC curves having AUC scores higher than a certain value for the search behavior modeling approach and the modeling approach based on application frequency. The average AUC score for all ROC curves achieved by these approaches was 0.75 and 0.67 respectively. The results of the search behavior modeling approach can be further improved by including features that add context to the user or masquerade activity; we achieve a 100% detection rate with a very low false positive rate of 0.1%, and an average AUC score of 0.996.

Figure 5 shows the total number of user models in relation to their AUC scores when context features are used. These results substantially outperform the results reported in the literature so far. However, it should be noted, that they should only be applied to a masquerade attack where the masquerader logs in to the victim's computer and starts a new session, since the masquerade data in our experimental setup was kept separate from the normal user's test data, while 2 of the features, describing the context of the session, were used for modeling.

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

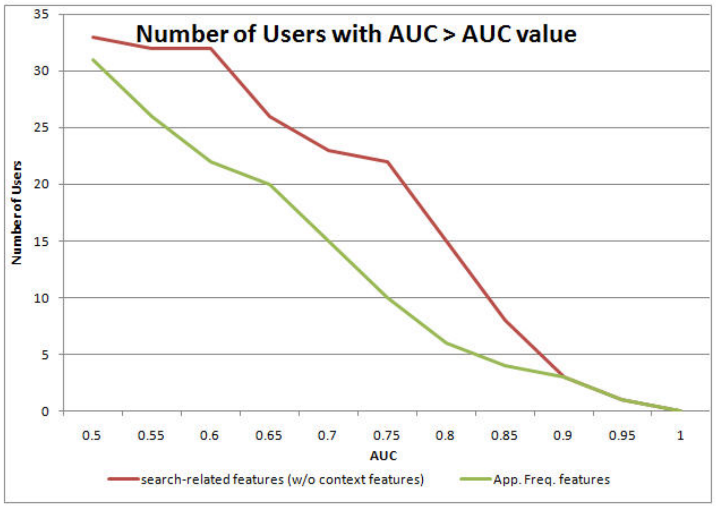


Figure 4: The number of user models with AUC values greater than the value displayed on the x-axis for the search behavior modeling and the application frequency modeling approaches using one-class SVMs. (The first point shows 33 user models with AUC scores greater than 0.5)

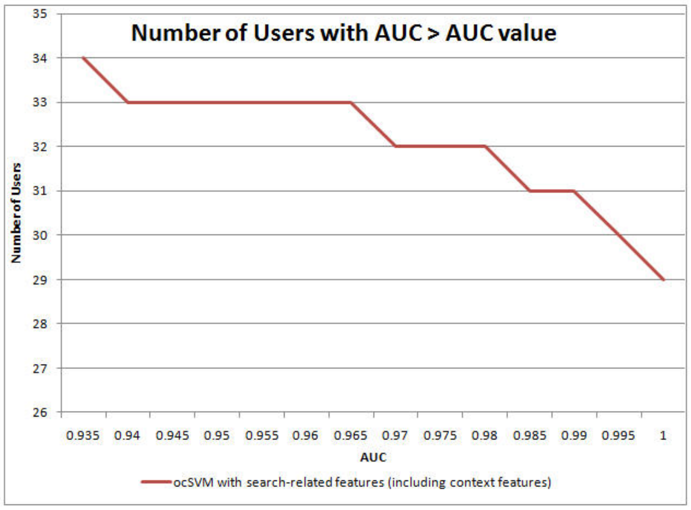


Figure 5: The number of user models with AUC values greater than the value displayed on the x-axis for the search behavior modeling with context features using one-class SVMs.

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 [19]. 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.

5.4 Performance Evaluation

5.4.1 Computational Complexity

Our experiment can be divided into five main steps: identifying the features to be used for modeling, extracting the features to build the training and testing files, normalizing those features using the LibSVM scaling routine, building a oCSVM for each normal user, and finally testing each ocSVM 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 used for modeling.

When using the application frequency features, this step requires reading all training data (about 0.75 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 $O(o + n \times (l + m))$.

The scaling operation is composed of two steps. During the first step, the scaling routine goes through all values of a particular feature in the training vectors, and determines the range of those values. Then, it determines the right scale to use in order to normalize them, and applies that scale to the feature values in the test vectors. That is done of course for each feature. The computational complexity for the whole operation is then $O(n \times (l + m))$.

Chang and Lin [20] 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 by l semidefinite 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.

5.4.2 Performance Results

We ran our experiments on a regular laptop with a 2.3GHz AMD Turion Dual Core processor and 4GB of memory in a Windows Vista environment. We measure the average running time of each step of the experiment over three runs. The results are recorded in table 4. As we pointed out in the previous section, the very first step is not executed in the our proposed search behavior modeling approach, but it takes more than 18 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 40 million records of data.

In the scaling step, which has a time complexity of $O(n \times (l+m))$, we clearly notice the impact of the number of features on the running time. The ocSVm modeling approach that uses application frequencies as features has a total of 829 features, which is the number of distinct applications that appear in the training data. The ratio of this number of features to the number of features used in the search-behavior based ocSVM modeling approach is 118.4, since we only use the 7 features described in section 5.1. The training and testing vectors are sparse, since only a limited number of the 829 different applications could conceivably run simultaneously within a 10-second epoch. This explains why the 118.4 ration 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. All of these differences in running times culminate in a total performance gain of 86% 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 sensors in an operation 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 w/ app. freq. feat.**	**ocSVM w/ search-beh. feat.**
Identifying Features (min)	18.5	0
Extracting Features (min)	99.5	86.5
Scaling (min)	361	3
Training (min)	9.5	1.5
Testing (min)	3	1.5
Total (min)	**743**	**104**

6 Discussion and Concluding Remarks

Masquerade attacks (such as identity theft and fraud) are a serious computer security problem. We conjectured that individual users have unique computer usage 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, and reducing the overhead of the masquerade attack detection sensor. Prior work modeling user commands shows very high false positive rates with moderate true positive rates.

In this paper, 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. We modeled search behavior of the legitimate user using a set of 7 features, and detected anomalies that deviate from that normal search behavior. We introduced a taxonomy of Unix commands and Windows applications in conjunction with one-class SVM modeling of user behavior in order to detect masqueraders in UNIX environments using a standard benchmark dataset, and in Windows environments using our own dataset. With the use of the latter 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 0.1% of false positives. This is partially due to incorporating temporal context via volumetric statistics in the modeling of user events. Other researchers are encouraged to use the data set we have made publicly available for download [21].

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. Furthermore, it can be easily deployed as profiling in a low-dimensional space reduces the amount of sampling required: An average of less than 5 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 including the case where masquerader activity happens within a legitimate user's session. The models can be refined by adding more features related to search, including search query contents, parameters used, and directory traversals, etc.. 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, known as concept drift. We will explore ways of improving the models so that they reflect a user's unique behavior that should be distinguishable from other legitimate user's behaviors, and not just from the behavior of masqueraders.

References

[1] M. Schonlau, W. Dumouchel, W. hua Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," *Statistical Science*, vol. 16, pp. 58–74, 2001.

[2] R. A. Maxion and T. N. Townsend, "Masquerade detection using truncated command lines," in *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks.* Washington, DC, USA: IEEE Computer Society, 2002, pp. 219–228.

[3] K. Wang and S. J. Stolfo, "One-class training for masquerade detection," in *Proceedings of the 3rd IEEE Workshop on Data Mining for Computer Security*, 2003.

[4] "Schonlau dataset." [Online]. Available: http://www.schonlau.net

[5] B. D. Davison and H. Hirsh, "Predicting sequences of user actions," in *Working Notes of the Joint Workshop on Predicting the Future: AI Approaches to Time Series Analysis, Fifteenth National Conference on Artificial Intelligence (AAAI98)/Fifteenth International Conference on Machine Learning (ICML98).* AAAI Press, 1998, pp. 5–12.

[6] ——, "Toward an adaptive command line interface," in *Proceedings of the Seventh International Conference on Human-Computer Interaction (HCI97).* Elsevier Science Publishers, 1997.

[7] T. Lane and C. E. Brodley, "Sequence matching and learning in anomaly detection for computer security," in *In AAAI Workshop: AI Approaches to Fraud Detection and Risk Management.* AAAI Press, 1997, pp. 43–49.

[8] S. Coull, J. Branch, B. Szymanski, and E. Breimer, "Intrusion detection: A bioinformatics approach," in *Proceedings of the 19th Annual Computer Security Applications Conference*, 2001, pp. 24–33.

[9] M. Oka, Y. Oyama, and K. Kato, "Eigen co-occurrence matrix method for masquerade detection," in *Publications of the Japan Society for Software Science and Technology*, 2004.

[10] M. Oka, Y. Oyama, H. Abe, and 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*, 2004.

[11] R. A. Maxion and T. N. Townsend, "Masquerade detection augmented with error analysis," *IEEE Transactions on Reliability*, vol. 53, no. 1, pp. 124–147, 2004.

[12] K. H. Yung, "Using self-consistent naïve bayes to detect masqueraders," in *PAKDD*, 2004, pp. 329–340.

[13] M. A. Maloof and G. D. Stephens, "elicit: A system for detecting insiders who violate need-to-know," in *RAID*, 2007, pp. 146–166.

[14] R. Baeza-Yates, C. Hurtado, M. Mendoza, and G. Dupret, "Modeling user search behavior," in *LA-WEB '05: Proceedings of the Third Latin American Web Congress.* Washington, DC, USA: IEEE Computer Society, 2005, pp. 242–251.

[15] J. Attenberg, S. Pandey, and T. Suel, "Modeling and predicting user behavior in sponsored search," in *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.* New York, NY, USA: ACM, 2009, pp. 1067–1076.

[16] M. O'Brien and M. T. Keane, "Modeling user behavior using a search-engine," in *IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces.* New York, NY, USA: ACM, 2007, pp. 357–360.

[17] B. Schölkopf, J. C. Platt, J. Shawe-taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," 1999.

[18] "Libsvm." [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

[19] B. M. Bowen, S. Hershkop, A. D. Keromytis, and S. J. Stolfo, "Baiting inside attackers using decoy documents," in *Columbia University Department of Computer Science, Technical Report CUCS-016-09*, 2009.

[20] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," 2001. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf

[21] "Ruu dataset." [Online]. Available: http://www1.cs.columbia.edu/ids/RUU/data/