Layered Security Architecture for Masquerade Attack Detection

Hamed Saljooghinejad, Wilson Naik Bhukya

Layered Security Architecture for Masquerade Attack Detection

Hamed Saljooghinejad,Wilson Naik Bhukya

Department of Computer and Information Science,
University of Hyderabad, Hyderabad, India
hamed.saljooghinejad@gmail.com
naikcs@uohyd.ernet.in

Abstract. Masquerade attack refers to an attack that uses a fake identity, to gain unauthorized access to personal computer information through legitimate access identification. Automatic discovery of masqueraders is sometimes undertaken by detecting significant departures from normal user behavior. If a user's normal profile deviates from their original behavior, it could potentially signal an ongoing masquerade attack. In this paper we proposed a new framework to capture data in a comprehensive manner by collecting data in different layers across multiple applications. Our approach generates feature vectors which contain the output gained from analysis across multiple layers such as Window Data, Mouse Data, Keyboard Data, Command Line Data, File Access Data and Authentication Data. We evaluated our approach by several experiments with a significant number of participants. Our experimental results show better detection rates with acceptable false positives which none of the earlier approaches has achieved this level of accuracy so far.

Keywords: Masquerade Detection, Intrusion Detection System, Anomaly Detection, User Profiling.

1 Introduction

Masquerade attacks are ranked second on the top five lists of electronic crimes perpetrated after viruses, worms or other malicious code attacks. The most common information, which can be used to detect masquerade attacks, is contained within the actions a masquerader performs. This set of actions is known as a behavioral profile. Behavior is not something that can be easily stolen. Masquerade detection techniques are based on the premise that when a masquerader attacks the system he will sufficiently deviate from the user's behavior and thus can be recognized using machine learning techniques [9]. In this paper we demonstrate an approach for detecting masqueraders in an efficient manner. We show how multiple layers of user data records together can construct a meaningful behavioral profile in order to have better detection results. The paper is organized as follows: next section introduces the related works on masquerade detection. Then, we describe architecture for the layered approach, which is followed by the experimental designs including data collection,

feature extraction, learning and classification phases. Results of several experiments and conclusion are presented in last sections.

2 Background and Related Work

Masquerade detection was done by observing the command line data and watching a user's behavior and then finding anomalies in their usage. In the category of the command line approaches, the initial activity was done by Schonlau et al. [6] which collected a dataset of Unix command line data from 50 users called the SEA dataset for testing and comparing and was used with different intrusion detection techniques. This research utilized different statistical techniques on the dataset and then compared the results. The NaïveBayes classifier was first applied on Schonlau's dataset by Roy A. Maxion [7]. They extended their previous work by applying the NaïveBayes classifier on Greenberg's enriched command line data [8]. Kim [5] applied SVM with a voting engine on SEA, 1v49 and Greenberg datasets.

However command line data detection mechanisms could not truly detect masquerade attacks in the modern graphical user interface (GUI) systems like windows and variants of Unix like Linux or Mac OSx. Today working with GUI systems is more common and the study of their different aspects is crucial. GUI based data is mostly related to data which comes from the interaction between the user and their mouse and keyboard. Poursa[12] considered Analysis of mouse data which was taken from users who worked with browsers. This approach has its disadvantages mainly because their work focused on the browser data, even though users may be working with applications other than browsers. Later works then focused on comprehensive GUI behavior. For this purpose, [1] developed an active system logger by using C# on a Windows XP System. This logger examined GUI event data captured from users and then useful parameters are extracted to construct the feature vectors. This profiling method, while good, comes with the disadvantage that they only implemented it for Microsoft GUI systems with much of the focus set only on mouse usage. Moreover, their detection rate was not impressive. [2] designed a logger in the KDE environment. The disadvantages of their work were that they collected data only from a single window and did not consider the complexity of profiling multi applications. It was [4] who later showed the advantages of user profiling across multiple applications. Other activity regarding GUI based detections can be found in [3] which does not use mouse movements or keystroke dynamics but rather profiles how the user manipulates the windows, icons and menus. They do not appear to consider time as a factor, which is crucial for intrusion analysis.

3 Our Layered Approach

We propose a new approach for detecting a masquerader in a system. A layered approach is introduced to collect comprehensive data across multiple applications. Fig.1 shows an overview of the architecture which will be discussed in this paper. As it is shown, the training phase is based on collecting the genuine user data from which a

behavioral profile will be created for each user and started with an event logger tool which is designed and implemented to collects all events during a session. Then a feature extraction tool is designed to generate the useful features. It constructs feature vectors which are generated from different layers; namely layer-1 as GUI data contains window data, mouse data, keyboard data, layer-2 as command line data, layer-3 as file access data and layer-4 as authentication data. This will then be used in the detection phase. In the detection phase the new user profile will be generated from the new user records and compared to the genuine profile. Any significant deviation between them can be recognized as an attack. This task can be possible with the help of a binary classifier which we can call the detection engine. We took the help of two well known classifiers for this task namely SVM and NaïveBayes. More details regarding the experiment will be addressed in later sections.
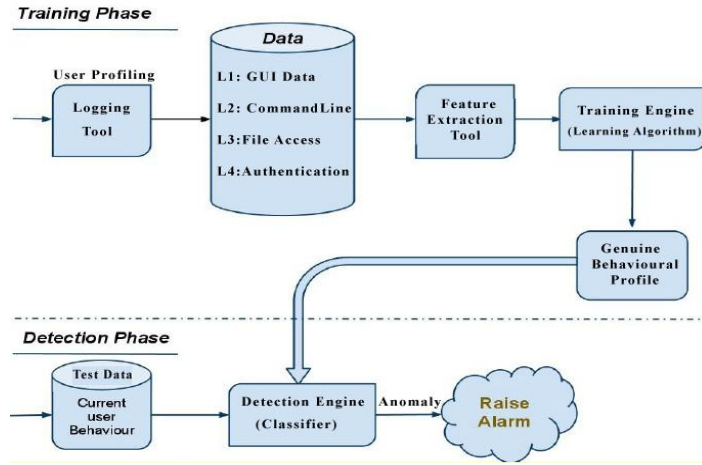

Fig. 1. Architecture of the Layered Approach

4 Data Collection & Calculation of Features

In the absence of a real-world data set for the study of masquerade attacks, we developed our own data collection project. In this section we described the data collection and feature extraction phases. For collecting the data our own logger was developed to collect the information from system. Since we needed data passing through Window Manager, we chose C as the programming language using X library which helped us communicate with Window Manager for capturing the events. The details of each event were logged to a file for processing in the next phase. The collected event details include identification of the window, the name of that window and time of occurrence along with different attributes of that particular event. At the preprocessing phase, by analyzing and observing the impact of each feature on detection rate and false positive rates, it was concluded that the following features were to be considered in the training phase. The features are generated from 6 categories including window data, keyboard data, mouse data as GUI data, command line data, file access data and authentication data as below:

4.1 Window Data(9features)

Users in the GUI environment try to interact with a particular window such as Maximizing, Minimizing, Opening, Closing and switching between windows. In total 9 features were extracted. Here is a window event sample:

```
Event Occurred at: Tue Feb 8 10:03:13 2011
WID=65011715-WName=Openoffice msg:The active window changed
from previous WID=69206091-Wname=Firefox
```

Window coordination and size (4). The average number of times that the user changes the x, y coordinates of a window or width or height of it per user session.
Window Maximize, Minimize (4). The average number of times that the user minimizes, maximizes, restores from the minimized or maximized position of a window per user session.
Window Switching (1). The average number of times that a user switches between different windows per user session.

4.2 Mouse Data(6featurs)

In this category mouse-related user activities like mouse click, mouse right click, mouse movement, etc. were captured for every application. In total 6 features were extracted. Here is a mouse event sample:

```
Event Occurred at: Tue Feb 8 09:58:22 2011
WID=650117—WName=Firefox--msg:Mouse left button clicked
```

Mouse Enter and Exit (2). The average number of times that the mouse enters to a window or exit from a window for each application per user session.
Mouse Clicks (2). The average number of left and right mouse clicks for each application per user session.
Mouse Scroll Up and Down (2). The average number of times that the mouse scrolls up or scroll down for each application per user session.

4.3 Keyboard Data(5features)

All the keyboard events are logged and stored separately for each application. The different keyboard events are key press, key release, and shortcut key (Ctrl, Alt, shift modifiers). In total 5 features were extracted. Here is a keyboard event sample:

```
Event Occurred at: Tue Feb 8 10:00:33 2011
WID=692060--WName=Firefox--msg:Shortcut key Pressed—Ctrl+z
```

Key Pressed (1). The average number of keys pressed in each application per user session.
Shortcut key Pressed (1). The average number of shortcut keys pressed in each application per user session.

Ctrl, Alt, Shift Modifier (3). The average number of times that Ctrl, Alt or Shift modifiers pressed for each application per user session.

4.4 Command Line Data(2features)

All the commands which are entered in command prompt are logged. They are divided into 2 major categories, Normal and Critical. Normal commands are those which are harmless and can be used with normal user privileges such as ls, clear, cd, etc. Critical commands are those used by administrators and contain any commands that are critical for the system and need root privileges such as su, sudo, etc.
Normal and Critical Commands (1). Number of normal and critical commands that the user enters to the command prompt.

4.5 File Access Data(38 features)

In this layer, users' accesses to crucial files were logged. Attackers usually try to access the victim's data. Analyzing this pattern of behavior can be helpful in order to determine when such an act is being conducted by a masquerader within the system. For example attackers try to access password files to obtain the users' password. Another example can be seen by the attempt to access log files by the attacker in order to delete any evidence they may have left behind. We used the Auditd tool to log the user access to specific files or folders. It is part of the Linux kernel auditing toolkit and captures auditing trails created by the kernel auditing facility from /proc/audit. Using the feature extraction tool, data was preprocessed and a total of 38 features are created. The first 19 features are successful attempts to access specific files or folders and the rest of them are features regarding failed access.
Successful and Unsuccessful access (38). Number of successful attempts to access particular files and folders include password file, log folder, etc folder, var folder, home directory, proc and bin folder as well as the number of unsuccessful attempts.

4.6 Authentication Data(2features)

Normal users rarely perform actions of administrative domain, so when unusual numbers of authentication occur, it can be suspected as malicious activity. From this 2 features are extracted, the first one is the total number of authentication request and the second one is the total number of failed authentication request.
Successful and Unsuccessful Authentication (2). Number of successful and unsuccessful authentications.

5 Learning and Classification

We collected the data from 16 distinct users who were the students of a particular course in our department. For each user an account was created on a shared computer in our lab and they were given an individual choice of operating conditions and appli-

cations. The data collection phase took about two months and the data was collected for multiple sessions, each lasting about 10 minutes. Roughly 3140 minutes of user data was collected and profiled which was significant due to the considerable number of users.

The dataset contains the different number of sessions for different users including 2 users with 21 sessions, 10 users with 20 sessions, 3 users with 19 sessions, and 1 user with 15 sessions as it is shown in table 1. This data was fed into the parsing engine to sanitize and extract features for each application per session. The methodology to train and test the data is explained below.

Table 1. Data Sets Description

Users	No of Sessions (each 10minutes)
A,B	2*21
C,D,E,F,G,H,I,J,K,L	10*20
M,N,O	3*19
P	1*15
Sum	314 (3140 minutes)

- The Collected feature vectors were then divided into different training and test sets. From these, the training sessions were used for the learning phase and for the test phase due to the lack of real masquerade records the other user's records were treated as non genuine or as masquerade records. Obviously if the real masquerade data was available, the anomaly detection would be easier due to the fact that masqueraders tend to put more effort into changing the system state than a normal user would.

- We started with the proportion of 50 percent for the training set vs. 50 percent of the testing set. To show the effect that the number of training sets has in relation to accuracy we repeated the experiments, increasing the proportion of the training set by 5 percent for each case. In total 8 different cases were constructed and it ended up with proportion of 85 percent for the training set and 15 percent for the testing set. Results show a better performance, with an increase in the number of training sets.

- The Collected feature vectors were divided into different training and test sessions per user. We defined a binary classification problem for our data due to the fact that users should be tagged as positive (genuine) or negative (masquerader). SVM [10] [11] and NaïveBayes [14] classifiers are used to classify the data and measure the detection rate, false positive and false negative rates. We used the Weka tool [13] to perform the classification. For improving the performance of classifiers we used the SMOTE filter [15].

6   Results and Discussion

We evaluated the performance of our approach with different experimentations as explained in previous section. Our best result was the detection rate of 97.5% with a false positive rate of 10.18% and false negative rate of 1.5% for SVM. Following sections explain more about the results and comparison between different approaches.

6.1 Detection Rate Evaluation by Different Number of Training and Test Sets

Table 2 shows the average detection rate for different numbers of training sets and test sets using different classifiers. In all cases the detection accuracy of SVM is far better than the other classifier. We can also see an improvement in detection rate by increasing the number of training sets.

Table 2. Average Detection Rate with Different Training and Test sets Size

Classifiers	50%Train 50%Test	55%Train 45%Test	60%Train 40%Test	65%Train 35%Test	70%Train 30%Test	75%Train 25%Test	80%Train 20%Test	85%Train 15%Test
SVM	95.83	96.02	96.59	96.55	96.92	96.89	97.05	97.55
NaïveBayes	91.06	91.2	91.51	91.65	91.61	91.83	91.28	91.34

6.2 ROC score Evaluation by Different Number of Training and Test Sets

A receiver operating characteristic (ROC), or simply ROC curve, is a graphical plot of the sensitivity, or true positive rate vs. false positive rate with equivalent value between 0 and 1. 1 means we have 100% detection rate with 0 false positive and 0 false negative. ROC analysis provides tools to select the optimal classifier. We calculated the average ROC scores for different numbers of training and test sets. For each user the ROC score is calculated and we then computed the average ROC for each classifier. Table 3 shows the better performance of SVM for our approach. Concretely SVM is famous for its use in this type of problem because it is a maximal-margin classifier as compared to NaïveBayes which is probabilistic and it has been known to be highly effective in text classification and providing better classification results with less training data.

Table 3. Average ROC Score with Different Number of Train and Test sets Size

Classifiers	50%Train 50%Test	55%Train 45%Test	60%Train 40%Test	65%Train 35%Test	70%Train 30%Test	75%Train 25%Test	80%Train 20%Test	85%Train 15%Test
SVM	0.894	0.905	0.914	0.914	0.919	0.924	0.932	0.941
NaïveBayes	0.848	0.851	0.861	0.863	0.868	0.883	0.895	0.902

6.3 Comparison With Other Approaches

To show the advantages of the layered approach, a brief comparison between the detection rate of our approach vs. the previous research works is been shown. As indicted in table 4, layered approach shows better detection rate than other methods either GUI [1] [2] [4] or Command line [5] [7] [8] approaches.

Table 4. Comparison of detection rate results achieved by different approaches

Metrics	Layered Approach	GUI Data			Command Line Data		
		[1]	[2]	[4]	[7]/[8]	[5]	[5]
Number of Users or Database Used	16	3	8	3	Greenburg	Greenburg	SEA/1vs49
Detection Rate	97.55	91.41	94.88	91.7	70.9/82.1	87.3	80.1/94.8

7   Conclusion

In this paper, we described and developed a new framework for constructing comprehensive feature vectors in different layers for the improvement of masquerade detection accuracy. After capturing the events of a user we went through preprocessing and then extracted relevant features for each application to then construct the relevant feature vectors. We considered six different layers to collect the data, consisting of window data, mouse data, keyboard data, command line data, file access data and authentication data. These feature vectors are classified for masquerade detection using two classifiers namely SVM and NaïveBayes. Our experiments show that this layered approach is well classified using SVM and thus provides better masquerade detection capabilities than single layer approaches. Moreover we observed the impact of increasing the number of training sets which led to an improvement of the detection rate.

References

1. Garg, A., Rahalkar, R., Upadhyaya: Profiling Users in GUI Based Systems for Masquerade Detection. In Proc. of 2006 IEEE Information Assurance Workshop(IAW),New York,2006
2. W. Bhukya, S. Kommuru, and A. Negi: Masquerade detection based upon GUI user profiling in Linux systems. ASIAN 2007, vol4846, LNC, pages 228–239. Springer, 2007.
3. Imsand, E.S., Hamilton Jr., J.A.: GUI Usage Analysis for Masquerade Detection. In: Proceedings of 2007 IEEE, Information Assurance Workshop (IAW 2007), New York(2007)
4. H. Saljooghinejad, W. Bhukya: Multi Application User Profiling for Masquerade Attack Detection. In Proceedings of International Conference on Advances in Computing and Communications (ACC 2011), Part II, CCIS191, pp. 676-684
5. Kim, H.S., Cha, S.D.: Empirical evaluation of svm-based masquerade detection using Unix commands. Computers and Security 24(2) (2005) 160-168
6. Schonlau, M., DuMouchel, W., Ju, W.-H., Karr, A.F., M.T., Vardi, Y.: Computer Intrusion: Detecting Masquerades. Statistical Science 16, 58–74 (2001)
7. Maxion, R.A., Townsend, T.N.: Masquerade Detection Using Truncated Command Lines. In: Proceedings of Int. Conf. on Dependable System &Networks (DSN 2002), pp. 219–228
8. Maxion, R.A.: Masquerade Detection Using Enriched Command Lines. In: Proceedings of Int. Conference on Dependable Systems and Networks (DSN 2003), CA (June 2003)
9. T. Lane and C. E. Brodley, "An Application of Machine Learning to Anomaly Detection," In Proceedings of 20th National Information System Security Conf,vol1.pp.366–380, 1997
10. T. Joachims, Text Categorization with SVM: Learning with many relevant features, In Proceedings of European Conference on Machine Learning ECML-98, pp.137-142, 1998.
11. Joachims, T., ''Transductive Inference for Text Classification Using Support Vector Machines,'' Proc. European Conf. Machine Learning (ECML'99), June 27–30, 1999.
12. Pusara, M., Brodley, C.: User Re-authentication via mouse movements. In: Proceedings of the ACM workshop on visualization and data mining for computer security, USA(2004)
13. http://www.cs.waikato.ac.nz/ml/weka/
14. A. McCallum, K. Nigam. A comparison of event models for naivebayes text classification. In Learning for Text Categorization, AAAI Workshop, 27 July 1998, Wisconsin, pp41–48.
15. N. V. Chawla, L. O. Hall, K. W. Bowyer. SMOTE: Synthetic Minority Oversampling Technique. Journal of Artificial Intelligence Research, 16:321--357, 2002.