San Jose State University
SJSU ScholarWorks

Master's Projects

Master's Theses and Graduate Research

Spring 2013

User Profiling in GUI based Windows Systems for Intrusion Detection

Arshi Agrawal
San Jose State University

Recommended Citation

Agrawal, Arshi, "User Profiling in GUI based Windows Systems for Intrusion Detection" (2013). *Master's Projects*. 303.
DOI: https://doi.org/10.31979/etd.7jsu-v3ms
https://scholarworks.sjsu.edu/etd_projects/303

User Profiling in GUI based Windows Systems for Intrusion Detection

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Arshi Agrawal

May 2013

The Designated Project Committee Approves the Project Titled

User Profiling in GUI based Windows Systems for Intrusion Detection

by

Arshi Agrawal

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2013

Dr. Mark Stamp	Department of Computer Science
Dr. Chris Pollett	Department of Computer Science
Dr. Sami Khuri	Department of Computer Science

ABSTRACT

User Profiling in GUI based Windows Systems for Intrusion Detection

by Arshi Agrawal

Intrusion detection is the process of identifying any unauthorized access to a system. This process inspects user behavior to identify any possible attack or intrusion. There exists two type of intrusion detection systems (IDSs): signature-based IDS and anomaly-based IDS.

This project concentrates on anomaly-based intrusion detection technique. This technique is based on the deviation of intruder's actions from the authenticated user's actions. Much previous research has focused on the deviation of command line input in UNIX systems. However, these techniques fail to detect attacks on modern GUI-based systems, where typical user activities include mouse movements and keystrokes.

Our project aims to create a dataset suitable for testing intrusion detection strategies on GUI-based operating systems. We have developed an event logging tool to capture GUI-based user data on Windows systems. We have collected a large dataset which we analyze using a intrusion detection strategy based on hidden Markov models (HMM).

ACKNOWLEDGMENTS

I would like to thank Dr. Mark Stamp for his constant support, guidance and encouragement provided throughout the project. His patience and thoughtfulness was one of the biggest supporting factor for me in this project. I would also like to express my sincere gratitude to my committee members Dr. Chris Pollett and Dr. Sami Khuri for their valuable time and guidance.

I am grateful to my friends and family for helping me in collecting a large amount of user data and spending their time and energy in making my project successful. Finally, I would like to thank my husband Mr. Arun Agrawal for his encouragement and unending patience throughout my Masters.

TABLE OF CONTENTS

CHAPTER

1 Introduction 1

2 Background 3

2.1 Introduction 3

2.2 Intrusion Detection 4

2.3 Hidden Markov Model 7

2.3.1 Notation 8

2.3.2 The Three Problems 9

2.4 ROC Curves 10

2.4.1 Possible Outcomes 10

2.4.2 ROC Curve Analysis 12

2.4.3 Area Under The Curve (AUC) 13

3 Event Logging for GUI-based Systems 14

3.1 Mouse Interaction 15

3.2 Keyboard Interaction 16

3.3 Active Applications 18

3.4 The Event Logger Tool 19

4 HMM-Based Intrusion Detector 21

4.1 Converting User Files To Dataset 21

4.2 Training HMM With Dataset 22

4.3 Experimental Setup 22

4.4 Training HMM With Mouse And Keyboard Data 23

5 Results 24

5.1 HMM Training With Mouse Logs 24

5.2 HMM Training With Keyboard Logs 29

6 Conclusion and Future Work 33

APPENDIX

Hooks 38

A.1 Introduction 38

A.2 Hook Procedures 38

LIST OF TABLES

1 HMM Notations dataset 8

2 Comparison of Command line dataset and GUI-based dataset 15

3 Sample dataset showing the Mouse Interaction 17

4 Sample dataset showing the Keyboard Interaction 18

5 Sample dataset showing the Active Applications 19

6 Combining mouse and keyboard scores, N = 3 32

LIST OF FIGURES

1 Hidden Markov Model 9

2 Possible outcomes 11

3 Distribution of results with the possible predictions 12

4 TPR and FPR 13

5 ROC curve 13

6 Architecture of anomaly-based intrusion detection based on GUI interactions 16

7 Log likelihood per observation of user 1 Vs intruders using mouse data, N = 2 25

8 Log likelihood per observation of user 1 Vs intruders using mouse data, N = 3 25

9 ROC curve for user 1, Mouse data, N = 2 27

10 ROC curve for user 1, Mouse data, N = 3 27

11 Log likelihood per observation of user 18 Vs intruders using mouse data, N = 2 28

12 Log likelihood per observation of user 18 Vs intruders using mouse data, N = 3 28

13 ROC curve for user 18, Mouse data, N = 2 29

14 ROC curve for user 18, Mouse data, N = 3 29

15 Log likelihood per observation of user 1 Vs intruders using keyboard data, N = 2 30

16 Log likelihood per observation of user 1 Vs intruders using keyboard data, N = 3 31

17 ROC curve for user 1, Keyboard data, N = 2 31

18 ROC curve for user 1, Keyboard data, N = 3 32

CHAPTER 1

Introduction

An intruder is an attacker who impersonates a legitimate user to perform some illegitimate activity on the user's computer system. In general, the intrusion detection systems observes the ongoing activities on the system and compare it with the user's normal activities to identify malicious behavior or an intrusion [6].

This project considers intrusion detection techniques based on anomaly-based intrusion detection [13]. In anomaly based detection systems, a model is generated based on normal user behavior. This model is then compared with the user activities during the detection phase, and any deviation is considered to be an anomoly or a possible attack. In genereal, a threshold value is used to determine the extent of deviation, based on which an anomaly is considered as a possible attack or an intrusion.

The intrusion detection technique used in this paper is also based on the deviation of intruder's actions from the authentic user's actions. In order to trace this deviation, we built user profiles by collecting and processing data from user sessions. This data was used to train user behavior model, to recognize the deviations from the normal behavior. The resulting model is then used to detect any malicious activity or intrusion. The detection mechanism becomes more difficult when the intruder is able to mimic the legitimate user's behavior to a large extent. In this paper, we consider a intrusion detection technique based on hidden Markov models (HMM).

There has been much research in the field of anomaly-based intrusion detection. Much of the previous research focused on Unix-like systems and relied on command line activities for intrusion detection [6, 13]. However, these techniques are not directly applicable to modern GUI-based systems, where typical user activities include mouse movements and keystrokes. Thus, command line data alone cannot efficiently detect intrusion attacks on such systems [4].

In this paper, we discuss an event logging tool that we developed. This tool has been used to capture a significant amount of GUI-based user data for Windows systems. We then analyze the effectiveness of an HMM-based intrusion detection technique on our collected data.

This paper is organized as follows. Chapter 2 discusses previous research done on intrusion detection and relevant background information for our research. Chapter 3 considers data capture in a GUI environment and provides details on the data capturing tool that we have developed. Chapter 4 discusses our HMM-based detector. In Chapter 5 we give results for the HMM-based detector when applied to data collected from a GUI environment. Finally, in Chapter 6 we provide our conclusions and discuss possible directions for future work.

CHAPTER 2

Background

The anomaly-based intrusion detection problem has been an area of research in the field of security for a very long time now. Therefore, it is important to get an overview of the previous research done in this area.

2.1 Introduction

Consumers and businesses spend billions of dollar every year for implementing various computer security measures. More and more start-ups, and other companies are trying to propose new approaches to computer security [21]. Even after the presence of many advanced security measures, username and password are still the most common authentication measures used by modern computer systems. These systems that use such trivial authentication measures are more prone to intrusions, as such credentials can be easily compromised to an insider. When an insider uses the compromised but legitimate credentials to gain access to the computer system of a user, the detection of the attack becomes extremely difficult. Anomaly-based intrusion attacks are considered to be difficult to detect as unlike signature-based detection techniques, here there are no known pattern (signature) of the attacks. The attackers use technically legal and legitimate means to get access to the system [9, 27].

The information used to detect intrusion attacks is contained in the actions of the intruder. This set of actions, considered as behavior profile of users forms the basis of intrusion detection system (IDS). IDSs monitor the activities occuring on the computer system and look for the malicious or unusual activities [5]. Signature-based detection and anomaly-based detection are the two approaches used by intrusion detection systems.

Signature-based detection systems are useful in identifying known attacks where the IDS looks for a predetermined pattern that represents an attack. If the system finds a match in the pattern, it identifies the attack and raises the alarm. However, this type of detection technique is not useful in detecting unknown attacks. Anomaly-based detection systems compare the ongoing user activities to the user's normal behavior model. Any deviation from the behavior model is considered a possible attack [13]. To make the distinction between the normal user behavior and the intruder's behavior, the detection system collects user data to build user profiles. This data is then trained for the detection system to understand what is normal and to identify the malicious behavior.

2.2 Intrusion Detection

Schonlau dataset is considered to be the first important accomplishment on the problem of intrusion detection. In his paper, Schonlau collected a dataset based on Unix command line data of 50 users for testing and comparing various intrusion detection methods [23]. However, as this dataset was based solely on command line data, it was not useful to represent users working on graphical user interface (GUI) based operating systems such as Microsoft Windows and Linux. This is due to the reason that command line data does not capture user's actions such as mouse or keyboard activities, which would determine user's behavior on these systems more accurately.

Some researchers used Schonlau dataset to detect intrusion attacks using various techniques [5]. There also exists a lot of research work in the area of masquerade detection, which is a specific case of anomaly-based intrusion detection [3, 4, 9, 17].

Research has been carried to compare the techniques used for intrusion detection [13]. According to the existing research, there are seven general approaches to anomaly-based intrusion detection (precisely, masquerade detection). These are summarized as follows [3, 13]:

- Information-theoretic - This approach is based on the theory that commands issued by the legitimate user will compress more than those issued by the intruder. However, this theory failed to give any promising results.

- Text mining - This is a data mining approach where repetitive command sequences are extracted from the training data and are used for scoring.

- Hidden Markov model (HMM) - This is the technique used in our paper and is discussed in detail in the following subsection.

- Naive Bayes - This technique is based on the frequency of commands, and does not take any sequence in consideration.

- Sequence and bioinformatics - This approach relies completely on extracting sequence related information.

- Support vector machines (SVM) - This is a machine learning algorithm used to separate data points by mapping the points in the original space to a high dimensional space. This mapping to a high dimensional space makes the separation easier, and is useful for classification and regression analysis.

- Other approaches - This category comprises of the approaches that do not fit into any of the above categories. These approaches were used by different researchers at different intervals of time. For instance, a hybrid Bayes one step

Markov approach and a hybrid multi step Markov approach [23]. Also, some researchers tried to improve the above techniques by combining them [2].

With the advent of GUI-based operating systems, researchers started to argue about the importance of similar dataset as Schonlau's dataset, but based on user actions in GUI-based environment. Different researchers used different approaches to the common problem of intrusion detection.

There exists research that models user behavior based on mouse movements [11, 12, 22]. Here the authors used the data collection technique in a controlled environment where users were asked to view same web pages by using Internet Explorer. However, this approach resulted in a large false positive and negative rates. Similarly, there were algorithms proposed for anomaly-based intrusion detection, based on the user's typing patterns [19, 25]. Authors considered characteristics such as typing speed, accuracy and inter-character delays for proposing this type of algorithm [15]. However, the algorithm required a two-way communication between server and client, for the validation of the login. This algorithm used a parameter termed criterion factor and reduced the false positive to a large extent. But the use of another parameter was not very clear from the paper.

There have also been several efforts of user behavior profiling methods by monitoring system calls, analyzing audit logs and program execution traces. These methods used call stack information for anomaly detection [8].

Although modeling of user behavior based on above criterias was able to detect intrusion attacks on GUI-based systems to a certain extent, the user profile created did not completely take all user actions into account. As a next step in this research area, a framework was proposed for collecting user behavior based on mouse activity, typing speed as well as background processes [9]. The authors used binary classification problem for user identification and intrusion detection, and used support vector machine for learning and classifying user profile parameter sets. The technique was claimed to have a high detection rate with few false positives. However, the dataset was not made available to public due to copyright issues.

On the similar research line, researchers extended the work by using other detection techniques in combination with the use of GUI manipulations for the identification. Researchers used Artificial Neural Networks (ANNs) as a basis for identification [14], Another framework called USim was also proposed, which generates user behavior data based on parameters like user intentions, user skill level, set of applications installed on a machine, mouse movement and keyboard activity [10].

All these research work contributed in providing useful frameworks and techniques for anomaly-based intrusion detection in GUI-based operating systems, and provided us with their useful research knowledge and findings. However, still the pace of research work in this area of intrusion detection is seriously hampered due to the lack of a publicly available dataset that can be trained to find efficient detection techniques. We aim to solve this problem by providing with such dataset. Also, we will use Hidden Markov model for identification and intrusion detection.

2.3 Hidden Markov Model

Hidden markov model (HMM) is a machine learning technique that is based on markov chains. In a machine learning technique, an algorithm is trained with samples of previous data and this trained data is used to get some feedback for unknown cases [18]. The data used for training purposes need to be sufficient to get a proper feedback for different cases.

Table 1: HMM Notations dataset

Symbol	Description
T	length of the observation sequence
N	number of states in the model
M	number of observation symbols
Q	distinct states of the Markov process
V	set of possible observations 0, 1, ..., M-1
X	hidden state sequence $(X_0, X_1, ..., X_{T-1})$
A	state transition probability matrix
B	observation probability matrix
π	initial state distribution
O	observation sequence $(O_0, O_1, ..., O_{T-1})$
λ	Hidden markov model, where λ = (A,B,π)

In markov chains the states are visible to the observer, while in hidden markov model the states are hidden from the observer. HMM is a finite state model, which has state transition probabilities as well as probability distribution over all possible output symbols in each state [20]. Each hidden state in HMM has two probabilities associated to them, the transition probability to another state as well as probability of being in the same state [7].

HMM is trained using a sequence of observations that is termed as training data. The chosen sequence of states by the model maximizes the probability of the observation sequences. After the construction of the model, the unknown or test data is tested to find the similarity between the test data and training data.

2.3.1 Notation

The HMM model can be understood using the notations shown in Table 1 [26].

In general, a hidden markov model can be represented as in Figure 1. The dashed line in the figure represents the hidden states of markov process, which are known by the current state and the matrix A. The observation sequence O_i is the only known sequence to the observer, which is related to the hidden states and the matrix B.

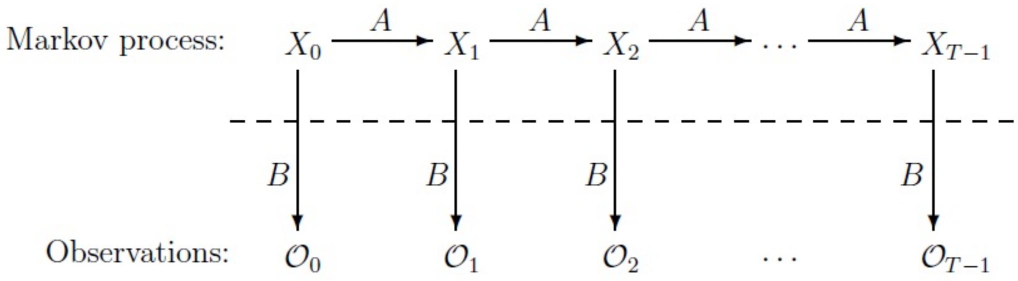


Figure 1: Hidden Markov Model

2.3.2 The Three Problems

HMM can be used to solve three types of problems [26].

Problem 1. Given the model λ = (A,B,π) and obesrvation sequence O, find P(O—λ). We need to score the observed sequence to see how well it fits into the model λ.

Problem 2. Given the model λ = (A,B,π) and obesrvation sequence O, find the optimal hidden state sequence. We need to find the hidden states of the given model.

Problem 3. Given an observation sequence O and number of states in model λ (N) and number of observable symbols (M), find the model λ = (A,Bπ) that maximizes the probability of O. We need to train a model for the observed data.

The GUI-based data we collected for this project from a large number of users is divided into training data and testing data. We choose the data from any one user (considered as good user) and divide it in two sets. One set of data is used as training data to train HMM (Problem 3). The second set of good user's data combined with the data from other users (considered as intrusionrs) is used as the testing data. The HMM is then used to score testing data (Problem 1). A higher score indicates similar characteristics between training data and testing data, and a lower score indicates distnction between the two types of data. Thus, we can seperate the intrusionr's data from the good user's data.

2.4 ROC Curves

Receiver operating characteristic (ROC) curve is a graphical plot that represents the trade off between true positive rate and false positive rate for every possible cut off. ROC curves ares used to show the performance of a binary classification problem where there are two types of outcomes, namely positive and negative [18]. To understand the concept of ROC curve better, let us first understand the possible outcomes of the binary classification problem.

2.4.1 Possible Outcomes

As discussed above, there are two possible outcomes of a binary classification problem, either positive (p) or negative (n). Based on these outputs and their predictions, there can be four combination of the outcomes of the problem, as described in Figure 2 [28].

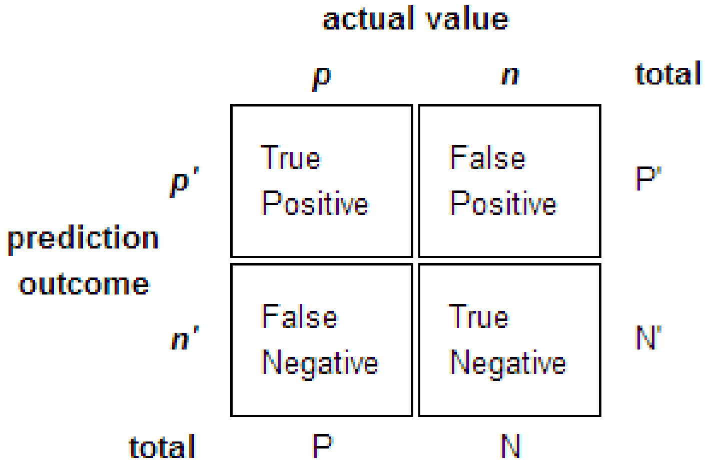
1. True positive (TP) - In case of both the prediction as well as the actual outcome being positive, the outcome is called true positive.

2. False positive (FP) - When the predicted outcome is positive and the actual outcome is negative, then it is said to be false positive.

3. True negative (TN) - When both the predicted and actual outcome is negative,

then it is a true negative outcome.

4. False negative (FN) - When the prediction outcome is negative, while the actual value is positive, then the outcome is false negative.



		actual value		
		p	*n*	total
prediction outcome	*p'*	True Positive	False Positive	P'
	n'	False Negative	True Negative	N'
	total	P	N	

Figure 2: Possible outcomes

This concept when applied to our project problem, we get two possible outcomes in the form of user file belonging to either good user or intruder. When a good user is correctly predicted as good user, then we get a true positive outcome. Similarly, with intruder correctly predicted as intruder, we get true negative. False positives and false negatives occur when any of the good user or intruder is misclassified as the other. These values are, however highly dependent on the threshold value we choose to distinguish between good user and intruder. Figure 3 shows how the results of the problem is distributed in typical cases. Here the x axis represents the threshold value, which is the deciding factor for the amount of overlap between different outcomes.

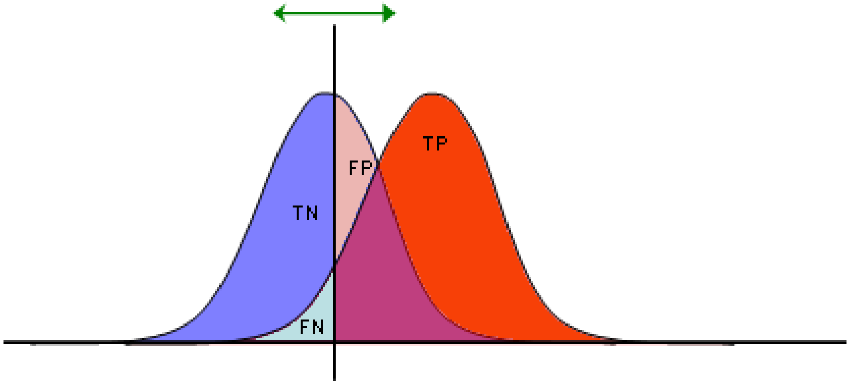


Figure 3: Distribution of results with the possible predictions

2.4.2 ROC Curve Analysis

The ROC curve is widely used for diagonostic test ealuation. It is plotted for true positive rate (TPR) also known as Sensitivity and false positive rate (FPR) also known as Specificity, for different cut-off points based on threshold values. Each point on the ROC curve corresponds to a particular decision threshold. Thus it gives a complete sensitivity/specificity report for a given problem [1, 28].

TPR represents the number of correct positive results among all positive outcomes available for the given problem. This represents the cases when a intruder is correctly identified as the intruder. Whereas FPR defines the number of incorrect positive outcomes among all negative outcomes available for the problem. FPR represent the cases when a good user is identified as a intruder. TPR and FPR are calculated as follows:

The ROC curve is plotted with FPR on x-axis and TPR or y-axis. A test with best prediction gives no false positives and no false negatives i.e. 100% sensitivity and 100% specificity. This type of test results in a ROC curve that passes through

$$TPR = \frac{TP}{TP+FN}$$

$$FPR = \frac{FP}{FP+TN}$$

Figure 4: TPR and FPR

the upper left corner (0,1) that represents a higher overall accuracy. On the other hand, a random guess results in a diagonal line (0.5,0.5) with 50% sensitivity and 50% specificity. All the tests that gives ROC curve above the diagonal line are considered to have good results and others point to poor results.

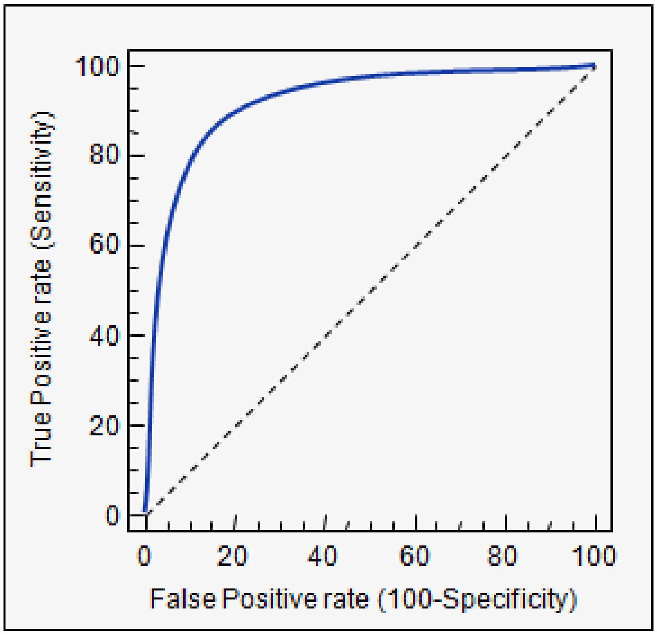


Figure 5: ROC curve

2.4.3 Area Under The Curve (AUC)

AUC is a measure of how well a parameter can distinguish between the two possible outcomes of the binary classification problem [1]. A perfect prediction of the results of the given problem gives us an AUC of 1.0.

CHAPTER 3

Event Logging for GUI-based Systems

Anomaly-based intrusion detection is generally based on the user interaction with the system. This interaction can be recorded in the form of list of commands issued by the system on terminal, or the skill level of the user or merely on his/her way of using a particular aspect of the system (for instance, how to use certain keystroke combinations or how to use mouse to open certain applications).

Intrusion detection based on command line data will generally involve typing certain commands with or without specific parameters for a certain purpose. However, intrusion detection for GUI-based systems for the same purpose may involve a series of different types of actions by different users. For instance, Table 2 gives a comparison of command line dataset and GUI-based dataset for two different users trying to find the contents of a directory in Windows system [9].

Thus, trying to detect intrusions based on user behavior on GUI-based systems can be more complex as compared from the command line based dataset. This is due to the fact that command line data gives us the information of what command is executed, whereas GUI-based behavior data gives us the information of how the command is executed. Similar to the architecture of anomaly-based intrusion detection based on command line data [13], Figure 6 depicts the basic architecture of anomaly-based intrusion detection based on GUI interactions of users.

The GUI features taken into account for collecting data in this paper includes the following higher level interaction of user with the system:

- Mouse interaction

Table 2: Comparison of Command line dataset and GUI-based dataset

User	Command line dataset	GUI-based dataset
1	dir or ls [with or without parameters]	1. Mouse Coordinates (movement) 2. Left Click (on Start menu) 3. Left Click (on Computer folder) 4. Click to open the folder
2	dir or ls [with or without parameters]	1. Press Windows key from keyboard 2. Press arrow keys to reach computer folder 3. Press Enter 4. Press arrow keys to reach the desired folder 5. Press Enter

- Keyboard interaction
- Active applications

3.1 Mouse Interaction

The real user data based on mouse interaction of user with the system consists of various types of lower level interaction. We log this interaction by observing mouse clicks (left and right) along with the time of event, mouse coordinates and the application on which event occurs. Table 3 shows a sample of mouse log file generated by the event logger. As we can see that double clicks are not listed as a seperate

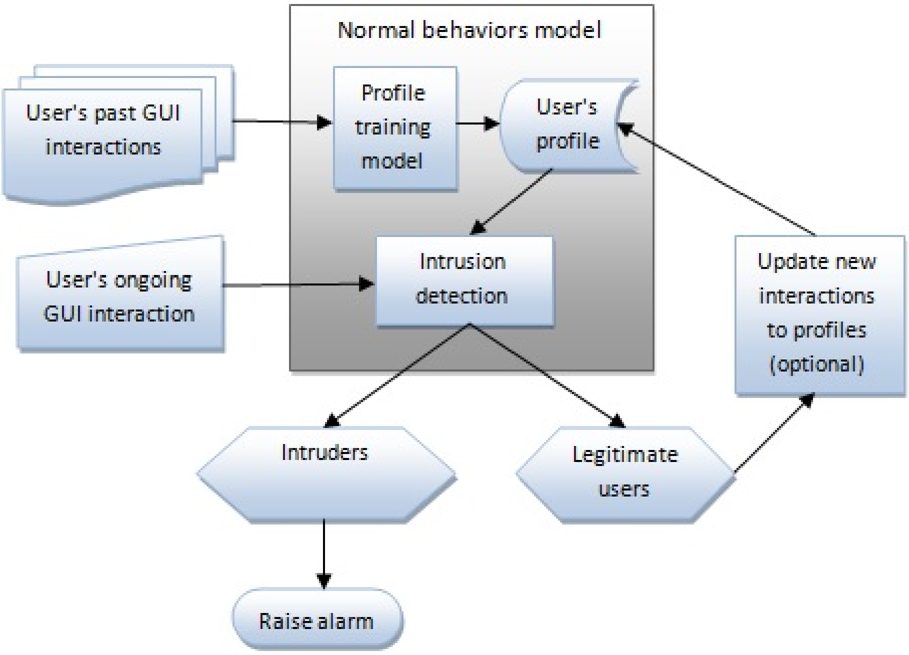


Figure 6: Architecture of anomaly-based intrusion detection based on GUI interactions

category in the table. However, we can observe double left click in the last two entries of the table which have same coordinates and same time stamps for the given application. This type of detailed data will allow us to extract useful information about user behavior in relation to mouse and system interaction.

3.2 Keyboard Interaction

Keyboard interaction in GUI-based sytems can give us a variety of behavioral information. We can not only extract information like typing speed by observing keyboard interaction, but can also get information about typing pattern of the user, the use of shortcut keys as a substitute for mouse and other information depending on the type of application in use and the skill level of the user. Table 4 shows a sample

Table 3: Sample dataset showing the Mouse Interaction

Mouse Click	Coordinates (X,Y)	Time of event	Application
Left	590,349	May 29 10:14:19 2012	file:///C:/Users/Documents/Visual Studio 2010/Projects/bin/Debug/LogActivePrograms.EXE
Right	1268,8	May 29 10:14:24 2012	LogActivePrograms (Running) - Microsoft Visual C# 2010 Express
Left	1026,87	May 29 10:14:56 2012	mouseLogs.txt - Notepad
Left	1026,87	May 29 10:14:56 2012	mouseLogs.txt - Notepad

dataset collected by the event logger. Like the dataset for mouse interaction, this dataset also contains information about the time of event and the current application in use. However, we collecting this dataset for every five seconds, assuming the current application remains same for those five seconds. The reason for adopting this method for data collection was to avoid logging keyboard activity for every event. We used a buffered string which stored the keys pressed for five seconds, and empty the buffer in log file at the end of the five seconds. This allowed an efficient use of resources by the logger tool. Moreover, we can also observe from the table that some keys are logged by the special character * instead of giving information about actual keys pressed. This allows the user to maintain his/her privacy. The special character represents alphabetical, numerical characters and space.

Table 4: Sample dataset showing the Keyboard Interaction

Time of event	Keys pressed	Application
May 29 10:09:29 2012	*****[SHIFT][TAB]***	Logs.txt - Notepad
May 29 10:09:34 2012	*[CTRL]**********	Google - Google Chrome
May 29 10:09:39 2012	[SHIFT]*****[ENTER][ENTER]	Google Maps - Google Chrome
May 29 10:09:44 2012	*****[SHIFT][SHIFT]***	TextPad - C:/Users/Documents/ report/ chap3.tex
May 29 10:09:49 2012	***[CAPSLOCK]***[F5]***	LogActivePrograms (Running) - Microsoft Visual C# 2010 Express

3.3 Active Applications

Getting information about only mouse and keyboard interaction in GUI-based systems will be incomplete, if we do not take into account the application the user may be interacting with while using mouse or keyboard. The dataset will be incomplete if we do not get information about the application on which the user might have clicked or for which the user may be using the keyboard for. The event logger tool not only logs the current application that the user is interacting with, but also logs all the applications that are active during the user session. This information is logged for every twenty seconds, so that it allows us to know the start and end duration of the applications. Table 5 shows a sample dataset collected by the event logger for logging the active applications.

Thus with the help of event logger tool we get the lower level information about user's interaction with the system. However, this information cannot be directly used for creating the dataset that will be trained for the intrusion detection techniques. This large amount of information need to be filtered to extract useful parameters of

Table 5: Sample dataset showing the Active Applications

Time of event	Current Application	Active Applications
May 29 10:09:29 2012	TextPad - C:/Users/Documents/ report/ chap.tex	Logs.txt - Notepad LogActivePrograms (Running) - Microsoft Visual C# 2010 Express LogActivePrograms.EXE TextPad - C:/Users/Documents/ report/chap.tex mouseLogger - Visual C++ 2008 Express Edition
May 29 10:09:49 2012	Google - Google Chrome	Google - Google Chrome LogActivePrograms (Running) - Microsoft Visual C# 2010 Express LogActivePrograms.EXE TextPad - C:/Users/Documents/ report/chap.tex intrusion detection using GUI behavior.pdf - Adobe Reader
May 29 10:10:09 2012	Google Maps - Google Chrome	LogActivePrograms (Running) - Microsoft Visual C# 2010 Express LogActivePrograms.EXE Google Maps - Google Chrome

the user behavior that will provide us with unique feature set for training and testing various detection techniques.

3.4 The Event Logger Tool

We have implemented an event logger tool for capturing user behavior actions based on mouse activity, keyboard activity and active applications that run on the system for each user session. The reason for selecting these behavioral actions is to create unique user profile based on the user's interaction with the GUI-based environment.

In order to capture different behavioral interactions, via mouse and keyboard interactions our tool needs to create certain lower level system hooks. Hooks are used to extend the functionality of applications by intercepting function calls, events or messages passed between software components. We created two system hooks for this tool, a mouse hook and a keyboard hook. These hooks allowed us to intercept mouse and keyboard event messages before they reach an application. More about the hooks can be found in the Appendix A of the report.

This tool makes use of multithreaded programming to call the dll files using separate threads. This allows our logger to log various activities simultaneously. These dll files contain the lower level hooks for the system. The logged events are observed along with their start and end time to allow us to get maximum data possible.

Apart from capturing various user behavior actions, it is also important for us to find a way of sending the large amount of captured user information from the user's machine to a server. Thus in order to facilitate this functionality, we implemented the method to compress the collected data in a zip format and send the compressed files to a common mail server at regular intervals.

This event logger is developed in C# language using Microsoft .Net framework on Windows operating system. Through the C# program we call dll files written in C++. These dll files contain keyboard and mouse hooks that are useful in logging keyboard and mouse activities. We chose to use .Net framework as our project was focussed on Windows operating system and .Net framework is easy to use with various Windows components. Moreover, the ability to call lower level dll files from .Net framework is very easy.

CHAPTER 4

HMM-Based Intrusion Detector

The dataset created from the collected GUI-based user interaction from various users is used for training HMM. However, in order to make the dataset from the collected data files, the data needs to be first cleaned to remove the garbage data and convert it in a format compatible with the HMM program.

4.1 Converting User Files To Dataset

For our project, we collect the user interactions with their systems in three different files based on the type of interaction. These files collect the information about user interactions until the user restarts the system or until the system reconnects to internet. Thus for each user session our mail server receives three separate files compressed in a single file. Chapter 3 shows the type of information that is included in each file. In addition to this information, the data files also contains a lot of garbage in the form of characters, missing attributes or encrypted data. Therefore, in order to use the data for training HMM we need to first clean the data. The steps involved for cleaning the data are:

- The files are read line-by-line to search for the occurrence of useful user information in each line.
- Extra characters, spaces and blank lines are ignored.
- Lines containing missing user information like timestamp or screen coordinates in case of mouse clicks or type of key pressed in case of keyboard interaction are ignored.

4.2 Training HMM With Dataset

Once we have the dataset ready for HMM training, we divide it into two sets: training set and testing set. For better results, it is always beneficial to have more training data as compared to the testing data [18]. The training data comprises of the data from good user and is used to generate models by HMM training. Multiple models are generated using different data files for scoring test files. The testing data comprises of the data from the good user that has not been used for training and data from other users, which are considered as intruders here. The testing data from the good user is used for the validation, to determine how good our model performs. Ideally, if we score the testing data then we expect to get high score for the data from the good user, and lower score for the data from the intruders.

4.3 Experimental Setup

For our experiment purpose, we have written the code for data cleaning and training HMM in Java. We collected data from around 150 users, and have received more than 50 files from each user. For training HMM, we download 50 files for the randomly chosen good user and give the raw user files to our code as input. 20 files of the chosen user is added to the training set and 30 files of the chosen user is added to the testing set. Random files of the remaining users are also added to the testing set.

For each file in the training set, a model file is generated in the model folder. Therefore, we get 20 model files for the chosen user. First line of each model file gives the details of the total number of observations (T), number of hidden states (N) and number of unique observations (M) for that file. Remaining part of the file gives the details of π matrix, A matrix and B matrix of the HMM.

Once the training phase completes, the testing files are scored and classified as good user file or bad user file. The scores are computed using the model files and are written in the result folder. We consider only those observations of the testing file that fall under the same time limit of the good user file. Each result file contains scores for all the files in the testing set. So we get 20 result files, one from each of the model file.

4.4 Training HMM With Mouse And Keyboard Data

The data files of users is first cleaned to filter the useful information. In case of mouse data information like types of mouse clicks (left or right) and x-y coordinates is filtered, whereas in case of keyboard data information about keys pressed is retrieved from the data files. For the mouse data, the screen is divided into 16 grids of size 16*16. The type of mouse click and the number of clicks in different grids is used as observation matrix while training the HMM, whereas, in case of keyboard data the set of keys pressed is used as observation matrix. The model files are then generated for different number of hidden states. This completes the training phase of HMM. For the testing phase, the model files are used for scoring the testing data.

CHAPTER 5

Results

While training Hidden Markov Models, we have considered different number of hidden states. Based on the scores of different users in the testing phase, we plotted graphs for all the users. Our methodology and dataset was sufficient to effectively separate the good users from the intruders with the help of the mouse logs. The log likelihood per obeservation for good users and intruders were separated to a significant extent for mouse data. However, the methodology was not that effective with keyboard data as we will see in the following sections. The expected result from HMM is to have lower scores for the good users and higher scores for bad users (i.e., intruders).

5.1 HMM Training With Mouse Logs

Let's see the results of training HMM with our mouse data for a good user. Figure 7 shows the results for a user with number of hidden states as 2. Whereas Figure 8 shows results with number of hidden states as 3. The x-axis on the graphs represent the user file number, whereas, the y-axis represent the log likelyhood score of the user file. The graphs shows the separation of good users and intruders to a significant extent, as the good users represented by blue dots are having lower scores as compared to intruders.

However, there is still misclassification of some intruders as good users and vice versa. When a intruder is misclassified as good user a false negative occurs, whereas when a good user is misclassified as a intruder a false positive occurs. Thus, from Figure 7 we can observe one false negatives and two false positive, if we take 200 as the threshold score; whereas, Figure 8 shows two false negatives and one false positive, if we take 150 as the threshold score.

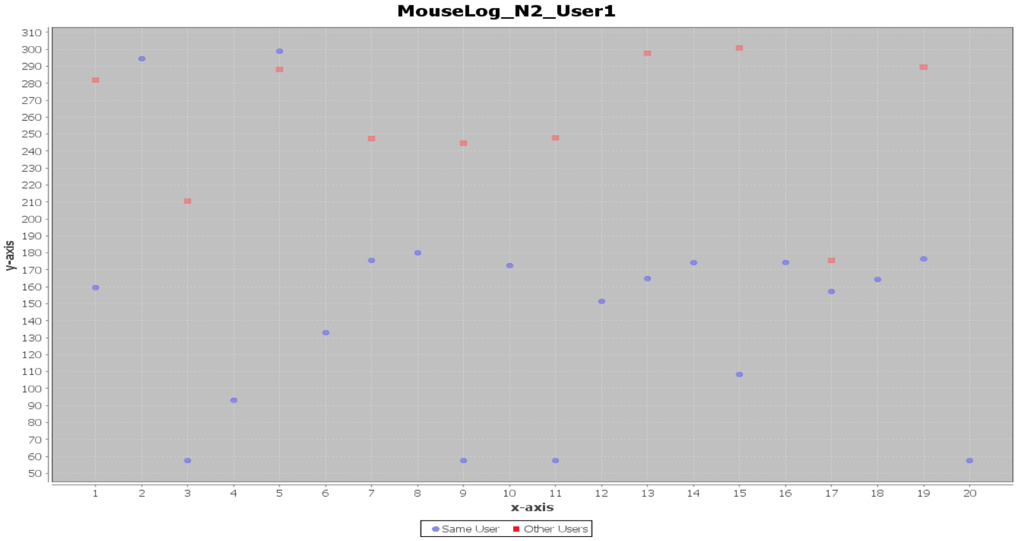


Figure 7: Log likelihood per observation of user 1 Vs intruders using mouse data, N = 2

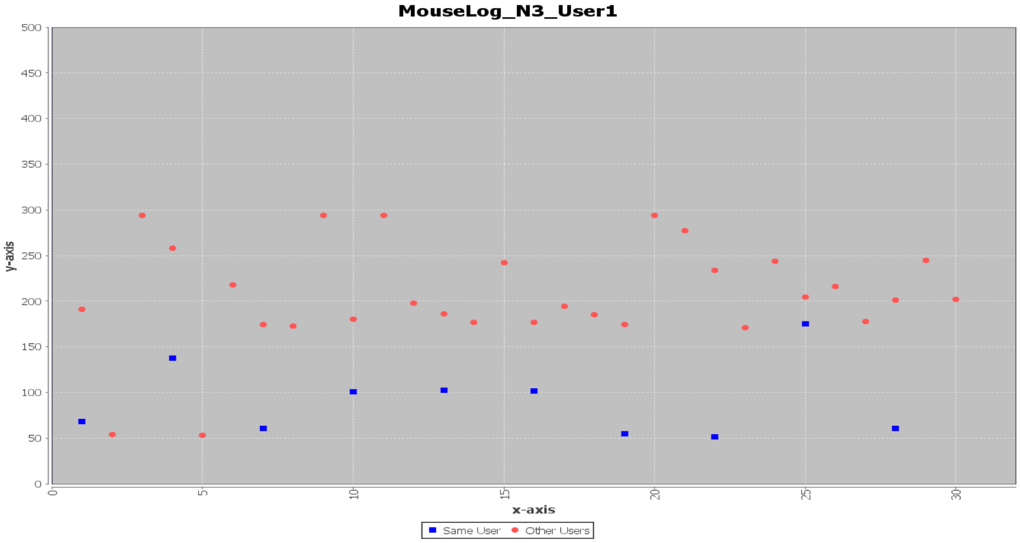


Figure 8: Log likelihood per observation of user 1 Vs intruders using mouse data, N = 3

We observe that by considering different threshold values, we can have different number of false positives and false negatives. Hidden Markov Model utilizes this concept and classifies the user as good user and bad user (intruder) by using different threshold values to compare the scores. Lower threshold values can result in more number of false positives, whereas higher threshold values will reduce the number of false positives but increase the number of false negatives.

Keeping in mind the importance of threshold value, we plotted the ROC curve by considering each good/bad user's scores as threshold value. Using each score as threshold value allows us to cover all the possible scenarios of getting false positives or false negatives.

Figure 9 depicts the ROC curve obtained for User 1's results with number of hidden states as 2, that was shown above in Figure 7. Area under the curve (AUC) for this ROC is 0.93, that represents 93% accuracy of the method used here to differentiate between good users and bad users. Whereas AUC for the ROC curve for the same user with number of hidden states as 3 comes out as 0.9567. This curve is shown in Figure 10.

On observing the results of HMM with different results, we find that the results vary for each user. Figure 11 and Figure 12 shows the scatter plots of the scores of user 18's files and intruder files with hidden states as 2 and 3 respectively. On plotting the ROC curves for these results (Figure 13 and Figure 14), we find that AUC for the scores with hidden state as 2 is 0.9, while for hidden state as 3, the AUC is 0.94. This variation in the results may be diagnosed as a result of the type of mouse usage by different users. Some users are reluctant in using mouse for every purpose such as copying and pasting images or text, they may be more accustomed to use keyboard shortcut keys for such purposes. While there may be some users who are frequent mouse users. Thus depending on various user behavioral parameters, the results of HMM may vary.

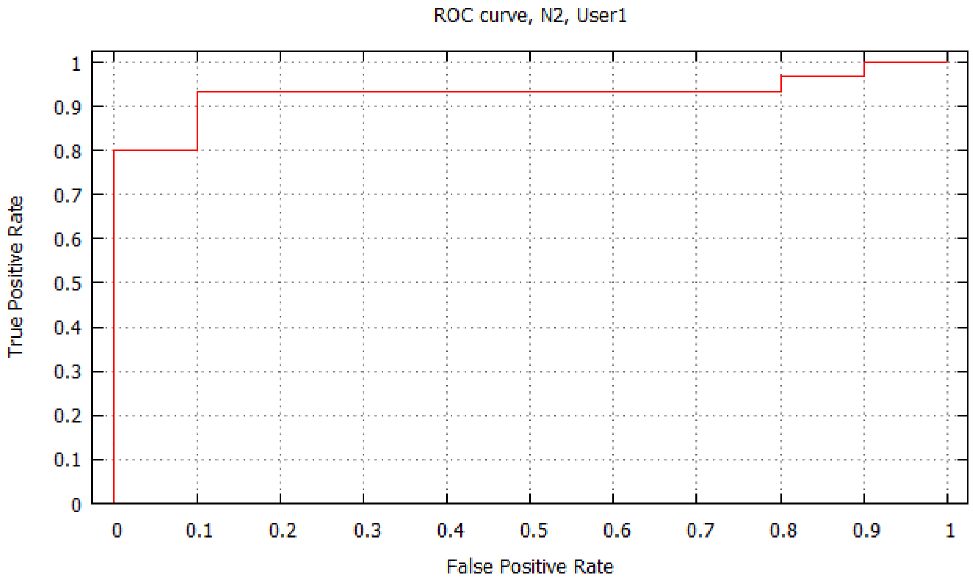


Figure 9: ROC curve for user 1, Mouse data, N = 2

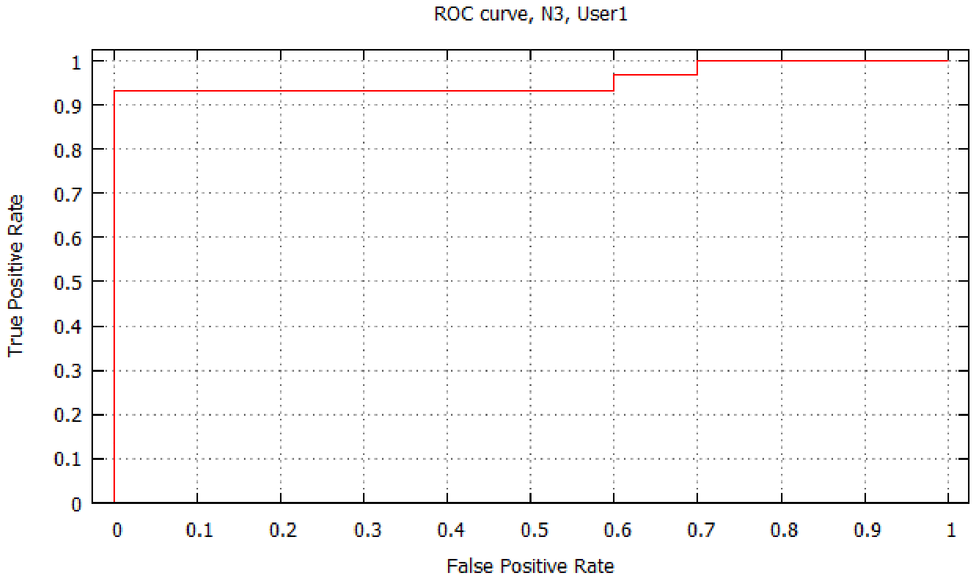


Figure 10: ROC curve for user 1, Mouse data, N = 3

By looking at the area under the ROC curve for various users with different number of hidden states, we can conclude that the HMM results are good for the mouse logs. We find that all the ROC curves fall in the upper half of the graph which

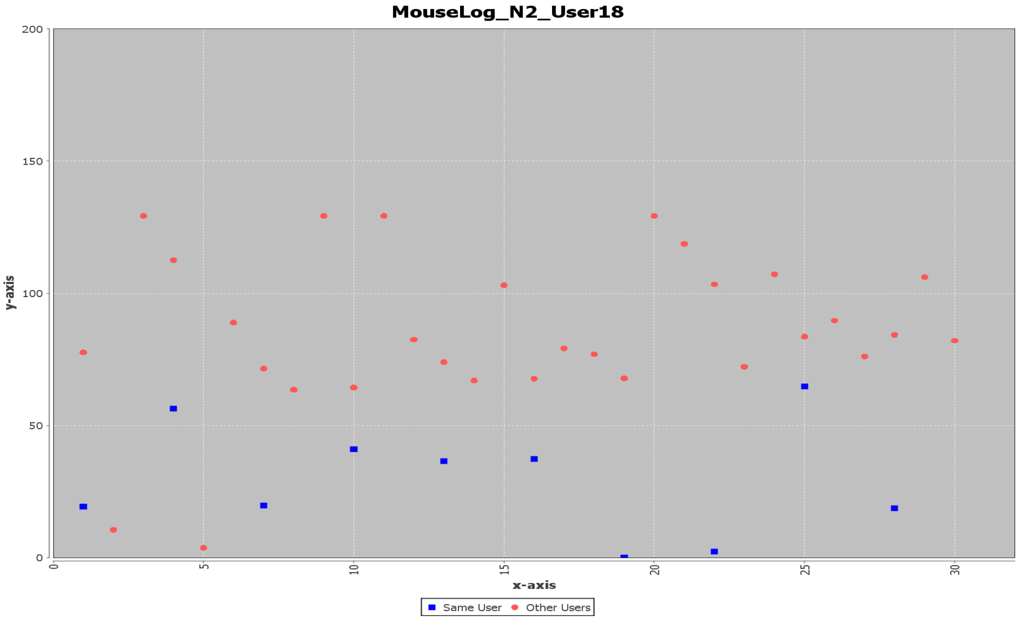


Figure 11: Log likelihood per observation of user 18 Vs intruders using mouse data, N = 2

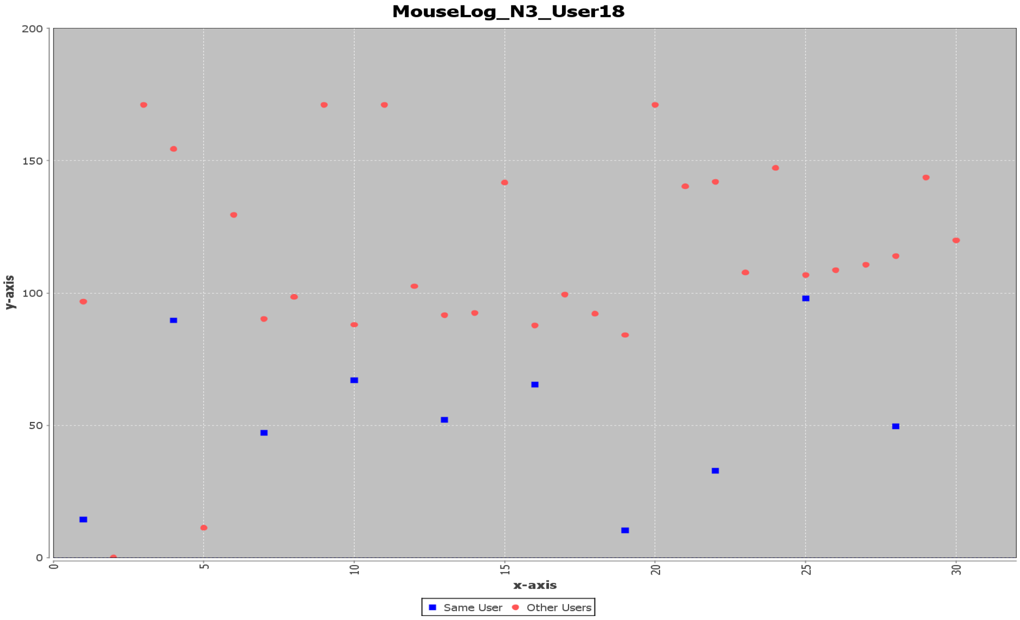


Figure 12: Log likelihood per observation of user 18 Vs intruders using mouse data, N = 3

shows higher accuracy rate of the methodology in separating the good users and bad users.

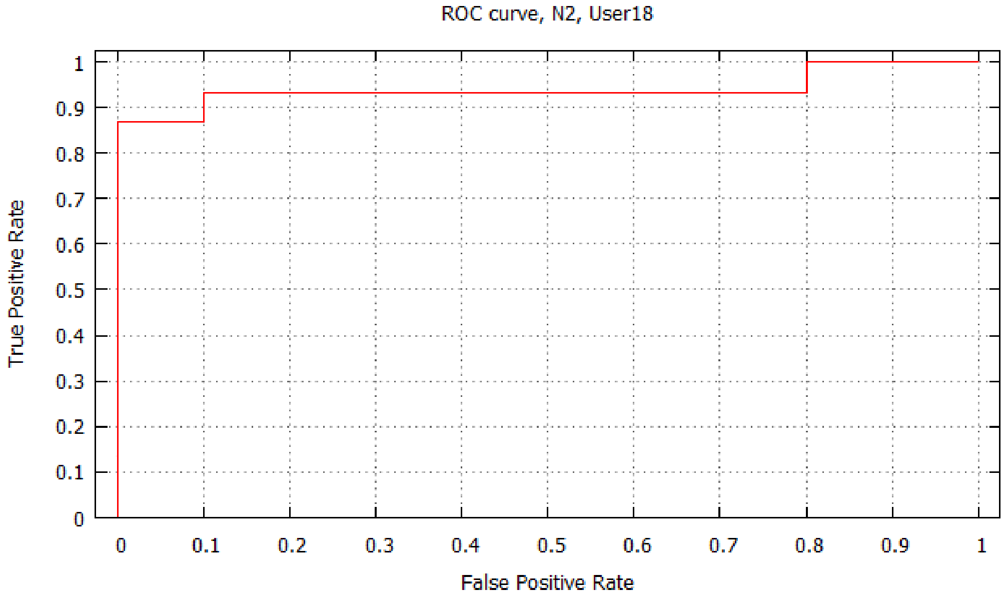


Figure 13: ROC curve for user 18, Mouse data, N = 2

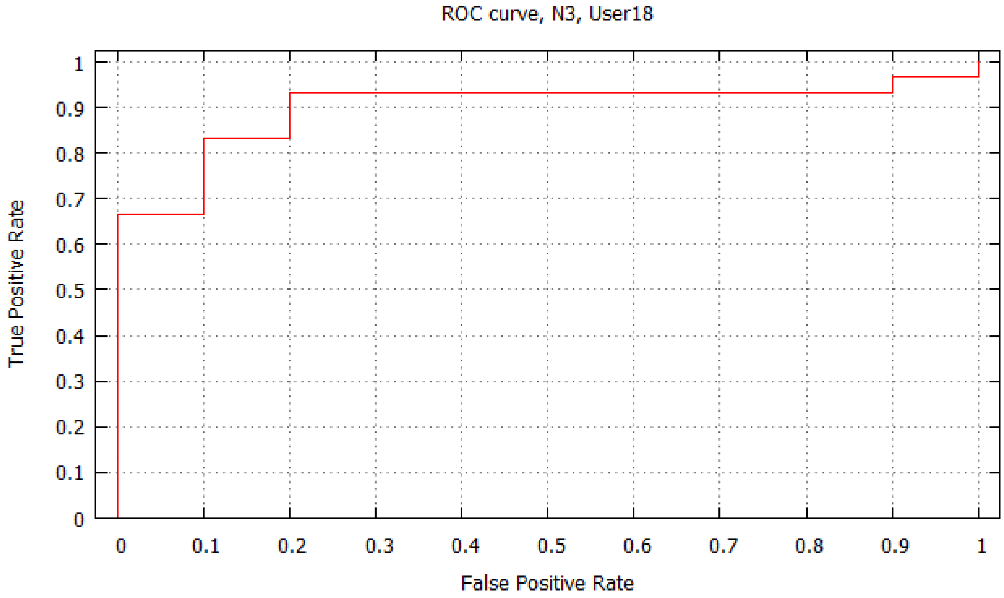


Figure 14: ROC curve for user 18, Mouse data, N = 3

5.2 HMM Training With Keyboard Logs

Similar methodology as with mouse logs, when applied to the keyboard logs proved to be inefficient in separating the good users and bad users. Figure 15 and

Figure 16 shows the scatterplot of the log likelihood per observation of good user vs intruders with hidden states 2 and 3 respectively. The graphs indicates that the good user and intruder's scores are mingled up to a great extent. Looking at the graphs it is difficult to identify the good user's files and intruder's files, as there is a significant amount of overlap here.

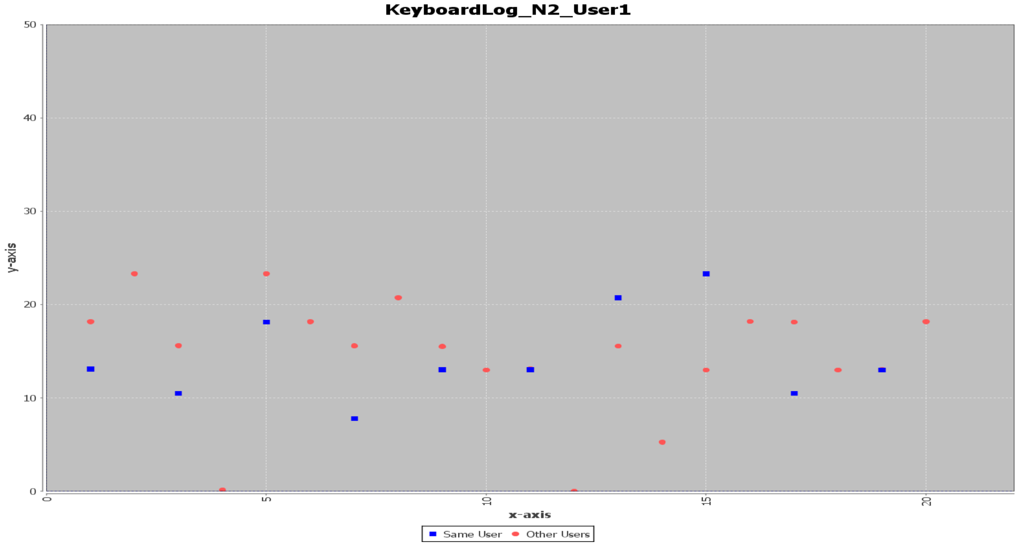


Figure 15: Log likelihood per observation of user 1 Vs intruders using keyboard data, N = 2

When ROC curves are plotted for these results, we get Figure 17 and Figure 18 for different hidden states. As can be guessed by looking at the curves, the AUC for these ROC curves is calculated as 0.5975 and 0.555 respectively. As this AUC value is very close to the random guess accuracy value (0.5), we can classify these results as poor results.

However, when we combine the results of mouse logs and keyboard logs by different factors, we found that the combined score with 80% contribution of mouse logs and 20% contribution of keyboard logs give us optimum results. Table 6 shows the results of combining mouse scores and keyboard scores (with hidden state as 3) by

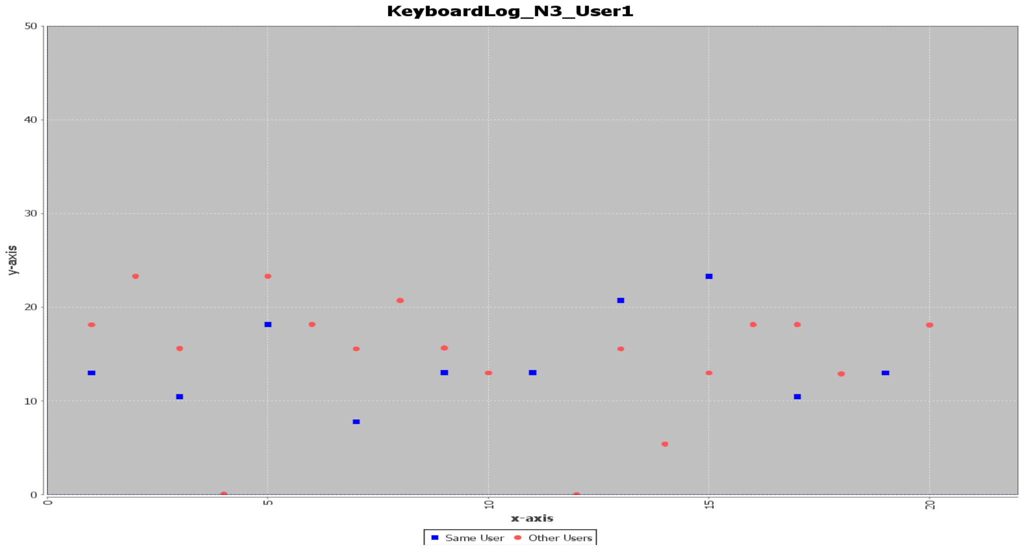


Figure 16: Log likelihood per observation of user 1 Vs intruders using keyboard data, N = 3

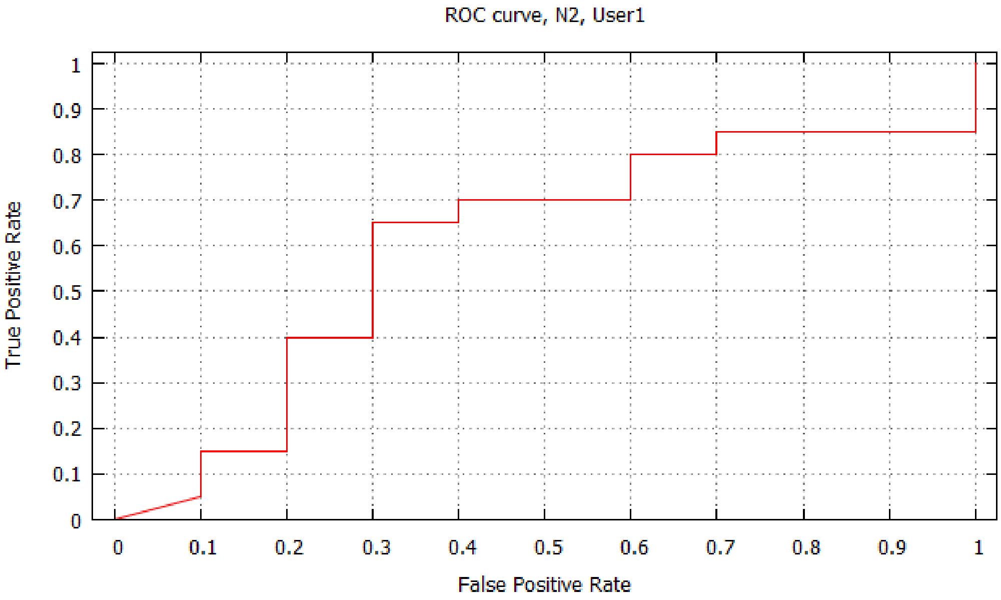


Figure 17: ROC curve for user 1, Keyboard data, N = 2

factors ranging from 0.0 to 1.0, with a difference of 0.1. The combinational formula applied here is as follows:

Combined score = (a*Mouse score) + (b*Keyboard score)

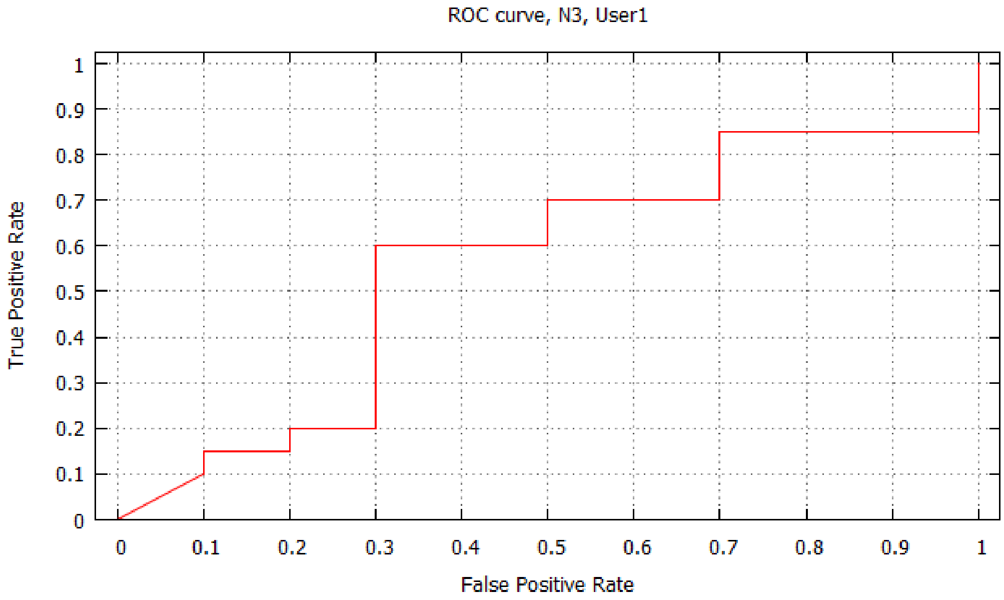


Figure 18: ROC curve for user 1, Keyboard data, N = 3

Table 6: Combining mouse and keyboard scores, N = 3

Mouse multiplier (a)	Keyboard multiplier (b)	Combined score
1.0	0.0	0.9385
0.9	0.1	0.9487
0.8	0.2	0.9538
0.7	0.3	0.9487
0.6	0.4	0.9487
0.5	0.5	0.9436
0.4	0.6	0.9410
0.3	0.7	0.9410
0.2	0.8	0.9410
0.1	0.9	0.9410
0.0	1.0	0.7487

CHAPTER 6

Conclusion and Future Work

Intrusion detection on GUI-based operating systems is a research area which is getting increasingly popular now with the introduction of more advanced GUI-based features in the new operating systems. Along with high popularity, this research topic is also an area which lacks a common platform for the researchers. After Schonalau dataset became publicly available, all the research done on intrusion detection based on Unix commands, had a common platform to showcase the effectiveness of their methodologies used in the research. However, this is not same with intrusion detection based on GUI commands. Our project was another step in bringing the research done in this area to another level.

In our project, we came up with an event logging tool to capture the GUI-based user data for Windows system. This user data was based on the mouse commands and keyboard commands issued by the users in a user session. This tool was distributed among various user groups of different demographics as well as residing in different geographical areas around the globe. We trained the user data collected with the machine learning technique, Hidden Markov Model (HMM). We trained both mouse data as well as keyboard data with this methodology and found that the mouse data was far more effective in intrusion detection, as compared to the keyboard data.

While results based on mouse data were good in separating good users from the bad users and have an average accuracy rate of 91%, the keyboard data gave us results equivalent to the random guess with an average of 53%. The reason behind this can be several. This can be due to the variation in the keyboard types used by the users or the due to the presence of similar frequent keys pressed by the users.

We also analyzed our experimental results with different number of hidden states in HMM and didn't notice any significant change in the results. The accuracy level of the experiments ran with different number of hidden states was found to be very close. With mouse logs the accuracy level increased slightly with higher number of hidden states, but the difference was no very significant.

Due to the time constraints and resource constraints, our project was mostly concentrated on the mouse commands and keyboard commands issued by the users. However, there are many other GUI features that can be taken in account for a better intrusion detection. Based on the fact that we managed to get good results with the limited data that we analysed, there can be much better results if future research capture other GUI features as well for modeling user behavior.

Our event logging tool also captured the application names that were used by the users during the user session. However, due to time constraints we could not study the effect of mouse/keyboard commands in conjunction with the applications in use. Future research work can definitely study the effect of application in use on the GUI commands issued by the user, and analyze the results.

As there are many more devices available that have GUI-based operating systems installed on them, for instance mobile phones, tablets, etc. This research can be extended to these devices. Moreover, intrusion detection is also an important area of research for banking and finance sector. Therefore, intrusion detection on fly can also help customers using online banking and payment modes to feel more secure.

LIST OF REFERENCES

[1] Ataman, K., and Zhang, Y. (2006); Learning to rank by maximizing AUC with linear programming; *International Joint Conference on Neural Networks*; 123–129

[2] Beauquier, J., and Hu, Y.J. (2007); Intrusion detection based on distance combination; *World Academy of Science, Engineering and Technology 2007*; 31, 172–180

[3] Bertacchini, M. and Fierens, P.L.(2007); Preliminary results on masquerader detection using compression based similarity metrics; *Electronic Journal of SADIO 2007*; 7(1)

[4] Bhukya, W.S., Kommuru, S.K., and Negi, A. (2007); Masquerade detection based upon GUI user profiling in linux systems; *Advances In Computer Science, ASIAN 2007*

[5] Claar, C.L., Couraud, J., Erbacher, R.F. and Prakash, S.; Intrusion detection: Detecting masquerade attacks using UNIX command lines; Utah State University

[6] Erbracher, R.F., Prakash, S., Claar, C.L. and Couraud, J.; Intrusion detection: Detecting masquerade attacks using UNIX command lines; usu.edu

[7] Devarakonda, N.R., Pamidi, S. and Kumari, V.V. (2011); ABIDS system using hidden markov model; *Information and Communication Technologies (WICT)*; 11(14), 319–324

[8] Feng, H., Kolesnikov, O., Fogla, P., Lee, W., and W. Gong (2003); Anomaly detection using call stack information; *Proceedings of IEEE Symposium on Security and Privacy (Oakland, California)*

[9] Garg, A., Rahalkar, R., Upadhyaya, S., and Kwiat, K. (2006); Profiling users in GUI based systems for masquerade detection; Information Assurance Workshop

[10] Garg, A., Vidyaraman, S., Upadhyaya, S., Kwiat, K. (2006); USim: a user behavior simulation framework for training and testing IDSes in GUI based systems; *Simulation Symposium, 2006. 39th Annual;*, 8, 2–6

[11] Goecks, J., and Shavlik, J. (1999); Automatically labeling web pages based on normal user actions; IJCAI Workshop on Machine Learning for Information Filtering

[12] Hashia, S., Pollett, C., and Stamp, M. (2004); On using mouse movements as a biometric; San Jose State University; `http://www.cs.sjsu.edu/faculty/pollett/masters/Semesters/Spring04/Shivani/shivanipaper.pdf`

[13] Huang, L., and Stamp, M. (2011); Masquerade detection using profile hidden markov models; *Computers and Security*; 30(8), 732–747

[14] Imsand, E.S., Garrett, D., and Hamilton, J.A. (2009); User identification using GUI manipulation patterns and artificial neural networks; *Computational Intelligence in Cyber Security, 2009*

[15] Peacock, A., Ke, X., and Wilkerson, M. (2004); Typing patterns: A key to user identification; *IEEE Security & Privacy*

[16] Kazi, S., and Stamp, M. (to appear); Hidden Markov Models for Software Piracy Detection; *Information Security Journal: A Global Perspective*

[17] Kothari, A. (2012); Defeating Masquerade Detection; *Master's projects*; 239; `http://scholarworks.sjsu.edu/etd_projects/239`

[18] Mahajan, A. (2012); Masquerade detection based on UNIX commands; *Master's projects*; 273; `http://scholarworks.sjsu.edu/etd_projects/273`

[19] Monrose, F., and Rubin, A. (1997); Authentication via keystroke dynamics; *ACM Conference on Computer and Communications Security;* 48–56

[20] Mungale, M. (2011); Robust watermarking using hidden markov models; *Master's projects*; 179; `http://scholarworks.sjsu.edu/etd_projects/179`

[21] Perlroth, N. (2012); Article on outmaneuvered at their own game, antivirus makers struggle to adapt; New York times; `http://www.nytimes.com/2013/01/01/technology/antivirus-makers-work-on-software-to-catch-malware-more-effectively.html?ref=technology&_r=1&`

[22] Pusara, M., and Brodley, C.E. (2004); User re-authentication via mouse movements; `http://www.csis.pace.edu/~ctappert/it691-11fall/projects/mouse-pusara.pdf`

[23] Schonlau, M. (1998); Masquerading user data; `http://www.shonlau.net/intrusion.html`

[24] Schonlau, M., DuMouchel, W., Ju, W.H., and Karr, A.F., Theus, M., and Vardi, Y. (2001); Computer intrusion: detecting masquerades; *Statistical Science*; 15(1)

[25] Shavlik, J., Shavlik, M., and Fahland, M. (2001); Evaluating software sensors for actively profiling Windows 2000 computer users; Fourth International Symposium on Recent Advances in Intrusion Detection

[26] Stamp, M. (February 2012); A Revealing Introduction to Hidden Markov Models; Department of Computer Science, San Jose State University, San Jose, CA; `http://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf`

[27] Upadhyaya, S., and Kwait, K.A. (2006); A comprehensive reasoning framework for information survivability (User intent encapsulation and reasoning about intrusion; implementation and performance); State University of NY at Buffalo

[28] Heagerty, P.J., abd Zheng, Y. (2005); Survival model predictive accuracy and ROC curves; *Biometrics*; 61, 92–105

[29] Windows dev center MSDN library; Hooks — overview and using hooks, `http://msdn.microsoft.com/en-us/library/windows/desktop/ms632589(v=vs.85).aspx`

[30] Wong, W. (2006); Analysis and detection of metamorphic computer viruses; *Master's projects*; 153; `http://scholarworks.sjsu.edu/etd_projects/153`

APPENDIX

Hooks

A.1 Introduction

Hooks are the points in the system message-handling mechanism where applications intercepts the function calls, events or message traffic by installing some subroutine to monitor these messages and process them before they reach their target software components. This hooking mechanism can be used to alter, augment or simply monitor the behavior of operating system, applications or software components. These hools can be inserted at runtime provided the operating system grants the process pemission to do so. Operating systems like Windows and Linux allow certain hooks to process or modify system events. For instance, our tool uses keyboard and mouse hooks available for Windows operating system. Similarly, Linux also provide hooks to process network events within its kernel through NetFilter.

Although hooks are helpful in debugging and extending functionality. These have their own downfalls as well. Due to the increase in processing time for each message, hooks tend to slow down the system. Moreover, hooks are also used by malicious code where certain outputs of the API calls are faked with the help of hooks.

A.2 Hook Procedures

Hook Procedures are the functions that intercept a particular type of event. These are installed by calling **SetWindowsHookEx** function, which specifies the type of hook, whether the procedure should be associated with all threads in the same desktop as the calling thread or with a particular thread, and a pointer to the procedure entry point. We used **WH_MOUSE_LL** and **WH_KEYBOARD_LL** hooks to install hook procedures that monitors low-level mouse and keyboard input events respectively.

Once the hooks are installed and the event that is monitored occurs, the hook procedure writes the information about the event to the client area of the application's main window. For instance, after installing the low level mouse hook, every time a new mouse input event occurs the system calls **LowLevelMouseProc** callback function. The type of mouse message, via left button click or mouse movement or right button click, is identified by the parameter WPARAM of the callback function.

After processing the messages, it is important to release the hook procedure by calling **UnhookWindowsHookEx** function. This allows the system's processing time to decrease and manages the resources efficiently.