User Identification Using GUI Manipulation Patterns and Artificial Neural Networks

Eric S Imsand, Deon Garrett, John A. Hamilton, Jr., *Member, IEEE*

***Abstract*— A masquerade attack is any attack in which the attacker is able to make the target system believe they are someone they are not. One particularly dangerous example of a masquerade attack occurs when an attacker uses an unattended, unlocked computer workstation. Recent studies have demonstrated that analyzing the manners in which a user manipulates a graphical user interface can be used as an effective means of authentication. We present the results of a study into the use of GUI manipulations in combination with Artificial Neural Networks (ANNs) as a basis for identification.**

I. INTRODUCTION

During a masquerade attack, the attacker attempts to make the target system believe that they are someone else. The simplest, and arguably most dangerous, form of masquerade attack occurs when an attacker uses a computer workstation that has been left unlocked and unattended. The result of this attack is that the attacker gains access to all resources that the rightful user has access to. The most troubling aspect of this attack is that it requires no technical expertise to carry out.

In order to combat this form of attack new technologies are needed. Previous attempts at masquerade detection have, among other things, focused either on command line input [9], [12] or on monitoring process table status [8]. While both of these techniques have been proven effective on previous generations of computer system they are less applicable to detecting masqueraders using modern GUI driven systems. They also fail to protect systems when the attacker is behaving in a manner consistent with the behavior of the victim. To provide further illustration, consider a nosy subordinate accessing their supervisor's system while he or she is away from their desk. Previous masquerade detection techniques have relied on the subordinate to behave in a manner that is inconsistent with their supervisor's observed behavior. If the subordinate's goal is to cause damage to their superior's system or the network at large, current masquerade techniques will be quite effective in foiling their attack. But, if the subordinate is simply after information, then they will behave in a manner that is quite similar to their supervisor – he or she might forge e-mail messages, modify sensitive files, or simply read privileged information. Unfortunately all of these tasks are relatively ordinary, and as a result, the attacker will not be caught using traditional masquerade detection means.

Instead of schemes that focus on *what* each user typically does, systems are needed that instead examine *how* the user typically accomplishes their work. Recent work has resulted in an approach known as "GUI-based Intrusion Detection" (GUIID). GUIID examines how a user is manipulating a graphical user interface and compares the observed behavior to a previously generated profile. Recent research has shown the viability of monitoring GUI interactions as a basis for user authentication. The next step in the evolution of GUIID is an evaluation of its effectiveness as a means of identification.

The remainder of this paper is organized as follows: section two discusses related work and section three describes provides a more through explanation of masquerade detection through GUI-based Intrusion Detection ("GUIID"). Section four describes the experiment conducted in this study. Section five describes analytical methods used in this study and the results that were obtained, while section six details our conclusions.

One final note: the experiment presented here was performed on the Microsoft Windows operating system using computer users that primarily used Windows. We do, however, believe that the results seen here would also be observed using other WIMP (Windows, Icons, Menus, and Pointers) based interfaces, provided such a pool of users were assembled.

II. RELATED WORK

Pursuit of preventative measures to thwart masquerade attacks is not a new endeavor. Commonly included with other work focusing on the broader issue of intrusion detection, research on masquerade attacks dates back to at least 1986, if not earlier. Unfortunately much of this research

E Imsand was a PhD student in the Department of Computer Science and Software Engineering at Auburn University. He is now with the Department of Computer Science, University of Memphis, Memphis TN 38152 USA (e-mail: eimsand@memphis.edu).

Deon Garrett is with the Department of Computer Science, University of Memphis, Memphis, Memphis, TN 38152, 38152, USA (email: deong@acm.org).

John A. Hamilton, Jr. is an Associate Professor of Computer Science and Software Engineering at Auburn University, Auburn, AL 36849 (e-mail: hamilton@auburn.edu).

has focused on techniques that do not apply well to computer systems commonly found in organizations today.

In [3], Denning outlines a framework for a real-time intrusion detection system. In this system she attempts to thwart masquerade attacks by monitoring the behavior exhibited by that user. More specifically the focus was on the commands issued by the user in comparison to a previously stored profile of how that user typically behaves. This idea of comparing user issued commands to a previously stored profile of presumably normal behavior has been used in further studies with good success [12], [9], [2].

These approaches have been shown to have promise and provide some defense against masquerade attacks. The problem with this research is that it does not apply particularly well to the modern computing environment found in most organizations. The data analyzed in the aforementioned studies utilized commands issued at a shell prompt using the computer keyboard. This environment is drastically different than the graphically driven computing environments such as Microsoft Windows found in most operating systems.

In an effort to address this gap in the research, Li [8] led a study using status information from systems using the Microsoft Windows NT operating system. This information included items such as the current process table and a list of windows that were open, as well as the titles of those windows. Li was able to achieve good results using this information as a means of distinguishing attackers from real users.

Recently work in masquerade detection has shifted from a focus of *what* the individual users are doing to *how* they are doing it. This approach has the benefit of being able to catch attackers when they are behaving in a manner consistent with the real users. Original attempts at this type of system dealt with the cadence and timing of users' typing patterns [13]. Known as keystroke dynamics, this type of system once showed promise, though it has only been deployed in limited cases as a real world defense system. More recently Pusara and Brodley presented the idea that users may use the mouse in unique manners [11]. Pusara and Brodley present compelling evidence that the manner in which users use the mouse may also be unique and suitable as a means of masquerade detection. Further study has confirmed the findings of Pusara and Brodley [1], [5]. Unfortunately Pusara and Brodley's work focused only on use of the mouse while viewing web pages in a browser, an activity which is commonly associated with heavy mouse usage. It is not clear whether their work would be applicable to the work performed in, for instance, a word processor. Furthermore, Pusara and Brodley acknowledge that, depending on the implementation, their masquerade detection system may not function well if the user does not use the mouse much or at all. The work presented in [1] and [5] also suffers from the same drawback – a user that does not use the mouse as a means of interaction may be able to evade detection.

Recent studies have examined the viability of differentiating between users based on their manipulation of the graphical user interface (GUI). These results, initially reported in [6] and further refined in [7] show tremendous potential. In [7] the authors were able to use manipulation patterns of graphical user interfaces, in combination with a data mining measure known as the Jaccard index, to successfully authenticate users with a high degree of accuracy. More specifically, their results found that simulated masquerade attacks could be detected in almost 94% of cases, with a false positive rate of 0%.

III. Overview of GUI-Based Intrusion Detection (GUIID)

The abstract concept behind GUIID is sometimes difficult to understand. To illustrate, consider how a user might accomplish the common task of copying text from one application window to another. There are many different execution paths a particular user might choose. For example, one user might select the text to be copied by dragging the mouse over the desired text while the left button was depressed. Another user might use the keyboard, depressing the shift key while using the arrow keys to select the desired text. The next step is for the user to "copy" the selected text to the clipboard. Again, there are multiple ways in which this can be accomplished. Some users might choose the "Copy" command under the "Edit" menu. Other users might choose to right-click on the selected text and select "Copy" from the pop-up menu displayed. Still other users might choose to use keyboard shortcuts and strike the "C" key while depressing the "Ctrl" key. Other users might use differing keyboard shortcuts (Alt + "E"dit + "C"opy). After copying the text to the clipboard the user must then switch to the desired window that the text will be pasted into. Again, (s)he is capable of using either the mouse or keyboard shortcuts (Alt+Tab). When it comes to pasting the text into the desired window the user has as many options as (s)he did when copying the selected text.

The previous task simply illustrates one example of a common task that different people may complete in different ways. Other examples include printing, saving, opening, closing, etc. When examining all of the common tasks performed by users when using a computer system our findings show that it is possible to build a profile of user behavior and to use that profile to detect potential masquerade attacks.

Individuals evaluating GUIID will undoubtedly find similarities between the usage patterns between any two individual users. For instance, two users may use the same procedure for printing a document, perhaps clicking on the File->Print menu. Our results, which are presented in section 5, show that the number of similarities between the actions performed by any two users is greatly outnumbered by the number of differences in those two users' actions.

IV. Experimental Setup

To test the hypothesis that GUIID could be used as a means of identifying the current user operating a computer system, an experimental study was conducted. The rough structure of this study was to provide participants with a set list of activities, have them complete these activities while being monitored using custom designed monitoring software, wait some period of time, then have them repeat the process.

A. Participant Selection

When constructing this experiment, one of the goals undertaken was to avoid collecting an experimental sample population that was overly skewed towards some personal trait that the users might have possessed. For example, a sample composed primarily of undergraduate students would not be indicative of the general population, making it undesirable for our purposes.

For this reason, the requirements for participating in the study were relatively minimal. The only requirements were that a participant had to be over eighteen years of age and have experience operating a Microsoft Windows workstation. No further requirements, such as the degree of computer proficiency, were placed on the participants.

To further ensure that the sample gathered more closely resembled the actual population, participants were solicited from two different sources. The first source was a high school, as it was believed that the wide variety of intellectual and professional experience exhibited by the faculty and staff would make an ideal recruitment source. The second location that participants were recruited from was a local engineering and assembly firm. This second organization was chosen for the same reason: the staff represented many different and varied professions (accountants, engineers, marketing/sales, executives, technicians, etc.) from a wide variety of backgrounds.

B. Task Selection

The users were given a single list of activities in an effort to ensure that the user's actions themselves were the only variable in the experiment. Had the users' computer activities not been regulated, there would have been two variables in the study: the user's actions AND the programs that the user had used. Furthermore, researchers have demonstrated that users can be differentiated based upon the programs that they are running [7].

Finally, it is worth remembering that an attacker may simply desire to learn confidential information. In these instances it is quite likely that a masquerader would use the same applications as the legitimate user. Moreover, the masquerading attacker would most likely be utilizing these applications with identical goals of the legitimate user: reading files, writing e-mails, etc.

The actual tasks that the participants were asked to complete were designed to be relatively common tasks that most computer users would know how to do. These tasks included word processing, web browsing and searching, and file/folder manipulation. Participants that were not able to complete the list of assigned tasks were not considered in the final analysis.

C. Testing Environment and Administration

Participants were provided computer workstations from a pool of workstations that were loaded with identical computer software. The workstations were configured as follows:

- Microsoft Windows XP Professional Service Pack 2
- Microsoft Office 2003
- Microsoft Internet Explorer 6.0

As previously alluded to, participants were asked to complete the task list and then asked to wait 30 minutes before completing the next session. During the thirty minute break the participants were allowed to return to their normal professional duties. The goal of these measures was to ensure that participants would not recall *how* they completed the tasks on the previous run, even if they did recall *what* they did.

Participants were asked to complete the experiment a total of five times. Of the five sessions that were gathered, only the final three sessions were analyzed. This was done for two reasons. First, the initial sessions may not truly reflect a user's normal behavior because of the change in environment (different computer, different desktop, shortcuts, etc.). Second, the composition of the final three sessions for each user was found to be consistent. On average, the variance between the final three sessions for any user was found to be just 0.03%.

D. Data Collection

As previously mentioned, custom monitoring software had to be developed for this study. It was determined early on that all efforts should be made to ensure that the data collected from the participants was not contaminated by actions performed by an administrator of the experiment. The only way to avoid this was to ensure that, once the monitoring software was active, only the participant interacted with the computer. This meant, though, that the participant would have to activate and deactivate the monitoring software.

Existing monitoring packages were consulted and either did not capture sufficiently detailed information about the user's actions or possessed interfaces that were deemed to be too complex for the average computer user. One example of a package that was considered, but ultimately rejected, was the *Spy++* utility published by Microsoft Corporation. While capable of capturing and logging all of the desired data, its user interface is quite complex, particularly for non-technical users. Numerous open source monitoring packages suffered from the opposite problem: not gathering sufficiently detailed information [4]. A package like this might tell you that the user clicked inside of the

"IExplore.exe" window, but not which button or menu (s)he clicked on.

The lack of a suitable, existing monitoring suite necessitated the development of a utility that was capable of capturing the detailed information needed for our analysis, while still presenting a relatively simple interface for the participants. The interface of our monitoring software was designed to be exceedingly simple, featuring only a single button that changed from "Record" to "Stop" as appropriate.

The monitoring software used APIs exposed in the Microsoft Windows operating system to inject a DLL library file into the process space of each process running on the system [10], [9]. From there the library intercepted event messages as they were being delivered from the operating systems to the client applications. The software recorded the following attributes of each message before transmitting the event message onto the client application: Process name (name of the application handling the event), type of event (left mouse click, 'a' key down, etc.), and the class of the control receiving the event (text box, button, dropdown menu, etc.). It is worth noting that the time of each event was not calculated. This was purposefully done to ensure that any sluggishness that might have resulted from OS services (running in the background) was not a variable in the study.

The attributes that were collected allow someone to more finely differentiate between actions committed by a user. For example, these pieces of data allow us to more precisely differentiate between a user that prints a document by clicking on the "print" button on the toolbar and a user that prints a document by using the "Print" command located beneath the *File* menu. We found this level of precision to be necessary when attempting to differentiate between two users.

V. Analysis

Data was gathered in the manner described in section 4 from a total of 31 participants, resulting in a total of 93 sessions being collected (three sessions from 31 participants). Every possible combination of known and unknown user, using every possible combination of sessions to generate the reference profile was tried. This resulted in a total of 8,711 simulated trials. Using an artificial neural network, authentication was attempted, then identification.

Several challenges were encountered in the adaptation of neural networks to the data collected in this study. For example, neural networks are typically designed to accept multiple numerical values in parallel as inputs. This posed an implementation challenge in this instance, since the data gathered in this study was more analogous to a corpus of text. Furthermore, when neural networks are used to analyze text documents, the input is frequently categorized by word frequency (i.e. how often each word occurs in the document). Often the number of distinct words in a document is a relatively small number that the neural network can cope with. Applying this model to the data gathered in this study, each event (action, control, executable) becomes analogous to a word in a text document. Unfortunately, this precise implementation method was not practical for use in this study. In this study, tens of thousands of distinct events were observed, translating to tens of thousands of "words" and resulting in the neural network needing to accommodate the same number of parallel inputs.

The workaround crafted for this problem still relied on a count of the number of times some object was involved in an event. Instead of considering an event as an atomic unit, though, events were instead broken up into their smaller components. This yielded a neural network structure that was given the following inputs:

- # of times a user action (left click, double click, 'A' press, etc.) was observed
- # of times a particular type of control (scroll_bar, text_box, etc.) was observed
- # of times a particular process (Winword.exe, Iexplore.exe, etc.) was observed

The number of distinct inputs being passed to the neural network was still quite large (over one thousand) but the network seemed capable of coping with this volume of input sources.

The neural network implementation used in this study was the Artificial Neural Network Toolbox (ANN) distributed as part of the Matlab numerical analysis suite (Mathworks Corporation 2007). The Neural Network Toolbox was relatively robust and easy to use. Furthermore, it is believed that the use of pre-written, commercially available analysis packages prevented coding errors that may have occurred in the implementation of a relatively complex algorithm such as back-propagation neural networks.

Other than specifying the number of hidden processing layers contained in the neural network, the recommended default settings provided by the Matlab developers were used in this study. The neural network that was developed contained one hidden processing layer of 45 neurons. The number of neurons used in the final experiment was derived through experimentation. Using more than 45 neurons resulted in slightly enhanced accuracy at the cost of much longer processing times. The use of fewer than 45 neurons resulted in significantly degraded accuracy with only modest processing time savings. The neural network featured N outputs, with one output corresponding to each participant in the sample.

The network was constructed by randomly selecting two data sessions from each user for use as training data. The remaining data session was set aside to be used as test data. The neural network was trained for five hundred epochs. This number was arbitrarily selected, empirical performance data did not indicate any impact of this selection. Training of the network was aborted by the Matlab package prior to reaching the training limit when the performance of the network hit a proverbial ceiling (i.e. the network was trained to its fullest possible extent).

VI. Results

Using the neural network, only 12 of 31 participants were correctly identified, yielding a successful identification rate of 38.7%. Undoubtedly, the small amount of available training data played a large role in the poor identification rate. In addition, the large number of inputs to the network (3185) compared to the number of examples available for the training (62) makes overfitting almost inevitable. The extent to which this performance could be improved through a larger study yielding more training examples is still an open question. However, the identification rate is far above that expected from random chance, and thus provides some rudimentary support for the notion that machine learning algorithms could achieve some success in identifcation based on this type of GUI user interaction.

The results from the experiment were further analyzed to see how many times the ANN would have to attempt identification before arriving at the right answer. The results of this analysis are shown in Table 1. The ANN required eight attempts at identification before correctly identifying the right user, on average.

Table 1 - Number of "Guesses" Before Correctly Identifying User

User ID	Number of Guesses
1	3
2	4
3	1
4	30
5	16
6	1
7	8
8	12
9	21
10	1
11	21
12	6
13	7
14	1
15	1
16	4
17	1
18	9
19	1
20	11
21	3
22	1
23	1
24	11
25	1
26	15
27	11
28	11
29	1
30	1
31	24

VII. Conclusions

This paper presents the results of an initial study of using GUIID in an effort to detect simulated masquerade attacks. The results of this study show that GUIID can be used to detect between 90% and 97% of simulated masquerade attacks in a controlled setting. The results presented here demonstrate the feasibility of developing real-world detection systems based on these techniques.

Authentication systems can traditionally be thought of as producing two possible outputs: either the current user is authenticated, indicating that the proper user is the one operating the computer, or the current user is not authenticated, indicating an attack is occurring. Identification, on the other hand, is a much more difficult function to construct. Whereas authentication functions returned one of two values (TRUE, FALSE), identification functions will return one of *N* values, where *N* is the number of users known to the system. So, while the results obtained here are poor, they do show the need for continued research into the application of GUIID as a means of identification. The success rate of 38% is far higher than the rate that would be observed by simple random guessing, indicating that higher success rates might be achieved by using alternative machine learning algorithms.

It is believed that the neural network used in this study performed poorly due to overfitting, due to the extreme dearth of training data. Though there is little direct evidence to support this explanation over others, it is well known that neural networks generally require much more training data than was available here to adequately learn complex concepts. Future experiments are being planned which will make use of the Jaccard Index given its high success rate reported in [7].

References

[1] Ahmed, A., Traore, I. 2007. *A New Biometric Technology Based on Mouse Dynamics*. IEEE Transactions on Dependable and Secure Computing. Vol. 4, Issue 3. Pages 165-179.

[2] Coull, S. et al. 2003. *Intrusion Detection: A Bioinformatics Approach*. Proceedings of the 19th Computer Security Applications Conference, December 8-12, 2003, Las Vegas, NV, USA. Pages 24-33.

[3] Denning, D. 1987. *An Intrusion Detection Model*. IEEE Transactions on Software Engineering. Volume SE-13. Issue 2. Feb. 1987. Pages 222-232.

[4] Esposito, D. 2002. *Cutting Edge: Windows Hooks in the .NET Framework*. MSDN Magazine, October, 2002. Retrieved from: http://msdn.microsoft.com/en-us/magazine/cc188966.aspx. Accessed 14 August, 2008.

[5] Garg, A., et al. 2006. *Profiling Users in GUI Based Systems for Masquerade Detection*. Proceedings of the

2006 IEEE Workshop on Information Assurance. June 21-23, 2006, West Point, NY, USA. Pages 48-56.

[6] Imsand, E., Hamilton, J. 2007. *GUI Usage Analysis for Masquerade Detection*. Proceedings of the IEEE Workshop on Information Assurance. June 20-22, 2007. West Point, NY, USA. Pages 270-276.

[7] Imsand, E., Hamilton, J.A. Jr. 2008. *Masquerade Detection through GUIID* (ACCEPTED). IEEE GLOBECOM Symposium on Computer and Communications Network Security. November 30 – December 3, 2008, New Orleans, Louisiana, USA.

[8] Li, L., Manikopoulos, M. 2004. *Windows NT One-class Masquerade Detection*. Proceedings of the 2004 IEEE Workshop on Information Assurance. June 10-11, 2004, West Point, NY, USA. Pages 82-87

[9] Maxion, R., Townsend, T. 2002. *Masquerade Detection Using Truncated Command Lines*. Proceedings of the 2002 IEEE International Conference on Dependable Systems and Networks. June 23-26, 2002. Pages 219-228.

[10] Microsoft Corporation. 2006. *Hooks*. MSDN Library. URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/hooks.asp. Accessed 10/23/06.

[11] Newcomer, J. 2001. *Hooks and DLLs*. URL: http://www.flounder.com/hooks.htm. Accessed: 10/23/06.

[12] Pusara, M., and Brodley, C. 2004. *User Re-authentication via Mouse Movements*. Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security. October 29, 2004, Washington D.C., USA.

[13] Schonlau, M., et al. 2001. *Computer Intrusion: Detecting Masquerades*. Statistical Science, 16(1): 58-74. February 2001.

[14] Umphress, D., Williams, G. 1985. *Identity Verification Through Keyboard Characteristics*. International Journal of Man-Machine Studies, Volume 23, Issue 3, Pages 263-273.