CUSTOMIZABLE METHOD OF AUTOMATICALLY DETECTING MALICIOUS USER ACTIVITY IN WEB APPLICATIONS

by

Han Gil Jang

2015

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

With the increase in the use of the web and security threats on web applications also at its highest point, the need for better security measures also increases. In this thesis we present a customizable method of automatically detecting malicious user activity for web applications. The customizable method has four phases. First, the method is uses information gathered from the web application access logs to represent the data in a certain way to create individual and typical user profiles. Then with the profile it goes through the training phase to compare the different profiles to create a threshold. Then threshold is then used to decide whether or not a new user is a malicious user or not. Finally, with new incoming information and the testing results, the system is calibrated to provide improved results in the future. In this thesis the design, implementation and results from a prototype following the method is presented as well as recommendations for security admins to follow in implementing this method into current web applications.

Chapter 1

INTRODUCTION

The popularity of the Internet has brought many changes to people's lives and along with it the popularity of web applications. A web application is a software that runs in a web browser through the web server and allows a user to enjoy the usage of a software without the downsides of having to take the risk or the complexity that comes with a traditional desktop software. A traditional desktop software would need to consider the differences in the system architecture and operating system when creating applications and thus lead to many more problems for the developer and the user. Also web applications have the advantage of being able to be used anywhere that has access to the Internet and a web browser. Finally web applications can update their software simultaneously without the need to remind the user to update the software and thus improves the user experience. The ease of update contributes to the ever changing aspect of web applications.

Due to this benefit, many companies have built business around a web application, such as the online retailer Amazon and search engines like Google, and the more traditional desktop softwares such as Adobe Photoshop and Intuit's TurboTax have moved their applications to the web. However, with the convenience comes another risk, which is the security that goes along with the web application.

With the recent growth in web applications, the security required for sensitive information, such as banking and private information, have yet to catch up. With alarming statistics that show that nearly 13 percent of all web applications can be compromised with purely automatic means[8] and nearly 50 percent of web applications containing a major security vulnerability and continuous innovations in the types of

web application attacks[8], such as the recent Heartbleed attack, it is more important then ever to find quick and simple ways to increase security of these web applications.

An important aspect of web application security is awareness: while it is important to invest in purchasing the latest security tools and hiring a specialist to make web applications more secure, it is important to understand and know when and who are making attacks on applications. Thus, I will present a new method of detecting malicious users through the use of user access logs to model user behavior to differentiate between a normal, harmless user and a user showing malicious intent.

The contributions that I made through this thesis are:

1. presenting the core of web application security, some common security tools used for web application vulnerability testing, and the issues and the goals my method is set out to accomplish,
2. designing and implementing an automated approach to analyzing user access logs to create a typical user profile, customizable with various representations of user behavior,
3. presenting a customizable approach to compare user profiles to identify malicious users,
4. designing and performing an experimental study of my approach's effectiveness, analyzing the results of the study, and providing recommendations to security administrators,
5. discussing possible improvements for future work.

The following sections include Chapter 2 Background, which will discuss the background information that is required to understand the justification for the approach I took as well as the technical concepts that are used in the further sections. In Chapter 3: Approach, I describe my problem statement, the goals that I am attempting to satisfy through my solution to the problem, and finally the detailed approach I took in solving the problem. Chapter 4: Results and Conclusions will have the results that I got through the use of my approach presented in graphs and tables and an analysis and interpretation of the results. Finally Chapter 5: Contributions and Future Works will discuss different contributions I have made through this solution and discuss the

parts of the project that need to be improved to have a more complete project in the future.

Chapter 2

BACKGROUND

This chapter will present background information to help the reader understand concepts that will be discussed in further chapters and be used as support for decisions made in the approach and experimental setup.

2.1 Web Applications

My thesis relies on the understanding of web applications. I have little doubt that you have heard of the Web: it's been with us for over two decades and has revolutionized many parts of our lives. Starting from communication applications such as Google hangouts, watching videos on YouTube, doing searches through the Google search engine, paying bills through online banking, and finally even buying a new supply of instant noodle products through Amazon, not a day goes without the Web. Knowing that it has taken a firm place in our modern lives, it seems obvious why people are researching issues regarding web applications and more specifically web application security.

While there are many aspects of web applications that we could discuss, for the sake of my thesis, it is important to understand the structure [14] and process of web applications that goes under the hood when a person accesses a web site. To understand how web applications are designed [18] let us go through an example. Suppose someone wishing to by a new book opens her browser and types the URL `www.amazon.com`. Within a few seconds, she will see the front page of the Amazon website. Seems rather simple, right? However, what actually happens when a person types the URL for the website is that the Amazon web application server sees that someone of a particular unique Internet Protocol (*IP*) address is sending a *request* for

the Amazon homepage (the *resource*, the 'R' in URL), and then serves the *response*, which contains the contents of the web page, back to the user. Another thing that likely happens at the same time is that the Amazon server keeps a record of this process in their *access logs*, which contains information about the requester's IP address, time, and the resource/page the user visited. When the user clicks the Books category on the page, the browser will then send another request to the server for the Books page, which the Amazon server processes. The user can also get to the Books section of the website by typing into the search bar "book". This process works in a similar way to the previous method, but the user has the ability to send Amazon's web application server a *parameter* with the value "book" along with the page, which the server processes when giving back its response to the request. Having discussed the basics of the process of user and web application interaction, we will discuss some specific methods in which a web application can be compromised and the importance of preventing such methods.

2.2 Web Application Security

Having established the prevalence of web applications, it is easy to understand why the most popular venue of attack is through web applications [2]. As our usage of the web increases, so does our need to have improved security for web applications. The increase in the number of attacks has been alarming, while the lack of proper defense still remains for many web applications that are currently deployed on the web. In fact, 13 percent of web applications currently up and running can be compromised with purely automatic means [7], and almost 50 percent of all web applications contain some kind of a major security flaw [8]. I will now describe a couple of these flaws as well as some current measures for prevention.

2.2.1 Code Injection

Code Injection attacks consists of injecting a piece of code to bypass authentication or authorization in the system [13] and thus manipulating parts of the system that the malicious user should have no control over. Ranking number 1 in the OWAS [5]

top 10 rankings, the attack's prevalence and threat is of very serious nature. The most common method of code injection is an SQL injection where the malicious user finds a part of a system that allows for inputting of a text as an input and uses this to inject a SQL command into the database as the application makes a query, and thus have access or the power to manipulate the system's database without authorization. This type of attack is also particularly dangerous as many web applications use SQL as a method of authentication and thus causing a serious threat of information for authentication leaking when the attack is successful. There are multiple ways applications protect against this type of attack, e.g., by using prepared statements in the Java programming language or doing a form of input validation to catch inputs that could cause a problem [3].

2.2.2 Cross-Site Scripting

Cross-Site Scripting is a method in which the malicious user injects a script into the web application, and then that script runs as if it were part of the application. This method of attack was recently ranked 3rd on Open Web Application Security Project's top 10 security attacks list [5]. The damage cross-site scripting can cause varies quite a bit, but a skilled user can take control of user sessions and even record key strokes [13]. Most of the attacks consist of finding a place in the application where it allows for text to be sent as an input and typing into the text box a JavaScript code that will execute in the system or for other users and thus can be used for exploitation.

2.2.3 Security Measures

2.2.3.1 Secure Construction

One way to improve the security of a web application is building the application in a secure manner. Applications can be constructed to minimize vulnerabilities from the time it is built, and many of the modern frameworks have built-in mechanisms that help root out certain types of vulnerabilities. The weakness of this method comes

from the fact that it is difficult to change an application once it is already built, and this is even more difficult for legacy systems.

Some examples of secure construction include SIF (Servlet Information Flow) model which is a framework for web applications that tracks application user input information to enforce security on both compile and run time [16]. Another method called SWIFT also does input validations but instead keeps track of end-to-end information flow policies and helps check if the user has the correct authorization for the information he is viewing [19]. Finally some other examples of secure construction is the use of HTML template systems that separate the user data from the HTML structure and thus preventing cross-site scripting attacks.

2.2.3.2 Security Testing

After creating a web application, it is important that the application is tested for bugs. However, it is equally as important that the application be tested for security [19]. The most common method of testing in security is penetration testing where the tester acts as if he were a malicious user and tries different known attacks to try to break the system. However, it is both costly and inefficient to have a security expert test every single part of the system using the most basic methods and thus it is very common to use vulnerability testing tools to automatically scan for vulnerabilities in an application.

2.2.3.2.1 Tools

One such tool is called Nikto [9], which is a web server vulnerability scanner. Nikto scans web servers and applications for out of date or unpatched software and is able to identify a wide range of configuration issues with the server. Another such tool is called WebSecurify [9], which is a simple way of automatically testing target web applications. It quickly scans and checks for SQL injection, cross-site scripting, file includes, cross-site request forgery, and vulnerability.

The final tool I will introduce is Metasploit [6]. Metasploit is a penetration testing software that both tests and assists attacks in customizable ways. It allows for the tester to easily develop their own custom attacks and thus it is popular as both a testing tool as well as a hacking tool used by malicious users. A tester can configure the tool by choosing an exploit or a module and the victim host and ports along with optional system versions, account information, and so on, and the tool will attempt to find exploits in the application.

2.2.3.3 Runtime Protection

The final method of web application security does not deal with the web application directly, but serves as a barrier of sorts by monitoring the traffic between the web server and the user [15]. Through this the augmented program catches at run time or through later analysis attacks that were made to the web applications. The two ways that this method is pursued is by first tracking what seems to be potential malicious user and quarantining such user when suspicious actions are made [4]. And another method is by blocking the suspicious attacks from happening through input validation [11]. Both of these methods rely on the accuracy of the program to correctly detect when an attack happens and the current method of detection relies on finding patterns in the user inputs. One example of such a system is Snort [10] which is a generic tool that checks user input for known exploits through blacklisting of patterns that are known to cause problems for the application.

While such methods are very useful in detecting known attacks are in many cases helpless against the every change nature of the web. Thus it is important for us search for ways to secure web applications against possible future attacks.

Chapter 3

APPROACH

Having outlined the background information required, now I will present the problem that I am trying to solve, the goals for my problem and finally the approach that I took in solving the problem.

3.1 Problem Statement

The problem that I am focusing on is finding an effective method of detecting malicious user activity in a web application.

3.2 Goals

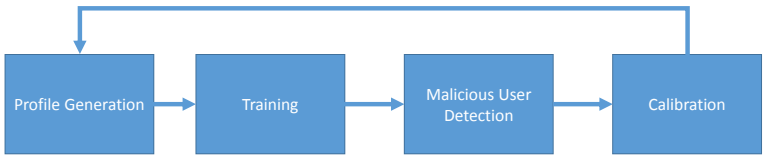
These are the goals that I will satisfy for the solution to the above problem.

1. Effectiveness: The first and perhaps the most important part to achieve is the effectiveness of the solution in identifying malicious users while not falsely accusing normal users of malicious activity and thus not impeding usability of the system.

2. Speed: The time it takes for our solution is important because of the potential of web applications to have large user-base and code-base, and thus it will be important that our project will be computationally feasible when giving us the results to be actually useful.

3. Scalability: The scalable and customizable aspect of the solution is of particular interest as the purpose for each web application is vastly different and thus the solution should be highly customizable to allow for the security admin to have customized settings that will work the best for their own application.

Figure 3.1: Overview of Approach



3.3 Design

To meet the goals outlined, I developed a customizable framework that creates a method of identifying malicious user activity. This section will be divided into four cyclical sections, the profile generation, training, malicious user detection and finally calibration sections. The diagram 3.1 will provide an outline of each section. Each section will explain how it fits into the whole process as well as give an example of the prototype that was to help better understand the whole process.

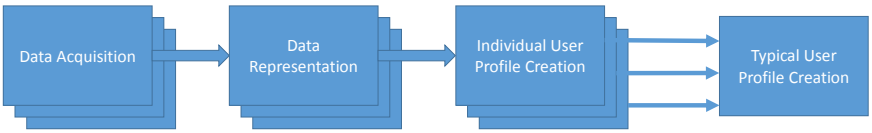
3.3.1 Profile Generation

The Profile Generation phase 3.2 creates a model of the behavior of the typical user for a web application. This is the initial process of picture where one starts with the input of user access logs gathered from a web application and produces an output of user session profiles that are used as a comparison in the training phase.

3.3.1.1 Access Log Acquisition

The initial data will be acquired from the user access logs, which contain several pieces of information such as the time stamp of each access, each page that is visited

Figure 3.2: Outline of Profile Generation Phase



and the parameter values that were input, from a web application and then passed on through the parser so that it can be read in a format to be represented in a particular way.

3.3.1.2 Data Representation

Using the information in the user access logs, a security admin could represent the user in multiple ways, such as the distribution of the pages visited, a set of single pages the user visited in a session, the amount of times passed in between each navigation, the order in which the different pages were visited and so on. When the security admin decides on the data representation and the raw data of the logs are parsed accordingly, then the parsed data can be passed on to the next section to create a model of the usage.

3.3.1.3 Individual User Profile Creation

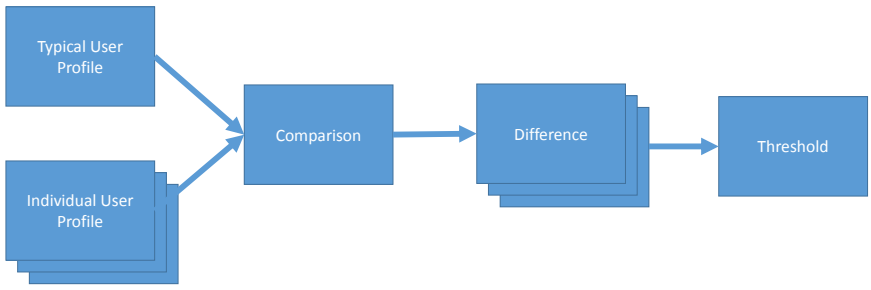
Having decided how the user session information is going to be represented, one must decide how to store that information into an individual user profile. In the case of using the distribution of pages each person visited or each set of pages a person

visited, a histogram would be an ideal method to store such information. In the case of using the order and likelihood of each page visited it might make sense to use other models such as the Hidden Markov Model (HMM).

3.3.1.4 Typical User Profile Creation

With the individual user profiles created, a profile representing a typical user behavior needs to be created to be used as a basis for comparison in the next phase. One can decide to aggregate the data in multiple ways from averaging the data where each user is given equal weight in calculation or an aggregation of the data where each page visit is given equal weight. Thus at the end of this process a profile, representing the aggregate user behavior is created in the same form as an user profile a histogram.

Figure 3.3: Outline of Training Phase



3.3.2 Training

The Training phase 3.2 will take as inputs the typical user profile and all the individual user profiles and compare them to find the threshold that can be used in the next phase for the malicious user detection phase. The threshold will determine the *false rejection rate* (**FRR**), the percentage of normal users that will be flagged

as malicious, and *false acceptance rate* (**FAR**), the percentage of malicious users that are flagged as normal. These metrics can be used to measure the effectiveness of the approach.

3.3.2.1 Comparison

Using the inputs of an individual user profile and the typical user profile, we can find the difference between the two profiles. The comparison method depends on how the data was stored in the previous phase as the Hidden Markov Model only gives a single method of comparison. However, histograms can use the simple distance comparison of the Manhattan distance or more complicated comparisons such as the Chi-Square and Bhattacharyya distance[12]. The resulting calculation would be the difference score that would represent how different a single user is from the typical user.

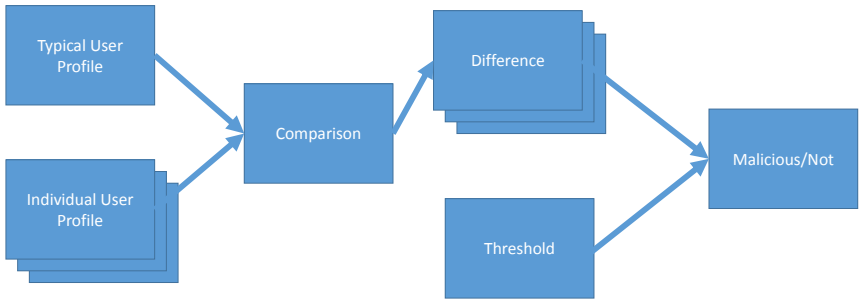
3.3.2.2 Threshold Calculation

The previous process of comparison is done for the entire set of individual user profiles with the typical user profile to find the difference score. Then using the set of difference scores, the threshold can be set by determining the False Rejection Rate (FRR). Thus the security admin chooses a threshold flagging a certain percentage of the users above the threshold as malicious. The higher the FRR the lower the False Acceptance Rate (FAR) will be in the Malicious User Detection phase.

3.3.3 Malicious User Detection

This phase 3.2 takes in as an input the typical user profile and a new individual user profile to determine the incoming user as a malicious user or not. When a new individual user finishes their session the user profile is created just like in profile generation step. Then the difference score is calculated in the same way as the Training section and then the difference score is then used by comparing it to the threshold to mark a user profile as malicious or not.

Figure 3.4: Outline of Malicious User Detection Phase



3.3.4 Calibration

After the three phases are completed the security admin will have to check the logs manually to see if real malicious user activity has been found and calculate the false acceptance rate (FAR). Then he can recalibrate the system by repeating the process over with new access logs and different models and methods for each step of the phases to improve the system.

3.4 Prototype

This section will explain the details of my prototype on how and why it was implemented this way. The prototype was implemented using Java following the design described above. [Add Code Online]

3.4.1 Profile Generation

In the Profile Generation phase, the prototype first takes the access logs generated by a web application and parses them, removing all other information other than the page that the user visited for each access and separating the accesses into different user sessions using the IP address and the time involved in between each activity.

I explored two different methods of data representation:

1. **Navigation Page Distribution**: The Navigation Page Distribution focused on the navigation of the user and calculated the frequency of each page visited. The idea for using navigation behavior was based on the idea that most users tend to follow a similar pattern[17] when using an web application to do particular tasks. The hypothesis is that an malicious user who is planning to gain access to sensitive information would significantly deviate from this behavior.
2. **Pair Navigation Page Distribution**: Similar to the navigation page distribution above, the navigation page pair distribution looked at navigation in terms of two consecutive pairs. This gave us the frequency information of the above while also giving us some information regarding the *order* which a typical user would navigate through a web application and possibly allowing us to further differentiate from a malicious user who could make unusual navigation.

After gathering all the parsed logs for each user session, the prototype implementation runs through each user session and creates an individual user profile for each of them by counting the number of visits per page and storing them into a pre-made array representing all the possible pages or page combinations in the web application. Then, dividing each cell by the total number of visits the user made in that session yields the distribution.

After all the individual user profiles were created, the aggregate user profile was created by following a similar process, but this time by going through all the user session logs and counting the number of visits for all users and then dividing by the total number of visits made by all the users. In our case the aggregation model was used as the variance in the user sessions was quite large and it seemed better to not give user sessions with only a couple of visits the same amount of weight as one that would visit more than a hundred pages.

3.4.2 Training

Then for the Training phase each individual user profile was compared to the aggregate user profile using the simple histogram comparison of finding the Manhattan distance between two histograms. This particular comparison was chosen because it was the simplest method of comparison and thus resulting in the maximization of

speed for our testing. After each comparison the difference score was generated and with the set of difference scores the threshold was generated for several FRR (false rejection rate) to see how different limits for the threshold would affect the FAR (false acceptance rate).

3.4.3 Malicious User Detection

The Malicious User Detection phase was also written in Java and the threshold found from the Training phase was used to compare against the difference score between the incoming user profile and the typical user profile. If the difference score was higher than the threshold it was flagged a malicious and if it was lower it would be marked as not malicious.

Chapter 4

EXPERIMENTAL STUDY

In this chapter I first outline the details of how I designed the experiment and then discuss the results that were produced from the study.

4.1 Experimental Study

4.1.1 Research Questions

I designed my research study to answer the following research questions:

1. What are the behavioral patterns of typical users and malicious users, as represented by penetration testing software?
2. What are the patterns of variance in behavior among typical users and malicious users?
3. What is our method's effectiveness in terms of accurately identifying malicious users?
4. What is the impact of modifications to the data representation on the effectiveness of identifying malicious users?

4.1.2 Subject Application

The study was done using the prototype described in the previous chapter on a web application named Logic, which is an academic application used at Washington and Lee University in the Philosophy 170 course. The application allows the professor to create logic questions for the students to answer and automatically grades those answers submitted. This particular application was chosen for its previous usage allowing for a wealth of logs that could be used for testing as well as its place as a generic web application that isn't overly complex and does not have built-in security features. It

was mainly chosen to model a real life situation of a typical web application that might not be able to afford significant security features.

Some characteristics of the application include 135 classes and 16491 non-comment lines of code with a total of 79 pages that are possible to visit.

The number of user sessions gathered from the access logs collected during several semesters were 6936 sessions in total, of which 2839 were from non-bot sources. There were a total of 11 admin user sessions, 201 user sessions without login information, 68 professor user sessions, and 895 student user sessions. The sessions that had more than one user levels were removed to keep the sessions of each user level consistent.

4.1.3 Methodology

4.1.3.1 Generating User Profiles

The individual user profiles and the aggregate user profile (Section 3.3.1) were created from previously collected access logs from the Logic application.The results for the set of user session logs that included more than 5 page visits were collected and presented to show the difference of the results with the omission of shorter length outliers that had minimal effect on the state of the application. The reasoning behind the exclusion is that a malicious user with such few page visits would not be able to perform many malicious actions.

We noted the time to generate the profiles.

4.1.3.2 Generating Malicious User Profiles

First, the Logic application was deployed on the web application server Apache Tomcat[1]. Next, malicious users were emulated using the penetration testing software Metasploit(Section 2.2.3.2.1). Metasploit was used to scan through the web application and test for vulnerabilities and was chosen because of its popularity. The thought behind using penetration software is that most attacks of cybersecurity are done through

automated tools, and thus it seemed most important to test the effectiveness of our method against these tools first before trying other means.

The vulnerability testing was done for several different Logic application authentication levels of (1) no login information, (2) student level, (3) professor level, and (3) the admin level, which each had different privileges in the application. The various runs were performed because the different types of users would perform vastly different actions when using the application, and, thus, to improve the effectiveness of our method, this distinction between the user levels were made. The Metasploit tests were run for .5, 1, 2, and 5 minutes at the 4 most popular starting positions as seen in the access logs for a total of 16 malicious user profiles for each user level.

4.1.3.3 Representing Data

Finally, the results were performed and analyzed again using another method that considers the order of navigation by counting the frequency of navigation pairs, which are the two consecutive pages visited in the logs, to observe the similarities and differences between the two results.

4.1.3.4 Identifying Malicious Users

The thresholds of 0, 5, 10, 25, 50 percent FRR (false rejection rate) were created, and the threshold was then used to determine the FAR (false acceptance rate) of malicious users.

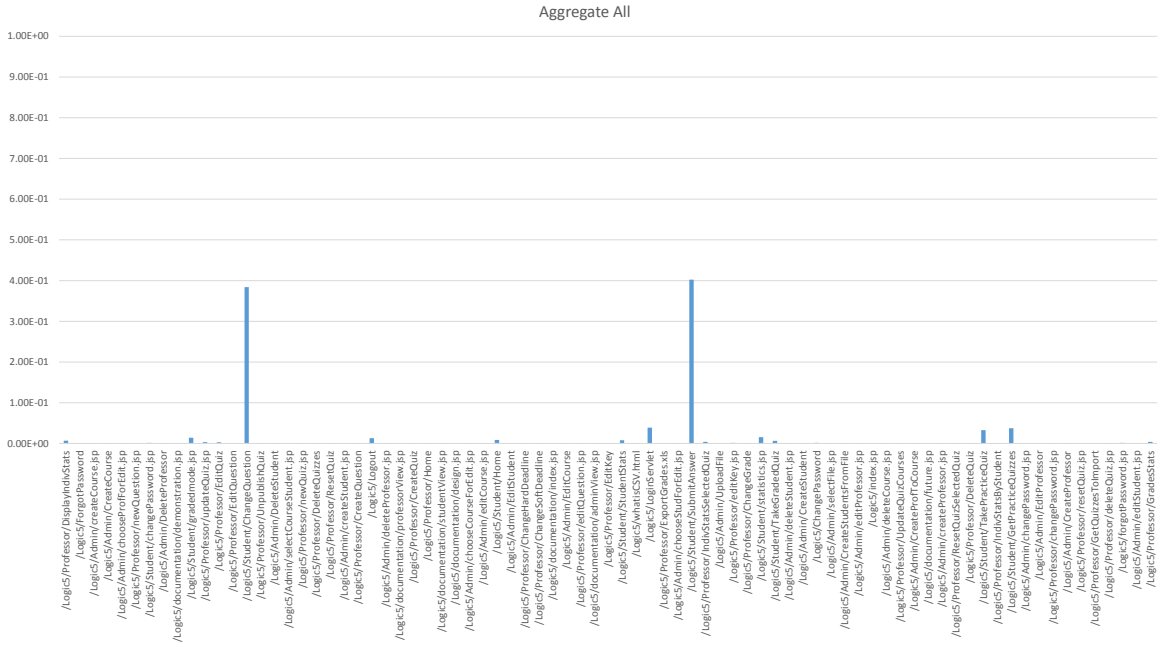
The comparison in the training phase was done as was described in the previous section (Section **??**) by comparing each individual user profile against the typical user profile to find the difference score.

4.2 Results

4.2.1 User Profiles

Figure 4.1 shows what a typical user profile of the Logic application looks like when considered in aggregate.The x-axis represents all the possible pages that the user

Figure 4.1: Typical User Profile for All



could visit in the Logic application while the y axis represents the relative frequency of the page visited out of all the pages visited for the user.

We can see that as the majority of the users are students because the pages `Student/ChangeQuestion` and `Student/SubmitAnswer` are represented quite heavily, while the other pages are still visited but seem almost negligible.

Figure 4.2 shows the of typical NoLogin user profile. Curiously, the outlier in the NoLogin Profile is the `Logout` page, which probably resulted from a user's session timing out due to inactivity; the user then clicked to Logout, which the the parser considered the start of a new session; then, the user never logged in again in that session.

Figure 4.3 shows the of typical Student user profile. As expected, the student profile does not differ too greatly from the typical user profile showing all users 4.1. The majority of the page visits in the typical student profile are concentrated along the expected `ChangeQuestion` and the `SubmitAnswer` pages, as those pages multiple

Figure 4.2: Typical User Profile for NoLogin

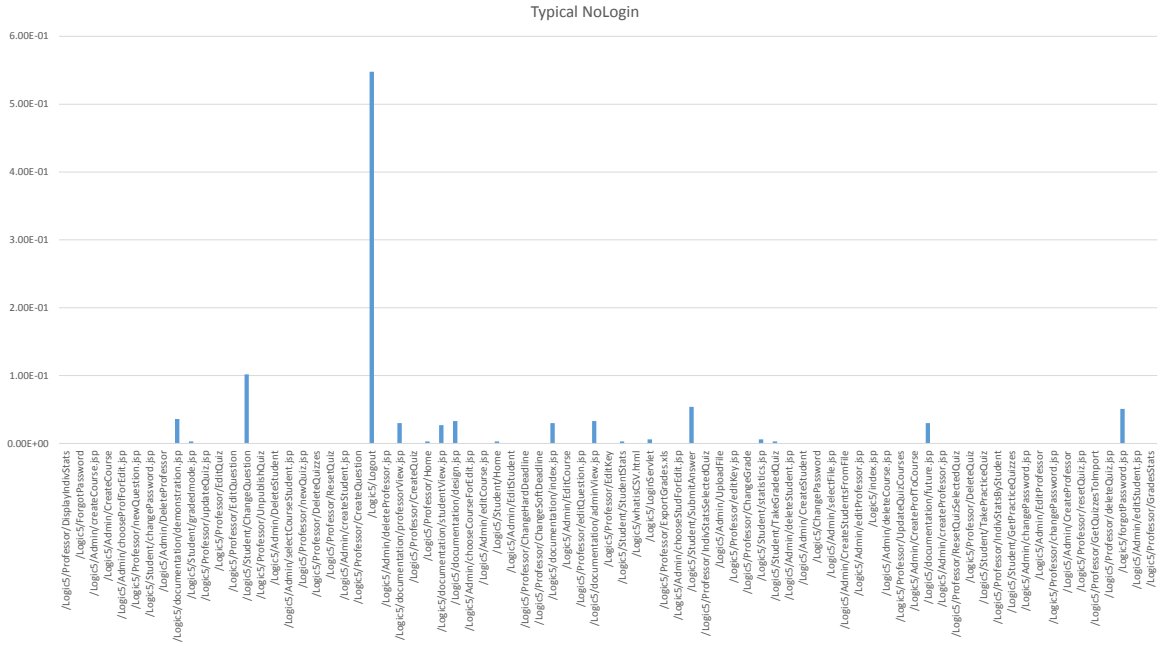
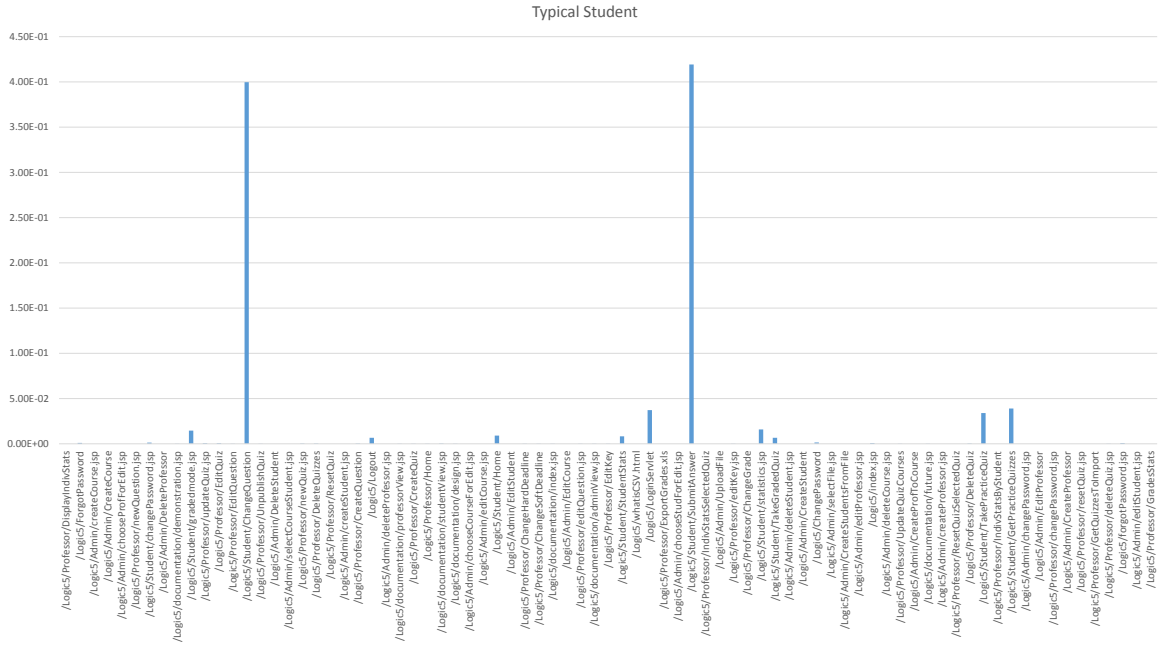


Figure 4.3: Typical User Profile for Student



times while taking a quiz.

Figure 4.4: Typical User Profile for Professor

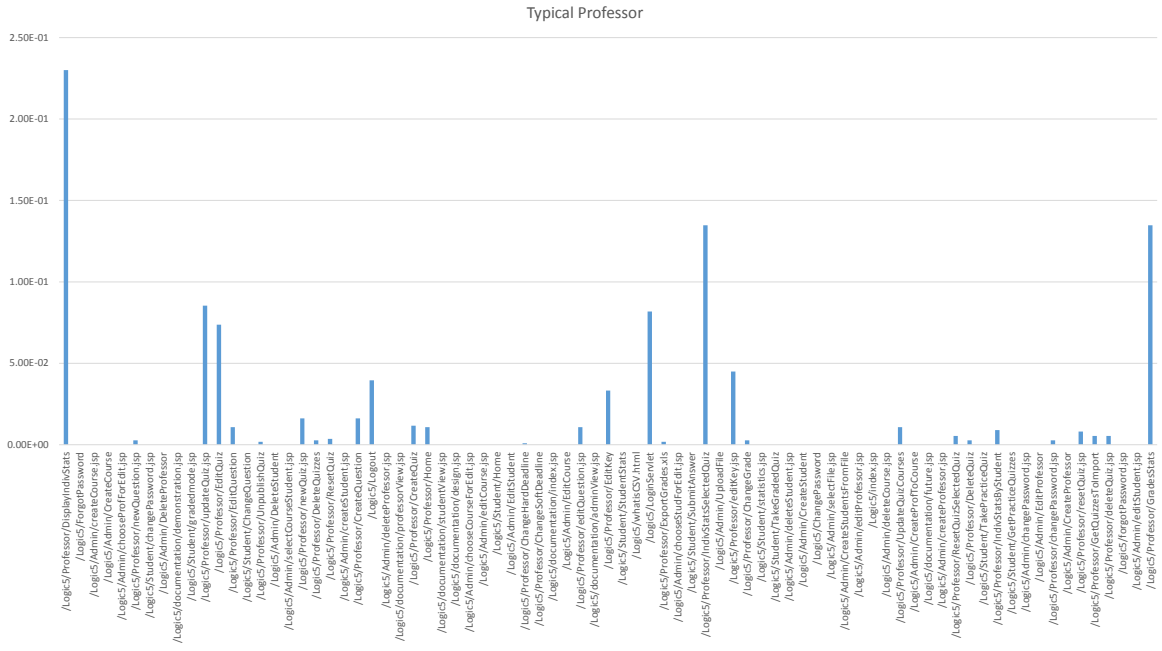


Figure 4.4 shows the of typical professor user profile. The typical professor profile differs greatly from the previous user profiles as it has access to larger numbers of pages, and the pages heavily visited by the professors are the statistics pages for the students as well the creation and the editing of quizzes.

Figure 4.5 shows the of typical admin user profile. The typical admin profile also visits quite a number of pages, but, because of the limited amount of available admin user sessions, we can see that `LoginServlet`, which is the page a user visits to log in, is quite high on the visited list as compared to the other user levels.

Figure 4.6 shows an example of a typical malicious user profile. The typical malicious profile has the widest and most evenly distributed profile, as expected from a bot who does not give preference of visiting pages for functionality but rather for exploitative purposes. Also because for the testing of bots equal number of tests for

Figure 4.5: Typical User Profile for Admin

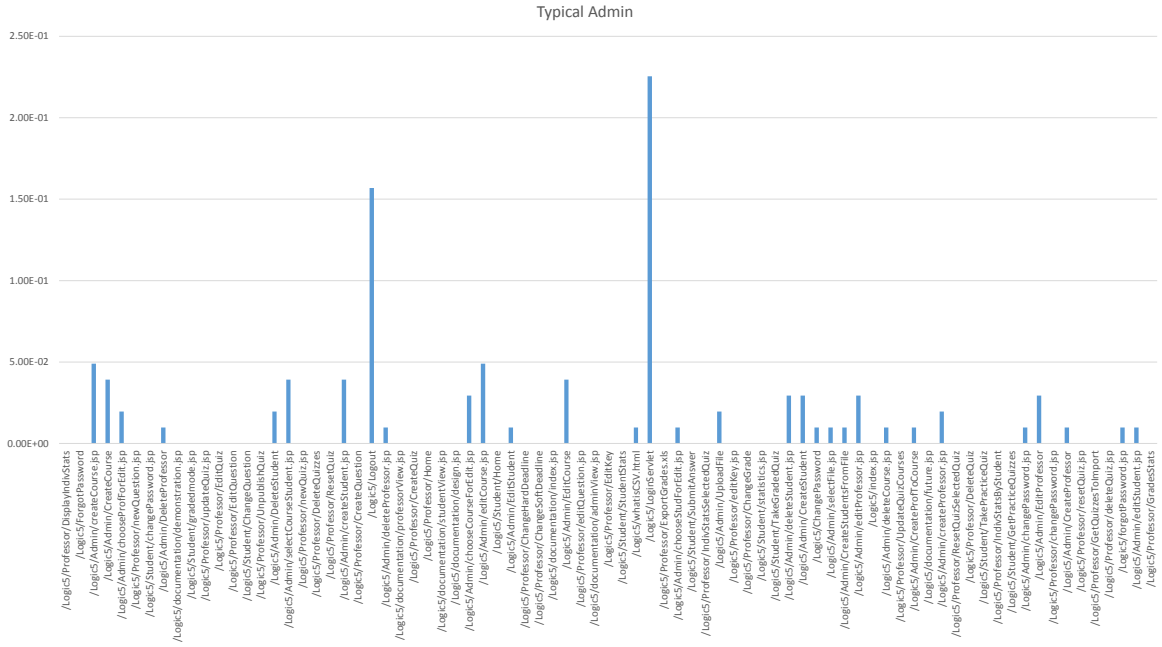
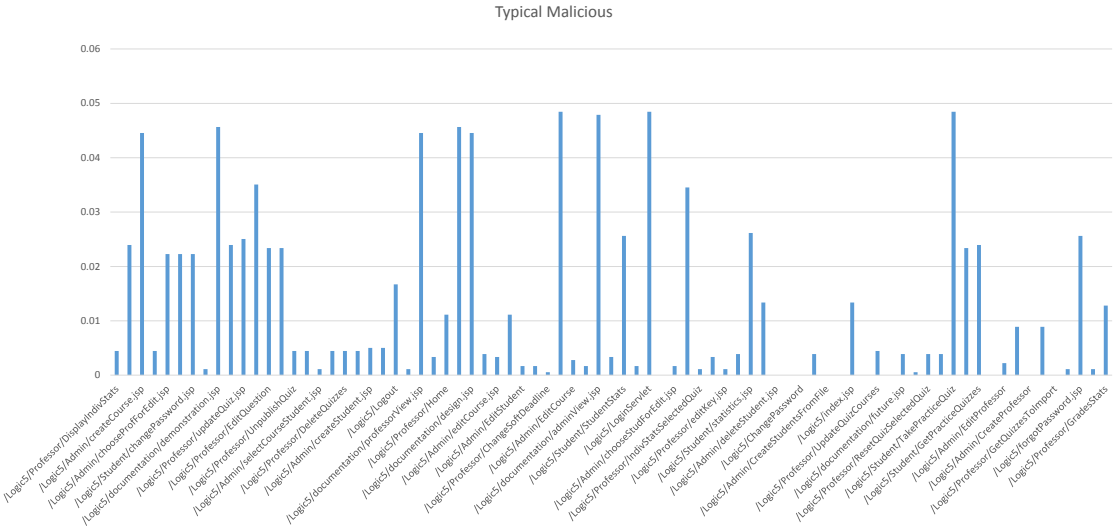


Figure 4.6: Typical User Profile for Malicious User



each of the 4 levels authority were run, the profile shows a more even distribution of pages and shows a concentration of pages that can be visited by multiple authorities.

4.2.2 Training

Figure 4.7: Difference Score of All Users

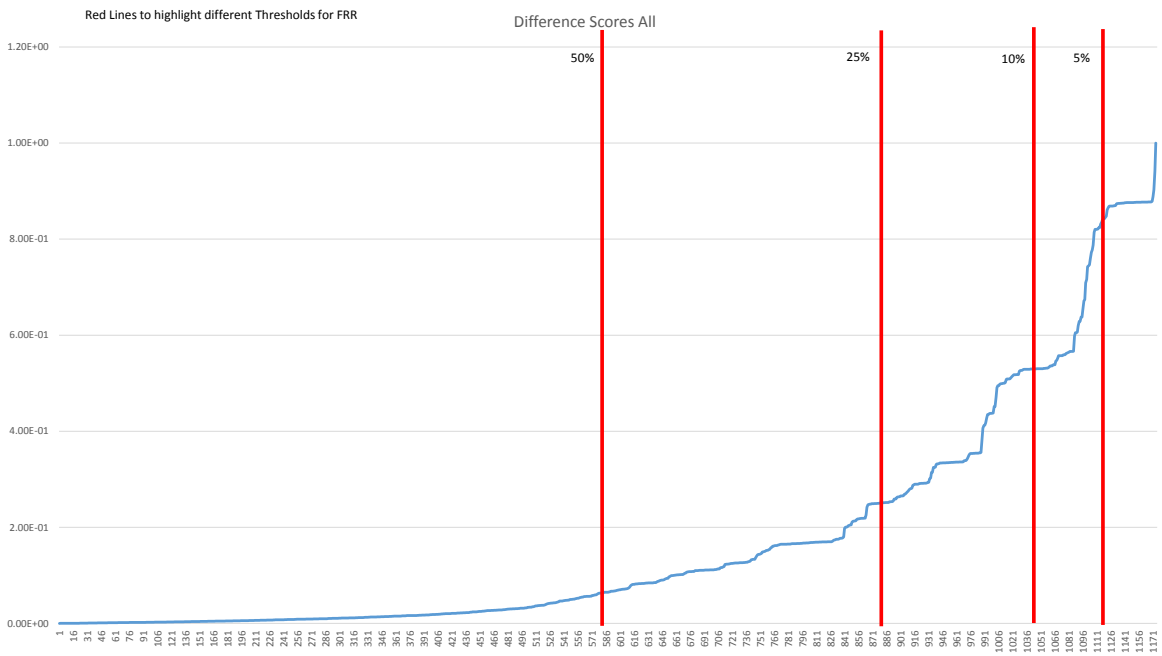


Figure 4.7 shows an example difference scores when comparing each user session's profile to the typical, aggregate user session's profile. The x-axis represents the id of the user session, in increasing order of difference score; the y-axis shows the value of the difference score for each user session. In other words, the x-axis shows the total number of user sessions that fall below the difference score shown on the y-axis.

The red vertical lines help to visualize what percentages of user sessions below certain difference scores to show what the threshold would be for the 0, 5, 10, 25, 50 percent FRRs. We can see that as the percentages increase there are great drops between the 5% and the 10% and some more between the 10% and the 25%. This shows that while the majority of the user sessions are similar to one another, the student user

FRR/Auth.	0%	5%	10%	25%	50%
NoLogin	93.75%	62.5%	37.5%	31.25%	6.25%
Student	93.75%	43.75%	37.5%	25.0%	0.0%
Professor	81.25%	43.75%	18.75%	12.5%	0.0%
Admin	75.0%	25.0%	25.0%	12.5%	6.25%
All	100%	56.25%	31.25%	18.75%	0.0%

Table 4.1: FRR vs FAR for different levels of Authorization

sessions, there are the other level user sessions in different concentrations thus causing the sharp drops in this graph.

4.2.3 Time for Tests

As for the time it took for each phase to run here are they are. For creating file for all 2839 for page names it took 428ms. For creating individual profiles for 2839 files 2310ms. For creating aggregate profiles for 2839 files it took 411ms. For comparing all user sessions to the typical user session it took 335ms.

4.3 Analysis of Results

Each table shows the false acceptance rate for each level of user for a given rule and a set false rejection rate. The trend as seen from 4.7 shows that the false acceptance quickly decreases as the false acceptance level increases.

Table 4.1 shows the FAR rate for different levels of FRR for different levels of authorization. The professor and the admin authorizations performed particularly well, which is to be expected because these users have more possible pages to visit and thus making it easier to differentiate between a real user and a malicious user. Looking at the different FRR percentages, we can see that at the lowest FRR of 0% it seems that the false acceptance rate is very high because, even among the user sessions of each level, there probably exists at least one user session that is a large outlier to the rest of the user sessions thus causing the difference score to be very large. However even

FRR/Auth.	0%	5%	10%	25%	50%
NoLogin	68.75%	43.75%	25.0%	12.5%	6.25%
Student	56.25%	18.75%	6.25%	6.25%	0.0%
Professor	50.0%	31.25%	18.75%	12.5%	0.0%
Admin	56.25%	25.0%	18.75%	12.5%	6.25%
All	75.0%	43.75%	25.0%	18.75%	0.0%

Table 4.2: FRR vs FAR for different levels of Authorization with low visit sessions removed

FRR/Auth.	0%	5%	10%	25%	50%
NoLogin	68.75%	62.5%	31.25%	25.0%	6.25%
Student	62.5%	37.5%	37.5%	31.25%	12.5%
Professor	68.75%	31.25%	25.0%	12.5%	0.0%
Admin	43.75%	25.0%	18.75%	12.5%	0.0%
All	87.5%	43.75%	25.0%	12.5%	0.0%

Table 4.3: FRR vs FAR for different levels of Authorization Using Pair Navigation

with the increase to 5% FRR the results are much improved and further improvements with the increase to FRRs.

Table 4.2 shows the FAR rate for different levels of FRR for different levels of Authorization with the low visit sessions removed. The idea here as explained above was that it would eliminate many outlier user sessions with very low amount of visits. This seemed like a reasonable assumption because a malicious user with such low number of visits wouldn't be able to accomplish anything of note to the application. We can see that compared to table 4.1 most of the sessions had improved results. Even at 0% FRR the results seemed favorable as many of the outliers were eliminated. The Student level user sessions particularly improved as there were quite a lot low visit outlier student sessions, and with those eliminated, the remaining user sessions converged to taking a quiz while the malicious user attempted other things in addition to just taking a quiz. The all session also improved a bit, with large portion of the student users outliers removed the other sessions were given greater weight, thus differentiating it from the malicious user.

Table 4.3 shows the FAR rate for different levels of FRR for different levels of Authorization using the pair navigation. The pair navigation as described above takes visits in pairs to account for order of navigation in addition to the frequency. The results here were somewhat mixed, but similarly to Table 4.2 the user sessions with just one visit had to be eliminated to allow for each user session to have at least a pair and thus the 0% FRR seems to have better results than Table 4.1. Also the Professor user, and Admin user sessions, were improved because having the extra information of the order helped differentiate a real admin or professor user which had long user sessions from the malicious bot user.

4.4 Recommendations to Security Administrators

Looking at the results of the prototype from each test one can conclude that the method can be successfully used for automatically detecting malicious users to varying degrees of success. To have a stronger conclusion requires a test with larger set of data and approaches. However even with results from our experiment some recommendations can be made to security administrators who are trying to implement this method into their current web applications. First as noted from Table 4.1, it is advantageous to start choosing a threshold with a minimum of 5% FRR as it eliminates much of the outliers and improves the FAR rate dramatically. Also depending on the application, it could be advantages to remove the user sessions with very low number of visits as it also eliminates the outliers and improves results as can be seen across all levels of user authorization and FRR rates. Finally for web applications that have activities that follow a certain order of navigation it can be advantages to use pair navigation data representation to add extra bit of information when differentiating between a normal and malicious user.

As for the speed and overhead of the prototype using the method, it can be considered almost negligible for our purposes. While the initial creation of the profiles could take some time for very large web applications with large sets of data, the comparison step for a single comparison took less than 1ms and thus the malicious user

detection phase of the process will can be completed in an instant and thus it will be possible to deploy even as a real-time malicious user detection tool.

Chapter 5

CONTRIBUTIONS AND FUTURE WORK

This chapter will serve to summarize all the contributions that were made through this thesis as well as future work that needs to be done to further improve the method in detecting malicious users in web applications.

5.1 Contributions

The contributions of my work presented in this thesis are:

1. Explaining the uniqueness of a web application and basic request-response structure of the process behind user and web server interaction through the web application. Presenting the current state of web application security and examples of common threats like cross-site scripting and SQL injection on web applications. Discussing different methods and limitations of web application security such as secure construction, testing, and augmented protection. Also discussing some popular tools used in both testing and attacking including Metasploit (Section 2.2.3.2.1).

2. Designing a customizable framework for automated malicious user detection that helps security administrators to experiment in improving security. The method is separated into four phases of User Profile Generation, Training, Malicious User Detection, and Calibration. For each phase in the process the security administrators have an option to choose what approach to take in depending on their application and some examples of these can be found in section 3.

3. Building a prototype (section 3.4) of the method above and making the source code open source so that other wanting to try out this method can build on this method.

4. Designing and performing an experimental study of my approach's effectiveness, analyzing the results of the study, and providing recommendations (section 4.4) to security administrators to help customize the method for their personal application.

5.2 Future Work

Future work in improving the model include:

1. Additional studies using several different subject applications to see how it affects the rates of FRR to FAR.

2. More extensive tests using several different methods of data representation that considers order of navigation even more than the pair navigation model. Others might include including time stamps and resources that are visible from the access logs.

3. Tests using other methods of profile generations perhaps using the Hidden Markov Model mentioned above or other models.

4. Tests that show the effectiveness of the method in the ever changing state of the web by showing the effectiveness in differentiating other malicious users from different tools while using the user profile of the current tool.

5. Further tests by collecting data from a real life application.

6. Using machine learning techniques to optimize the process of finding the best customization for different types of applications.

BIBLIOGRAPHY

[1] Apache tomcat. http://tomcat.apache.org/. Accessed: 2015-3-30.

[2] Web application security 101. AppliCure Technologies, 2014.

[3] Sabah Al-Fedaghi. Developing web applications. *International Journal of Software Engineering and Its Applications*, 5:57–68, 2011.

[4] Alejandro Perez-Villegas Carmen Torrano-Gimenez and Gonzalo Alvarez. An anomaly-based approach for intrusion detection in web traffic. *Journal of Information Assurance and Security*, 5:224–231, 2010.

[5] Intrusion detection for web applications. ukasz pilorz. OWASP Web Intrusion Detection Publication, 2009.

[6] Patrick Engebretson. *The Basics of hacking and penetration testing*. Elsevier Inc, Waltham, MA, 2011.

[7] Mohammad Shkoukani Hesham Abusaimeh. Survey of web application and internet security threats. *International Journal of Computer Science and Network Security*, 12:67–76, 2012.

[8] Security Innovation. 2012 application security gap study:a survey of it security & developers. *Independently Conducted by Ponemon Institute LLC*, 2012.

[9] Peter Kim. *The Hacker Playbook*. Secure Planet LLC, North Charleston, South Carolina, 2014.

[10] Rakesh D. More Manoj E.Patil. Survey of intrusion detection system in multitier web application. *International Journal of Emerging Technology and Advanced Engineering*, 2:570–575, 2012.

[11] Drew Miller. Application intrusion detection. Black Hat Consulting, 2013.

[12] Aaron Paloski Nan Zheng and Haining Wang. An efficient user verification system via mouse movements. *ACM CSS*, 2011.

[13] Pankaj Sharma Rahul Johari. A survey on web application vulnerabilities(sqlia,xss)exploitation and security engine for sql injection. *International Conference on Communication Systems and Network Technologies*, 10:453–458, 2012.

[14] Jennifer Niederst Robbins. *Learning Web Design, Fourth Edition*. Littlechair, Inc, Canada, 2012.

[15] Laila M. El-Fangary Yehia K. Helmy Shaimaa Ezzat Salama, Mohamed I. Marie. Web server logs preprocessing for web intrusion detection. *Computer and Information Science*, 4:123–133, 2011.

[16] Laila M. El-Fangary Yehia K. Helmy Shaimaa Ezzat Salama, Mohamed I. Marie. Web anomaly misuse intrusion detection framework for sql injection detection. *International Journal of Advanced Computer Science and Applications*, 3:123–129, 2012.

[17] Sara Sprenkle, Lori Pollock, and Lucy Simko. A study of usage-based navigation models and generated abstract test cases for web applications. In *International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, Mar 2011.

[18] Iyakutti K Sridaran R, Padmavathi G. A survey of design pattern based web applications. *Journal of Object Technology*, 8:61–70, 2009.

[19] YUAN XUE XIAOWEI LI. A survey on server-side approaches to securing web applications. *ACM Transactions on Computing Surveys*, 5:1–31, 2013.