Examensarbete
15 högskolepoäng, grundnivå

Web-Based Intrusion Detection System
Webbaserat intrångsdetekteringssystem

Muhamet Ademi
Abstract

Web applications are growing rapidly and as the amount of web sites globally increases so do security threats. Complex applications often interact with third party services and databases to fetch information and often interactions require user input. Intruders are targeting web applications specifically and they are a huge security threat to organizations and a way to combat this is to have intrusion detection systems. Most common web attack methods are well researched and documented however due to time constraints developers often write applications fast and may not implement the best security practices. This report describes one way to implement a intrusion detection system that specifically detects web based attacks.
## Contents

1 Introduction ................................................................................................. 1  
1.1 Background .......................................................... 1  
1.2 Problem Description ................................... 2  
1.3 Purpose ........................................................ 2  
1.4 Limitations ............................................... 2  
1.5 Target Audience ....................................... 2  
1.6 Outline of Thesis ....................................... 2  
2 Central Concepts ....................................................................................... 4  
2.1 Intrusion Detection System ........................................ 4  
2.2 Common Web Attacks ......................................... 5  
3 Methodology .............................................................................................. 12  
3.1 Literature Review ............................................... 12  
3.2 Software Experiment Environment ....................... 13  
3.3 Software Implementation ..................................... 15  
3.4 Software Experiment Tests .................................... 16  
4 System Design & Implementation ....................................................... 20  
4.1 System Overview ................................................ 20  
4.2 IDS Filters .................................................... 21  
4.3 Application Structure .......................................... 22  
4.4 Database Design ............................................... 23  
4.5 Program Flow ................................................ 24  
4.6 XML File Structure ............................................ 25  
4.7 Administration Panel .......................................... 25  
5 System Impact & Analysis ......................................................................... 28  
5.1 Efficiency ....................................................... 28  
5.2 Performance ................................................ 30  
6 Discussion ................................................................................................. 35  
6.1 Effectiveness .................................................. 35  
6.2 Performance ................................................ 35  
6.3 Administration .............................................. 36  
7 Conclusions and Future Work .................................................................. 38  
7.1 Future Work .................................................. 38  
Bibliography ................................................................................................ 40
1 Introduction

Web applications are increasing and software development firms are starting to deploy web applications and replacing traditional client applications. Web applications growth of complexity has also increased the number of vulnerable web applications in terms of security and is a growing concern. In this paper we implement a intrusion detection system and evaluate the effectiveness of the proposed system and present the results.

1.1 Background

We live in a dynamic and rapidly changing environment and technology is becoming a huge factor in our daily lives and one that we use to great lengths to aid in solving problems. Web applications have always played a critical role in this and they are becoming more important as many applications are designed to deal with a specific area of interest. It does however have some side effects, for instance web applications can potentially be a security risk [13] and there are many attackers targeting web applications specifically. Modern web applications have a tendency to utilize third party APIs, services and this can be seen as additional room for security vulnerabilities. Over the past few years the amount of intrusion attempts has been on a steady increase and statistics show that as much as 75% of the cyber attacks [13] target web applications specifically. In a survey conducted [13] it was found that web attacks cost organizations over 100 times more than malware, and 50 times more than viruses, worms and trojans annually. In this thesis we propose a intrusion detection system that operates in the application layer and is built specifically to detect web based intrusion attempts. The field is well researched, thus a large number of scientific papers propose multiple solutions but often center around a single attack method. Other problems with previous work that exist today is that they are able to identify intrusion attempts after they occurred which presents difficulties, if the particular application was altered to communicate with a third party software application, for instance a firewall. In this case, if the presented solution was altered to have the functionality to suspend a intruder by IP address by communicating with the firewall the suspension would be after the intrusion attempt. It is therefore more suitable to have the IDS execute prior to the web application so that intrusion attempts can be detected before the web application executes the user request.
1.2 Problem Description
The nature of this thesis is to implement a software based intrusion detection system and evaluate the effectiveness, thus the research questions that this particular paper will answer are the following:

- How can a stand-alone software application be used to identify security intrusion attempts?
- In what way can we measure the system to check that it is appropriate for real world usage?

1.3 Purpose
The purpose of this paper is to investigate common web attack methods and to identify intrusion attempt patterns and their effectiveness in detecting intrusion attempts. The goal of this study is to implement a software based intrusion detection system and evaluate the effectiveness of intrusion detections. Intrusion detection systems can also be closely tied to third party security software to enhance the benefits of having a intrusion detection system. Contribution to the research field is by presenting data of how effective a system of this nature can be to detect intrusion attempts and a possible implementation method. It is our intent to encourage further development within the field of computer security and web applications specifically.

1.4 Limitations
In this thesis, we chose to focus our research on a number of intrusion attack methods and these are the ones presented in the central concepts chapter. Other attack methods are not detected in our system due to time constraints.

1.5 Target Audience
This thesis is intended for readers with a computer scientific background and basic knowledge of computer networking and general computer security. It is also recommended that the reader has had previous experience with the development of web applications, and is familiar with the HTTP protocol.

1.6 Outline of Thesis
Chapter 1: Introduces the reader to the problem background, previous work and presents what this thesis will contain.

Chapter 2-3: Present the methodology and the theoretical background required to the implementation and evaluation of this experiment.
Chapter 4-5: Presents the implementation and the evaluation of the proposed system, and its effectiveness in detecting intrusion attempts.

Chapter 6-7: Discusses and presents the conclusion of this thesis, discusses weaknesses, possible improvements and recommends a list of future research.
2 Central Concepts

This chapter presents information about the theoretical background that is required to develop a intrusion detection system. It also lists the attacks that our intrusion detection system detects.

2.1 Intrusion Detection System

In simple words, a intrusion detection system can be described as a device or software that monitors specific activities for policy violations and stores a report of that particular incident. Intrusion detection systems, in most cases, have a number of rules predefined [18, 21].

In the figure, we display how a intrusion detection system might operate inside the web server. A user requests a page and upon request the IDS scans the HTTP header and the user data sent along for malicious content. In case of malicious data, a report is generated and stored for future use. There are two types of intrusion detections:

- Misuse detection is an approach where we define abnormal system behavior primarily and then define any other behavior as normal behavior [18, 21].
- Anomaly detection is the reverse approach where we define normal system behavior and any other behavior as abnormal and thus is considered a potential threat [18, 21].

In this particular thesis, we present a intrusion detection system that detects intrusion attempts using the misuse detection method. The primarily reason, is that most intrusion attempt methods are documented and patterns are easily
identified and the system can be moved to various web applications with no customization.

2.2 Common Web Attacks

2.2.1 SQL Injections

The SQL injection method is one of the most common attack methods, and often consist of short injection strings. Usually, this form of attack method attempts to alter the default working operations on the application by misforming the interpretation of the code. They can pose a great threat, as they are directly targeting the database engine [2].

![Figure 2: A basic authentication form](image)

In the examples, we have an authentication form that is generated and if the content of the input fields were changed to reflect the figure below, the user would attempt to perform a SQL injection attempt.

![Figure 3: A basic authentication form with input data](image)

In web applications that do not utilize a safe approach, and do not sanitize user input, the SQL injection will bypass the authentication process. This is achieved by, altering the standard SQL query and running a logical operator as presented above. The case is explain in-depth on the next page. We make the assumption that the web application has the following query that is used to authenticate an individual and verify that the provided authentication details are correct:
SELECT * FROM members WHERE user = 'Username' AND password = 'Password'

Provided the user input from the previous figure and inserting the input inside the SQL statement, the query will show as:

SELECT * FROM members WHERE user = " OR 1=1-- AND password = "inputdata"

In the statement above, after analyzing, we can see that there is a OR statement that voids the first condition. Thus, a user does not have to exist because the OR statement will void that. Followed by the OR statement are values, interpreted as comments by the database engine, so the password field is of no relevance. The query is therefore interpreted as the following query:

SELECT * FROM members WHERE user = " OR 1=1--

This statement will always yield positive results, because the OR statement voids the previous condition. SQL injections are simple, but can pose a great threat when user input is not sanitized properly.

2.2.2 XSS Injections

Cross site scripting (also known as XSS injections) [4] are common attack methods that are interpreted by the client, and are harmful for web applications. It achieves this by taking advantage of the client scripting, thus, intruders inject malicious javascript strings that are interpreted by the end users web browser:

- Non persistent cross site scripting attacks are attacks that are generated by spreading a link that contains client side scripting code in the URL variables. The unprotected page does not sanitize and when it is returned on the page, the browser interprets the code and it is executed.

- Persistent XSS attacks on the other hand are the opposite where the XSS code is already stored somewhere, usually a database. The code which presents itself somewhere on the web page is executed when the user visits that particular page. This type of attack is often the most problematic method, as it does not require the attacker to spread links and as a result reach a larger target audience.
Cross site scripting injections are not harmful for the server, and the complexity of these attacks vary greatly. Thus, they are often harder to detect than SQL injections. To illustrate how this attack can be used are two case scenarios below.

Case (1)

- The intruder in this case identified a XSS vulnerability on list.php where a GET variable is not being sanitized and the variable is being returned directly to the page after being processed.
- The intruder shares the following URL with victims:
  
  list.php?search=guest<script>alert(“XSS vulnerability”)</script>

- When the client visits the link, upon loading, there will be a popup dialog that pops up. This is not a harmful attack, yet it demonstrates the vulnerability that XSS vulnerabilities pose.

Case (2)

- We have a web application that has two pages and one of the pages is dynamic (communicates with a database) and the other is a simple form where you can enter a few details.
- The intruder fills the form and in one of the inputs he enters the following code:
  
  <script>alert(“XSS vulnerability”)</script>

- Form is processed by the server after the intruder submits it. We make the assumption that there is no XSS protection.
- In the first page (the dynamic page) that retrieves listings from the database; if the user were to browse the page the browser will execute the code.

Persistent cross site scripting attacks are common and the preferred choice when it comes to client side scripting attacks. Usually, they reach a much larger audience because of the fact that it will appear in the pages that retrieve that specific data from the database.

2.2.3 Remote and Local File Inclusion Attacks

This is attacks that may not be as widely used as SQL injections or XSS injections but can potentially be a great security risk for web applications that do not have security in place. These file inclusion attacks work by including files that are determined by an external factor, for instance a URL variable. It is often that developers try to minimize code lines by not using repetitive code one way of doing this is to have one page that takes one parameter to determine what content to display [22].
If we look at the figure seven we can see that the web application relies on the page variable to tell the web application what to present to the user. Normally, in such implementations there is a default action for instance load the primarily index page or such. There are implementations where the server side fails to primarily validate the user input or lacks sufficient server side checks to validate the request.

```php
if(isset($_GET['page'])) {
    include($_GET['page']);
}
```

This is extremely poor practice in general, yet is common across many web applications. This is a huge vulnerability for the web application, and all its services because by having this coded as above we can have both local and external files included (even external server-side code which would be executed on the server). The vulnerability is illustrated clearly when URL is shaped the following way:

  
  Given the implementation above, after processing the content on the evilcode.php file is parsed and interpreted by the web application. This can be used to perform a large number of server tasks, deleting folders, files and pose a great threat.

  
  This is the same, the only difference is that in this intrusion attempt the intruder attempts to parse a local file. This can be used to display passwords, and other sensitive information.

Potentially one of the more severe attack methods that web applications can be exposed to. The danger lies in the ability to inject both local and remote files, even local files may be dangerous as certain web applications may allow file
uploads for instance, and this in itself can then be linked to the file inclusion vulnerability.

2.2.4 Null byte Exploits

A null byte is essentially a character consisting of value zero and is available in almost all languages. What is interesting about this particular character is limited to some languages, for instance in C and program languages derived from C. What it essentially is used for in C is to end strings. This is where a potential threat comes, many web applications today are written in PHP and is one of the preferred languages when it comes to web development. The PHP interpretor is written in C and therefore utilizes the C language [9].

```
$file = $_GET['file'];
require_once("/var/www/$file.php");
```

We analyse the following code snippet above and what it essentially does is request a file that is determined by the file variable. We can also check that because of the way the require method is written the user has no choice to enter the file extension. The statement is predefined and can only include PHP files. But because the PHP interpretor is written in C we can escape this by adding a nullbyte in the end of the file parameter in the URL. Adding a nullbyte is as simple as adding %00 to the end of the URL. In short, to bypass this type of security we can essentially include any file (very much like RFI, LFI attacks). This null byte exploit is essentially derived from the LFI/RFI attacks however has a extra problem that we thought was useful to bring up in this context. As previously stated this type of exploit is only working in C and languages that utilize the C programming language [9].

2.2.5 Hexadecimal Characters

We introduced a few attack methods that intruders tend to attempt intrusions with. A hexadecimal digit is represented using four binary digits and hexadecimal characters range from 0 to 255 in decimal form (the hexadecimal representation would be 0x00 – 0xFF). Hexadecimal digits are used to represent the characters in URLs [1] and due to this certain filters may be evaded by simply using the hex-encoded version of the parameter data. Let us presume that there is a intruder that is using the following URL

```
index.php?page= <script>alert("XSS vulnerability");</script>
```

however the page filters out the data due to the fact that it checks for the script tag. What the intruder might attempt after this unsuccessful attempt is to inject the same attack method using a different approach, and this time taking advantage of the way URLs are encoded and using the counterparts represented in hexadecimal digits. So the URL might end up looking as following:
What this essentially does is wrap the script tags within hexadecimal digits and for naïve implementations or hasty security function implementations the attack may be successful and hence most of the attacks can be successful even if the alphabetical counterparts are being checked [1].

2.2.6 HTTP Header

HTTP headers are one of the key components for HTTP and they are used to transmit and receive data, and can define the operating parameters of an HTTP transaction. There are many unstandardized fields in the HTTP header which are differentiated by a capital X in front of the field name. What is essential to know is that HTTP headers can be altered and thus the web application should never rely on data straight from HTTP headers without checking for harmful data. We will take an example for instance, let us presume our intruder is using a web based proxy that he set up, so in theory his actual IP address will be hidden from us. There is a HTTP field specifically which should be used to transmit the users original IP address however in this case the intruder could specifically alter the HTTP header to essentially include SQL injection. Other fields that spring to mind that may be used specifically on other web platforms are the fields for user agent, referrer. This becomes a potential security risk when people use the data without sanitizing it or validating that it is actually the content you are requesting [5].

2.2.7 CSRF Attacks

Cross site request forgery, or CSRF attacks as they are commonly referred to are simple attacks that are becoming more increasingly common and many large web applications are prone to. CSRF attacks are by definition means of injecting unauthorized commands that the web application trusts. These type of attacks exploit the trust that a web application has for a users web browser. This is a simple attack method which has proven to be very effective, and is often neglected in terms of security. The case scenario below is the best to demonstrate this attack method.

There are a variety of web applications, discussion boards are one type and they allow people to interact with each other using a virtual profile. A profile can describe the person, such as age, location and other personal information. Each attribute has a specific subpage that handles that particular area, and to update your profile location the application would construct the following URL ...

index.php?page=%3c%73%63%72%69%70%74%3ealert("XSSvulnerability");
%3c%2%73%63%72%69%70%74%3c

The attacker, also a member of this board, knows at least two things about the victim in this case: (a) the victim is already authenticated to the forum, (b) the URL structure to submit new location data. Next, the attacker makes a post and masks the link described above with a link label. When a user clicks on this particular link, his update will contain the new location data with no interaction on his part. The attacker will have succeeded, and this particular case scenario is fairly harmless but the same principle be applied on a variety of web applications on different markets.

Preventing this type of attack can be dealt with multiple ways, one is to construct web applications to match the HTTP protocol where GET is defined only to retrieve information, and POST is used to submit data. If most applications followed this, the number of vulnerable web applications would decrease drastically. Other ways to deal with this particular vulnerability is to scan the referrer field which is sent through the HTTP header, generating a secret token using the users information, restricting the lifetime of sessions and cookies.

2.2.8 Previous Research

In our literature review we found that there was previous work in the same field and a variety of ways to design a solution for this problem. In some implementations [17] we found that the entire intrusion detection system takes place after the intrusion attempt takes place by analyzing log files which is not ideal if the intrusion detection system is enhanced with further development and has the ability to communicate with third party services. In such implementations, the detection of intrusion attempts are also limited to the ones found in the web servers logs and as such may not provide as effective results in terms of detecting intrusion attempts. In a paper by Frank [2] he proposes a solution to detect SQL injection attempts using an anomaly detection method which defines what normal usage is. The problem with such an approach is that normal usage can vary from web applications, thus it has a tendency to report false positives. Furthermore, this proposal only detects SQL injection attempts. The most relevant research [3] is an approach that is similar to our implementation and has the ability to detect multiple injection attempts such as SQL/XSS using multiple methods from other scientific papers. This particular proposal only deals with certain type of traffic and that is it only handles web services that are based on protocols such as XML, SOAP, WDSL and UDDI standards. Their proposal has the ability to parse this information and identify any harmful patterns. In our implementation, we scan the entire HTTP header for harmful content and the data is not separated from the header which means that the entire header is safe to be used. Thus, the solution proposed in the paper is specifically oriented for web services whereas our implementation is more of a generic version that can read the entire content as long as the HTTP protocol is being used.
3 Methodology

3.1 Literature Review

Web security is a growing concern as the web is increasingly becoming the primarily source for application deployment, and their complexity is increasing which presents a number of challenges. In a competitive market, time to market is becoming the primarily goal and as a result, web developers may neglect and not fully implement the best security practices. This field has always been a field under constant research but the amount of research is heavily growing over the past few years due to the growth of cloud applications which are deployed on the Internet, thus the web security field has been highlighted more than before and is currently a topic that a lot of research and development is invested in. Intrusion attempts are documented properly and a variety of methods are available to help in patching vulnerabilities within web applications and the majority of the programming languages often have specific methods available to prevent attacks. In other areas, we have application based firewalls that attempt to detect intrusion attempts on a network level and a variety of software based implementations that often deal with a particular set of intrusion attempts. Intrusion detection systems are becoming increasingly popular within this field, as a means to detect intruders and is the area in which this thesis is categorized in. The majority of previous research and development has been in the application layer, where software has been developed to detect and prevent intrusion attempts. Moreover, the modern programming languages have implemented previous research so that a number of methods are available that specifically cleanse data so that it is safe to use. Until today, most of the research has been invested into prevention, but we see that other areas within this field are rapidly growing in terms of research and development. The area of detection as an example, where intrusion detection systems fit, is becoming increasingly popular and this particular field can aid in detecting intrusion attempts and thus prevent damage from being caused. The area of prevention is often to be preferred, as it hinders intrusion attempts, but the majority of web applications are built today rapidly due to time constraints, thus the code may be prone to exploits. The web in particular is a rapidly changing environment with new technologies emerging, and this may open room for more attack methods and this is primarily why this research field will be under constant development. Most methods are documented, but the field is lacking in terms of presenting solutions, specifically hybrid approaches, where detection and prevention is put together to develop something that has the ability to detect, and cleanse it and return it safe. The ideal solution is to add an extra layer, so that the traffic transmitted by the client is scanned for harmful content and cleansed if required. This field will most likely see a slight shift to more unconventional research papers, where more focus will be towards dealing with this problem on a lower level, by either proposing a new architecture for todays web servers where they may handle the
data transmitted and verify whether it is safe to be used by the web application. The future may also introduce more interpreted programming languages, that automatically have built in safety precautions and utilize a hybrid approach where they automatically identify intrusion attempts, and attempt to sanitize the harmful data. Moreover, software that operates in the application layer will most likely be tied better with the local environment, so that intrusion detection systems can make decisions based on a number of factors and lock individuals from accessing the web application. The contribution to the research field in this thesis is primarily to develop a intrusion detection system that detects a group of intrusion attempts and evaluate the effectiveness. Previous work within this, has in most cases, specifically dealt with a specific intrusion attempt and often consisted of alternative implementation methods. Our thesis is more of a further development based on a number of papers published, but making it more effective by integrating detection for a large number of intrusion attempts. Moreover, the majority of intrusion detection systems are types that monitor the entire network which often proves to be ineffective when dealing with web intrusion attempts. The design of our system also differs from previous work, because this intrusion detection system is written as a stand alone web application, that is invoked for each page request by a client. Other implementations most often intercept traffic, whilst our is executed when needed (on-demand execution).

3.2 Software Experiment Environment

In order to conduct our experiment a number of tools were used to construct the intrusion detection system and measure the effectiveness of the system. This section will present the software and the local computer environment and the basic design of the experiment setup.

3.2.1 Computer Hardware

The experiment was conducted on a notebook running the specifications presented below. All the tests conducted were performed under this computer hardware and no other background services were running. The proposed system in this thesis was running on the Windows operating system.

- Windows 7 Home Premium x64
- Intel Core i5-3317U @ 1.70GHz
- 8GB RAM DDR3 1600MHz
- 128GB SSD HD
3.2.2 Virtual Operating System Setup

In order to conduct the tests to answer our research questions, a Linux-based operating system was virtualized and ran simultaneously with the Windows system. It was primarily to act as a regular client and to provide better benchmarking results. Results of the majority of the experiments were plotted using gnuplot. To accomplish the virtualization of the operating system we utilized VirtualBox.

Operating System
- CentOS 6.3
- 1024MB RAM allocated

Installed Extensions
- Apache Benchmark
- Gnu Plot

3.2.3 Software Deployment

In order to deploy the system the software prerequisites must be fulfilled and the configuration file named config.php within the conf folder must be altered to reflect the correct database authentication details. Furthermore, the database dump must be restored and the credentials in the configuration file must reflect the database name, and table.

The final step is to locate the PHP interpreters configuration file, which is named php.ini, and follow the instructions below:

1. Open the file and search for "auto_prepend_file" [7] without the quotes
2. Uncomment by removing the ; from the start of the file
3. Add the path to initIDS which is within the IDS catalog (full path)
4. Save the file and restart the web server

The IDS should now be invoked automatically for each page request that is generated. Enable the debugging functionality to see whether the intrusion detection system is active.

3.2.4 Development Environment

The proposed system in this thesis was developed using PHP as the programming language. In the list below are a number of software and services that were chosen during the implementation phase and are thus required to run the system:
• PHP 5.3.6
• SimpleXML extension for PHP
• Apache 2.2.22
• MySQL 5.5.24

All are running default configurations so that the conducted experiments can be reproduced with the same results. They are running on the Windows server listed in the previous chapter.

3.3 Software Implementation

This section explains our software implementation process and briefly explains the extensions and libraries used to construct our intrusion detection system.

3.3.1 Software Development Methodology

In this thesis we decided to develop our application using the agile development method [19] which is an alternative to traditional development methods. In agile development methods, software is being developed incrementally and work is iterative. The flexibility with this development method lies within the iterative work method that allows developers to respond to changes and this particular development methodology emphasizes on faster delivery. The entire development is split into several sprints that are dates when a specific set of features are expected to be finished and during each sprint the system is reviewed and potential improvements are discussed. This development methodology is iterative and allows greater flexibility than traditional software development methods that often plan most details and document the entire system before any development is actually started. In agile development methods code is also written in a way that changes can occur with minimal code changes and emphasizes on the quality of code. There is often no formal documentation produced, instead the code is produced within the source code. Often test scenarios within a specific set of features are written as real life scenarios where test cases are completed and results are compared to the expected results.

3.3.2 Libraries and Extensions

SimpleXML
It is an extension for the PHP programming language that allows developers to perform basic XML tasks. It can perform basic tasks such as parsing XML data from strings, reading external XML files and altering values.
Google Maps
Google Maps is a map service provided by Google and provides an API for developers to integrate Google Maps with their web applications. It features a powerful API with much flexibility and customization available. It is also possible to populate the map with custom content and pinpoint specific locations.

jQuery
It is a javascript library that is available and allows developers to quickly develop powerful javascript based functionality with less code. It reduces the code needed to handle event listeners, document manipulation, animation, and Ajax requests. It emphasizes on fast development and provides a organized API to handle most web tasks.

ipMapper.js
Is a small extension that asynchronously pinpoints IP addresses in Google Maps only by utilizing the jQuery library. Unlike other extensions of this type, it has the ability to operate with no server side scripting and provides greater flexibility as the computation is done locally.

3.4 Software Experiment Tests
In order to measure the effectiveness of our solution it was required to conduct tests on various areas, thus requiring a number of alternative test methods. Certain tests were conducted several times since parameters altered and it was required to see how the system would react to changes in different scenarios.

3.4.1 Intrusion Detection Effectiveness
The primarily task in this test scenario was to gather a large number of injection strings to be used. In order to conduct this particular test we had to use resources from a variety of Internet sources [11, 15, 20]. From these sources, injection strings were extracted and stored on separate text files. The next step was to utilize the injections strings and inject them using a web browser. Furthermore, the source code of the intrusion detection system was altered to enable the debugging functionality so that the intrusion detection results were printed on the page. Each injection string was used once and injected through a GET parameter. Furthermore, this test scenario was completed in two series, the first test scenario was to determine how effective the intrusion detection system was and potential improvements in terms of the filters that were in place. During this period, filters were modified to patch the vulnerabilities in the detecting algorithm. A second test was completed to see how the filters affected the test and reflect upon the changes. CSRF detection
was made by spoofing the referrer and verifying that the CSRF was detected and printed on the page with the debugging functionality activated.

### 3.4.2 Benchmarking Variables

In order to conduct this test a timer was initialized in the start and was closed once the entire intrusion detection system and the web application had executed. In this test, the web application consisted of a simple Hello World program. This allowed us to benchmark the time required for the intrusion detection system to scan every variable that was sent over with the page request. In this test, the number of page variables sent using the GET varied and for each test the execution time was noted down on a separate text file. As the overall scanning mechanism and the overall structure of the system, similar requests will yield similar results. This makes it a reliable test scenario and should yield identical results for.

![Image]

**Figure 5:** A basic view of the test setup

The overall structure of this test can be seen in the figure above. At the start of the intrusion detection system the timer is initialized and after the system has scanned all data, and the web application has executed the timer is closed and the result is printed. The test was conducted manually using a web browser and the amount of page variables were increased manually. The test scenarios were duplicated to see how the system would run with the intrusion detection system enabled and disabled.

### 3.4.3 Benchmarking Response Times with Concurrent Users

This test was conducted in three series using different parameters to see how the system would react to these changes. The goal of this test is to benchmark how a system of this kind will cope with real world usage and how it scales with concurrent users. This test was primarily conducted using Apache Benchmark which is available with Apache and is built specifically to benchmark web servers. It provides a flexible structure which allows you to conduct benchmark tests in a variety of virtualized scenarios. In our tests we
specifically see how the application scales with concurrent user requests. In this test we take advantage of the following parameters:

**Parameters**


Parameter [n]: This particular parameter is the amount of requests that is sent to the web server. The value entered in this field will control the amount of page requests that is sent to the web server.

Parameter [c]: This particular value is the number of concurrent connections. The value X in this field will result in X multiple requests at the same time.

The results can vary depending on a number of external factors such as the configuration of the web server, processing power available and such. This reason encouraged us to benchmark the same web application with the intrusion detection system disabled to provide more insight as to how the system reacts to a larger number of page requests that are sent simultaneously. Each test result was stored on a local text file and the contents of each text file were plotted using GnuPlot. It is worth noting that this particular benchmark test was performed on the virtualized operating system.

### 3.4.4 Benchmarking Memory Usage

In this test, the following PHP methods (*memory_get_usage()* and *memory_get_peak_usage()* were used and their purpose are to return memory usage details. The first method prints the currently allocated memory (which is the method used in the IDS initialization, and the latter is used to return the peak memory consumption (at the end). These methods will aid in presenting the memory consumption and evaluate the results.

![Figure 6: Memory usage test setup](image-url)
In the figure above we see that the memory usage is first printed once the page request is executed followed by the execution of the intrusion detection system. Once this process has completed the regular Hello World application starts to execute and prints Hello World, and is followed by another memory retrieval that prints the peak usage. This setup allows us to determine the amount of memory allocated for the intrusion detection system and the web application combined. This test was divided into two series, one where the variables by the user contained no harmful data and the other where they contained only harmful injection strings. The reason for this is because incidents are stored in the local memory, thus presenting different results for the parameters. The execution of these tests were automated using a simple JAVA developed program that automatically increased the amount of variables and printed the memory usage before and after the execution. In the next page we present the developed software to perform this particular test.

```java
public static void main(String[] args) throws Exception {
    int urlVars = 2;
    String tempURL = "";
    String contents = "string";
    for(int i = 1; i <= urlVars; i++)
    {
        tempURL = tempURL + "var" + i + "=" + contents + ";";
    }

    URL link = new URL("http://localhost/index.php?" + tempURL);
    BufferedReader in = new BufferedReader(new InputStreamReader(link.openStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null)
    {
        if(inputLine.contains("before:")){
            String[] temp = inputLine.split("<br>");
            temp = temp[0].split(" ");
            System.out.println(Double.parseDouble(temp[1]) / 1024);
        } else if(inputLine.contains("after:")){
            String[] temp = inputLine.split("<br>");
            temp = temp[0].split(" ");
            System.out.println(Double.parseDouble(temp[1]) / 1024);
        }
    }
}
```

Figure 7: Memory usage experiment source code

The contents variable holds the injection string which is replicated X amount times which is controlled by the variable urlVars. This builds the URL string and populates the page request variables with the content defined in contents. Next the software parses the response and checks for specific content within the response, based on the memory usage printed by the application it will store and print this on the console.
4 System Design & Implementation

4.1 System Overview

We can view our system as an extra virtual layer that is integrated between the web server and the web application. All the traffic that flows from the client to the web application has to pass our system. We achieve this by modifying the normal operations of a web server and the interpreter to call our IDS for each HTTP request that is received by the web server. Due to the design of our system, we have the ability to scan the traffic before it is received by the web application.

![Figure 8: Application layers](image)

We can retrieve all the field values from the HTTP header and this is the content that is scheduled to be scanned. In the figure below, a sample of the content available to the intrusion detection system is presented.

```plaintext
array (size=30)
  'HTTP_HOST' => string 'localhost' (length=6)
  'HTTP_USER_AGENT' => string 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' (length=72)
  'HTTP_ACCEPT' => string 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' (length=36)
  'HTTP_ACCEPT_LANGUAGE' => string 'en-US,en;q=0.5' (length=20)
  'HTTP_CONNECTION' => string 'keep-alive' (length=11)
  'HTTP_COOKIE_CONTROL' => string 'false' (length=5)
  'REMOTE_ADDR' => string '192.168.1.101' (length=10)
  'Accept-Encoding' => string 'gzip, deflate' (length=15)
  'Content-Length' => string '0' (length=7)
  'Content-Type' => string 'application/x-www-form-urlencoded' (length=26)
  'User-Agent' => string 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' (length=72)
  'Accept-Language' => string 'en-US,en;q=0.5' (length=20)
  'HTTP_HOST' => string 'localhost' (length=6)
  'HTTP_REFERER' => string 'http://www.example.com' (length=25)
  'HTTP_USER_AGENT' => string 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' (length=72)
  'HTTP_ACCEPT' => string 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' (length=36)
  'HTTP_ACCEPT_LANGUAGE' => string 'en-US,en;q=0.5' (length=20)
  'HTTP_CONNECTION' => string 'keep-alive' (length=11)
  'HTTP_COOKIE_CONTROL' => string 'false' (length=5)
  'REMOTE_ADDR' => string '192.168.1.101' (length=10)
  'Accept-Encoding' => string 'gzip, deflate' (length=15)
  'Content-Length' => string '0' (length=7)
  'Content-Type' => string 'application/x-www-form-urlencoded' (length=26)
  'User-Agent' => string 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' (length=72)
  'Accept-Language' => string 'en-US,en;q=0.5' (length=20)
  'HTTP_HOST' => string 'localhost' (length=6)
  'HTTP_REFERER' => string 'http://www.example.com' (length=25)
  'HTTP_USER_AGENT' => string 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' (length=72)
  'HTTP_ACCEPT' => string 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' (length=36)
  'HTTP_ACCEPT_LANGUAGE' => string 'en-US,en;q=0.5' (length=20)
  'HTTP_CONNECTION' => string 'keep-alive' (length=11)
  'HTTP_COOKIE_CONTROL' => string 'false' (length=5)
  'REMOTE_ADDR' => string '192.168.1.101' (length=10)
  'Accept-Encoding' => string 'gzip, deflate' (length=15)
  'Content-Length' => string '0' (length=7)
  'Content-Type' => string 'application/x-www-form-urlencoded' (length=26)
  'User-Agent' => string 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' (length=72)
  'Accept-Language' => string 'en-US,en;q=0.5' (length=20)
  'HTTP_HOST' => string 'localhost' (length=6)
  'HTTP_REFERER' => string 'http://www.example.com' (length=25)
  'HTTP_USER_AGENT' => string 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' (length=72)
  'HTTP_ACCEPT' => string 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' (length=36)
  'HTTP_ACCEPT_LANGUAGE' => string 'en-US,en;q=0.5' (length=20)
  'HTTP_CONNECTION' => string 'keep-alive' (length=11)
  'HTTP_COOKIE_CONTROL' => string 'false' (length=5)
  'REMOTE_ADDR' => string '192.168.1.101' (length=10)
  'Accept-Encoding' => string 'gzip, deflate' (length=15)
  'Content-Length' => string '0' (length=7)
  'Content-Type' => string 'application/x-www-form-urlencoded' (length=26)
  'User-Agent' => string 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' (length=72)
  'Accept-Language' => string 'en-US,en;q=0.5' (length=20)
```

![Figure 9: HTTP client header information and server side information](image)

We can examine that all the information you need to detect intrusion attempts is there and PHP provides an array that retrieves GET, POST and SESSIONS [3] specifically although what is important to note is that as previously mentioned attacks can appear in many forms, even taking advantage of the HTTP header fields so to properly scan and clean the user data we have to examine the HTTP fields for harmful data.
4.2 IDS Filters

In our system we store the filters in a XML file which is loaded upon program start. Our filters consist of regular expressions which are used to identify harmful patterns that are used to exploit vulnerabilities. In the research phase, we found a list of filters compiled by K.K. Mookhey et al [6]. The problem with the list of filters was that it only detected cross site scripting and SQL injections, other intrusion attempts would pass our system. Furthermore, we found that the regular expressions were outdated and did not provide effective detection results. Therefore, certain rules were adjusted and redesigned to improve the effectiveness. Filters are formulated as regular expressions, and a sample filter is provided below:

```
/((\%3C)|<)((\%69)|i|((\%49)) ((\%6D)|m|((\%4D))) ((\%67)|g|((\%47))[^\n]+((\%3E)|>)/i
```

We can start by analyzing the regular expression above as this is one of the regular expressions that was published in that article. At the beginning and the end we have a delimiter. We will split the regular expression into several parts to simplify the process.

- `((\%3C)|<)` - It looks for the starting bracket and its hex equivalent
- `((\%69)|i|((\%49))` - It looks for i and the hex equivalent of i, I.
- `((\%6D)|m|((\%4D)))` - It looks for m and the hex equivalent of m, M.
- `((\%67)|g|((\%47))` - It looks for g and the hex equivalent of g, G.
- `[^\n]+` - Anything but a new line following the previous statements
- `((\%3E)|>)` - Closing bracket and its hex equivalent

What we can gather from the regular expression above is that it will protect us from img attacks containing some additional field. It checks for both capital and regular letters so it can not be evaded by using capital letters. Thus, attacks using this particular pattern will be detected by the system.
4.3 Application Structure

![Class diagram to demonstrate connections between classes](image)

Figure 10: Class diagram to demonstrate connections between classes

4.3.1 Controller

The controller class is the class used to initialize and responsible for handling the program flow. It creates the necessary objects and binds classes together for communication between classes. Initially, the class will create references to the following classes Detection, Rules, Scan, DB. Object reference creation is done within the application's constructor. A method can also be found within the class which calls another method in the (Scan) class.

4.3.2 Database

The database class is responsible for the communication between the application and the database. It is used to store and retrieve data from the database and is utilizing prepared statements to query the database which provides good protection against SQL injections. The class has methods to specifically retrieve, insert incidents and relevant information.

4.3.3 Detection

Detection class is responsible for temporarily storing information regarding attempted intrusions in the form of an array. It has a two dimensional array where attempted intrusions are stored to be processed later by the system. The class has specific methods to add a new incident to the array, retrieve and update the array. It makes calls to the static class (Functions) for certain data.
4.3.4 Functions

A static class which only provides the system common functions to reduce and minimize repetitive code. It provides a number of methods available to fetch data that is related to the intruder, and methods to retrieve data about time, date in specific forms. It also has data sanitization methods to clean data from harmful data.

4.3.5 Rules

Class which handles the rules for the intrusion detection system. By definition the rules are the regular expressions used to identify harmful data patterns. When a class object is created the main constructor will automatically preload an external XML file containing the filters for the intrusion detection system. It has methods to retrieve specific ruleset information or the rules all together.

4.3.6 Scan

One of the most vital classes which contains the control logic of the software. It is the class that essentially scans for harmful data. It retrieves data from two class references and these are Detection, Rules. It retrieves the rules in order to analyze for harmful data, and in case a match is found it utilizes the (Detection) class to store the incident.

4.4 Database Design

Our database table structure is simple and straightforward because we do not actually store that much information a two table design was more than sufficient to store the data needed for each intrusion attempt.

![Entity relationship diagram](image)

Figure 11: Entity relationship diagram

The main reason the database was designed this way rather than storing everything in a single table was that should there be a need to further expand the system functionality it can be added with ease. In this particular design adding another table is fairly easy and it follows the normalization standards whilst a single table may prove to be more problematic in the future. In this
design, we store the intruders IP address in the intruder table and the attempted intrusions in the incident table. In the incident table we store information such as the date and time of the incident, description which is fetched from the XML filters file and last we have the attempted intrusion method which presents the way the user attempted to break into the system.

4.5 Program Flow

![Activity diagram over the program](image)

At program launch the system will import the objects and the Controller class will initialize object references to the other classes. After, the scanning algorithm is launched and scans the content iteratively, by scanning each variable at one time until every variable has been scanned. In case of a intrusion attempt detected, a call is made during the scanning mechanism to an external function that stores the incident temporarily on the computer memory. The remaining operation is to store the incidents in the database and this is done iteratively. This is done by iteratively going through the array which is built dynamically and making a call to the Database class which handles all the database queries.
4.6 XML File Structure

As previously stated our filters for the intrusion detection system are stored in a XML file that allows the users to simply add more, or re-define existing filters to better suit their needs. The main design criteria, was that we needed to find a good way to organize the amount of filters so that additions could be made with little effort. A sample of the XML file is presented below:

```xml
<rules>
  <rule>
    <id>1</id>
    <tag><![CDATA[ Rule filter for the IDS ]]> </tag>
    <description> Rule description </description>
  </rule>
</rules>
```

This is essentially how our XML file is defined, a rule is a part of the global rules and to a specific rule there are multiple attributes. Rules are other words for filters, and below are a description of the content for each attribute:

- **Rule ID**
  This particular attribute is a identifier for a specific rule, which is used to retrieve relevant information about the specific ruleset.

- **Rule Tag**
  This attribute is used to define the IDS filter, so in short it is the actual filter for that specific vulnerability/intrusion attempt. It is wrapped in a character data section specifically to avoid the parse to parse the XML because of certain illegal characters in XML.

- **Rule Description**
  This attribute contains the description of the filter, what type of injection attempt it detects.

4.7 Administration Panel

4.7.1 Overview

In addition to the core functionality of the intrusion detection system we needed to design and implement an extension for the administrator to access, browse the recorded incidents. The panel supports the following features:

- Ability to list the most recent intrusions in the front page in tabular form. It lists information such as the intruders IP address, date, description and attempt intrusion method.
• Ability to list intruders by IP addresses so that you can look a specific IP address in tabular form. It will also list the amount of intrusion attempts committed by that IP address.

• It will demonstrate graphically (dynamically created graphs) the amount of intrusions were made by that intruder compared to total logged incidents. The graph is constructed dynamically and displays the percentages in a pie chart. [8]

• Present a pinpointed location on Google Maps [14] using an extension to display the location of that specific intruder. The location is marked and provides a number of details such as country, region, city, longitude and latitude etc.

4.7.2 Implementation

We used a variety of languages to develop the front end of the administration panel such as XHTML 1.0 Strict, JavaScript, jQuery [12] framework and other extensions. It was built using the three-layer-structure so that it is divided into several layers, where important parts such as the HTML, the scripting files, and the design stylesheet are separated. Additionally, we use a specific identifier to call the appropriate file based on a URL variable so that we minimize and reduce repetitive code. Below are a few figures to display our end result of the administration panel:

![Figure 13: Result of the administration panel](image)

It was divided into several subpages that are called upon request, so for instance the multiple subpages have a specific PHP file that is called when that particular file requests it. The design of the system allows developers to easily add more subpages and functionality with minimal lines of code. It is divided into two sections, the header and the main content area. The header is the same for all subpages and depending on what subpage the main content area can be different. We will demonstrate this by using a simple example, for a client that opens the administration panel and has no additional URL variables we default the page to call the index subpage. This is done by including a file within a specific folder which contains all the subpages server side code. The
code mainly consists of pure PHP code with HTML front-end formatting made dynamically using PHP. This file is then ran and the content is displayed for the client.

![File Structure Diagram]

Figure 14: A basic illustration of the file structure

In the figure you can clearly see the overall file structure and the way the administration panel is written.

4.7.3 Sorting

One of the core features within the administration panel is the search and filter functionality, a crucial feature that was developed specifically with performance in mind. What we essentially do instead of querying the database multiple times is that we allow the administrator to filter the incidents based on criterias using a input field which then looks for table rows that contain that particular data. It was developed using JavaScript and utilizing the jQuery framework and it uses a event handler to monitor when that particular input field has had a new character entered and then proceeds with the filter. It is worth noting that it only hides the other fields, so that should you remove the input from the field it will reset and display all the rows again. It works dynamically because it utilizes client side scripting language, so there is no overload on the server which is to be preferred as we wanted to reduce the number of database queries.
5 System Impact & Analysis

5.1 Efficiency

5.1.1 Detected Intrusion Attempts

This particular test was primarily conducted to see how effective the intrusion detection system is when it is under attack. Intrusion detection systems need to be tested in order to verify that they detect the majority of the intrusion attempts, under ideal circumstances, it should detect almost all attacks. The injection strings used in this attempt can be found in the appendix.

Test 1

![Figure 15: Results of the first test conducted](image)

In the first test, the majority of the attacks were detected however they demonstrate potential flaws in the filters. In the field of SQL injections, we noticed that a number of SQL commands were not in the current ruleset, so the intruder would be able to bypass the current ruleset. A number of attacks were utilizing specific commands available on various database engines, an example is the concat function which is used to concatenate multiple strings and return the concatenating string. In XSS injections, it was a variety of reasons but even here there were a number of factors that caused the detection rate to go down. First, the ruleset at this test scenario was designed so that all cross site scripting attacks were wrapped inside script tags but the problem is that XSS attacks can be injected through alternative ways without wrapping the code in script tags. There was a gap in the theoretical framework, and a number of filters were redesigned to match other conditions and not necessarily check for the script tags primarily. On the other hand, event handlers were not
inside the ruleset and a good way to detect intrusion attempts is to check for specific event handlers. The result, was an increase of the filters and more checks. Once these changes had been implemented, a new test scenario was conducted using the exact same injection strings to verify what impact they had on the overall results.

**Test 2**

![Figure 16: Results of the second test conducted](image)

In this test, we can see that the detection rates improved drastically from the changes. Furthermore, almost all cross site scripting intrusion attempts are detected. On the contrary, the SQL injections are better than before but there is room for potential improvement and future work. The results of this test has proven that an intrusion detection system, is able to detect almost every SQL/XSS injection with great success.

**Misc intrusion attempts**

The problem with certain intrusion attempts is that they are often restrictive and are not able to be tested as above. Therefore, we decided that certain intrusion attempts such as CSRF will be tested manually to verify that the protection is there and provides sufficient protection. In this particular test, the remaining intrusion attempts were combined and tested simultaneously with a few test scenarios. In a test scenario, we tested the systems detection rate when attempting to exploit local and remote file inclusion methods. All attempts were detected in the intrusion detection system. CSRF also proved to be reliable, the way it is implemented is that, it will detect any potential cross site request forgery when a link on an external site is clicked and a page request is initialized. Under our testing, we found the implementation to work reliably although there are a number of vulnerabilities with this implementation. Given
the test scenario in the chapter that presents CSRF forgery, the referrer header will be the same since the injection takes place on the same discussion board; at this stage, the intrusion detection system is helpless and cannot detect intrusion attempts. It only provides, with the current implementation, under circumstances where the individual is sent from an external web site.

5.2 Performance

5.2.1 Benchmarking Variables

In this particular test the goal of the study is to conduct how the execution time grows with use of the intrusion detection system. The primarily test was to examine how the loading time for a single request would affect the performance of the web application. In this particular benchmark we see that the execution time for the web application when the intrusion detection system is not in use is unaffected. On the contrary, the results for the intrusion detection system when it is active are completely dependent on the amount of parameters that are scheduled to be scanned by the intrusion detection system. This is because of the way the system works, and all parameters are treated as security vulnerabilities so each of them are scanned for harmful content.

Figure 17: Results of the benchmark evaluation
5.2.2 Benchmarking Concurrent Requests

In this particular study we used a fixed number of URL parameters and one of these parameters contained harmful data. The system recorded the incident.

Test 1

![Graph](image1.png)

Figure 18: Results of test one in benchmarking concurrent requests

**Total Requests**: 1000, **Concurrent Requests**: 100

In this particular study we found that at most the response time spiked at 1003 ms which is about a second delay for a regular visitor. In this particular test the first 900 requests are fairly close only taking approximately 200 ms more. The response time grew larger as the number of requests increased and the web server's CPU usage and memory continued to grow due to default web server configuration.

Test 2

![Graph](image2.png)

Figure 19: Results of test two in benchmarking concurrent requests
**Total Requests:** 1000, **Concurrent Requests:** 25

The test varies from the previous test by reducing the concurrent requests to a smaller amount and the load of the web server was reduced slightly and this is displayed in this diagram. At request number 100 we see an increase of the response time by 25 ms which is acceptable. We continue to see the same pattern from the previous tests and the response times grow with the number of increasing requests due to poor web server configuration. Compared to the previous test we can see that for the last request the response time differed only by approximately 130 ms which in the previous test differed by 400 ms.

**Test 3**

![Graph showing response times for test 3](image)

Figure 20: Results of test three in benchmarking concurrent requests

**Total Requests:** 1000, **Concurrent Requests:** 1

In this benchmark we see how the response times differ for a single request which is repeated 1000 times. Throughout the graph we see that the differences are closer to the ones presented in the previous graphs and this is due to the web server being able to handle a single request better than concurrent requests due to the poor web server configuration. We can see that response times are affected from about 10 ms to about 23 ms which should not have large impact on the response times for regular web applications.
5.2.3 Benchmarking Memory Usage

In this particular section we split the benchmark into two phases, the first phase was to investigate the memory usage for multiple variables with no harmful data in the contents of the variables. In the second phase we replaced the contents of the variables with harmful data to see how the memory usage differed.

Test 1

![Memory Benchmark Graph](image)

Figure 21: Results of test one in measuring memory usage

In this particular test we can see that the amount of memory allocated increases significantly with the use of the intrusion detection system. In the graph above the values are represented in kilobytes, and in our benchmark we found that all test scenarios use less than one megabyte memory allocated for the execution. Across the tests we also see that whilst the memory usage increased largely with the use of our intrusion detection system it is a stable value and does not change heavily so the memory usage stays within the same range. The reason that the memory usage remains the same no matter the parameters is simply because when there is no intrusion attempts detected nothing is stored in the memory. The increase reflected in this chart is because of the program initializing arrays and allocating space in the computer memory. At the first test using two variables the memory usage was at 795 kilobyte, and in the last test using 32 variables the memory usage had increased to 796 kb which is a increase of a single kilobyte.
Test 2

Figure 22: Results of test two in measuring memory usage

In this particular test the variables were filled with harmful data, thus the amount of parameters is the amount of intrusion attempts that will be detected by the system. In the graph the memory usage is presented and compared to the previous test the IDS memory usage is increased on all tests which is to be expected due to storing the intrusion attempts in the computer memory. For instance in the last test using 32 variables (and all of them contain harmful data) it is stored 32 times in the array since the IDS treats them as separate incidents. Even at the worst case scenario, the memory usage is below one megabyte which is acceptable but further optimization can be achieved in the source code. In other instances, where the IDS is inactive the memory usage continues to be the same and whilst there is more memory consumption it should not affect regular web applications.
6 Discussion

6.1 Effectiveness
Intrusion attempts vary depending on multiple factors and range from basic to more complex methods and this presents a number of challenges when identifying security threats. The same intrusion string can be designed in multiple ways, with use of hexadecimal characters and other character encoding techniques as was described in the previous chapters thus identifying characters can potentially pose problems and a large number of filters can affect the overall performance of the system. In the test conducted on the detection rate for the intrusion detection system it was found a large proportion of the intrusion attempts were detected and the results varied from 90% up to 100% which are good values. Certain intrusion attempts were specifically targeted for specific software, and in particular cross site scripting attack methods that on certain cases only affected specific web browser versions. It was demonstrated that a system of this nature can provide a good extra layer to the general security of web applications and identify most intrusion attempts. The system was built by defining abnormal traffic and the overall effectiveness of the system was demonstrated to maintain good detection rates. The effectiveness of the system can be improved additionally by adding or altering the existing filters for the intrusion detection systems but this may affect other factors such as performance constraints, number of false positives. On a broad scale the system can be used to identify security threats prior to execution by the web applications and as future development it may be viable to consider integration with the local operating system so that frequent offenders can be blocked. In closer integrated environments with third party services such as firewalls or operating systems it would be possible to block intruders before the intrusion attempt hits the web application which could help to solve a critical problem. Modern web applications often interact with databases to store information, APIs available thus making them the ideal target for hackers since they would only have to find a vulnerability within the web application to potentially retrieve access to other services. The implementation in this thesis has room for improvement, and CSRF attacks specifically can benefit from a more complex algorithmic design as the current design will not detect attempts that are made from one web application. The system could benefit from introducing more complex solutions to detect this form of attack.

6.2 Performance
In the performance benchmarks our tests clearly indicate that a system of this nature can be effective whilst having a low impact on response times and performance. A number of stress tests were conducted to see how the system would react to a larger amount of requests using concurrent users to see if the system would come to a point where it was unable to handle the traffic and in

35
all our tests we found that the system was able to handle the load and changes in the performance charts were often impacted by the web server configuration. The system was able to execute properly and had a extremely small footprint both in terms of memory usage and from a client to server standpoint it had a minor impact on the response times. Potential risks with this system is the storage choice where incidents are logged on a database. Depending on the load of the storage server the response times would be greatly affected by the database server's response times for executing queries and this may pose a problem for a system of this nature. Due to the design and the fact that the IDS is executed prior to the web application any delays for processing database queries will affect the response times and a potential method is to find an alternate storage method by storing the incidents locally in form of text files. The fewer number of external constraints should be preferable for a system of this nature on crowded applications where the database server may be under high load. Overall from the test results gathered from the charts the system had a small impact on the response times making it ideal for web applications of different sizes in terms of popularity. More active web applications can also benefit from having an intrusion detection system deployed and it is suitable for all type of web applications. The system can benefit more from optimisation and potentially yield better results for the results in the charts. In the test conducted in chapter 5.2.1 we see that there is a significantly higher value for the IDS even when the URL contains no parameters and this is because the system is initializing the program flow even though there should be a conditional statement to check that there is actually data to scan. A more suitable approach would be to first identify whether there is a need to scan and then based on the result of that determine whether the program flow should initialize or not.

6.3 Administration

Web applications come as a variety of applications and are often constructed and maintained by either single individuals, companies or organisations. A thing they have in common is that all web applications require maintenance to some extent and under successful intrusions where the intruder managed to bypass the security of the web application this presents larger difficulties. It is normal that the web application is taken offline to deal with this issue and this often includes reading the logs to see whether there is any suspicious data and this process is time consuming. Intrusion detection systems present a solution to this particular problem as they log the intrusion attempts which makes the detection process easy for the administration. In our implementation we log data about the specific file that was served by the web server, and its parameters which should indicate what parameters were injected with harmful data and the particular file where the injection took place. Companies can benefit greatly from an intrusion detection system due to the sole reason that
they log every incident and thus isolating the security vulnerabilities within applications can be done within a much shorter time frame and as a result prevent income losses.

Overall the results have demonstrated that it is possible to construct a reliable intrusion detection system which detects most attacks and have a minor impact on performance. Furthermore it was also demonstrated that detection of attack methods can be by either using strict filters or loose filters. Strict filters check for specific strings and have often many possible combinations declared whilst loose filters generally contain much less details. Strict filters have a tendency to report less false positives and its counterpart has a tendency to report more but can potentially detect even the most complex of attack methods. Choice of method is depending on the type of web application and certain web applications might benefit from loosely based written filters.
7 Conclusions and Future Work

In this Bachelor thesis, we showed that it is possible to construct a software based intrusion detection system and detect intrusion attempts with a very high detection rate. In the results, we can see that the detection rates for the proposed system are high and the system itself should have a minor performance impact.

7.1 Future Work

In this particular section we recommend future work which can enhance our systems functionality and potentially yield better results.

7.1.1 Intrusion Prevention Enhancement

Intrusion prevention systems are generally used to prevent intrusion attempts rather than to detect them. They are often closely tied with intrusion detection systems and our system could benefit greatly by implementing a IPS system. A potential way to do this is to sanitize the HTTP headers, so that the web applications would not be vulnerable to harmful data. Enhancing the systems functionality with this particular feature would combat the majority of intrusion attempts and provide an extra layer of security.

7.1.2 Local Environment

A potential enhancement to this system is to tie it closer to the local operating system so that there are multiple functions available. The data is stored on a local database server so the data available for specific intruders are available. Enhancing the system with closer operating system ties would ensure the ability to suspend particular IP addresses from accessing the web applications. Alternatively, this can be integrated on a web server level by utilizing the functions of a web server.

7.1.3 Storage Alternative

The system utilizes a database server to store intrusion attempts and for more active web applications the performance aspects may have a larger impact. The system should therefore be rewritten to at least support file storage. Incidents would instead be stored on a local folder with text files for each day. This may potentially yield better performance results and reduce the impact on response times.
7.1.4 Performance

The intrusion detection system in this thesis is written as a web application and is executed for each request. The majority of the web servers available support custom modules to execute for each user request, and a potential enhancement and a interesting research area would be to transfer this to another platform, and rewrite the IDS in C and compile it as a web server module. It would be interesting to see how the performance aspects of the system would differ.
Bibliography


[16] PHP, PHP Request, 


[21] Wikipedia, Intrusion detection systems, 