UiO **Department of Informatics**
University of Oslo

Security Regression Testing Framework for Web Application Development

Usman Waheed

Master's Thesis Autumn 2014

Security Regression Testing Framework for Web Application Development

Usman Waheed

December 12, 2014

Abstract

Securing web applications from malicious intruders is high priority for many companies and organizations. The prime reasons to protect your assets on the World Wide Web is to deter internet identity theft, minimize monetary loss and most importantly to keep the trust of your clients who transact through your web portals and services [43]. Web developers should incorporate security testing in the Software Development Life-cycle (SDLC) [34] to catch the flaws early. Security testing of web applications can be tedious because there are numerous input attack vectors which, if executed manually, will be time consuming and less efficient. To make this process efficient, this paper proposes a framework based on Open Source Software Tools that can be jigsawed together to achieve automation, regression testing and catching security flaws early in the software development cycle. Identifying the security flaws early will circumvent embarrassment imposed by hackers, loss of credibility in the business community and provide protection against some of the very basic security principles that are overlooked during the development of web applications. The idea is to create awareness about web applications security testing and let it traverse through your software development centers in order to create high quality products that are not just robust but also secure.

Acknowledgements

I would like to Thank Opera Software ASA / Oslo - Norway for giving me the opportunity to conduct this research. The support given to me by my employer and specifically my ex-manager Espen Myhre at Opera Software ASA deserves the first gratitude. I would also like to thank the senior management of Opera Software ASA who gave me permission to submit my work on Web Application Security Regression Testing at Opera as a master thesis topic. [1].

Next, I would like to thank all my professors at UiO, HiOA and specifically my thesis supervisor Andrew Seely for his valuable time, guidance and support.

Last but not the least, my family and friends for their love and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer and Internet Security has many layers. It starts at client interfaces, through proxies in the middle and finally the back-end servers where information gets processed and returned to the interacting users. The network connects these various pieces together in order for data to flow back and forth. All these pieces including the transport medium needs to be secure from hacking, exploits, information and identity theft. This paper focuses on web applications and their security.

Web applications are autonomous pieces of software that interact with users or clients in the Internet. When you purchase a travel package to an exotic destination using a website, you as the user transact with software in the Internet by providing both personal (name, date of birth, etc) and financial credentials (credit card numbers, online banking, etc). Security in web applications is the aspect that ensures your information is not exploited by others, transactions you commit are safe and the trust between the users and services in the Internet are upheld and maintained.

The waterfall [11] model is a sequential process comprising of five stages. It starts with gathering requirements, followed by design, then development, unit functional testing and the final implementation.

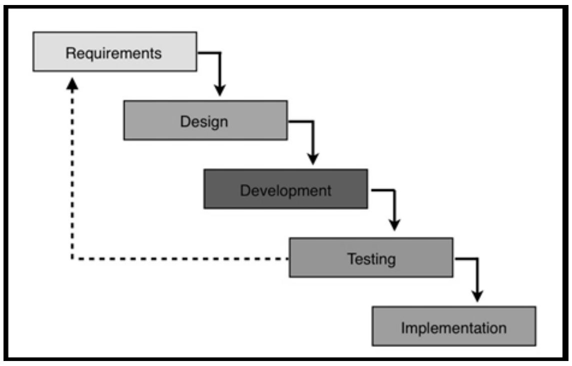


Figure 1.1: Software Development Life-cycle (SDLC) [2]

During the test phase it is possible that the requirements change which will result in an iteration of the process. Where does security testing fall in this apparatus? Security testing of web applications should become an early part of this process so that the final product is at least immune to trivial vulnerabilities. The Security Regression Testing Framework approach will add value by improving the quality of the product and addressing security concerns early during the design and development stages. Security testing can run in parallel during the design and development phases instead of being a separate additional step. Often websites are published to the public without much concern for security and inadequate security testing will leave security holes in web properties. Sometimes an entire website can become compromised via a trivial attack vector which could have been detected and prevented if security testing was incorporated early in the process.

"Web applications change and are upgraded frequently due to security attacks, feature updates, or user preference changes. These fixes often involve small patches or revisions, but still, testers need to perform regression testing on their products to ensure that the changes have not introduced new faults" [26]. Security regression testing plays a vital role in the software development process because code changes are committed frequently to apply fixes, patches or even feature enhancements by developers. Whenever new code is pushed or existing code is modified a series of security test cases should execute to ensure that all cases which passed previously continue to do so and no new security bugs got introduced. This is the concept of security regression testing which is expensive in both time and labor. By automating security regression testing which is what this paper discusses, one can reduce cost and close the release gaps.

Data and metrics are required to measure the state of web applications security. According to the website Security Statistics Report (May2013) [41] one of the key performance indicators for measuring website security is "Window-of-Exposure". By definition: Window-of-Exposure is the number of days in a year a website is exposed to at least one serious vulnerability [41]. According to the horizontal stacked bar chart in Figure 1.2 noted below, 33% of all websites across all categories (Banking, Education, IT, Insurance, etc) are always vulnerable to at least one serious vulnerability in one full year. For the retail industry this Key Performance Indicator (KPI) stands at 54%, which is alarming because online commerce has become more prevalent over the years.

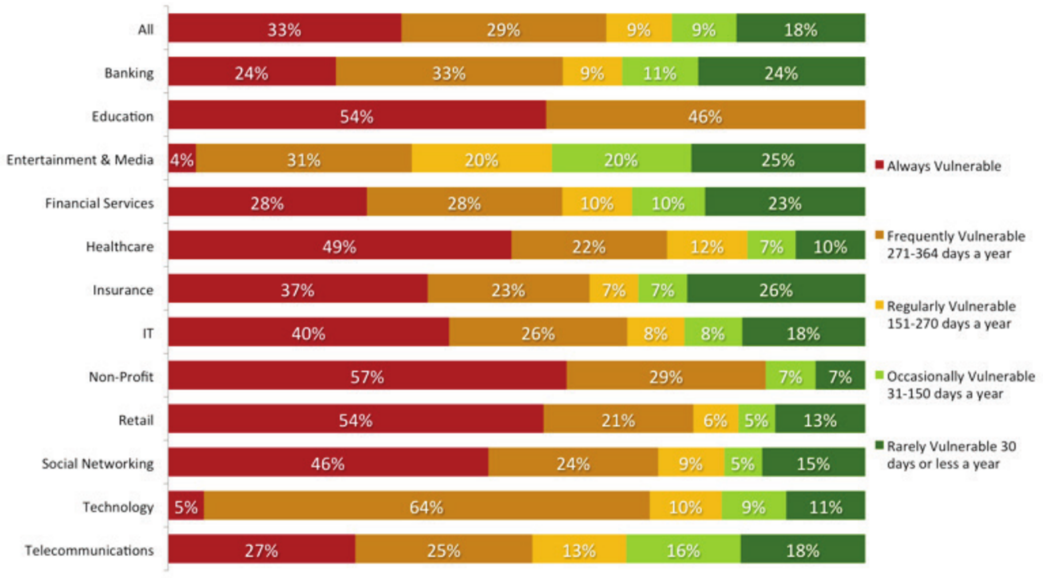


Figure 1.2: Overall Window Of Exposure To Serious Vulnerabilities 2012 [41]

There are various kinds of vulnerabilities where each is composed of different variations in attack vectors. The Open Web Application Security Project [36] top 10 for 2013 will be introduced in more detail in a latter section but some key vectors are mentioned below:

- Cross Site Scripting (XSS)
- Information Leakage
- Content Spoofing
- Cross Site Request Forgery (CSRF)
- Brute Force
- Insufficient Transport Layer Protection
- Insufficient Authorization
- SQL-Injection (SQLi)

WhiteHat Security published in the same report mentioned above statistics that illustrate prevalence by class in overall vulnerability population [41]. This distribution can change yearly so the list is not static. According to this report "Cross Site Scripting" accounts for 43% of the overall vulnerability population which means web applications are not sanitizing inputs properly. Content Spoofing stands at 11%, followed by Information Leakage, Cross-Site Request Forgery and so forth. The results are depicted in Figure 1.3.

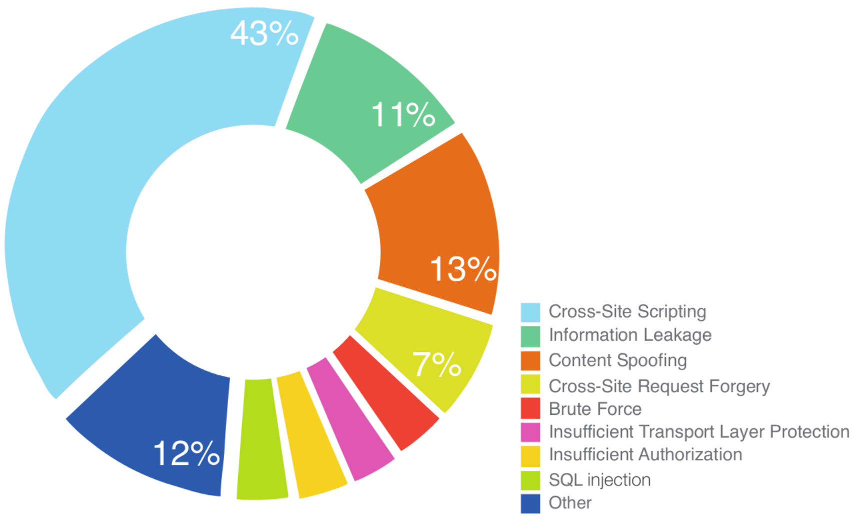


Figure 1.3: Overall Vulnerability Population 2012. Percentage Breakdown Of All Serious Vulnerabilities Discovered [41]

According to the global internet report for 2014 [42], over 3 billion users will be online accessing the Internet by 2015. The growth rate will triple since September 2013. This means the number of web application transactions will also increase and the need for websites to be secure will become more important. Automated Security Regression Testing of web applications is part of a process that uses a framework of open source tools to achieve the objective to publish secure web properties. Hackers and users with malicious intent are looking to exploit holes in websites for personal gains or whatever reasons. This paper will demonstrate how you can successfully fit security regression testing within a software development process in order to produce good quality Internet products and services. You cannot be 100% secure but you can take the right steps to have an early warning system against some of the well-known exploits and security holes that exist out there. At times you are dependent on the tools in the framework that fall short because of lack of functionality and support but at the same time it opens up more avenues to explore tools you have not considered yet. Web Applications Security is a process that will need to evolve over time in order to keep up to date against sophisticated attacks. This is a never ending journey that requires us to be more proactive and resilient.

1.1 Motivation

This research was inspired by testing Opera's web properties for security holes and perform quality assurance. Opera had numerous websites which were public with continuous development and feature enhancements done by software developers working on these web properties. Security testing was a manual task which was tedious. All security flaws were discovered and reported through this manual process which was time consuming. The need was to research and investigate for a more efficient solution that could perform security regression testing using automation.

The idea was to setup a framework that could become part of the software development life-cycle in our department at Opera. One additional goal was to create web application security awareness within Opera's web developer work-force. At the time and even now there was a lot of ongoing cybercrime activity and Opera took the initiative to protect and secure its web properties from malicious intruders.

1.2 Problem Statement

Web applications security is a process that needs to be part of the SDLC of an organization in order to close trivial security holes in the early stages of software development. This process is explained by constructing a framework using Open Source Software Tools. The research conducted for this paper will address the following questions:

1. Why should security testing of web applications become a part of the SDLC?
2. What are the advantages of building an automated security regression test setup?
3. Which open source tools can be utilized to construct a framework to achieve this automation?
4. What are the measurable benefits of this process?

1.3 Thesis Structure

This thesis is organized into seven chapters. Chapter 1 provides an introduction to the topic, author's motivation and problem statement. Chapter 2 provides a more detailed explanation on the background and prior work in the field of web applications security. It discusses the foundation that made it possible to construct an automated security regression test framework. Chapter 3 explains the approach and open source software tools that can be used to construct the setup. Test cases are also outlined in this section. Chapter 4 covers the results from the test experiments. The results are analyzed in chapter 5. Chapter 6 is a discussion on the benefits and advantages of testing your web application for security holes early in the SDLC. In this chapter we also look at future work and possible extension to the proposed open source framework. Chapter 7 is the author's conclusion of the topic and summarizes what we achieve from creating early awareness about web applications security. Supporting sections include the references used for this research paper, appendix and documentation on the framework setup.

Chapter 2

Background

2.1 Three Tier Architecture

A 3-tier client-server application consists of a front-end client (e.g. browsers), middle-tier application servers (e.g. web servers) where the business logic runs, and the back-end databases [13]. The front-end is where the user interacts by submitting inputs, the middle-tier applies the business logic on the requests it receives and databases are used to store information. The figure below depicts a 3 tier application.

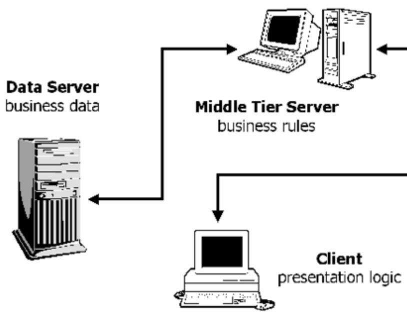


Figure 2.1: Three Tier Architecture [37]

From a web application security perspective you can apply testing to these three tiers separately. The focus of this paper is on the middle-tier web servers and back-end databases. The primary reason we will not focus on client-side validation is that we assume that all kinds of inputs both valid and tainted can be injected by clients. Client side form validation is usually done using JavaScript but there are ways to go around it. Our goal is to show how in a process you can perform robust security testing for the middle and back-end tiers.

2.2 WebApp Anatomy

Web Applications can be built with various technologies and programming languages. There are two paradigms, imperative (such as C, Java, and JavaScript) and declarative (such as CSS, HTML, and SQL). Declarative languages specify what a program should do rather than the intricacies of the algorithms needed to achieve desired results. Imperative languages are harder to use and understand but have more expressive power. When you want to define the structure, styling of a web page or managing data stored in a relational database you are using a more declarative language that is solving domain-specific problems, while imperative languages are more general purpose and applicable for solving a variety of problems [45].

Website development requires a combination of both imperative and declarative languages as technology has evolved so rapidly [25]. The point to note is that during the development of web applications regardless of which programming languages you choose it is important to identify injection points within your web applications. Injection points are basically entry doors from where client requests come through with a payload. A payload is the actual data the user has sent to the middle-tier and will be processed by the web server. This information in the payload can further be used to communicate with a database or other sub-routines in the business logic layer. Web application developers should check and sanitize these inputs. It is through these user inputs that malicious attacks are initiated which can exploit either flaw in the business logic or a vulnerability in the underlying software libraries being used by the web application.

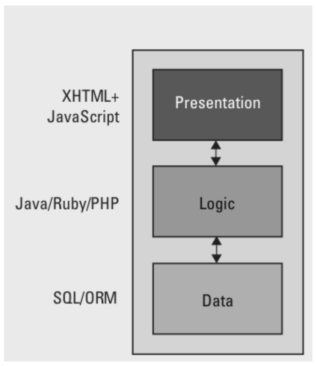


Figure 2.2: Web Application Anatomy [45]

Building secure web applications is an iterative step in the software development process. Sometimes developers will place code and data on the client side in order to improve interactive performance [48]. This is done because web apps are client-server applications so there is a latency cost associated when you send requests to the middle and back-ends for processing. We know cookies which are widely used to track user sessions, are stored on the client side browser. Later we will see an example of session hijacking where a hacker might try to capture a user's session cookie information to gain access to their information (e.g. online banking). It is good practice to always deploy security-critical code and data on the server and back-ends [48] rather than on the client side.

Running automated security regression tests against a three tier web application is not trivial. We have to assume that the user can submit both a mix of good and bad data inputs. The security testing process should account for such scenarios and perform validation checks to ensure that the data injected is sanitized properly on the server side.

2.3 Classes Of Vulnerabilities

Recent study conducted by the SANS Institute estimates that up to 60% of Internet attacks target web applications [22]. The complexity of the attacks have not changed much given that many of the web problems are simple in nature. Organizations like MITRE [8], SANS [18] and OWASP [36] have helped to create web application security awareness through their programs but the average web developer seems either unaware of classes of vulnerabilities or does not know how to protect against exploits effectively [22]. This section explains classes of vulnerabilities and some common mistakes web application developers make during design and implementation.

2.3.1 Cross Site Scripting - XSS

In a reflected XSS attack the victim is lured into sending malicious code to the trusted site and the trusted server will echo back the clients input at the clients browser. For example, take the hyperlink noted below which could be sent in the body of an email message to a potential victim who is unaware and proceeds to click on it. The JavaScript script tag passed to the comment.cgi script via the mycomment variable will get executed within the page from the trusted server [14] [46]. This "badfile" could try to steal the victim's session cookies.

```
<A HREF="http://example.com/comment.cgi?mycomment=
<SCRIPT SRC='http://attacker.org/badfile'></SCRIPT>">;Click Here</A>
```

Bulletin board applications are candidates for stored XSS. In these types of attacks malicious code is placed at the trusted site by the attacker [14] [46]. For example, a victim will click on a hyperlink on a bulletin board website and it will execute the stored XSS which in turn will try to access the victim's session cookie.

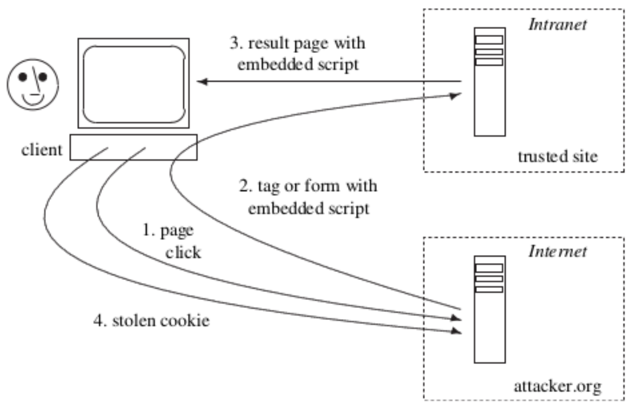


Figure 2.3: Reflected Cross-Site Scripting Vulnerability [14]

2.3.2 Information Leakage

There are many web scanners that can be used to fingerprint websites and perform reconnaissance to obtain information about the services running. At times web servers are left configured with default settings from first installation and as a result they end up disclosing information that should be private. For example, failing to disable directory traversal in your Apache Web Server will allow a user to view all files and sub-directory listings in their browser [36]. Another common case is error message configuration that give away information about the underlying web server version, operating system and database schema.

2.3.3 Content Spoofing

Content Spoofing is similar to XSS but it does not use the <SCRIPT> tags to execute JavaScript. Take the example of Text Injection where we have a URL that is called using the "GET" method and takes in two parameters. One is stockid and the other is recommendation message [31].

http://example.com/stock_info.php?stockid=1234&rec="We recommend you to buy these stocks"

The above PHP script will display the following in the browser when executed:

Stock ID: 1234
We recommend you to buy these stocks

If the URL is malformed to:

http://example.com/stock_info.php?stockid=1234&rec="We recommend you to sell these stocks"

We now get:

Stock ID: 1234
We recommend you to sell these stocks

The victim can be influenced to make a bad decision which will result in selling stocks. The web developer should have sanitized the inputs in their PHP application script to not allow text injection that would mislead a user.

2.3.4 Cross Site Request Forgery - CSRF

Web applications can receive requests that may have been forged by another web page opened in the same browser [32]. This malicious web page can assume the identity of the unaware user and send requests to other websites on their behalf. These attacks are known as CSRF.

Take the example below where we have a GET request to change a user's password within a web application:

GET http://example.com/changePassword?value=newpassword HTTP/1.1

Assume that we have a malicious web page which has some standard HTML in it including the <IMG SRC> markup tag.

<IMG SRC="http://example.com/changePassword?value=hackedpassword">

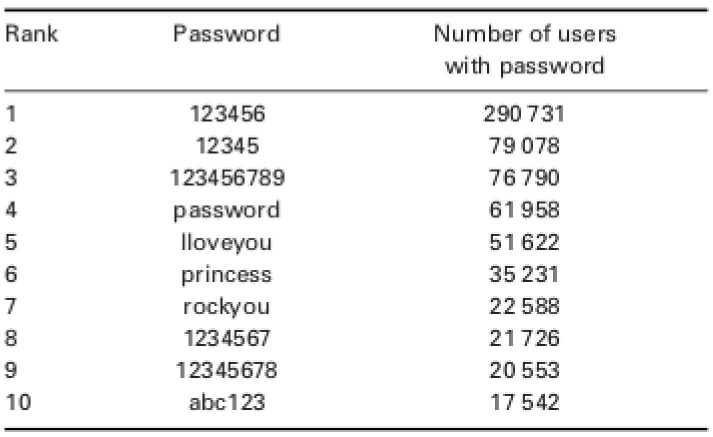
When this IMG tag gets loaded in the browser it will send a request to example.com and execute changePassword. The password will get modified. One way to mitigate is to generate a random token string and pass it with the request. Now it will be hard for the attacker because they have to guess the token value which is randomly generated and not easily predictable. The cross site request forgery will now fail to execute.

GET http://example.com/changePassword?value=newpassword
&token=933a96f6ea1c8abf9cc103a7ff02df77 HTTP/1.1

2.3.5 Brute Force

"Brute-force attacks are often used for attacking authentication and discovering hidden content/pages within a web application. These attacks are usually sent via GET and POST requests to the server. In regards to authentication, brute force attacks are often mounted when an account lockout policy in not in place" [36].

A brute force attack can use a dictionary where the source is a list of words. These words can be permutations of alphanumeric characters. Figure 2.4 shows some popular password strings and their frequency collected from a breach that lead to the release of 32 million passwords to the internet in 2009 [17] [21].



Rank	Password	Number of users with password
1	123456	290 731
2	12345	79 078
3	123456789	76 790
4	password	61 958
5	Iloveyou	51 622
6	princess	35 231
7	rockyou	22 588
8	1234567	21 726
9	12345678	20 553
10	abc123	17 542

Figure 2.4: Passwords Popularity [17] [21]

Based on the data in the table above over 290,000 password strings were just the digits 123456. If a user's password is set to '123456' it becomes relatively easy to crack the account. As web developers we can make it difficult for such attacks to work by implementing *CAPTCHA's*, password strength detectors and also incorporate account lockdown if failed login attempts exceed a set threshold. It is good practice to use a combination of alphanumeric and special characters in your password strings with both lower and uppercase letters. The acronym CAPTCHA stands for *"Completely Automated Public Turing test to tell Computers and Humans Apart"* [15]. This is a challenge-response test used in computing to determine if a user is a human or a machine. Brute force attacks are performed by automated computer programs, thus CAPTCHA's provide a mechanism to thwart such penetration attacks.

2.3.6 Insufficient Transport Layer Protection

There are websites in the Internet that transact login credentials over HTTP and not HTTPS. SSL which stands for Secure Socket Layer provides data encryption, server authentication, message integrity check, and optional client authentication for a transmission control protocol TCP/IP connection [9]. SSL transforms HTTP requests to HTTPS and makes them more secure by encrypting the data before it is sent.

Web applications that use HTTPS transfer private user data over the network in an encrypted form back and forth between client browsers and web servers. Authentication over HTTP (non-secure) will transmit in plain text and someone sniffing in the middle can capture the credentials. This makes the website vulnerable to what is called *Man In The Middle Attack* [23].

2.3.7 Insufficient Authorization

This problem surfaces where web applications do not enforce adequate access control policies on users. After a user authenticates to a website it does not mean they should have access to all resources on the server. The web application developer has to enforce restrictions so that access to sensitive information on the web server are prohibited [14].

2.3.8 SQL Injection - SQLi

"An SQL Injection attack is when a web page allows users to enter text into a text box that will be used to run a query against a back-end database" [29].

This attack gives a hacker the ability to take control of the back-end database and the data that resides within. SQL injections are a prevalent exploit and frequently make it in the OWASP top10 vulnerabilities list every year [36]. An attacker will piggy-back on a legitimate SQL query by injecting code that will help to leak more information from the database tables. Take the SQL query below:

select ProductName from products where ProductID = 40;

If the hacker submits "40 or 1=1" for the ProductID then the query will become:

select ProductName from products where ProductID = 40 or 1=1;

This resultant query will return all the product names and not just the one for the ProductID=40. The best protection against such attacks is to never use dynamic SQL statements in your code, user input should be filtered, use query parameterization and limit database privileges by context.

2.3.9 Session Management

HTTP is a stateless protocol. Web applications can create sessions on top of HTTP by generating session identifiers (SID) and send them to clients via the HTTP response headers. The client will include these session identifiers in subsequent requests so that the user does not have to re-authenticate for every request. Remember HTTP is stateless. It is important these session identifiers are unpredictable and they should also be stored in a safe place [14].

If cookies are used for access control then it must be ensured that the client is not able to elevate security permissions by modifying the session identifiers. These kind of attacks are known as *Cookie Poisoning*. Using a complex attack with XSS an attacker can steal a victim's session cookie to gain access to the victim's resources (e.g. bank account details).

2.3.10 Software Package Updates

It is important to ensure that the underlying software libraries are always up to date and security patches are applied routinely on the hosts that run the software. Sometimes the exploit is not in the web application but in the supporting libraries. One of the most recent cases was the Heartbleed security bug in the OpenSSL library [47]. This problem had to do with a buffer over-read which essentially means that the library allowed more data to be read than should be allowed.

"At the time of disclosure, some 17% (around half a million) of the Internet's secure web servers certified by trusted authorities were believed to be vulnerable to the attack, allowing theft of the servers' private keys and users' session cookies and passwords" [30].

It became imperative for businesses to patch their servers and re-generate and revoke certificates. In general it is best to keep both hardware and software up to date with the latest security updates and advisories in order to prevent any sort of compromise.

2.4 Tools

There are many Web Application Security Testing tools in the market which can be classified into two respective categories:

- Commercial Tools
- Open Source Tools

Commercial tools come with a licensing fee per deployment and the user receives some sort of service level agreement and support. At times the software companies that design these tools will also provide technical support and customization to meet the needs of the client but this is usually provided with an additional cost.

Open Source tools are free to use and fall under the GNU General Public License [12]. There is a development community associated with them which provide updates and enhancements. The source code to these tools is readily available and can be modified by anyone who uses them granted the changes they make are published so others can share and build upon each other's work. In this section we will explore some of these web application security testing tools and their capabilities. We will also see what they bring to the table as far as cost and functionality are concerned keeping in mind that our focus is on the process of web application security regression testing.

2.4.1 Commercial Tools

There are numerous commercially available web application scanners in the market. Listed below are some of them:

Scanner	Vendor
AppScan	IBM
Burp Suite Professional	PortSwigger
Nessus	Tenable Network Security
NetSparker	Mavituna Security
WebInspect	HP

Table 2.1: Commercial Web Application Scanners

2.4.2 Open Source Tools

Listed below are some of the open source web application scanners.

Scanner	Vendor
Arachni	Tasos Laskos
Skipfish	Michal Zalewski
Wapiti	Nicolas Surribas
ZAP	OWASP
Paros	Chinotec

Table 2.2: Open Source Web Application Scanners

2.4.3 Performance And Price

WIVET is an open source benchmarking project that aims to statistically analyze web link extractors and adopted as an extension to the WAVSEP - The Web Application Vulnerability Scanner Evaluation Project [36]. The WIVET score is a KPI metric used to evaluate the performance and coverage of a web application scanner [36]. The data in the tables below are from tests conducted on Oct 31st 2014 by WAVSEP. Product prices are subject to change and are from the year 2012 for single user license.

Scanner	WIVET Score	SQLi	RXSS	LFI	RFI	Redirect	Price
AppScan	92	100	100	100	100	36.67	$4500
Burp	16	100	96.97	56.13	72.22	30.0	$300
NetSparker	92	100	100	96.32	100	36.67	$6000
WebInspect	96	100	100	91.18	100	50.0	$1500
Arachni	19	100	66.67	100	100	50.0	Free
Skipfish	48	76.47	93.94	82.35	31.48	36.67	Free
Wapiti	44	100	66.67	51.47	57.41	N/A	Free
ZAP	73	100	100	75	100	16.67	Free
Paros	19	93.38	98.48	12.75	N/A	N/A	Free

Table 2.3: WIVET Score, Detection Rate (%) For Classes Of Vulnerabilities And Price [7]

Scanner	SQLi	RXSS	LFI	RFI	Redirect
AppScan	0	0	0	0	11.11
Burp	0	0	0	0	0
NetSparker	30	0	0	0	0
WebInspect	0	0	0	0	0
Arachni	20	0	0	0	0
Skipfish	0	0	2.0	16.67	0
Wapiti	20	42.86	12.5	0	N/A
ZAP	30	0	0	16.67	0
Paros	0	0	37.5	N/A	N/A

Table 2.4: False Positive Rate (%) For Classes Of Vulnerabilities [7]

Abbreviations:

- SQLi = Structured Query Language Injection
- RXSS = Reflected Cross Site Scripting
- LFI = Local File Injection
- RFI = Remote File Injection
- Redirect = Redirect used in phishing attacks
- WIVET = Web Input Vector Extractor Teaser

Based on the results from Figure 2.3 above all commercial scanners except Burp have a higher WIVET Score compared to the open source tools. The commercial scanners have a higher detection and lower false positive rate compared to the open source tools. ZAP (Zed Attack Proxy) performs well compared to others in the free of cost category and will be used as one of the components in the Security Regression Testing Framework.

2.5 Auto Regression Testing

"Testing is a destructive task in which the goal is to find relevant defects as early as possible. It requires automation to reduce cost and ensure high regression, thus delivering determined quality." [10]

Automation will minimize labor hours and money. Regression testing is the process of re-testing the modified parts of the software application thus ensuring that no new errors have been introduced into previously tested code [19]. Web developers fix bugs, implement new features and submit code periodically to production systems. Whenever new code is introduced an automated regression test should run to ensure that new bugs are not introduced and test cases that ran prior to the change also pass in this process.

Web Application Security Regression Testing should be an automated process that runs a series of security tests against the web application to determine a pass or fail outcome. The focus is to test the web application for security holes. These security tests should check the web application for XSS, SQLi, RFI, LFI, Redirect and other classes of vulnerabilities in an automated manner. It is not possible to cover all kinds of security tests but the basic vulnerabilities should be addressed in the setup. The web application should be screened and quarantined every time a web developer submits code to the projects source code control system. A source code control system is a repository that stores revisions of software code implemented by a group of developers who work together on the same project.

General software testing can consume 30 to 60 percent of all software life-cycle cost depending upon the complexity of the product [10]. Security testing is not trivial but a specialized skill that requires in-depth knowledge of application protocols like HTTP and networking protocols like TCP/IP. As a tester you need to identify all entry points to the web application and adequately perform security testing on them. Running a series of automated security tests that check the web application for security holes will improve the quality of the product. Two methods which can be used to security test web applications are blackbox and whitebox testing.

2.5.1 Blackbox Testing

Blackbox security testing or black-box penetration testing is where the ethical hacker has no knowledge of the web application he or she is attacking. The objective is to simulate a cyber attack and analyze the outcome through the responses and results [44].

2.5.2 Whitebox Testing

Whitebox security testing or glass box penetration testing is where the ethical hacker has internal perspective of the web application. The objective here is to simulate a malicious insider who has knowledge of the target web application. In this scenario, the tester knows the source code and will inject attack vectors that are specially crafted to test the internals for security flaws [44].

2.5.3 What Is The Goal?

"The fundamental argument is zero or full knowledge - which is best?" [28]. A penetration test regardless of its nature should generate a report that details the test cases, vulnerabilities, exploits discovered and recommendations on how to improve security. Blackbox and Whitebox are different methods that should be both incorporated in the automated security test regression suite. New attack vectors and vulnerabilities are reported on a daily basis and repeating tests using both ways is essential to keep your web applications secure.

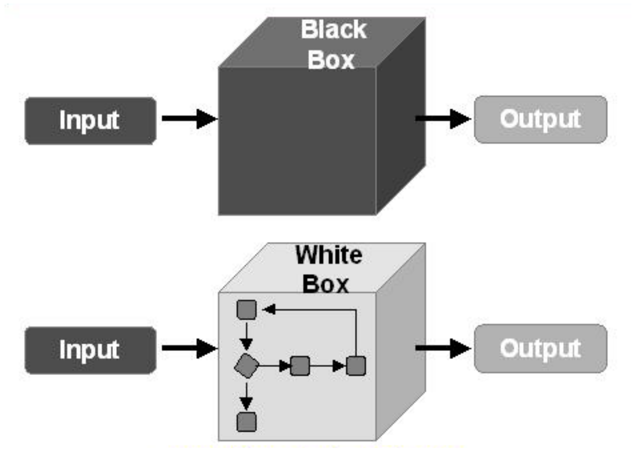


Figure 2.5: Blackbox And Whitebox Penetration Testing

2.6 Literature Review

2.6.1 OWASP

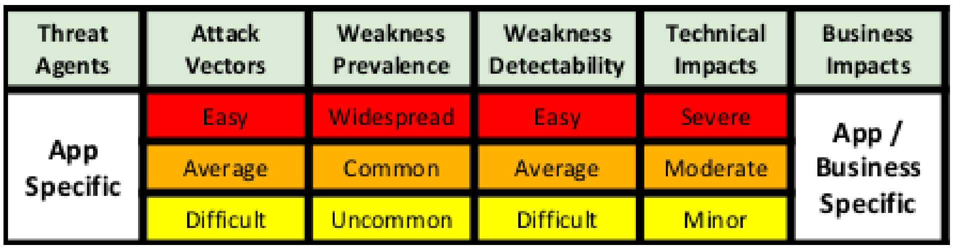
"OWASP is an international organization and the OWASP Foundation supports OWASP efforts around the world. OWASP is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted" [36].

Open Web Application Security Project is one of the best resources for those interested in software security. The foundation established itself as a not-for-profit charitable organization in the United States on April 24th 2004. This global group has over 42,000 participants where all its material is available under a free and open license.

When it comes to web applications security, OWASP plays a significant role in providing both tools and information on how to secure software. All OWASP projects fall under the following four categories:

- **Documentation:** Seek to communicate information or raise awareness about topics in application security.
- **Tools:** Create software that allows users to test, detect, protect or educate themselves using a facet of application security.
- **Code Library:** Libraries and frameworks that can be used by developers to enhance the security of their applications.
- **Operational:** Provide operational support in order to develop media content for the foundation.

Currently there are over 100 projects that fall into one of the four categories noted above. OWASP also publishes an awareness document on the most critical top 10 web application security flaws for the year. This report provides a description for each risk, example attacks, how to avoid and related references to other resources.



Threat Agents	Attack Vectors	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
App Specific	Easy	Widespread	Easy	Severe	App / Business Specific
	Average	Common	Average	Moderate	
	Difficult	Uncommon	Difficult	Minor	

Figure 2.6: OWASP Risk Rating Scheme [36]

2.6.2 ZAP

ZAP is an OWASP project, available to download for free and use within a framework to test websites for security holes. ZAP provides an API a program can use to operate it. Developers can automate security testing of their web application using this API. The tool is written in the JAVA programming language and latter this paper shall demonstrate how the JAVA API can be used to communicate with ZAP in order to run security tests [4].

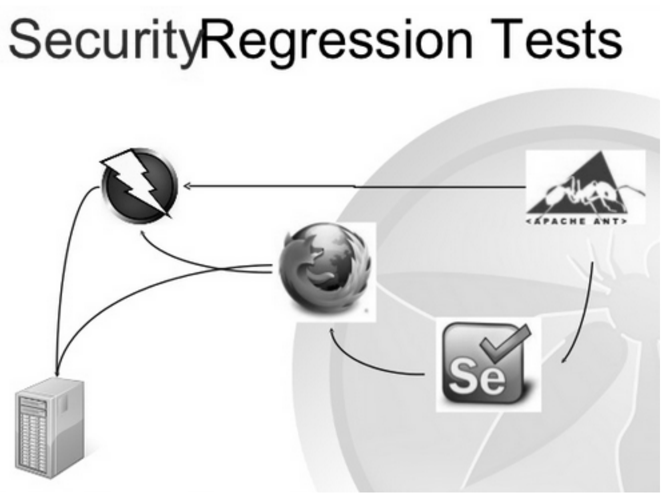


Figure 2.7: ZAP In Action [3]

In Figure 2.7 ANT which is a JAVA build tool starts ZAP and then uses Selenium to load a set of URLs using the Firefox browser. These URLs will be proxied through the ZAP scanner by the Firefox browser. The ZAP scanner will send requests to the target webserver and return responses back to the browser through it. As you can see, one can use ZAP to proxy requests and responses through it using a web browser and selenium software. Later we shall demonstrate how to extend this setup using a GIT repository, Jenkins Continuous Integration Server and MySQL database in order to achieve an automated web application security regression test process.

Noted below are some of the features in ZAP:

- Intercepting Proxy - Analyze requests and responses
- Active and Passive Scanners - Discover and detect vulnerabilities
- Spider - Crawl a target website
- Report Generation - Scan results

- Brute Force - Perform dictionary style attacks
- Fuzzing - Input of random data strings in request headers and form parameters
- Extensibility - Customized scripts to detect security flaws

ZAP has gained much support over the past few years and was reported as the best security tool for 2013 [33]. It is easy to use and can run as a stand-alone JAVA application or in daemon mode on a remote server. ZAP is designed for an audience that encompass experienced penetration testers and novices both.

2.6.3 Nessus

Nessus is a proprietary comprehensive vulnerability scanner developed by the company Tenable Network Security. It is free of charge for personal use in a non-enterprise environment. For enterprise environments there is a licensing fee. Nessus received the most popular network security tool award in 2000, 2003 and 2006 according to the security tools survey conducted by sectools.org [39].

On Linux you can run the Nessus daemon which performs the scanning and the Nessus client controls the scan and presents the results to the operator. Nessus starts with a port scan with one of its four internal portscanners, or it can optionally use amap or nmap port scanners to determine which ports are open on the target host. Once it has determined which ports are open it can then try exploits on these available ports. The vulnerability tests are available as subscriptions and are written in Nessus Attack Scripting Language (NASL). This is a scripting language optimized for custom network interaction. Tenable Network Security produces several dozen new vulnerability check plugins each week, usually on a daily basis. These checks are available for free to the general public but the professional feed is not free of cost and also provides access to support and additional scripts (e.g. audit files, compliance tests, and additional vulnerability detection plugins). Scan reports can be generated in various formats, such as plain text, XML, HTML and LaTeX. The results can also be saved in a knowledge base for debugging. On Linux, the scanning step can be automated through a command-line client [40].

2.6.4 Metasploit

Metasploit is another computer security project that provides information about security vulnerabilities and aids in penetration testing. It is owned and developed by the company Rapid7 LLC. There is a sub-project called the Metasploit Framework which can downloaded free of cost and is open source, and is delivered with a limited set of known exploits. Also there is no direct support available for the free version. With this tool you can develop and execute code against a remote target machine in order to exploit it. The Metasploit Project is famous for its anti-forensic and evasion tools, some of which are built into the Metasploit Framework [24].

This framework uses the following steps to exploit a system:

1. Choose and configure an exploit.
2. Check the intended target system is susceptible to the exploit chosen above.
3. Choose and configure a payload to send to the target host.
4. Choose encoding technique in order to bypass the intrusion-prevention system (IPS) on the target host.
5. Execute the exploit.

This framework provides a modular approach, allowing the combination of any exploit with any payload. This is the major advantage of this framework. It facilitates the tasks of attackers, exploit writers and payload writers.

Metasploit runs on Unix (including Linux and Mac OSX) and also on Windows. Before choosing an exploit and payload, information about the target system is required. It is good to fingerprint the target system for information like operating system version and installed network services. Nmap is one tool that can be used to get this fingerprint information. The commercial versions of this framework are Metasploit Express and Metasploit Pro. The free edition comes with capabilities like command line interface, third-party import, manual exploitation and manual brute force [24]. There are a set of developers from Rapid7 LLC that interface with users from the open source community and provide them with support and information.

2.6.5 Threat Modeling

"Threat modeling, an engineering technique you can use to shape your software design, will help inform and rationalize your key security engineering decisions. In its simplest form, a threat model is an organized representation of relevant threats, attacks and vulnerabilities in your system" [27].

During the design phase it is best to ask various "what if" questions related to the security of your web application. These questions will help evaluate the security concerns in your web applications and also partition infrastructure versus application threats. When you partition by network, servers, desktops and applications it helps to identify key areas and their ownership. There will also be intersections where the application and infrastructure both require security hardening. An infrastructure threat can become the source of inputs to the application resulting into a compromise [27].

Threat modeling is a more sophisticated way of creating test cases to test specific vulnerabilities. During the modeling phase a list of vulnerabilities that your web application is most susceptible to should be created. You can then use data-flow analysis or question-driven approaches to identify and test for specific cases. This is a method that should be used for web application security testing.

Chapter 3

Approach

In order to demonstrate the actual automated process of web application security regression testing both hardware and software components will be setup according to the architecture diagram noted below. The software components will be deployed on different servers. Once the construction of the framework is complete a series of experiments will be conducted to demonstrate the process and discovery of vulnerabilities. The setup will use Linux as the underlying operating system and open source software. This framework will be called SRT (Security Regression Tester). The idea is to show the automated build process where a sequence of web application security regression tests are executed when a developer performs a commit. The focus is to analyze the advantages from the automated process versus conventional manual testing. The results from the experiments will be saved to flat text files and made available on the continuous integration server for perusal. The Bodgeit Store is an Open Source Vulnerable Web Application that can be downloaded for free and installed within Tomcat7 which is the version used in this setup.

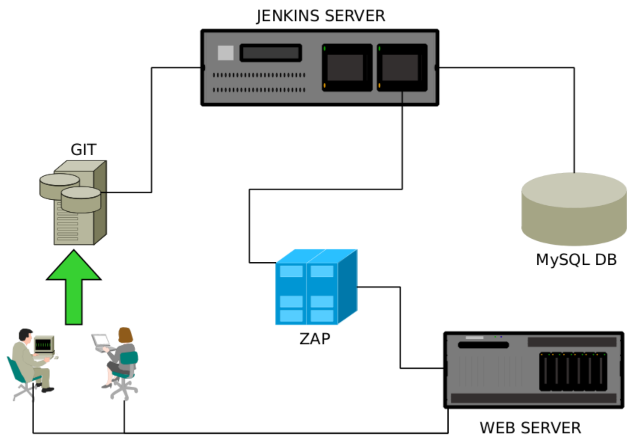


Figure 3.1: Architecture

3.1 Setup And Installation

3.1.1 Hardware

Role	Host	OS	CPU	RAM	DISK
Jenkins	ahs-m1.ams.osa	Debian6	4 vCPUs 1.7GHz	16GB	100GB
MySQL	ahs-db1.ams.osa	Debian6	4 vCPUs 1.7GHz	16GB	100GB
GIT	ahs-s1.ams.osa	Debian6	4 vCPUs 1.7GHz	16GB	100GB
Webserver	t09-09.oslo.osa	Debian6	16 Cores 2.27GHz	24GB	450GB
ZAP	owasp-t01.oslo.osa	Debian7	2 vCPUs 2.2GHz	8GB	100GB
Sandbox	rumi.oslo.osa	Debian7	Core 2 Duo 2.3GHz	4GB	200GB

Table 3.1: Hardware Setup

Openstack will be used to host the Jenkins Server, MySQL DB and the GIT server. The target web application will run on a physical rack server and ZAP will be configured on a KVM instance. Openstacks graphical web interface will be used to build three VM's using an image. This standard VM image that will be used shall provide 100GB of disk in the root partition. Security TCP rules will be deployed on the VM instances to open up ports in order to allow connectivity to and from the rest of the servers in the architecture. Both ams.osa and oslo.osa are domains pre-configured in private internal subnets. The Sandbox is a developer workstation where implementation and testing of the Bodgeit Store project is conducted in a local environment before pushing changes to the staging server which in this setup is the Webserver for the Bodgeit Store web application.

3.1.2 Software

3.1.3 Webserver

Tomcat7 will be installed on **t09-09.oslo.osa**. This node is the staging server for the Bodgeit Store web application. A GIT code commit from the sandbox machine will rsync code changes via a GIT hook to this staging server. The Bodgeit Store vulnerable web application will be downloaded and installed on t09-09.oslo.osa. The web application which be accessible through the link *http://t09-09.oslo.osa:8080/bodgeit*. Post installation of the web application the directory */var/lib/tomcat7/webapps* and its contents will be constructed in a GIT repository on the GIT server. This can be done by creating a tarball which can then be extracted on the GIT server latter. Tomcat7 will also need to be configured so it can perform detailed logging such as request headers and GET/POST payloads (request body). This is explained in the experiments section.

3.1.4 GIT

GIT [6] software will be used for source code management. The GIT server will reside on the node **ahs-s1.ams.osa**. A repository called *bodigeit.git* will be created. The version of GIT that will be used is 1.7 and a user *git* will also be created on the GIT server. The tarball created earlier will be untarred and added into the GIT repository. This ensures the structure of the Bodgeit Store web application on the staging Webserver and GIT repository are in sync with the same files and directories to start with.

3.1.5 Jenkins Continuous Integration Server

Jenkins Continuous Integration Server [20] will be installed and configured on **ahs-m1.ams.osa** to run the builds and communicate with the rest of the nodes in the setup. Workspace for the project *Bodgeit Store* [35] will be created on the Jenkins server which shall run on port 8080. Binaries for the Java program, launcher shell script and Java libraries will be installed and deployed on the Jenkins Server under the directory */home/jenkins*. Jenkins version 1.588 will be installed. A public and private key pair will be generated for the user *jenkins* using *ssh-keygen* on the Jenkins server under the directory */home/jenkins/.ssh*. The public key for the user *jenkins* will be added to the *authorized_keys* file on the GIT server under the directory */home/git/.ssh*. This public key authorization is needed for Jenkins to poll the GIT server.

3.1.6 MySQL Database

MySQL database will be installed and configured on the node **ahs-db1.ams.osa**. All web URLs to be proxied through ZAP for the Bodgeit Store project will be stored in this MySQL database. A database called *SRT* will be setup with two tables. These table are *projects* and *urls*. They will be populated with the necessary meta data and corresponding urls to webpages that need to be tested. A database user *db_read* will also be created which will have read only permissions. DB connections from the Jenkins Server will be allowed using the credentials of this user *db_read*. The bind-address for the MySQL database in the config file */etc/mysql/my.conf* will need to be changed to the ipaddress of ahs-db1.ams.osa. MySQL Server version 5.1 will be used.

3.1.7 Java Application

A software application using the Java programming language will be written, compiled and tested on the Jenkins server **(ahs-m1.ams.osa)**. This application will control the run of the regression build and security testing apparatus. It will use various available Java packages, Selenium Web Driver, HtmlUnit (headless browser), ZAP API calls and MySQL connectors to perform its functions. This component will act as the central intelligence piece in the setup. The behavior of the Java application and settings in ZAP will be controlled by passing command line arguments to this program. *Open JDK (Java Development Kit) version 1.7.0_03* will be used for development and execution.

3.1.8 ZAP

ZAP version 2.2.2 will be downloaded and installed as the attack proxy on **owasp-t01.oslo.osa**. It will be configured to run as a daemon in the background. A root certificate for ZAP will need to be generated. This can be done by launching ZAP in standalone graphical user interface mode or after startup in daemon mode. ZAP will listen on port 8080.

3.1.9 Apache2

Apache2 will be installed on the Jenkins server **ahs-m1.ams.osa**. It will run on port 80. A directory called *srt* will be created under */var/www*. Below the directory */var/www/srt* a project specific directory called *bodgeit* will also be created. The path will look like */var/www/srt/bodgeit* and the user *jenkins* will be given permission to write to this directory. This directory is where all the security testing results will deposit.

3.1.10 Developer Sandbox

A sandbox will be created on **rumi.oslo.osa** which is a developer workstation used for development purposes. This machine will need access to both the GIT server and Webserver. The GIT repository *bodgeit.git* will be cloned on this node. This is required so that anytime a developer commits a change to the Bodgeit Project via their local GIT sandbox an rsync initiates to push the changes to the Webserver as well. The rsync will run via a GIT hook which will be setup on the developers local GIT repository on their development machine. As stated earlier, the developer's public ssh-key will need to be added on both the GIT server (/home/git/.ssh/authorized_keys) and the Webserver (/root/.ssh/authorized_keys).

3.1.11 Configurations

The Bodgeit Store is a set of Java Server Pages running in Tomcat7 on t09-09.oslo.osa. Tomcat7 will be configured to dump request headers and payload to the file: 'request_dumper.DATE.log'. It will also be configured to log HTTP GET and POST request information to the 'access.DATE.log' file. Once these configuration changes have been applied, Tomcat7 will need to be restarted.

ZAP's Active Scan runs with different ATTACK STRENGTH settings. This will be controlled by the launcher shell script */home/jenkins/bodgeit/bin/run_bodgeit.sh* on the Jenkins Server. ZAP can be run in 4 different levels which are noted below:

ATTACK STRENGTH	DESCRIPTION
LOW	Limit to around 6 requests per scan call
MEDIUM	Limit to around 12 requests per scan call
HIGH	Limit to around 24 requests per scan call
INSANE	No real limit, can go into thousands.

Table 3.2: ZAP Attack Strength

ZAP's ALERT THRESHOLD [5] determines how strictly to check for vulnerabilities. This threshold value will be set to MEDIUM for all test cases.

ALERT THRESHOLD	DESCRIPTION
LOW	More false positives
MEDIUM	Default
HIGH	More false negatives

Table 3.3: ZAP Alert Threshold

ZAP's RISK [5] reporting indicates the gravity of the alert.

RISK ALERT	DESCRIPTION
INFO	Informational
LOW	Low level vulnerability
MEDIUM	Medium level vulnerability
HIGH	High level vulnerability

Table 3.4: ZAP Risk Alert

The RELIABILITY [5] is an indication of how likely the alert that is reported is a real problem. There are two values that can be reported and they are listed below:

- SUSPICIOUS - A lower level of confidence
- WARNING - A higher level of confidence

3.1.12 Load URLs Into MySQL DB

The following URL's will be inserted into the projects table in the SRT MySQL database on ahs-db1.ams.osa.

1. http://t09-09.oslo.osa:8080/bodgeit/ (GET home page)

2. http://t09-09.oslo.osa:8080/bodgeit/login.jsp (POST login form with username=cosmicrhyhm@hotmail.com, password=123456 and submit)
3. http://t09-09.oslo.osa:8080/bodgeit/contact.jsp (POST contact form with textarea and submit)
4. http://t09-09.oslo.osa:8080/bodgeit/about.jsp (GET about page)
5. http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=28 (POST prodid form with quantity=empty and submit)
6. http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=3 (GET typeid)
7. http://t09-09.oslo.osa:8080/bodgeit/search.jsp (POST search form with q=search_string and submit)
8. http://t09-09.oslo.osa:8080/bodgeit/register.jsp (POST register form with username, password1, password2 and submit)
9. http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=14 (GET prodid=14)
10. http://t09-09.oslo.osa:8080/bodgeit/logout.jsp (GET logout)

3.2 Data Analysis

A session is defined as one complete security regression build starting from a developer GIT commit till the end when we collect the scan results. The request dumper, access and ZAP logs will be saved for each test and will be processed by Perl scripts to generate data and statistics for analysis. This step will be repeated for each of the test cases.

The logfiles request_dumper.DATE.log , access.DATE.log, zap.log and bodgeit-srt.DATE_TIME.txt are parsed to generate results and statistics. All Active Scan requests initiated by ZAP are logged to the request_dumper and access logs. The zap.log stores information on active scan session and the bodgeit-srt.DATETIME.txt file provides the scan results and ZAP recommended resolutions. All logs will be parsed using scripts written in Perl. There will be 3 scripts called **zap.pl**, **access.pl** and **request.pl** for each respective log type. The contents from the results log will not need parsing as the log itself will have the summary of alerts at the very bottom. By parsing the contents in these log files metrics will be generated which are described individually for each log type below.

3.2.1 ZAP Log

The ZAP log will provide the execution time taken by each ZAP plugin and the overall execution time for the entire scan. The results from this log are compiled and presented in a table. The metrics extracted from this log are:

- Execution time for each ZAP plugin
- Total Execution Time by ZAP Active Scanner

3.2.2 Access Log

The access logs record all the GET and POST requests received by the Tomcat7 server. The log entries are parsed and the following metrics will be compiled:

- Total scan requests to proxied URLs
- # of scan requests broken down by method (GET/POST) under each proxied URL
- # of Requests by each ZAP plugin using plugin execution time (This is a function metric calculated using total scan requests)

3.2.3 Request Dumper Log

The request dumper logs store headers and POST payload parameter information. The log entries are parsed and the following metrics will be compiled:

- # of times a unique header was fuzzed by ZAP
- # of times a unique parameter was fuzzed by ZAP

3.2.4 Results Log

The results log follow the naming convention bodgeit-srt.DATE_TIME.txt and are generated by the Java Application running on the Jenkins Server. The information in this log is retrieved from ZAP via an API call by the Java application. This log will provide a summary of results from the Active Scan and also resolution recommendations by ZAP which is one of its built in feature. These logs will deposit in the Apache directory */var/www/srt/bodgeit/*. No parsing of these logs is required.

3.3 Experiments

Here we outline experiments and some configuration changes needed before conducting the tests. The web application to test here is the Bodgeit Store which runs on the staging Webserver t09-09.oslo.osa.

3.3.1 Test Case: 1(a)

Execute manual test to run build and check for any setup errors. Click on "Build Now" under the Bodgeit Store project on the Jenkins Server to initiate test.
Check Jenkins Server project Bodgeit Store for build results.

TEST SETTING	VALUE
ATTACK STRENGTH	LOW
ALERT THRESHOLD	MEDIUM

Table 3.5: Settings Test Case 1(a)

3.3.2 Test Case: 1(b)

Execute automated test by performing GIT commit to the repository *bodgeit.git* from developer sandbox.

TEST SETTING	VALUE
ATTACK STRENGTH	LOW
ALERT THRESHOLD	MEDIUM

Table 3.6: Settings Test Case 1(b)

Check Jenkins Server project Bodgeit Store for build results.

3.3.3 Test Case: 2(a)

Execute automated test by performing GIT commit to the repository *bodgeit.git* from developer sandbox.

TEST SETTING	VALUE
ATTACK STRENGTH	MEDIUM
ALERT THRESHOLD	MEDIUM

Table 3.7: Settings Test Case 2(a)

Check Jenkins Server project Bodgeit Store for build results. Collect data from request_dumper.DATE.log, access.DATE.log and zap.log for analysis.

3.3.4 Test Case: 2(b)

Execute automated test by performing GIT commit to the repository *bodgeit.git* from developer sandbox.

TEST SETTING	VALUE
ATTACK STRENGTH	HIGH
ALERT THRESHOLD	MEDIUM

Table 3.8: Settings Test Case 2(b)

Check Jenkins Server project Bodgeit Store for build results. Collect data from request_dumper.DATE.log, access.DATE.log and zap.log for analysis.

3.3.5 Test Case: 2(c)

Execute automated test by performing GIT commit to the repository *bodgeit.git* from developer sandbox.

TEST SETTING	VALUE
ATTACK STRENGTH	INSANE
ALERT THRESHOLD	MEDIUM

Table 3.9: Settings Test Case 2(c)

Check Jenkins Server project Bodgeit Store for build results. Collect data from request_dumper.DATE.log, access.DATE.log and zap.log for analysis.

3.4 Return On Investment (ROI)

The Return on Investment for this specific paper is a metric that will be calculated to show expected gains or losses for using test automation versus traditional manual testing [38]. In order to calculate this ratio which is expressed below variables, equations have to be defined and some assumptions have to be set.

$$ROI = \frac{Benefits}{Investment} \text{ [38]} \tag{3.1}$$

All the tools used in this automation setup are Open Source so the cost of software = 0. The learning cost is 80 hours (2 weeks) multiplied by $50/hour which gives us = $4000. This learning cost is the same for both manual and automated testing. The hardware cost for manual testing is just 1 server = $1000 whereas for automated testing it is 5 servers = $5000.

(eq1) Manual Investment Cost = $Software_{cost} + Learning_{cost} + Hardware_{manual}$
(eq2) Automation Investment Cost = $Software_{cost} + Learning_{cost} + Hardware_{auto}$

There is also an initial implementation cost for both manual and automated. The time spent to implement the automated = 80 hours and for the manual = 8 hours the first time. The total number of test cases = 10 (URLs loaded in the MySQL DB) and the tester is paid $50/hour.

(eq3) Manual Implementation Cost = TestCases * $Tester_{hourlyrate}$ * $TimeSpent_{manual}$
(eq4) Automated Implementation Cost = TestCases * $Tester_{hourlyrate}$ * $TimeSpent_{auto}$

Next is to calculate the manual and automated cost of testing 10 URLs. It takes 4 hours for a tester to manually test 10 URLs and it takes 1 hour via automation. The tester is paid $50/hour.

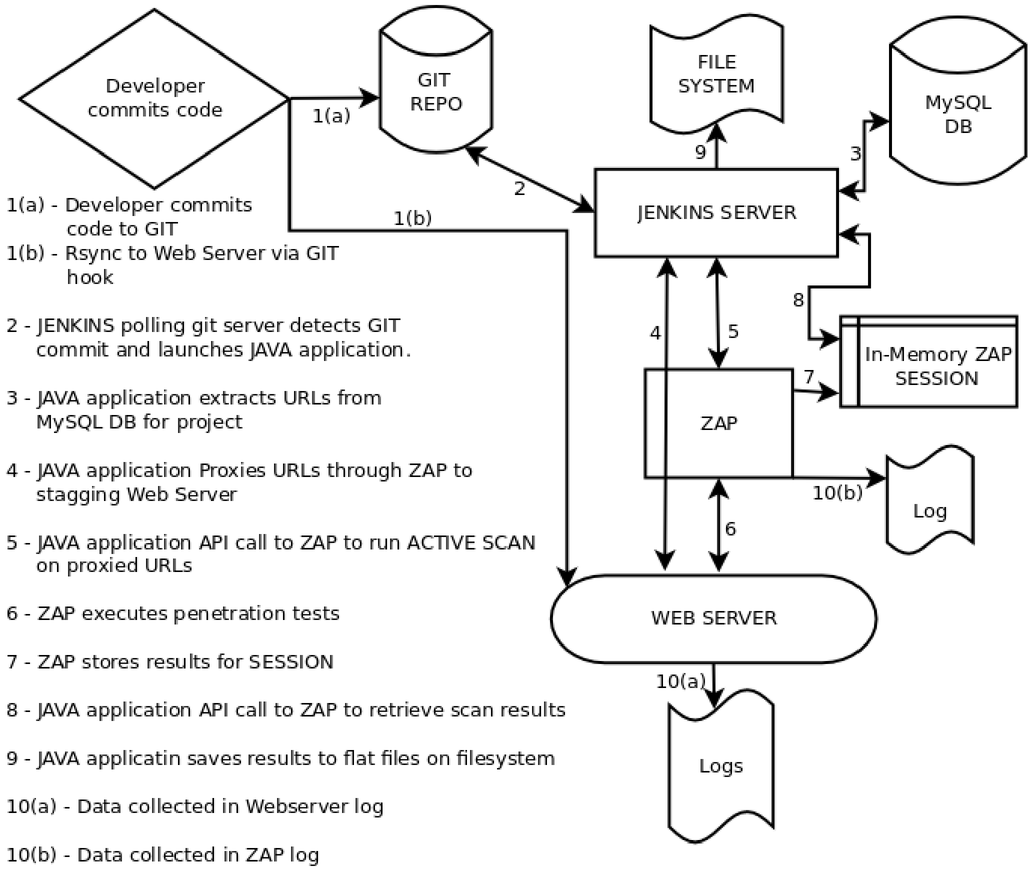
(**eq5**) Manual Testing 10 urls Cost = TestCases * $Tester_{hourlyrate}$ * $Time_{manual}$
(**eq6**) Automated Testing 10 urls Cost = TestCases * $Tester_{hourlyrate}$ * $Time_{auto}$

Total Manual Cost = *eq1* + *eq3* + (*Builds* - 1) * eq5
Total Automated Cost = *eq2* + *eq4* + (*Builds* - 1) * eq6

Builds is defined as the number of times code commits are performed by the developer. Our assumption is that with every code commit a tester has to perform a manual security test of the 10 URLs (test cases). In case of the automation the 10 URLs are security tested through the automated setup.

Calculations will be performed for total manual and automated cost using the assumptions, variables and their values noted above. We shall start with Builds = 1 and increment by 10 till we get to 100. The manual and automated costs for the Builds variable will be calculated and provided as a table in the Results section. The analysis of these results will be performed in Chapter 5.

3.5 Process Flow Chart



1(a) - Developer commits code to GIT
1(b) - Rsync to Web Server via GIT hook

2 - JENKINS polling git server detects GIT commit and launches JAVA application.

3 - JAVA application extracts URLs from MySQL DB for project

4 - JAVA application Proxies URLs through ZAP to stagging Web Server

5 - JAVA application API call to ZAP to run ACTIVE SCAN on proxied URLs

6 - ZAP executes penetration tests

7 - ZAP stores results for SESSION

8 - JAVA application API call to ZAP to retrieve scan results

9 - JAVA applicatin saves results to flat files on filesystem

10(a) - Data collected in Webserver log

10(b) - Data collected in ZAP log

Figure 3.2: Process Flow Chart

Chapter 4

Results

This chapter shows the installation commands executed on servers used in the setup. The edits to configuration files, Java libraries that are used and log results are all highlighted. The data collected from the experiments is also tabulated into tables and presented in this section so it can be analyzed later on.

4.1 Setup And Installation

4.1.1 Install Webserver (t09-09.oslo.osa)

Install Tomcat7

```
apt-get install tomcat7
```

Download Bodgeit Store Vulnerable Web Application and install in Tomcat7 /var/lib/tomcat7/webapps directory.

```
wget http://bodgeit.googlecode.com/files/bodgeit.1.4.0.zip

mv bodgeit.1.4.0.zip /var/lib/tomcat7/webapps/
cd /var/lib/tomcat7/webapps/
unzip bodgeit.1.4.0.zip
```

Restart Tomcat7, open browser and surf to Bodgeit Store main page

```
/etc/init.d/tomcat7 restart

http://t09-09.oslo.osa:8080/bodgeit/
```

Tarball /var/lib/tomcat7/webapps to deploy on GIT Server repository latter on.

```
tar -cvf /var/tmp/bodgeit.tar /var/lib/tomcat7/webapps/bodgeit
```

4.1.2 Install GIT (ahs-s1.ams.osa)

Create Linux user *git* on GIT server.

```
adduser git
```

Install GIT, create bodgeit.git directory under /home/git as user *git*.

```
apt-get install git
su git
mkdir /home/git/bodgeit.git
```

Copy bodgeit.tar file from Webserver to GIT server and untar in /home/git/bodgeit.git
Initialize GIT repository and add all files and directories to new GIT repo bodgeit.git.

```
cd /home/git/bodgeit.git
tar -xvf /path/to/bodgeit.tar .
git init
git add --all
git commit -am "Adding Bodgeit Store to GIT repo"
git push origin master
```

4.1.3 Install Jenkins (ahs-m1.ams.osa)

Install Jenkins and also install GIT (git commands are needed by the Jenkins Continous Integration Server).

```
apt-get update
apt-get install jenkins
apt-get install git
```

Check Jenkins is installed by opening webpage in browser

```
http://ahs-m1.ams.osa:8080
```

Generate public and private keys for user *jenkins* using ssh-keygen. The public key will be added to the authorized_keys file on GIT server latter.

```
su jenkins
cd /home/jenkins

# Save keys to /home/jenkins/.ssh
ssh-keygen

jenkins@ahs-m1:~/.ssh$ ls
id_rsa        id_rsa.pub
```

Create directories under /home/jenkins for Java application binaries, libraries and launcher script. Please see appendix for list of libraries, Java source files, binaries and other scripts.

```
cd /home/jenkins
mkdir lib
cp list_of_java_libraries /home/jenkins/lib

mkdir /home/jenkins/bodgeit
mkdir /home/jenkins/bodgeit/bin
cp java_binaries_and_launcher_script /home/jenkins/bodgeit/bin

jenkins@ahs-m1:~/bodgeit/bin$ ls
run_bodgeit.sh        Run_SRT.class   ZapperReport.class
```

4.1.4 SSH Access From Jenkins To GIT

Add user *jenkins* public key generated on Jenkins Server to /home/git/.ssh/authorized_keys on GIT server. Then test connection from Jenkins Server.

```
# Public ssh key of user jenkins on ahs-m1.ams.osa
# added to /home/git/.ssh/authorized_keys on
# ahs-s1.ams.osa

echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDSzx
LPd6EcHNCTIJQ3No7nox4iNZNRbDzttt1QqCHsUS8eno1BhK
UhE1/QEpea/5t7Wng8JY7SiyqDT/D68FUhpZA13ahWf5rcgo
XgPy90rOoK5rGQUEgoAwolnIIorxj05BetqPLoTULbwQXa8k
qsxpmkW1ul1Z4HKTHn/2tlP74Jlf3WFiZO279gKEr0F4vSKY
qEP4ycXDIyg1nVJUd/6uE+mu8mOnzRC3mOag3gubHIWJAFeI
NXBhmCkj4HrB2oDK0Lgo7T6YTQR2AwRVcr2mYGtmct50cseP
3plvMqJUL6daQwGjjLfR0lCuEgPgD5Fu1NoCoF4+DJseA3Xb
B3 jenkins@ahs-m1.ams.osa" > /home/git/.ssh/authorized_keys
```

4.1.5 Test SSH Access From Jenkins To GIT

Test ssh connection for user jenkins to GIT server

```
jenkins@ahs-m1:~$ ssh git@ahs-s1.ams.osa
Linux ahs-s1.ams.osa 3.2.0-0.bpo.4-amd64 #1 SMP
Debian 3.2.46-1+deb7u1~bpo60+1 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY,
to the extent permitted by applicable law.
Last login: Sun Nov  9 19:28:10 2014 from ahs-m1.ams.osa

git@ahs-s1:~$
```

4.1.6 Create Bodgeit Store Project In Jenkins

Create and configure Bodgeit Store Project on Jenkins Server. Make sure GIT plugin v2.7.7 is installed on Jenkins Continous Integration Server.

```
Step 1:  Load http://ahs-m1.ams.osa:8080 in web browser
Step 2:  Give project Item name Bodgeit Store
Step 3:  Select radio button Freestyle Project and click OK
Step 4:  Click Project Bodgeit Store -> Configure
Step 5:  Fill in Project name and Description on top
Step 6:  Under Source Code Management select Git
Step 7:  Paste git@ahs-s1.ams.osa:/home/git/bodgeit.git into textarea for
         field Repository URL
Step 7:  Under Build Triggers select Poll SCM
Step 8:  Paste */1 * * * * into text area for field Schedule
Step 9:  Click Add Build Step and select Execute Shell
Step 10: Paste /home/jenkins/bodgeit/bin/run_bodgeit.sh
         into textarea for Command
Step 11: Click Apply Settings
Step 12: Click Save
```

4.1.7 Install MySQL (ahs-db1.ams.osa)

Start by installing MySQL server and client and create database SRT

```
apt-get install mysql-server mysql-client
```

```
mysql -u root -h localhost -p
mysql> CREATE DATABASE SRT;
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| SRT                |
| mysql              |
| performance_schema |
+--------------------+

# Change bind-address in /etc/mysql/conf to the IP of ahs-db1.ams.osa
# Save configuration change.
bind-address          = 10.210.65.13

# Restart MySQL Server
/etc/init.d/mysql restart

# Create tables "projects" and "urls" in SRT database

CREATE TABLE `projects` (
  `project_id` smallint(6) NOT NULL,
  `name` varchar(40) DEFAULT NULL,
  `owner` varchar(30) DEFAULT NULL,
  `contact` varchar(40) DEFAULT NULL,
  `status` tinyint(1) NOT NULL DEFAULT '1',
  `email_active` tinyint(1) NOT NULL DEFAULT '1',
  `report_active` tinyint(1) NOT NULL DEFAULT '1',
  PRIMARY KEY (`project_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

CREATE TABLE `urls` (
  `datetime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `url_id` int(11) NOT NULL AUTO_INCREMENT,
  `project_id` smallint(6) NOT NULL,
  `active` tinyint(1) NOT NULL DEFAULT '1',
  `db_store` tinyint(1) NOT NULL DEFAULT '1',
  `url` varchar(255) DEFAULT NULL,
  `sort_order` smallint(6) NOT NULL,
  `method` varchar(4) NOT NULL DEFAULT 'GET',
  `params` varchar(255) DEFAULT NULL,
  `submit` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`url_id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;

# Create DB user called "db_read" and provide it read privileges
# only to SRT database from jenkins server: ahs-m1.ams.osa
CREATE USER 'db_read'@'ahs-m1.ams.osa' IDENTIFIED BY '5r1r3aD';
```

```
GRANT SELECT ON SRT.* TO 'db_read'@'ahs-m1.ams.osa';
FLUSH PRIVILEGES;
```

Results for DB, tables and db_read user.

```
use SRT;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
show tables;
Tables in SRT
projects
urls

describe projects;
project_id    smallint(6)
name          varchar(40)
owner         varchar(30)
contact       varchar(40)
status        tinyint(1)
email_active  tinyint(1)
report_active tinyint(1)

describe urls;
datetime    timestamp
url_id      int(11)
project_id  smallint(6)
active      tinyint(1)
db_store    tinyint(1)
url         varchar(255)
sort_order  smallint(6)
method      varchar(4)
params      varchar(255)
submit      varchar(50)

use mysql;
select Host,User,Password from user;
ahs-m1.ams.osa db_read  *C7C82041FDB0B48EE38332...
```

4.1.8 Java Application On Jenkins

The Java application resides in /home/jenkins/bodgeit/bin on the Jenkins server ahs-m1.ams.osa. There are two java class files, Run_SRT.class and ZapperReport.class. Listed below are some of the libraries that are used by the Java application. Please see appendix for the complete list.

```
jenkins@ahs-m1:~/lib$ ls -l /home/jenkins/lib
...
...
commons-lang3-3.1.jar
selenium-server-standalone-2.35.0.jar
hsqldb.jar
commons-cli-1.2-sources.jar
htmlunit-2.12.jar
htmlunit-core-js-2.12.jar
commons-io-2.4.jar
httpcore-4.2.2.jar
log4j-1.2.17.jar
zap-api-v2-6.jar
commons-jxpath-1.3.jar
httpmime-4.1.3.jar
mysql-connector-java-5.1.18-bin.jar
zaphelp.jar
commons-lang-2.6.jar
httpmime-4.2.3.jar
nekohtml-1.9.15.jar
zap.jar
...
...
```

4.1.9 Install ZAP (owasp-t01.oslo.osa)

Create Linux user *zap*

```
adduser zap
```

Download ZAP 2.2.2, uncompress file and extract tar ball to directory /home/zap.

```
http://sourceforge.net/projects/zaproxy/files/2.2.2/ \
    ZAP_2.2.2_Linux.tar.gz/download
```

As a pre-requisite you need Java 1.7.x.

```
# Java version
java version "1.7.0_65"
OpenJDK Runtime Environment (IcedTea 2.5.1) (7u65-2.5.1-5~deb7u1)
OpenJDK Server VM (build 24.65-b04, mixed mode)

ls /home/zap
ZAP_2.2.2
```

Launch ZAP in GUI mode first in order to generate a root certificate which will be needed for SSL sites.

Edit /home/zap/.ZAP/config.xml and change the two fields listed below:

```
<ip>owasp-t01.oslo.osa</ip>
<port>8080</port>
```

Launch ZAP in daemon mode using the switch noted below and specify port as well. The zap log resides in the /home/zap/.ZAP directory.

```
# Start ZAP as a daemon
/home/zap/ZAP_2.2.2/zap.sh -daemon -port 8080 &

# Run ps -afe and netstat to check it is running and listening on port 8080
ps -afe  | grep zap
root      18843      1  0 Sep30 ?         04:00:10 java -Xmx512m \
 -XX:PermSize=256M -jar ./zap.jar -daemon -host 10.20.41.13 -port 8080



netstat -nple | grep 8080
tcp6 0 0 10.20.41.13:8080 :::* LISTEN 0 1766873 18843/java
```

Check Log to ensure no errors were generated

```
tail -f /home/zap/.ZAP/zap.log
Found Java version 1.7.0_65
Available memory:  8044 MB
Setting jvm heap size: -Xmx512m
0 [main] INFO org.zaproxy.zap.ZAP  - OWASP ZAP 2.2.2 started.
331 [main] INFO hsqldb.db.HSQLDB379AF3DEBD.ENGINE  - dataFileCache open start
429 [main] INFO hsqldb.db.HSQLDB379AF3DEBD.ENGINE  - dataFileCache open end
812 [main] INFO org.parosproxy.paros.network.SSLConnector- Reading supported SSL/TLS protocols...
812 [main] INFO org.parosproxy.paros.network.SSLConnector- Using a SSLEngine...
1025 [main] INFO org.parosproxy.paros.network.SSLConnector- Done reading supported SSL/TLS protocols:
   [SSLv2Hello, SSLv3, TLSv1, TLSv1.1, TLSv1.2]
...
...
...
2014-10-25 23:00:40,845 INFO  PluginFactory - loaded plugin Path Traversal
2014-10-25 23:00:40,846 INFO  PluginFactory - loaded plugin Remote File Inclusion
2014-10-25 23:00:40,846 INFO  PluginFactory - loaded plugin Server side include
2014-10-25 23:00:40,846 INFO  PluginFactory - loaded plugin Cross Site Scripting (Reflected)
2014-10-25 23:00:40,846 INFO  PluginFactory - loaded plugin Cross Site Scripting (Persistent)
2014-10-25 23:00:40,846 INFO  PluginFactory - loaded plugin SQL Injection
2014-10-25 23:00:40,847 INFO  PluginFactory - loaded plugin Server Side Code Injection Plugin
2014-10-25 23:00:40,847 INFO  PluginFactory - loaded plugin Remote OS Command Injection Plugin
2014-10-25 23:00:40,847 INFO  PluginFactory - loaded plugin Directory browsing
2014-10-25 23:00:40,847 INFO  PluginFactory - loaded plugin Secure page browser cache
2014-10-25 23:00:40,847 INFO  PluginFactory - loaded plugin External redirect
2014-10-25 23:00:40,847 INFO  PluginFactory - loaded plugin CRLF injection
2014-10-25 23:00:40,848 INFO  PluginFactory - loaded plugin Parameter tampering
2014-10-25 23:00:40,848 INFO  PluginFactory - loaded plugin Cross Site Scripting (Persistent) - Prime
2014-10-25 23:00:40,848 INFO  PluginFactory - loaded plugin Cross Site Scripting (Persistent) - Spider
2014-10-25 23:00:40,848 INFO  PluginFactory - loaded plugin Script active scan rules
```

4.1.10 Install Apache2 (ahs-m1.ams.osa)

Install Apache2 and create sub-directories noted below. Provide user *jenkins* permission to write to /var/www/srt/bodgeit

```
# Installation of Apache2 on ahs-m1.ams.osa
apt-get install apache2

# Create /var/www/srt/bodgeit
mkdir /var/www/srt
mkdir /var/www/bodgeit
chown -R jenkins:jenkins /var/www/srt/bodgeit
```

All results from the ZAP scan are written to the directory: /var/www/srt/bodgeit

4.1.11 Developer Access To GIT

Add developer's public key on rumi.oslo.osa to /home/git/.ssh/authorized_keys on GIT server.

```
echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDTxQMP5W
ULLoWQWkmeK73y9Rnhqmko5vnTB1l/thuypvNQ/zuOOrXDQFd9/n
vFGYPLhQFqPyZ+cvJugNOHEHv6kCVjH1y9SLHlvfEDJ5hD+0TFcr
MbDAIu4Y/EU3wtdz8mIWn2xf5SMxdMycpWtER+drpQvibUAJvFlV
rFXS/WuF3+FbmCMMv4r29MJFTioT00nVuuZ+ArX/9h8qqNxl7Kc3
mI6vSLYDs7BOebw87U+UYRMJuvon9+4Zm5QQw5yLW035OZLFpjHO
C+3NxzLLA7eNvn4ykD4l7XJkQHxNDURLYGA4wiBFCa34S2V/qSNQ
exUUxM+zc2c1rJE38nuKCURcIdL9LjiaGqp6m1inT+kqIkUnE2Fg
xCXjZoMWRpqSzYSi4aCrYBGFIE7urBypZs4nS2wKQV7/NRpqv6CO
FND9viINSXEp6RfSaEqKrZ+teWiPpgx8MSNR5kqJI1VkE9wWWFJ/
l3EnjZxplIFH3OY6w3lYVMaaKKEhRdoghazzqLL847h3aABiqAAb
P/IW8OSGb7s9f2MOCedOd6tOGfVlPHQLxOlg05k+r9oFRnepeDOU
UgBwDIpfGnb+fUiZYBRknwf1HfWnp/qzJa+utgq8fTY1pHFqDc1W
1dvSAX9kIy7FaDB7qLpKgX5JrQbJwdESxsUzY9G1FTjEQowLPzuw
== usmanw@rumi.oslo.osa" \
> /home/git/.ssh/authorized_keys
```

4.1.12 Developer Access To Webserver

Add developer's public key on rumi.oslo.osa to /root/.ssh/authorized_keys on Webserver.

```
echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQDTxQMP5W
ULLoWQWkmeK73y9Rnhqmko5vnTB1l/thuypvNQ/zuOOrXDQFd9/n
vFGYPLhQFqPyZ+cvJugNOHEHv6kCVjH1y9SLHlvfEDJ5hD+0TFcr
```

```
MbDAIu4Y/EU3wtdz8mIWn2xf5SMxdMycpWtER+drpQvibUAJvFlV
rFXS/WuF3+FbmCMMv4r29MJFTioT00nVuuZ+ArX/9h8qqNxl7Kc3
mI6vSLYDs7BOebw87U+UYRMJuvon9+4Zm5QQw5yLW035OZLFpjHO
C+3NxzLLA7eNvn4ykD4l7XJkQHxNDURLYGA4wiBFCa34S2V/qSNQ
exUUxM+zc2c1rJE38nuKCURcIdL9LjiaGqp6m1inT+kqIkUnE2Fg
xCXjZoMWRpqSzYSi4aCrYBGFIE7urBypZs4nS2wKQV7/NRpqv6CO
FND9viINSXEp6RfSaEqKrZ+teWiPpgx8MSNR5kqJI1VkE9wWWFJ/
l3EnjZxplIFH3OY6w3lYVMaaKKEhRdoghazzqLL847h3aABiqAAb
P/IW8OSGb7s9f2MOCedOd6tOGfVlPHQLxOlg05k+r9oFRnepeDOU
UgBwDIpfGnb+fUiZYBRknwf1HfWnp/qzJa+utgq8fTY1pHFqDc1W
1dvSAX9kIy7FaDB7qLpKgX5JrQbJwdESxsUzY9G1FTjEQowLPzuw
== usmanw@rumi.oslo.osa" \
> /root/.ssh/authorized_keys
```

4.1.13 Test SSH Access From Developer To GIT And Webserver

Test ssh connection from developers machine to Webserver and GIT server.

```
# ssh to GIT Server
usmanw@rumi:~$ ssh git@ahs-s1.ams.osa
Linux ahs-s1.ams.osa 3.2.0-0.bpo.4-amd64
    #1 SMP Debian 3.2.46-1+deb7u1~bpo60+1 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY,
to the extent permitted by applicable law.
Last login: Sun Nov  9 21:24:14 2014 from rumi.oslo.osa
git@ahs-s1:~$

# ssh to Webserver
usmanw@rumi:~$ ssh root@t09-09.oslo.osa
Linux t09-09 3.2.0-4-amd64 #1 SMP Debian 3.2.63-2+deb7u1 x86_64
Linux t09-09 2.6.32-5-amd64 #1 SMP Sun Sep 23 10:07:46 UTC 2012 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY,
to the extent permitted by applicable law.

Last login: Fri Nov 14 19:56:20 2014 from rumi.oslo.osa
t09-09:~#
```

4.1.14 Clone bodgeit.git On Developer Node

Clone Bodgeit Store GIT repository on developer's machine.

```
git clone git@ahs-s1.ams.osa:/home/git/bodgeit.git bodgeit
Cloning into 'bodgeit'...
remote: Counting objects: 97, done.
remote: Compressing objects: 100% (82/82), done.
remote: Total 97 (delta 15), reused 0 (delta 0)
Receiving objects: 100% (97/97), 991.19 KiB, done.
Resolving deltas: 100% (15/15), done.
```

Create GIT hook script in developer's local bodgeit cloned repository in order to rsync changes to the staging Webserver. Make sure rsync is installed on developer machine, Webserver and GIT server

```
cd /home/usmanw/bodgeit/.git/hooks

echo "#!/bin/sh" > post-commit
echo "cd /home/usmanw/bodgeit" > post-commit
echo "git pull /home/usmanw/bodgeit" > post-commit
echo "rsync -rvz --delete --exclude '*.git' --exclude '*-INF' \
/home/usmanw/bodgeit/ \
root@t09-09.oslo.osa:/var/lib/tomcat7/webapps/bodgeit" > post-commit

chmod 755 post-commit
```

Test GIT hook post-commit

```
sh /home/usmanw/bodgeit/.git/hooks/post-commit
From /home/usmanw/bodgeit
 * branch            HEAD       -> FETCH_HEAD
Already up-to-date.
...
...
sending incremental file list
deleting tests/
Readme
about.jsp
admin.jsp
advanced.jsp
basket.jsp
contact.jsp
footer.jsp
header.jsp
home.jsp
```

```
init.jsp
login.jsp
logout.jsp
password.jsp
product.jsp
register.jsp
score.jsp
search.jsp
style.css
js/encryption.js
js/util.js
...
...
sent 2426 bytes  received 10005 bytes  24862.00 bytes/sec
total size is 1136353  speedup is 91.41
```

4.1.15 Configurations

In order to log request headers, GET/POST payloads the following configuration changes need to be applied. Add the fields listed below to /var/lib/tomcat7/conf/server.xml so more information is logged in the access log file.

FIELD	DESCRIPTION
%t	Date Time
%h	Remote Hostname
%H	Request Protocol
%m	Request Method
%q	Query String
%U	Requested URL Path
%r	First Time of Request
%s	HTTP Status Code of Response
%b	Bytes Sent

Table 4.1: Tomcat7 Server Configuration Settings

```
# /var/lib/tomcat7/conf/server.xml
<Valve className="org.apache.catalina.valves.AccessLogValve"
  directory="logs"
  prefix="access_" suffix=".log"
  pattern="%t %h %H %m &quot;%{User-Agent}i&quot;
         %q %U &quot;%r&quot; %s %b"
/>
```

Add to /var/lib/tomcat7/conf/web.xml to use Tomcat7s Request Dumper Filter.

```
<filter>
    <filter-name>requestdumper</filter-name>
    <filter-class>
      org.apache.catalina.filters.RequestDumperFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>requestdumper</filter-name>
    <url-pattern>*</url-pattern>
</filter-mapping>
```

Modify /var/lib/tomcat7/conf/logging.properties to create a separate log file for the Request Dumper Filter output. The POST payloads are not recorded in the access logs so we create a separate logfile called 'request_dumper.DATE.log'.

```
handlers = 1catalina.org.apache.juli.FileHandler,
           2localhost.org.apache.juli.FileHandler,
           java.util.logging.ConsoleHandler,
           1request-dumper.org.apache.juli.FileHandler

1request-dumper.org.apache.juli.FileHandler.level = INFO
1request-dumper.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
1request-dumper.org.apache.juli.FileHandler.prefix = request_dumper.
1request-dumper.org.apache.juli.FileHandler.formatter = \
                        org.apache.juli.VerbatimFormatter
org.apache.catalina.filters.RequestDumperFilter.level = INFO
org.apache.catalina.filters.RequestDumperFilter.handlers = \
  1request-dumper.org.apache.juli.FileHandler
```

Ensure to restart tomcat7 once the above configuration changes have been applied.

```
/etc/init.d/tomcat7 restart
```

4.1.16 Load URLs Into MySQL DB

Load project Bodgeit Store data into projects table

```
INSERT INTO ‘projects‘ VALUES ( \
     4, \
    'Bodgeit Store', \
    'Usman Waheed', \
    'cosmicrhythm@hotmail.com' \
    ,1 \
    ,1 \
    ,1 \
);
```

Load URLs data into table: *urls*.

```
INSERT INTO ‘urls‘ VALUES \
(default,default,4,1,1,’http://t09-09.oslo.osa:8080/bodgeit/’,1,’GET’,”,”), \
(default,default,4,1,1,’http://t09-09.oslo.osa:8080/bodgeit/login.jsp’,2, \
 ’POST’,’username=cosmicrhythm@hotmail.com&password=1212’,’submit’), \
(default,default,4,1,1,’http://t09-09.oslo.osa:8080/bodgeit/contact.jsp’, \
 3,’POST’,’comments=empty’,’submit’), \
(default,default,4,1,1,’http://t09-09.oslo.osa:8080/bodgeit/about.jsp’, \
 4,’GET’,”,”), \
(default,default,4,1,1, \
 ’http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=28’, \
 5,’POST’,’quantity=empty’,’submit’), \
(default,default,4,1,1, \
 ’http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=3’, \
 6,’GET’,”,”), \
(default,default,4,1,1,’http://t09-09.oslo.osa:8080/bodgeit/search.jsp’,7, \
 ’POST’,’q=fjord’,’submit’), \
(default,default,4,1,1,’http://t09-09.oslo.osa:8080/bodgeit/register.jsp’,8, \
 ’POST’,’username=t@t.com&password1=testing&password2=testing2’, \
 ’submit’), \
(default,default,4,1,1, \
 ’http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=14’, \
 9,’GET’,”,”), \
(default,default,4,1,1,’http://t09-09.oslo.osa:8080/bodgeit/logout.jsp’,10, \
 ’GET’,”,”);
```

Box below shows the sort_order and URLs loaded for Bodgeit Store Project into the *urls* table in database SRT.

```
mysql> select sort_order,url from urls where project_id=4;
+------------+----------------------------------------------------------+
| sort_order | url                                                      |
+------------+----------------------------------------------------------+
|          1 | http://t09-09.oslo.osa:8080/bodgeit/                     |
|          2 | http://t09-09.oslo.osa:8080/bodgeit/login.jsp            |
|          3 | http://t09-09.oslo.osa:8080/bodgeit/contact.jsp          |
|          4 | http://t09-09.oslo.osa:8080/bodgeit/about.jsp            |
|          5 | http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=28 |
|          6 | http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=3  |
|          7 | http://t09-09.oslo.osa:8080/bodgeit/search.jsp           |
|          8 | http://t09-09.oslo.osa:8080/bodgeit/register.jsp         |
|          9 | http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=14 |
|         10 | http://t09-09.oslo.osa:8080/bodgeit/logout.jsp            |
+------------+-----------------------------------------------------------+
```

4.2 Data Analysis

This section describes the results collected and compiled for one test run. It also shows sample output from the logs that are collected and parsed latter on using Perl scripts to generate the tables with data.

Project: Bodgeit Store
Build #: 33
ATTACK STRENGTH = LOW , ALERT THRESHOLD = MEDIUM
URLs Proxied = 10

4.2.1 ZAP Log

Records from zap.log from one test run

```
2014-11-15 16:34:40,662 INFO  Scanner - scanner started
2014-11-15 16:34:40,679 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestPathTraversal strength LOW threshold MEDIUM
2014-11-15 16:34:42,627 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestPathTraversal in 1.947s
2014-11-15 16:34:42,627 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestRemoteFileInclude strength LOW threshold MEDIUM
2014-11-15 16:34:43,840 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestRemoteFileInclude in 1.213s
2014-11-15 16:34:43,840 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestRedirect strength LOW threshold MEDIUM
2014-11-15 16:34:44,886 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestRedirect in 1.046s
2014-11-15 16:34:44,887 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestServerSideInclude strength LOW threshold MEDIUM
2014-11-15 16:34:46,067 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestServerSideInclude in 1.18s
2014-11-15 16:34:46,067 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestCrossSiteScriptV2 strength LOW threshold MEDIUM
2014-11-15 16:34:47,195 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestCrossSiteScriptV2 in 1.127s
2014-11-15 16:34:47,196 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestSQLInjection strength LOW threshold MEDIUM
2014-11-15 16:34:49,085 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestSQLInjection in 1.889s
2014-11-15 16:34:49,085 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestDirectoryBrowsing strength LOW threshold MEDIUM
2014-11-15 16:34:50,302 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestDirectoryBrowsing in 1.217s
2014-11-15 16:34:50,302 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestInfoSessionIdURL strength LOW threshold MEDIUM
2014-11-15 16:34:50,312 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestInfoSessionIdURL in 0.01s
2014-11-15 16:34:50,312 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestClientBrowserCache strength LOW threshold MEDIUM
2014-11-15 16:34:50,321 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestClientBrowserCache in 0.008s
2014-11-15 16:34:50,321 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestExternalRedirect strength LOW threshold MEDIUM
2014-11-15 16:34:51,371 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestExternalRedirect in 1.05s
2014-11-15 16:34:51,371 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestInjectionCRLF strength LOW threshold MEDIUM
2014-11-15 16:34:53,008 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestInjectionCRLF in 1.636s
2014-11-15 16:34:53,008 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | TestParameterTamper strength LOW threshold MEDIUM
```

```
2014-11-15 16:34:54,246 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | TestParameterTamper in 1.237s
2014-11-15 16:34:54,247 INFO  HostProcess - start host http://t09-09.oslo.osa:8080 \
        | ScriptsActiveScanner strength LOW threshold MEDIUM
2014-11-15 16:34:55,058 INFO  HostProcess - completed host/plugin http://t09-09.oslo.osa:8080 \
        | ScriptsActiveScanner in 0.811s
2014-11-15 16:34:55,058 INFO  HostProcess - completed host http://t09-09.oslo.osa:8080 in 14.395s
2014-11-15 16:34:55,098 INFO  Scanner - scanner completed in 14.436s
```

Data Extracted from /home/zap/.ZAP/zap.log

ZAP PLUGIN	EXECUTION TIME (seconds)
TestInfoSessionIdURL	0.01
ScriptsActiveScanner	0.811
TestCrossSiteScriptV2	1.127
TestServerSideInclude	1.18
TestRemoteFileInclude	1.213
TestSQLInjection	1.889
TestExternalRedirect	1.05
TestPathTraversal,1.947	1.947
TestClientBrowserCache	0.008
TestParameterTamper	1.237
TestRedirect	1.046
TestInjectionCRLF	1.636
TestDirectoryBrowsing	1.217
Total	14.436

Table 4.2: ZAP Plugin Execution Times In Seconds

4.2.2 Access Log

Sample records from access log

```
...
...
[15/Nov/2014:16:34:41 +0000] 10.20.41.13 HTTP/1.1 GET "Mozilla/4.0 (compatible; MSIE 8.0;  \
  Windows NT 6.0)" ?q=fjord /bodgeit/search.jsp "GET /bodgeit/search.jsp?q=fjord HTTP/1.1" 200 2011
[15/Nov/2014:16:34:41 +0000] 10.20.41.13 HTTP/1.1 GET "Mozilla/4.0 (compatible; MSIE 8.0;  \
  Windows NT 6.0)" /bodgeit/register.jsp "GET /bodgeit/register.jsp HTTP/1.1" 200 2485
[15/Nov/2014:16:34:41 +0000] 10.20.41.13 HTTP/1.1 POST "Mozilla/4.0 (compatible; MSIE 8.0; \
  Windows NT 6.0)" /bodgeit/register.jsp "POST /bodgeit/register.jsp HTTP/1.1" 200 2562
...
...
...
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "Mozilla/4.0 (compatible; MSIE 8.0; \
  Windows NT 6.0)" /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "/etc/passwd" \
        /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "/Windows\system.ini" \
        /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "/WEB-INF/web.xml" \
        /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "\etc/passwd" \
        /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "\Windows\system.ini" \
        /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "\WEB-INF/web.xml" \
        /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "thishouldnotexistandhopefullyitwillnot" \
        /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
[15/Nov/2014:16:34:42 +0000] 10.20.41.13 HTTP/1.1 POST "Mozilla/4.0 (compatible; MSIE 8.0; \
  Windows NT 6.0)" /bodgeit/login.jsp "POST /bodgeit/login.jsp HTTP/1.1" 200 2530
...
...
```

Data Extracted from /var/lib/tomcat7/logs/access.2014-11-15.log

URL	GET Reqs	POST Reqs
http://t09-09.oslo.osa:8080/bodgeit/	2	0
http://t09-09.oslo.osa:8080/bodgeit/login.jsp	3	214
http://t09-09.oslo.osa:8080/bodgeit/contact.jsp	3	253
http://t09-09.oslo.osa:8080/bodgeit/about.jsp	3	0
http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=	2	0
http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=	2	0
http://t09-09.oslo.osa:8080/bodgeit/search.jsp	3	0
http://t09-09.oslo.osa:8080/bodgeit/register.jsp	3	253
http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=	2	0
http://t09-09.oslo.osa:8080/bodgeit/logout.jsp	3	0

Table 4.3: ZAP Active Scan Generated # Of GET and POST Requests

4.2.3 Request Dumper Log

One sample record (START to FINISH) from request_dumper log

```
http-bio-8080-exec-2 ===============================================================
http-bio-8080-exec-3 START TIME        =15-Nov-2014 16:34:41
http-bio-8080-exec-3         requestURI=/bodgeit/login.jsp
http-bio-8080-exec-3           authType=null
http-bio-8080-exec-3  characterEncoding=null
http-bio-8080-exec-3      contentLength=49
http-bio-8080-exec-3        contentType=application/x-www-form-urlencoded
http-bio-8080-exec-3        contextPath=/bodgeit
http-bio-8080-exec-3             cookie=JSESSIONID=37B19BFB810E27FA981C3FE1738F5610
http-bio-8080-exec-3             header=user-agent=Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0)
http-bio-8080-exec-3             header=accept-language=en-us
http-bio-8080-exec-3             header=referer=http://t09-09.oslo.osa:8080/bodgeit/login.jsp
http-bio-8080-exec-3             header=accept=*/*
http-bio-8080-exec-3             header=content-length=49
http-bio-8080-exec-3             header=content-type=application/x-www-form-urlencoded
http-bio-8080-exec-3             header=proxy-connection=Keep-Alive
http-bio-8080-exec-3             header=cookie=JSESSIONID=37B19BFB810E27FA981C3FE1738F5610
http-bio-8080-exec-3             header=host=t09-09.oslo.osa:8080
http-bio-8080-exec-3             locale=en_US
http-bio-8080-exec-3             method=POST
http-bio-8080-exec-3          parameter=username=cosmicrhythm@hotmail.com
http-bio-8080-exec-3          parameter=password=1212
http-bio-8080-exec-3           pathInfo=null
http-bio-8080-exec-3           protocol=HTTP/1.1
http-bio-8080-exec-3        queryString=null
http-bio-8080-exec-3         remoteAddr=10.20.41.13
http-bio-8080-exec-3         remoteHost=10.20.41.13
http-bio-8080-exec-3         remoteUser=null
http-bio-8080-exec-3 requestedSessionId=37B19BFB810E27FA981C3FE1738F5610
http-bio-8080-exec-3             scheme=http
http-bio-8080-exec-3         serverName=t09-09.oslo.osa
http-bio-8080-exec-3         serverPort=8080
http-bio-8080-exec-3        servletPath=/login.jsp
http-bio-8080-exec-3           isSecure=false
http-bio-8080-exec-3 ------------------=--------------------------------------------
http-bio-8080-exec-3 ------------------=--------------------------------------------
http-bio-8080-exec-3           authType=null
```

```
http-bio-8080-exec-3         contentType=text/html;charset=ISO-8859-1
http-bio-8080-exec-3          remoteUser=null
http-bio-8080-exec-3              status=200
http-bio-8080-exec-3 END TIME          =15-Nov-2014 16:34:41
http-bio-8080-exec-3 ===============================================================
```

Data Extracted from /var/lib/tomcat7/logs/request_dumper.2014-11-15.log

TYPE	NAME	# of Uniques
header	content-length	83
header	set-cookie	7
header	proxy-connection	1
header	content-type	1
header	Set-Cookie	32
header	accept-language	1
header	referer	48
header	host	1
header	user-agent	39
header	cookie	1
header	accept	1
parameter	anticsrf	35
parameter	productid	32
parameter	password1	35
parameter	password	37
parameter	password2	35
parameter	username	39
parameter	price	30
parameter	null	32
parameter	typeid	36
parameter	q	34
parameter	comments	35
parameter	prodid	37
parameter	quantity	31

Table 4.4: Metrics from request dumper log

4.2.4 Results Log

Sample output from bodgeit-srt_2014-11-15-16-34-37.txt

```
Project: bodgeit
Filename: bodgeit-srt_2014-11-15-16-34-37
Risks: High, Medium, Low and Informational

URL: http://t09-09.oslo.osa:8080/bodgeit/
URL Params (if any):
Risk: Low
Reliability: Warning
Description: The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'
```

```
Reference:
Solution: This check is specific to Internet Explorer 8 and Google Chrome. \
Ensure each page sets a Content-Type header and the X-CONTENT-TYPE-OPTIONS if the Content-Type\
  header is unknown
--------------------------------------------------------------------------------------------------------------------
URL: http://t09-09.oslo.osa:8080/bodgeit/
URL Params (if any):
Risk: Informational
Reliability: Warning
Description: X-Frame-Options header is not included in the HTTP response to protect against \
  'ClickJacking' attacks
Reference: http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/ \
combating-clickjacking-with-x-frame-options.aspx?Redirected=true
Solution: Most modern Web browsers support the X-Frame-Options HTTP header, \
ensure it's set on all web pages returned by your site \
(if you expect the page to be framed only by pages on your server \
(e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, \
otherwise if you never expect the page to be framed, you should use DENY).
--------------------------------------------------------------------------------------------------------------------
...
...
...
Summary of Alerts
High Risk Alerts: 1
Medium Risk Alerts: 0
Low Risk Alerts: 14
Informational Risk Alerts: 12
Total: 27
```

Alerts Extracted from http://ahs-m1.ams.osa/srt/bodgeit/bodgeit-srt_2014-11-15-16-34-37.txt

Risk Type	# of Alerts
High	1
Medium	0
Low	14
Informational	12
Total	27

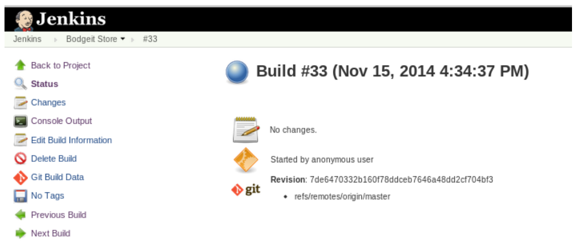
Table 4.5: Summary Of Alerts

4.3 Experiments

4.3.1 Test Case: 1(a)

Result from Manual Build Run

```
No changes.
Started by anonymous user
Revision: 7de6470332b160f78ddceb7646a48dd2cf704bf3
refs/remotes/origin/master
```



Build #33 (Nov 15, 2014 4:34:37 PM)

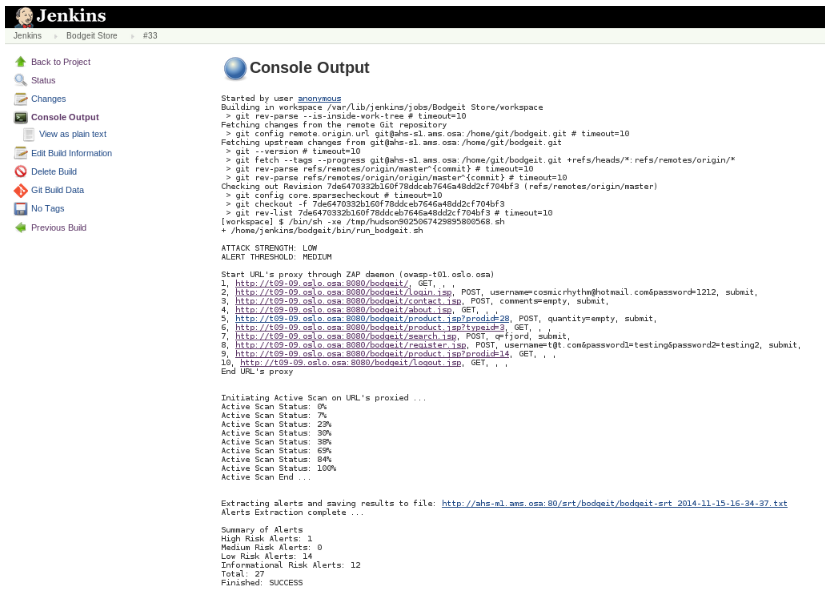
No changes.

Started by anonymous user

Revision: 7de6470332b160f78ddceb7646a48dd2cf704bf3

- refs/remotes/origin/master

Figure 4.1: Triggered By Manual Click



Console Output

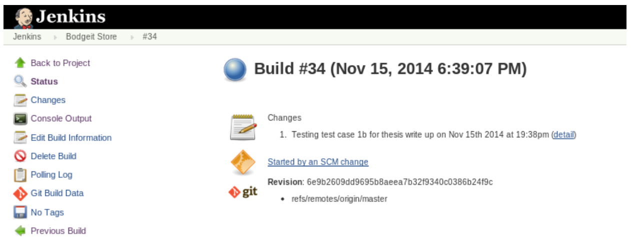
```
Started by user anonymous
Building in workspace /var/lib/jenkins/jobs/Bodgeit Store/workspace
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@ahs-sl.ams.osa:/home/git/bodgeit.git # timeout=10
Fetching upstream changes from git@ahs-sl.ams.osa:/home/git/bodgeit.git
 > git --version # timeout=10
 > git fetch --tags --progress git@ahs-sl.ams.osa:/home/git/bodgeit.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 7de6470332b160f78ddceb7646a48dd2cf704bf3 (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 7de6470332b160f78ddceb7646a48dd2cf704bf3
 > git rev-list 7de6470332b160f78ddceb7646a48dd2cf704bf3 # timeout=10
[workspace] $ /bin/sh -xe /tmp/hudson9025067429895800568.sh
+ /home/jenkins/bodgeit/bin/run_bodgeit.sh

ATTACK STRENGTH: LOW
ALERT THRESHOLD: MEDIUM

Start URL's proxy through ZAP daemon (owasp-t01.oslo.osa)
1, http://t09-09.oslo.osa:8080/bodgeit/, GET, , ,
2, http://t09-09.oslo.osa:8080/bodgeit/login.jsp, POST, username=cosmicrhythm@hotmail.com&password=1212, submit,
3, http://t09-09.oslo.osa:8080/bodgeit/contact.jsp, POST, comments=empty, submit,
4, http://t09-09.oslo.osa:8080/bodgeit/about.jsp, GET, , ,
5, http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=28, POST, quantity=empty, submit,
6, http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=3, GET, , ,
7, http://t09-09.oslo.osa:8080/bodgeit/search.jsp, POST, q=fjord, submit,
8, http://t09-09.oslo.osa:8080/bodgeit/register.jsp, POST, username=t@t.com&password1=testing&password2=testing2, submit,
9, http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=14, GET, , ,
10, http://t09-09.oslo.osa:8080/bodgeit/logout.jsp, GET, , ,
End URL's proxy


Initiating Active Scan on URL's proxied ...
Active Scan Status: 0%
Active Scan Status: 7%
Active Scan Status: 23%
Active Scan Status: 30%
Active Scan Status: 38%
Active Scan Status: 69%
Active Scan Status: 84%
Active Scan Status: 100%
Active Scan End ...


Extracting alerts and saving results to file: http://ahs-ml.ams.osa:80/srt/bodgeit/bodgeit-srt_2014-11-15-16-34-37.txt
Alerts Extraction complete ...

Summary of Alerts
High Risk Alerts: 1
Medium Risk Alerts: 0
Low Risk Alerts: 14
Informational Risk Alerts: 12
Total: 27
Finished: SUCCESS
```

Figure 4.2: Manual Test Build

4.3.2 Test Case: 1(b)

Result from Automated Test Build triggered from source code commit

Changes
1. Testing test case 1b for thesis write up on Nov 15th 2014 at 19:38pm (detail)
Started by an SCM change
Revision: 6e9b2609dd9695b8aeea7b32f9340c0386b24f9c
refs/remotes/origin/master



Build #34 (Nov 15, 2014 6:39:07 PM)

Changes

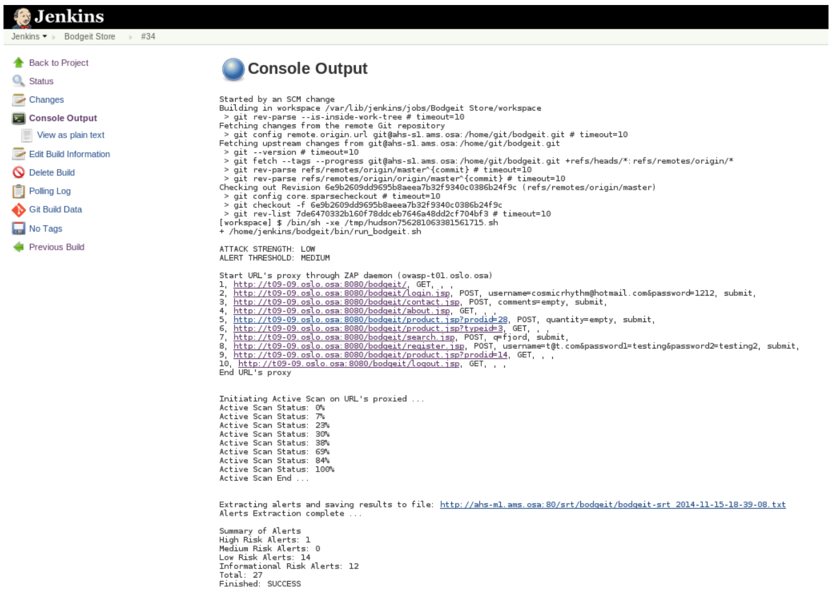
1. Testing test case 1b for thesis write up on Nov 15th 2014 at 19:38pm (detail)

Started by an SCM change

Revision: 6e9b2609dd9695b8aeea7b32f9340c0386b24f9c

- refs/remotes/origin/master

Figure 4.3: Code Commit Triggered Build



Console Output

```
Started by an SCM change
Building in workspace /var/lib/jenkins/jobs/Bodgeit Store/workspace
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url git@ahs-sl.ams.osa:/home/git/bodgeit.git # timeout=10
Fetching upstream changes from git@ahs-sl.ams.osa:/home/git/bodgeit.git
 > git --version # timeout=10
 > git fetch --tags --progress git@ahs-sl.ams.osa:/home/git/bodgeit.git +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 6e9b2609dd9695b8aeea7b32f9340c0386b24f9c (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 6e9b2609dd9695b8aeea7b32f9340c0386b24f9c
 > git rev-list 7de6470332b160f78ddceb7646a48dd2cf704bf3 # timeout=10
[workspace] $ /bin/sh -xe /tmp/hudson7562810633815617l5.sh
+ /home/jenkins/bodgeit/bin/run_bodgeit.sh

ATTACK STRENGTH: LOW
ALERT THRESHOLD: MEDIUM

Start URL's proxy through ZAP daemon (owasp-t01.oslo.osa)
1, http://t09-09.oslo.osa:8080/bodgeit/, GET, , ,
2, http://t09-09.oslo.osa:8080/bodgeit/login.jsp, POST, username=cosmicrhythm@hotmail.com&password=1212, submit,
3, http://t09-09.oslo.osa:8080/bodgeit/contact.jsp, POST, comments=empty, submit,
4, http://t09-09.oslo.osa:8080/bodgeit/about.jsp, GET, , ,
5, http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=28, POST, quantity=empty, submit,
6, http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=3, GET, , ,
7, http://t09-09.oslo.osa:8080/bodgeit/search.jsp, POST, q=fjord, submit,
8, http://t09-09.oslo.osa:8080/bodgeit/register.jsp, POST, username=t@t.com&password1=testing&password2=testing2, submit,
9, http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=14, GET, , ,
10, http://t09-09.oslo.osa:8080/bodgeit/logout.jsp, GET, , ,
End URL's proxy


Initiating Active Scan on URL's proxied ...
Active Scan Status: 0%
Active Scan Status: 7%
Active Scan Status: 23%
Active Scan Status: 30%
Active Scan Status: 38%
Active Scan Status: 69%
Active Scan Status: 84%
Active Scan Status: 100%
Active Scan End ...


Extracting alerts and saving results to file: http://ahs-ml.ams.osa:80/srt/bodgeit/bodgeit-srt_2014-11-15-18-39-08.txt
Alerts Extraction complete ...

Summary of Alerts
High Risk Alerts: 1
Medium Risk Alerts: 0
Low Risk Alerts: 14
Informational Risk Alerts: 12
Total: 27
Finished: SUCCESS
```

Figure 4.4: Build Through SCM Change

4.3.3 Test Case: 2(a)

Build #: 42

ZAP PLUGIN	EXECUTION TIME (seconds)
TestSQLInjection	4.809
TestServerSideInclude	1.171
ScriptsActiveScanner	1.01
TestExternalRedirect	1.042
TestInfoSessionIdURL	0.01
TestClientBrowserCache	0.014
TestPathTraversal	2.935
TestParameterTamper	1.448
TestRedirect	1.047
TestRemoteFileInclude	2.013
TestDirectoryBrowsing	1.215
TestCrossSiteScriptV2	1.116
TestInjectionCRLF	1.633
Total	19.526

Table 4.6: ZAP Plugin Execution Times, Test Case 2(a)

SORTID	URL	GET Reqs	POST Reqs
1	http://t09-09.oslo.osa:8080/bodgeit/	3	0
2	http://t09-09.oslo.osa:8080/bodgeit/login.jsp	6	356
3	http://t09-09.oslo.osa:8080/bodgeit/contact.jsp	4	407
4	http://t09-09.oslo.osa:8080/bodgeit/about.jsp	4	0
5	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=	3	0
6	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=	3	0
7	http://t09-09.oslo.osa:8080/bodgeit/search.jsp	4	0
8	http://t09-09.oslo.osa:8080/bodgeit/register.jsp	6	434
9	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=	3	0
10	http://t09-09.oslo.osa:8080/bodgeit/logout.jsp	4	0
	Total	40	1197

Table 4.7: ZAP Active Scan Generated # Of GET And POST Requests, Test Case 2(a)

Risk Type	# of Alerts
High	2
Medium	0
Low	14
Informational	12
Total	28

Table 4.8: Summary Of Alerts, Test Case 2(a)

TYPE	NAME	# of Uniques
header	accept	2
header	user-agent	70
header	content-type	1
header	referer	127
header	content-length	109
header	if-none-match	2
header	Set-Cookie	49
header	cookie	2
header	host	1
header	accept-language	2
header	proxy-connection	1
header	set-cookie	7
header	ETag	2
header	accept-encoding	1
header	Last-Modified	1
header	if-modified-since	1
header	Accept-Ranges	1
header	connection	1
parameter	password2	61
parameter	prodid	63
parameter	anticsrf	61
parameter	productid	49
parameter	comments	61
parameter	quantity	55
parameter	password1	61
parameter	q	60
parameter	null	56
parameter	price	56
parameter	username	76
parameter	password	61
parameter	typeid	62

Table 4.9: Metrics From Request Dumper Log, Test Case 2(a)

4.3.4 Test Case: 2(b)

Build #: 43

ZAP PLUGIN	EXECUTION TIME (seconds)
TestRemoteFileInclude	3.491
ScriptsActiveScanner	0.811
TestDirectoryBrowsing	1.216
TestParameterTamper	1.44
TestClientBrowserCache	0.009
TestInjectionCRLF	1.436
TestPathTraversal	3.781
TestExternalRedirect	1.044
TestRedirect	1.044
TestServerSideInclude	1.144
TestInfoSessionIdURL	0.01
TestSQLInjection	6.855
TestCrossSiteScriptV2	1.316
Total	23.679

Table 4.10: ZAP Plugin Execution Times, Test Case 2(b)

SORTID	URL	GET Reqs	POST Reqs
1	http://t09-09.oslo.osa:8080/bodgeit/	2	0
2	http://t09-09.oslo.osa:8080/bodgeit/login.jsp	3	521
3	http://t09-09.oslo.osa:8080/bodgeit/contact.jsp	3	557
4	http://t09-09.oslo.osa:8080/bodgeit/about.jsp	3	0
5	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=	2	0
6	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=	2	0
7	http://t09-09.oslo.osa:8080/bodgeit/search.jsp	3	0
8	http://t09-09.oslo.osa:8080/bodgeit/register.jsp	3	632
9	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=	2	0
10	http://t09-09.oslo.osa:8080/bodgeit/logout.jsp	3	0
	Total	26	1710

Table 4.11: ZAP Active Scan Generated # Of GET And POST Requests, Test Case 2(b)

Risk Type	# of Alerts
High	4
Medium	0
Low	14
Informational	12
Total	30

Table 4.12: Summary Of Alerts, Test Case 2(b)

TYPE	NAME	# of Uniques
header	content-type	1
header	Set-Cookie	48
header	host	1
header	proxy-connection	1
header	set-cookie	7
header	cookie	1
header	content-length	124
header	accept-language	1
header	user-agent	100
header	accept	1
header	referer	196
parameter	password1	90
parameter	password	85
parameter	prodid	88
parameter	price	86
parameter	productid	69
parameter	quantity	91
parameter	null	79
parameter	q	97
parameter	anticsrf	86
parameter	comments	86
parameter	password2	90
parameter	typeid	87

Table 4.13: Metrics From Request Dumper Log, Test Case 2(b)

4.3.5 Test Case: 2(c)

Build #: 43

ZAP PLUGIN	EXECUTION TIME (seconds)
TestClientBrowserCache	0.008
TestParameterTamper	1.229
TestPathTraversal	21.49
TestServerSideInclude	1.139
TestRedirect	1.244
TestCrossSiteScriptV2	1.112
TestExternalRedirect	1.043
TestRemoteFileInclude	4.48
TestInjectionCRLF	1.431
TestSQLInjection	8.249
TestDirectoryBrowsing	1.218
ScriptsActiveScanner	1.011
TestInfoSessionIdURL	0.008
Total	43.721

Table 4.14: ZAP Plugin Execution Times, Test Case 2(c)

SORTID	URL	GET Reqs	POST Reqs
1	http://t09-09.oslo.osa:8080/bodgeit/	2	0
2	http://t09-09.oslo.osa:8080/bodgeit/login.jsp	3	1492
3	http://t09-09.oslo.osa:8080/bodgeit/contact.jsp	3	1619
4	http://t09-09.oslo.osa:8080/bodgeit/about.jsp	3	0
5	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=	2	0
6	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=	2	0
7	http://t09-09.oslo.osa:8080/bodgeit/search.jsp	3	0
8	http://t09-09.oslo.osa:8080/bodgeit/register.jsp	3	1839
9	http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=	2	0
10	http://t09-09.oslo.osa:8080/bodgeit/logout.jsp	3	0
	Total	26	4950

Table 4.15: ZAP Active Scan Generated # Of GET and POST Requests, Test Case 2(c)

Risk Type	# of Alerts
High	4
Medium	0
Low	14
Informational	12
Total	30

Table 4.16: Summary Of Alerts, Test Case 2(c)

TYPE	NAME	# of Uniques
header	host	1
header	content-type	1
header	set-cookie	7
header	content-length	124
header	accept	1
header	Set-Cookie	225
header	proxy-connection	1
header	accept-language	1
header	cookie	1
header	user-agent	306
header	referer	443
parameter	quantity	291
parameter	null	250
parameter	prodid	259
parameter	productid	240
parameter	q	272
parameter	price	297
parameter	username	310
parameter	anticsrf	257
parameter	password	256
parameter	password1	273
parameter	comments	257
parameter	password2	273
parameter	typeid	258

Table 4.17: Metrics From Request Dumper Log, Test Case 2(c)

4.4 Return On Investment (ROI)

eq1 = Manual Investment Cost = 0 + (80 * 50) + $1000 = $5000
eq2 = Automation Investment Cost = 0 + (80 * 50) + $5000 = $9000

eq3 = Manual Implementation Cost = 10 * 50 * 8 = $4000
eq4 = Automated Implementation Cost = 10 * 50 * 80 = $40000

eq5 = Manual Testing 10 urls Cost = 10 * 50 * 4 = $2000
eq6 = Automated Testing 10 urls Cost = 10 * 50 * 1 = $500

Total Manual Cost = *eq1* + *eq3* + (*Builds* - 1) * eq5
Total Automated Cost = *eq2* + *eq4* + (*Builds* - 1) * eq6

The Investment in Automation = eq2 + eq4 = $49000
For Builds = 1
Total Manual Cost = $5000 + $4000 + (1 - 1) * $2000 = $9000
Total Automated Cost = $9000 + $40000 + (1 - 1) * $500 = $49000

$$ROI = \frac{TotalManualCost - TotalAutomatedCost}{TotalAutomatedCost} \tag{4.1}$$

$$ROI = \frac{\$9000 - \$49000}{\$49000} = -0.8163 \tag{4.2}$$

Results for Builds = {1,10,20...100} noted in the table below:

Builds	Total Manual Cost ($)	Total Automated Cost ($)	ROI
1	9000	49000	-0.8163
10	27000	53500	-0.5408
20	47000	58500	-0.2347
30	67000	63500	0.0714
40	87000	68500	0.3776
50	107000	73500	0.6837
60	127000	78500	0.9898
70	147000	83500	1.2959
80	167000	88500	1.6020
90	187000	93500	1.9082
100	207000	98500	2.2143

Table 4.18: ROI Calculations

Chapter 5

Analysis

5.1 Test Case: 1(a)

The results for this test case show that the manual build which was initiated by clicking on *"Build Now"* in the Jenkins GUI executed without any error messages.
In this test the Java application via the shell script */home/jenkins/bodgeit/bin/run_bodgeit.sh* retrieved URLs from the MySQL DB using project= 4 (Bodgeit Store) and proxied them through ZAP to the staging Webserver. ZAP was set to run with *ATTACK STRENGTH = LOW*. The GIT server was not involved in this manual test.

The results from the scan were made available at:
http://ahs-m1.ams.osa/srt/bodgeit/bodgeit-srt_2014-11-15-16-34-37.txt.
This was a successful manual test that ensured the setup excluding GIT works as intended.

5.2 Test Case: 1(b)

In this test a code commit was executed from the developer machine rumi.oslo.osa. The code commit was a change to a Readme file in the Bodgeit Store project. The GIT commit output obtained from the Jenkins GUI is noted below:

```
Commit 6e9b2609dd9695b8aeea7b32f9340c0386b24f9c by usmanw
Testing test case 1b for thesis write up on Nov 15th 2014 at 19:38pm
```

The GIT commit noted above executed a GIT hook *post-commit* script in the developers local sandbox. This hook script rsynced the local GIT repository in the developers sandbox for the Bodgeit Store project to the staging Webserver. A developer can first test on their local machine and post unit testing, commit their changes to the GIT server and at the same time rsync to the staging Webserver. Jenkins polling the GIT server triggers a security regression test build on the changes that were committed. It reports what changed, runs the execution of URL testing using ZAP and generates a report. All URLs in the MySQL DB for the project will be tested with ZAP plugins. If any vulnerability is introduced by the developer it will be either flagged given ZAP catches it or a regression might be caught. A regression would mean that some URL that passed the security test before now reports a vulnerability. The *Risk Alerts* provide this information.

The results demonstrate a complete automated process using a framework which has Jenkins, ZAP, MySQL, GIT, staging Webserver and developer sandbox. The Active Scanner in ZAP performs the security tests using its built in plugins. The execution times of the plugins are reported to the zap.log. The number of requests from ZAP can be retrieved from the access log on the staging Webserver. The Tomcat7 request dumper log will provide more detailed information about the headers and GET/POST parameters that were fuzzed. The ATTACK STRENGTH of ZAP helps determine how much thorough testing can be performed and what results we obtain. One good feature of ZAP that we see from the results is its ability to provide a resolution to the reported *Risk Alert*. These resolutions are also made available in the scan results txt file that resides on the Jenkins server at */var/www/srt/bodgeit/* directory.

5.3 Test Case: 2(a)

5.3.1 Verification Of High Risk Alerts

Let us first analyze the results from the Active Scan executed by ZAP with ATTACK STRENGTH = MEDIUM. Based on the results two high risk alerts were reported which are listed below:

1. URL: http://t09-09.oslo.osa:8080/bodgeit/search.jsp?q=hello<script>alert("xss");</script> vulnerable to XSS
2. URL: http://t09-09.oslo.osa:8080/bodgeit/basket.jsp SQL Injection may be possible

The first one can be verified easily by loading the search URL into the browser address bar to see if anything is reflected back. The figure below verifies that the search page is indeed susceptible to a Cross Site Scripting Attack.

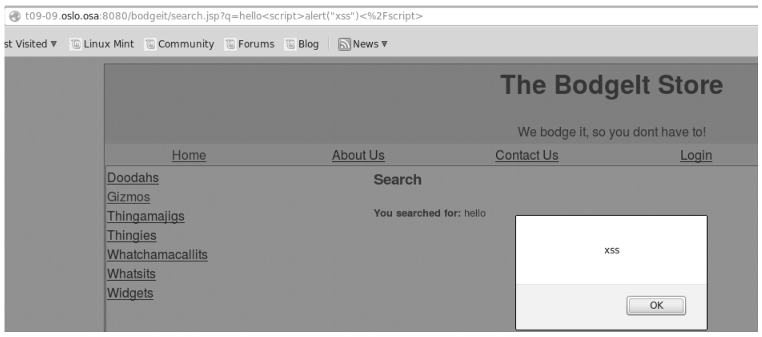
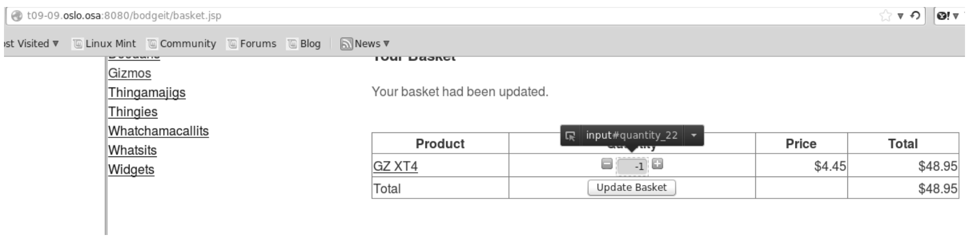


Figure 5.1: XSS Vulnerability In search.jsp

The second alert is not really an SQL Injection but the parameters to basket.jsp can be tampered. The quantity field can be set to a -1 value and then submitted through the POST form. This means basket.jsp is not checking the validity of the inputs correctly. See image below:



Your basket had been updated.

Product	Quantity	Price	Total
GZ XT4	-1	$4.45	$48.95
Total			$48.95

Figure 5.2: Tamper Quantity Parameter In basket.jsp

5.3.2 Plugins Execution Time And Fuzzing

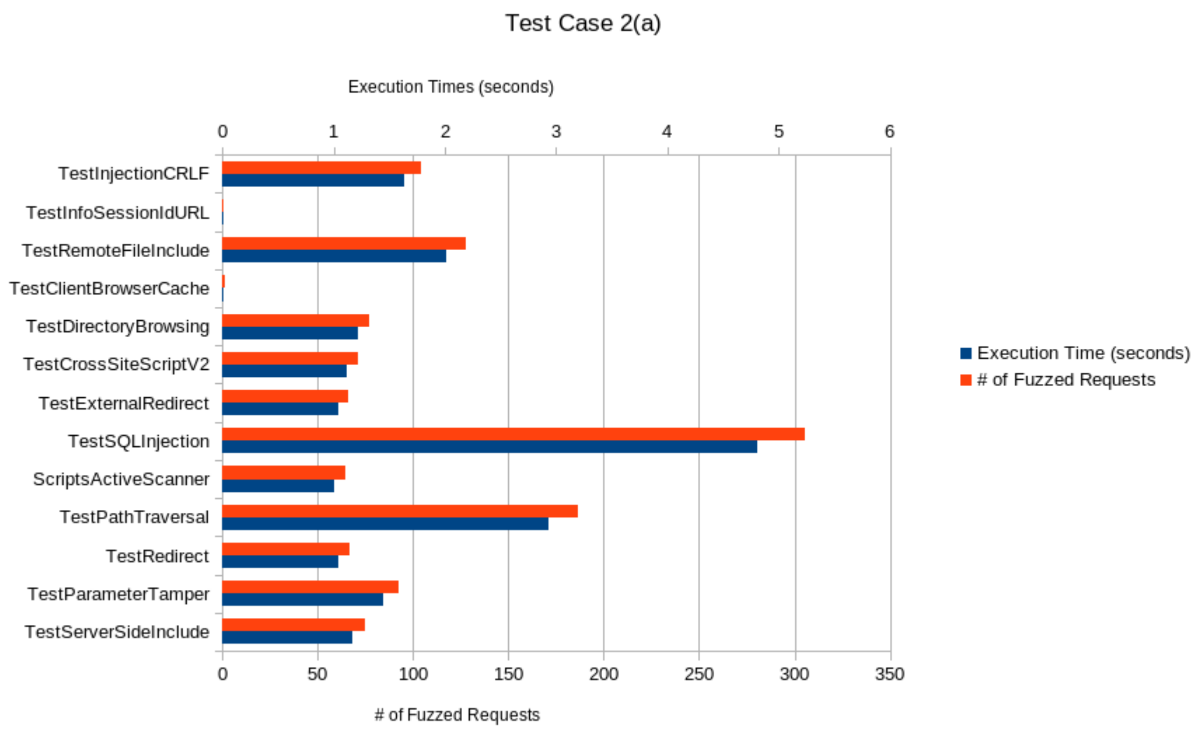


Figure 5.3: ZAP Plugins Execution Time And Fuzzing Of Requests, Test Case 2(a)

From the graph above we can infer that the ZAP plugin *TestSQLInjection* did most of the work followed by *TestPathTraversal* and then *TestRemoteFileInclude*. The 2 plugin executed approximately 71 fuzzed requests over a duration of 1.16 seconds. Both plugins *TestSQLInjection* and *TestCrossSiteScriptV2* succeeded in catching 1 high risk vulnerability each.

5.3.3 Requests To Proxied URLs

In the next graph we analyze how ZAP distributed the GET and POST requests across the 10 URLs that were proxied through it on their way to the staging Webserver. This should help us identify which URLs were fuzzed more compared to others.

The SORTIDs {2,3.8} all POST requests received the most number of requests from ZAP. These three URLs were:

- (2) http://t09-09.oslo.osa:8080/bodgeit/login.jsp

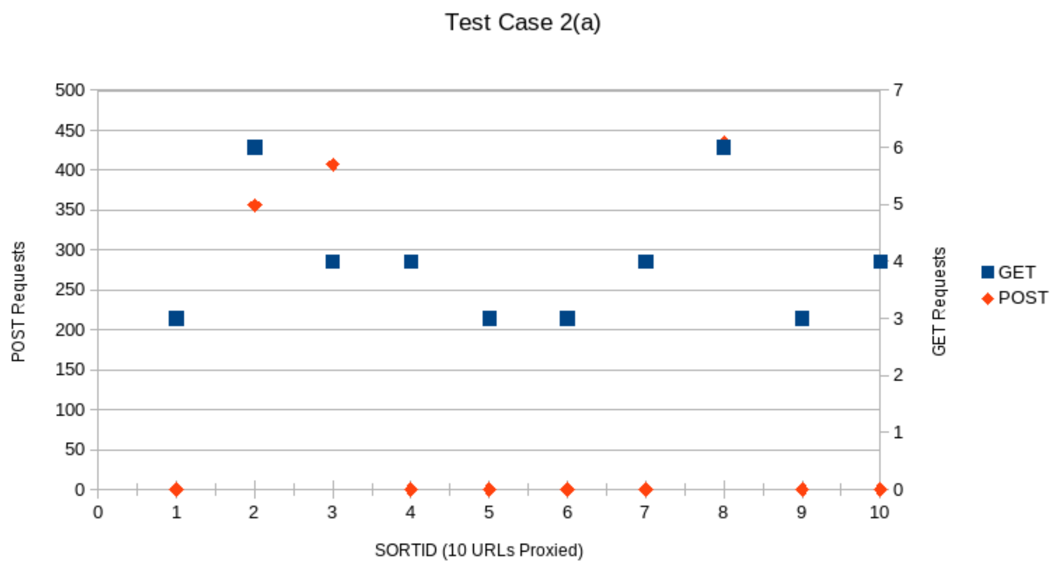


Figure 5.4: Distribution Of GET And POST Requests, Test Case 2(a)

- (3) http://t09-09.oslo.osa:8080/bodgeit/contact.jsp
- (8) http://t09-09.oslo.osa:8080/bodgeit/register.jsp

All of the three Java Server Pages were input forms where the user had to fill with information before submitting. The GET URLs were evenly distributed averaging around 4 fuzzed requests per URL. ZAP did not spend much time fuzzing the GET URLs as compared to the POST forms.

5.3.4 Fuzzing HTTP Headers And FORM Payload Parameters

The next graph shows how many unique strings each HTTP header and payload FORM parameter received from ZAP. This data was collected from the request dumper log on the Tomcat7 staging Webserver. It is interesting to observe that the FORM payloads (parameters) all average around 60 unique strings which means that ZAP sent a uniform distribution of input strings to all of them. As far as the HTTP headers are concerned:

- referrer
- user-agent

Referrer and *User-Agent* had the highest number of unique strings in the HTTP Header set. This means these two were probably fuzzed the most out of all. The *content-length* HTTP header should be ignored here because that depends on the number of characters in the REQUEST BODY of the HTTP Request.

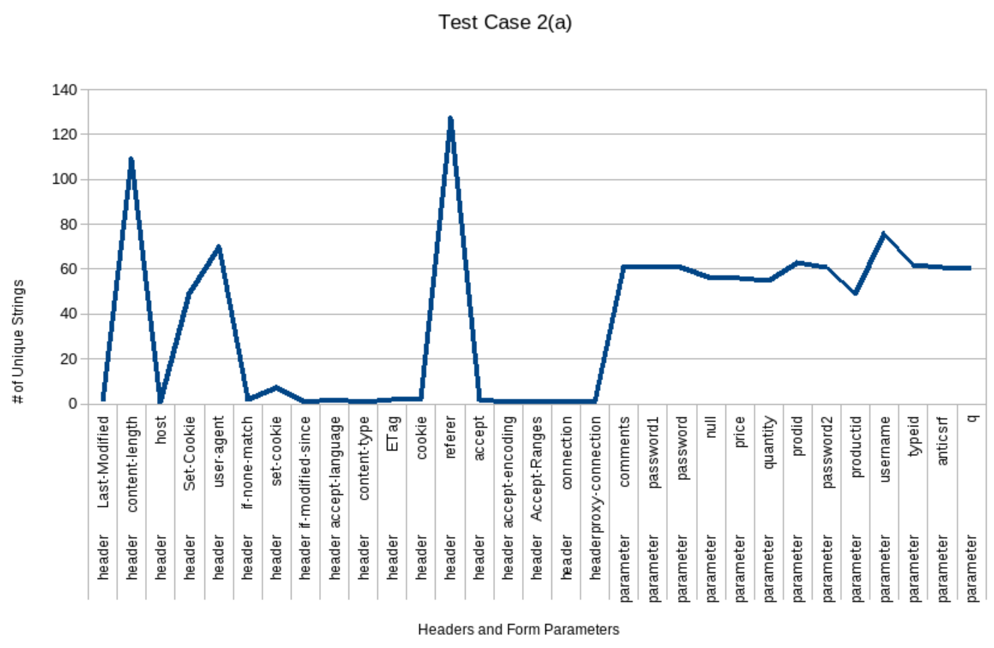


Figure 5.5: # Of Unique HTTP Header And FORM Parameter Strings, Test Case 2(a)

5.4 Test Case: 2(b)

5.4.1 Verification Of High Risk Alerts

The Active Scan was set to ATTACK STRENGTH = HIGH and with this setting four high risk alerts were reported. These four high risk alerts are noted below:

1. http://t09-09.oslo.osa:8080/bodgeit/search.jsp?q=hello<Script>alert('XSS');</Script> vulnerable to XSS

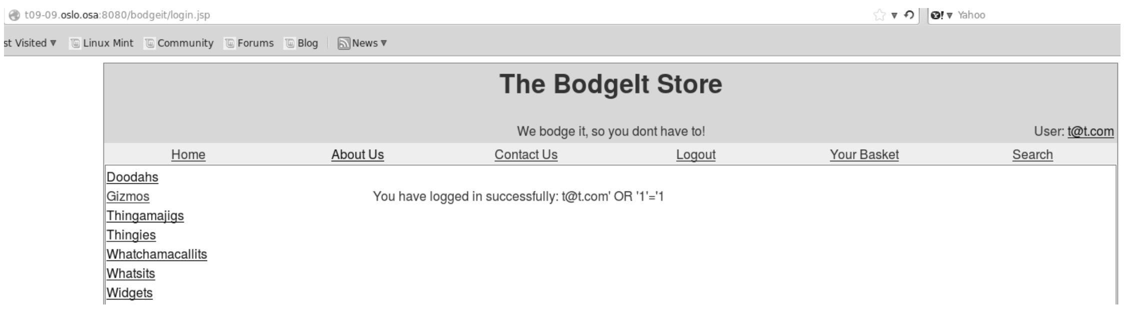
2. http://t09-09.oslo.osa:8080/bodgeit/login.jsp SQL Injection maybe possible

3. http://t09-09.oslo.osa:8080/bodgeit/basket.jsp SQL Injection maybe possible

4. http://t09-09.oslo.osa:8080/bodgeit/contact.jsp vulnerable to XSS

Number 1 and 3 were already discussed in test case 2(a) so let's verify 2 and 4. In order to check the login.jsp page for SQL injection we shall inject for the *Username* and leave the password empty. Please keep in mind that the Username 't@t.com' is a registered user. The SQL injection string is:

t@t.com' OR '1'='1



The BodgeIt Store

We bodge it, so you dont have to!

User: t@t.com

Home About Us Contact Us Logout Your Basket Search

Doodahs
Gizmos
Thingamajigs
Thingies
Whatchamacallits
Whatsits
Widgets

You have logged in successfully: t@t.com' OR '1'='1

Figure 5.6: SQL Injection In login.jsp, Test Case 2(b)

The result as we see in the image above shows that we were able to login without a password. Basically some part of the SQL in login.jsp got bypassed using this attack vector.

Load contact.jsp in the browser address bar and input the string *<Script>alert('XSS');</Script>* in the web form to see if anything gets reflected back. One other way to verify is to look at the source of the HTML page returned.

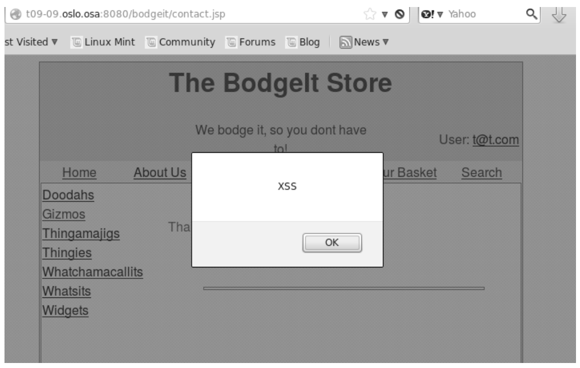


Figure 5.7: XSS Vulnerability In contact.jsp, Test Case 2(b)

5.4.2 Plugins Execution Time And Fuzzing

With ATTACK STRENGTH set to HIGH we expect ZAP to send more requests to the Webserver. Basically sending more combinations of input injection strings to the 10 URLs we are interested in security testing. Let us look at the plugin time execution and # of requests graph below and see if that is the case.

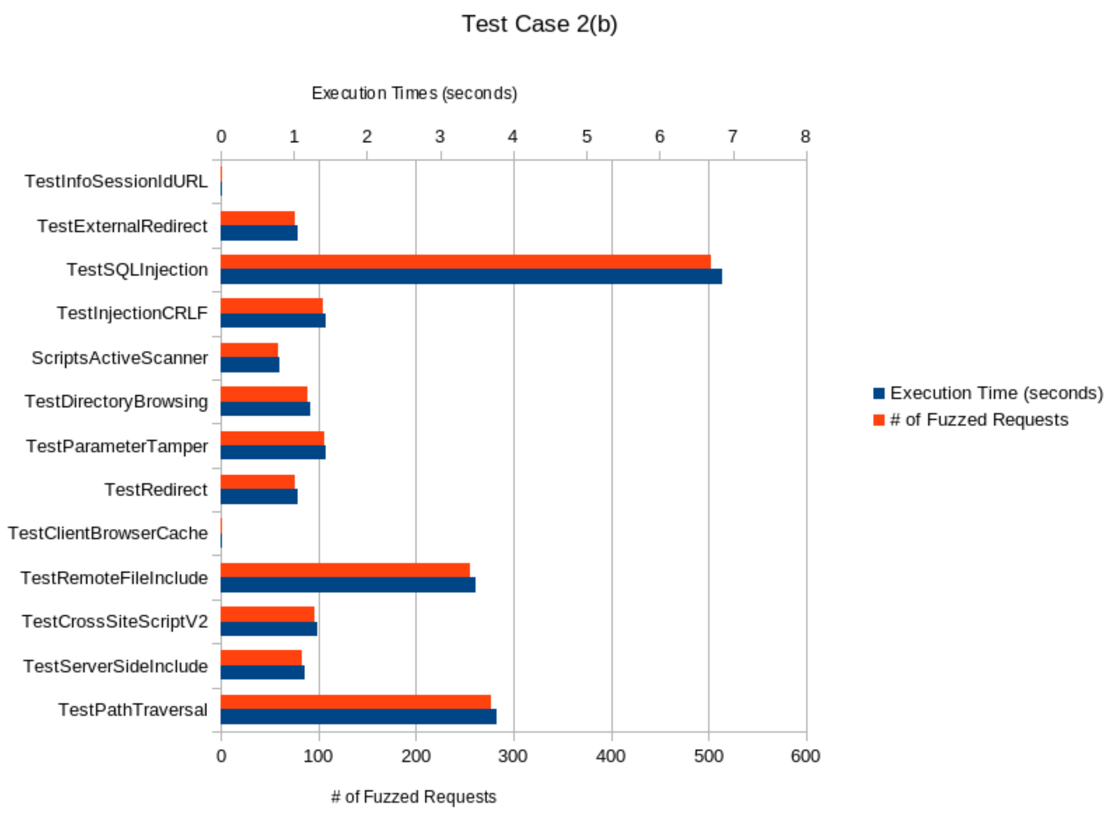


Figure 5.8: ZAP Plugins Execution Time And Fuzzing Of Requests, Test Case 2(b)

With a more potent attack setting (HIGH), ZAP increased the number of attack strings by approximately 65% for *TestSQLInjection*, 100% for *TestRemoteFileInclude* and 45% for *TestPathTraversal* when compared to the MEDIUM settings in test case 2(a).

5.4.3 Requests To Proxied URLs

The number of requests to the 10 proxied URLs for attack strength = HIGH is noted in this sub-section. The observation should focus to see if the same URLs as in test case 2(a) receive a higher number of input injections or not. This can be analyzed by looking at the graph below:

The SORTIDs {2,3,8} which are the same POST URLs as in test case 2(a) this time received a higher number of input injection strings. The increase was of the following magnitude:

- (2) http://t09-09.oslo.osa:8080/bodgeit/login.jsp , increase by **46%**

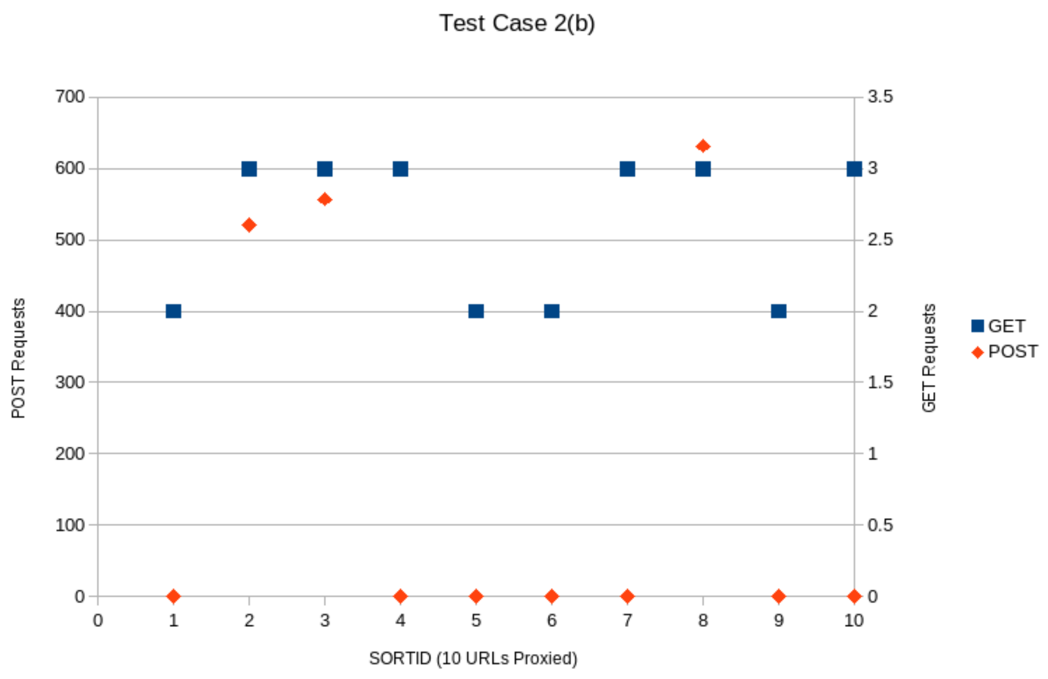


Figure 5.9: Distribution Of GET And POST Requests, Test Case 2(b)

- (3) http://t09-09.oslo.osa:8080/bodgeit/contact.jsp , increase by **36%**
- (8) http://t09-09.oslo.osa:8080/bodgeit/register.jsp , increase by **46%**

The GET URLs got an average of approximately around 2 fuzzed requests which is a slight drop from 4 which we observed for test case 2(a).

5.4.4 Fuzzing HTTP Headers And FORM Payload Parameters

The two HTTP headers that had the most number of unique string inputs where *referrer* and *user-agent* for test case 2(a). With a stronger attack strength = HIGH we observe the following from the data compiled:

- referrer , increase in unique strings by **54%**
- user-agent , increase in unique strings by **43%**

There was also an increase in the overall average number of unique strings to the form parameters by **46%**. In test case 2(a) the average was hovering around 60 but in test case 2(b) the average is approximately 89. The distribution is uniform for the form parameters just like in test case 2(a). The graph below depicts the picture.

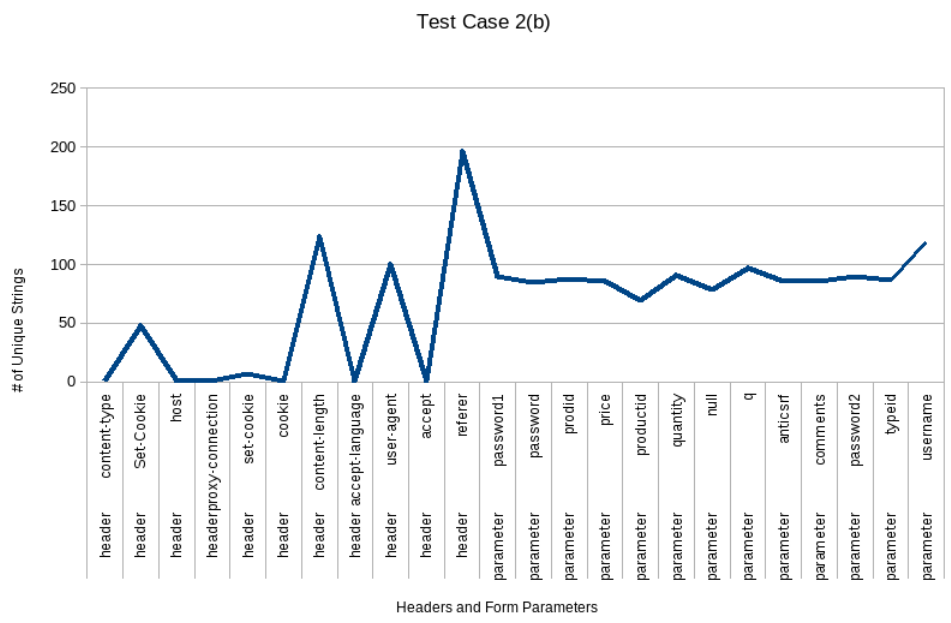


Figure 5.10: # Of Unique HTTP Header And FORM Parameter Strings, Test Case 2(b)

5.5 Test Case: 2(c)

ZAP running in INSANE mode detected the same number of high risk alerts = 4 as the build run with HIGH. The reported scan results were the same so no impact with running at the highest ATTACK STRENGTH. The total ZAP plugin execution time was **43.721** seconds which clearly states that ZAP pounded the staging Webserver with significantly more requests as compared to test cases 2(a) and 2(b). The total number of fuzzed requests to all proxied URLs went up by almost **200%** comparing to test case 2(b) only. Same is the case for the HTTP headers and FORM parameters. The **INSANE** setting did not make any impact in discovering more high risk or medium risk alerts. ZAPs documentation recommends not to run the Active Scan in INSANE mode but because we are testing in a controlled environment hitting non-production webservers we could afford to conduct this test. Based on the results it seems like running ZAP with ATTACK STRENGTH = HIGH is optimal to catch the trivial security flaws in web applications.

5.6 Return On Investment (ROI)

The total number of URLs or test cases used for the ROI calculations were = 10. The developer/tester (just one) was assumed to be paid $50/hour and the assumption we only need 1 server if manual testing is desired versus 5 if the setup needs to be automated. There is no software cost associated here because all the tools and software components are open source. The training cost is the same for both manual and automated. The developer time to implement the automation was set to 80 hours and if the 10 URLs are tested manually the first time it takes 8 hours. Subsequent manual testing of the 10 URLs is 4 hours. A *Build* was defined as a code commit operation. With these variables, equations and assumptions the ROI ratio calculations were performed and the results are graphed below:

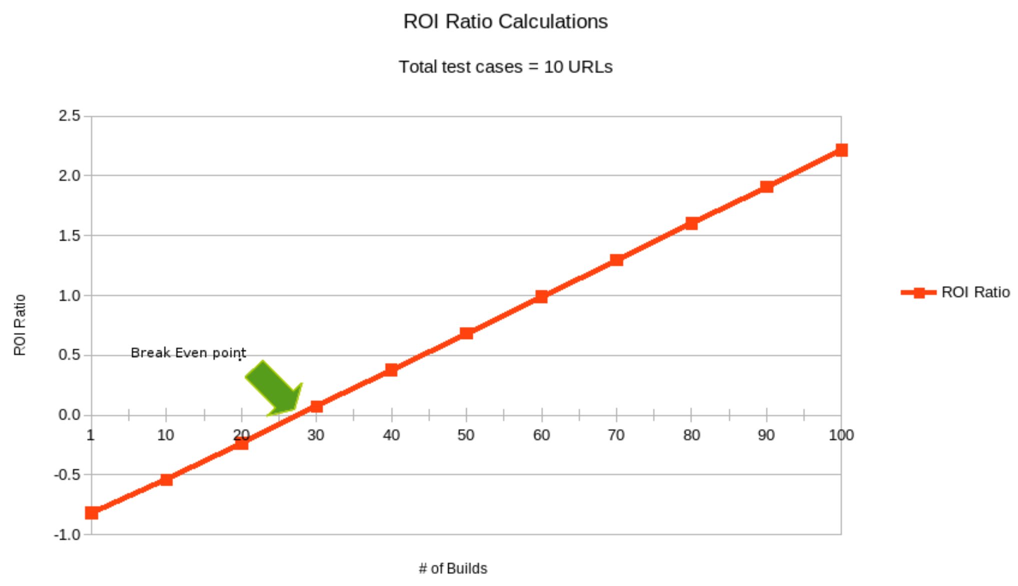


Figure 5.11: Return On Investment

The red line cuts the X-axis between Build = 27 and Build = 28. This is called the *Break Even Point* from where onwards automation starts to produce investment gains. Initially at Build = 1 automation has a high setup cost and manual testing is more cost effective. As more code commits are performed which translate into more Builds we see the gains start to go into the positive. So the return on investment increases in the long run by implementing automation.

Chapter 6

Discussion

This chapter will discuss what was explained and demonstrated in this paper pertaining automated security regression testing of web applications.The observations that were made from the different sections will be highlighted and how the automation process achieved its goals using the various open source tools that were used in the setup. Future work will also be discussed.

6.1 Setup

There are many commercial solutions that work out of the box but getting them to function synchronously with company proprietary components is not trivial. The process and framework that was demonstrated in this paper using Open Source Tools shows how a setup can be constructed that catches security flaws, provides automation and fits into the Software Development Life-Cycle (SDLC) of an organization. Web application security testing should be part of the software development process so that security bugs can be caught and addressed early on. In order to achieve this objective this paper integrates the organizations source code management system (GIT) and uses continuous integration server (Jenkins) in the setup. All code commits automatically run through a security regression apparatus where MySQL stores the test cases and ZAP (Zed Attack Proxy) performs the penetration testing. All the wheels work together in order to provide a process and framework for web application security regression testing.

The Java program running on the Jenkins server performed the task of command and control. Initiated by a shell script on the Jenkins server because of a code commit, it meticulously coordinated all tasks. This task tracker Java application retrieved test URLs from the MySQL Database, opened socket connections to ZAP daemon and started the headless HTMLUnit driver to use ZAP as a proxy. The Selenium component facilitated automatic web form GET and POST submissions and the final task of retrieving scan results to save to the local filesystem. Many of ZAPs internal intricacies were dealt with by this Java program using ZAP Java API calls. This small but complex piece of the framework is critical to understanding of how this framework functions.

The ZAP testing tool plays a fundamental role in the security testing process. It is the central piece of intelligence that detects, discovers and reports security bugs found in the web application being tested. Communicating with ZAP running as a daemon on a remote machine via its Java API enabled the framework to run security regression test builds in an automated manner. Without the Java API from ZAP the automation would have been non-existent.

The rest of the components, Jenkins Server, GIT Server, and MySQL DB, also played important roles for the automated setup to work and function as intended. Without any of these the framework in this paper would be incomplete. One of the advantages of this setup was that all the tools and software used was Open Source with negligible cost. The only associated cost was the time spent learning and executing in order to demonstrate the process and framework for web application security regression testing.

This framework is flexible enough to replace individual components with others. For example, GIT can be replaced by SVN, PostgreSQL can be used instead of MySQL or even ZAP can be replaced with a commercial web application scanner. The framework will function the same way if you interchanged some or all of the components. Open source is not a strict requirement.

6.2 Enhancements

The three components, GIT server, Jenkins Continuous Integration Server and MySQL DB contributed towards making the setup more robust and provided features that were previously not available under the OWASP Security Regression Testing project. This paper can benefit the OWASP project. The GIT server provides a mechanism where code changes are used to trigger the builds. The MySQL Database allows to store test cases for various projects that can easily be tweaked on the fly, turned ON or OFF and provide the Selenium library meta data on how to seek HTML elements, fill them with appropriate information before submission to the staging Webserver. The Jenkins server provides a useful web interface to create new projects, setup automated build environments and open up more possibilities for additional security testing tools that can be incorporated into the framework other than ZAP. This is a major advantage and will be discussed in more detail in the section *Future Work*. The Jenkins Server also provides a web interface where both developers and quality assurance engineers can see the status of the most current builds and also what is archived. The code changes that introduced security bugs can also be easily perused for detailed information. These capabilities as enhancements have added value to the process and security testing framework constructed in this paper.

6.3 Catching Vulnerabilities

No matter how well an automated security framework is designed but if it cannot catch trivial web application security flaws then it fails at its primary purpose. This paper demonstrated how proxying a subset (total 10) of URLs for the Bodgeit Store Vulnerable Web Application some basic XSS, SQL Injection and tampering with HTML inputs can be discovered and reported via automation. Only the base set of plugins that come with the ZAP installation were used excluding any updates for new attack vectors. With just these base set of plugins and ZAP settings (ATTACK levels: LOW, MEDIUM, HIGH, INSANE) vulnerabilities were discovered and reported. Two high risk alerts were caught with ATTACK LEVEL = MEDIUM and a total of 4 with ATTACK LEVEL = HIGH. Metrics were also collected on ZAP to understand how the input injection attacks are distributed across the 10 urls (test cases), plugin execution times and which HTTP headers and form parameters were fuzzed in order to detect security vulnerabilities in the target Bodgeit Store web application. These metrics become important information for devising new set of test cases and provide knowledge on how injection strings work with ZAP. In addition to the detection and discovery of security vulnerabilities in this paper by ZAP, the reporting provided content explaining the security flaw discovered and mitigation steps. This is a bonus to the novice tester and helps to create overall awareness about security in web applications.

6.4 Benefits

This paper not only demonstrated the automation process and framework but also explained the cost benefit in the long run. Manual testing is needed for web application security testing but once the test cases have been devised manually they should be automated to save time and money. The ROI calculations clearly showed that over the long run the gains from automation outweigh a fully functional manual process. The initial investment in automation compared to manual is higher in magnitude but becomes an asset over a longer time period when more builds are executed.

6.5 Future Work

The Security Regression Testing Framework For Web Application Development proposed in this paper can be improved and enhanced further. ZAP provides the functionality to update with the latest scan rule sets and one can add new plugins for security testing. Plugins can be custom made to suit individual requirements and incorporated to be used by ZAP's Active and Passive scanners.

One can also craft user-defined scripts that will define new input vectors i.e. the elements of a request that ZAP will attack. This is a powerful feature because as a tester/developer of your web application you can test with a specific set of attack vectors. This customization option takes web application security testing from a generic set of test inputs to a more specific set. It also reduces the overhead of attack vectors that take up execution time which do not effect the web application being tested.

ZAP version 2.3.x allows users to tweak individual scanner rules and define scan policies for different test conditions. There is also more support for the ZAP API and core functionality has been moved into add-ons which allows to deliver updates dynamically to ZAP [16].

The framework in this paper can be extended to include other tools that can be used for web application security testing. ZAP is only a component that works independently and communicates with the Java application running on the Jenkins server. One can replace ZAP with a commercial scanner and use its API to operate it from the Java program. If an API exists for a better scanner it can be used in this framework. One could potentially even add an additional component and have it perform network security scans.

Chapter 7

Conclusion

The problem statement for this paper stated that web application security is a process that needs to be part of the SDLC. In order to explain this process a framework using Open Source tools was used to demonstrate how automated security regression testing can be performed on web applications. The research that was conducted in this paper addressed the four questions which were part of the problem statement. The answers to the four questions are stated below:

1. Software development is implemented in stages which is composed of a design, implement and test phase with *n* number of iterations. By automating web application security practices into each of these phases the final product that comes out will have addressed the basic security flaws. The sooner you catch the security bugs, the better the quality of the final product in terms of both security and cost. To catch security bugs in the early stages of software development one needs to ensure that security testing is considered during each of the SDLC phases. A final product with security flaws not only damages the brand name but it also becomes acutely expensive to fix the security flaws after deployment. SDLC of web applications is a process and security testing should play an adequate role so it can add value to the quality of the final product or service.

2. By automating the manual tests, you reduce both human error and expense. There are thousands of web application attack vectors and growing as cybercrime becomes more prevalent. Testing each attack vector manually will exhaust the resource and consume too much time. Consider submitting 1000 attack strings that ZAP used in its fuzzing to test one URL manually. This approach is neither practical nor cost effective. In addition to time savings, automation also provides a mechanism to test with a broad range of injection inputs. This way one can test for different security vulnerabilities within web applications.

3. This paper successfully demonstrated how GIT, Jenkins, Tomcat7, MySQL, Java Programming Language and ZAP can be used to achieve an automated

web application security regression testing setup. All the tools used in the framework are Open Source. The security testing process is functional, automated and also extendible. In the future a better open source web application scanner than ZAP can show up but integrating it into this framework is possible given it has an API for communication and it can be deployed as a daemon.

4. What are the benefits of this process? First and foremost the framework and process demonstrated in this paper facilitates to replace manual security testing. It provides a mechanism through which the quality of the web applications are improved by continuously running regression security tests. The framework makes it easier to add more test cases and cover different classes of vulnerabilities. The plugins in ZAP conduct various kinds of attacks like fuzzing HTTP headers, form input injections, reflected content analysis, SQL Injection and more. The return on investment ratios calculated in this paper clearly support the fact that despite a higher initial investment and implementation cost, automation is the winner in the long run.

Bibliography

[1] Opera Software ASA. Approval letter from opera software asa. *Please see Appendix A*, 2014.

[2] Thomas E.; T. A. Thayer Bell. Software requirements: Are they really a problem? *Proceedings of the 2nd international conference on Software engineering. IEEE Computer Society Press*, pages 60–67, 1976.

[3] Project Lead: Simon Bennetts. Owasp - zap, security regression testing. *http://code.google.com/p/zaproxy/wiki/SecRegTests*, 2014.

[4] Project Lead: Simon Bennetts. Owasp - zap, zed attach proxy, the open source penetration testing tool. *https://code.google.com/p/zaproxy/wiki/Introduction*, 2014.

[5] Simon Bennetts. Official blog for the owasp zed attack proxy project. *http://zaproxy.blogspot.no*, 2014.

[6] Scott Chacon. Pro git. *http://git-scm.com/book*, 2014.

[7] Shay Chen. Price and feature comparison of web application scanners. *http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html*, 2014.

[8] The MITRE Corporation. The mitre corporation is a not-for-profit company that operates multiple federally funded research and development centers in the us. *http://www.mitre.org*, 2014.

[9] Gerald P. Hancke Deep Vardhan Bhatt, Stephen Schulze. Secure internet access to gateway using secure socket layer. *Instrumentation and Measurement, IEEE Transactions on (Volume:55 , Issue: 3)*, pages 793–800, 2006.

[10] Macario Polo ; Pedro Reales ; Mario Piattini ; Christof Ebert. Test automation. *IEEE Software, Jan.-Feb. 2013, Vol.30(1)*, pages 84–89, 2013.

[11] R. Everett, G. ; McLeod. Software testing: Testing across the entire software development life cycle. pages 1–28, 2007.

[12] Inc Free Software Foundation. Gnu general public license. *http://www.gnu.org/copyleft/gpl.html , Version 3, 29 June 2007*, 2007.

[13] R. Frolund, S. ; Guerraoui. e-transactions: End-to-end reliability for three-tier architectures. *Software Engineering, IEEE Transactions on (Volume:28 , Issue: 4)*, pages 378–395, 2002.

[14] Dieter Gollmann. Securing web applications. *Information security technical report - Elsevier Ltd*, pages 1363–4127, 2008.

[15] Thomas Hannagan ; Maria Ktori ; Myriam Chanceaux ; Jonathan Grainger. Deciphering captchas: What a turing test reveals about human cognition. *PLoS ONE volume:7 issue:3*, pages 1–4, 2012.

[16] Samantha Groves. Owasp zap 2.3.0. *http://owasp.blogspot.no/2014/04/owasp-zap-230.html*, 2014.

[17] White Paper: Imperva. Consumer password worst practices, the imperva application defense center (adc). *http://www.imperva.com/docs/WP*$_{C}onsumer_{P}assword_{W}orst_{P}ractices.pdf$, 2009.

[18] SANS Institute. The most trusted source for computer security training, certification and research. *http://www.sans.org*, 2014.

[19] Seifedine Kadry. A new proposed technique to improve software regression testing cost. *International Journal of Security and Its Applications Vol. 5 No. 3*, 2011.

[20] R.Tyler Croy ; Andrew Bayer ; Kohsuke Kawaguchi. Jenkins continous intergration server. *http://jenkins-ci.org/*, 2014.

[21] I. Kim. Keypad against brute force attacks on smartphones. *Information Security, IET (Volume:6 , Issue: 2)*, pages 71–76, 2012.

[22] Theodoor Scholtea ; Davide Balzarottib ; Engin Kirdac. Have things changed now? an empirical study on input validation vulnerabilities in web applications. *Computers and Security - Elsevier Ltd*, pages 344–356, 2012.

[23] Xiaogang Wang ; Junzhou Luo ; Ming Yang ; Zhen Ling. A potential http-based application-level attack against tor. *Future Generation Computer Systems - Elsevier Ltd*, pages 67–77, 2010.

[24] Rapid7 LLC. Penetration testing software. *https://www.metasploit.com/*, 2014.

[25] G. Q. Huang ; K. L. Mak. Issues in the development and implementation of web applications for product design and manufacture. *International journal of computer integrated manufacturing*, pages 125–135, 2001.

[26] N. Marback, A. ; Hyunsook Do ; Ehresmann. An effective regression testing approach for php web applications. *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 221–230, 2012.

[27] J D Meier. Web application security engineering. *Security and Privacy, IEEE Volume:4 , Issue: 4*, pages 16–24, 2006.

[28] Paul Midian. Perspectives on penetration testing , black box vs white box. *Network Security, Volume 2002, Issue 11*, pages 10–12, 2002.

[29] Steve Moyle. The blackhat's toolbox: Sql injections. *Network security, vol:2007 iss:11*, pages 12–14, 2007.

[30] Paul Mutton. Netcraft internet services company. *http://news.netcraft.com/archives/2014/04/08/half-a-million-widely-trusted-websites-vulnerable-to-heartbleed-bug.html*, 2014.

[31] Bhavesh Naik. Content spoofing. *http://resources.infosecinstitute.com/content-spoofing/*, 2014.

[32] Louis Nyffenegger. Evolution of cross site request forgery attacks. *Journal in Computer Virology, Volume 4, Issue 1*, pages 1772–9904, 2008.

[33] NJ Ouchn. 2013 top security tools as voted by toolswatch.org readers. *Toolswatch Hackers Arsenal , http://www.toolswatch.org/2013/12/2013-top-security-tools-as-voted-by-toolswatch-org-readers/*, 2013.

[34] R. ; Haeng-Kon Kim Patel, J. ; Lee. Architectural view in software development life-cycle practices. *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on*, pages 194–199, 2007.

[35] Open Source Project. Bodgeit - the bodgeit store is a vulnerable web application suitable for pen testing. *https://code.google.com/p/bodgeit/*, 2012.

[36] Open Web Application Security Project. The free and open software security community. *http://www.owasp.org*, 2014.

[37] Ariel Ortiz Ramirez. Three-tier architecture. *Linux Journal, (Issue: 75)*, 2000.

[38] Stefan Munch ; Peter Brandstetter ; Konstantin Clevermann ; Oliver Kieckhoefel ; Ernst Reiner Schafer. Test automation roi (return on investment. *PHARMACEUTICAL ENGINEERING, Vol 32, No4*, pages 1–8, 2012.

[39] sectools.org. Network security tools. *http://sectools.org*, 2014.

[40] Tenable Network Security. Identify vulnerabilities, reduce risk and ensure compliance. *http://www.tenable.com/*, 2014.

[41] WhiteHat Security. Website security statistics report - may 2013. *http://info.whitehatsec.com/2013-website-security-report.html*, 2013.

[42] The Internet Society. Global internet report 2014. *http://www.internetsociety.org/sites/default/files/Global_Internet_Report_2014_0.pdf*, 2014.

[43] James Joshi ; Walid Aref ; Arif Ghafoor ; Eugene Spafford. Security models for web-based applications. *Communications of the ACM, Feb2001, Vol:44, No2*, pages 38–44, 2001.

[44] Marco Vieira. Defending against web application vulnerabilities. *Computer, (Volume:45 , Issue: 2) , Published by the IEEE Computer Society*, pages 66–72, 2012.

[45] Markku Laine ; Denis Shestakov ; Evgenia Litvinova ; Petri Vuorimaa. Toward united web application development. *IEEE, IT Professional (Volume:13 , Issue: 5)*, pages 30–36, 2011.

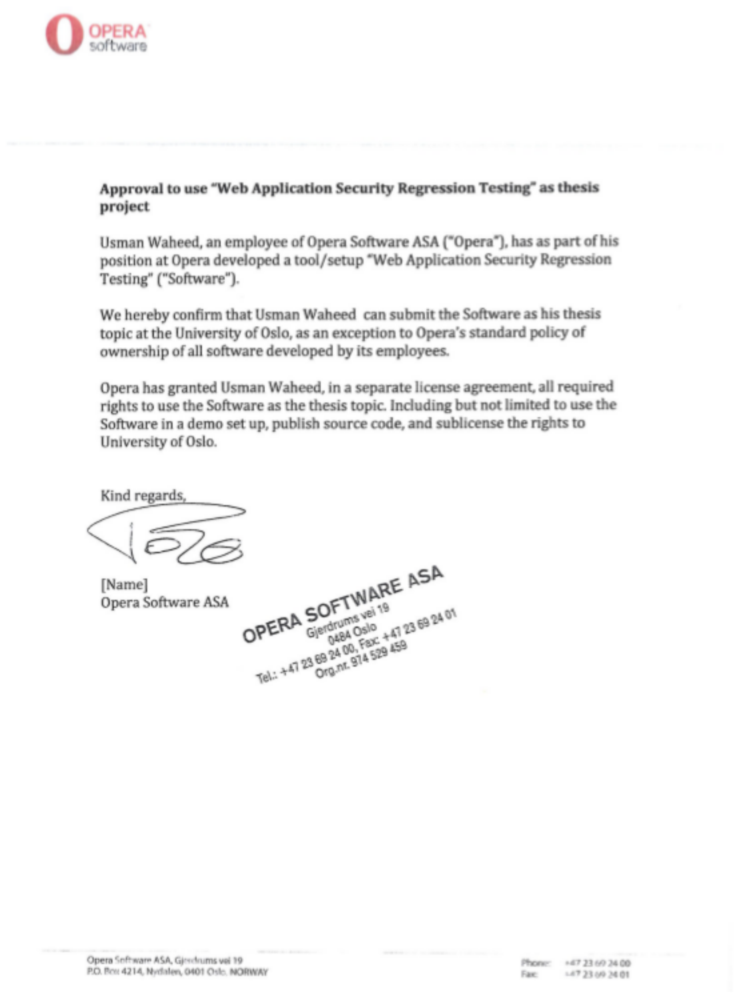
[46] Paco Hope ; Ben Walther. Web security testing cookbook, systematic techniques to find problems fast. *Published by O'Reilly Media Inc - ISBN: 978-0-596-51483-9*, pages 1–269, 2009.

[47] Bill Wong. What heartbleed should teach embedded developers. *Electronic Design. Jun2014, Vol. 62 Issue 6*, pages 64–64, 2014.

[48] Stephen Chong ; Jed Liu ; Andrew C Myers ; Xin Qi ; K Vikram ; Lantian Zheng ; Xin Zheng. Building secure web applications with automatic partitioning. *Communications of the ACM , Volume 52 Issue 2*, pages 79–87, 2009.

Appendix A

Approval Letter From Opera Software ASA



OPERA software

Approval to use "Web Application Security Regression Testing" as thesis project

Usman Waheed, an employee of Opera Software ASA ("Opera"), has as part of his position at Opera developed a tool/setup "Web Application Security Regression Testing" ("Software").

We hereby confirm that Usman Waheed can submit the Software as his thesis topic at the University of Oslo, as an exception to Opera's standard policy of ownership of all software developed by its employees.

Opera has granted Usman Waheed, in a separate license agreement, all required rights to use the Software as the thesis topic. Including but not limited to use the Software in a demo set up, publish source code, and sublicense the rights to University of Oslo.

Kind regards,

[illegible]

[Name]
Opera Software ASA

OPERA SOFTWARE ASA
Gjerdrums vei 19
0484 Oslo
Tel.: +47 23 69 24 00, Fax: +47 23 69 24 01
Org.nr. 974 529 459

Opera Software ASA, Gjerdrums vei 19
P.O. Box 4214, Nydalen, 0401 Oslo, NORWAY

Phone: +47 23 69 24 00
Fax: +47 23 69 24 01

Appendix B

Java Code

Run_SRT.java

```java
class Run_SRT
{

    public static void main(String[] args) throws Exception {


        // Results directory
        String path_to_files = "/var/www/srt/";

        // Read in 4 command line arguments
        String project_name = args[0];  // PROJECT_NAME
        String projectid = args[1];      // PROJECT_ID
        String baseURL = args[2];        // BASEURL
        String Attack_str = args[3];     // ATTACK STRENGTH
        String context = project_name;
        int project_id = Integer.parseInt(projectid);

        ZapperReport auth = new ZapperReport();
        auth.StartSetup(project_name,baseURL,context);
        auth.ProxyURLS(project_id,Attack_str);
        auth.StopSetup();
        auth.RunAScan(baseURL,Attack_str);
        // auth.printAlerts(baseURL,0,0); FOR DEBUGGING
        auth.saveAlertstoFile(baseURL,0,0,path_to_files);

    }

}
```

ZapperReport.java

```java
import java.sql.*;
import org.openqa.selenium.Proxy;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.CapabilityType;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.By;
import org.zaproxy.clientapi.gen.Core;
import org.zaproxy.clientapi.core.ClientApi;
import org.zaproxy.clientapi.core.ApiResponse;
import org.zaproxy.clientapi.core.Alert;
import org.zaproxy.clientapi.gen.Ascan;
import org.zaproxy.clientapi.gen.Spider;
import org.zaproxy.clientapi.gen.Context;
import org.apache.commons.lang.time.FastDateFormat;
import java.util.Properties;
import java.util.List;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.Writer;

public class ZapperReport {

    WebDriver driver;
    Connection connection = null;
    Statement statement = null;
    ResultSet resultSet = null;
    String Driver = "com.mysql.jdbc.Driver";
    String JDBC_URL = "jdbc:mysql://ahs-db1.ams.osa:3306/SRT";

    String ZAP_HOST = "owasp-t01.oslo.osa";
    int ZAP_PORT = 8080;
    String CONTEXT = "true";
    String RECURSE = "true";
    String INSCOPEONLY = "false";
    String ATTACK_STR = "LOW";     // If not set by user, default
    String ALERT_THRESHOLD = "MEDIUM"; // default
    String homeDirectory = "/home/zap/session_logs/";
    String project = null;
    String session_name = null;
    Properties props = System.getProperties();

    ClientApi clientApi;
    Core core;
    Context context;

    public ZapperReport() {

        //Load driver
        try {
                Class.forName(Driver);
            }
            catch (ClassNotFoundException e) {
                e.printStackTrace();
            }

    }

    public void ProxyURLS(int project_number, String attack_str) {

        String sort_order,url,method,params,submit;

        try {

            // Class.forName("com.mysql.jdbc.Driver");
             connection = DriverManager.getConnection(JDBC_URL, "db_read", "5r1r3aD");

            PreparedStatement statement = connection.prepareStatement("select sort_order, \
                url,method,params,submit from urls where active=1 and project_id = ? order by sort_order");
            statement.setInt(1,project_number);
            resultSet = statement.executeQuery();

            System.out.println();
            System.out.println("ATTACK STRENGTH: " + attack_str);
            System.out.println("ALERT THRESHOLD: " + ALERT_THRESHOLD);
            System.out.println();
            System.out.println("Start URL's proxy through ZAP daemon (owasp-t01.oslo.osa)");

            while (resultSet.next()) {

                // System.out.println("Record: " + resultSet.getString("url,sort_order"));
                System.out.print(resultSet.getString(1));
                System.out.print(", ");
                System.out.print(resultSet.getString(2));
                System.out.print(", ");
                System.out.print(resultSet.getString(3));
                System.out.print(", ");
                System.out.print(resultSet.getString(4));
                System.out.print(", ");
                System.out.print(resultSet.getString(5));
                System.out.print(", ");
                System.out.print("\n"); //new line

                sort_order = resultSet.getString(1);
                url = resultSet.getString(2);
                method = resultSet.getString(3);
                params = resultSet.getString(4);
                submit = resultSet.getString(5);

                proxyURL(sort_order,url,method,params,submit);

            }

            System.out.println("End URL's proxy");
            System.out.println();


        } catch (SQLException e) {
            e.printStackTrace();
            System.out.println("DB Query execution failed");
        }
        finally {

          try {
            if (resultSet != null)
               resultSet.close();
            if (statement != null)
               statement.close();
            if (connection != null)
               connection.close();
          }
          catch (SQLException e) {
            e.printStackTrace();
          }
        }

    }

    public void StartSetup(String project_name, String baseURL, String context_name) throws Exception {

         project = project_name;

             String datetimestamp = FastDateFormat.getInstance \
                ("yyyy-MM-dd-HH-mm-ss").format(System.currentTimeMillis());
             //session_name = project + "-srt_"+datetimestamp;
             session_name = project + "-srt_"+datetimestamp;

         // Setup the proxy details
         Proxy proxy = new Proxy();

         String proxy_settings = ZAP_HOST+":"+ZAP_PORT;
         proxy.setHttpProxy(proxy_settings);
         proxy.setSslProxy(proxy_settings);

         DesiredCapabilities capabilities = DesiredCapabilities.htmlUnit();
         capabilities.setJavascriptEnabled(false);
         capabilities.setCapability(CapabilityType.PROXY, proxy);
         WebDriver driver = new HtmlUnitDriver(capabilities);
         this.setDriver(driver);

         // Client API object with host and port
         clientApi = new ClientApi(ZAP_HOST, ZAP_PORT);

         // Core API object for session
         core = new Core(clientApi);
         core.setHomeDirectory(homeDirectory);
         core.newSession(session_name,"override");

         // Context object
         context = new Context(clientApi);
         context.newContext(context_name);

         clientApi.addIncludeInContext(context_name,baseURL);

        // Set the User Agent such that we know where we are coming from.
        // Keep in mind that ZAP also fuzzes this field
        props.setProperty("http.agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;) SRTv2.2");

    }

    public void StopSetup() {
            driver.close();
    }

    public void sleep() {
            try {
                    Thread.sleep(3000);
            } catch (InterruptedException e) {
                    // Ignore
            }

    }

    protected void setDriver(WebDriver driver) {
            this.driver = driver;
    }


    public void proxyURL(String sort_order, String url, String method, String params, String submit) {


            if (method.equals("GET")) {

                driver.get(url);

            } else {

                driver.get(url);

                WebElement link;

                String[] keypairs = params.split("&");
                for (String keypair: keypairs) {

                    String[] paramvalue = keypair.split("=");
                    String param = paramvalue[0];
                    String value = paramvalue[1];

                    if ( param.equals("empty")) {

                        // This field does not need any inputs from Selenium, use its default settings.
                      System.out.println("Using default settings for field: "+param);

                    } else {

                      link = driver.findElement(By.name(param));
                        link.sendKeys(value);
                    }


                }


                if ( submit.contains("button")) {
                     link = driver.findElement(By.xpath(submit));
                    link.click();
                } else {
                    link = driver.findElement(By.id(submit));
                    link.click();
                }

            }


    }


    public void RunAScan(String baseURL, String attackstr) throws Exception {

            String [] resp_status;
            String status = null;



            final Ascan ascan_url = new Ascan(clientApi);
            ascan_url.setOptionAttackStrength(attackstr);



            ascan_url.scan(baseURL,RECURSE,INSCOPEONLY);


            // Loop through till Active Scanner is done
            resp_status = ascan_url.status().toString(0).split("=");
            status = resp_status[1];
            status = status.replaceAll("(\\r|\\n|\\s)","");
            System.out.println();
            System.out.println("Initiating Active Scan on URL's proxied ...");
            while ( ! status.equals("100") ) {

                resp_status = ascan_url.status().toString(0).split("=");
                status = resp_status[1];
                status = status.replaceAll("(\\r|\\n|\\s)","");
                System.out.println("Active Scan Status: " + status + "%");
                Thread.sleep(2000);

            }
            System.out.println("Active Scan End ...");
            System.out.println();


    }

    public void saveAlertstoFile(String baseURL, int start, int count, String path_to_files) \
      throws Exception {

        String session_file = session_name + ".txt";
        String path_to_output_file = path_to_files + project + "/" + session_file;
        String file_uri = "http://ahs-m1.ams.osa:80/srt/" + project + "/" + session_file;
        System.out.println();
        System.out.println("Extracting alerts and saving results to file: " + file_uri);

        try (Writer writer = \
        new BufferedWriter(new OutputStreamWriter(new FileOutputStream(path_to_output_file), \
        "utf-8"))) {

            // Extract Alerts
            Alert alert;
            Alert.Risk risk_obj;
            Alert.Reliability rel_obj;
            int total_alerts=0,high_alerts=0,medium_alerts=0,low_alerts=0,informational_alerts=0;
            String url,risk,reliability,param,other,attack,desc,ref,solution;
            List<Alert> Alerts = clientApi.getAlerts(baseURL, start, count);

            writer.write("Project: " + project + "\n");
            writer.write("Filename: " + session_name + "\n");
            writer.write("Risks: High, Medium, Low and Informational\n\n");

            for (int i=0; i<Alerts.size(); i++) {

                total_alerts++;

                alert = Alerts.get(i);
                risk = alert.getRisk().toString();

                switch (risk) {
                    case "High":    high_alerts++;
                        break;
                    case "Medium":  medium_alerts++;
                        break;
                    case "Low":     low_alerts++;
                        break;
                    case "Informational":
                        informational_alerts++;
                        break;
                }

                // Process all levels of risks
                if ( risk.equals("High") || risk.equals("Medium") || \
                     risk.equals("Low") || risk.equals("Informational") ) {

                    // Extract rest of the elements
                    url = alert.getUrl();
                    rel_obj = alert.getReliability();
                    param = alert.getParam();
                    other = alert.getOther();
                    attack = alert.getAttack();
                    desc = alert.getDescription();
                    ref = alert.getReference();
                    solution = alert.getSolution();

                    // Output to file the alert extracted
                    writer.write("URL: " + url + "\n");
                    writer.write("URL Params (if any): " + param + "\n");
                    writer.write("Risk: " + risk + "\n");
                    writer.write("Reliability: " + rel_obj + "\n");
                    writer.write("Description: " + desc + "\n");
                    writer.write("Reference: " + ref + "\n");
                    writer.write("Solution: " + solution + "\n");
                    writer.write("---------------------------------------- \
                    ----------------------------------------------------------------------\n");

                }
            }

            // Output to standard out
            System.out.println("Alerts Extraction complete ...");
            System.out.println();

            // Summary of alerts
            System.out.println("Summary of Alerts");
            System.out.println("High Risk Alerts: " + high_alerts);
            System.out.println("Medium Risk Alerts: " + medium_alerts);
            System.out.println("Low Risk Alerts: " + low_alerts);
            System.out.println("Informational Risk Alerts: " + informational_alerts);
            System.out.println("Total: " + total_alerts);

            // Output to file
            writer.write("\n");
```

```
                writer.write("Summary of Alerts\n");
                writer.write("High Risk Alerts: " + high_alerts + "\n");
                writer.write("Medium Risk Alerts: " + medium_alerts + "\n");
                writer.write("Low Risk Alerts: " + low_alerts + "\n");
                writer.write("Informational Risk Alerts: " + informational_alerts + "\n");
                writer.write("Total: " + total_alerts + "\n");
                writer.write("\n");
                writer.close();

        } catch (IOException ex) {
                // Handle me
        }


    }

}
```

Appendix C

Launcher Shell Script

run_bodgeit.sh

```
#!/bin/sh

# Path to JAVA HOME
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64/

# Ensure libs that are needed are in the CLASSPATH
export CLASSPATH=/home/jenkins/bodgeit/bin: \
        /home/jenkins/lib/ant.jar: \
        /home/jenkins/lib/apache-mime4j-0.6.jar:
        /home/jenkins/lib/bsh-1.3.0.jar:
        /home/jenkins/lib/cglib-nodep-2.1_3.jar:
        /home/jenkins/lib/commons-codec-1.4.jar:
        /home/jenkins/lib/commons-collections-3.2.1.jar:
        /home/jenkins/lib/commons-exec-1.1.jar:
        /home/jenkins/lib/commons-io-2.4.jar:
        /home/jenkins/lib/commons-jxpath-1.3.jar:
        /home/jenkins/lib/commons-lang-2.6.jar:
        /home/jenkins/lib/commons-logging-1.1.1.jar:
        /home/jenkins/lib/cssparser-0.9.5.jar:
        /home/jenkins/lib/guava-12.0.jar:
        /home/jenkins/lib/hamcrest-core-1.1.jar:
        /home/jenkins/lib/hamcrest-library-1.1.jar:
        /home/jenkins/lib/htmlunit-core-js-2.12.jar:
        /home/jenkins/lib/httpclient-4.2.3.jar:
        /home/jenkins/lib/httpcore-4.2.2.jar:
        /home/jenkins/lib/httpmime-4.2.3.jar:
        /home/jenkins/lib/ini4j-0.5.2.jar:
        /home/jenkins/lib/jcommander-1.13.jar:
        /home/jenkins/lib/jna-3.4.0.jar:
        /home/jenkins/lib/jna-platform-3.4.0.jar:
        /home/jenkins/lib/json-20080701.jar:
        /home/jenkins/lib/junit-dep-4.10.jar:
        /home/jenkins/lib/nekohtml-1.9.15.jar:
        /home/jenkins/lib/netty-3.2.7.Final.jar:
        /home/jenkins/lib/operadriver-0.14.jar:
        /home/jenkins/lib/protobuf-java-2.4.1.jar:
        /home/jenkins/lib/sac-1.3.jar:
        /home/jenkins/lib/serializer-2.7.1.jar:
        /home/jenkins/lib/testng-6.0.1-nobsh-noguice.jar:
        /home/jenkins/lib/webbit-0.4.8-SNAPSHOT.jar:
        /home/jenkins/lib/xalan-2.7.1.jar:
        /home/jenkins/lib/xercesImpl-2.9.1.jar:
        /home/jenkins/lib/xml-apis-1.3.04.jar:
        /home/jenkins/lib/zap-api-v2-6.jar:
        /home/jenkins/lib/java-mail-1.4.4.jar:
        /home/jenkins/lib/json-lib-2.4-jdk15.jar:
        /home/jenkins/lib/zap.jar:
        /home/jenkins/lib/selenium-server-standalone-2.35.0.jar:
        /home/jenkins/lib/mysql-connector-java-5.1.18-bin.jar:
        /home/jenkins/lib/htmlunit-2.12.jar:
        /home/jenkins/lib/xml-apis-1.4.01.jar:
        /home/jenkins/lib/commons-cli-1.2.jar:
        /home/jenkins/lib/commons-cli-1.2-javadoc.jar:
        /home/jenkins/lib/commons-cli-1.2-sources.jar:.

# Run Build with 4 params (bodgeit 4 http://t09-09.oslo.osa:8080/bodgeit/ LOW)
```

```
/usr/bin/java Run_SRT bodgeit 4 http://t09-09.oslo.osa:8080/bodgeit/ LOW
```

Appendix D

Log Parsing Perl Scripts

zap.pl

```
#!/usr/bin/perl

use strict;
my %data;
my @fields;
my ($test_type,$scan_time,$total_scan_time);

my $path_to_file = $ARGV[0];

open(ZAP,"< $path_to_file") or die "Cannot open file in specified directory\n";

while(<ZAP>) {

       chomp;
       next if ( /^\s|^\t/ );
       if ( /HostProcess \- completed/ && ( /TestPathTraversal/ || /TestRemoteFileInclude/ || \
           /TestRedirect/ || /TestServerSideInclude/ || /TestCrossSiteScriptV2/ ||  \
           /TestSQLInjection/ || /TestDirectoryBrowsing/ || /TestInfoSessionIdURL/ || \
           /TestClientBrowserCache/ || /TestExternalRedirect/ || /TestInjectionCRLF/ || \
           /TestParameterTamper/ || /ScriptsActiveScanner/ ) ) {
               @fields = split(/\s/,$_);
               $test_type = $fields[10];
               $scan_time = $fields[12];
               $scan_time =~ s/s$//g;
               # DEBUG
               # print "$test_type - $scan_time\n";
               $data{$test_type}=$scan_time;
       }

       if ( /Scanner \- scanner completed/ ) {
               @fields = split(/\s+/,$_);
               $total_scan_time = $fields[8];
               $total_scan_time =~ s/s$//g;
               # DEBUG
               # print "$test_type - $scan_time\n";
       }




}
close(ZAP);

foreach my $key (keys %data) {
       print "$key & $data{$key} \\\\ \\hline\n";
}
print "Total,$total_scan_time\n";
```

access.pl

```
#!/usr/bin/perl

use strict;
my %data;
my ($method,$url_string,$url_path);
my @url_fields;

my $path_to_file = $ARGV[0];

my %URLS=(

        1 => ’http://t09-09.oslo.osa:8080/bodgeit/’,
        2 => ’http://t09-09.oslo.osa:8080/bodgeit/login.jsp’,
        3 => ’http://t09-09.oslo.osa:8080/bodgeit/contact.jsp’,
        4 => ’http://t09-09.oslo.osa:8080/bodgeit/about.jsp’,
        5 => ’http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=’,
        6 => ’http://t09-09.oslo.osa:8080/bodgeit/product.jsp?typeid=’,
        7 => ’http://t09-09.oslo.osa:8080/bodgeit/search.jsp’,
        8 => ’http://t09-09.oslo.osa:8080/bodgeit/register.jsp’,
        9 => ’http://t09-09.oslo.osa:8080/bodgeit/product.jsp?prodid=’,
       10 => ’http://t09-09.oslo.osa:8080/bodgeit/logout.jsp’,

);



my %result;

open(ZAP,"< $path_to_file") or die "Cannot open file in specified directory\n";

while(<ZAP>) {

        next if ( /^\s|^\t/ );
        chomp;
        my $record = $_;
        $record =~ /(\[.+?\])\s(\d+\.\d+\.\d+\.\d+) \
        \s(HTTP\/\d+\.\d+)\s(\w+)\s(\".+?\")\s+(.+?)\s(\".+?\")\s(\d+)\s(\d+)/g;
        $method=$4;
        $url_string=$7;
        @url_fields=split(/\s/,$url_string);
        $url_path=$url_fields[1];

        # print "m -> $method\n";
        # print "u -> $url_path\n";
        # $data{$method}{$url_path}+=1;

        foreach my $urlid (keys %URLS) {
                if ( $URLS{$urlid} =~ /$url_path/ ) {
                        $result{$urlid}{$method}+=1;
                }
        }
}
close(ZAP);

foreach my $url (sort { $a <=> $b }  keys %result) {

        print "$URLS{$url} & $result{$url}{’GET’} & $result{$url}{’POST’} \\\\ \\hline\n";

        # foreach my $m ( sort keys %{$result{$url}}) {
        #         print "\t$m = $result{$url}{$m}\n";
        #
        # }
}
```

request.pl

```
#!/usr/bin/perl

use strict;
my $flag=0;
my %data;
my $file_handle;
my ($parameter,$header);
my $path_to_file = $ARGV[0];

open(READ,"< $path_to_file") or die "Cannot open file to read\n";

while(<READ>) {

        chomp;

                my $record = $_;
                $record =~ s/\s+/ /g;

                if ( $record =~ /header\=/ ) {
                        $record =~ /(.+?) header\=(.*)/g;
                        $header = $2;
                        my ($key,$value) = split(/\=/,$header,2);
                        $data{’header’}{$key}{$value}=1;
                }

                if ( $record =~ /parameter\=/ ) {
                        $record =~ /(.+?) parameter\=(.*)/g;
                        $parameter = $2;
                        my ($key,$value) = split(/\=/,$parameter,2);
                        $data{'parameter'}{$key}{$value}=1;
                }


}
close(READ);

foreach my $type ('header','parameter') {

        foreach my $key (keys %{$data{$type}}) {
                # foreach my $value (keys %{$data{$type}{$key}}) {
                #       print "$type:\t$key => $value\n";
                # }

                my $unique_values = scalar keys %{$data{$type}{$key}};
                print "$type & $key & $unique_values \\\\ \\hline\n";


        }

}
```