Samir Kumar Paudel

VULNERABLE WEB APPLICATIONS AND HOW TO AUDIT THEM

Use of OWASP Zed Attack Proxy effectively to find the vulnerabilities of web applications

VULNERABLE WEB APPLICATIONS AND HOW TO AUDIT THEM

Use of OWASP Zed Attack Proxy effectively to find the vulnerabilities of web applications

Samir Kumar Paudel
Bachelor's Thesis
Spring 2016
Degree Program in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Information Technology

Author: Samir Kumar Paudel
Title of the bachelor's thesis: Vulnerable Web Applications and How to Audit Them
Supervisor: Lauri Pirttiaho
Term and year of completion: Spring 2016 Number of pages: 59

This thesis work was done as a private project for completing a Bachelor's Degree in Information Technology. The main objective of this work was to find out the effectiveness of OWASP Zed Attack Proxy, an open source and free integrated penetration testing tool for finding vulnerabilities in web applications. Besides that, the secondary objectives were to learn how to make web applications and try to find out the security loopholes of them.

For this project, Notepad++, Localhost, and OWASP Zed Attack Proxy were used as tools, PHP, HTML, JavaScript, and CSS as languages, and MySQL Database for making a prototype web application. Notepad++ is a text editor and it supports various programming languages for writing programs or edit files. Localhost was used as a web host. And OWASP Zed Attack Proxy was used as a testing tool. The reason for using OWASP ZAP is that it is an open source and free application and it is a very popular tool among all available web application penetration testing tools either commercial or open source.

Some vulnerabilities were successfully found by the application (OWASP Zed Attack Proxy). Besides that, the developed prototype web application is a simple one. To test the effectiveness of OWASP Zed Attack Proxy in more detail, the web application should be more complex with various features. Being a prototype, it has limitations regarding its full intended features. As only few features were implemented in the prototype, there is a possibility to add more features to the web application as well as testing it in the future.

Keywords: PHP, MySQL, JavaScript, HTML, CSS, Web Application, Security

PREFACE

This Bachelor's Thesis was submitted to Oulu University of Applied Sciences, School of Engineering, Kotkantie Campus, as a part of study, necessary to complete a bachelor's degree in IT engineering, during the fall of 2016. The thesis work was carried out as a private project with the main objective to find out the effectiveness of OWASP ZAP, an open source web application security testing tool. Other objectives were to gain some knowledge in the area of web applications development and testing web applications to find out the security loopholes.

I would like to express my sincere gratitude to my thesis supervisor, Mr. Lauri Pirttiaho for his constructive criticisms, patience and support. Big thanks to Mrs. Riitta Rontu, for approving the thesis topic, Mrs. Kaija Posio, for the language checking and the staff and respected teachers of the Oulu University of Applied Sciences, School of Engineering, Raahe Campus, and Kotkantie Campus for their support during my study in Raahe and in Oulu.

I would also like to thank Raahe Municipal Library for providing me a working environment during my thesis work.

Last, but not least, I would to like to thank my family and friends for motivating me throughout this whole process.

Raahe, May 2016
Samir Kumar Paudel

CONTENTS

VOCABULARY

Terms and Abbreviations	Explanation
OWASP	The Open Web Application Security Project
HTML	Hyper Text Markup Language
ZAP	Zed Attack Proxy
PHP	Hypertext Preprocessor
CSS	Cascading Style Sheet
HTTP	Hyper Text Transfer Protocol
SQL	Structured Query Language
XSS	Cross Site Scripting
CRUD	Create, Read, Update, and Delete

1 INTRODUCTION

Over the past few decades, information technology has made a big leap. If we see the scenario of past 10 years, information technology has been gradually making a significant positive impact on human life. The Internet and web applications are one of the most rapidly developed sector of information technology. Because the bandwidth costs are significantly getting lower, and developments of new technologies are continuously growing over the years, Internet services are widely accessible around the globe and so is the use of Internet services. The chart presented below shows that the global Internet users in 1997 were just around 2 percent of human population which has been increased to around 40 percent in 2014.

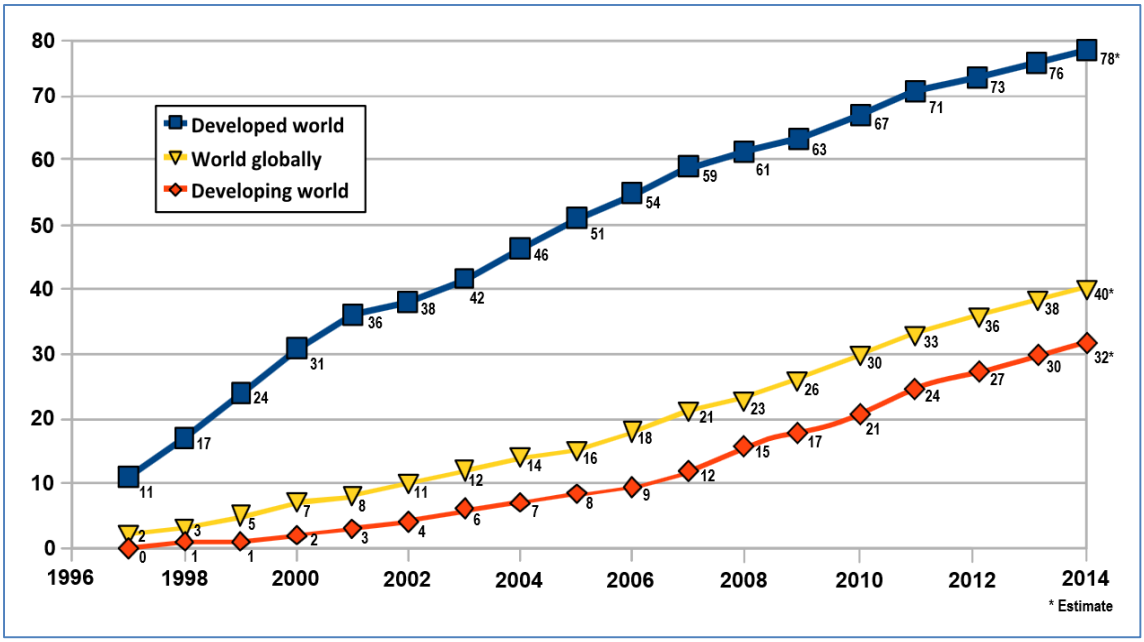


FIGURE 1. Internet users per 100 inhabitants (1)

Nowadays, web applications are playing a significant role towards making a human's life easier. They are making a noble presence in the areas such as education, banking, entertainment, marketing, and communication. Some notable examples are online shopping, online banking, social networking, online document editing, online media editing, online maps services, online dictionaries, online search services, and gaming. These applications provide services to people according to their needs both efficiently and cost effectively. However, only being

efficient and cost effective is not enough. These applications should be secure and reliable too. The usage of unsecure and unreliable applications may always have serious impacts as they can make a company out of business.

Along with the increasing dependency of human life on web services, security issues are becoming more challenging. Over the past decade, along with the development of web technologies, new attacking techniques are also emerging. “Two of the most widely spread and dangerous vulnerabilities in web applications are SQL injection and Cross Site Scripting (XSS), because of the damage they may cause to the victim business” (2). Before attacking, attackers first try to find out vulnerabilities in the application they want to attack, and then use the vulnerabilities they found to perform desired attacks. The usage of a vulnerable application is always dangerous for all stakeholders. So, before making available for practical use, the application should always be tested thoroughly for any kind of possible vulnerabilities.

Many tools are available for testing vulnerabilities in web applications. Some of them are free, and others are commercial. They scan the applications by both automatic and manual ways. The topic of this thesis is also related to finding vulnerabilities in web applications using one of such tools. For the thesis, a sample web application will be developed, a web application penetration testing tool from OWASP, called Zed Attack Proxy, will be used for testing the application to find different kinds of vulnerabilities, and the final results will be summarized to see its effectiveness. HTML, PHP, JavaScript, and MySQL database will be used to develop the application. The application is about a grading system for a school. It can perform some tasks, for example, the admin can register new students and teachers in the database, teachers can insert grades, students can view their grades.

2 DEVELOPMENT OF THE SAMPLE WEB APPLICATION

This chapter is a summary of brief introductions of the tools and techniques that were used to carry out the thesis. In Section 2.1, an overview of the web application is described. Section 2.2 presents a short description of the web server, language, and scripts that were used to develop the application. Section 2.3 describes the database that was used for the web application. There is a different Chapter for thoroughly describing Zed Attack Proxy.

2.1 Web Application

Acunetix.com defines Web applications as:

> Computer programs allowing website visitors to submit and retrieve data to/from a database over the Internet using their preferred web browser. The data is then presented to the user within their browser as information is generated dynamically (in a specific format, e.g. in HTML using CSS) by the web application through a web server. (3)

According to the definition, web applications have certain characteristics. They are computer programs, which are used to perfom intended tasks. Users of web applications can both send and receive data through Internet browsers. Mostly, the Internet is required to use web applications. Data is dynamically generated in a web format and displayed through the browser. The application accomplishes its tasks through a web server. Web applications cannot run outside a web server.

A web application is a software application which utilizes web and web technologies to perform certain tasks. Generally, it requires the Internet to run and accomplish tasks. But it can also be used on local networks, for example intranet. It works on a client - server model. A server is a web server and a client is a web browser. The user sends HTTP requests through the browser (client) to the web server and the web server accomplishes the requested tasks and sends the required information back to the browser. To allow support by all kinds of browsers, it uses HTML or XHTML as a format to generate documents. A combination of

both the server-side scripts (e.g. PHP, asp.net) and client-side script (e.g. JavaScript, html) are used to develop web applications. Server-side scripts are used to interact with the database server while client-side scripts are used for dynamically presenting of information in the browser.

There are some advantages of web applications over desktop applications. Unlike desktop applications, web applications are universal. They can run on any device or operating system. It is easy and cost effective to maintain and update the web applications. As web applications use internet browsers to interact with users, in most of the cases, no installations are required at the client side. The centralized data and its accessibility over the web from any platform at any time allows users worry-free usage of applications. Users do not need to pay any serious attention to backup and security while using the application.

To understand how web applications work, Acunetix has given a simple example of a web application model, which consists of a client, the Internet, and a server. The client consists of a PC with a browser. All HTTP/S requests and responses pass through the Internet. And the Server side system consists of web servers, a web application server and database servers.

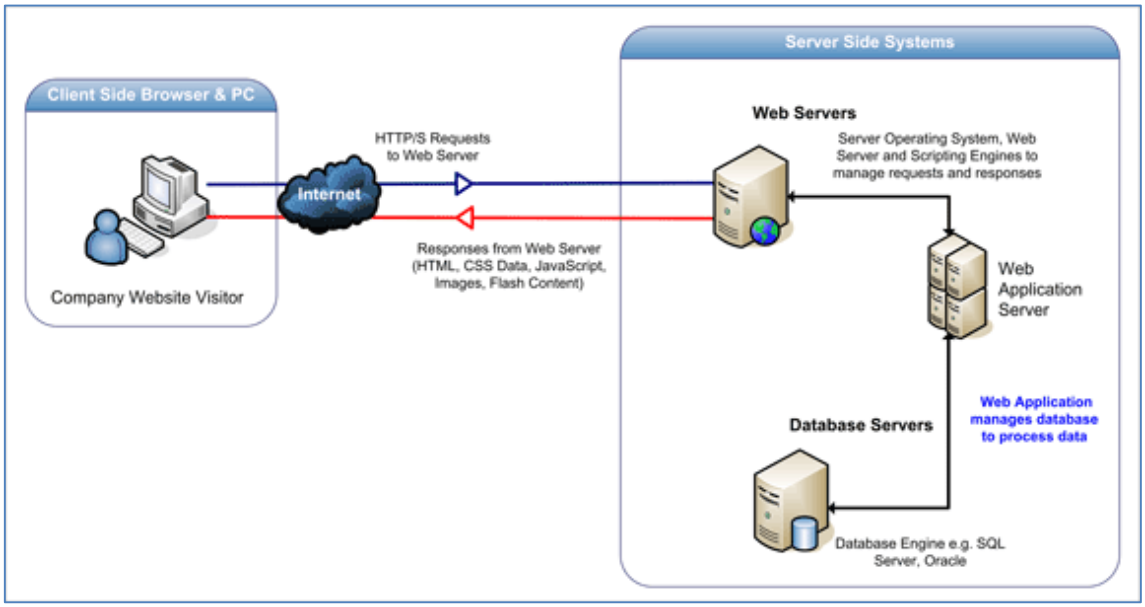


FIGURE 2. Client-Server web application model (3).

In the above figure, the client's web browser sends an HTTP request to the web server over the Internet. The web server handles the request, and interacts with

the web application server. The web application server then interacts with the database server and prepares the requested document and returns an HTTP response back to the client through the web server.

2.2 Components used for developing the application

As it was mentioned above, web applications are developed using server side and client side scripting languages and they require a web server to run and a database server to function. The following components were used to develop the sample web application.

2.2.1 Web Server

A Web Server is a host server for web applications and is necessary to run them. Uniform Server was used as a host server to run the sample application. It is a free and open source WAMP server. WAMP is a web server for Windows Operating System which basically consists of Apache, MySQL, and PHP. It stands for Windows, Apache, MySQL, and PHP. There are many free WAMP servers available. Different servers may have different features and support for scripts and database. For example, some are portable and can be installed on a portable drive. Some support other database and server side script too, apart from just MySQL and PHP. Different features of server make developers easy to choose one according to their need. Uniform Server supports Perl as a server side scripting language along with PHP. It is easy to use, portable, and light. It only takes about less than 30MB space. The main reason why Uniform Server was chosen is its portability. The portability feature is very useful at the development phase, as the developer might frequently need to carry it. Figure 3 shows the control panel of a running Uniform Server.

2.2.2 Language and Scripts

Computer programs are a set of organized instructions (4) and in simple terms, those instructions are codes which can be written in different programming or scripting languages. In that sense, web applications are also created with languages, and, or scripts. For the development of the sample web application, HTML and CSS, PHP, and JavaScript were used.

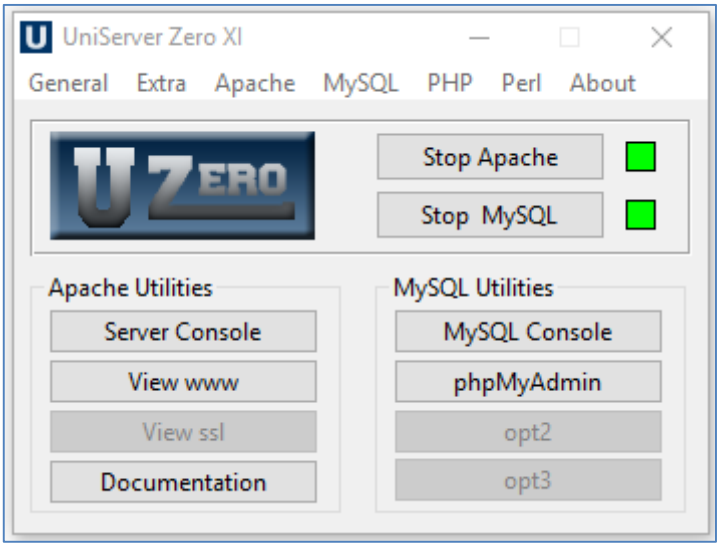


FIGURE 3. Control Panel of Uniform Web Server running on local host.

HTML

HTML stands for HyperText Markup Language. It is used along with CSS and JavaScript to create web pages as well as to make a user interface for web applications (5). HTML sets the layout and structure of a web page. To do so, it has various elements and for each element there may have several attributes. Each element starts up and ends with tags and also describes a different component of a web page. The following figure gives a glimpse of html elements and attributes.

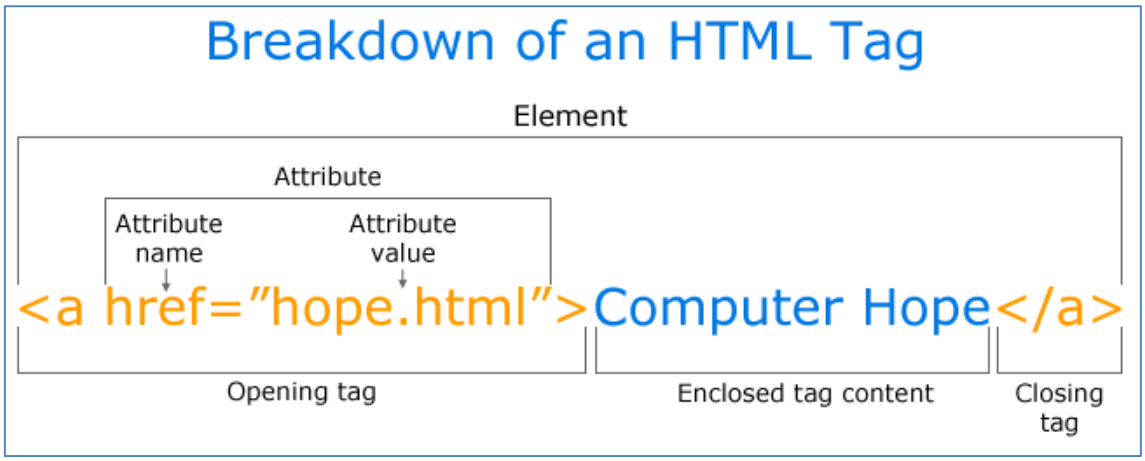


FIGURE 4. HTML Element and its component (6)

CSS

CSS stands for Cascade Style Sheet. It is a language which describes how a document written in a markup language is presented. In HTML, it is used to style the content of an HTML Element or of a whole HTML document. Mozilla Developer Network describes CSS in the following way:

> Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or other markup languages such as Scalable Vector Graphics (SVG). CSS describes how the structured elements in the document are to be rendered on screen, on paper, in speech, or on other media. The ability to adjust the document's presentation depending on the output medium is a key feature of CSS. (7.)

CSS can be used in two ways. Either it is embedded with an HTML file or placed as a separate file in the web server. In later case, a link to it is given in the HTML file. Sometimes both ways can be used at the same time. The following figures (Figure 5 – 8) clarify the different ways of using CSS in HTML.

```
<head>
<style>
body {
    background-color: linen;
}

h1 {
    color: maroon;
    margin-left: 40px;
}
</style>
</head>
```

FIGURE 5. Example of CSS embedded in HTML (8)

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

FIGURE 6. Example of a link to a CSS file in HTML (8)

```css
body {
    background-color: lightblue;
}

h1 {
    color: navy;
    margin-left: 20px;
}
```

FIGURE 7. Example of a CSS file (8)

```html
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
<style>
h1 {
    color: orange;
}
</style>
</head>
```

FIGURE 8. Example of a mixed way of using CSS (8)

JavaScript

JavaScript is the most popular client side scripting language. It is a light weight, interpreted, prototype-based object-oriented language with a first class function, mainly used for web pages (9). It can change the web page behavior according to an event change thus making the page dynamic. But it can also be used in a non-web environment such as pdf documents and desktop widgets (10). The ways of using JavaScript in web applications are same as using CSS. Either it can be embedded with HTML, or it can be linked to a separate JavaScript file located in the web server, or sometimes both ways can be applied at the same time.

PHP

“PHP (recursive acronym for PHP: *Hypertext Preprocessor*) is a widely-used open source general-purpose scripting language that is especially suited for web

development and can be embedded into HTML" (11). "PHP code may be embedded into HTML code, or it can be used in combination with various web template systems, web content management system and web frameworks" (12). In a file, PHP starts with a '<?php' tag and ends with a '?>' tag. The PHP processor only processes codes inside those tags.

Unlike JavaScript, PHP codes are executed in the server. It generates outputs in the server and sends them to the client browser. Because the browser receives responses as documents, it can never know the actual PHP codes used to generate the document. Apart from HTML output, it can output many kinds of files including images and pdf files.

The main use of the PHP is to develop dynamically generated web pages. It has a wide scope of use. It can be used for different major operating systems and it has support for many database systems and web servers. It can be used both for procedural programming and object oriented programming.

2.2.3 Database

Database is a collection of data in a structured way. But generally in computer terms, the word 'database' is used to refer Database Management System, or DBMS in short. "A database management system (DBMS) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data" (13). Basically, there are 2 types of databases – Relational, and –Non Relational. Most of the relational databases use the SQL query language. MySQL and MongoDB are examples of the relational and non-relational database management systems. In the relational database, data is organized in one or more tables of unique rows and columns and each table may have a connection to others by some common attribute. All relational databases including MySQL use SQL queries for querying and maintaining data.

A database is an essential part of web applications. It is needed to store different kind of information necessary to interact with users. For the sample application, the MySQL database was used. "MySQL is the world's most popular open source database" (14).

3 VULNERABILITIES AND ASSOCIATED RISKS

In this section, some common types of web application vulnerabilities and risks associated with them are briefly described. The term “vulnerability” is used here to refer to weak points in web applications. They can be used to attack easily by attackers.

Vulnerability is a weakness in a computer security which allows attackers to reduce a system’s information assurance (15). From security point of view, it is a kind of hole in the web application system. So, it serves as a gateway for attackers to fulfill their interests in the system. There might be more than one type of vulnerabilities in the system. Attackers can use one or a combination of more than one of such vulnerabilities to attack. Each vulnerability is associated with some kind of risk or threat to the application. Based on the impact of damage on the application, the degree of risk associated with vulnerabilities is different. The impact of damage depends upon the nature of the application and also the nature of business which owns that application. Some damages are more serious than the other ones.

Ron Lepofsky (16) has classified vulnerabilities in a web application as follows:

- authentication
- session management
- access control
- input validation
- redirects and forwards
- injection flaws
- unauthorized view of data
- error handling
- cross-site scripting
- security misconfigurations
- denial of service
- related security issues

OWASP has released a document called OWASP Top 10 2013. The document is about the most critical web applications vulnerabilities and was produced with the help of different kinds of security experts around the world (17). The top 10 most critical vulnerabilities, mentioned in the document, are as follows (18):

- Injection
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Missing Function Level Access Control
- Cross-Site Request Forgery (CSRF)
- Using Known Vulnerable Components
- Unvalidated Redirects and Forwards

It should be noted that both of these lists are not a complete set of vulnerabilities. These are the vulnerabilities, only prevalent for a particular time. Along with the development of new web technologies, new types of vulnerabilities may also emerge and the degree of their harmful effect may change.

If we look at the both lists of vulnerabilities, many of them are common and few others are different with each other. Brief descriptions of the common ones are being presented here.

3.1 Injection Flaws

On its website, University of Pennsylvania, Information Systems and Computing defines:

> Injection flaws allow attackers to relay malicious code through a web application to another system. These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL (i.e., SQL injection). (19.)

Injection is a kind of weak point in the web application through which attackers can inject some sort of malicious code of their interest. Malicious code is embedded with the user input data and passed to the application. If user input data is not properly filtered in the system, in that case, the interpreter processes the malicious code as a normal legitimate user input and the system outputs accordingly.

The level of risk associated with the Injection flaw is very high. An attacker can severely damage the application or steal confidential information. For example, they can get access to the confidential data, delete data, change the content, and destroy the whole system of the application. Some examples of injection flaws are the SQL injection, OS injection, and LDAP injection.

3.2 Lack of Proper Authentication and Weak Session Management

Authentication is an act of “verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system” (20). It refers to the process of verifying either a user, or a process, or a device, which requests to communicate with either the whole application or with a part of it in order to make sure that only the intended user, process, or device can get access to the application and its resources.

A session is a sequence of all the activities between a client (web browser) and a web server for a particular log in and log out period. The activities are generally associated to within the log in and log out period of the same user. (21.) So, there is a different session for each different user. For a number of different reasons, a web application requires to hold session information. For example, there may be different contents to deal with according to the user’s preference, or the type of user; or there may be security issues.

Effective authentication and proper session management are vital for a web application to be secure. The level of risk associated with the authentication and session management is high. Attackers usually gain access to the system through hijacking a username and a password or session IDs. They can access secret information and data while pretending that they are the legitimate user of the application.

3.3 Cross Site Scripting (XSS)

“Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application” (22). In Cross-Site Scripting, attackers exploit the user’s trust over a vulnerable web application. In this attack, they insert malicious JavaScript or html codes through user input fields to a page. It generally occurs when the application sends user input data as a part of a webpage, without properly validating to the user’s browser.

The risks associated with Cross-Site Scripting include a hijacking session, an unauthorized changing of the contents of application, redirecting the application to another website, and insertion of some malicious codes or links. The level of risks is high. By the hijacking session, attackers can get secret and important information.

3.4 Insecure Direct Object References

“The threat of insecure direct object reference flaws has become commonplace with the increased complexity of web applications that provide varying levels of access to enable users to gain entry to some components, but not others” (23). It refers to a provision in the web application where references to internal objects are directly exposed. According to OWASP,

> “A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.” (18.)

Vulnerability comes from the direct reference of an object that is used to get access to that object. Objects can be a file, or a directory, or a database key. Applications provide direct access to those objects based on a user supplied input value. For example, a user who has certain authorities to the application can alter the reference value of an object to get access to it even they are not authorized.

If the application does not have a mechanism to verify the user for each of such objects, then the application is vulnerable.

This vulnerability allows attackers to bypass authorization and access resources in the system directly, for example database records or files. To protect from attacks, direct references should be avoided. Instead of using direct references, easy to validate indirect methods should be used, such as index, and indirect reference map. If the direct exposure of objects and references is necessary, users should be verified to see if they are authentic to use before giving access to such objects. (24.) The risk associated with insecure direct object references is that an attacker can compromise confidential data. The degree of risk depends upon the nature of accessed resources. The following figure clarifies how the parameter can be altered to get access to an unauthorized resource.

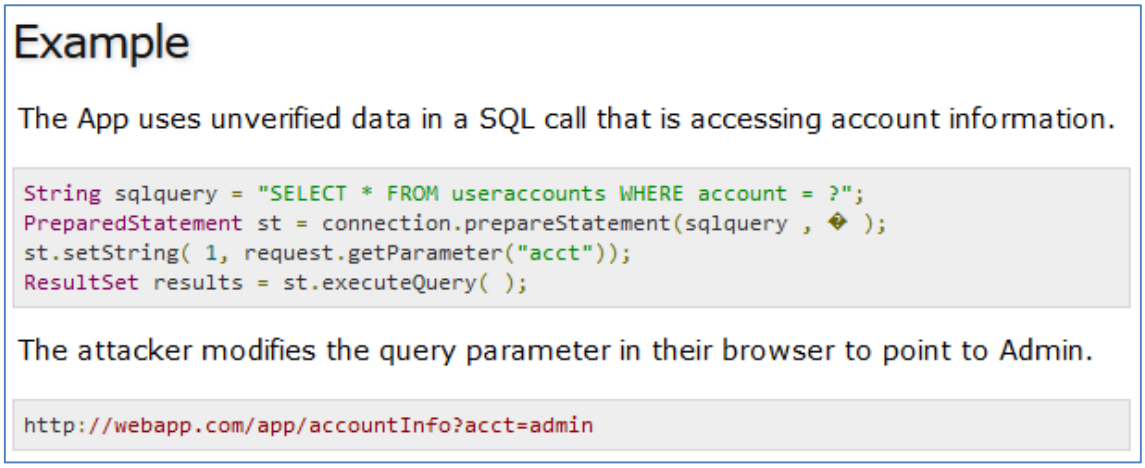


FIGURE 9. Insecure direct reference of database key (25)

3.5 Security Misconfiguration

A good security configuration is always essential for secured web applications as the misconfiguration exposes the application to a risk of attack. The components of the web applications such as database, libraries, platform, servers, and software modules should be configured properly. Unless necessary, no components should be given access with full privileges and they should always be up-to-date. According to OWASP, “Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, database, framework, and custom code” (26).

The risk associated with the security misconfiguration is that attackers may exploit components and compromise the database and web servers. To protect from this attack, both developers and system administrators need to work together to properly configure each component.

3.6 Sensitive Data Exposure

“Sensitive data exposure vulnerabilities can occur when an application does not adequately protect sensitive information from being disclosed to attackers” (27). Sensitive data is the data that should be kept secret and protected from outsiders. The data related to a personal identity such as Name, Address, Personal ID, and Tax ID and confidential data such as Bank Account Number, Credit Card Number, session ID, and authentication credentials are examples of sensitive data. This kind of data should be protected with extra security. If the application does not have a provision for enough protection to such data either in a rest mode or in a transit mode, it is dangerous to use the application. Data is exposed if it is not encrypted. Or even if it is encrypted, a weak key generation or algorithm may have been used.

The risks associated with a sensitive data exposure are that attackers can steal the personal identity and credit card information or modify other sensitive data as their interest. So, to protect the sensitive data exposure from prospective attackers, sensitive data should be properly encrypted and it should not be stored if not necessary. Auto fill options for forms with sensitive data should be disabled (28).

3.7 Missing Function Level Access Control

In most of the instances, the applications only show the functionality in their UI based on the user. For example, if the user is a teacher, then only functionalities for the teacher are appeared in it and if the user is a student, then only functionalities for the student are appeared. The reason to do so is to prevent users to view data or resources to which they are not authorized. But this approach is just hiding functionalities and it is not enough to make the application secure. A user may somehow know the link to get unauthorized resources, or the link to the unauthorized resources might be on the page but just not visible in the UI. They

can simply guess the parameter and change it in the browser address bar. In the above mentioned case, for a hacker, it is easy to get unauthorized resources. If they can get it, then this is an example of the Missing Function Level Access Control vulnerability. (29.)

This vulnerability has little difference with the Insecure Direct Object Reference vulnerability. The Insecure Direct Object Reference vulnerability allows an attacker to get access to unauthorized data whereas the Missing Function Level Access Control provides unauthorized access to the functionalities. But the final outcome of both vulnerabilities is the same: data is stolen. (29.) This can have serious consequences. The company may suffer a financial loss as well as loose its reputation and its customers' trust. The users on the other hand may have their identity stolen.

Sensitive functional requests should be protected so that outsiders, or general users who are not allowed, cannot get access to those functionalities. Functionalities, which are in the UI, should be verified upon each request. Giving an equal privilege to all users should be avoided. Instead, a privilege should be given according to users' need.

3.8 Cross Site Request Forgery (CSRF)

Cross-Site Request Forgery is an attack which tricks to the victim's browser and does undesirable things which the application considers has been done by an authentic user but the user does not have any knowledge about it. So, it actually exploits the trust of the application to the user. GitHub Security describes:

> Cross-site request forgery, or CSRF, takes advantage of a user's authenticated browser state to make requests on their behalf from a malicious website. For request handlers that do not require an additional piece of authenticating information (e.g. a CSRF token) this could lead to the unauthorized modification of a user's data or settings. (30.)

An attacker can perform this kind of attack only when the user has opened and is logged on to the vulnerable application and also has open the attacker's page. Attackers use a scripting language like JavaScript to perform such undesirable activities. They trick the victim (user) to click on a link. The link is generally not

shown in the text but it is embedded in an image or other elements. If the victim has a valid session on the target application, and if they click the attacker's link, the code will then be executed by the server and the victim does not know it immediately.

To prevent a CSRF attack, the use of anti CSRF Tokens, strictly to apply a re-verification provision in the server when performing activities that have a significant impact on database, are recommended. The risk associated with Cross-Site Request Forgery can be minor to major, depending upon the type of user (victim). If an attacker can attack with an admin credential, then the damage might be severe.

3.9 Using Known Vulnerable Components

"Vulnerabilities in third-party libraries and software are extremely common and could be used to compromise the security of systems using the software" (31). Web Applications have been built using different kinds of third-party libraries and frameworks. In many applications, while at the developing phase, developers usually do not pay enough attention to using up-to-date components such as libraries, frameworks, or modules, which makes it easy for attackers to attack the application. To prevent this vulnerability, developers should pay enough attention while choosing such libraries and framework. They should be confirmed that the components they are going to use are updated and secure.

3.10 Unvalidated Redirects and Forwards

"Unvalidated redirect vulnerabilities could allow an attacker to redirect a user to an untrusted site using functionality in a trusted site" (32). During a session, the web application may need to redirect and forward to other pages or websites several times for different purposes. And while doing so, they use untrusted and unvalidated data to determine those pages and websites. That untrusted and unvalidated data can be used by attackers to redirect to phishing and malicious sites. The malicious site can easily gain the user's trust because in the malicious site's URL, the original site's name, from which the user has been redirected, also

appears (33). To prevent this vulnerability, OWASP (34) has given the following suggestions to follow:

a. Simply avoid using redirects and forwards.
b. If used, do not involve user parameters in calculating the destination. This can usually be done.
c. If destination parameters cannot be avoided, ensure that the supplied value is valid and authorized for the user.

4 PENETRATION TESTING

4.1 Definition

Kevin M Henry defines “Penetration testing is the simulation of an attack on a system, network, piece of equipment or other facility, with the objective of proving how vulnerable that system or "target" would be to a real attack” (35). For this thesis context, Penetration Testing is an attack to a web application. It is done intentionally and with the permission of the owner of that application, in order to find vulnerabilities in it. It helps developers to determine the areas where they are strong enough for defending outside attacks and also flaws in their codes that they need to improve in order to make the application secure. It is a part of a security audit. It not only identifies the vulnerabilities but also suggests remedies.

4.2 Objective

The main objective of Penetration Testing is to help making an application secure by finding vulnerabilities so that the developers can fix them before it is attacked in real. But it also depends on the goal of the company regarding Penetration Testing. For example, it can be done to test the organization's security policy compliance, its employees' security awareness and the organization's ability to identify and respond to security incidents (36). It also helps to access the possible loss or other consequences on resources or data in case of attack. Some general objectives include to prevent a data breach, to test an application’s security control, to ensure that the application is secure before making available for a real use and to get a baseline information about overall security strengths and weaknesses for making security policies.

4.3 Testing Needs and Benefits

Depending on the types of data or resources an application holds, the vulnerability in the application, and the service interruption associated with it, might be costlier to the company. The security status of an application will not always remain the same. An application, which is considered to be secure at a particular point of time, might turn to vulnerable in near future. It is impossible to always protect

all the data and resources. So, a company continuously needs to identify, monitor, and prioritize security risks for its application. The following paragraphs describe the benefits of Penetration Testing (see (37)).

One of the benefit of Penetration Testing is that it helps wisely manage vulnerabilities. It provides detail information on all feasible threats, so that the company can group the threats by their possible impact and make a security policy accordingly.

It also helps minimize the possible loss of service interruption of the application. Once the application is attacked, it can be costlier in terms of both time and money to retrieve data and resources again. Penetration Testing lets the authorities know in advance about the possible attack so that they can be well prepared to tackle it. In this way the company can minimize the chances of heavy loss due to a service interruption.

Penetration Testing can help preserve the company reputation and maintain customer loyalty. Compromised confidential data results losing reputation and customer loyalty. Penetration Testing lets the authorities know the vulnerabilities beforehand so that they can take preventive actions to protect confidential data and other resources from attackers.

4.4 Testing Frequency

Penetration Testing should be performed on a regular basis to ensure consistency, security, and smooth running of the application. It helps find out the new and emerging security threats that attackers may exploit so that preventive actions can be taken before attackers do their job. But apart from regular testing, the following are some specific instances when it is necessary to perform Penetration Testing (see (37)):

- Before starting the real use of the application
- If new applications are added to the system
- When significant upgrades or modifications are applied to the application
- After security patches are applied
- If end user policies are modified

4.5 Steps or Process of Penetration Testing

The Process or Steps are all the activities that are involved from the beginning to the end of Penetration Testing. The following are only a brief description of steps involved in the testing. Further information can be acquired from SANS Institute InfoSec Reading Room (see (38)).

- Determine the immediate goal of the test, for example to breach a personal information database
- Collect information about the way to get to the target, for example to the database
- Discover or identify the entry points to the network, for example performing port scanning
- Start exploitation of vulnerabilities using different techniques, for example brute forcing or phishing
- Take control of the application, for example doing things which are not allowed to do
- Evidence collecting, for example evidence collection of things done while taking control of the app
- Reporting, it involves writing a report about everything from the beginning to the end of testing
- Suggesting remedies for the vulnerabilities found while testing

5 OWASP ZED ATTACK PROXY (ZAP)

OWASP Foundation is a non-profit organization. “OWASP is an international organization and the OWASP Foundation supports OWASP efforts around the world” (39). OWASP is working in the field of web application security. ZED Attack Proxy is an OWASP flagship project, also known as ZAP. It is a tool used to find vulnerabilities in web applications. OWASP defines ZAP as an easy to use integrated web application penetration testing tool to find vulnerabilities (40). It is a free and open source software designed to use both by beginners and professional penetration testers. It is ideal for developers and functional testers for automated security test. But it should be used either by own applications or the ones which have been authorized to test “(ibid.)”

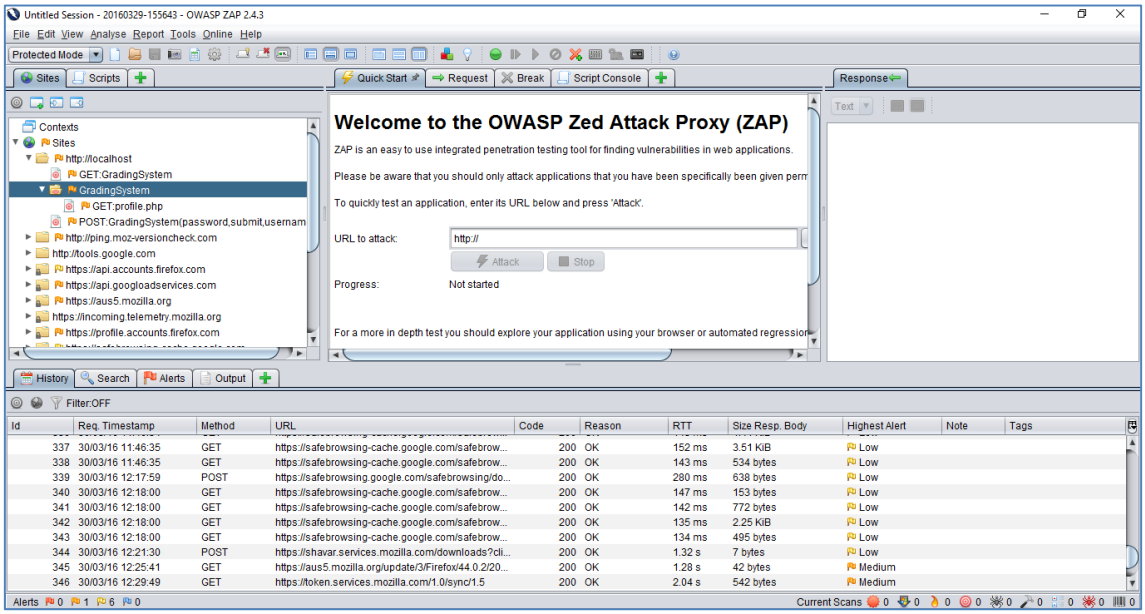


FIGURE 10. User Interface of ZAP

5.1 Principles behind ZAP

According to OWASP, there are some key principles behind ZAP. It is a free and an open source software. It does not have and will never have a commercial or pro version. It is a Cross Platform software, i.e. it can be used in different Operating Systems. It is easy to install and use. It requires Java pre-installed to install ZAP. Nothing else is needed. There are some videos available in the youtube. They help installing the software and learning how to use it. It has a full set of

documentations to get help with. It can work well with other tools. Tools can be found from add-ons. It has supports many languages. Involvement is actively encouraged. It can reuse well regarded components. (41.)

5.2 Main Features

Intercepting Proxy

ZAP is an intercepting proxy. This means that all the requests from the user to the web application and all the responses from the web application to the user browser can be seen through ZAP. It operates as a man-in-the-middle between the browser and the target application. It can intercept or modify any http/s traffic passing in both directions.

Active and Passive Scanners

Active Scanner actively attacks to the target application to find vulnerabilities while Passive Scanner only scans the responses from the application to the browser. So active scanning is riskier as it can make damage to the application. Therefore, it cannot be used without the permission of the owner of the application. And before starting Active Scanner for scanning the application, a backup of all data is strongly recommended. Passive Scanner is safe to use as it does not modify the responses received from the application.

Traditional and Ajax Spiders

The Spider is used to search for new pages (URLs), and links of other websites on a particular website. First, when the application is browsed manually, ZAP lists some URLs found on the manually visited pages. When Spider starts, it first looks those listed URLs to find new links or URLs. If found any, it adds the URLs on the list and again visits those newly found URLs. And this process will continue until it finds new URLs or links. Both Traditional and Ajax Spiders are for the same purpose. The first one is used for finding other than Ajax rich resources while the second one is to find Ajax rich web pages because they are more effective than Traditional Spiders.

WebSockets Support

WebSocket is a protocol that provides a two-way communication (full duplex) channel through a single TCP socket over the web (42). ZAP is able to provide WebSocket support. ZAP can see, intercept, change, and even fuzz all the WebSocket communications, or it can send new WebSocket messages. Detail information can be found from github zaproxy article (see (43)).

Forced Browsing (using OWASP DirBuster code)

Forced Browsing is a kind of attack where the attacker tries to enumerate or access the restricted resources which have no reference or any link in the application but exist and can be accessible (44). Brute Force techniques are used for a Forced Browsing attack in which the attackers either guess or use automated tools to find unlinked URLs within the application (45). The OWASP Forced Browsing attack is based on their DirBuster project (46). It is a multi-threaded Java application which is designed to brute force the unlinked directories in the application. For further reading, please refer to OWASP (47).

Fuzzing (using fuzzdb and OWASP JBroFuzz)

Fuzzing or Fuzz Testing is a software testing technique to find implementation bugs and coding errors. In Fuzz Testing, an attempt is made to make the application (software) crash by delivering a random, invalid or unexpected user inputs value to the application (software) and then monitoring to see if it crashes. If the application crashes or fails with the random user input value, then there may be a security issue. ZAP performs Fuzzing through the JBroFuzz project code which includes files from the fuzzdb projects (48).

Online Add-ons Marketplace (Extensibility)

ZAP is an open source project of OWASP. One of the ZAP principle is involvement of people as much as possible. It helps ZAP grow in terms of its usage and also extend the services it provides. To make active participation and contribution to a further development easier, there is an online marketplace provision for add-

ons in ZAP, where one can write and upload (through Google code project), download, and install add-ons dynamically. Add-ons extend ZAP functionality.

5.3 Developer Features

As OWASP mentioned (49), there are so many developer features in ZAP. It has an easy-to-use quick start tab. One just needs to enter the URL and click the attack button and attack the application. There is a provision of REST API which allows to interact with ZAP programmatically. It is useful for security regression tests (50). It can be accessed directly or via one of the client implementations. It has Java and Python API Clients support. When using ZAP UI, if one wants to use API, it should be enabled in the options API screen in the UI. ZAP can also be run in Headless Mode. If it runs in Headless Mode, API is automatically enabled.

ZAP has an Anti CFRS Token Handling mechanism. ”Anti CSRF tokens are (pseudo) random parameters used to protect against Cross Site Request Forgery (CSRF) attacks” (51). ZAP has provision for different kinds of authentication to use in the web application. Authentication methods have been defined in the context according to which authentication is handled. It has an Auto Updating feature for its add-ons. Add-ons can be updated even if ZAP is running. One does not even need to restart ZAP. It is always a good idea to check for updates for different add-ons before testing the application.

The latest version of ZAP (ZAP 2.4.3) has 4 different modes of operation, namely safe mode, protected mode, standard mode and attack mode. Safe mode can be used with any web application as no harmful actions are allowed in safe mode. But it is not useful for security testers. It is only useful for passive scanning. In protected mode, only the URLs in the scope can be attacked. It is safe to use with URLs outside of scope. Anything can be done in standard mode. So one should be careful while using ZAP in standard mode. In attack mode, if new nodes are found in scope, ZAP starts active scanning of the nodes immediately.

5.4 Simple Penetration Test Procedure

Simple Penetration Testing is easy to perform. The first thing is to configure the browser to proxy via ZAP. Then the application is explored manually. While exploring the application manually, Passive Scanning runs automatically. Then Spiders are used to find hidden contents of the application. After Spider scanning completes, Active Scanner is run to find the vulnerabilities. Before starting the test, or during the test, ZAP tools can be tuned according to the need of testing, depending upon the Web Application. After all the automated testing has been done, Manual Testing can be performed.

An easier way to start testing, after setting a browser proxy via ZAP, is to go to the quick start tab, type the address of the web application, and then click the attack button. It automatically starts Spider first and then an active scan. But this will not be a complete assessment of a web application. It can only find basic issues. For example, the logical vulnerabilities cannot be found with active scanners. Manual testing should also be performed to find vulnerabilities. And there are many tools available for manual testing within the ZAP. ZAP is a framework for combining other tools in a more robust way. So adding more functionality to ZAP is easier by either developing tools by own or downloading from online market place.

5.5 Finding Issues

After Scanning completes, ZAP shows the result summary in the form of different categories of alerts. Basically, alerts are potential vulnerabilities and have been categorized as high priority, medium priority, low priority, and informational priority, which indicates the degree of associated risks. A high priority alert means that an issue under this category is more serious than other priority alerts. Likewise, medium priority alerts, low priority alerts, and informational priority alerts are consecutively less and less serious. Alerts categories are indicated by different colour flags. As in the figure 11, alerts flags can be seen in the bottom left of ZAP window with numbers. The number beside the flags indicate the number of potential issues within that category. Starting from the left most flag and consecutive flags indicate high, medium, low, and informational priority alerts.

One notable thing regarding these alerts is that ZAP provides full detail of each issue it finds. It includes attack, evidence, description, other information and reference. It also suggests the solution of the issues. One can edit the alert details via the Edit Alert dialogue by double clicking an alert, saving it in an html or xml format and also comparing it with another session via Report Tab in the main menu. This can really help developers to make their application more secure. In figure 12, an edit alert dialogue and in figure 13, an html report of an alert generated by ZAP can be seen.

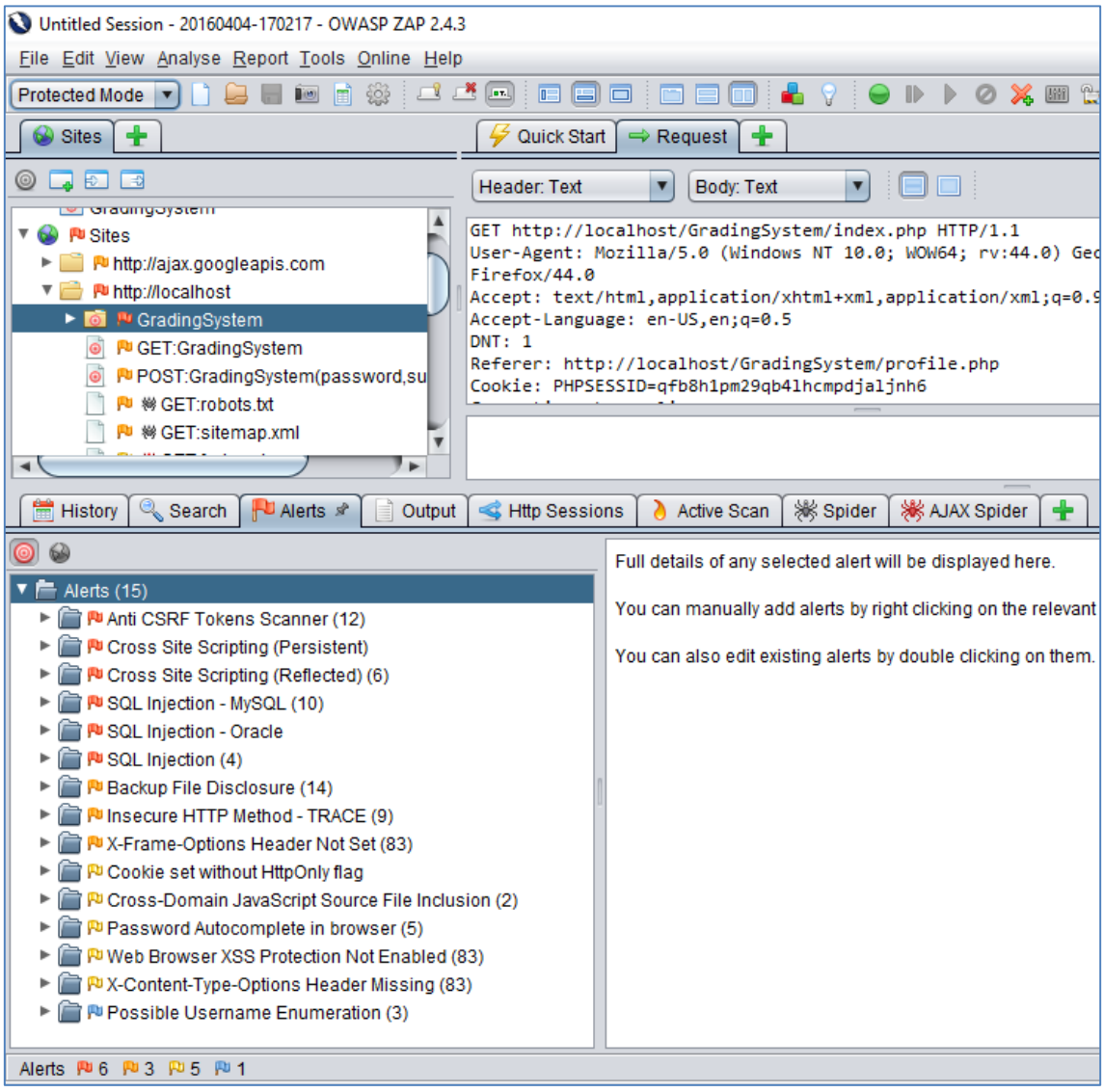


FIGURE 11. Alerts and Categories

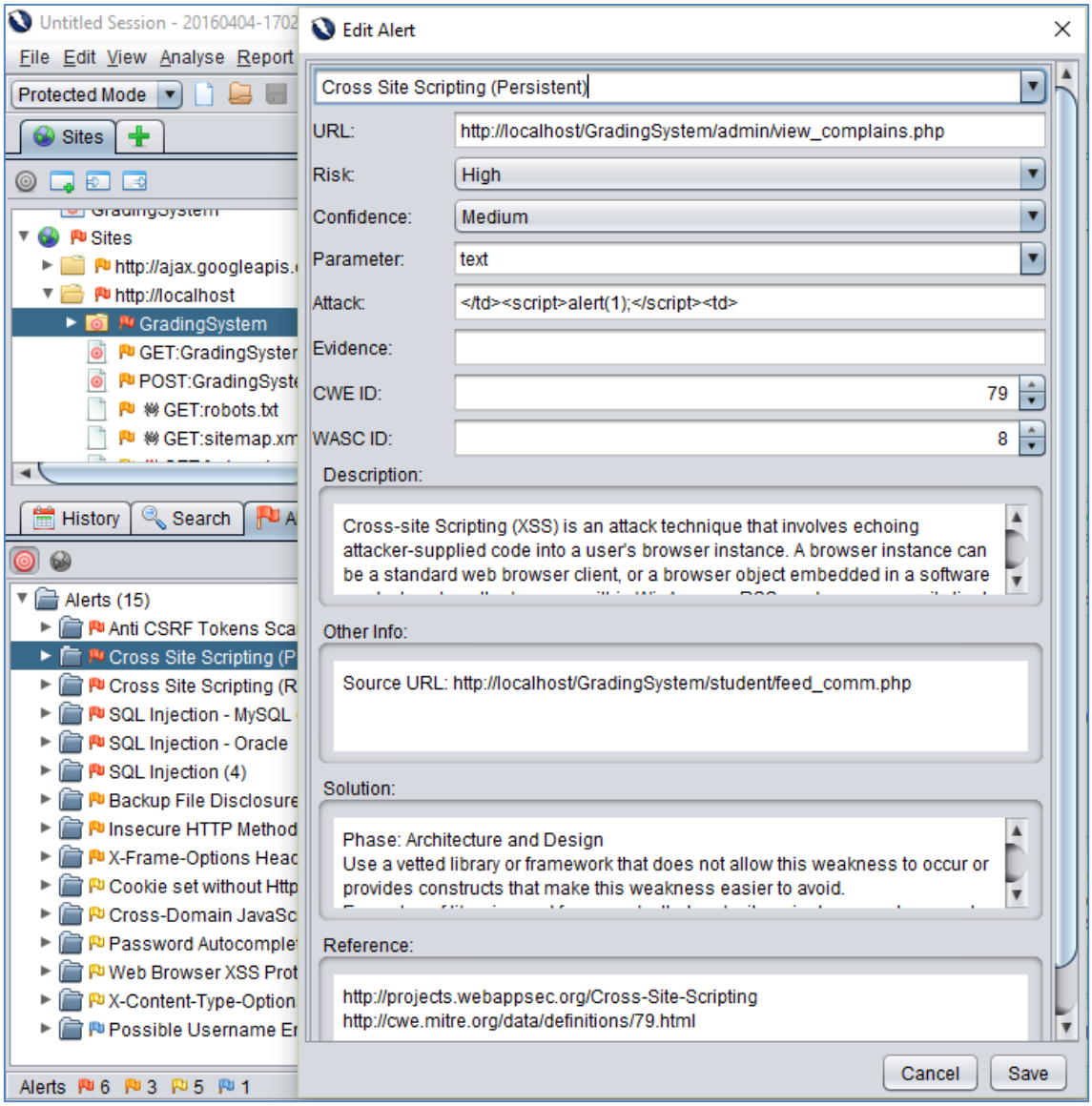


FIGURE 12. Alert in detail

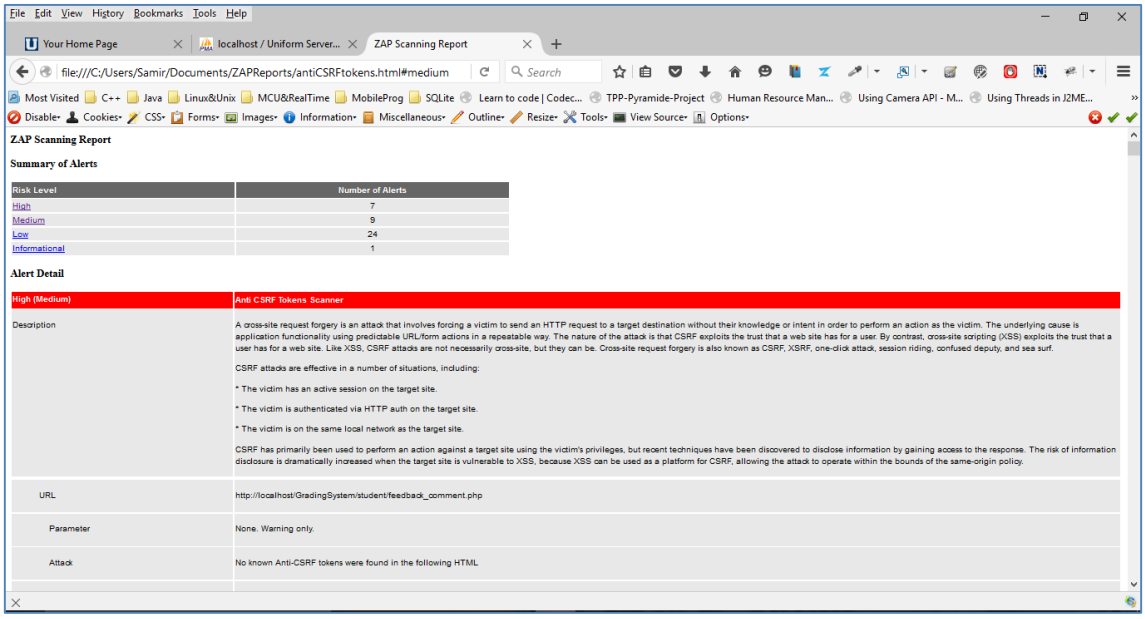


FIGURE 13. HTML Report on Alerts

6 SECURITY TESTING OF THE SAMPLE APPLICATION

For testing the application, the Firefox browser was used and simple steps that ZAP tutorial suggests for starting the test were followed. ZAP was set to protective mode to prevent any possible mistake of attacking real websites which there is no authority to test. The browser was configured to proxy via ZAP, the application was set in a context, and the application was explored manually, almost every pages. After that, ZAP normal Spider first and then AJAX Spider was started to crawl the application. And finally Active Scanner was run. The results of those scans are presented in the following tables.

TABLE 1. High priority alerts

S.N.	Name of Vulnerability	Number of Vulnerabilities found
1	Anti CSRF Tokens Scanner	10
2	Cross Site Scripting (Persistent)	3
3	Cross Site Scripting (Reflected)	6
4	SQL Injection – MySQL	24

TABLE 2. Medium priority alerts

S.N.	Name of Vulnerability	Number of Vulnerabilities found
1	Backup File Disclosure	14
2	Insecure HTTP Method – TRACE	9
3	X-Frame Option Header Not Set	83

TABLE 3. Low priority alerts

S.N.	Name of Vulnerability	Number of Vulnerabilities found
1	Cookies Set without HTTP Only Flags	1
2	Cross-Domain JavaScript Source File Inclusion	2

3	Password Autocomplete in browser	5
4	Web Browser XSS Protection Not enabled	83
5	X-Content-Type-Options Header Missing	83

TABLE 4. Informational priority alerts

S.N.	Name of Vulnerability	Number of Vulnerabilities found
1	Possible Username Enumeration	3

To include all kinds of issues in this thesis is not practical as it takes too much time and space. So, I would like to go through in detail each issue on high priority alerts that ZAP found on the sample application. Basically, there are 3 types of issues found in the application. The first one is CSRF, the next one is XSS, and the third one is the SQL injection.

6.1 CSRF

Description

As already has been described briefly in the vulnerability chapter, CSRF is an attack where an attacker tricks a valid user (a user with an active session) to perform a certain action, intended by the attacker but not intended by the user. The user does not have any prior knowledge about that. Sometimes there might not be any visible response of the action. So, in that case the user does not even know immediately which action was performed. To perform a CSRF attack effectively, certain conditions should be met. For example, the user should actively be using the application at the moment. In other words, they should have a valid user session. Along with the target application, they should have opened the hacker's site, or email, or chat window. And in the application server side, if there is no re-authentication provision before performing certain tasks, then it is more prone to CSRF vulnerability.

Generally, a piece of JavaScript code is used for a CSRF attack but not always necessarily. The code is embedded into the attacker's own page or is sent as a

link through an email or a chat application. While the user has an active session on the targeted web application, if they click on the link, their browser sends a request to the server. Because the user has a valid session, the server takes the request as a valid one and executes it. In this way the attacker's intended action is performed.

In a CSRF attack, the attacker tries to exploit the application server's trust for the users. Whenever a request comes, the server checks whether it is coming from a valid session or not. If it finds that the session is valid, then it executes the code, no matter whether it is forgery or valid. To prevent it to some extent, developers should develop an application in a way that it should not trust a user for actions which have a severe impact on the database, even if the user has a valid session. Next, they should always use POST instead of GET or REQUEST for any kind of request that performs an action. But these are not a complete and effective solutions. A more effective way to prevent a CSRF attack is to use Anti CSRF Tokens.

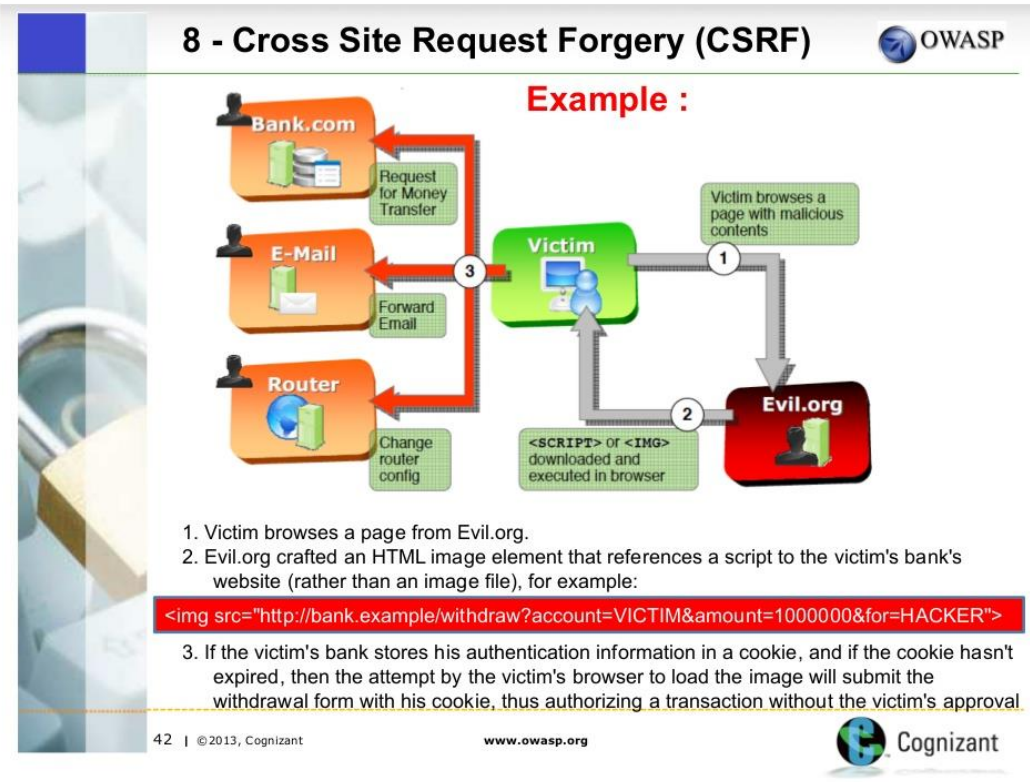


FIGURE 14. CFRS example (52)

Anti CSRF Tokens are usually implemented through a random token and generated each time when a form is submitted by the user. It is embedded in the form as a hidden field. Whenever a request is sent to the server, it compares the token with the value stored in the session. The server only executes the request if the token matches. Below is one example of how token is set.

```
1.  <?php
2.
3.  $token = md5(uniqid(rand(), TRUE));
4.  $_SESSION['token'] = $token;
5.  $_SESSION['token_timestamp'] = time();
6.
7.  ?>
8.
9.  <form action="/post.php" method="POST">
10.  <input type="hidden" name="token" value="<?php echo $token; ?>"
11.  <p>Subject: <input type="text" name="subject" /></p>
12.  <p>Message: <textarea name="message"></textarea></p>
13.  <p><input type="submit" value="Add Post" /></p>
14.  </form>
```

FIGURE 15. Example of using random Token (53)

Manifestation in sample application

According to the testing result, there are 10 different files which are vulnerable for a CSRF attack. Each of them was examined manually and it was found that each of those files is associated with some kind of database editing functions. Some of them are creating new fields and others are either deleting or updating existing fields of data. Considering the nature of the sample application, the risk seems severe as the attacker can manipulate the database according to their will.

One of the things that was mentioned earlier applies in the sample application. There is no provision of re-authenticate for activities that are directly related to database editing. This facilitates a CSRF attack. Apart from this, there are few situations which help a CSRF attack more effectively. For example, if the victim has an active session on the target site, or if the victim is authenticated via HTTP

auth on the target site, or if the victim is on the same local network as the target site, then it will be easy for attackers to attack. And the most obvious thing in the application is that no anti CFRS token has been used to prevent a CSRF attack.

Fixing the Problems

To fix CSRF vulnerability, the ZAP testing report suggests the following preventions:

To prevent a CSRF attack usually requires to include an unpredictable random token in each HTTP request by the user. The token should be unique per each user session and placed in a hidden field. This causes the value to be sent in the body of the HTTP request avoid its inclusion in the URL, which is more prone to exposure ((16), 74).

Anti-CSRF packages such as the OWASP CSRF Guard for the Java Application, and the CSRF Protector Project for the PHP Application can be used (54). “OWASP CSRF Guard is a library that implements a variant of the synchronizer token pattern to mitigate the risk of Cross-Site Request Forgery (CSRF) attacks” (55).

A re-authentication mechanism can be utilized if the request coming from a user is for a risky kind of operation, such as payment or purchasing online. Instead of using a GET method for any request that triggers a state change, a POST method should be used.

6.2 XSS

Description

A Cross-site scripting (XSS) attack is a kind of injection which attackers use to inject their malicious code into a website so that it appears to be a part of that particular page of that website to the user’s browser. When a visitor visits the infected page, their browser considers the malicious code as valid code from the website, and executes it. To perform an XSS attack, generally 3 parties are involved, namely the attacker, the victim, and the website where the malicious code is injected. The following figure is an example of how an XSS attack is performed.

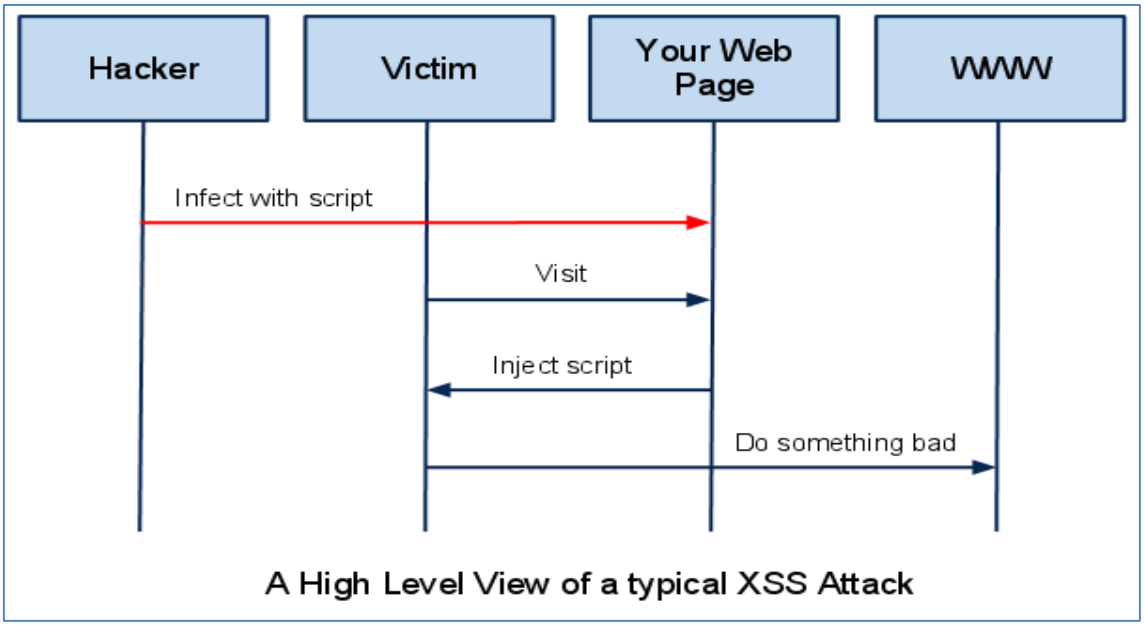


FIGURE 16. Example of XSS Attack (56)

Basically, there are 2 types of XSS attacks: Persistent and Non Persistent. A Persistent attack, also known as Stored or Type-I XSS, is an attack in which the attacker is successfully able to store their malicious codes into the database of a web application through user input fields. For example, in a web application there may be user input fields on pages such as change profile, make comments, message forums where the user can insert any kind of text inputs they want. When a victim visits the web application and requests a page in which the malicious code has been stored, the malicious code is downloaded to their browser and the browser executes the code. There is no need to click on some links or visit malicious websites.

In a Non-Persistent Attack on the other hand, nothing is stored into the web application database. Instead, attackers make a link and send it to the victims, for example through an email. The link is embedded with malicious codes. When the victim clicks on the link, their browser requests the server and gets a response from the server. If the server does not have any provision of filtering the parameters of user requests, then it responses the request as it is. It means that if the request contains malicious codes, the response will also contain them. And when the victim's browser gets the response with the malicious codes, it executes accordingly. A Non Persistent attack is also called Reflected, or Type-II. The following figures clarify more about a Persistent and a Non Persistent XSS attack.

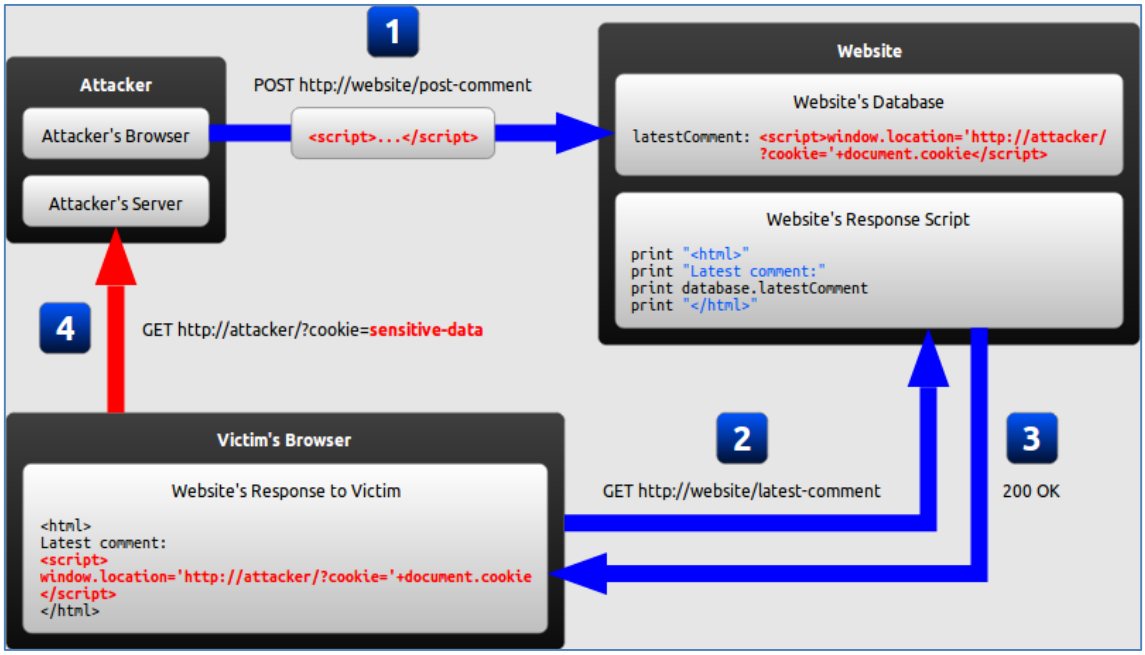
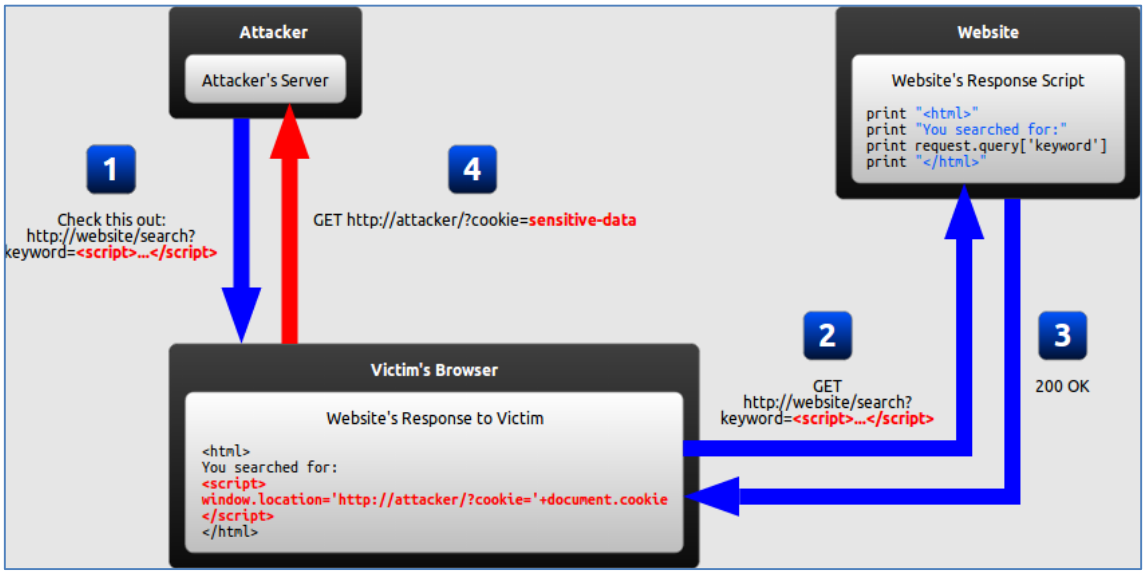


FIGURE 17. Persistent XSS attack (57)



FFIGURE 18. Non Persistent XSS attack (57)

The difference between CSRF and XSS is the way of attack. In a CSRF attack, the user or a victim clicks on the malicious link in the attacker's website or email. By clicking the link, the victim unknowingly performs a particular activity which the attacker intends to perform. So, only one user is affected at a time, but is directly affected. That means that the attacker is certain about the victim before a real attack is performed. But in XSS, the number of visitors affected depends upon how many of them visited the malicious code injected website at the same time

and also the attacker does not know beforehand who will be going to suffer from it.

To perform an XSS attack, attackers generally use vulnerable but popular web-sites which many people are likely to visit. An XSS attack is done with the help of a client side scripting language. Because the injected code is not going to be executed in the web application server but in the user's browser, they inject some JavaScript code inside the vulnerable website. This is only possible if the vulner-able website takes the user input without validating or filtering it and keeps it on the web page. For example, in a blog or a social networking site, there are user input fields. If the website does not have a provision to validate and filter those input fields, one can successfully inject JavaScript codes through them.

Attackers may have different reasons for XSS attacks. For example, they may want to access cookies, session tokens, or other sensitive information retained by the browser in order to hijack the session, change the contents of the website, or redirect a website to another website. These scripts can even rewrite the con-tent of the HTML page. "The attacker can register a keyboard event listener using `addEventListener` and then send all of the user's keystrokes to his own server, potentially recording sensitive information such as passwords and credit card numbers" (57).

Manifestation in prototype application

The testing result shows that there are a total of 6 instances of Reflected and 3 instances of Persistent XSS vulnerabilities in the application. According to the result, the following are some sample URLs which are vulnerable:

TABLE 5. Instances of Persistent XSS in the example application

URL	http://localhost/GradingSystem/admin/view_complains.php
Parameter	text
Attack	</td><script>alert(1);</script><td>

TABLE 6. Instances of Reflected XSS in the example application

URL	http://localhost/GradingSystem/Teacher/class_grade.php?choiceclass=%3C%2Fh1%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Ch1%3E
Parameter	Choiceclass
Attack	<script>alert(1);</script>
Evidence	<script>alert(1);</script>
URL	http://localhost/GradingSystem/Student/teacher_list.php?course=%27%22%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E
Parameter	Course
Attack	'"<script>alert(1);</script>
Evidence	'"<script>alert(1);</script>
URL	http://localhost/GradingSystem/Teacher/studentListGrade.php?class=%27%22%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E
Parameter	Class
Attack	'"<script>alert(1);</script>
Evidence	'"<script>alert(1);</script>
URL	http://localhost/GradingSystem/Teacher/student_grade.php?choicestudent=%27%22%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E
Parameter	Choicestudent
Attack	'"<script>alert(1);</script>
Evidence	'"<script>alert(1);</script>

All the vulnerable files were checked manually. As in the case of CSRF, an XSS vulnerability also lies on codes which are associated with some kinds of database functions. The intensity of risk is always severe if an attacker is able to manipulate the database of the application. In the case of Reflected XSS, I tested with a JavaScript code to redirect to another website, which successfully redirected to an external website. Also I tried ZAP attack code which successfully created an alert window. One thing I noticed in my code is that I had not made any provision for filtering user inputs. As ZAP suggests, we should never use user inputs directly. Figure 19 shows an example code file which is vulnerable for an XSS attack.

Vulnerability lies on a ‘choiceclass’ parameter which directly comes from the user input. One can send malicious code through that parameter to the application. Or

one can directly type a URL with malicious JavaScript code in the address bar of the browser. While the session was active, I tested a URL with embedded JavaScript code to redirect the page to an external website. The URL http://localhost/GradingSystem/Teacher/class_grade.php?choiceclass=<script>setTimeout(function () { location.href = 'http://www.google.com'; }, 0);</script> successfully redirects the browser to the Google website.

Fixing the Problems

To prevent an XSS attack, both the user inputs and application outputs should be properly encoded. In the ZAP report, OWASP recommends to use libraries and frameworks to avoid XSS vulnerabilities. Examples of such libraries and frameworks include Microsoft's Anti-XSS library (58), the OWASP ESAPI Encoding module (59), and Apache Wicket (60). These help properly encoding output or response as it is necessary to fix the XSS vulnerability. Html Sanitizer, OWASP Java HTML Sanitizer, Ruby on Rails SanitizeHelper and PHP Html Purifier are some examples of libraries which can be used for input encoding (61). Another recommended task to be performed is that all kinds of security checking and validation should be done on the server side, no matter whether the client side provides such checking and validation. In most of the cases, attackers can easily bypass client side provisions. To make the application secure, security provisions on the client side should not be relied on.

```
<?php
session_start();
if(!isset($_SESSION['login_user'])){
        echo "You are not allowed to view this page.";
        header("location: ../index.php");
}
?>
<?php
ini_set('display_errors', '1');
require '../databaseConnection.php';
$class = ($_REQUEST['choiceclass']);
?>
<div class="customjob">
        <center><h1>CLASS <?php echo $class; ?></h1></center>
        <table class="work" border="1" cellspacing="2" cellpadding="2">
                <tr>
                        <td colspan="2" align="center">STUDENTS</td>
                        <td colspan="5" align="center">GRADES</td>
                </tr>
                <tr>
                        <td align="center">STUDENT ID</td>
                        <td align="center">STUDENT NAME</td>
                        <?php
                        $sql = "SELECT course_name, course_id FROM course WHERE class=" . $class;
                        $stmt = $conn->query($sql);
                        $result = $stmt->fetchAll(PDO::FETCH_ASSOC);
                        $courses = [];
                        foreach ($result as $column) {
                                echo '<td align="center">' . $column['course_name'] . '<br/>(' . $column['course_id'] .')'. '</td>';
                                $courses[] = $column['course_id'];
                        }
                        ?>
                </tr>
                <?php
                $sql = "SELECT student_id, name FROM student_class INNER JOIN user ON student_class.student_id=user.user_id
                WHERE student_class.class=" . $class;
                $stmt = $conn->query($sql);
                $result = $stmt->fetchAll(PDO::FETCH_ASSOC);

                foreach ($result as $row) {
                        echo '<tr><td>' . $row['student_id'] . '</td><td>' . $row['name'] . '</td>';
                        $sql2 = "SELECT course_id, grade FROM grades WHERE grades.student_id='" . $row['student_id']."'";
                        $stmt2 = $conn->query($sql2);
                        $result2 = $stmt2->fetchAll(PDO::FETCH_ASSOC);
                        $gradeMap = [];
                        foreach ($result2 as $row2) {
                                $gradeMap[$row2['course_id']] = $row2['grade'];
                        }
                        foreach ($courses as $courseId) {
                                if (array_key_exists($courseId, $gradeMap)) {
                                        echo '<td>' . $gradeMap[$courseId] . '</td>';
                                } else {
                                        echo '<td>*</td>';
                                }
                        }
                        echo '</tr>';
                }
                ?>
        </table>
</div>
```

FIGURE 19. Example of XSS vulnerable code

6.3 SQL Injection

Description

Among other kinds of Injections, SQL Injection is an attack where attackers use to inject or insert SQL queries through user supplied parameters into the database. So, it means that “An SQL injection needs just two conditions to exist – a relational database that uses SQL, and a user controllable input which is directly used in an SQL query” (62).

Modern web applications use database to interact with users. Users give some inputs in the form of parameters or database queries through user input fields to the application. In the application, the server processes the request and with the help of a database server, it gives the output to the user according to the request. When the database server receives the parameter, it compares the user supplied parameter to the actual database according to the query and then it gives output. Usually, attackers inject their own queries along with the user input parameter with the help of some SQL logical operators and a semi colon. By doing so they can easily trick the database server to take their queries as a valid request.

SQL Injection vulnerabilities are always risky. According to the OWASP (63), the loss of confidentiality and integrity of the database, and bypassing the authentication and changing authorization information are the main consequences of SQL Injection vulnerabilities. Based on users’ privileges on the database, attackers can perform any of the CRUD (Create, Read, Update, and Delete) operation on the database. Example risks include that they can steal secret information from the database, create new fields or update fields with false data, or they can even delete fields, tables, or the whole database. These risks are associated with integrity and confidentiality and integrity of the database. “If poor SQL commands are used to check user names and passwords, it may be possible to connect to a system as another user with no previous knowledge of the password” (63). By bypassing the authentication, they can gain unauthorized access to the resources of the application. If authorization information is placed on the database, attackers can alter or change the information through the SQL Injection.

Manifestation in the sample application

The testing report shows that there are a total of 24 instances of SQL – MySQL vulnerabilities in the application. I studied those vulnerable files manually and found that most of them are files which are used to display the requested information from the database dynamically by performing the READ operation on the database. One file is associated with the WRITE operation on the database. The following table contains some example vulnerable files reported by ZAP.

TABLE 7. Instances of SQL Injection

URL	http://localhost/GradingSystem/Student/class_teachers.php?choiceclass=1
Parameter	Choiceclass
Attack	1 UNION ALL select NULL --
Evidence	The used SELECT statements have a different number of columns
URL	http://localhost/GradingSystem/Student/courses_list.php?class=1
Parameter	Class
Attack	1 UNION ALL select NULL --
Evidence	The used SELECT statements have a different number of columns
URL	http://localhost/GradingSystem/Student/teacher_list.php?course=cou001%27+AND+%271%27%3B
Parameter	Course
Attack	cou001' AND '1';
Evidence	SQLSTATE[HY
URL	http://localhost/GradingSystem/student/feed_comm.php
Parameter	courseID
Attack	cou004' / sleep(5) / '
URL	http://localhost/GradingSystem/Teacher/studentList2.php?courseID=cou003
Parameter	courseID
Attack	cou003' / sleep(5) / '
URL	http://localhost/GradingSystem/Teacher/student_grade.php?choicestudent=stud_00001%27+AND+%271%27%3D%271%27+--+
Parameter	Choicestudent
Attack	stud_00001' AND '1'='1' --

A database functions as a backbone of the web application as it stores all kind of information from users' personal and bank information to information required to run the application. It includes, but is not limited to, e.g. different kinds of cookies, usernames, passwords, personal records and bank records. This information is necessary to run the application and to provide services to users. That might be the reason that the SQL Injection is one of the most prevalent attack on web applications. The risk associated with the SLQ injection is always severe. The impact of an attack also depends upon given privileges into the database. Based on this fact, the sample application is extremely risky as it has provided all kinds of privileges to the database for all kinds of users.

The next notable thing in the application is that in many instances no Parameterized Queries have been used as it is essential to avoid the SQL Injection if the application dynamically performs database functions. Another general precaution includes not to directly use the data that is received from the user without escaping and validating it on the server which has not been followed in the application.

Because of all the above mentioned reasons, the sample application is extremely vulnerable for an SQL Injection attack. The next section will provide a brief summary of how to prevent an SQL Injection.

Fixing the Problems

To prevent an SQL Injection attack, OWASP recommends few simple steps to follow. The first thing is the user privilege on the database. It should always be as little as possible. In most of the cases, users do not actually need the privileges they are given. For easiness or simply a lack of awareness, developers generally do not mind providing equal privileges for all users. For the real application, developers should develop a mechanism where database privilege is based on the need of specific users.

The next thing is an error displaying mechanism which provides certain information to attackers. They can utilize that information for further attacking. Although displaying an error with certain information is worthy for developers while developing the application, it should be avoided when the application goes live.

Another worth noting thing is user input mechanism in the web application. User inputs should always be validated before using it on the database functions. This applies also in the case of XSS. Validation should be done in the server. Client validation should never be trusted. There are many ways of validating user input data, which depends upon the logic of the input. For example, if the input field is for numbers, then inputs other than numbers should be discarded. HTML escaping is also a kind of validation. It removes html tags from the user input.

The most crucial thing is to always use Parameterized Queries while using user supplied data to perform database functions. In the PHP web application, it is done with the help of PDO and prepare statements. The following example shows how a Parameterized Query is used to verify a user while logging into the system.

```
// Define $username and $password
$user=$_POST['username'];
$pass=$_POST['password'];

// To protect MySQL injection for Security purpose
$user = stripslashes($user);
$pass = stripslashes($pass);

// Include Database connection file
require 'databaseConnection.php';

// SQL query to fetch information of registerd users and finds user match.
$sql = "SELECT * FROM user WHERE user_id=? AND pass_word=?";
$stmt = $conn->prepare($sql);
$stmt->execute(array($user, $pass));
```

FIGURE 20. Example of Parameterized Query

In the above example, the values of $user and $pass variables are coming from the user input. But instead of using that user supplied data directly into the database query, a prepare statement has been created with an incomplete SQL Query. Later, the query is executed by passing that user input data as an array. This is the safe way of passing the user input data to a query.

7 CONCLUSION

The main objective of this thesis work was to study the effectiveness of OWASP ZAP for finding vulnerabilities in a web application. For that purpose, a simple web application was developed and its vulnerabilities were tested using ZAP. It successfully found different kinds of vulnerabilities in the sample application. It is worth to mention that it not only finds the vulnerabilities but also suggests how to prevent such vulnerabilities.

The thesis helped me learn quite many things about a web application and different aspects of security issues associated with it. For example, what are the current prevalent security threats in the market, what impact can a particular threat have on a web application, and how to tackle those threats to make web applications secure. Along with security issues, I also got a good knowledge about designing and using MySQL database in a web application.

To use ZAP was totally new thing for me. When I started learning to use ZAP in the sample application, it was hard to understand. It has a wide scope for testing web applications. The only means to learn was watching videos from youtube. But learning is a process of acquiring knowledge, and it is not always effective by only studying the content but also by applying it to real use. With the regular guidance of my teacher, I think I have gained knowledge to some extent, although there are lots of things I still need to learn regarding a complete use of ZAP.

Although I am fully satisfied with ZAP results in testing the sample application, I think the application is not complex enough to test the effectiveness of ZAP. It has many tools for advance testing. To use all kind of testing that ZAP provides and see its effectiveness, I feel that the application should be bigger and more complex one than the sample application. So in my opinion, there is an ample opportunity to gain expertise in the field of penetration testing of web applications using ZAP. In conclusion, as I already mentioned before, ZAP is effective in finding vulnerabilities in the sample application but more than that it has successfully inspired me towards further testing of web applications and developing a secure database driven web application.

REFERENCES

1. Ogden, J. 2015. File:Internet users per 100 inhabitants ITU.svg (CC BY-SA 3.0), Date of retrieval 21.04.2016 https://commons.wikimedia.org/w/index.php?curid=18972898

2. Fonseca, J., Vieira, M. & Madeira H. 2007. Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks. 13th IEEE International Symposium on Pacific Rim Dependable Computing, 365 – 372.

3. Acunetix. 2016. Web Applications: What are They? What of Them?. Date of retrieval 16.03.2016 http://www.acunetix.com/websitesecurity/web-applications/

4. Rochkind, Marc J. 2004. Advanced Unix Programming, Second Edition. Addison-Wesley. p. 1.1.2.

5. Wikipedia. 2016. HTML. Date of retrieval 30.04.2016 https://en.wikipedia.org/wiki/HTML

6. Computer Hope 2016. HTML. Date of retrieval 21.04.2016 http://www.computerhope.com/jargon/h/html.htm

7. MDN Mozilla Developer Network. 2016a. CSS. Date of retrieval 11.05.2016 https://developer.mozilla.org/en-US/docs/Web/CSS

8. w3schools.com. 2016. CSS How To... Date of retrieval 30.04.2016 http://www.w3schools.com/css/css_howto.asp

9. MDN Mozilla Developer Network. 2016b. JavaScript. Date of retrieval 11.05.2016 https://developer.mozilla.org/en-US/docs/Web/JavaScript

10. Wikipedia. 2016. JavaScript. Date of retrieval 30.04.2016 https://en.wikipedia.org/wiki/JavaScript

11. php.net. 2016. What is PHP?. Date of retrieval 16.03.2016 http://php.net/manual/en/intro-whatis.php

12. Wikipedia. 2016. PHP. Date of retrieval 05.05.2016 https://en.wikipedia.org/wiki/PHP

13. Wikipedia. 2016. Database. Date of retrieval 30.04.2016 https://en.wikipedia.org/wiki/Database

14. Oracle. 2016. MySQL. Date of retrieval 05.05.2016 http://www.oracle.com/us/products/mysql/overview/index.html

15. Wikipedia. 2016. Vulnerability (computing). Date of retrieval 16.03.2016 https://en.wikipedia.org/wiki/Vulnerability_(computing)

16. Lepofsky, R. 2014. The Manager's Guide to Web Application Security: A Concise Guide to the Weaker Side of the Web. New York: Apress. p. 21.

17. OWASP. 2015. OWASP Top Ten Project. Date of retrieval 16.03.2016 https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

18. OWASP. 2013. OWASP Top 10 – 2013: The Ten Most Critical Web Application Security Risks. Publication of OWASP 2013:6.

19. University of Pennsylvania, Information Systems and Computing. 2016. Top 10 Web Application Security Vulnerabilities: A6: Injection Flaws (Shell Commands and SQL). Date of retrieval 30.04.2016 http://www.upenn.edu/computing/security/swat/SWAT_Top_Ten_A6.php

20. NIST. 2013. Security and Privacy Controls for Federal Information Systems and Organizations. NIST Special Publication 800-53, B2.

21. OWASP. 2016. Session Management Cheat Sheet. Date of retrieval 01.05.2016 https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

22. Acunetix. 2016. Cross-site Scripting (XSS) Attack. Date of retrieval 01.05.2016
http://www.acunetix.com/websitesecurity/cross-site-scripting/

23. CodeDX. 2016. Insecure Direct Object References. Date of retrieval 05.05.2015
http://codedx.com/insecure-direct-object-references/

24. OWASP. 2007. Top 10 2007-Insecure Direct Object Reference. Date of retrieval 05.05.2016
https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference

25. tutorialpoints. 2016. Security Testing - Insecure Direct Object References. Date of retrieval 05.05.2016
http://www.tutorialspoint.com/security_testing/insecure_direct_object_reference.htm

26. OWASP. 2013. Top 10 2013-A5-Security Misconfiguration. Date of retrieval 01.05.2016
https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration

27. GitHub. 2016. Sensitive Data Exposure. Date of retrieval 01.05.2016
https://bounty.github.com/classifications/sensitive-data-exposure.html

28. OWASP. 2013. Top 10 2013-A6-Sensitive Data Exposure. Date of retrieval 01.05.2016
https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure

29. McMullin, M. 2015. OWASP Top Ten Series: Missing Function Level Access Control. Date of retrieval 06.05.2016
http://kemptechnologies.com/blog/owasp-top-ten-series-missing-function-level-access-control/

30. GitHub. 2016. Cross-Site Request Forgery (CSRF). Date of retrieval 01.05.2016 https://bounty.github.com/classifications/cross-site-request-forgery.html

31. GitHub. 2016. Using Components with Known Vulnerabilities. Date of retrieval 01.05.2016 https://bounty.github.com/classifications/using-components-with-known-vulnerabilities.html

32. GitHub. 2016. Unvalidated Redirect or Forward. Date of retrieval 01.05.2016 https://bounty.github.com/classifications/unvalidated-redirect-or-forward.html

33. Gaskill, C. 2014. Top 10 Web Security Risks: Unvalidated Redirects and Forwards (#10). Date of retrieval 07.05.2016 https://www.credera.com/blog/technology-insights/java/top-10-web-security-risks-unvalidated-redirects-forwards-10/

34. OWASP. 2013. Top 10 2013-A10-Unvalidated Redirects and Forwards. Date of retrieval 01.05.2016 https://www.owasp.org/index.php/Top_10_2013-A10-Unvalidated_Redirects_and_Forwards

35. Kevin M. Henry. 2012. Penetration Testing: Protecting Networks and Systems. IT Governance Ltd. ISBN 978-1-849-28371-7.

36. Rouse, M. 2011. pen test (penetration testing). Date of retrieval 21.04.2016 http://searchsoftwarequality.techtarget.com/definition/penetration-testing

37. Core Security. 2015. Penetration Testing Overview. Date of retrieval 01.05.2016 https://www.coresecurity.com/penetration-testing-overview

38. Wai, C.T. 2002. Conducting a Penetration Test on an Organization. SANS Institute InfoSec Reading Room, 3 – 9. Date of retrieval 02.05.2016 https://www.sans.org/reading-room/whitepapers/auditing/conducting-penetration-test-organization-67

39. OWASP. 2016. About The Open Web Application Security Project. Date of retrieval 08.05.2016 https://www.owasp.org/index.php/About_OWASP

40. zaproxy/zap-core-help. 2015. OWASP ZAP User Guide. Date of retrieval 08.05.2016 https://github.com/zaproxy/zap-core-help/wiki/HelpIntro

41. psiinon. 2012. OWASP Zed Attack Proxy - official tutorial: Overview. Date of retrieval 04.03.2016 https://www.youtube.com/watch?v=eH0RBI0nmww&list=PLEBitBW-Hlsv8cEIUntAO8st2UGhmrjUB

42. Wikipedia. 2016. WebSocket. Date of retrieval 08.05.2016 https://en.wikipedia.org/wiki/WebSocket

43. zaproxy / zap-core-help. 2015. WebSocket. Date of retrieval 08.05.2016 https://github.com/zaproxy/zap-core-help/wiki/HelpAddonsWebsocketIntroduction

44. Hackingheart. 2012. Forced Browsing Attack. Date of retrieval 08.05.2016 https://hackingheart.wordpress.com/2012/07/03/forced-browsing-attack/

45. OWASP. 2009. Forced browsing. Date of retrieval 08.05.2016 https://www.owasp.org/index.php/Forced_browsing

46. zaproxy / zap-core-help. 2015. Forced Browse. Date of retrieval 08.05.2016 https://github.com/zaproxy/zap-core-help/wiki/HelpAddonsBruteforceConcepts

47. OWASP. 2015. Category:OWASP DirBuster Project. Date of retrieval 08.05.2016 https://www.owasp.org/index.php/Category:OWASP_DirBuster_Project

48. zaproxy / zap-core-help. 2015. Fuzzing. Date of retrieval 08.05.2016 https://github.com/zaproxy/zap-core-help/wiki/HelpAddonsFuzzConcepts

49. psiinon. 2013. FOSDEM 2013: Practical Security for developers using OWASP ZAP. Date of retrieval 08.05.2016 https://www.youtube.com/watch?v=QG2RCZHMEkM&list=PLEBitBW-Hlsv8cEIUntAO8st2UGhmrjUB&index=5

50. zaproxy / zaproxy. 2015. The ZAP API. Date of retrieval 08.05.2016 https://github.com/zaproxy/zaproxy/wiki/ApiDetails

51. zaproxy / zap-core-help. 2015. Anti CSRF Tokens. Date of retrieval 21.01.2016 https://github.com/zaproxy/zap-core-help/wiki/HelpStartConceptsAnticsrf

52. Paul, B. 2013. Oh, Wasp! Security Essentials for Web Apps. Date of retrieval 08.04.2016 http://www.slideshare.net/TechWellPresentations/bw8-paul

53. Shiflett, C. 2003. Foiling Cross-Site Attacks. Date of retrieval 10.04.2016 http://shiflett.org/articles/foiling-cross-site-attacks

54. OWASP. 2015. Cross-Site Request Forgery (CSRF). Date of retrieval 09.05.2016 https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

55. Open Hub. 2016. OWASP CSRFGuard. Date of retrieval 09.05.2016 https://www.openhub.net/p/OWASP-CSRFGuard

56. Shanmuga Priya, S. 2014. Cross-Site Scripting. Date of retrieval 12.04.2016 http://www.pitsolutions.ch/blog/cross-site-scripting

57. Kallin, J & Valbuena, Irene L. 2013. Excess XSS: A comprehensive tutorial on cross-site scripting. Date of retrieval 15.04.2016 http://excess-xss.com/

58. MSDN. 2015. Security: Anti-Cross Site Scripting Library. Date of retrieval 09.05.2016 https://msdn.microsoft.com/en-us/security/aa973814.aspx

59. Williams, L. 2012. Input Validation Vulnerabilities. Computer Science, NC State University. Date of retrieval 09.05.2016 http://agile.csc.ncsu.edu/SE-Materials/3_InputValidation.pdf

60. Xu, J. 2016. Cross-Site Scripting Counter-Measures. Date of retrieval 09.05.2016 https://sites.google.com/site/jimmyxu101/design/cross-site-scripting-counter-measures

61. OWASP. 2016. XSS (Cross Site Scripting) Prevention Cheat Sheet. Date of retrieval 09.05.2016 https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

62. Acunetix. 2016. SQL Injection (SQLi). Date of retrieval 22.04.2016 http://www.acunetix.com/websitesecurity/sql-injection/

63. OWASP. 2016. SQL Injection. Date of retrieval 25.04.2016 https://www.owasp.org/index.php/SQL_injection