Aalto University
School of Science
Degree Programme in Computer, Communication and Information Sciences

Can Zhu

Experimental study of vulnerabilities in a web application

Master's Thesis
Espoo, February 18, 2017

Supervisor: Professor Tuomas Aura, Aalto University

Aalto University
School of Science
Degree Programme in Computer, Communication and Information Sciences

Aalto University
School of Science

ABSTRACT OF
MASTER'S THESIS

Author:	Can Zhu	
Title:	Experimental study of vulnerabilities in a web application	
Date:	February 18, 2017	**Pages:** vi + 44
Major:	Mobile computing, services and security	**Code:** T-110
Supervisor:	Professor Tuomas Aura	

As web services have become business critical components, it is very vital to improve their security. Many businesses define penetration testing as the web vulnerabilities scanners automatically operate the site, however, the true penetration testing is more than that. It needs sophistic skills and experience of the testers.

Web vulnerability scanners can detect weaknesses in a black-box way, and they are easy to use. There are various scanners to choose; organizations should select them based on their requirements and conditions. In this thesis, we study vulnerabilities in one web application named Virtual Environment Manager (VEM) of Tieto company. After scanning VEM with two scanners, 11 types of vulnerabilities are detected. Then, we exploit every vulnerability based on the application's source code, and also evaluate their severity levels. Finally, the solutions of remedying these vulnerabilities are provided. Because of some limitations, the security testing of the VEM is not fully implemented. For example, the cloud infrastructure is not detected. Still, this experiment contributes to security testing of VEM web application. We hope that this project can help Tieto company improve the security level of VEM.

Keywords:	web application, security testing, vulnerability, exploit
Language:	English

Acknowledgements

I would like to thank Professor Tuomas Aura for his excellent instruction and guidance. I also would like to thank the Security Services group of Tieto for offering me this project opportunity. Their commitment and understanding enable I selecting the thesis topic and gaining guidance in vulnerability testing and exploitation.

Thank you!

Espoo, February 18, 2017

Can Zhu

Abbreviations and Acronyms

VEM	Virtual Environment Manager
ZAP	Zed Attack Proxy
HTML	HyperText Markup Language
SGML	Standard Generalized Markup Language
HTTPS	Hypertext Transfer Protocol Secure
MIME	Multipurpose Internet Mail Extensions
IaaS	Infrastructure as a Service
CDN	Content Delivery Network

Contents

Chapter 1

Introduction

Security testing is important in today's software development. The thesis studies vulnerabilities of a web application from a Finland Company. The author used web scanners to implement security testing during the experiment. 11 kinds of vulnerabilities were detected after scanning, then, each specific type is analyzed and evaluated in detail. The risk-based approaches were used when analyzing the web weaknesses: considering both the system architectural implementation and the attackers' mindset. The results show which types of vulnerabilities are most serious; which types are less serious; which types are the false positive. The methodology of vulnerability exploit is presented in the thesis as well. What is more, this thesis provides the solutions of these vulnerabilities. We hope the work can help Tieto improve VEM's security level, and also help testers with their exploitation.

1.1 Problem statement

The World Wide Web has changed our world, and it is the focus of 21st economy. At the same time, numerous web sites have suffered from various risks and attacks. In the history, sites belonging to Yahoo, CNN, Amazon, and others were attacked and had to be shut down for hours. The website's owners cost much for this action, for instance, Yahoo claimed it lost more than a million dollars per minute in advertising revenue during the attack. [12] What is more, an article published in ZD-Net claims that 30% to 40% of e-commerce sites throughout the Internet are vulnerable to one simple attack: the attacks only practice saving HTML page to disk, modifying the price and reloading the page in a browser to decrease the goods price. Attacks on the Internet are not only limited to e-commerce sites. A significant number of high-profile websites had to rewrite their pages due to attacks, which include the U.S. Department of Justice, the U.S. Air Force, UNICEF, and the New York Times. [12] Since Web services are often deployed with software bugs that can be maliciously exploited, security testing is critical during software development process.

Although it is impossible to achieve zero vulnerabilities in web applications, the number can be decreased through security testing. White-box and black-box methods are both used in software security testing, and they use different approaches depending on whether the tester has access to source code. The white-box method is very useful in finding programming errors; however, it sometimes reports weaknesses that do not exist. Black-box testing can run a program without knowing the source code, while the effectiveness is lower than

white-box testing. Since both approaches are with advantages and drawbacks, it is wise to combine them in security testing practice.

A simple way to test applications against security weaknesses is to use web vulnerability scanners. They provide an automatic way to detect various types of vulnerabilities avoiding the repetitive and tedious task of doing hundreds or even thousands of manual tests. Most of these scanners are designed for commercial use (e.g., IBM Rational AppScan and AppSpider), while there are also some free application scanners (e.g., Zed Attack Proxy, W3af and Vega). They offer remediation advice and generate vulnerability reports, based on which testers can improve the application security level. Researchers have shown that the effectiveness of scanners varies a lot during vulnerability detection. [35] This thesis focuses on two web scanners, the Zed Attack Proxy and Burp suite, and analyzes their scan result in detail. By scanning a real commercial web application VEM of Tieto company, verifying these vulnerabilities, classifying the severity, reporting and proving solutions, one completed web security test process is presented. Besides, another goal of this thesis is to report the results and improve the experimental website security.

1.2 Contribution

Testers should use risk-based approaches, considering both the system's architectural reality and the attacker's mindset, to enhance software security. After identifying risks in a system, the tester can create test cases by focusing on areas of code where an attack may succeed. This approach provides a higher level of software security than black-box testing. Because security testing involves two approaches, the testers need more expertise and experience than we think. Standard testing organizations use the traditional approach to perform functional security testing; however, they have more difficulty in performing risk-based security test. Security testers may have difficulties to think like an attacker because of lacking expertise, and they usually do not develop direct security exploits.

In our project, we tried to think from both the application's architecture and the attacker's perspective to security test VEM. First, we could access the application source code as a developer. Second, we implemented exploits to the detected vulnerabilities.

1.3 Structure of the Thesis

The structure of the thesis is as follows: Chapter 2 gives the background of the thesis, then chapter 3 presents vulnerability testing tools and methodology. Next, the case study of the experiment on Tieto application is shown in chapter 4. We analyze the scanning reports and give practical remediation advice in chapter 5. The evaluation and measurement of the whole work are presented

in chapter 6. Finally, chapter 7 concludes and summarizes the thesis.

Chapter 2

Web application security vulnerabilities testing

This chapter gives an overview of the web applications and vulnerabilities in them. The main idea is how the applications have been developed, and how is the situation of vulnerabilities.

2.1 Web application

We describe how the web application is developed in this section. In the beginning, there are some other ways to transmit information such as FTP protocol. However, the World Wide Web is outstanding because of its excellent design. Also, the section will discuss the basic ideas of web application testing.

2.1.1 History and architecture

The success of the Internet and the Web should be attributed to a combination of market need, determinism, and consumerism. [12] Due to powerful desktop computers, reasonably fast modems, sophisticated computer users, the Web became wildly popular in the mid-1990s and gained mass increase. It is also to claim that the web was pushed by companies including IBM, Cisco, etc. They made lots of efforts to convince business leaders to became involved in online business models. After that, the success of a large number of Internet startups such as Amazon, Yahoo, and VeriSign promoted the web to create an online business climate in the world. However, the Internet was just one of many large-scale computer networks deployed in the 1970s, 80s, and 90s, and it was never considered "the most likely to succeed network". For example, IBM and HP spent hundreds of millions of dollars developing a network product called Open System Interconnection (OSI); OSI was mandated by the U.S. government, which regarded the Internet and TCP/IP as a transitional step. [12] The writer of book "Web Security, Privacy & Commerce" contributes the success of Internet and Web largely to their design, which was technically superior to its competitors, open to developers and easy for customers to use.

To provide physicists with a convenient way to publish their papers on the Internet, the World Wide Web was invented in 1990 by Tim Berners-Lee. Though there were other ways of transmitting information over the Internet and accessing remote databases at that time, for example, the FTP protocol, the World Wide Web is outstanding. It had two genius ideas. One is that the World

Wide Web has a single addressing scheme, which is critical for successful large scale communication systems. Considering the telephone system, it includes the country code, city code, and then phone number. Another example is the postal system; it finds the target based on the person's name, street address, city, state and postal code. Instead, the addressing solution in the World Wide Wed is called URL, composed of a protocol (HTTP), the host name, followed by the resource name. Another great insight of Berners-Lee was that he realized that the Internet addresses could be embedded within other resources [8]. Thus, one document on the Internet could contain links to other documents. Guided by this idea, Hyper Text Markup Language (HTML) was also invented, borrowing its syntax from an earlier markup language, SGML. What is more, Berners-Lee created two software programs: a web server and a browser, to prove his ideas. After that, to make this technology commercial, browsers named Mozilla, Netscape Navigator were pushed in the business market.

The modern Web architecture addresses scalability among component interactions, independence of components, generality of interfaces, communication latency, security, and encapsulating legacy systems. With these characteristics, it is one instance of an REST-style architecture. REST is a type of structure that attempts to minimize latency in network communication while maximizing the scalability and independence of component implementations. [11] Through placing constraints on connector semantics, REST achieved its goals. It enables reuse and caching of communications, dynamic substitutability of components, and processing of actions by intermediaries to become an Internet-scale distributed hypermedia system. The primary focus of Web-based application protocols and performance concerns is distributed hypermedia, even though it allows other styles of interaction.

2.1.2 Development and test

The vast spread of Internet has produced a significant growth in the demand for web applications, which are required to have properties of reliability, usability, and security. Due to these strict requirements from market pressure as well as short time to market, the web application testing has become a challenging work in the software development process. Usually, a web application with a client-server or multi-tier architecture mainly includes the following three characteristics:

- A significant number of users distributed all over the world who can access it concurrently.
- Heterogeneous execution environments consist of multiple types of hardware and software (e.g., many different operating systems, web servers, web browsers and network connections).
- Heterogeneous nature that depends on the various of software components it usually includes [8]. For example, these components can be constructed

with different programming languages and models.

Each of the list described above produces a new testing challenge and perspective. For instance, practical solutions for verifying the web application behavior when a large number of users access it at the same time should be looked for. Another critical feature of a web application is its security and ability to prevent unauthorized accesses. Moreover, the different technologies used to implement the web applications will increase the cost of creating the testing environment and the testing complexity. Since the main aim of the testing of a web application is to discover failures in the required services and functionality, the testing can be classified as functional and non-functional testing.

The responsibility of functional testing is to find the failures of applications when implementing specified technical requirements. One point that needs to be paid attention to is that testers should avoid observing failures are due to the running environment, or reduce them to the minimum [8], because the running environment mainly affects the non-functional requirements of applications. When referring to non-functional testing of a web application, it is required to satisfy requirements including portability, performance, scalability, compatibility, accessibility, usability, and security.

The aim of security testing is to guarantee the overall Web system defenses against access by unauthorized users, to prevent system resources from going to improper uses, and to grant the access of authorized users to authorized resources and services. Testers must use risk-based approaches, considering both the system's architectural implementation and the attacker's mindset, to test Web application adequately [30]. System security vulnerabilities may be contained in the application code, in different hardware, software, and middleware components of the system. Both the application and its running environment can trigger the security problems. In a Web application, various implementation and execution technologies, as well as a very vast number of users, and the possibility of accessing from anywhere make the Web application more vulnerable than traditional ones, and security testing is also harder to perform [8]. The focus of this thesis is on web application security testing. Security vulnerability analysis tools, methodology, and the process will be discussed in detail. Moreover, there will be a vulnerability testing experiment of one practical software from Tieto company presented in chapter 4.

2.2 Web Vulnerabilities

We will give an overview of web vulnerabilities: the types and the growth of them.

2.2.1 Definition

A vulnerability is a weakness or a hole in an application, that allows attacks to cause harm to the owner of the application. There are some examples of

vulnerabilities:

- lack of input validation on user input
- lack of sufficient logging mechanism
- fail-open error handling
- not closing connections properly

Typically, vulnerabilities fall into two categories: bugs at the implementation level and flaws in the design level [20]. When hackers attack the application, they do not care if a vulnerability is a flaw or a bug. As attackers are becoming more sophisticated, and the notion of which types of vulnerabilities matter is changing. For instance, a race condition was considered exotic a few years ago, although they are common today. Similarly, two-stage buffer-overflow attacks were the domain of software scientists, while now they appear in zero-day exploits. [30]

2.2.2 Distribution of various vulnerabilities

The German company SAP, the market leader in enterprise application software, published a global survey in 2013. It described vulnerabilities in their products sorted by their frequency, criticality and the affected system from 2001 to 2013 [33]. Every month on SAP Critical Patch Day, one or more SAP Security Notes were released. In figure 2.1, the number of the notes was rarely small from 2001 to 2007. In 2007, the number of published notes began to

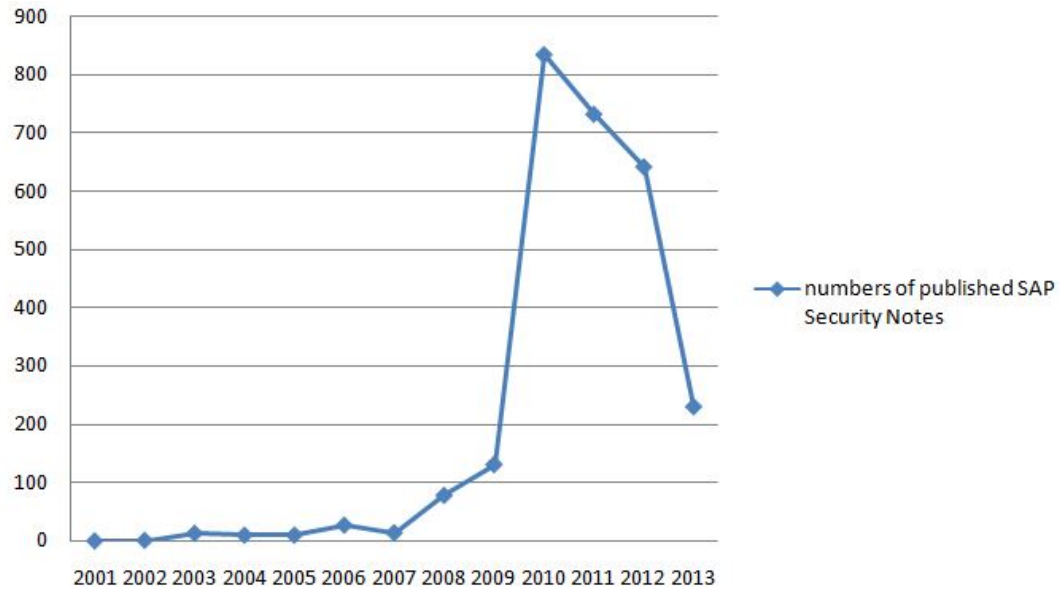

Figure 2.1: Published SAP Security Notes from 2001 to 2013

grow exponentially. The figure also shows that the number of security notes had decreased a little since the peak in 2010; however, the number is still huge. Moreover, most of the security notes (69%) were considered to have high priority when sorted by criticality according to the statistical survey. This means that about 2/3 of the published vulnerabilities must be remedied as soon as possible. SAP Security Notes were also analyzed by their frequency, and the most common types of vulnerabilities presented in figure 2.2. The survey of SAP is representative in business application issues, while the OWASP Top 10 2013 covered all web-based applications. OWASP is a non-profit organization that provides unbiased, practical, cost-effective information about application security. The comparison between these two reports is presented in table 2.1, and the situation here changed slightly. After considering the data, it is reasonable to gain four conclusions [33]:

1. Growing number of discovered XSS vulnerabilities is predictable due to the growing popularity of web-based applications, and also because of the higher level of static code analysis.

2. The number of directory traversal issues would fall because they were easy to find, and most of them had already been found before.

3. The number of code injection vulnerabilities would grow because of the high criticality and the fact that any injection flaws would be easier to find with the improvement of static code analysis tools.

4. Moreover, such issues as hardcoded credentials would be harder to find with every year precisely because they are very easy to find.

Vulnerability type	Ranking in SAP in mid 2013	Growth by percent (from mid 2011 to mid 2013)	Place in OWASP TOP 10
XSS	1	53%	3
Missing authorization check	2	28%	7
Directory traversal	3	10%	4
SQL Injection	4	5%	1
Information disclosure	5	36%	6
Code injection	6	57%	1
Authentication bypass	7	18%	2
Hardcoded credentials	8	17%	2
Remote code execution	9	13%	1
Verb tampering	10	11%	7

Table 2.1: Differences between SAP top 10 security notes and OWASP Top 10

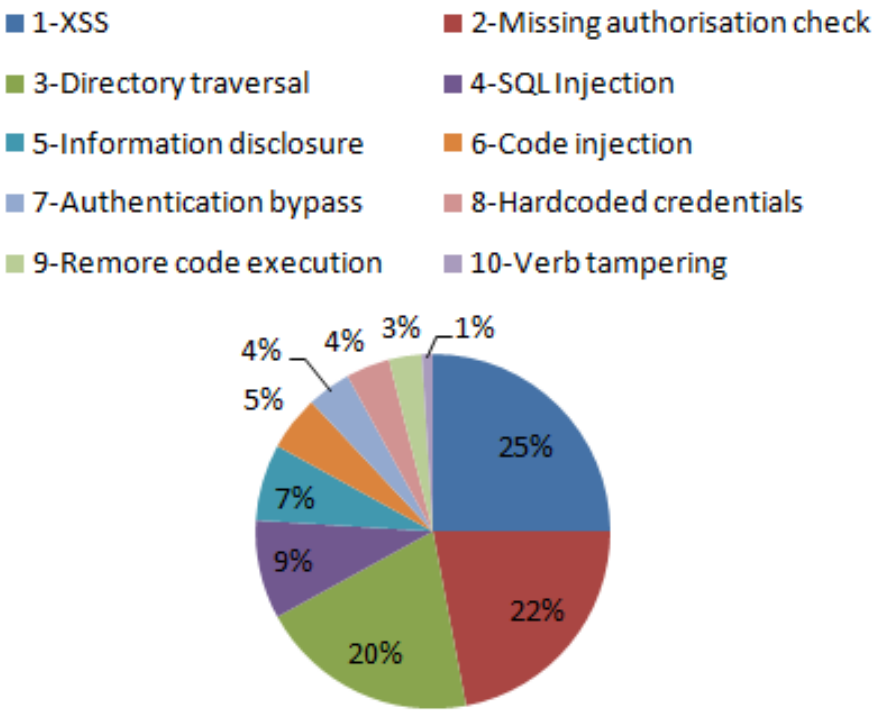

Figure 2.2: 10 types of vulnerabilities

Except for the listed vulnerabilities, there are hundreds of issues that could affect the overall security of a web application [21]. OWASP suggests web developers to read its security guides [22, 23] before writing the code. It also provides related guidance [24] on testing web vulnerabilities effectively. As the Top 10 may change regularly, and other new vulnerabilities may come instead of the current common ones, application developers should pay more attention to their design and implementation phases. They should always keep following the latest security information, and wisely use tools to ensure their application works in a secure way.

Chapter 3

Vulnerability testing tools and methods

The attackers can attack one application through various paths. We describe how to evaluate the risk of a particular vulnerability in this chapter, and give the methodology for security testing.

3.1 Usage scenario

There are many different paths in an application that can be used by attackers to do harm to a business or organization. As figure 3.1 shows, each of these paths represents a risk that may be severe enough to warrant attention. These paths are difficult to find and exploit, sometimes, while they may be trivial to find sometimes. Similarly, the harm caused by these paths may ruin the business or have no consequences at all. Thus, testers should evaluate the severity of consequences with each threat agent, attack vector, security weakness and combine it with an estimate of the technical and business impact of the organization [21]. After all, all of them together determine the overall risk.

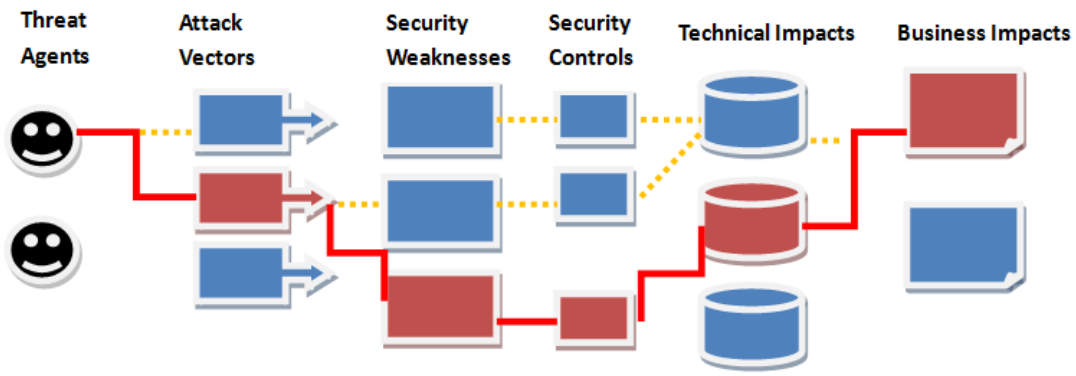

Figure 3.1: Different paths to harm the businesses in an application

Depending on whether the tester has access to the source code, the software test method can be classified as white and black-box testing. It is very effective to find the programming errors when involving white-box testing, due to the requirement of understanding the code and design. For example, risk analysis is a white-box approach depending on a good understanding of the system. On the other hand, black-box testing refers to analyzing the application by probing it with various inputs. This approach is relatively straightforward without

accessing the code. One case of black-box testing is the use of web scanners to find the application vulnerabilities. In some cases, these two approaches are combined: using the black-box testing to probe the weaknesses; then the white-box one is used to navigate the source code to verify or remedy the application. We use the combination of white and black-box testing in our experiment in this thesis.

Black-box web application vulnerability scanners are automated tools to probe the Web application for vulnerabilities. Except that, they also offer remediation advice and generate compliance reports. There are intrinsic limitations to these scanners, and they have the false positives to some extent. For instance, scanners may state the vulnerabilities that do not exist in one application. However, they provide a cost-effective method to probe some significant weaknesses in a request by mimicking the attackers and may configure and test defenses such as web application firewalls [5]. A large number of web scanners exist. Some of them are produced to make a profit by business owners (e.g., Rational AppScan of IBM, AppSpider), while some of them are open source for non-profit use (e.g., Vege, WebInspect of HP).

3.2 Methodology

Our study steps and method are outlined now. Firstly, the principles of vulnerability scanners are explained. Then, we describe how we set the web vulnerability scanners to maximize the application scanning coverage. Finally, we propose how to verify the severity of the vulnerabilities.

3.2.1 Principles of the web scanner

A Web application security scanner is an automated program that examines web applications for potential security vulnerabilities. The basic functional requirements for scanners used in evaluations of application layer software on the web are listed below [6]:

- Define a minimum level of functionality for the purchaser and vendor to qualify the product.
- Produce unambiguous clauses as to what is required to conform.
- Build consensus on tool functions and requirements toward the evaluation of software security assessment tools for behavior and effectiveness.

The web scanners have capabilities far beyond those. Some scanners can scan other artifacts such as requirements documents, bytecode or binary code; some scanners are system security tools, e.g., firewalls, anti-virus software, gateways, routers; some can check infrastructure vulnerabilities, deployment, or configuration issues, such as running an obsolete version of a web server. In

our experiment, we use the web scanners that can set characteristics such as protocols and methods; they need authentication as well.

3.2.2 Web scanners setting

The URL of the web applications needs to be entered as well as an individual user's login credentials to start a scan session in a scanner. Then, the tester must specify options for the scanner page crawler to maximize page scanning coverage. Through setting the "crawl-only" mode, we can verify the provided login and whether the crawler options work well as expected. After setting the crawler, the scanning profile, or test vector set was specified by the tester, which was used in the vulnerability detection run. This step happened before launching the scan. Finally, it was time to start scan process. Every scanner can proceed automatically with the scanning profile selection and most allows interactive modes in which the tester may direct the scanner to scan each page [5]. In the study of this thesis, the scanners were always set to run with automatic mode, to maximize the vulnerability detection capability under the most comprehensive available test setting. Moreover, we improve the detection results by scanning the target application multiple times because some of the tests are randomized.

3.2.3 Vulnerability evaluation and verification

The standard risk model referencing from OWASP [25] can be listed as follows:

$$Risk = Likelihood * Impact \tag{3.1}$$

The first step is to estimate the "likelihood" to determine how serious the risk is. This is a rough measure of how likely each vulnerability is to be exploited by attackers. The final goal is to identify if the likelihood is low, medium, or high. The factors and their points that are helpful in determining the risk are listed:

- Ease of discovery (1,3,7,9): is it easy for threat agents to discover the vulnerability?
- Ease of exploiting (1,3,5,9): how easy for attackers actually to exploit this vulnerability?
- Awareness (1,4,6,9): how familiar attackers with this vulnerability, and is it known to the public or a new one?
- Skill level (1,3,5,6,9): the security penetration skill, network and programming skill, and no technical skill of threat agents.
- Motivation (1,4,9): how motivated is the attacker to find and exploit this vulnerability?

- Opportunity (0,4,7,9): how easy the attack can access the application, are they internal users, developers?
- Threat agent size (2,4,5,6,9): how large is the threat agents? Developers, users, system administrators, and so on.

When considering the impact of an attack, both "technical impact" and "business impact" work. The "technical impact" involves data, and functions provided by the application. The "business impact" is more important because it may directly influence the profit or fame of a company. Technical impact factors include:

- Loss of confidentiality (2,6,7,9): how much data would be disclosed and how sensitive is it?
- Loss of integrity (1,3,5,7,9): how much data would be disrupted and the damage of it?
- Loss of availability (1,5,7,9): how much service would be lost and is it vital?
- Loss of accountability (1,7,9): Are the attackers traceable?

The business impact factors list below:

- Financial damage (1,3,7,9): how much damage from an exploit and the cost to fix it?
- Reputation damage (1,4,5,9): could the exploit harm the reputation of the company?
- Privacy violation (3,5,7,9): how much individual private information would be disclosed?

The next step to determine the severity of the risk is to estimate the points (taking the median number) of "Likelihood" and "Impact" according to the listed items. The number of points refers the different levels of them, and the relationship is presented in Table 3.1. Finally, the severity level can be gained through Table 3.2. For example, when the likelihood is medium, and the impact is high, the overall severity level is high. The last step is to fix these risks, according to their priority, and the most severe risks should be fixed first. However, not all risks are worth fixing, and the cost should be considered when deciding to repair risks issues. In chapter 5, we will have a detailed analysis of these vulnerabilities, and give the solutions as well.

Likelihood and Impact Levels	
0 to <3	LOW
3 to <6	MEDIUM
6 to 9	HIGH

Table 3.1: Relationships between points and levels

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Table 3.2: Likelihood and Impact levels

Chapter 4

Case study

We post our security test methodology of the Web application in chapter 3. In this chapter, a case study of a security testing is presented. VEM (virtual environment manager) in Finnish company Tieto was regarded as the target application to conduct the security test. Zap and Burp Suite were utilized as tools to scan the VEM, and they generated eleven types of security vulnerabilities finally.

4.1 Test system architecture

Figure 4.1 is the architecture of test system in our experiment. The two main components are code manager (GitHub) and issue manager (Jira) [4]. The test script and tools are used to probe the application code; then, the results are posted into Jira to be tracked. The results can show which vulnerabilities need to be fixed, and the code is modified after tracking the issues. After that, the code is checked and fixed in the next round of the test.

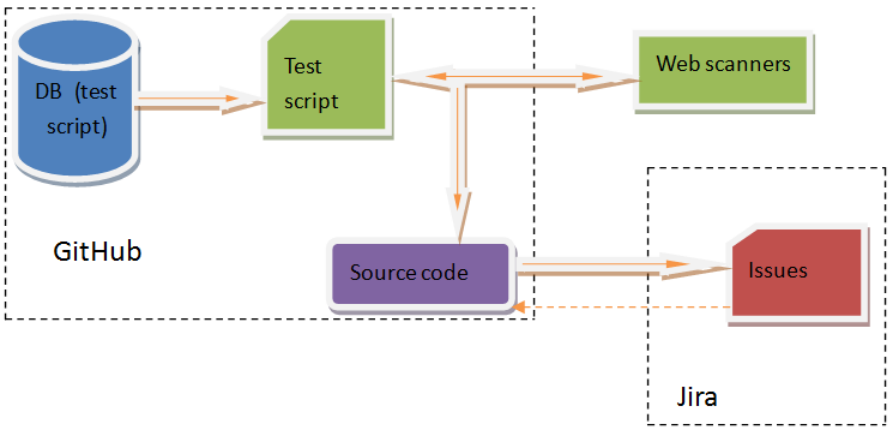

Figure 4.1: Test system architecture of VEM

4.2 Test tools

Table 4.1 lists the tools applied in this thesis. Burp Suite has two versions, free and commercial version. In this thesis, the test was carried with commercial one, due to the writer doing the experiment during work in Tieto company. Considering the limited experimental conditions, the author decided to use

ZAP (Zed Attack Proxy) as another tool to probe the target application vulnerabilities. This choice was made due to its fitness for the Linux experimental platform, being very simple to use and having the capability of finding a wide range of web application vulnerabilities.

Owners	Tools	Version
OWASP	Zed Attack Proxy	2.5.0
PortSwigger	Burp Suite	1.7.16

Table 4.1: Used web vulnerability scanners

4.3 Test target

We use VEM (Virtual Environment Manager) as our experimental target. It is a tool that helps clients to create their test environment on Tieto's private OpenStack cloud. VEM allows users creating virtual instances on it, like operating on the OpenStack dashboard. Simultaneously, users can also view and manage their instances when logging into OpenStack account on Tieto's cloud. Comparing with the OpenStack, VEM has two obvious advantages:

- It helps create instances conveniently by setting the properties of instances (e.g., IP, network) automatically. Otherwise, users may spend some time to type these properties when creating the instances.
- It also helps remove instances from OpenStack quickly.

The aim of VEM is to improve the effectiveness when using the OpenStack. The VEM refers to a wide range of technologies, which include HTML/CSS/-JavaScript, Docker, python, Kafka and so on.

4.4 Test results

The detailed threat test profile information of two scanners (ZAP, Burp Suite) was obtained after using the tools, and Figures 4.2 concludes the result. It plots 11 types of vulnerabilities were found by our scanners: some of them are detected by both scanners, while some of them are only detected by one scanner. It is also necessary to rank these risks to repair the application bugs, due to the limitation on development time. Figure 4.3 presents the percentage of each severity level.

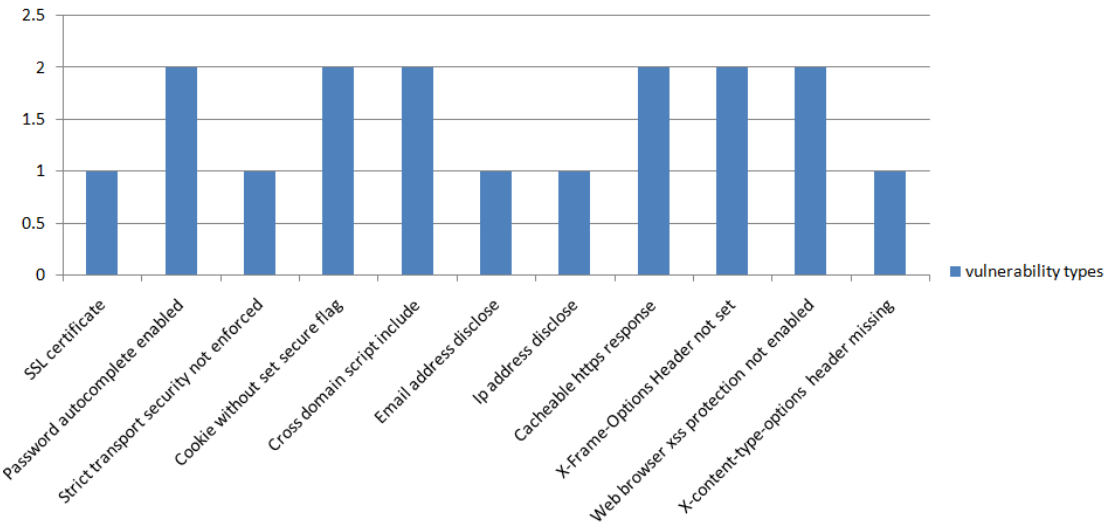

Figure 4.2: Average ratio of each vulnerability

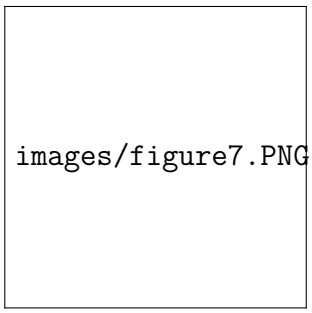

Figure 4.3: Severity level percentage

Chapter 5

Detailed analysis of the vulnerabilities

In the chapter 4, we have presented our experiment environment, including target system, and scanning tools. Figure 5.1, 5.2 are snapshots of scan results. In this chapter, we will discuss the details of the vulnerabilities. The analysis of each vulnerability consists of four parts: description, exploitation, severity evaluation and solution.

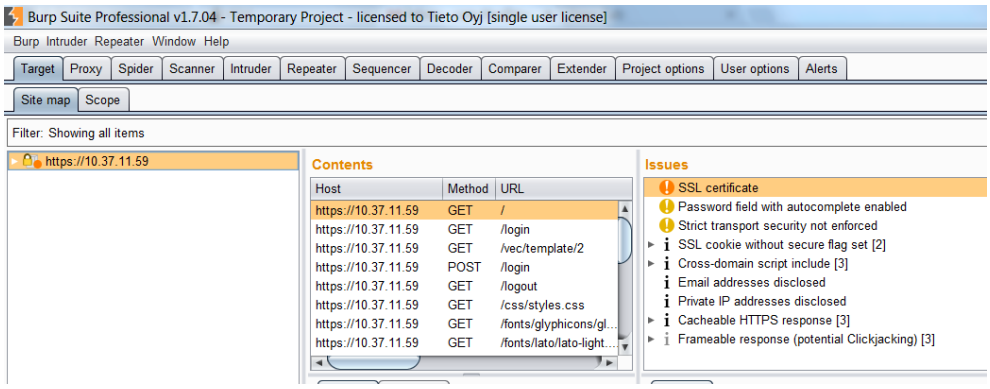

Figure 5.1: Burp scanner results

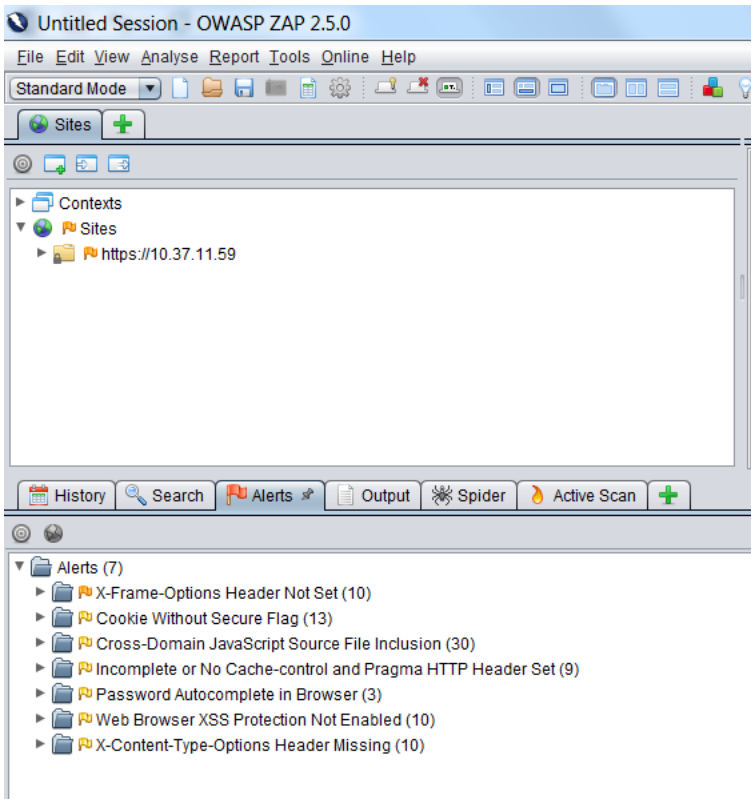

Figure 5.2: Zap scanner results

5.1 SSL cookie without secure flag set

Description: If a cookie is set with security flag, browsers will not send it in any requests that use HTTP connection. Therefore, setting the secure flag prevents the cookie from being intercepted by an attacker monitoring network traffic.
Exploitation: Authentication is required for communication between web server and its client. A client usually authenticates web server through SSL/TLS and a certificate, while there are three general ways for the server to verify who the client is. These techniques include cookies, hidden forms, and encrypted URL [36]. When a cookie is used as the authentication token, anyone can access user session when it is gained. If the secure flag is not set during a cookie's transmission, it will allow being transmitted over an insecure connection, for instance, public Wi-Fi or another network shared with a comprised computer. As a result, attackers can eavesdrop on the victim's network traffic and stolen the cookie. In the next, the attacker is entitled to access all user's information, which can cause a serious consequence. In a form-based authentication, the following similar codes will appear:

```
<form method='POST' action='https://www.example.com'>
  <input type='hidden'....>
```

```
</form>
```

Furthermore, corresponding configuration is needed in server [17]. For instance,

```
<login-config>
  <auth-method> FORM</auth-method>
  ............
</login-config>
```

The encrypted URL authentication method is used in a situation such as SSL/TLS transmission. Three keys play essential roles in whole phases, which are the public key, private key and session key. This infrastructure will be further introduced later in the thesis. HTTP [9] is a stateless protocol that uses a mechanism such as a cookie to keep track of the state of a transaction [31]. Simultaneously, a cookie can be used as a session key during HTTPS (HTTP Secure) transaction. The VEM is an example that a cookie contains session key. According to OWASP [28], browsers' extension tools, e.g. Chrome Extension Cookie Editor, Firefox Cookie Editor, are helpful in determining if a cookie is an authentication token in a web application. After testing with Chrome Extension Cookie Editor, we found that VEM is not built with this mechanism while based on encrypted URL transmission. The client request and server response were tracked in Burp to prove cookie's usage, and they are listed in table 5.1. The table presents that cookie value is maintained the same after SSL connection established. Therefore, it is a session key. When considering this situation, the risk is lower than expected if the cookie is exposed to attackers.
Severity level: Low
Solution: Each cookie has five attributes, that are secure, HttpOnly, domain, path, and expires. A secure tag informs the browser that the cookie can only be sent over a secure channel such as HTTPS. HttpOnly tag disallows the cookie to be accessed via a client-side script such as JavaScript. The domain is used to compare against server domain with requested client domain, and only hosts within the specified domain are allowed to set a cookie for that domain. The path attribute is supplemented domain, and expires to set persistent cookies, within which the cookie will not expire. [28] All the attributes can be set in configuration files on servers. Application developers should pay particular attention to cookie attributes to fulfill their requirements.

	URL	Request	Response
step1	https://10.37.11.59/	GET / HTTP/1.1 Host: 10.37.11.59 ...	HTTP/1.1 200 OK Server: nginx ... set-cookie:connect.sid=s%3AteH9DFvQtLL7nMabLQttu7li2o27RIxl.M710HYAB65DQOUGZYbWP5XW6%2FTK0%2BLFHrD7X30uRcOU; Path=/;HttpOnly
step2	https://10.37.11.59/login	GET /login HTTP/1.1 Host: 10.37.11.59... Cookie: connect.sid=s%3AteH9DFvQtLL7nMabLQttu7li2o27RIxl.M710HYAB65DQOUGZYbWP5XW6%2FTK0%2BLFHrD7X30uRcOU; Path=/;HttpOnly	HTTP/1.1 200 OK Server: nginx ...
step3	https://10.37.11.59/login	POST /login HTTP/1.1 Host: 10.37.11.59... Cookie: connect.sid=s%3Av0Pw4YBxAL9rVMMRpIQDWdLzVJ1KSfJc.nBW3FMiGN8HTSBGuAESezPqJKK9FQGSq3Xru3VOOSOU	HTTP/1.1 302 Moved Temporarily Server: nginx ...
step4	https://10.37.11.59/vec/workspace	GET /vec/workspace HTTP/1.1 Host: 10.37.11.59... Cookie: connect.sid=s%3Av0Pw4YBxAL9rVMMRpIQDWdLzVJ1KSfJc.nBW3FMiGN8HTSBGuAESezPqJKK9FQGSq3Xru3VOOSOU	HTTP/1.1 200 OK Server: nginx ...
step5	https://10.37.11.59/logout	GET /logout HTTP/1.1 Host: 10.37.11.59... Cookie: connect.sid=s%3Av0Pw4YBxAL9rVMMRpIQDWdLzVJ1KSfJc.nBW3FMiGN8HTSBGuAESezPqJKK9FQGSq3Xru3VOOSOU	HTTP/1.1 302 Moved Temporarily Server: nginx ...

Table 5.1: Cookie value changes in transaction

5.2 SSL certificate

Description: SSL/TLS establishes a secure channel between server and client. To be authenticated by its client, the server must provide an SSL certificate that holds its hostname, issued by a trusted authority in a valid time. Only meet both requirements, can the transmission be fully protected by SSL/TLS. VEM uses a self-signed certificate, which generates alarms from browsers. The alerts describe that the server's certificate is not valid for the server's hostname and the server's certificate is not trusted.
Exploitation: SSL certificates are data files that bind a cryptographic key to an organization's details [13], which include:

- A domain name, server name or hostname.
- An organizational identity and location.

When installing on a web server, the SSL certificate activates the HTTPS protocol and allows secure connections between a web server and a browser. The SSL is initially used for secure credit card transactions, data transfer and logins, and is also becoming popular when securing browsing of social media sites. An organization needs to install the SSL certificate onto its server to establish a secure connection from a web server to a browser. An SSL certificate works based on public key cryptography. The public key of the server is exposed, and the browser sends the message encrypted with this public key. Only the private key of the server can decrypt the message. A trusted Certificate Authority issues a legal SSL certificate, and browsers, operating systems, mobile devices maintain lists of trusted CA root certificates. To be trusted, the certificate must present its root certificate on the end user's machine. Otherwise, the browser will display an untrusted error message to end user. In some cases, for instance, e-commerce website, such error message indicates lacking confidence of the site and organization and can make losing of trust and business from most consumers. One well known, trusted Certificate Authority is GlobalSign, since operating system and browser vendors such as Microsoft, Mozilla, Opera, etc., believe that it is a legitimate CA and can be relied on to issue trustworthy SSL certificates.

VEM did not issue its Certificate to a trustworthy Certificate Authority. It used a self-signed certificate to authenticate to end users. At this stage, VEM is mainly for internal usage, which means users know this website is trustful. Thus, the severity level is low. However, as the VEM is open to more external users, its certificate must be issued by a Certificate Authority to gain users' confidence.
Severity level: Low
Solution: Two ways listed below can solve the problem:

1. Issue VEM's certificate to a trusted Certificate Authority.
2. Distribute self-signed certificate to all users.

There is a way to make a self-signed certificate secure and no need to pay to Certificate Authorities. An organization could first securely distribute it to all users who will connect to its website through SSL to ensure the self-signed certificate is not vulnerable to forgery. In other words, transmit a copy of the license to guarantee anyone can not disrupt it. This may involve posting the certificate on a secure file-sharing site or giving website users a physical copy using a disk. Then, the recipient of the license must install it in their browser as a Certificate Authority before connecting to the organization's website. Finally, the browser can connect to the target URL and verify the authenticity of the certificate. This free self-signed certificate is as secure as the paid one. However, it needs to know the website users first, then gains their agreement. If Tieto does not want to spend money on VEM certificate, it can distribute self-signed one to its users. Users install certificate in their browsers as a "Certificate Authority" before connecting with VEM. This way also guarantees the secure communication between VEM server and clients.

5.3 Strict transport security not enforced

Description: The application fails to prevent from connecting to it over unencrypted connections. An attack able to modify a legitimate user's network traffic could bypass the application's use of SSL/TLS encryption, and use the application as a platform for attacks against its users. This attack is performed by rewriting HTTPS links as HTTP so that if a targeted user follows a link to the site from an HTTP page, their browser never attempts to use an unencrypted connection.
Exploitation: If a connection through HTTP and redirects to HTTPS is allowed by a website, the user may initially talk to the non-encrypted version of the site before being redirected in this situation. For instance, the user types http://www.example.com or just example.com. This provides a potential of a man-in-the-middle attack, where the redirect could be exploited to direct the user to a malicious site instead of the secure version of the original page [7]. Figure 5.3 shows such a case: middleware attack leads browser to connect to a proxy, rather than directly to the server. The proxy uses SSL connect to the server and get resources to the user, while the user uses HTTP connection with it. The fake HTTP site gains user's secret, and tools such as sslstripe [18] can be used as such proxy. HSTS (HTTP Strict Transport Security) is a solution for this problem. By using HSTS, the browser will know to automatically use only HTTPS, which prevents hackers from performing a man-in-the-middle-attack. The first time one site is accessed using HTTPS, and it return the Strict-Transport-Security header, then the browser records this information so that it will automatically use HTTPS instead when the site is loading using HTTP in the future.

However, when the duration time specified by the Strict-Transport-Security header expires, the next attempt to load the site via HTTP will proceed as

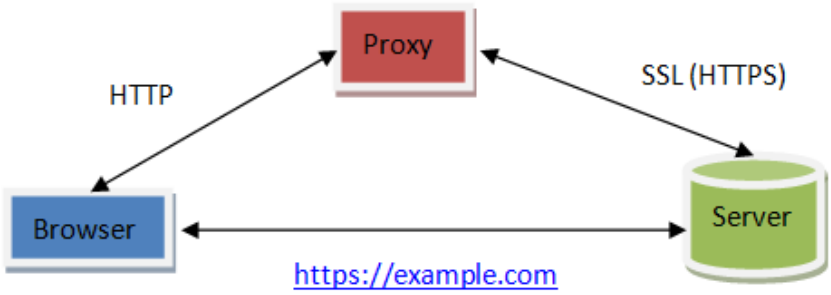

Figure 5.3: A man-in-the-middle attack scenario

usual without using HTTPS. Whenever the Strict-Transport-Security header is delivered to the browser, the expiration time for that site will be updated. The expiration time parameter can also be set to meet different requirements. Excepting for setting Strict-Transport-Security, another way is to use SSL/TLS connection to redirect to HTTPS automatically. VEM also utilizes this mechanism, so that HSTS is not a critical parameter. Due to the initial request in SSL/TLS connection is sent with a unencrypted way, there is still a possibility to suffer from a man-in-the-middle attack. HSTS can prevent this sort of attack entirely.
Severity level: Low
Solution: Enabling "strict transport security" feature in HTTP header by configuring VEM web server:
Strict-Transport-Security: max-age=expireTime [; includeSubDomains] [; preload]
The parameter "includeSubDomains" is optional, and it applies to all of the site's subdomains as well. Google maintains a "preload" service. Browsers will never connect domains using an insecure connection by following this guideline.

5.4 X-Frame-Options header not set

Description: "X-Frame-Options" is not included in the HTTP response to protect against "Clickjacking" attacks. Clickjacking is a malicious technique that involves deceiving a web user into interacting with another different website, and the user has no awareness. This technique can be combined with other attacks to cause severe impact such as sending unauthorized commands or revealing confidential information while the web user is interacting with seemingly harmless pages. The victim is forced to perform undesired actions (e.g., clicking a button that appears to activate another operation) by using specific features of HTML and javascript in a clickjacking attack.
Exploitation: OWASP provides a method to test clickjacking attack of one web application. Based on that, we found that the VEM is vulnerable and its pages can be embedded into another website by using HTML tag <iframe></iframe>.

Attackers may take advantage of this vulnerability to do something harmful. In such a case, one user is opening a site to play some games when he is logging into the VEM at the time. This game site can trick users with few buttons. It seems that the user is clicking the buttons on the game pages. However, this user is interacting with VEM's page without any awareness. The mimic interactions are shown in the Figures [5.4,5.5,5.6] below, and the opacity of the malicious page can be set to zero to prevent the user from seeing the embedded site.

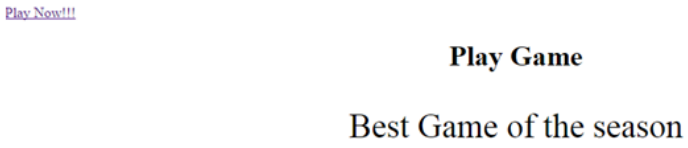

Figure 5.4: VEM page is embedded into malicious game page with opacity equals zero

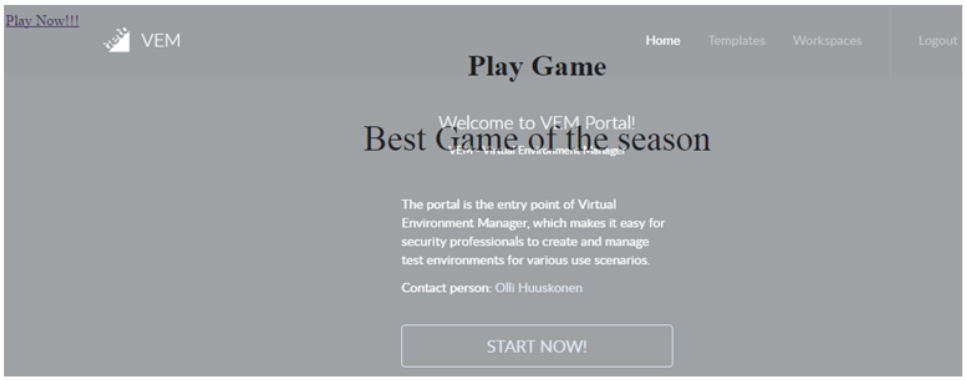

Figure 5.5: Victim acts in game site also interacting with VEM

In Figure 5.4, VEM is embedded with a seemly game website. Due to the HTML/CSS feature, the VEM page can be hidden and not seen by victims. When the user is clicking a button on the game page, he/she is activating the button in VEM page and enter an administration page in Figure 5.5. Figure 5.6 shows "Delete" button in VEM page is enabled while the user clicks "play" on the game site, where the malicious action succeed. The two buttons were connected with javascript functions related to "iframe" of the HTML page, and the completed exploitation source code is attached in First appendix A.

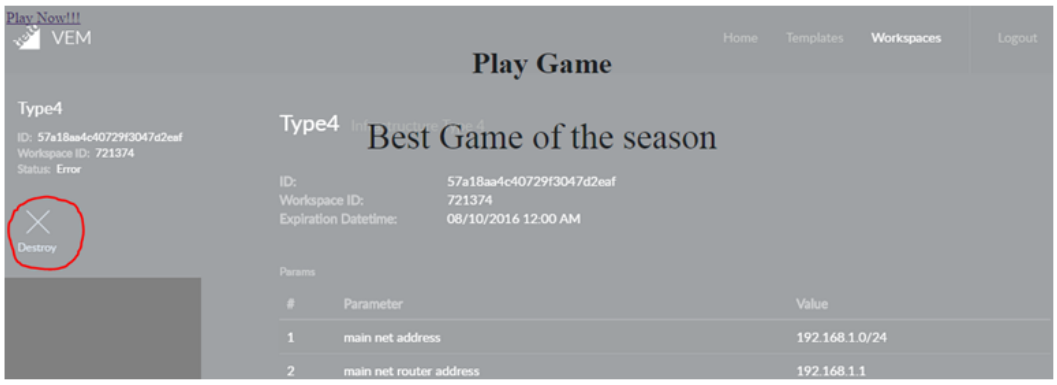

Figure 5.6: Victim click "play" button on game site and perform harm in VEM without knowing

Severity level: High
Solution: There are two common approaches to defense clickjacking attack. Microsoft implemented the X-Frame-Options header that was sent from the server on HTTP responses to mark web pages shouldn't be framed. The header contains values DENY, SAMEORIGIN, ALLOW-FROM origin, and non-standard ALLOWALL. DEDY is recommended to protect a website from clickjacking. Major browsers have adopted the X-Frame-Options header. However, it has main three limitations that may cause a possibility of clickjacking exploitation:

1. The header is not compatible with the old browser, since introducing in 2009.

2. The header may be stripped by web proxies. As a result, the site loses framing protection.

3. Though not every page of the site has implemented the header, the mobile version of the website has not to be protected.

The most common defense is called frame busting, which prevents a site from functioning when loaded inside a frame. [32] It consists of a script in the web page that should not be framed. The important techniques refer to mobile website version, double framing, disabling javascript, onBeforeUnload event, XSS Filter and redefining location [27, 32].

5.5 Web browser XSS protection not enabled

Description: Web Browser XSS Protection is not enabled, or disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server.
Exploitation: Cross Site Scripting (XSS) are among the most common types of attacks against the Web application. It involves echoing attacker's code

into a user's browser or client via web pages viewed by the target users [15]. These codes are written in different static or dynamic contents such as HTML, JavaScript, Active X, Flash or any other technology supported by the browser. A successful XSS attack helps attackers to hijack user's account via a cookie, redirect a user to another website from the original one and to increase other types of attacks such as phishing. Where the XSS causes significant risk is in the browser that interacts closely with the file system on the end user's computers for loading content.

The most two common XSS prevention mechanisms are Filtering and Escaping [1]. XSS attack code could come from a simple <form> submitted by website users, or could take a more complicated route such as a JSON script, or XML web service. Filter XSS protection is to pass all external data through a filter which will remove dangerous keywords, such as the infamous <SCRIPT> tag, JavaScript commands, CSS styles and another dangerous HTML markup. Developers can implement own filter mechanisms using existed libraries. When using escaping, the browser is told effectively that the sending data should be treated as data, rather than interpreted in any other way. An example list escape codes of HTML is presented in Table 5.2.

Original characters	After escaping
"	&#34
#	&#35
&	&#38
(	&#40
)	&#41
;	&#59
<	&#60
>	&#62

Table 5.2: List of escaped characters

If an attack manages to put a script on the target page, the victim will not be affected because the browser will not execute the script if it is properly escaped. Implementation filtering is more sophisticated than escaping. When looking through the VEM codes, neither escaping nor filtering mechanism was found. It seems that the possibility of XSS attack is relatively high.
Severity level: Middle
Solution: The header X-XSS-Protection enables XSS filter in browsers. By setting different parameters, it can be enabled or disabled. The X-XSS-Protection HTTP response header should be set to '1' to allow the filter [26]. One example is:
X-XSS-Protection: 1; mode=block
Another way is to implement escape mechanism in VEM's source code.

5.6 X-Content-Type-Options header missing

Description: The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially interrupting the body of reply and displaying as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
Exploitation: The MIME type is the mechanism to tell the client the variety of the document transmitted: the extension of a file name has no meaning on the web [19]. It is critical to correctly set up the server to ensure that each document transmits the correct MIME type. Browsers often use the MIME-type to determine what to do when fetching a resource. Table 5.3 is a list of example MIME types.

Extension	Type of document	MIME Type
.aac	AAC audio file	audio/aac
.bin	Any kind of binary data	application/octet-stream
.bz	BZip archive	application/x-bzip
.css	Cascading Style Sheets(CSS)	text/css
.csv	Comma-separated values(CSV)	text/csv
.gif	Graphics Interchange Format(GIF)	image/gif
.js	JavaScript	application/javascript
.html/htm	HyperText Markup Language(HTML)	text/html

Table 5.3: List of MIME types

In the case that the MIME type absence, browsers may conduct MIME sniffing to guess the exact MIME type by fetching the resource. The action is performed differently in various browsers under different circumstances. However, there are some security concerns with this practice due to some MIME types represent executable content and others not. MIME sniffing is a way of attempting or deducing the file format or change the content. The files uploaded by attacks could contain malicious contents or which is intentional payloads, while these files seem benign when considering their content types or Multipurpose Internet Mail Extension (MIME) information. [34] There is a list to quickly assess whether an application is vulnerable to MIME sniffing attack:

1. The application allows uploads.
2. The application does no post-processing on the uploaded content.
3. The application is downloadable from the application.

4. The content is not checked for MIME type mismatches.

MIME sniffing only exists on a website where users can upload files according to content sniffing criteria, VEM is not satisfied. Thus, this type of attack is a false positive.
Severity level: None
Solution: X-Content-Type-Options header was introduced Microsoft in IE8 [19] as a way to block content sniffing that could transform non-executable one MIME type into another executable one. The server can block MIME sniffing by sending the X-Content-Type-Options according to the Content-Type that is used to indicate the media type of the resource. This header should be set to avoid the vulnerability warnings:
X-Content-Type-Options: nosniff

5.7 Cross domain JavaScript source file include

Description: The page includes one or more script files from a third-party domain.
Exploitation: Combining data and code from third-party sources have enabled adding creativity and functionality to the Web application. However, this is also involving new security challenges to the Web application. Current practices include giving uncontrolled cross-domain execution through the use of <script> tags and extending the browsers with plugins to interaction cross domains [14]. The former has a high potential security risk due to one site gets complete control over another. External JavaScript files can harm website aspects such as reading cookies, reading user input, changing what the user sees, executing forms as the user, performing requests to other servers and so on. External JavaScript files can only be included when developers trust the domains involved will never harm them. If the external code is served by an extensive CDN (e.g., Google's CDN), the possibility is lower to be exploited. The reason is that large companies have enough money to spend on security. There are many various third-party JavaScript source files included in VEM's source code. Thus VEM is vulnerable to this type of attack.
Severity level: Low
Solution: Using static content from a CDN, or copying the third-party scripts into VEM own domain and including them from there. If that is not possible, for instance, some licensing reasons, then considering re-implement the scripts' functionality within application code.

5.8 Incomplete or no cache-control and pragma HTTP header set

Description: The cache-control and pragma HTTP header have not been set properly or are missing allowing the browser and proxies to cache content.
Exploitation: Web content caching can classify with basic caching and advanced caching. The former one is the simplest caching options. A web page is stored in a cache when it is rendered by the Web Content Manager application the first time. Then, users can access this page from the cache until it expires. The main advantage of basic caching is to improve the performance of the web application, and should only be used for static content that does not require "real-time" access. Five types of advanced caching are list below:

1. Site caching: Same as the basic web content cache except that cache parameters in connect tags and URL requests can be used to override the server's default advanced web content caching settings.

2. Session caching: When it is enabled, a copy of each page the user visited is stored in the session cache. The user accesses the cached version of a webpage until they start a new session, or until it is expired.

3. User caching: The same as session cache except that the page is saved in the user cache.

4. Secured caching: This is used on sites where the item security features are used to grant different users access to different web pages and components based on the groups they belong to.

5. Personalized caching: It is used to cache web pages of users who have the same "personalization profile". This means that users who have selected the same customization categories and keywords, and belong to the same group, share a single cache.

There are two major differences between basic caching and advanced caching. One is that advanced caching can cache pages based on different user profiles. Another is that those cache parameters in connect tags and URL requests can be used to override the server's default advanced web content caching settings making set individual web pages possible. In most cases, user, secured and personalized caching would mostly be used when using custom caching in connect tags and URL requests. While basic, site and session caching would only be used as the server's default web content cache.

Applications such as Google, Facebook tend to keep user's session always alive. They use session caching. It appears that both Firefox and Chrome have restored session cookies, for the convenience of their users, between browser shut down and restart. However, this is insecure. Only persistent cookies

should restore in this way. A typical example of the usage of persistent cookie would check "keep me logged in" or "remember me" when logging into one site. At the same time, websites such as a bank never cache anything. Depending on the functionality of a website, it can choose whether using caching or not, or use which type of caching. The cache-control head is not set in VEM, and the following figure 5.7 is a cache page found in Firefox after user logout VEM. Some sensitive information such as user's ID, created workspace can be tracked. It is possible for attackers to do some exploitation with this information. If VEM does not need to keep user always online, it should disable the cache functionality provided by the browser.

Figure 5.7: VEM cache page in Firefox

Severity level: Low
Solution: To prevent the individual user profile from being seen, VEM could disable the cache feature. According to the definition of HTTP/1.1 [10], HTTP header should be set with "Cache-Control: no-cache, no-store" and "Pragma: no-cache" in VEM.

5.9 Email addresses disclosed

Description: One private email address was disclosed in the response.
Exploitation: Email addresses may appear intentionally with contact information, and many applications include third-party email addresses within their core content. However, email addresses of developers or other individuals may disclose information that helps the attacker. Imagine an email address represents username that can be used to the application's login, and it is possible to be used in social engineering attacks against the organization's personnel. The

private email is used to receive feedback from users in VEM. Disclosing it may increase the number of spam emails, but it is not a security vulnerability.
Severity level: None
Solution: Removing any email address that is unnecessary, or replacing it with anonymous one such as helpdest@example.com. Furthermore, considering hide the email address and instead with a form that generates the email server-side to reduce the spam.

5.10 Private IP addresses disclosed

Description: The following RFC 1918 IP addresses were disclosed in the response:
192.168.1.0
192.168.1.1

Exploitation: RFC 1918 specifies ranges of IP addresses that are used in private networks and cannot be routed on the public Internet. An attacker can determine the public IP addresses in use by an organization through various ways. However, it is difficult to determine the private addresses. The web vulnerability scanner alarmed "Disclosing the private addresses used within an organization can help an attacker in carrying out network-layer attacks aiming to penetrate the organization's internal infrastructure". In this VEM case, it is still very hard for attackers to perform evil events after having a glance of private IP address.
Severity level: None
Solution: If the private addresses are being returned in service banners or debug messages, this service should be configured to mask the private addresses. If they are being used to track back-end servers for load balancing purpose, the addresses should be rewritten with innocuous identifiers from which an attacker disables to infer any useful information about the infrastructure. [29]

5.11 Password field with auto-complete enabled

Description: Most browsers have a facility to remember user credentials that are entered into HTML forms. This function can be configured by the user and also by applications that employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.
Exploitation: Passwords saved in the browser can be accessed locally, which means an attack can get victim's user credentials. However, the browser will usually ask users if they allow their passwords to be remembered or not. If users share browser with others, they can choose not remember their passwords.

When users use their private browser, automatic password filling is convenient for them. Thus, this alarm should not be considered as a type of vulnerability.
Severity level: None
Solution: Do not allow the browser remember user's password by hand.

Chapter 6

Discussion

This section will discuss the whole experiment: the summary of the evaluated risks; the completeness and uncovered part of the experiment; the limitation and future work.

6.1 Evaluation of the results

We conclude all the risks in this part. Some of them have higher severity level, and need to be fixed in advance. The other security testing types in the VEM's system are also discussed here.

6.1.1 Risk analysis

In the last section, we have analyzed the security vulnerabilities in VEM of Tieto. Based on methodology we proposed, table 6.1 presents the evaluation results. We could conclude that VEM is vulnerable to "Click jacking" attack due to lack of X-Frame-Options header or has not implemented the frame busting technology. It has a relatively high possibility to suffer from XSS attack as well, because VEM has no defense to this type of attack. Setting X-XSS-Protection header can enable XSS filter, and escape mechanism iss also effective to prevent the XSS attack. There are 5 types of vulnerabilities that have low severity level, taking the largest part in classification groups by severity level. These vulnerabilities are cookie without the secure flag, SSL certificate not issued by a CA, strict transport security not enforced, cross domain JavaScript file include, and no cache control set. Developers should follow the severity levels in decreasing order when improving the application. We also found a false positive detected by the two scanners. The "X-Content-Type-Options header not set" report is harmless because the system does not need to upload files, which does not meet the MIME attack requirements. Sensitive information such as email addresses and private IP disclosure cannot be regarded as a type of security vulnerability in this case. Also, the password automatic filling function cannot be exploited, either.

We have reviewed the web application development process and weaknesses in this thesis, especially web vulnerabilities. The vulnerability definition, detection tools, verification and evaluation methodology are emphasized in it. More importantly, one practical web application VEM (Virtual Environment Manager) is used to do the experimental based on our security test methodology. After using scanner tools, 11 types of vulnerabilities were discovered. Each of these was fully analyzed and evaluated by the author.

Vulnerability Name	Severity Level
SSL cookie without secure flag set	Low
SSL certificate	Low
Strict transport security not enforced	Low
X-Frame-Options header not set	High
Web browser XSS protection not enabled	Medium
X-Content-Type-Options header missing	None
Cross domain JavaScript source file include	Low
Incomplete or no cache-control and pragma HTTP header set	Low
Email addresses disclosed	None
Private IP addresses disclosed	None
Password field with auto-complete enabled	None

Table 6.1: VEM security vulnerability distribution

6.1.2 Evaluation

Our work studied one web system and evaluated the severity level of vulnerabilities based in this application. Most existing research has studied web weaknesses from a statistical perspective and does not give the exploitation process. For example, some papers covered a large number of websites and examined various vulnerabilities without exploitation source code; some papers analyzed the effectiveness of various scanners under different situations. However, our project has two distinct advantages. One is that we have access to application source code, which makes exploitation possible. Thus, the evaluation of the weaknesses is more complete. For example, if the code does not contain XSS defense, it is vulnerable to this attack. Another advantage is that the author is both the inside developer and user of this application. In this case, the thesis can dive into both the user's and attacker's mindsets to consider the attacks. For instance, automatic password filling should not be considered a real vulnerability because it is a useful feature to the user. It also can be avoided if they choose not remember the password when sharing devices or web accounts with other users.

This thesis initially states the background of web vulnerability testing, then, presents the tools and methodology for it. Based on the model, we implemented our own web vulnerability testing project, and the OWASP TOP 10 2013 covers most of the results. We hope that the test report can help Tieto improve Virtual Environment Manager. The exploitation processes are beneficial to later research as well.

Many current businesses define penetration testing as the application of automated network vulnerability scanners to operate a site. However, the true penetration testing is much more than that. [16] It depends on the skill and experience of the tester. The VEM is a view for users to create their own virtual

instances; the users and administrator of the system can manage their virtual machines in an OpenStack environment. To implement the full penetration testing for the VEM, there are still many test cases to consider. Some of them are listed below:

1. No authentication in the client-server system. The VEM is designed for particular users; however, it appears that any host who can access the address can create virtual instances.

2. As the IaaS, VEM cannot guarantee that there is no security hole in the virtual images it uses; no images signing for virtual machines.

3. The physical security of the infrastructure is extremely important in IaaS. The hard in VEM should also be tested frequently such as preventing damage from both natural and intentional way.

4. The paths by which the data is obtained or transmitted are also vulnerable in a cloud environment, because the data will be transmitted from source to destination through numerous third-party devices.

All in all, the completed test for the IaaS should consider storage, network, backup and fault. [3]

6.1.3 Limitations

There exist a large number of web scanners ranging from commercial to open source, and from Windows to Linux. This thesis just chooses ZAP and Burp Suite for some specific reasons (e.g., test platform, tight budget), which may cause the scan result to contain some bias. Different scanners may have various reports, and they also have different ability to test certain types of vulnerabilities. Thus, testers should try to use more scanners to detect web weakness.

When we exploit the application, we can only think from our perspectives. While potential attackers are always more intelligent than developers, they may have talent ways to attack the system. Thus, the evaluation result of these vulnerabilities may be not stable.

6.2 Future work

When exploiting found vulnerabilities, the private IP address disclosure case is not so fully understood. PortSwigger [29] states that this practice can help an attack in carrying out network-layer attacks aiming to penetrate the organization's internal infrastructure. Another source [2] shows that this alarm may be a false positive. Due to lack of related knowledge, the author can't exploit this attack. However, the private IP must be seen by the system user, because they are allowed to configure their private IP addresses. As considering all of

this, this attack is regarded as a false positive and has no effect to the application security. In the future, we have to improve the exploitation technology to provide a more accurate assessment.

Though we used two different scanners in this project, each generated a different report. It is reasonable to conclude that there are still some vulnerabilities that have not been found. Future work will also focus on the search for more weaknesses to improve the VEM security level.

Chapter 7

Summary

Numerous websites have suffered from various attacks since e-commerce is significant in our economic. If a risk happening on a great web organization such as Amazon, Yahoo, they may lose millions of dollars even in a short time. There have few such events occurred in recent years. Hackers can maliciously exploit the majority of web applications with software bugs, and the Web vulnerabilities exist in different types.

Though it is hard to prevent attacks or risks from happening, their number can be controlled by performing the security testing during applications development. All organizations should implement security testing during their application development, where vulnerability scanning is a good practice. Just like in other tests, white-box and black-box testing are common in a security testing. Web vulnerability scanner is a black-box way to detect weaknesses in a system. They can identify vulnerabilities automatically after proper setting. There exists a significant number of scanners range from commercial use to free use, for instance, AppScan from IBM, WebInspect of HP, Vega from Subgraph and so on. They can be potential tools to help organizations increase the security level of their applications. Except that, organizations such as OWASP have summarized the most common vulnerabilities. OWASP also provides vulnerability exploiting methods and appropriate testing tool. The team needs to identify which devices are suitable for them based on their situations, and the purpose of the scanning, platform, cost, ease of use and the software support.

In this thesis, we presented a methodology to measure the severity of VEM application from Tieto in chapter 3, and used two web scanners (ZAP and Burp Suite) to check its weaknesses. They detect 11 types of vulnerability in the system. After that, we verified, exploited every vulnerability carefully from various perspectives, evaluated their severity levels, and gave the relative solutions in chapter 5. We conclude five types of vulnerability have low severity level; one with very high gravity; one with medium severity level and rest four categories are the false positive. They are listing below:

1. High severity: X-Frame-Options header not set
2. Medium severity: Web browser XSS protection not enabled
3. Low severity: SSL cookie without secure flag set; SSL certificate; Strict transport security not enforced; Cross domain JavaScript source file include; Incomplete or no cache-control and pragma HTTP header set
4. False positive: X-Content-Type-Options header missing; Email addresses

disclosed; Private IP addresses disclosed; Password field with auto-complete enabled

We have access to the source code of VEM so that it is possible to do exploitation. Some exploitations use the third-party tools (e.g., sslstrip to mimic man-in-the-middle attack), and some need the tester to implement attack code. It is also necessary to read a broad range of references when evaluating a vulnerability. During the whole experiment, the author has a glance of the principles of common web vulnerabilities and how to detect, resolve them.

Due to a limited number of scanners, the exposed vulnerabilities were not so fully, and the small bias may produce when evaluating severity level. Though these limitations exist, this project still contributes to the VEM application of Tieto. It helps development team to improve their product based on the security awareness report. What is more, the later security testing can also refer the detailed exploiting process described in the thesis.

In the future, we will continuously find the vulnerabilities in VEM to improve its security level.

Appendix A

First appendix

```
<!DOCTYPE html>
<html>
<h1 style="text-align:center">Play Game</h1>
<script>
   window.onbeforeunload = function()
   {
      return " Do you want to leave click.site?";
   }
</script>
<head>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>

<body>
<p style="font-size: 38px;"align=center>Best Game of the season</p>
 <div style="z-index:10; opacity:0.4; position:absolute; top:0px; left:200px; ">
   <iframe name="clickjacking" src="https://10.37.11.59/" width="1500" height="1500"> </iframe>

   <script type="text/javascript"> console.log("test");

      $(document).ready(function(){
          $('#link').click(function() {
            destroy();
          });
      });

      function iframeRef( frameRef ) {
      return frameRef.contentWindow
         ? frameRef.contentWindow.document
         : frameRef.contentDocument
      }

      var inside = iframeRef( document.getElementsByClassName('pull-left') );
```

Figure A.1: Clickjacking attack exploitation code(1)

```
34        function destroy(){
35
36            var click = inside.getElementsByTagName("destroy");
37        }
38
39        </script>
40
41        //how to connect click with a elment in javascript?
42      <a href="https://10.37.11.59/vec/workspace"
43            target="clickjacking"
44            id = "link"
45            >Play Now!!!</a>
46
47   </div>
48  <div style="top:0px; left:200px;">
49  </div>
50  </body>
51
52  </html>
53
54
55  <div class="pull-left">
56      <button data-toggle="modal" data-target="#destroy57a18aa4c40729f3047d2eaf" name="destroy" title="
    Destroy Infrastructure" class="btn btn-transparent">
57          <span class="icon icon-destroy"></span><span class="icon-help-text">Destroy</span></button>
58  </div>
```

Figure A.2: Clickjacking attack exploitation code(2)

Bibliography

[1] Acunetix. Preventing xss attacks, 2011. `http://www.acunetix.com/blog/articles/preventing-xss-attacks/`. Accessed 4.3.2017.

[2] Acunetix. Possible internal ip address disclosure, 2016. `https://www.acunetix.com/vulnerabilities/web/possible-internal-ip-address-disclosure`. Accessed 4.3.2017.

[3] Hala Albaroodi, Selvakumar Manickam, and Parminder Singh. Critical review of openstack security: Issues and weaknesses. *Journal of Computer Science*, 10(1):23–33, 2014.

[4] Atlassian. Jira software, 2017. `https://www.atlassian.com/software/jira`. Accessed 15.3.2017.

[5] Jason Bau, Elie Bursztein, Divij Gupta, and John Mitchell. State of the art: Automated black-box web application vulnerability testing. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 332–345. IEEE, 2010.

[6] Paul E Black, Elizabeth Fong, Vadim Okun, and Romain Gaucher. Software assurance tools: Web application security scanner functional specification version 1.0. *Special Publication*, pages 500–269, 2008.

[7] Franco Callegati, Walter Cerroni, and Marco Ramilli. Man-in-the-middle attack to the https protocol. *IEEE Security & Privacy*, 7(1):78–81, 2009.

[8] Giuseppe A Di Lucca and Anna Rita Fasolino. Testing web-based applications: The state of the art and future trends. *Information and Software Technology*, 48(12):1172–1186, 2006.

[9] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999.

[10] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, 1999.

[11] Roy T Fielding and Richard N Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2 (2):115–150, 2002.

[12] Simson Garfinkel and Gene Spafford. *Web security, privacy & commerce.* " O'Reilly Media, Inc.", 2002.

[13] GlobalSign. What is an ssl certificate?, 2016. https://www.globalsign.com/en/ssl-information-center/what-is-an-ssl-certificate/. Accessed 4.3.2017.

[14] Collin Jackson and Helen J Wang. Subspace: secure cross-domain communication for web mashups. In *Proceedings of the 16th international conference on World Wide Web*, pages 611–620. ACM, 2007.

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

[16] J. P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 Workshop on New Security Paradigms*, NSPW '00, pages 15–21, New York, NY, USA, 2000. ACM. ISBN 1-58113-260-3. doi: 10.1145/366173.366183. URL http://doi.acm.org/10.1145/366173.366183.

[17] Mikalai Zaikin. Compare and contrast the authentication types, 2005. http://java.boot.by/wcd-guide/ch05s03.html. Accessed 4.3.2017.

[18] Moxie Marlinspike. Software: sslstrip, 2012. https://moxie.org/software/sslstrip/. Accessed 4.3.2017.

[19] Mozilla Developer Network. X-content-type-options, 2017. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options. Accessed 4.3.2017.

[20] OWASP. Category:vulnerability, 2016. https://www.owasp.org/index.php/Category:Vulnerability. Accessed 4.3.2017.

[21] OWASP. Category: Owasp top ten project, 2016. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project#tab=OWASP_Top_10_for_2013. Accessed 4.3.2017.

[22] OWASP. Owasp guide project, 2016. https://www.owasp.org/index.php/OWASP_Guide_Project. Accessed 4.3.2017.

[23] OWASP. Owasp cheat sheet series, 2016. https://www.owasp.org/index.php/Cheat_Sheets. Accessed 4.3.2017.

[24] OWASP. Owasp testing project, 2016. https://www.owasp.org/index.php/Category:OWASP_Testing_Project. Accessed 4.3.2017.

[25] OWASP. Owasp risk rating methodology, 2016. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology. Accessed 4.3.2017.

[26] OWASP. Owasp secure headers project, 2016. https://www.owasp.org/index.php/OWASP_Secure_Headers_Project#X-XSS-Protection. Accessed 4.3.2017.

[27] OWASP. Testing for clickjacking, 2016. `https://www.owasp.org/index.php?title=Testing_for_Clickjacking_(OTG-CLIENT-009)&setlang=en`. Accessed 4.3.2017.

[28] OWASP. Testing for cookies attributes, 2016. `https://www.owasp.org/index.php/Testing_for_cookies_attributes_(OWASP-SM-002)`. Accessed 4.3.2017.

[29] PORTSWIGGER. Private ip addresses disclosed, 2017. `https://portswigger.net/KnowledgeBase/issues/Details/00600300_PrivateIPaddressesdisclosed`. Accessed 4.3.2017.

[30] Bruce Potter and Gary McGraw. Software security testing. *IEEE Security & Privacy*, 2(5):81–85, 2004.

[31] Guy Pujolle, Ahmed Serhrouchni, and Ines Ayadi. Secure session management with cookies. In *Information, Communications and Signal Processing, 2009. ICICS 2009. 7th International Conference on*, pages 1–6. IEEE, 2009.

[32] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. *IEEE Oakland Web*, 2(6), 2010.

[33] SAP. Sap-security-in-figures-a-global-survey-2013, 2013. `https://erpscan.com/wp-content/uploads/2013/11/SAP-Security-in-Figures-A-Global-Survey-2013.pdf`. Accessed 4.3.2017.

[34] Bhupendra Singh Thakur and Sapna Chaudhary. Content sniffing attack detection in client and server side: A survey. *International Journal of Advanced Computer Research*, 3(2):7, 2013.

[35] Marco Vieira, Nuno Antunes, and Henrique Madeira. Using web security scanners to detect vulnerabilities in web services. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 566–571. IEEE, 2009.

[36] David A. Wheeler. Secure programming howto, 2015. `http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/web-authentication.html`. Accessed 4.3.2017.