VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATED WEB APPLICATION VULNERABILITY DETECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE FRANTIŠEK KOLÁČEK
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÁ DETEKCE ZRANITELNOSTI WEBOVÝCH APLIKACÍ
AUTOMATED WEB APPLICATION VULNERABILITY DETECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE AUTHOR	FRANTIŠEK KOLÁČEK
VEDOUCÍ PRÁCE SUPERVISOR	RNDr. MAREK RYCHLÝ, Ph.D.

BRNO 2015

Abstrakt

Tato práce si dává za cíl analyzovat a porovnat implementaci již existujících open source scannerů webových zranitelností (pomocí nástrojů Wivet a Wavsep) a na základě této analýzy navrhnout a implementovat efektivnější způsob testování jednotlivých skupin útoku do open source scanneru Revok.

Abstract

The aim of this thesis is to analyze and compare implementation of already existing open source web application vulnerability scanners (using test suites Wivet and Wavsep) and according to this analysis to propose and implement more effective way of testing each class of vulnerabilities to open source scanner Revok.

Klíčová slova

web, bezpečnost, detekce zranitelností, revok, wivet, wavsep, jenkins

Keywords

web, security, vulnerability detection, revok, wivet, wavsep, jenkins

Citace

František Koláček: Automated Web Application Vulnerability Detection, bakalářská práce, Brno, FIT VUT v Brně, 2015

Rozšířený abstrakt

Tato práce se zabývá analýzou a porovnáním implementací již existujících *open source* scannerů webových zranitelností a možnostmi zlepšení hodnocení a efektivity testování *open source* scanneru Revok.

Úvod práce se zaobírá základními pojmy v oblasti testování bezpečnosti webových aplikací a také projektem OWASP, jež sdružuje bezpečnostní experty z celého světa, vydává odborné publikace a také zastřešuje vývoj nástrojů určných pro testování bezpečnostni webových aplikací. Dále následuje stručný přehled testovaných zranitelností včetně vysvětlení principu jejich zneužití. Testované zranitelnosti jsou *SQL Injection*, *Cross-site scripting*, *Local file inclusion*, *Path traversal* a *Remote file inclusion*.

Pro analýzu, porovnání implementace a efektivity testování jednotlivých scannerů bylo použito testovacích nástrojů WIVET a WAVSEP. Na základě této analýzy bylo navrhnuto několik způsobů, jak zlepšit výsledné hodnocení scanneru Revok v testech, jež jsou obsaženy v nástrojích WIVET a WAVSEP. Součástí analýzy bylo srovnání hodnocení scanneru Revok s 4 nejpoužívanějšími open source scannery (Arachni, w3af, Wapiti a ZAP).

Jednou z navržených změn bylo vytvoření automatizovaného testovacího prostředí scanneru Revok pomocí nástroje Jenkins pro zjednodušení následného testování aplikovaných změn. Mezi další navrhnuté a implementované změny scanneru Revok patří vylepšení webového crawleru pro lepší hodnocení v testech nástroje Wivet a vylepšení detekce *Local file inclusion* zranitelností v testech nástroje WAVSEP (Revok získal před implementací změn v obou zmiňovaných testech hodnocení 0%).

Výsledkem této práce je funkční prototyp automatizovaného testovacího prostředí Jenkins obsahující definice pro oba testovací nástroje (WIVET a WAVSEP). Dále byly implementovány a otestovány změny zajištující lepší hodnocení v obou testovacích nástrojích. Veškeré výsledky a implementace byly poskytnuty komunitě scanneru Revok.

Automated Web Application Vulnerability Detection

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval sám pod vedením RNDr. Marka Rychlého, Ph.D. a s pomocí odborných konzultací od pana Jana Rusnačka, jenž je odborným vedoucím práce za společnost Red Hat Czech, s. r. o.

....
František Koláček
May 18, 2015

Poděkování

Rád bych poděkoval svému vedoucímu práce RNDr. Marku Rychlému, Ph.D. za odborné vedení, motivaci a cenné rady při řešení této práce a svému technickému vedoucímu práce za společnost RedHat Czech, s. r. o. panu Jánu Rusnačkovi za poskytnutí cenných rad a technických konzultací.

Bachelor's Thesis Assignment

Unofficial Translation

Student

František Koláček

Specialisation

Information Technology

Topic

Automated Web Application Vulnerability Detection

Category

Web

Instructions

1. Study the state of the art in web application security scanning and vulnerability detection. Study, evaluate, and compare detection capabilities and features of existing open-source web application scanners, including Revok.
2. Based on the analysis of the existing scanners and Revok, propose several new features of Revok. The features will address missing detection capabilities in Revok, for common classes of web vulnerabilities.
3. Propose the most effective approach to the detection of each addressed class of web vulnerabilities.
4. Implement the proposed features of Revok and evaluate the results in comparison with selected scanners applied on available web applications for penetration testers and on real open source web applications. Any found vulnerabilities shall be disclosed responsibly.
5. Document the project results and possible further enhancements in a technical report.

Supervisor

Marek Rychlý, RNDr., Ph.D., UIFS FIT VUT

Contents

Chapter 1

Introduction

Subject of this bachelor thesis is to analyze, compare and evaluate functionality of already existing open source web application vulnerability scanners and to propose and implement new features to recently open sourced scanner Revok [34] to make it more effective in finding security vulnerabilities in web applications.

In the first part of this thesis selected open source web application scanners are analyzed and tested and their efficiency and number of implemented features are evaluated and compared. The results discussed in Evaluation section 3.2 were used to identify weak points of the current version of Revok scanner and also to propose missing features. Based on this evaluation selected features have been implemented.

1.1 Web application security

According to the Technopedia definition [43], *„Web application security is the process of securing confidential data stored online from unauthorized access and modification“*. Web sites, web applications and web services are unfortunately really prone to security risks since web servers are open to internet by design. The most serious sources of security risks are misconfigured or not updated servers and website sources themselves. Exploiting or misusing discovered vulnerabilities or security threats can lead to potential private data leak, service unavailability or data loss.

With every new feature developer or/and every change performed on deployed web application the chance of creating new vulnerability is increasing. It means that securing web applications against hacker's malicious intentions is always a long-term and time-consuming project. Since more and more important applications have become accessible over network and number of performed attacks on these applications is growing rapidly, their security is becoming more and more important topic these days.

1.2 Testing of web application security

Testing of web application security is a process of analyzing web application's behavior and finding security risks and possible vulnerabilities, which can lead to data corruption. This analysis is performed to minimize the possibility of unauthorized access, sensitive data modification or theft either by person or by malicious script.

1.3 Automation in testing

Nowadays the security of web applications is mostly tested by automated programs also called web application security scanners. Web application security scanner usually performs, unlike source scanners, only black-box testing. It means that it does not perform any source code analyses (it does not even have access to the source code of tested application) but it communicates with a web application only through its web front-end. For this reason, these scanners are trying to identify potential security vulnerabilities by actually performing attacks to known security issues and then analyzing the application's output and behavior.

1.4 OWASP and OWASP foundation

OWASP Foundation is worldwide not-non-profit charitable organization [37] which supports OWASP (The Open Web Application Security Project) globally around the world. OWASP is focusing on making security of application more visible and easily understandable by creating freely available articles, documentation and tools. Their main goal is to openly inform about security risks and potential vulnerabilities so that either individuals or organizations can make informed decisions on how to secure their applications.

1.5 Top 10 web vulnerabilities in 2013

The OWASP 1.4 every year publishes updated version of document where the most critical web application vulnerabilities are listed. The security flaws mentioned on this list occur very frequently in web applications and are very dangerous because they could be easily found and exploited. These vulnerabilities could be used for stealing data and sensitive information, preventing your software to work properly or causing complete data loss. According to the OWASP [24], the list of the most common and exploited vulnerabilities in 2013 is following:

- A1 - Injection
- A2 - Broken Authentication and Session Management
- A3 - Cross-Site Scripting (XSS)
- A4 - Insecure Direct Object References
- A5 - Security Misconfiguration
- A6 - Sensitive Data Exposure
- A7 - Missing Function Level Access Control
- A8 - Cross-Site Request Forgery (CSRF)
- A9 - Using Components with Known Vulnerabilities
- A10 - Unvalidated Redirects and Forwards

Each of vulnerabilities is described in detail in OWASP Top10 document also with examples of possible attacks and information on how to avoid them. OWASP also organizes special trainings which cover (not only) Top10 problematic. The main purpose of these trainings is to keep users and developers informed as much as possible and to increase awareness of potential consenquences of these attacks so they can analyze and protect their applications.

Selected security flaws from this OWASP Top10 document will be explained in section 3.3. These flaws will be also tested with open source web application scanners mentioned in chapter 2 and results of these tests are presented later in section 3.5.

1.6 Structure of the document

In the next chapter called Analysis of existing solutions 2 selected open source scanners will be introduced. For the full list of analyzed scanners refer to the table 2.1. Information about their authorship, licenses, tested versions and used programming languages and also publicly available list of features can be found in section 2.6.

In the next chapter called Testing and evaluation 3, techniques which have been used for testing and evaluating functionality of each scanner are introduced. The list of the tested vulnerabilities is mentioned in section Tested vulnerabilities 3.3 and brief description of each vulnerability and approaches of their detection are mentioned in section Approaches commonly used to detect vulnerabilities 3.4. Evaluation of functionality and efficiency of each scanner by using testing suites WIVET 3.1.1 and WAVSEP 3.1.2 is mentioned in section 3.2.

The results of efficiency and functionality of each web application scanner are introduced in chapter 3.5.

Analysis of missing features in Revok scanner along with the implementation suggestions for new features are introduced in chapter Missing features in Revok 4.

At the end of this thesis, there is a conclusion 6 of my contribution to Revok web application scanner.

The content of the attached DVD is listed in appendix A. Installation and usage guides are mentioned in appendix B. The specific tested scenarios are mentioned in table 3.2 and more detailed results of each scanner are listed in appendix C.

Chapter 2

Analysis of existing solutions

For my bachelor thesis analysis I have only chosen commonly used open source scanners because of a possibility to check their implementation and efficiency. The complete list of chosen scanners, including information about their development, is mentioned in table 2.1.

Scanner	Developer	Version	Language	License
Arachni [40]	Tasos Laskos	1.0.6	Ruby	Apache v2.0
Revok [34]	Yubin Yan	0.8.0	Ruby	GNU AGPLv3.0
w3af [31]	Andres Riancho	1.6	Python	GNU GPLv2.0
Wapiti [38]	Nicolas Surribas	2.3.0	Python	GNU GPLv2.0
ZAP [32]	OWASP Foundation	2.3.1	Java	Apache v2.0

Table 2.1: List of tested scanners

All of the scanners mentioned have their source code publicly accessible. Each of the tested scanners is introduced more closely in the following sections.

2.1 Arachni

Arachni [40] is an open source, modular and high-performance framework developed in Ruby, aimed at helping security specialists, administrators and penetration testers to test and evaluate web application's security.

Arachni also has an integrated browser support so it is able to audit and test client-side code. Due to its feature Arachni can be used for testing highly complicated web applications which are using dynamic technologies like JavaScript, AJAX and it can be used for DOM [26] manipulation.

There are two possibilities to operate this framework: either from command line or via its web interface. It can be also configured for use in multi-node high effective grids. Arachni also provides easy RPC API [23] for managing the whole framework.

2.2 Revok

Revok [34] is an easy to use online web app security scanner developed in Ruby. It is completely open source (since Nov 17th 2014) and it is released under GNU AGPLv3.0 [8]. Revok aims to be as easy as it could possibly be. With this in mind, any software engineer cat test and evaluate security of web application constantly and through the full software development life cycle.

Revok also provides JavaScript crawler for testing basic client side scripts. It is also designed to be deployed in both single-node and multiple-nodes environment, which makes it possible to run scans in parallel. Revok itself consists of the following components:

- Web console (user interface for submitting new scan requests)
- REST API (API for handling requests from web console)
- Messaging server (distributing messages/requests across Caroline nodes)
- Caroline nodes (working nodes performing scans)
- Database (storing details for all scan tasks)

The communication workflow is illustrated on figure 2.1.

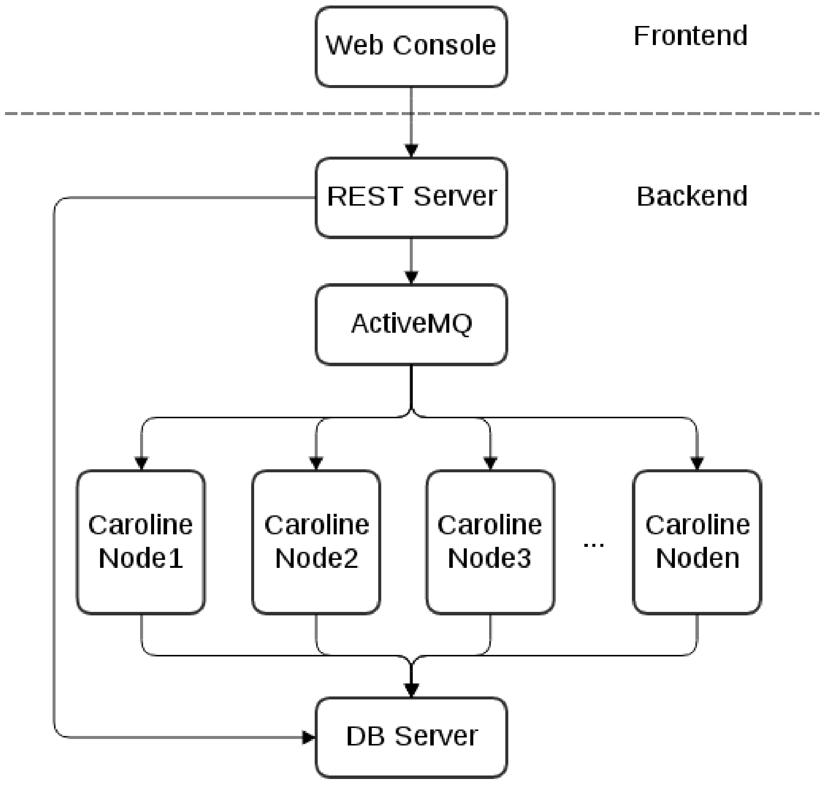


Figure 2.1: Revok components overview [20].

2.3 w3af

Tool w3af (short form for Web Application Attack and Audit Framework) [31] is an open source framework used for securing web applications by analyzing and exploiting their vulnerabilities. This cross-platform framework is developed in the Python programming language so it is available for all of the popular operating systems such as Linux, FreeBSD, Mac OS X, Microsoft Windows and others.

w3af can be used via command line but it also provides graphical interface and functionality could be easily extended by using more than 130 available plugins. This framework was started as one man show by Andres Riancho in March 2007, but later (in July 2010) sponsorship and partnership with Rapid7 was announced (which allowed rapid development of this framework).

2.4 Wapiti

Wapiti [38] is a web application vulnerability scanner using blackbox approach for testing and evaluating web application's security. Wapiti is developed in Python and it supports both GET and POST HTTP methods for discovering possible vulnerabilities. Once the structure mapping of targeted application is finished, Wapiti will try to test payloads to test whether discovered scripts and forms are vulnerable or not.

Wapiti provides only command line interface so it is not as user friendly as other scanners providing web interface, so it is more suitable for experienced users. Wapiti, like other tested scanners, is completely open source released under GPLv2.0 [9] license.

2.5 ZAP

ZAP (short for Zed Attack Proxy) [32] is penetration testing tool developed by OWASP foundation 1.4 and used for finding vulnerabilities in web applications. It provides both passive and active scanners and can be used either for automatic or manual testing. ZAP behaves as intercept proxy and could be used in combination with properly configured web browser to browse pages manually and manipulate all of the traffic coming through.

It is suitable for absolute beginners as well as for skilled testers because it provides very wide range of functionalities. In daemon mode it could also be controlled by powerful REST API [18].

ZAP is one of the most active OWASP projects and it is a part of OWASP flagship project. It is developed in Java and distributed under Apache License v2.0 [2]. It was also awarded in recognizing competitions (e.g. second place in the Top Security Tools of 2014 as voted by ToolsWatch.org readers [1]).

2.6 Features summary

Each of the tested scanners has its own subset of implemented features. In the table 2.2 only general features of each scanner are mentioned.

Scanner	CLI	GUI	WEB	API	Report	Log	Pause	Parallel scans
Arachni	✓	✗	✓	✓	✓	✓	✓	✓
Revok	✗	✗	✓	✓	✓	✓	✗	✓
w3af	✓	✓	✗	✗	✓	✓	✓	✗
Wapiti	✓	✗	✗	✗	✓	✓	✗	✗
ZAP	✗	✓	✗	✓	✓	✓	✓	✗

Table 2.2: List of implemented features.

2.6.1 User interface

Most of the tested scanners have some form of graphical interface (represented by columns GUI and WEB) which can be also used by less experienced users for submitting their tests. Only one of them (Wapiti) has command line interface (column CLI in table above) only.

2.6.2 Reporting and logging

All of the tested scanners are able to create scan report (usually in form of web page) so it is easier for user to check the results of performed scan. These results usually contain also useful hints and recommendations on how to improve security of tested web application. All of tested scanners also provide functionality of logging results to some log files (in form of plain text or formatted for example using JSON [5]).

2.6.3 Remote access and API support

Some of the tested scanners (Revok, Arachni and ZAP) provide remote API (REST [18] or RPC [23]) which can be used for integration these scanners to automated testing workflows like continuous integration testing. The rest of the tested scanners do not support any kind of remote API.

2.6.4 Parallel scans and pause functionality

Some of the tested scanners are also able to pause running tests and resume them later (represented by column Pause in table 2.2). Only two of the tested scanners (Revok and Arachni) are capable to be deployed on multiple nodes and provide parallel scans functionality.

Chapter 3

Testing and evaluation

The selected tools have been used for testing and evaluating possibilities and efficiency of each web application vulnerabilities scanner mentioned in the chapter Analysis of existing solutions 2. Each one of these tools is briefly described bellow. The results of all tested scanners are presented in chapter Results 3.5. Installation of each scanner and testing suite is described in appendix B and arguments, enabled modules and targets used during testing are described in table 3.2.

3.1 Testing methodology

All tested scanners have been running multiple times with various arguments. In the first run, all of the tested scanners, including Revok, have been tested in default configuration and with all default modules enabled. In the second run, there were only appropriate modules enabled for each attack vector and scanner. These two runs have been compared with each other and only the highest earned score for each vector type and each scanner has been recorded.

Later testing has been realized using open source CI (continuous integration) testing tool called Jenkins [28]. For both testing suites (WIVET and WAVSEP) have been created jobs which fully automated testing of Revok scanner against these testing suites. Details of Jenkins CI implementation is presented in chapter Jenkins CI tool for Revok 5.3.

3.1.1 WIVET

WIVET [41] (short for Web Input Vector Extractor Teaser) is a benchmarking project that measures how effective is tested scanner in crawling web applications and in finding possibly exploitable input vectors. WIVET project is developed in PHP [1], which means that it could be hosted on almost any common hosting (which supports PHP). Deployment and usage of this tool is straightforward and does not require any additional configuration. WIVET itself provides very easy-to-use web interface where you can see actual progress of testing and also the total coverage of each scanner (statistics are bounded to the PHPSESSID[2]). Detailed installation and usage of this tool is described in appendix B.1.

[1]PHP - Hypertext Prepropcessor, popular general-purpose scripting language

[2]PHPSESSID - Unique session identificator used by PHP applications to determine different clients.

3.1.2 WAVSEP

WAVSEP [39] (short for Web Application Vulnerability Scanner Evaluation Project) is the evaluation platform which contains a collection of unique web pages. Each of them is vulnerable in different way so the whole framework provides possibility to test various properties of web application scanners. WAVSEP is developed in Java and it uses Tomcat [27] application server for running. Detailed installation and usage of this tool is described in appendix B.2. This project does not contain any kind of evaluation so custom evaluation method has been used (details will be presented in following section).

3.2 Evaluation

This section describes steps which have been performed in order to evaluate efficiency and detection accuracy of each class of vulnerabilities using testing suites WIVET and WAVSEP. Specific testing scenarios which have been used for testing and evaluating are mentioned in table 3.2.

3.2.1 WIVET

Each one of tested scanners has been tested first against WIVET 3.1.1 to determine coverage of each attack vector. WIVET itself is capable to evaluate ability of each scanner to use particular attack vector and also to present the results via its web interface. WIVET includes tests for 56 different attack vectors so it is possible to express the success ratio of each scanner in percentage. WIVET results are presented in chapter 3.5, section 3.5.1. For testing coverage of each scanner has been used WIVET in version 4.

3.2.2 WAVSEP

WAVSEP is vulnerable web application which contains a collection of unique web pages. The WAVSEP contains 1261 vulnerable pages in total (142 SQLi, 98 XSS, 828 LFI, 115 RFI and 78 other vulnerabilities which are not mentioned in this thesis). Each class of web attacks (SQLi, XSS, LFI, RFI) was tested separately and only dedicated plugins/modules were enabled during these tests. For testing abilities of selected scanners was used WAVSEP in version 1.5.

3.2.3 Evaluation using WAVSEP

This testing platform provides only links to vulnerable pages and description of each vulnerability but does not provide any way of evaluating the detection accuracy of each scanner. For this reason I have created scripts which helped me to gather links of all vulnerable pages from WAVSEP to the testing database and parse logs of each scanner/attack. I have determined discovered vulnerabilities from these logs and then evaluated the overall detection accuracy. All of these scripts are located on attached DVD in directory *bp-jenkins-scripts* and more complex usage along with dependencies are described in appendix B.2. Wavsep testing and evaluation of Revok's results have been simplified and transformed to separate Jenkins job. This job was used later for testing Revok's improved testing capabilities (after applying all changes mentioned in chapter Features implementation 5).

Gathering links

The list of the unique URLs for testing pages was generated using script *wavsep-gather-links.sh*. This script is using the same configuration file as the one used by Jenkins scripts (which will be introduced later 5.3). This script takes only one parameter from command line – the name of the vector which it should generate links for (allowed values are: *sql, xss, lfi and rfi*). The rest of the configuration options is located in the shared configuration file which is also used for Jenkins scripts. This script also uses the location of your WAVSEP installation (e.g. *http://localhost:8082/wavsep/*).

Database initialization

For initializing database was used script *wavsep-init-db.sh* which uses as a parameter the name of the SQL file containing links of vulnerable pages. I have used this database as a reference for evaluating whether the scanner discovered this particular vulnerability or not.

Log file analysis

Analysis of generated log files from each scanner was performed by running script *wavsep-eval.sh* which takes as an argument the name of the analyzed scanner. It searches for log files in directory *results* and performs analysis of logs which match the name (e.g. *revok-xss.html*, *revok-sql.html* and others).

Gathering statistics

Script *wavsep-results.sh* was used for the gathering of final statistics, which prints on standard output the information about successfully discovered attacks of each class for each tested scanner.

3.2.4 Testing and evaluation using Jenkins

The continuous integration testing for Revok open source web application scanner has been proposed in section Proposed features 4.2 and later implemented and described in section Features implementation 5. Jenkins jobs allow fully automated testing of Revok's capabilities and these jobs have been created for both testing suites (WIVET and WAVSEP). Details about them, their configuration and usage is described later in this thesis.

3.3 Tested vulnerabilities

In this section there are the tested vulnerabilities presented and possibility of its exploiting is explained.

3.3.1 SQLi

According to the OWASP [19], SQL injection (also known as SQLi) is one of the commonly used attacks performed and tested against web (and not only) applications. It consists of insertion SQL [3] query via some input vector to the application which causes its execution. SQLi depends on incorrectly sanitized user's input which is used for injection attacker's

[3]SQL - Structured Query Language

code into the original query. This attack could lead to revelation of sensitive data stored in application database, allowing data manipulation (update,insert or even delete) and even administration operations can be executed on database.

3.3.2 XSS

The Cross-site Scripting (also known as XSS) is, according to the OWASP [6], another type of remote injection attack which uses injecting of malicious script for exploiting legitimate web site. This attack occurs whenever are the untrusted data received by the application and sent to the victim's web browser without proper escaping or validation (XSS allows executing of malicious script in victim's browser). This attack has very wide range of use and it is very difficult to detect by the victim. As stated by OWASP [25], there are three variants of this attack:

- Stored XSS
- Reflected XSS
- DOM Based XSS

This was the basic classification but it is also possible to divide Cross-site Scripting attacks by accessibility:

- Server XSS - occurs when malicious code is included in an HTML response received from the server
- Client XSS - occurs when malicious code is used to update the DOM by the Javascript call

Exploiting of these vulnerabilities could be used for accessing victim's sensitive data (e.g. cookies, session tokens and others) and also running malicious script in victim's browser (it could be used for example for updating content of the affected page).

Non-Persistent XSS

It is the most common type of XSS vulnerability because malicious code is not stored anywhere persistently and therefore does not require any additional interaction with any storage system (e. g. database system). For exploiting this vulnerability any user input which is not properly sanitized could be used.

Persistent XSS

This type of XSS attack is similar to the previous one but there is one difference - malicious code is stored persistently on the server (in database, files or any other resource providing storage) and can be exploited multiple times with every refresh of infected page.

DOM based XSS

This kind of attack is performed by changing DOM on client side without any interaction with server so there is no way how to check if client was compromised or not.

3.3.3 LFI and Path traversal

According to the OWASP [21], Local File Inclusion (also known as LFI) is the vulnerability, that allows the attacker to include files that are already present in the system (unlike Remote file inclusion which will be mentioned later in section RFI 3.3.4) to an application via the web browser. This vulnerability occurs when page does not use properly validated and sanitized input. This attack can be used for revealing sensitive data from files present on the system (e.g. configs, files containing passwords and others). This vulnerability is commonly exploited in PHP applications, but can be found also in other technologies (e.g. ASP, JSP).

This vulnerability also allows to use another attack called Path or Directory traversal. Principle of this attack is in exploiting page includes (the same way as in LFI) so an attacker would be able to access files and directories which are not present in the web folders [17].

3.3.4 RFI

The Remote file Inclusion (also known as RFI) is very similar to the Local File Inclusion vulnerability. The difference between these two vulnerabilities is in including remote files (files hosted on different server) instead of the local ones like in LFI [22].

3.4 Approaches commonly used to detect vulnerabilities

This section contains brief description of used approaches for detecting each of all mentioned vulnerabilities in web applications.

3.4.1 SQLi

SQL Injection is still a common web application vulnerability and according to the OWASP Top10 it is the most exploited vulnerability ever. Exploiting an SQL injection is nowadays very easy thanks to automated tools specialized on this kind of vulnerability such as SQLMap [35] or SQLNinja [36]. All of them are using set of already predefined payloads and they are trying to inject the same set of payloads to the web application. There are 6 basic payload types used for exploiting SQL injection according to the Arne Swinnen [3]:

- boolean-based blind
- time-based blind
- error-based
- UNION query
- stacked queries
- out-of-band queries

All of these payload types will be briefly described bellow. More details about these payloads and examples are above the scope of this thesis.

Boolean-based blind method

This method is based on injecting payloads (boolean subquery returning true) which leads to altering the original query and returning different content which was not originally requested.

Time-based blind method

This method is based on injecting payload which causes delay on database server and slows down the speed of the whole application in return.

Error-based method

This method is using payloads to break the original SQL request and force the database server to generate SQL error presented back to the client.

UNION query

This method is using SQL command UNION for merging results from multiple tables and printing information which was not originally supposed to be returned to the client.

Stacked queries

Method very similar to the UNION query method using multiple queries stacked together.

Out-of-band queries

This payload method can use for example a file which could be executed inside the web server directory.

3.4.2 XSS

Cross-site scripting are attacks where an attacker is able to get control of victim's web browser and use it for executing malicious script (usually in the form of Javascript code). According to the OWASP Top10 it is the third most exploited vulnerability these times. As already mentioned in previous section 3.3.2 there are 3 types of XSS attacks:

- Non-Persistent attack (also known as Reflected XSS)
- Persistent attack
- DOM based attack

There are mainly three kind of approaches for detecting XSS vulnerabilities and they are as following:

- Static analysis
- Dynamic analysis
- Combination of static and dynamic analysis

Some of the approaches will be briefly described bellow but this topic is too complex to be covered in this thesis. For more details about approaches for detecting XSS please refer to master thesis Approaches to detect SQL injection and XSS in web applications created by Abhishek Kumar Baranwal [4].

Static analysis approach

There are many types of static analysis used for detecting XSS vulnerability in web application according to the Abhishek Kumar Baranwal's thesis [4]:

- Bounded Model Checking
- Analysis of String
- Software Testing Approaches
- Taint Propagation Approach
- Using Untrusted Scripts

Dynamic analysis approach

- Syntactical Structure Approach
- Interpreter-based Approaches
- Browser-Enforced Embedded Policies Approach
- Proxy-based Approach

Static and Dynamic Analysis Approach

- Lattice-based Approach

3.4.3 Path traversal and LFI

This vulnerability is the result of insufficient validation and sanitization of browser input from users. Path traversal (also known as directory traversal) vulnerability can be located either in web server implementation or application itself and this vulnerability can exist in a variety of programming languages (it is not limited only to PHP and web applications).

Attackers exploiting this vulnerability send malicious URL to the server which instructs web server to return specific files. Attacker has to find URL which can be modified to retrieve the file from the server. The '../' sequence is commonly used for exploiting this vulnerability which basically instructs the web server to retrieve file from parent directory.

Local File Inclusion attack is very similar to the Path Traversal attack. The difference between them is that for exploiting LFI attacker uses files which are located on the targeted server. The most often used file on Linux systems is */etc/passwd* because attacker can be absolutely sure that it exists and it is accessible and readable by the web server. Attacker can use this vulnerability for retrieving of any file located on targeted server (any file which can be accessed by the user running web server or another exploited service).

3.4.4 RFI

Remote File Inclusion vulnerability is very similar to the LFI. The only difference between them is that in this case attacker uses file which is not stored locally but it is located on remote server (usually on server controller by attacker). Attacker's goal is to include remote file and force web server or hosted application to execute this included file. Exploiting this vulnerability is commonly used for uploading malicious scripts to victim server (very often used are tiny scripts which provide remote shell support or FTP access so attacker can easily upload new files and continue with exploiting application/server).

3.5 Results

This sections presents WIVET and WAVSEP results of each tested scanner. Used testing methodology and information how to interpret results was already described in the chapter Testing and evaluation 3. Specific testing scenarios used for testing scanners are mentioned in table 3.2.

3.5.1 WIVET results

All of the scanners has been tested against WIVET testing suite and results of evaluating ability of each scanner to use various attack vectors are mentioned in table 3.1. All of the tested scanners were tested multiple times with different modules enabled and only the highest score was recorded for comparison.

	Arachni	Revok	w3af	Wapiti	ZAP
Total coverage	96%	0%	16%	44%	69%

Table 3.1: WIVET results

Results of testing scanners against WIVET testing suite with information about used modules are mentioned in table 3.2.

Scanner	Enabled modules	Coverage
Wapiti	xss	44%
w3af	web-spider	12%
w3af	web-spider, digit-sum	16%
Revok	all	0%
Arachni	trainer, audit-links, audit-forms	96%
ZAP	basic	10%
ZAP	subtree	16%
ZAP	subtree, ajax	69%

Table 3.2: WIVET results according to enabled modules

For more detailed WIVET results please refer to the appendix C.1 where you can find the list of all tested vectors and corresponding results for each of tested scanners.

3.5.2 WAVSEP results

All of the tested scanners have also been tested against WAVSEP testing suite and results of each scanner are presented in table 3.3. Only relevant modules of each scanner were used for testing each class of vulnerability.

Vulnerability	Arachni	Revok	w3af	Wapiti	ZAP
SQLi	100%	32%	35%	100%	100%
XSS	90%	88%	37%	66%	100%
LFI	100%	0%	57%	51%	75%
RFI	100%	31%	16%	57%	100%
Total score	97.5%	37%	36.2%	68.5%	93.7%

Table 3.3: WAVSEP results

3.5.3 Results overview

The highest score reached by all of the tested scanners is mentioned in table 3.4. According to this table it is evident that the Arachni scanner performed the best with total score 96.8%, followed by ZAP which reached score 81.35%. The difference between the 2nd (ZAP) and 3rd position (Wapiti) is relatively big because Wapiti reached only 56.25%. The rest of the tested scanners reached even lower score (scanner w3af reached 26.1% and Revok only 18.5%).

	Arachni	Revok	w3af	Wapiti	ZAP
WIVET	96%	0%	16%	44%	69%
WAVSEP	97.5%	37%	36.2%	68.5%	93.7%
Total score	96.8%	18.5%	26.1%	56.25%	81.35%

Table 3.4: Results overview

Chapter 4

Missing features in Revok

4.1 Revok's results evaluation

According to the results of each scanner mentioned in the previous section, Revok is not the most effective and powerful scanner on the list of tested scanners although it performed well. As stated in table 3.1, Revok scored with result 0% only in WIVET test and it did not even passed basic tests.

As stated in table 3.3 Revok is the most effective in finding XSS vulnerabilities, on the second place was discovering SQL injection and Remote File Inclusion vulnerabilities. It is also apparent, that Revok has problems with detection of Local File Inclusion and Path Traversal vulnerabilities although it already contains dedicated plugin for finding Path Traversal vulnerabilities.

4.2 Proposed features

For testing new features implementation it is necessary to establish a testing environment which allows the possibility of an automated testing of efficiency and reliability of the new implemented features. This automated testing environment can be also used for ensuring that no regressive problem will be integrated to the new version of the Revok. It also means that the possibility that regressive problem will be integrated to the next release during the development cycle. For this purpose Jenkins CI Continuous Integration project has been used. More information about the configuration, host topology and usage of the Jenkins CI testing environment used in this specific project is presented in the following section Features implementation 5.

4.2.1 Enhancement of web crawler

The another proposed change/feature is to identify the problem causing the zero score for Revok in WIVET tests, develop fix for this problem, create pull request and inform developers and community where and what was the problem (ideally in form of creating issue and pull request afterwards).

4.2.2 Improve LFI detection

Enhance the testing and detection capability of already existing Path Traversal module and implement/add new checks for finding Local File Inclusion vulnerabilities.

Chapter 5

Features implementation

In this chapter will be presented new features and their implementation to the open source scanner Revok which was proposed earlier in section Proposed features 4.2.

5.1 Enhancement of web crawler

First proposed change was enhancing testing capability of Revok's web crawler to improve results in WIVET testing.

5.1.1 Problem with relative links handling

As I already mentioned in previous section, Revok has obvious problems with WIVET testing suite because it scored only 0%. After some research I've discovered that this poor score was caused by the lacking capability of handling html base tag during Revok's crawlering phase.

Revok was not able to handle relative links in case that there is HTML base tag present on the scanned page. This issue is not actually caused by Revok itself because Revok only uses project PhantomJS [33] for the whole process of analyzing / crawling the targeted page. PhantomJS is scripted headless browser based on Webkit which provides full web stack (including Javascript). There is already known issue [15] with handling relative URLs in PhantomJS but this problem is still not resolved.

Due to this bug, Revok was not able to resolve relative links properly and it led to 0% score reached in WIVET testing. The problem is that WIVET uses multiple frames on one page and there is base tag located in one of them. Although Revok was able to identify all links on scanned paged properly it was not able to combine the address stored in base tag with the relative links found on page. Because of this, Revok did not even reach these pages - server returned *Error not found* instead.

This problem has been already reported to the Revok team (on their original GitHub repository [34]). Issue request has been created with detailed description of this problem [13]. This lacking capability caused problems with proper web crawling, more specifically with the relative link handling. Possible fix/workaround for this issue has been developed and also introduced to the Revok's developers and community. Pull request [11] has been created for this purpose. Patch containing this fix/workaround is also located on attached DVD A.

5.1.2 Problem with multiple events on single page

Fixing problems with handling relative links mentioned in previous section improved WIVET testing coverage, but only to 19%. After this change Revok was able to handle basic attack vector like redirects. It was also able to handle some click and mouse events but not on all of them.

There was an implementation of basic method called *scanOthers* in web crawler module *webcrawler.js* which was supposed to perform basic click events on elements *li* and *span*. This implementation relied on fact that there is only one event with click event binded at the same time on single page. Web crawler module is also using jQuery [29] javascript library for easier orientation on scanned page (it allows easier DOM manipulation and provides powerful DOM selectors).

The base functionality of *scanOthers* method was to match all occurences of *li* and *span* elements and perform delayed click on each of them. The problem with this approach was that the first executed click event triggers redirect event and the whole page was redirected to different page. Redirect causes that none of the rest of delayed click events will be processed anymore.

There is one limitation which prevents from implementing easy and straightforward solution of this problem. Every action performed on scanned page using PhantomJS is sandboxed (method *evalute* which takes callback function as an argument). It also means that jQuery framework can be only used inside this sandboxed function and it is problematic to pass arguments to this function and return multiple values from it.

This problem has also already been reported to the Revok team and issue request for this problem has already been created [12].

I have developed possible solution for this problem and I already pushed this change to my forked repository. I have implemented method called *scanEvents* which is able to go through all elements specified in array *objectSelectors* and perform all events specified in array *events*. Supported elements currently are: *div*, *li* and *span*, but it is possible to add other elements as well. There are also new events supported in this implementation:

- click - event occurs when the user clicks on an element
- mouseout - event occurs when the mouse pointer is moved out of an element
- mouseover - event occurs when the mouse pointer is moved onto an element
- mousedown - event occurs when the user press' a mouse button over an element

This method uses workaround for identifying all selected elements with ID before triggering events even start. For this purpose is every element which should be tested (it means that events should be triggered on this element) marked with unique ID (in case that it does not have ID already). Then we are able to track down which element (by ID) was already tested and which was not and we can go through all of them creating sandbox for triggering every single event (it will not affect other events).

Pull request [11] has been created for this purpose. Patch containing this fix/workaround is also located on attached DVD A.

5.2 Improve LFI detection

The second proposed change was improving capability of detecting Local file inclusion vulnerability in WAVSEP tests.

5.2.1 Problem with limited list of entry points

The Revok scanner uses only predefined and very limited list of entry points for testing both Local and Remote file inclusion vulnerabilities. There is only one module present in Revok scanner which is responsible for scanning all three vulnerabilities (LFI, RFI and Path traversal) and it is called *path_traversal* module (stored in file *path_traversal.rb*). Revok provides possibility to select which modules will be used for requested scan so this module can be easily tested separately to ensure that results will not be influenced by another modules.

This module contains a list (it is an array *f_name*) of predefined entry points and only these entry points are tested and evaluated in this module. The list of these predefined entry points is as following: *doc, file, f, page, p, dir, filename, fname.*

The problem in this approach is that WAVSEP uses different entry point (namely *target*) so Revok was not able to test file inclusion attacks only with *path_traversal* module enabled. Revok is able to discover some RFI vulnerabilities with XSS modules enabled but it does not work with Local file inclusion and Path traversal vulnerabilities.

Solution of this problem is to add entry point used by WAVSEP to list of predefined entry points which uses Revok's *path_traversal* module. Issue request [10] has been created with suggested workaround which can be used as temporary solution.

5.2.2 Problem with stability of WAVSEP testing suite

During testing Revok's ability to detect Local file inclusion vulnerabilities using WAVSEP testing suite I have experienced various problems with stability of the whole testing suite. When I was testing some LFI pages, which are part of the WAVSEP testing suite, Tomcat server stopped responding. The problem occurred only when all LFI pages were tested in one scan. I have also tried to separate LFI pages to smaller groups divided by used protocol and expected return value (WAVSEP itself provides this separation using different index files for each combination of used protocol and expected return code) and it worked without any problem. This problem occurred also when I was testing LFI testing capability of other open source scanners mentioned in this thesis.

This problem with WAVSEP was already reported by Isaac Dawson [7] and Tasos Laskos [14].

The default behavior of the crawler module implemented in Revok scanner is to return only the target URL when remote server stops responding. This, with combination of previously mentioned problem, led to 0% testing score in WAVSEP tests.

Solution of this problem is to test each combination of tested protocol and expected return value separately and then merge these results together to get the final score. WAVSEP contains tests for 2 HTTP methods (POST and GET) and following HTTP responses:

- Errorneous HTTP 500 Responses
- Errorneous HTTP 404 Responses
- Errorneous HTTP 200 Responses
- HTTP 302 Redirect Responses
- HTTP 200 Responses With Differentiation
- HTTP 200 Responses with Default File on Error

5.3 Jenkins CI tool for Revok

For automated testing of new Revok features implementation has been used open source continuous integration tool Jenkins [28]. Jenkins is an open source continuous integration tool developed in Java. This project is licensed under MIT [16] license and it was originally forked from project Hudson. This tool has been chosen for this purpose because it offers many advantages such as:

- Easy installation (can be installed from rpm or via yum package manager)
- Easy configuration (the entire Jenkins can be configured via web user interface)
- Change set support (Jenkins can generate a list of changes between builds)
- Distributed builds (builds can be distributed to multiple nodes)
- Plugin support (Jenkins can be extended via many 3rd party plugins)

5.3.1 What is continuous integration

Continuous integration (also known as CI) is according to the TechTarget definition [42] *„software engineering practice in which isolated changes are immediately tested and reported on when they are added to a larger code base. The goal of CI is to provide rapid feedback so that if a detect is introduced into the code base, it can be identified and corrected as soon as possible“*.

5.3.2 Jenkins job for testing Revok against WIVET

For testing and evaluating Revok against WIVET testing suite Jenkins job called *Wivet-testing-suite* has been created. Configuration file for this job in form of XML is located on attached DVD A. Job is parameterized and can be executed in production and testing mode. The difference between these two modes is in the path of the Revok repository (these two paths are stored in configuration file for Jenkins scripts and can be adjusted). This job is also configured to store all log and result files from latest builds on Jenkins master so they can be analyzed later manually. Job itself consists from calling 3 separate scripts.

Script wivet-start.sh

This script is responsible for installing any necessary dependencies needed for running Revok and WIVET test suite. It also creates necessary folders with proper rights, clones both tools from paths specified in configuration file to destinations and ensures that service httpd is started and running prior executing tests.

Script wivet-test.sh

This script is used for testing Revok against WIVET test suite. It ensures that httpd service is running and also that all directories and tools created or cloned are on the right places. Testing against WIVET does not require having all Revok's services running because it is needed to test only web crawler module. Revok uses mitmdump [30] as intercept proxy for analyzing the whole communication between web crawler module and the target so this script ensures that this proxy is started prior the crawler itself. After this operation

PhantomJS process is started with proper arguments, it performs initialization of web crawler module and starts the scan. This script also ensures that once scan finishes both, mitmdump and PhantomJS processes are terminated.

Script wivet-stop.sh

Last script, which is used for testing and evaluating Revok's capability using WIVET in Jekins, is responsible for stopping all services that are not needed anymore, cleaning all files created only for testing purposes and also evaluating test results from statistic files generated by WIVET suite. The example of results generated by this Jenkins job is presented on screenshot 5.1.

5.3.3 Jenkins job for testing against WAVSEP

For testing and evaluating Revok against WAVSEP testing suite very similar Jenkins job to the previous one called *Wavsep-testing-suite* has been created. Configuration file for this job in form of XML is located on attached DVD A. Job is parameterized and can be executed in production and testing mode. The difference between them is the same as is in the job mentioned earlier. There is one additional parameter which can be used for customizing this job. This parameter is specifying attack vectors which should be tested. Allowed choices are: *all, sql, xss, lfi-partially*. The last one represents the workaround mentioned earlier in section 5.2.2. This job is also configured to store all log and result files from latest builds on Jenkins master so they can be analyzed later manually. Job itself also consists from calling 3 separate scripts.

Script wavsep-start.sh

This script is responsible for installing all needed dependencies for running Revok and WAVSEP test suite. It also creates necessary folders with proper rights, clones both tools from paths specified in configuration file to destinations and ensures that service tomcat [27] is started and running prior executing tests.

Script wavsep-test.sh

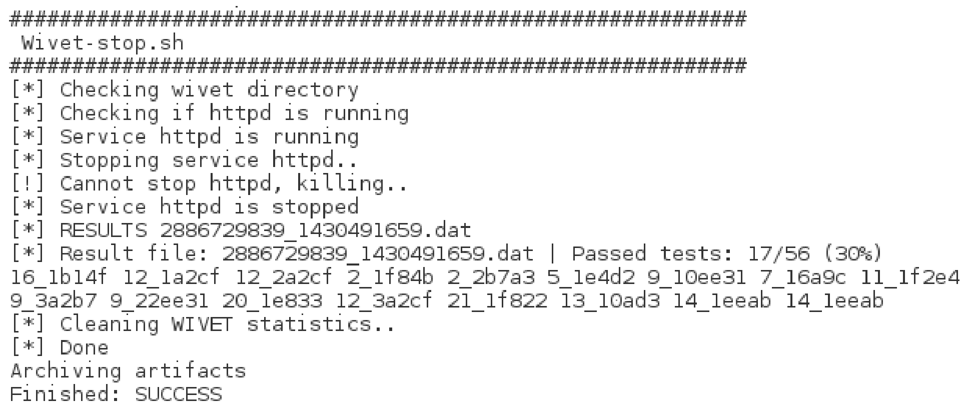
WAVSEP testing suite is more complex (in sense of preparing test environment) than WIVET one. For testing internal modules it is needed to start all Revok services and not only the two as in case of WIVET. This script ensures that all Revok's services are started and running properly and then requests all scans of attack vectors specified at the beginning. Requesting scans is done via using REST API which Revok provides (using specifically crafted PUT request with corresponding arguments).

Script wavsep-stop.sh

This script is responsible for cleaning all testing files and stopping services that are not needed anymore. It is also responsible for evaluating test results of every attack vector enabled and for printing statistics. The example of results generated by this Jenkins job is presented on screenshot 5.2.

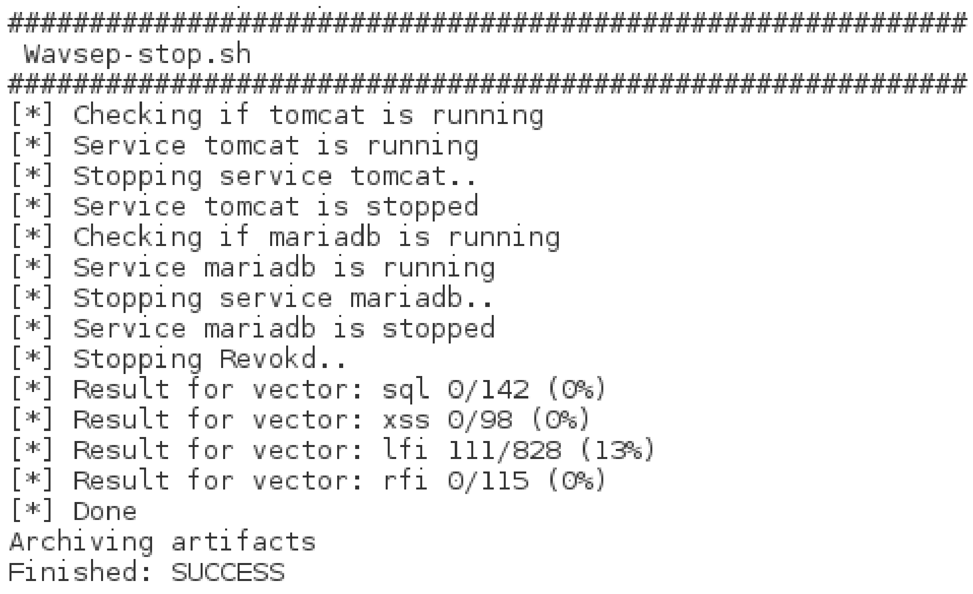
5.3.4 Results after features implementation

In this subsection screenshots of Jenkins job containing results for each of used testing suites are presented. The results mentioned below were reached after applying all implemented changes proposed in section 4.2 and implemented in section 5.



```
#######################################################
 Wivet-stop.sh
#######################################################
[*] Checking wivet directory
[*] Checking if httpd is running
[*] Service httpd is running
[*] Stopping service httpd..
[!] Cannot stop httpd, killing..
[*] Service httpd is stopped
[*] RESULTS 2886729839_1430491659.dat
[*] Result file: 2886729839_1430491659.dat | Passed tests: 17/56 (30%)
16_1b14f 12_1a2cf 12_2a2cf 2_1f84b 2_2b7a3 5_1e4d2 9_10ee31 7_16a9c 11_1f2e4
9_3a2b7 9_22ee31 20_1e833 12_3a2cf 21_1f822 13_10ad3 14_1eeab 14_1eeab
[*] Cleaning WIVET statistics..
[*] Done
Archiving artifacts
Finished: SUCCESS
```

Figure 5.1: Results of WIVET Jenkins job



```
###########################################################
 Wavsep-stop.sh
###########################################################
[*] Checking if tomcat is running
[*] Service tomcat is running
[*] Stopping service tomcat..
[*] Service tomcat is stopped
[*] Checking if mariadb is running
[*] Service mariadb is running
[*] Stopping service mariadb..
[*] Service mariadb is stopped
[*] Stopping Revokd..
[*] Result for vector: sql 0/142 (0%)
[*] Result for vector: xss 0/98 (0%)
[*] Result for vector: lfi 111/828 (13%)
[*] Result for vector: rfi 0/115 (0%)
[*] Done
Archiving artifacts
Finished: SUCCESS
```

Figure 5.2: Results of WAVSEP Jenkins job

Chapter 6

Conclusion

The goal of this thesis was to study, analyze and compare implementation and vulnerability detection capabilities of selected open source scanners. Based on this analysis, the next step was to propose and implement new features or extend the functionality of already existing ones in open source scanner Revok.

This goal has been successfully achieved and detection capabilities of Revok have been extended. After applying all patches mentioned in section 5 Revok was able to reach **30%** testing coverage in WIVET and **13%** in WAVSEP LFI testing which is considerably better than the previous 0% coverage reached in both suites. Continuous integration environment based on open source tool Jenkins has been created and successfully tested, and all bugs and issues discovered during the progress on this thesis have been already reported back to the Revok community.

6.1 Future development

Revok has been recently open sourced, which allowed contribution also from developers and users outside of Red Hat. That definitely helped a lot with improving functionality of this scanner (either in form of creating issue requests or pull requests). There are still plenty of ways of how to improve vulnerability detection (e.g. implementing detection of non-discovered attack vectors in WIVET C.2) and Revok has the potential to be more used and become more powerful in detecting vulnerabilities in web applications. However, there are open source projects which already provide detection capabilities with much better results. They are usually supported by already existing large communities (development of some of them is funded by sponsors) so it will be very hard for Revok to compete with such mature projects.

Bibliography

[1] *2014 Top Security Tools as Voted by ToolsWatch.org Readers.* URL: https://www.toolswatch.org/2015/01/2014-top-security-tools-as-voted-by-toolswatch-org-readers/ (visited on 05/16/2015).

[2] *Apache License, Version 2.0.* http://www.apache.org/licenses/LICENSE-2.0. cit. 2015-05-16. 2004.

[3] *Automated SQL Injection Detection.* URL: https://www.arneswinnen.net/2013/09/automated-sql-injection-detection/ (visited on 05/16/2015).

[4] Abhishek Kumar Baranwal. "Approaches to detect SQL injection and XSS in web applications". MA thesis. University of British Columbia, 2012.

[5] Tim Bray. *The JavaScript Object Notation (JSON) Data Interchange Format.* RFC 7159. cit. 2015-05-16. RFC Editor, 2014. URL: http://www.rfc-editor.org/rfc/rfc7159.txt.

[6] *Cross-site Scripting.* URL: https://www.owasp.org/index.php/XSS (visited on 05/16/2015).

[7] Isaac Dawson. *LFI test cases throwing: java.lang.IllegalArgumentException: URI has an authority component.* http://code.google.com/p/wavsep/issues/detail?id=8. cit. 2015-05-16.

[8] *GNU Affero General Public License, version 3.* http://www.gnu.org/licenses/agpl-3.0.html. cit. 2015-05-16. 2007.

[9] *GNU General Public License, version 2.* http://www.gnu.org/licenses/gpl-2.0.html. cit. 2015-05-16. 1991.

[10] František Koláček. *Missing target entry point in path_traversal module.* https://github.com/Revok-scanner/revok/issues/44. cit. 2015-05-16.

[11] František Koláček. *Possible fix of bad base html tag handling.* https://github.com/Revok-scanner/revok/pull/40. cit. 2015-05-16.

[12] František Koláček. *Problems with multiple events on single page.* https://github.com/Revok-scanner/revok/issues/43. cit. 2015-05-16.

[13] František Koláček. *Relative links are not handled properly (due to base html tag).* https://github.com/Revok-scanner/revok/issues/39. cit. 2015-05-16.

[14] Tasos Laskos. *LFI test case 37 & similar test cases don't function under linux.* http://code.google.com/p/wavsep/issues/detail?id=10. cit. 2015-05-16.

[15] Mike Mired. *Relative URLs in set content property not resolved properly.* https://github.com/ariya/phantomjs/issues/10610. cit. 2015-05-16.

[16] *MIT License*. http://opensource.org/licenses/MIT. cit. 2015-05-16. 1988.

[17] *Path Traversal*. URL: https://www.owasp.org/index.php/Path_Traversal (visited on 05/16/2015).

[18] Zach Shelby. *Constrained RESTful Environments (CoRE) Link Format*. RFC 6690. cit. 2015-05-16. RFC Editor, 2012. URL: http://www.rfc-editor.org/rfc/rfc6690.txt.

[19] *SQL Injection*. URL: https://www.owasp.org/index.php/SQL_Injection (visited on 05/16/2015).

[20] Revok team. https://github.com/Revok-scanner/revok. cit. 2015-05-16.

[21] *Testing for Local File Inclusion*. URL: https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion (visited on 05/16/2015).

[22] *Testing for Remote File Inclusion*. URL: https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion (visited on 05/16/2015).

[23] Robert Thurlow. *RPC: Remote Procedure Call Protocol Specification Version 2*. RFC 5531. cit. 2015-05-16. RFC Editor, 2009. URL: http://www.rfc-editor.org/rfc/rfc5531.txt.

[24] *Top 10 2013-Top 10 - OWASP*. URL: https://www.owasp.org/index.php/Top_10_2013-Top_10 (visited on 05/16/2015).

[25] *Types of Cross-Site Scripting*. URL: https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting (visited on 05/16/2015).

[26] W3C. *Web Application Vulnerability Security Scanner Framework*. Tech. rep. cit. 2015-05-16. World Wide Web Consortium, 2005.

[27] WWW stránky. *Apache Tomcat*. http://tomcat.apache.org/. cit. 2015-05-16.

[28] WWW stránky. *Jenkins*. https://jenkins-ci.org/. cit. 2015-05-16.

[29] WWW stránky. *jQuery*. https://jquery.com/. cit. 2015-05-16.

[30] WWW stránky. *mitmdump*. https://mitmproxy.org/doc/mitmdump.html. cit. 2015-05-16.

[31] WWW stránky. *Open Source Web Application Security Scanner*. http://w3af.org. cit. 2015-05-16.

[32] WWW stránky. *OWASP Zed Attack Proxy Project*. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. cit. 2015-05-16.

[33] WWW stránky. *PhantomJS*. http://phantomjs.org/. cit. 2015-05-16.

[34] WWW stránky. *Revok*. https://github.com/Revok-scanner/revok. cit. 2015-05-16.

[35] WWW stránky. *SQLMap*. http://sqlmap.org/. cit. 2015-05-16.

[36] WWW stránky. *SQLNinja*. http://sqlninja.sourceforge.net/. cit. 2015-05-16.

[37] WWW stránky. *The Open Web Application Security Project.* https://www.owasp.org. cit. 2015-05-16.

[38] WWW stránky. *The web-application vulnerability scanner.* http://wapiti.sourceforge.net. cit. 2015-05-16.

[39] WWW stránky. *Web Application Vulnerability Scanner Evaluation Project.* https://code.google.com/p/wavsep. cit. 2015-05-16.

[40] WWW stránky. *Web Application Vulnerability Security Scanner Framework.* http://www.arachni-scanner.com. cit. 2015-05-16.

[41] WWW stránky. *Web Input Vector Extractor Teaser.* https://github.com/bedirhan/wivet. cit. 2015-05-16.

[42] WWW stránky. *What is continuous integration (CI).* http://searchsoftwarequality.techtarget.com/definition/continuous-integration. cit. 2015-05-16.

[43] WWW stránky. *What is Web Application Security? - Definition from Techopedia.* http://www.techopedia.com/definition/24377/web-application-security. cit. 2015-05-16.

Appendix A

DVD Contents

The attached DVD contains the source code of the latest stable version of Revok. It contains also scripts used by Jenkins CI tool and job definitions for both Jenkins job in form of XML configuration file. Both testing suites are also included on attached DVD in form of GIT repository.

A.1 Content of the DVD

- *bp-jenkins-scripts/* (Jenkins scripts used in Jobs)
- *bp-revok/* (GIT repository containing scanner Revok)
- *bp-wivet/* (GIT repository containing WIVET test suite)
- *bp-wavsep/* (GIT repository containing WAVSEP test suite)
- *patches/* (All changes stored in form of patches)

Appendix B

Installation and Usage

In this chapter are described steps needed to perform the for basic installation and usage of each tool used in this thesis.

B.1 WIVET

For installation of your local `WIVET` you should follow this steps:

1. `# yum install -y php httpd git`
2. `# git clone https://github.com/fkolacek/bp-wivet.git /var/www/html/wivet`
3. `# restorecon -Rv /var/www/html/`
4. `# chown -R apache:apache /var/www/html/wivet`
5. `# systemctl start httpd`

After performing these steps your WIVET instance should be accessible via address: *http://localhost/wivet* .

B.2 WAVSEP

For installation of your local `WIVET` you should follow this steps:

1. `# yum install -y tomcat git java-1.8.0-openjdk`
2. `# git clone https://github.com/fkolacek/bp-wavsep.git /var/lib/tomcat/webapps/`
3. `# restorecon -Rv /var/lib/tomcat/webapps/`
4. `# chown -R tomcat:tomcat /var/www/html/wivet`
5. `# systemctl start tomcat`

After performing these steps your WAVSEP instance should be accessible via address: *http://localhost:8080/wavsep* .

B.3 Revok

For installation of your local `Revok` instance you will have to install following dependencies and perform some post installation tasks:

1. `# yum install -y tcl java-1.8.0-openjdk readlink git`
2. `$ git clone https://github.com/fkolacek/bp-revok.git`
3. `$ cd bp-revok`
4. `$ ./revokd init`
5. `$ ./revokd start`

After performing these steps your Revok instance should be accessible via address: *http://localhost:3030* .

B.4 Arachni

For installation of your local `Arachni` instance you will have to install following dependencies and perform some post installation tasks:

1. `# yum install -y wget`
2. `$ wget http://downloads.arachni-scanner.com/arachni-1.1-0.5.7-linux-x86_-64.tar.gz`
3. `$ tar xvzf arachni-1.1-0.5.7-linux-x86_64.tar.gz`
4. `$ cd cd arachni-1.1-0.5.7`
5. `$ bin/arachni_web`

After performing these steps your Arachni instance should be accessible via address: *http://localhost:9292* .

B.5 w3af

For installation of your local `w3af` instance you will have to install following dependencies and perform some post installation tasks:

1. `# yum install -y git`
2. `$ git clone --depth 1 https://github.com/andresriancho/w3af.git`
3. `$ cd w3af`
4. `$ ./w3af_gui`

B.6 Wapiti

For installation of your local `Wapiti` instance you will have to install following dependencies and perform some post installation tasks:

1. `# yum install -y wget`
2. `$ wget http://heanet.dl.sourceforge.net/project/wapiti/wapiti/wapiti-2.3.0/wapiti-`
3. `# yum localinstall -y wapiti-2.3.0-1.noarch.rpm`
4. `$ wapiti`

B.7 ZAP

For installation of your local `ZAP` instance you will have to install following dependencies and perform some post installation tasks:

1. `# yum install -y wget`
2. `$ wget http://freefr.dl.sourceforge.net/project/zaproxy/2.4.0/ZAP_2.4.0_-`
   `Linux.tar.gz`
3. `$ tar xvfz ZAP_2.4.0_Linux.tar.gz`
4. `$ ZAP_2.4.0/zap.sh`

Appendix C

Detailed results

C.1 WIVET detailed results

In this section you can find more detailed results of testing scanners' ability to use various attack vectors.

Attack vector	Arachni	Revok	w3af	Wapiti	ZAP
self referencing link	✓	✗	✓	✓	✓
self referencing link with random query string	✓	✗	✓	✓	✓
iframe	✓	✗	✓	✓	✓
302 redirection	✓	✗	✓	✓	✓
html encoded links	✓	✗	✓	✓	✓
protocol relative links	✓	✗	✓	✓	✓
link creation after button click	✓	✗	✗	✓	✓
link href js protocol	✓	✗	✗	✓	✓
div onmouseover window.open	✓	✗	✗	✓	✓
form submit thru select onchange w/ simple alert	✓	✗	✗	✗	✓
form submit button onclick	✓	✗	✗	✓	✓
td onclick window.location.href	✓	✗	✗	✗	✓
link href js protocol window.location w/ alert override	✓	✗	✗	✓	✓
td onmouseout window.location.href	✓	✗	✗	✗	✓
td onmousedown window.location.href	✓	✗	✗	✗	✓
link href jquery	✓	✗	✗	✓	✓
td onmouseup window.location.href	✓	✗	✗	✗	✓
frame created dynamically	✓	✗	✓	✓	✓
iframe created dynamically	✓	✗	✓	✓	✓
a href javascript protocol window.open	✓	✗	✗	✗	✓
tr onclick window.location.href	✓	✗	✗	✗	✓
xhr initiating	✓	✗	✗	✓	✓

link created thru xhr response	✓	✗	✗	✓	✓
tr onmouseout window.location.href	✓	✗	✗	✗	✓
tr onmousedown window.location.href	✓	✗	✗	✗	✓
tr onmouseup window.location.href	✓	✗	✗	✗	✓
form action with javascript protocol set	✓	✗	✗	✗	✓
span onclick window.location	✓	✗	✗	✓	✓
span onmouseout window.location.href	✓	✗	✗	✓	✓
span onmousedown document.location.href	✓	✗	✗	✓	✓
xhr with a busy mode page 1	✓	✗	✗	✗	✓
span onmouseup document.location	✓	✗	✗	✓	✓
heavy js library standard form creation	✓	✗	✗	✗	✓
div onclick window.location.href	✓	✗	✗	✗	✓
div onmouseout window.location.href	✓	✗	✗	✗	✓
div onmousedown window.location.href	✓	✗	✗	✗	✓
div onmouseup window.location.href	✓	✗	✗	✗	✓
li onclick window.location.href	✓	✗	✗	✗	✓
li onmouseout window.location.href	✓	✗	✗	✗	✓
link creation after some time w/ setTimeout	✓	✗	✗	✓	✗
multi-page form with a single path to final destination	✓	✗	✗	✗	✗
link in html comment	✓	✗	✗	✗	✗
relative link in html comment	✓	✗	✗	✗	✗
p onclick window.location.href	✓	✗	✗	✗	✗
p onmouseout window.location.href	✓	✗	✗	✗	✗
p onmousedown window.location.href	✓	✗	✗	✗	✗
p onmouseup window.location.href	✓	✗	✗	✗	✗
li onmousedown window.location.href	✓	✗	✗	✗	✗
li onmouseup window.location.href	✓	✗	✗	✗	✗
radio onclick window.location.href	✓	✗	✗	✗	✗
unattached js function document.location	✓	✗	✗	✓	✗
meta refresh tag	✓	✗	✗	✓	✗
302 redirection link in response body	✓	✗	✓	✓	✗
xhr with a busy mode page 2	✓	✗	✗	✗	✗
link attached to a swf simple button onclick event	✗	✗	✗	✓	✗
link attached to a swf simple button parameterized onclick event	✗	✗	✗	✗	✗
total coverage:	96%	0%	16%	44%	69%

Table C.1: WIVET results

C.2 WIVET detailed results after modification

In this section you can find more detailed results of Revok's ability to use various attack vectors with all patches applied.

Attack vector	Revok
self referencing link	✓
self referencing link with random query string	✓
iframe	✓
302 redirection	✓
html encoded links	✓
protocol relative links	✓
link creation after button click	✗
link href js protocol	✗
div onmouseover window.open	✓
form submit thru select onchange w/ simple alert	✗
form submit button onclick	✓
td onclick window.location.href	✗
link href js protocol window.location w/ alert override	✗
td onmouseout window.location.href	✗
td onmousedown window.location.href	✗
link href jquery	✓
td onmouseup window.location.href	✗
frame created dynamically	✓
iframe created dynamically	✓
a href javascript protocol window.open	✗
tr onclick window.location.href	✗
xhr initiating	✓
link created thru xhr response	✗
tr onmouseout window.location.href	✗
tr onmousedown window.location.href	✗
tr onmouseup window.location.href	✗
form action with javascript protocol set	✗
span onclick window.location	✗
span onmouseout window.location.href	✗
span onmousedown document.location.href	✓
xhr with a busy mode page 1	✗
span onmouseup document.location	✗
heavy js library standard form creation	✗
div onclick window.location.href	✗

div onmouseout window.location.href	✓
div onmousedown window.location.href	✗
div onmouseup window.location.href	✗
li onclick window.location.href	✗
li onmouseout window.location.href	✓
link creation after some time w/ setTimeout	✗
multi-page form with a single path to final destination	✗
link in html comment	✗
relative link in html comment	✗
p onclick window.location.href	✗
p onmouseout window.location.href	✗
p onmousedown window.location.href	✗
p onmouseup window.location.href	✗
li onmousedown window.location.href	✗
li onmouseup window.location.href	✗
radio onclick window.location.href	✗
unattached js function document.location	✗
meta refresh tag	✓
302 redirection link in response body	✗
xhr with a busy mode page 2	✗
link attached to a swf simple button onclick event	✗
link attached to a swf simple button parameterized onclick event	✗
total coverage:	30%

Table C.2: WIVET results with all patches applied