Linköping University | Department of Computer science
Master thesis, 30 ECTS | Informationsteknologi
2017 | LIU-IDA/LITH-EX-A--17/001--SE

A Survey of Automated Tools for Probing Vulnerable Web Applications

Milad Barsomo

Supervisor : Anders Fröberg (Linköpings University) and Ross Tsagalidis (Swedish Armed Forces)
Examiner : Erik Berglund

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

The development of web applications have increased exceedingly in the last few years. Without the concern of security development, these web applications are exposed to a great amount of cyber threats. This thesis provides a survey of automated tools, or so called black box web scanners, that are used to find vulnerabilities, without any internal knowledge, in a web application. The web scanners was evaluated by running them on an vulnerable web application called XVWA and comparing the scanning results with two criteria. First criterion is to see if it is as accurate as stated, and the second criterion is to check if they pass the requirements of NIST for a web scanner of this type. All of the web scanners included in this thesis are open source/free to use. The results of eight different web scanners shows that most of the scanners does not follow the NIST requirements fully, however the majority still performs well. It has also been seen that the newer and most active developed scanners performs the best which is logical. One of the drawn conclusions is that none works perfect or is above all the other scanners.

Contents

List of Figures

List of Tables

1 Introduction

The development of web applications continuously grows as we are going in a technical era where we, as much as possible, want to be connected to the Internet. For example, we have a new concept called "Internet of Things", which in short can be described as the network of physical objects. In other words, this concept allows objects to be connected and controlled remotely for different purposes. Not only that, every day new websites, online games and other web applications are created.

As these applications are rapidly developed without enough concern of the security challenges, we face an increasingly amount of cyber threats. This can be confirmed by simply looking at the latest and biggest data breaches shown in a website called informationisbeautiful [9].

Today, in the Internet, exists several automated tools to generate thousands daily queries to probe the web for vulnerable web applications. In the context of "Internet of Things", this could have severe consequences. For example, consequences could be that the attacker may spy on people through the connected television or take control of a vehicle that have Internet connected computer systems. This thesis will therefore address the automated tools for probing vulnerable web applications without internal knowledge of the system.

1.1 Motivation

To be able to protect against these automated, web vulnerability detection tools, it is important to know how and when they are successful. Therefore, a survey and an evaluation of these tools, to assess how successful they are is a highly important task.

This survey and thesis project has been performed with the interest and guideline of Swedish Armed Forces (SwAF), which is an authority that works under the Swedish parliament and government. SwAF is the main security policy resource for Sweden with the aim to always be ready to support major conflicts in the society, both in Sweden and internationally. [14]

Of course there is a broader interest in doing this thesis work. This survey can be used to prepare web applications in general to be safe from attackers using these web scanners. The results of the scan by the tools can be used to find out which scanners that may be the biggest threat to a certain web application. This information can later on be used for example to test the web application. Depending on how successful the scanner is at finding vulnerabilities, this may give very critical and important information about vulnerabilities that otherwise might not have been detected.

1.2 Aim

The aim of this thesis is to raise awareness about cyber threats in form of finding vulnerabilities in web application. First by doing research about the available probing tools. Then by evaluating these tools to find out how well they perform. The goal is to make it easy for anyone to know which of these tools is dangerous for finding most threats of a web application. Provided this information several actions can be taken to make their web application more secure.

1.3 Research questions

To specify the aim more concretely and with the given information earlier in this paper, the following questions have been identified, that this thesis will emanate from.

1. Which web scanners are most accurate of finding the specified vulnerabilities stated by its description?
2. How successful at completing the requirements, as specified by National Institute of Standards and Technology (NIST) [10], are these web scanners?

The requirements by NIST can be seen in chapter 3.3.

1.4 Delimitations

This thesis work will only focus on scanners that looks for vulnerabilities on web applications without the access of the targets source code, also called black box web scanners. Hence the other type of web scanners, that search for vulnerabilities in the source code, will not be investigated.

There will also be a limit on the amount of scanners that will be investigated. Since there are a great amount of web scanners available, the focus will be of the most popular ones.

Another limitation is the fact that no full version of commercial scanners will be used. There are several reasons behind this choice which is discussed in a chapter 5.2.

2 Theory

This chapter is divided into three main sections which provide a great significant information for this survey. First off, a characterization of several different vulnerabilities are described. This is followed by a section which presents ways of evaluating black box web scanners. Subsequently, a section of related work is described.

2.1 Terminology

In the world of information security there are many terms that may confuse people who are not familiar with this context. This section makes this thesis persistent and easier for the reader to know the definitions of the terms used.

True Positive is when a web scanner correctly reports a vulnerability that exists in the web application. [25]

False positive is a non-existing, but still reported vulnerability by a web scanner. For example if a web scanner reports a XSS vulnerability which really does not exists in the web application [10].

False negative is when a web scanner fails to report a vulnerability that exists in the web application. [10]

True negative is when a web scanner correctly rejects a vulnerability that does not exists in the web application. [25]

2.2 Web scanners

As mentioned in section 1.4, this thesis is focused on a type of web scanners called black box web scanners. The other type, white box web scanners, has been showed to have more disadvantages and is less used. Since white box web scanners analyzes the source code to find vulnerabilities, it has to be created for specific programming languages, which of course limits the use of the scanner. Another drawback is that they often tend to lead to more false positive results. Having several false positives leads to more human analyzation which basically makes the web scanner useless. Looking at an attackers perspective, white box web scanners are completely useless unless the web application is an open source project. It is better for the owners of the web application to know which vulnerabilities an attacker may find and use that knowledge to mitigate the consequences. [12]

2.3 Vulnerabilities

Characterizing web application based vulnerabilities is of course important to get a better understanding on how and why the web scanners looks for them. Hence, this section introduce the many types of vulnerabilities which should be detected according to [10].

Cross Site Request Forgery

One of the most common vulnerabilities that is exploited by attackers is Cross-site request forgery (CSRF). This is because many developers does not have this issue in their mind due the lack of knowledge and the confusion with cross-site scripting. The idea of this attack is to trick the web application to perform a user action without the actual user performing it on purpose. This is done from the attacker by sending a malicious link (which of course does not seem suspicious, e.g. a picture) to the user. When the user enter this link, a HTTP request is sent to the web server of the targeted web application without the knowledge of the user. This HTTP request performs an action which the attacker has specified. However, this attack only works if the user has not properly signed out (e.g. if the user just closed the web application) from the web application. [36]

Cross Site Scripting

Cross-site scripting (XSS) can be defined as malicious JavaScript code input from a user with the intention of stealing confidential information or hijacking user sessions. However, it is not only limited to JavaScript code, the technologies Active X, Flash and others can also be used to for XSS attacks. [41] Most commonly, XSS attacks are used to get access to a cookie, which contains credential information, of an authenticated user [19]. There are different versions of XSS attacks that are referred as stored, reflective and DOM based. [17]

HTTP Response Splitting

The basic idea of this vulnerability is to trick the web application it got two HTTP responses from the web server instead of one, which it should be in normal cases when sending a single HTTP request. The request that the attacker sends contain a CRLF (carriage return line feed) sequence followed by malicious code of the attacker. The CRLF splits the header which makes it possible for the attacker to form own responses (with the malicious code used when the request was sent). Having fully control of the second response, the attacker can cause other vulnerabilities such as XSS. [20]

Improper Error Handling

Error messages can provide useful and detailed information about the underlying system to an attacker if it is not handled in a good way. An web application should only show simple messages about what went wrong and at the same time save necessary information for the developers to fix an error. This could be for example code dumps or stack traces, but should be well hidden for users. [24]

Even if the above problem is solved, developers still have to be careful with the error messages because the attacker may find out the structure of the system. For example if the attacker tries to access a file without having the permission for it, the error message could be "access denied". Of course it is more safer if the attacker don't even know it exists. [24]

Information Leakage

This vulnerability is similar to Improper Error Handling in the sense that an application may reveal sensitive data. There are many examples that cause this vulnerability. One is based on that developers tends often to comment HTML code to make it easier for each other to understand the structure and other important notices. If the comments are done early in the development phase, it is easy to forget to delete the comments before releasing the web application. Since anyone can read the HTML files, an attacker may find sensitive information in these comments. [6]

Insecure Communication

Insecure Communication is a vulnerability that can have very high consequences because it means that the traffic communication (e.g.) between client and server is not encrypted. If an attacker who sniff networks can see this will be able to find out very sensitive information. Depending on the use of the web application, it could be passwords, credit card, pin code that may be leaked. [29]

Insecure Direct Object Reference

Exploiting this vulnerability enables the attacker to manipulate internal implementation object references to get access to other objects instead [18]. A concrete example of this is if a web application allows pictures to uploaded with a specified path, an attacker can specify a path to a sensitive directory (e.g. /etc/passwd) [30].

Malicious File Inclusion

Just as it sounds by the name, this vulnerability allows an attacker to upload remote files with malicious code. Hence it also goes by another name, Remote File Inclusion. In particular, PHP is the most vulnerable language for this attack if the developers does not validate the input. [31]

By uploading a malicious file, an attacker cause a lot of damage both on the server side and client side. Practically the attack could take control of the system or steal client session cookies. [7]

OS Command Injection

This vulnerability is mainly caused due to insufficient input validation. Essentially it allows the attacker to execute and/or inject any command at will if the vulnerability is at place. However, for this to work the attacker must know the underlying language of the application since the commands are being sent as requests to the web server. [37]

SQL Injection

SQL Injection attacks are one of the most common attacks in web applications and classed as one of the highest risks [32]. It occurs when the web application allow an attacker to insert malicious SQL statements into an input field instead of expected input such as normal strings. Consequences could be severe for example destroy the whole database, print out sensitive information from database, log in as other users and many more. [2]

Session Fixation

For an attacker with the purpose of effectively steal a user identity on a web application, stealing the session ID for that user is normally the aim. There are many ways to do this, based on that the user already has authenticated to the application. However Session Fixation is a less common way and it is done before the user even has authenticated. This is possible when the web application does not assign a new session ID during authentication of a user. The attacker can establish a session ID and then trick the user to access the site (e.g. with a malicious URL containing the session ID). Since the attacker know the session ID, it is possible to access the user account. In other words, the user actually logged in the attacker's session. This is the most common way to do a Session Fixation attack, however there are of course more ways to do it depending on the session management by the web application. [21]

Unrestricted URL Access

Pretty much as it sounds, this vulnerability allows attackers to access hidden pages without the need to authenticate to the server. This is possible when there are no access control checks for these pages. It is important to check the access of every sensitive request. For example, a web application may have a hidden page, only for admins called /AdminFunctions.php, but can be accessed by anyone who knows it. [5]

Weak Authentication and Session Management

This vulnerability area is pretty big and includes all aspects on how to handle the authentication and session management. For example the process of changing a password or recovering a password is important to implement correctly. The web application should make sure that there are rules on the password such as minimum characters. The system should also restrict and log the number of failed attempts to authenticate. For the Session Management, the session ID should be protected via SSL. [23]

XML Injection

XML Injections attacks are similar to SQL Injections, but the user sends in a query for XML data. This allow attackers to manipulate the logic of the web application by for example adding a fake user with admin privileges. Attackers can also insert extremely large and legal XML documents to fill up the whole memory and cause a DoS attack. [42]

2.4 Evaluation

Undoubtedly, describing evaluation methods that has been used to decide the quality of a web scanner is of great importance to this survey to be able and deduct how to solve research question two. Another reason is that one should never completely trust these scanners since they do not guarantee that all reported vulnerabilities actually are vulnerabilities [13].

First of all a custom built web application must be used when testing the web scanners. This is because it is practically impossible to compare the results of different web scanners using a web application with no knowledge about the true vulnerabilities in the web application. [12]

Thereon, different metrics can be used to compare the web scanners. For example, only comparing the amount of false positives and the code coverage [12]. Another way is by comparing the vulnerabilities found by the scanners with the actual vulnerabilities that exists on the custom built website (i.e. both false positive and false negatives). The optimal web scanner will report all the correct vulnerabilities, which means that the scanner should not give any false negatives nor false positives. [13]

2.5 Related work

There exist several papers about black box web scanners with different aspects. In [8] eight web scanners were analyzed to find out how effective they are. For their evaluation, a custom web application was built and the result showed that stored XSS and SQL injections are hardly found by the web scanners. Other than that they found the scanners to be promising.

The work in [13] evaluates eleven different scanners. The authors concluded that the web scanners are not good enough since many vulnerabilities are missed and that more research is needed.

Year 2013 an evaluation of six open source web scanners using a vulnerable web application called WackoPicko was conducted. The results showed that the web scanners have a high rate of false negatives. [47]

SecToolMarket is a website that shows the results from an evaluation and comparison of a huge amount of web scanners. Every web scanner is evaluated and assessed in many categories such as features, detection accuracy and adaptability. The web scanners was tested against six type of vulnerabilities but implemented in different ways. [11]

In [12] a black box scanner was created which was compared to three already existing ones. The result showed that their scanner found more vulnerabilities than the other three.

3 Method

The methodology for this thesis presents two stages of how this work was carried out. The first stage contains information about which web scanners were picked, why and how these were chosen. Second stage motivates and describes how the evaluation was performed.

3.1 Choosing web scanners

There are several web scanners available in the web and in the market. The scanners that were picked for analysis is limited to only open sourced ones. All information about these web scanners have either been provided or indirectly agreed and accepted to be published by the vendors (e.g. from other publications).

The web scanners were chosen mainly from the lists of web scanners in SecTools [34] and WebAppSec [35]. These websites provides the most popular web scanning tools. Some scanners has also been suggested by SwAF. Table 3.1 lists all the web scanners that is going to be included in the survey.

Web scanner	License	Version
Grabber	Open source	v0.1
w3af	Open source	v1.6.49
Arachni	Open source	v1.4
Zap	Open source	v2.5.0
Acunetix WVS	Trial edition	v11
Vega	Open source	v1.0
Skipfish	Open source	v2.10
Wapiti	Open source	v2.3.0

Table 3.1: Table of the studied web scanners

Questions to study

The survey aims to answer the question below for each web scanner.

- How is the scanner used?

- According to the description, which vulnerabilities does it find?
- How does it present the result of the scan?

This information was then used as part of the evaluation, where a comparison will be made between how well it works and how it fits the description of the tools.

3.2 The evaluation platform

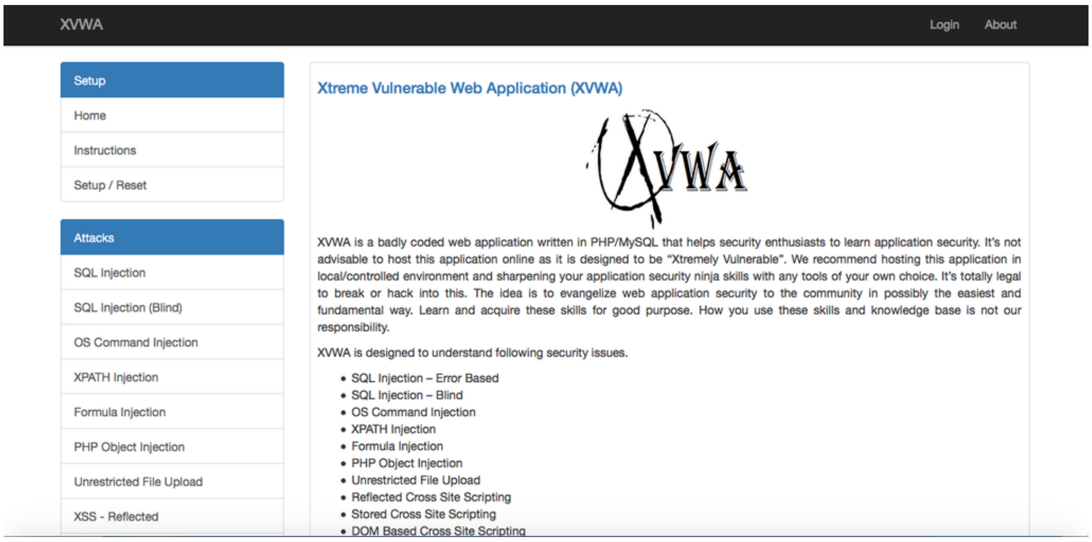
The Open Web Application Security Project (OWASP) is a popular free and open community for raising awareness of software security. OWASP completely non-profitable and does not in any way support commercial more than open source tools. OWASP has many active projects done by thousands of users. [33]

One of the projects, The OWASP Vulnerable Web Applications Directory Project, provides a list of all known vulnerable web applications built for security learning and testing purposes. [26] This list is the starting point of the selection process for choosing an appropriate application for my evaluation.

The selection process is mainly based on three factors, last update, popularity and of course vulnerabilities implemented. First of all, it is important to have a modern and up to date web application otherwise you might question how valid the implementations of the vulnerabilities still are. Secondly, most likely since the more popular a vulnerable web application are, the higher is the probability that scanning tools has been tested on these application and developed to find those. If that is the case, then the results would not be representative of how well they perform. Hence the web application should not be overly popular (e.g if it has appeared several times in similar works). Lastly, the amount of vulnerabilities implemented in the application. All the different sources used in the Theory chapter also help finding most suitable application. For example, as mentioned in [47], the WackoPicko application has been concluded to be outdated, hence it is not selected. Also it is good (but not a necessity) if the application has been developed in either PHP or JavaScript as these are most common. Conclusively, the most suitable application based on these factors is called Xtreme Vulnerable Web Application.

Xtreme Vulnerable Web Application

The Xtreme Vulnerable Web Application (XVWA) is written in PHP and MySQL and consist of the vulnerabilities listed in section 2.3 except response splitting. This is because in PHP version 5.1.2 the vulnerability has been mitigated and made the attack impossible [16]. It also consist of other vulnerabilities that is not listed in this report, but will not be included in the evaluation since they are not part of NIST requirements. The last update of this application was on 20 December, 2015. Figure 3.1 shows how the application look like. [40]



Xtreme Vulnerable Web Application (XVWA)

XVWA is a badly coded web application written in PHP/MySQL that helps security enthusiasts to learn application security. It's not advisable to host this application online as it is designed to be "Xtremely Vulnerable". We recommend hosting this application in local/controlled environment and sharpening your application security ninja skills with any tools of your own choice. It's totally legal to break or hack into this. The idea is to evangelize web application security to the community in possibly the easiest and fundamental way. Learn and acquire these skills for good purpose. How you use these skills and knowledge base is not our responsibility.

XVWA is designed to understand following security issues.

- SQL Injection – Error Based
- SQL Injection – Blind
- OS Command Injection
- XPATH Injection
- Formula Injection
- PHP Object Injection
- Unrestricted File Upload
- Reflected Cross Site Scripting
- Stored Cross Site Scripting
- DOM Based Cross Site Scripting

Figure 3.1: The homepage of the application

3.3 Evaluation of the web scanners

The evaluation consist of running the web scanners on XVWA using a Virtual Machine (VM) to avoid compromising the real system. Continuing with comparing the results to the requirements of a web scanner which is presented in National Institute of Standards and Technology [10]. More details about the requirements and the setup is described in the next section.

In chapter 2.4 was different evaluation methods described. But the main reason that this thesis evaluates towards the NIST requirements is because this was something SwAF was very positive about doing. It also fits well considering that both NIST and SwAF are part of the government (different countries).

Requirements to be analyzed

NIST has a project called Software Assurance Metrics and Tool Evaluation (SAMATE) with the purpose of creating requirements for software assurance tools, in this case web scanners. The goal is to establish a confidence in these tools, which can be seen like a standard. This is done by specifying a functionality (requirement) list of which every scanner should fulfill at minimum. However, this list does not consider factors such as ease of use or implementation details. [10]

The minimum requirements are:

- Detecting any type of the 14 vulnerabilities listed in section 2.3.
- Describe in detail how to exploit the vulnerability
- Have the ability to maintain a logged-in state if the application require authentication.
- Have a low rate of false positives

False positive rate

The formula which was used for calculating the false positive rate (FPR) is:

$$FPR = \frac{FP}{FP + TP} \tag{3.1}$$

Where FP is the number of false positives and TP is the number of true positives (together, all the vulnerabilities that are reported). This means that if a user scans a web application with a scanner that have a FPR of 10%, then the user could expect that 10% of the vulnerabilities reported are false positives.

The setup

The VM used is Oracle VM VirtualBox Manager 5.0 with two guest operating systems. This is because some web scanners only works for linux while other only works for windows. Hence, a Windows 10 and Ubuntu guest operatives system will be used.

4 Results

4.1 Grabber

Grabber is a small web application scanner with the purpose of finding XSS, SQLI, FI and few other vulnerabilities which are not mentioned in this thesis. Since it is a simple and small web scanner, it has no GUI and is executed using command lines. To run this web scanner, simply execute the main python script called grabber.py. Another important note is that Grabber was released as Beta state. [15]

This web scanner was last updated four years ago, 2012. Running this web scanner did not work on XVWA application using the default configuration. Even trying different configurations several errors occurred with the spider.py file. The results is presented in XML format.

4.2 W3af

A popular open source web scanner which has been evaluated several times in other works is w3af (Web Application Attack and Audit Framework). It was founded in 2007 by Andres Riancho and the last stable release happened 7 April 2015, but it is still under active development (with the latest commit on the 17th February). It is also written in python and can be used as GUI application or with command lines. This tool works on all major operating systems, however the latest version (v1.6) has not a working Windows installer and have not been officially tested on. To avoid the hassle of installing this application on windows, the tool was installed and evaluated using the the Ubuntu guest operating system. [43]

W3af uses three main types of plugins called Crawl plugins, Audit plugins and Attack plugins. The Crawl plugin is responsible for finding injection points such as new URL points (e.g. web spider). These injection points is then sent to the Audit plugins which crafts data that is sent in with the purpose of finding vulnerabilities. Lastly is the Attack plugin which is an additional feature which simply try to exploit the vulnerabilities found by the previous plugin. There are several more plugins to use as well. [44]

The web scanner is used by enabling all plugins that wants to be used for the scan. For this setup, I configured to enable the web_spider as the crawl plugin, all the grep plugins and all the relevant audit plugins. This is the recommended setup by their documentation [44]. The relevant audit plugins means all the vulnerabilities to search for, which in this case ideally is the listed ones in 2.3. However the audit plugins does not cover all those vulnerabilities, the missing ones are Insecure Direct Object Reference, Session Fixation and Unrestricted URL Access. That means that the scanner optimally will find ten of the listed vulnerabilities. Worth to note is that there are more than the audit plugins can found other than the listed vulnerabilities, but those plugins has been disabled.

For the evaluation two tables were created. Table 4.1 shows which of the ten listed vulnerabilities was found. Worth to mention is for example the "3 x XSS" which means that the web application has three types of cross site scripting vulnerabilities. The second row simply state if the vulnerability was found or not.

CSRF	3 x XSS	IEH	IL	IC	MFI	CI	2 x SQLI	WASM	XMLI
Yes	Yes, 3/3	Yes	Yes	Yes	Yes	Yes	No	Yes	No

Table 4.1: Vulnerabilities found by w3af

Table 4.2 shows how many false positives and true positives were found, which then is used to calculate the false positive rate with the formula stated in section 3.3.

False Positives	True Positive	False positive rate
10	10	50%

Table 4.2: False positives and true positives found by w3af

W3af have several ways to represent the results of the scanner. The different choices are to show the result directly in console, csv file, html file, text file and xml file. They all show the severeness, vulnerable URL and the description of the vulnerability found. However the HTML file was most detailed since it had more detailed descriptions and also described how to fix the found type of vulnerability. The results does not include a specific explanation of how to exploit the vulnerability on the web application, only how to generally exploit the vulnerability. W3af passes the requirement of being able to maintain a logged in state.

4.3 Arachni

Arachni is an another free, open source web application security framework that can be used for many purposes within security in web applications. It has many similarities to w3af such as it is under development still, works on all major operating system, and have several deployment options such as command line interface (CLI) and WebUI. Arachni recommend to use this scanner in Linux or OS X for best experience, hence the evaluation is performed on the Ubuntu guest operating system. [3] Arachni is very simple to set up and start using. It has three main components, these are checks, plugins, reporters. Checks is similar to the audit plugins for w3af, which is to decide the issues to scan for. As defualt it is set to all. Plugins are used to extend the functionality of the scanner, such as maintaining a logged in state. Reporters simply to control how the results will be presented. [3]

All the checks, plugins and reporters can be seen on the website [4]. But for this evaluation, i excluded the irrelevant vulnerability from the checks. Note that there are no checks for Improper Error Handling, Information Leakage and Insecure Direct Object Reference. The found vulnerabilities is shown in Table 4.3 and the fals positve rate is shown in Table 4.4.

CSRF	3 x XSS	IC	MFI	CI	2 x SQLI	SF	UURLA	WASM	XMLI
No	Yes, 3/3	Yes	Yes	Yes	No	No	Yes	Yes	No

Table 4.3: Vulnerabilities found by Arachni

False Positives	True Positive	False positive rate
1	8	11%

Table 4.4: False positives and true positive found by Arachni

The result of scan is given in AFR format. AFR stands for Arachni Framework Report and is the reference point for all scan results. Using the reporters, this file can then be converted to any of the following formats:

- HTML
- Text
- JSON
- XML
- YAML
- Marshal

Figure 4.1 Results from Arachni scan using the HTML report format.

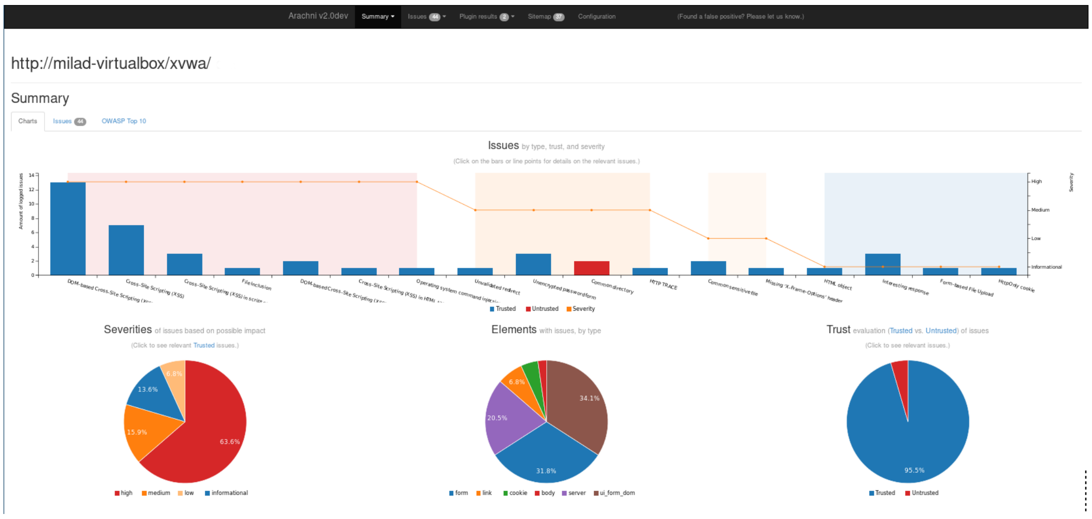

Arachni v2.0dev
Summary
Issues 44
Plugin results 2
Sitemap 37
Configuration
(Found a false positive? Please let us know.)
http://milad-virtualbox/xvwa/
Summary
Charts
Issues 44
OWASP Top 10
Issues by type, trust, and severity
(Click on the bars or line points for details on the relevant issues.)
Amount of logged issues
Severity
High
Medium
Low
Informational
DOM-based Cross-Site Scripting (XSS)
Cross-Site Scripting (XSS)
Cross-Site Scripting (XSS) in script
File Inclusion
DOM-based Cross-Site Scripting (XSS)
Cross-Site Scripting (XSS) in HTML
Operating system command injection
Unvalidated redirect
Unencrypted password form
Common directory
HTTP TRACE
Common sensitive file
Missing 'X-Frame-Options' header
HTML object
Interesting response
Form-based File Upload
HttpOnly cookie
Trusted
Untrusted
Severity
Severities of issues based on possible impact
(Click to see relevant Trusted issues.)
63.6%
15.9%
13.6%
6.8%
high
medium
low
informational
Elements with issues, by type
34.1%
31.8%
20.5%
6.8%
form
link
cookie
body
server
ui_form_dom
Trust evaluation (Trusted vs. Untrusted) of issues
(Click to see relevant issues.)
95.5%
Trusted
Untrusted

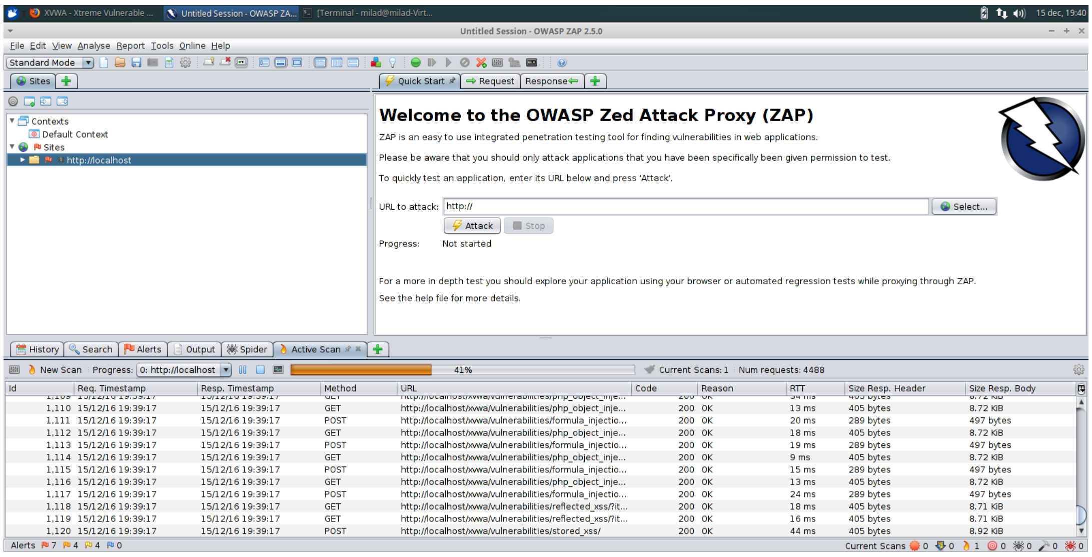
Figure 4.1: Result of Arachni scan in HTML format

The results are represented both as a summary and in more detail. For every issue it find, it present the general description of the vulnerability, general suggestions of mitigation, affected page, injection seed and proof.

4.4 ZAP

One of the worlds most popular security tool is the OWASP Zed Attack Proxy (ZAP) as it was voted to be best free/open source Security tool 2015 and 2013 by ToolsWatch readers [22]. Zap is a cross platform tool that is actively maintained by volunteers from all around the world [28]. It requires Java 7 or higher to run on Windows and Linux.

As stated above, Zap has several features, however for this thesis we are only interested in how well automated scanners can perform. That is why the main feature used is the "Spider scan" and the "Active scanner". The Spider scan crawl the application to find the pages of the web application. Then it is possible to use the Active scanner, which uses known attacks on all the pages the spider has found to expose vulnerabilities. [27]


XVWA - Xtreme Vulnerable ...
Untitled Session - OWASP ZA...
[Terminal - milad@milad-Virt...
15 dec, 19:40
Untitled Session - OWASP ZAP 2.5.0
File Edit View Analyse Report Tools Online Help
Standard Mode
Sites
Quick Start
Request
Response
Contexts
Default Context
Sites
http://localhost
History
Search
Alerts
Output
Spider
Active Scan
New Scan
Attack
Stop
Select...

Welcome to the OWASP Zed Attack Proxy (ZAP)

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

To quickly test an application, enter its URL below and press 'Attack'.

URL to attack: http://

Progress: Not started

For a more in depth test you should explore your application using your browser or automated regression tests while proxying through ZAP.

See the help file for more details.

```
Progress: 0: http://localhost   41%   Current Scans: 1   Num requests: 4488

Id     Req. Timestamp      Resp. Timestamp     Method  URL                                                     Code  Reason  RTT    Size Resp. Header  Size Resp. Body
1,110  15/12/16 19:39:17   15/12/16 19:39:17   GET     http://localhost/xvwa/vulnerabilities/php_object_inje...  200  OK      13 ms  405 bytes          8.72 KiB
1,111  15/12/16 19:39:17   15/12/16 19:39:17   POST    http://localhost/xvwa/vulnerabilities/formula_injectio... 200  OK      20 ms  289 bytes          497 bytes
1,112  15/12/16 19:39:17   15/12/16 19:39:17   GET     http://localhost/xvwa/vulnerabilities/php_object_inje...  200  OK      18 ms  405 bytes          8.72 KiB
1,113  15/12/16 19:39:17   15/12/16 19:39:17   POST    http://localhost/xvwa/vulnerabilities/formula_injectio... 200  OK      19 ms  289 bytes          497 bytes
1,114  15/12/16 19:39:17   15/12/16 19:39:17   GET     http://localhost/xvwa/vulnerabilities/php_object_inje...  200  OK      9 ms   405 bytes          8.72 KiB
1,115  15/12/16 19:39:17   15/12/16 19:39:17   POST    http://localhost/xvwa/vulnerabilities/formula_injectio... 200  OK      15 ms  289 bytes          497 bytes
1,116  15/12/16 19:39:17   15/12/16 19:39:17   GET     http://localhost/xvwa/vulnerabilities/php_object_inje...  200  OK      13 ms  405 bytes          8.72 KiB
1,117  15/12/16 19:39:17   15/12/16 19:39:17   POST    http://localhost/xvwa/vulnerabilities/formula_injectio... 200  OK      24 ms  289 bytes          497 bytes
1,118  15/12/16 19:39:17   15/12/16 19:39:17   GET     http://localhost/xvwa/vulnerabilities/reflected_xss/?it... 200 OK      18 ms  405 bytes          8.71 KiB
1,119  15/12/16 19:39:17   15/12/16 19:39:17   GET     http://localhost/xvwa/vulnerabilities/reflected_xss/?it... 200 OK      16 ms  405 bytes          8.71 KiB
1,120  15/12/16 19:39:17   15/12/16 19:39:17   POST    http://localhost/xvwa/vulnerabilities/stored_xss/        200  OK      44 ms  405 bytes          8.92 KiB

Alerts 7 4 4 0                                                     Current Scans 0 0 1 0 0 0 0
```

Figure 4.2: Picture of Zap while the active scan is running

The attacks for the active scanner is defined as "Active scanner rules". These rules aim to find several vulnerabilities, but not all the vulnerabilities listed in this thesis are searched for. This scanner can not find Insecure Direct Object Reference vulnerabilities. [46]

The results of the scan is seen in Table 4.5 and Table 4.6 show the false positive rate. ZAP also have the ability to maintain a logged-in state.

CSRF	XSS	IEH	IL	IC	MFI	CI	SQLI	SF	UURLA	WASM	XMLI
Yes	2/3	Yes	Yes	No	No	Yes	1/2	No	Yes	Yes	No

Table 4.5: Vulnerabilities found by ZAP

Note that ZAP could find the reflected and stored XSS vulnerabilities, but not the dom based XSS vulnerability. Similar with the SQLI, it could find the blind based SQL Injection, but not the error based SQL Injection vulnerability.

False Positives	True Positive	False positive rate
1	9	10%

Table 4.6: False positives and true positive found by ZAP

ZAP reports all vulnerabilities found in its UI under the "Alert" tab. Similar to the previous scanners, the report contains a general description of the vulnerability, the risk classification, the targeted URL, the attack and the general solution.

4.5 Acunetix WVS Trial Edition

Acunetix Web Vulnerability Scanner (WVS) is a commercial tool only available for Windows. However they do offer a 14 day trial edition which contains some limitations. For example, the trial edition will not provide the exact location for the vulnerabilities it finds. [1]

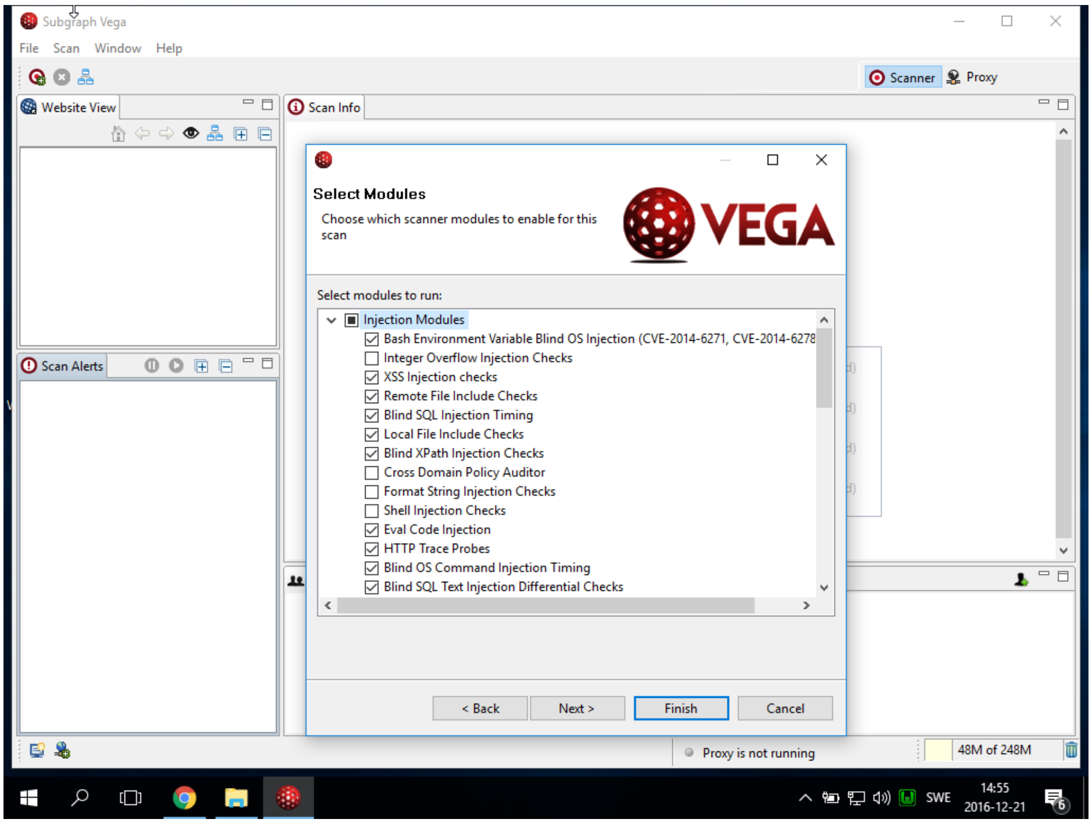
Acunetix WVS has a very friendly user interface which makes it easy to start a scan. All that needs to be done is setting a target and choose if authentication is needed. There is also plenty of advanced options for more advanced users. For this thesis, the only configurations that was made, was to crawl the address and its sub directories only and to scan for vulnerabilities listed by NIST (which coincidentally is an option).

The scan found all the vulnerabilities, however as stated above, the exact URL and attack information is not provided. Hence it is impossible to find out the false positive rate. Otherwise it has a very detailed report which can be exported in several formats. Similarly to the previous scanners, it describe the vulnerability in general, the impact of the vulnerability, how to fix, classification, and gives web references. The "Attack details" and "HTTP request" is missing for the Trial edition.

4.6 Vega

Vega is an another free to use and open source security application which consist of two features. These features are an automated scanner and can be used as an intercepting proxy. It is a cross platform application that requires Java to run as it comes with a GUI. [38]

The GUI makes the scanner easy to use. After choosing the target URL the user may choose which "Modules" to enable for the scan. These modules contains the vulnerabilities the user want to scan for, see Figure 4.3 as example. Afterwards, the user may choose if Vega will authenticate to the target URL.



Select Modules

Choose which scanner modules to enable for this scan

Select modules to run:

- ▣ Injection Modules
  - ☑ Bash Environment Variable Blind OS Injection (CVE-2014-6271, CVE-2014-6278
  - ☐ Integer Overflow Injection Checks
  - ☑ XSS Injection checks
  - ☑ Remote File Include Checks
  - ☑ Blind SQL Injection Timing
  - ☑ Local File Include Checks
  - ☑ Blind XPath Injection Checks
  - ☐ Cross Domain Policy Auditor
  - ☐ Format String Injection Checks
  - ☐ Shell Injection Checks
  - ☑ Eval Code Injection
  - ☑ HTTP Trace Probes
  - ☑ Blind OS Command Injection Timing
  - ☑ Blind SQL Text Injection Differential Checks

Figure 4.3: Screen shot of some of the modules in Vega

The modules in Vega are missing some vulnerabilities relevant to this thesis. These are Cross Site Request Forgery, Insecure Direct Object Reference, Session Fixation and Unrestricted URL Access, all the other vulnerabilities are included in the modules. The scan was run using the Windows guest operating system. Table 4.7 shows which vulnerabilities were found. Of the XSS vulnerabilities, it did not find the DOM based one. Table 4.8 shows the false positive rate based on one false positive and eight true positive that was found.

XSS	IEH	IL	IC	MFI	CI	2 x SQLI	WASM	XMLI
2/3	Yes	Yes	Yes	Yes	Yes	No	Yes	No

Table 4.7: Vulnerabilities found by Vega

False Positives	True Positive	False positive rate
1	8	11%

Table 4.8: False positives and true positive found by Vega

Vega does not have as detailed or "fancy" description as seen with previous scanners in this theses. However all the necessary information is there. The vulnerabilities are reported in the GUI and is prioritized based on the risk classifications. It does give a discussion of the vulnerability, which is basically a general description, the request it sent and response (the attack proof), possible impacts, remediation and sometimes references with additional information.

4.7 Skipfish

Skipfish is a console based web application security tool mainly used in Linux. Skipfish is also free and open source which is written C. The latest release was Dec 4, 2012 and has not been updated in 4 years. [45]

Skipfish is easy to use even if its with command lines. The only mandatory flag is -o or –output, it specifies the directory of the resulting output. However there are several optional flags to use, but the most important ones for this thesis are -I -A and -X. -I or –include flag is a crawl scope option, it tells skipfish to only crawl URL that matches a certain string. In this case, -I /xvwa, was used. -A or –auth is to be used for users that want skipfish to authenticate to the web application. Lastly the -X or –exclude flag which is used to not crawl a certain URL or directory. Here it was used as -X /logout to stop skipfish from logging out.

The relevant vulnerabilities that skipfish can find are Cross Site Request Forgery, Command Injections, Blind SQL Injection, XML Injections, Cross site scripting (non DDOM based), File Inclusion, Weak Authentication Management, Unrestricted URL Access and Insecure Communications [45]. As usual, Table 4.9 shows which vulnerabilities the scanner found and Table 4.10 shows the False Positive Rate

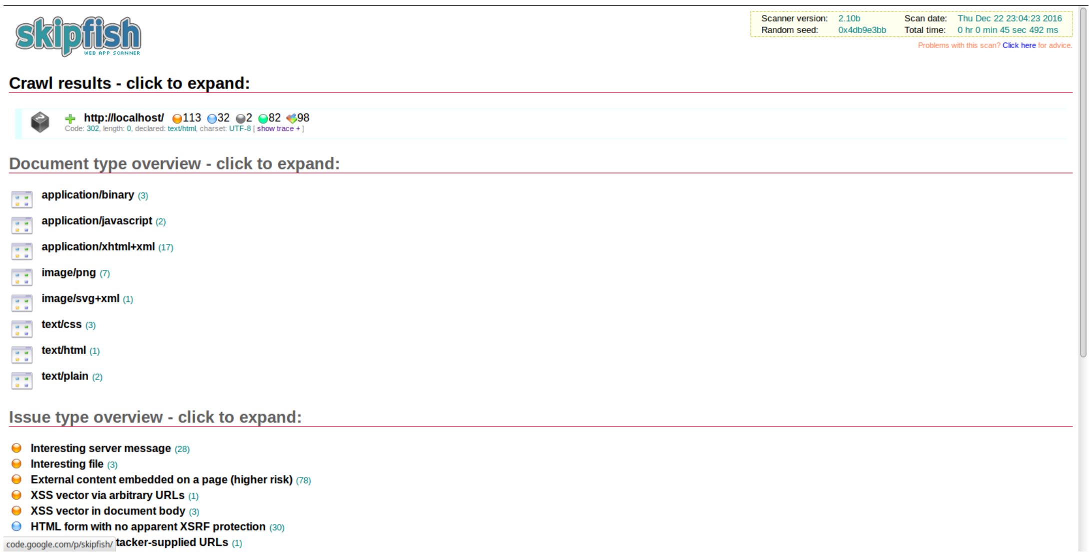
CSRF	XSS	IC	MFI	CI	SQLI	UURLA	WASM	XMLI
Yes	1/2	No	Yes	No	No	Yes	Yes	No

Table 4.9: Vulnerabilities found by Skipfish

False Positives	True Positive	False positive rate
2	5	29%

Table 4.10: False positives and true positive found by Skipfish

Skipfish creates an HTML file for the results of the scan in the directory specified by the -o flag as mentioned two paragraphs above. This HTML page is very simple and does not include much details about vulnerabilities. The only information about the vulnerabilities is which and how an URL was attacked and the trace. This information is valuable of course but should not be classified as detailed information. Picture 4.4 shows the simple HTML page of the result.



Scanner version: 2.10b   Scan date: Thu Dec 22 23:04:23 2016
Random seed: 0x4db9e3bb   Total time: 0 hr 0 min 45 sec 492 ms

Problems with this scan? Click here for advice.

Crawl results - click to expand:

http://localhost/ 113 32 2 82 98
Code: 302, length: 0, declared: text/html, charset: UTF-8 [show trace +]

Document type overview - click to expand:

application/binary (3)
application/javascript (2)
application/xhtml+xml (17)
image/png (7)
image/svg+xml (1)
text/css (3)
text/html (1)
text/plain (2)

Issue type overview - click to expand:

Interesting server message (28)
Interesting file (3)
External content embedded on a page (higher risk) (78)
XSS vector via arbitrary URLs (1)
XSS vector in document body (3)
HTML form with no apparent XSRF protection (30)
code.google.com/p/skipfish/ tacker-supplied URLs (1)

Figure 4.4: Screen shot of the resulting HTML page by Skipfish scan

4.8 Wapiti

Just as Skipfish, Wapiti is also a console based web application scanner mainly used in Linux. However it is a newer scanner, the latest version (2.3.0) was released in October year 2013. Wapiti is open source and free to use as well but is written in Python, hence Python 2.7 or later is required. [39]

The used flags for the scan using Wapiti is almost the same as Skipfish. These flags are -o, -a and -x which has the same meaning as the flags used did for Skipfish. However Wapiti has as default to crawl only the sub directories of the target URL, hence the -I flag was not needed.

Wapiti can detect the following, relevant for this thesis, vulnerabilities:

- Malicious File Inclusion
- SQL Injection
- XML Injection
- Cross Site Scripting (not DOM based)
- Session Fixation
- Command Injection
- Information Leakage [39]

The results of the scan is presented in Table 4.11 and the false positive rate in Table 4.12.

Wapiti can generate the resulting report in four different format, HTML, XML, JSON and TXT [39]. Having the report in the HTML file shows same kind of information like in most of the previous scanners. That is General description, the URL and attack, the HTTP Request,

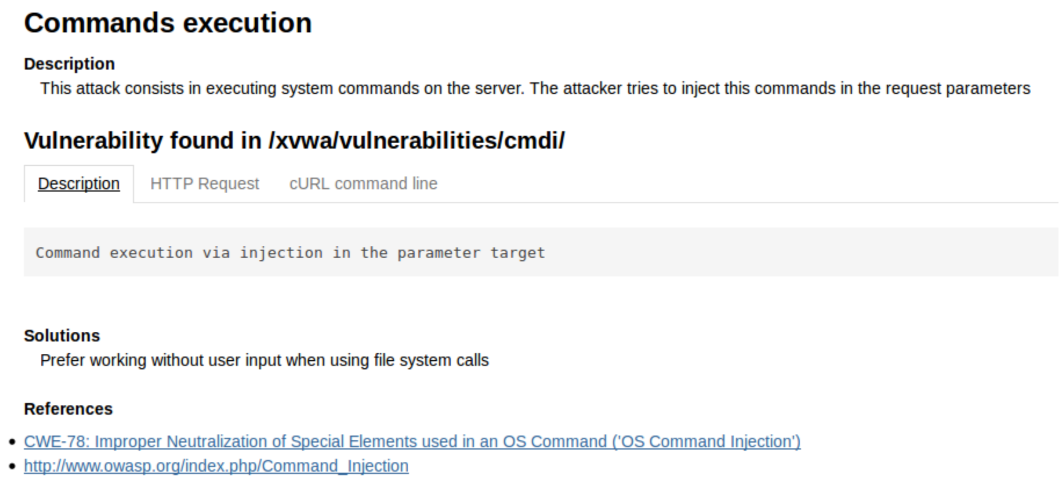
XSS	IL	MFI	CI	SQLI	SF	XMLI
2/2	No	Yes	Yes	0/2	No	No

Table 4.11: Vulnerabilities found by Wapiti

False Positives	True Positive	False positive rate
0	4	0%

Table 4.12: False positives and true positive found by Wapiti

general solution and References for more information. As an example, Picture 4.5 shows the information given from finding the Command Injection vulnerability.



Commands execution

Description

This attack consists in executing system commands on the server. The attacker tries to inject this commands in the request parameters

Vulnerability found in /xvwa/vulnerabilities/cmdi/

```
Command execution via injection in the parameter target
```

Solutions

Prefer working without user input when using file system calls

References

- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
- http://www.owasp.org/index.php/Command_Injection

Figure 4.5: Picture showing the Command Injection vulnerability which Wapiti found.

4.9 Summary of the scans

To summarize the scanning results, eight automated black box scanners was used on XVWA web application. Table 4.13 shows how many of the vulnerabilities that were not reported (false negatives), the false positive rate (FPR), if the reports of the scans contains detailed description and if the scanner is able to maintain a logged-in state.

Web scanner	False negative	FPR	Detailed description	Can log in?
Grabber	6/6	-	-	No
w3af	3/13	50%	Yes	Yes
Arachni	5/13	11%	Yes	Yes
ZAP	6/15	10%	Yes	Yes
Acunetix WVS Trial	0/16	-	Semi	Yes
Vega	4/12	11%	Yes	Yes
Skipfish	5/10	29%	Semi	Yes
Wapiti	5/9	0%	Yes	Yes

Table 4.13: A summary of the results

To make it more clear what the second column also means, the results for Vega will be used as example. Vega failed to report 4 out of 12 vulnerabilities which should have been found according to the scanner. This also means that Vega correctly reported 12-4=8 vulnerabilities (true positives).

5 Discussion

The goal of this chapter is to discuss of what I believe the most important factors of this thesis. To do this in a structured way, the chapter is divided into three main sections, Results, Method and Wider context.

5.1 Results

This first section, results, will discuss any interesting information that stands out from the Result chapter and discuss the results in overall to the research questions stated in introduction section 1.3.

Maintenance

One thing seen that is pretty obvious is that the actively maintained web scanners performs better than inactively maintained scanners. For example we see that Grabber which is in "beta" since four years ago could not find anything and gave many error messages during the scan. Of course applications in beta are not expected to work perfectly which could be another reason it did not work for this web application. (Maybe also it was stopped being developed because it was seen to not perform well?). While we can also see that the active ones such as Arachni and Acunetix WVS did perform pretty good. Though we do not have information about the false positive rate for Acunetix WVS, we can see that the vulnerabilities checks are many and did report all of them. Acunetix which is a commercial scanner and probably the most maintained one, can be expected to do really well. Another example, Skipfish, which was also last updated four years ago, has a high false negative rate, a pretty high false positive rate and not very detailed report also shows this fact.

We know that web application technologies constantly evolves. This means that if web scanners are not maintained enough to follow the trends then it should be expected to not get the best results of those scanners. This is most likely the biggest reasons of their performance. Four years ago, they could have been the leading web scanners in the market, since all these web scanners are the most popular ones out there (excluding the commercial ones).

w3af's high false positive rate

The false positive rate for w3af is surprisingly high. For some reason it reported several vulnerabilities which were not true. Maybe it has something to do with the web application it was tested on, however it is doubtful considering the results of other scanners. Maybe it had something to do with the scanner itself and its configurations, specifically the plugins. It is impossible to know the reason without going into the details in the source code of the scanner, however it is out of the scope for this thesis.

A web scanner with high false positive rate is not good at all. In fact, it could make it more expensive for web developers since they would have to spend more time going through all of reported vulnerabilities. However w3af is generally a good web scanner, we see that it found most vulnerabilities with a low false negative rate. Since w3af has a lot of configuration options to tweak, I believe that with more testing of this scanner, it is possible to find a great combination to make this scanner report less false positives. I would definitely not rule out using this web scanner.

Wapiti's zero false positive rate

I believe it is worth mentioning that Wapiti's 0% false positive rate is probably a bit misleading. The reason for that is because it found only a few vulnerabilities and have the highest false negative rate among the web scanners. Logically if a web scanner can only find few vulnerabilities then the chance of the reported vulnerabilities being a false positive is low.

Wapiti was also last updated four years ago, and as discussed in the maintenance section, this could be a rather good explanation for its high false negative rate. This does not necessarily mean that the web scanner is bad. Since the false positive rate is zero, we do know that the reported vulnerabilities are all true. This gives the idea of using Wapiti together with another web scanner to confirm which vulnerabilities are true without having to double check.

Overall discussion for the first research question

For convenience sake we repeat the first research question:

- Which web scanners are most accurate of finding the specified vulnerabilities stated by its description?

This thesis has shown that none of the web scanners are perfect which is as expected. The scanner with the least false negatives has been most accurate of finding the vulnerabilities stated by the scanners description and features, on XVWA. As seen in Table 4.13, Acunetix WVS followed by w3af has the best false negative rates. However I do believe it is important to mention that even though most of the scanners did not find all the vulnerabilities stated by its description, does not mean that the description was a lie. A web scanner can be built in many different ways, which can lead to being optimal for different kind of web applications and practically useless for other kind of web applications.

This is why it is impossible to say with 100% certainty that Acunetix and w3af is absolutely the best ones, to answer this question. To be that sure, all these scanners needs to be tested on several different web applications. On the positive note it can be said that since they still have proven to result in a low rate false negatives it can be a good first scanner to start using. But let us not forget the high false positive ratio that w3af got which may make w3af less appealing. However it is not relevant for this first research question.

Overall discussion for the second research question

The second research question is as follows:

- How successful at completing the requirements, as specified in [10] by National Institute of Standards and Technology (NIST), are these web scanners?

The exact requirements can be seen in section 3.3. Firstly we can already conclude, from the results, that none of the web scanners have successfully completing all the requirements. Most of the web scanners falls on the first requirement. Otherwise the majority of the scanners does complete the remaining ones.

Several reasons could be behind why the web scanners failed. I believe that some of the vulnerabilities listed in the NIST requirements are outdated. That is because these requirements were created in 2008. While Web technologies have developed very quickly since then and most likely have included mitigation for some of the vulnerabilities. There could of course been other reasons too. Simply that the creators did not want to follow these requirements. Some scanners could possibly only want to focus and optimize on the most harmful threats, some maybe were created at first to be a personal tool but then decided to make it for the public. As most of the scanners are open source and free to use, they don't have an obligation to make the web scanner as NIST want the standard to be.

Interesting note is that Acunetix WVS Trial edition was the only one to complete the first requirement which all the other web scanners failed on. However it did fail to complete two other requirements, the ones about detailed report and false positive rate. This is done on purpose by the owners to attract the users to buy the full versions by not giving the user all the data.

Below we discuss each requirement and which scanners passed or failed. But since Grabber did not work on XVWA we exclude it from the discussions.

Identification of the vulnerabilities

Every open source scanners used in this thesis fails this requirement. This is because all of them left out one or more of the vulnerabilities that is required to be scanned for. For example we see that most of them does not look for Insecure Direct Object Reference vulnerability, hence failing the requirement. As mentioned earlier, Acunetix WVS Trial edition was the only scanner that completed this requirement.

Detailed descriptions

All web scanners except Acunetix WVS and Skipfish completed this requirement great. Acunetix failed solely on the fact that it is a trial version, otherwise it would have passed. While Skipfish made the choice to not focus about providing information about the vulnerabilities, instead only provided the attack information. Basically it means that the aim was for users that already have some knowledge in web security.

Maintaining a logged-in-state

Not much to discuss here, every web scanner had the ability to authenticate to the application and maintaining a logged-in-state during the scan.

Acceptably low false positive rate

In the NIST specification the "acceptably low false positive rate" was never specified to an exact number. But based on the results on the thesis, w3af and Skipfish was the ones that points out in a negative way. Skipfish could possibly be on the limit of acceptable since it is still between one fourth and one third of the reported vulnerabilities. w3af's 50% is definitely not acceptable based on my opinion.

5.2 Method

This section discuss the many different aspects of how the method may have affected the results, what could have been done differently and other aspects important to consider for this thesis and any future work that may be done.

Commercial scanners

One important aspect is that none of the chosen web scanners are commercial except the free/trial version. The initial aim was to include even commercial scanners and there were many different reasons why these were not included in the end. For example, many would only agree if it was an anonymous evaluation, others denied because there are other similar work already done and ongoing and some simply did not want to be a part of it. We may discuss why it was not done anonymously to include the commercial scanners. The answer to that is because this thesis is supposed to show how well different scanners pass the requirement by NIST and to show different strengths and weaknesses in the different web scanners. This can not be done anonymously.

Performing an evaluation of the commercial scanners would be very interesting as these are presumably the most effective web scanners. Because normally an open source tool is not nearly maintained as much as commercial tools. But open source tools have an enormous advantage of being free which attracts many people.

As with most free versions of a commercial tool, they do not provide all the functionality of the full version of the tool. This raises questions about how representative these results are for the the full versions of the commercial web scanner. However the point is not to view them as representative. Since the full version could be very different and have much different result. That is important to keep in mind. The free versions could be though of as a separate web scanner.

Choosing the evaluation platform

Choosing a platform for evaluation of the web scanners is probably the most important factor of this thesis. The ideal platform would be a web application using the most recent technology that has implemented all the stated vulnerabilities in section 2.3 in a realistic way. There exists platforms that are implemented in a realistic way but only with few vulnerabilities hence I did not choose them. Another reason is that several of these applications was last updated years ago which does not use the recent technology.

Another thing to bear in mind is that any open vulnerable web application may have already been tested against which will question the validity of the results. Because if that is the case, the web scanners may have adapted to find these exact vulnerabilities in the web application to optimize the results. An own created vulnerable application would most likely give more valid results for the evaluation, but this would require much more extra resources, specially time.

Maybe the most obvious way to get more valid results is to test the web scanners on different applications. This of course would need more resources unless testing only few web scanners.

To summarize, there are several things that have affected the results and could be done differently. This is why I believe how to choose the platform to perform the evaluations on is most important to get as valid results as possible.

Configuration of the web scanners

All of the web scanners used for this thesis can be configured in several ways which probably can optimize the scanning results. However only the default or slightly changed configuration has been used in this thesis. One reasons is because it would take a great deal amount of time to go through all the configuration options to try find the optimal settings. Of course it is also interesting to see how the default configurations work because a user, such a developer or a tester, may not want to spend a lot of time to find the optimal settings. Therefore by having (almost) the default configuration, may show how effective the scanner is without the need of changing variables.

The obvious downside with this is that, the results may not fully represent how good the web scanner actually is. This part is definitely something that could have been done differently. One could focus more in depth on few scanners to find the optimal settings to run on a web application.

5.3 The work in a wider context

This thesis can of course be discussed in both ethical and societal perspectives. Scanning web applications can obviously be used with malicious intention. However it can be countered by any company working with web applications.

Ethical analysis

As stated in the introduction of this thesis many security challenges needs to be taken in consideration while developing web applications. If not, attackers will have plenty of opportunities to attack the application. Since this thesis has shown that there are several web application scanners that finds vulnerabilities, this could very well be used as a weapon on the attackers perspective. If the attacker is successful in the attempt of finding non false positive vulnerabilities using these scanners it could lead to several consequences. The severeness of the consequences completely depends on the purpose of the web application.

Privacy issues

Probably the most severe consequence could be personal privacy leaks. Many types of application may store sensitive information. Some examples of web applications that has this kind of information could be health care and bank type of applications. An attacker can find a vulnerability which enables to access this information by using these web scanners (assuming the application was not developed secure). This would be absolutely terrible for the company because the users would feel exposed which may lead to the society losing trust in the company.

Economic issues

Another underestimated consequence is the fact that it would cost the company a lot of money to repair the issues to eventually go back online. There are many reasons behind this with the most obvious one that they probably need to hire security experts to analyse the complete application resolve all the eventual vulnerabilities. It will not be enough to only fix the vulnerability that was exploited because of course it is not affordable to get hacked again. Depending on how complex the system is, it may require a lot of time which naturally means that higher economic expenses. It is also important to consider the psychological aspects.

6 Conclusion

The purpose of this thesis was to evaluate popular web vulnerability application scanners, in specific the so called black box scanners, that does not have information about the source code. This is important because today web applications are rapidly developed with not much concern of security, hence the amount of cyber threats also increases. This research has showed that there exists several tools, which performs well in finding vulnerabilities. This information can be used by developers to improve the security in a web application as it reports the attack itself. Therefore it is important to raise awareness of cyber threats.

The research conducted was to see how reliable these web scanners are by performing the scan on a vulnerable web application called Xtreme Vulnerable Web Application. The evaluation was compared to the standard specification for a web scanner by NIST. It was concluded that most web scanners performs well in regards of finding the vulnerabilities stated by their website. Arachni, Zap and Vega got the best results overall, however none of the web scanners fully passed the standard requirement specified by NIST.

6.1 Future work

There are many ideas of possible future works in this area, some already suggested in the discussion chapter. As mentioned, a similar evaluation using commercial scanners could be done if permission is granted by the vendors. This would be very interesting since they are more maintained than the open sourced scanners. Or an evaluation like in this thesis, but on several vulnerable web applications could be performed to get even more reliable results. However the drawback would be that less scanners would be evaluated if the same amount of time was used as in this thesis.

Another idea could to evaluate the true negative rate of these scanners. This has not been done in this thesis. To evaluate true negatives, a web application must be created by implementing vulnerable "alike" features while in fact they are not real vulnerabilities. Then running the scanners to see if it will be reported as vulnerability (which would be a false positive) or if it correctly does not report it as a vulnerability (true negative).

The most interesting future work idea, in my opinion, would be if an evaluation was tested on a more realistic web application. One idea is if the researcher could ask someone or a couple of people that knows how to make web applications, to make one. Then the researcher could use this application to run the scanners on. The reason it should not be the researcher itself that build the web application, is because the awareness of the cyber threats will already be in mind since the research is about it. Doing this way will will definitely give more representative results since the potential vulnerabilities would not have been implemented on purpose. Also the fact that nobody else has access to this web application, the scanners could not have already been tested at.

The evaluation would have to be different than its done in this thesis. Since the vulnerabilities are not known or implemented on purpose, it is impossible to compare with NIST specification. Therefore the ideal method would be to, first, hire security professionals to manually find all the vulnerabilities, then running the automated scanners on the web application. Henceforth, the researcher can find all the false negatives and false positives.

This however would require a lot of resources such as time and money (to pay the people that build web application and the security professionals), depending on the size of the web application, this work could take longer than a year.

Bibliography

[1] Acunetix. *Download Acunetix 14 Day Trial*. https://www.acunetix.com/vulnerability-scanner/download/. Accessed: 2016-12-19.

[2] Chris Anley. *Advanced SQL injection in SQL server applications*. 2002.

[3] Arachni. *Arachni - Web Application Security Scanner Framework*. http://www.arachni-scanner.com/. Accessed: 2016-08-24.

[4] Arachni. *Checks*. http://www.arachni-scanner.com/features/framework/#Checks. Accessed: 2016-08-24.

[5] Hasty Atashzar, Atefeh Torkaman, Marjan Bahrololum, and Mohammad H Tadayon. "A survey on web application vulnerabilities and countermeasures". In: *Computer Sciences and Convergence Information Technology (ICCIT), 2011 6th International Conference on*. IEEE. 2011, pp. 647–652.

[6] Robert Auger. *Information Leakage*. http://projects.webappsec.org/w/page/13246936/Information%20Leakage. Accessed: 2016-02-03.

[7] Robert Auger. *Remote File Inclusion*. http://projects.webappsec.org/w/page/13246955/Remote%20File%20Inclusion. Accessed: 2016-02-05.

[8] 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*. IEEE. 2010, pp. 332–345.

[9] Information is Beautiful. *World's Biggest Data Breaches*. http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/. Accessed: 2017-01-27.

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

[11] Shay Chen. *The Web Application Vulnerability Scanners Benchmark*. http://sectooladdict.blogspot.jp/2014/02/wavsep-web-application-scanner.html?m=1. Accessed: 2016-03-23.

[12] Adam Doupé, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. "Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner." In: *USENIX Security Symposium*. 2012, pp. 523–538.

[13] Adam Doupé, Marco Cova, and Giovanni Vigna. "Why Johnny can't pentest: An analysis of black-box web vulnerability scanners". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2010, pp. 111–131.

[14] Swedish Armed Forces. *About the Swedish Armed Forces*. `http://www.forsvarsmakten.se/en/about/`. Accessed: 2015-11-16.

[15] Romain Gaucher. *Grabber*. `http://rgaucher.info/beta/grabber/`. Accessed: 2016-04-04.

[16] The PHP Group. *PHP 5.1.2. Release Announcement*. `https://secure.php.net/releases/5_1_2.php`. Accessed: 2016-03-16.

[17] Isatou Hydara, Abu Bakar Md Sultan, Hazura Zulzalil, and Novia Admodisastro. "Current state of research on cross-site scripting (XSS)–A systematic literature review". In: *Information and Software Technology* 58 (2015), pp. 170–186.

[18] Dr M Jayamsakthi Shanmugam. "Cross Site Scripting-Latest developments and solutions: A survey". In: *Int. J. Open Problems Compt. Math* 1.2 (2008).

[19] Nenad Jovanovic, Christopher Kruegel, and Engin Kirda. "Pixy: A static analysis tool for detecting web application vulnerabilities". In: *Security and Privacy, 2006 IEEE Symposium on*. IEEE. 2006, 6–pp.

[20] Amit Klein. "Divide and conquer". In: *HTTP Response Splitting, Web Cache Poisoning Attacks and Related Topics, Sanctum whitepaper* (2004).

[21] Mitja Kolšek. "Session fixation vulnerability in web-based applications". In: *Acros Security* (2002), p. 7.

[22] MaxiSoler. *2015 Top Security Tools as Voted by ToolsWatch.org Readers*. `http://www.toolswatch.org/2016/02/2015-top-security-tools-as-voted-by-toolswatch-org-readers/`. Feb. 2016.

[23] OWASP. *Broken Authentication and Session Management*. `https://www.owasp.org/index.php/Broken_Authentication_and_Session_Management`. Accessed: 2016-02-09.

[24] OWASP. *Improper Error Handling*. `https://www.owasp.org/index.php/Improper_Error_Handling`. Accessed: 2016-01-28.

[25] OWASP. *OWASP Benchmark Project*. `https://www.owasp.org/index.php/Benchmark1`. Accessed: 2016-12-22.

[26] OWASP. *OWASP Vulnerable Web Applications Directory Project*. `https://www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project`. Accessed: 2016-02-22.

[27] OWASP. *OWASP ZAP 2.4 Getting Started Guide*. `https://github.com/zaproxy/zaproxy/releases/download/2.4.0/ZAPGettingStartedGuide-2.4.pdf`. Accessed: 2016-12-12.

[28] OWASP. *OWASP Zed Attack Proxy Project*. `https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project#tab=Main`. Accessed: 2016-12-12.

[29] OWASP. *Top 10 2007-Insecure Communications*. `https://www.owasp.org/index.php/Top_10_2007-Insecure_Communications`. Accessed: 2016-02-03.

[30] OWASP. *Top 10 2007-Insecure Direct Object Reference*. `https://www.owasp.org/index.php/Top_10_2007-Insecure_Direct_Object_Reference`. Accessed: 2016-02-04.

[31] OWASP. *Top 10 2007-Malicious File Execution*. `https://www.owasp.org/index.php/Top_10_2007-A3`. Accessed: 2016-02-05.

[32] OWASP. *Top 10 2013-Top 10*. https://www.owasp.org/index.php/Top_10_2013-Top_10. Accessed: 2016-02-08.

[33] OWASP. *Welcome to OWASP*. https://www.owasp.org/index.php/Main_Page. Accessed: 2016-02-22.

[34] SecTools. *Top 125 Network Security Tools*. http://sectools.org/tag/web-scanners/. Accessed: 2015-12-16.

[35] Brian Shura. *Web Application Security Scanner List*. http://projects.webappsec.org/w/page/13246988/Web%20Application%20Security%20Scanner%20List. Accessed: 2015-12-16.

[36] Mohd Shadab Siddiqui and Deepanker Verma. "Cross site request forgery: A common web application weakness". In: *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*. IEEE. 2011, pp. 538–543.

[37] Zhendong Su and Gary Wassermann. "The essence of command injection attacks in web applications". In: *ACM SIGPLAN Notices*. Vol. 41. 1. ACM. 2006, pp. 372–382.

[38] Subgraph. *Vega Vulnerability Scanner*. https://subgraph.com/vega/index.en.html. Accessed: 2016-12-21.

[39] Nicolas Surribas. *Wapiti The web-application vulnerability scanner*. http://wapiti.sourceforge.net/. Accessed: 2016-12-24.

[40] Sanoop Thomas and SaMaN. *Xtreme Vulnerable Web Application (XVWA)*. https://github.com/s4n7h0/xvwa. Accessed: 2016-02-23.

[41] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. "Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis." In: *NDSS*. 2007.

[42] Artem Vorobiev and Jun Han. "Security attack ontology for web services". In: *Semantics, Knowledge and Grid, 2006. SKG'06. Second International Conference on*. IEEE. 2006, pp. 42–42.

[43] w3af. *Open Source Web Application Security Scanner*. http://www.w3af.org. Accessed: 2016-04-28.

[44] w3af. *Welcome to w3af's documentation*. http://docs.w3af.org/en/latest/. Accessed: 2016-04-28.

[45] Michal Zalewski, Niels Heinen, and Sebastian Roschke. *skipfish - web application security scanner*. https://github.com/spinkham/skipfish. Accessed: 2016-12-22.

[46] zaproxy. *Scanner Rules*. https://github.com/zaproxy/zaproxy/wiki/ScannerRules. Accessed: 2016-12-12.

[47] Nataša Šuteva Dragi Zlatkovski and Aleksandra Mileva. "EVALUATION AND TESTING OF SEVERAL FREE/OPEN SOURCE WEB VULNERABILITY SCANNERS". In: (2013).