Masaryk University
Faculty of Informatics

Tools for dynamic security analysis of web applications

Bachelor's Thesis

Vojtěch Polášek

Brno, Spring 2016

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Vojtěch Polášek

Advisor: RNDr. Andriy Stetsko Ph.D.

Acknowledgement

I would like to thank Andriy Stetsko for professional supervision, Tomáš Kuba for technical help while facing numerous problems, Radim Göth for invaluable help while collecting initial testing data, and Jiří Pecl for help while polishing final look of the thesis. My thanks also go to my girlfriend Věruška and my family for support during writing of this thesis.

Abstract

This thesis deals with tools for dynamic security analysis of web applications. It introduces 14 tools divided into 4 categories: reconnaissance tools, tools for discovery of specific vulnerabilities, intercepting web proxies, and complex vulnerability scanners. Tools are compared according to their features, licence, price, OWASP Top 10 coverage, and ability to be integrated into Atlassian stack. The thesis researches three selected tools in more details: Sqlmap, W3af, and Arachni. In the end, the thesis contains results produced by the three tools while performing audit of open-source deliberately vulnerable web applications.

Keywords

security, web application, dynamic analysis, SQL injection, software vulnerability, continuous integration

Contents

List of Tables

1 Introduction

This bachelor thesis deals with tools for dynamic security analysis of web applications. It introduces fourteen chosen tools for dynamic analysis of web applications and reviews their features in chapter 2. Furthermore, the thesis researches three tools in greater detail; Sqlmap (chapter 3), W3af (chapter 4), and Arachni (chapter 5). Sqlmap is a specific tool which searches for SQL injection vulnerabilities [12], whereas W3af and Arachni are frameworks which perform complex security audit [42], [16].

Features of three mentioned tools were evaluated with help of deliberately vulnerable open-source web applications. `Sqlmap` was evaluated with two applications running on different web technologies to prove its results (Mutillidae on PHP, WebGoat on Java). `Arachni` and `W3af` were evaluated with one application (WebGoat) and results of those two tools were compared. During the research several bugs were discovered and fixed in `Sqlmap` and `Arachni`. As a part of the research, I created a framework for collection of HTTP traffic and its transformation into HTTP requests. Such requests can be used as input for `Sqlmap`.

To get the most information out of this thesis, a reader should be familiar with basic concepts of contemporary web technologies, databases and SQL syntax. Knowledge of types of vulnerabilities found in contemporary web applications is highly recommended, as they are not thoroughly described in this thesis.

Mentioned category of tools should complement software for static security analysis. It can be inferred from the name that dynamic security analysis does not evaluate source code of an application, but it rather audits its behaviour in certain situations, verifies security measures, and tries to find vulnerable points while an application is running [19]. Sometimes tools try to mimic behaviour of real hackers.

An advantage of dynamic analysis over static analysis lies in providing coverage of situations which cannot be covered by static analysis. For example, dynamic analysis monitors interaction of application code with back end database engines, third party services or libraries etc. These interactions are outside the scope of static analysis. Dynamic analysis monitors reaction of an application to various legal or illegal

input values. This area can be only partially covered by static analysis, because sometimes format of the input cannot be known in advance. However, this approach has its disadvantages. If an application provides large amount of functional paths, an analysis can take long time. It is also necessary to prepare inputs which will be submitted to an application (especially form inputs).

Market with tools for dynamic security analysis grows, but it faces several serious obstacles. Surprisingly, the nature of contemporary web applications which becomes more and more dynamic is the largest one. Tools for dynamic security analysis need to walk through (or be walked through) an application, looking for potential vulnerabilities. The problem is that it becomes harder to follow more complex structures of contemporary web applications [21]. Old methods comprising crawling of links and submitting of forms become insufficient because contemporary applications often use dynamic user interfaces utilising complicated input methods (forms with automatic data validation, pop up menus). Another problem lies in dynamic changing of a website without actually visiting a new URL. This is typical for web applications which try to mimic desktop applications. Today we live in the era of Ajax, HTML5, Flash, Java, and other modern web technologies, and security tools are not always well prepared to handle them properly. Many tests, which are easily automated, when run against applications written in server-side scripting language such as PHP, require manual intervention when being run against rich web applications written for example in Java. As shown by three researched tools, there exist several ways of solving this problem.

There are other problems which need to be solved by developers of software for dynamic security analysis. If they want their tools to be used in large companies and corporations, they have to consider features for integrating their tools into a security testing process. Most tools are designed to be run from a command line, several of them offer some kind of graphical user interface, but this does not automatically mean that they are suitable for being included in a complex testing process. The problem is that their output is often readable for humans, but difficult to process for computers. A possible solution to this problem is to provide a server component which is deliberately designed to communicate with a computer. All three tools analysed

in the thesis feature a server which can be controlled through an API, and output information in machine-readable format.

During the research, I cooperated with Y Soft corporation. I evaluated `Sqlmap`, `W3af`, and `Arachni` on parts of their products.

2 Review of available tools

This chapter provides an overview of a range of tools for dynamic security analysis. It describes their capabilities, limitations and possible uses. I divided tools into four categories: reconnaissance tools, tools aimed at discovery of particular vulnerabilities, intercepting web proxies, and complex web application scanners.

When selecting appropriate tools, I concentrated on their suitability for defined scenario, flexibility, possibility to include them into larger security testing framework and usability in general. Suitability is defined as number of features and their applicability in given situation. Flexibility is defined as ability to customize various configuration options to meet specific situations (HTTP parameters, resource usage limitations…). Possibility to be included in larger testing environment is defined as number of available features which allow integration of researched application into a testing environment (ability to be run without user interaction, programmatic control through API). General usability is defined as overall user experience while using an application. Note that this parameter is heavily affected by my disability (blindness) and my need to use screen reader. Therefore, I was not able to fully test some applications which have inaccessible graphical user interface.

While searching for relevant tools, I used [28] as my primary source regarding OWASP as an authority in a web application security field. I also used a list of tools provided at [7] which provides information about less known but interesting tools.

Every section in this chapter contains a table summarizing general parameters of chosen tools; name, version, release date of last version, supported operating systems, licence, available interfaces for controlling an application, categories of relevant vulnerabilities according to OWASP Top 10, report formats supported by an application. Sections 2.1, 2.3 and 2.4 contain also a table comparing relevant features of selected tools. Section 2.2 does not contain this table, because chosen tools focus on different vulnerabilities and, therefore, it does not make any sense to compare their features.

2.1 Reconnaissance tools

Before an actual attack, an attacker has to gain some information about target web application. This phase is called reconnaissance. The attacker tries to collect various information about web application while causing no damage and drawing minimal attention of administrators. Therefore, most tools used in this phase are passive and they do not try to attack the application actively. Acquired information are further used in planing of actual attack.

Following list summarizes areas of information, which may be useful for attacker's next steps and which may be collected during application reconnaissance [44, pp. 73–114].

- domains, subdomains and DNS records associated with an application may uncover other parts of an application and information about mail servers which may be used e.g. in a phishing campaign
- URLs and their parameters provide map of an application and its features, parameters are potential input vectors
- web pages and their source code may contain valuable information in form of comments left by developers (default credentials, debugging instructions, hidden pages…)
- responses to unexpected requests may display error output of back end technologies (versions, database table names, application tracebacks…)
- SSL/TLS configuration may allow degrading or circumventing SSL connection permitting man-in-the-middle attack if incorrectly configured
- strength of an authentication tokens may allow an attacker to predict tokens and for example access sessions of different users, if it is low
- password policies may accept easily guessable password, if they are too weak

I compare four tools: Maltego, Recon-NG, Whatweb and Stompy. The table 2.1 compares their general parameters. The taable 2.2 compares tools according to information which can be collected and anal-

ysed. The table 2.3 compares tools according to several other features which may help while collecting information.

Table 2.1: Reconnaissance tools - general parameters

Name	Maltego	Recon-NG	Whatweb	Stompy
Version	3.6.0	4.7.3	0.4.8	0.04
Last updated	April 2015	November 2015	August 2015	2007
Supported Oss	Windows, Linux, OS X	Windows, Linux, OS X	Windows, Linux, OS X	Windows, Linux, OS X
Licence	custom	GNU GPLv3	GNU GPLv2	unknown
Price	free community version, $760 for commercial edition, $25000 for private server	free	free	free
Interface	GUI	CLI, RPC, console	CLI	CLI
OWASP Top 10	A5	A5, A9	A5, A9	A2
Report format	PDF	TXT, HTML, CSV, JSON, XLSX, XML	TXT, XML, JSON, Magictree, MongoDB	TXT

Table 2.2: Reconnaisance tools - gathered information

Information	Maltego	Recon-NG	Whatweb	Stompy
Domain	yes	yes	yes	no
IP address	yes	yes	yes	no
DNS servers	yes	no	no	no
DNS records	yes	no	no	no
Contact information	yes	yes	no	no
Interesting files	yes	yes	no	no
Social networking profiles	yes	yes	no	no
Autonomous systems	yes	no	no	no
Authentication token strenght	no	no	no	yes
Geographical locations	yes	yes	no	no
Net blocks	yes	yes	no	no
URLs	yes	no	no	no
Server technologies	yes	no	yes	no
Login credentials	no	yes	no	no
Vulnerabilities	no	yes	yes	no
Source code repositories	no	yes	no	no
Underlying web server or embedded device	no	no	yes	no

Table 2.3: Reconnaisance tools - other features

Feature	Maltego	Recon-NG	Whatweb	Stompy
requires manual supply of API keys	no	yes	n/a	n/a
Exploits found vulnerabilities	no	yes	no	n/a

2.1.1 Maltego

Maltego is a tool for modelling of relationships among objects. It is produced and supported by Paterva company. These objects may represent various entities; people, organizations, web sites, routers, DNS servers etc. Maltego is able to search for this information and visualize connections among them [36]. It uses open-source intelligence for finding connections among objects. Note that the tool is not an open-source software. I chose this tool because it offers large amount of information sources and it is a well-known tool in information gathering category.

Maltego uses a concept of so called transforms. Each transform takes some information as an input and transforms it into an output using public resources and services. For example, it can transform someone's name into URLs of their social networking accounts, or a domain into list of subdomains through dictionary attack.

Some public search engines require API keys to use their features or to perform large batches of queries (Google, Bing, Yahoo…). This should prevent excessive number of queries performed by unsolicited users. A user has to provide personal information before receiving an API key. However, Maltego users do not need to obtain API keys because Paterva maintains active keys for public servers.

Maltego contains 72 documented transforms created by Paterva and more free or paid transforms created by Maltego community. Built-in transforms perform tasks connected with reconnaissance, but there exist other custom transform sets, which can be used for detection and exploitation of vulnerabilities [35]. All the information may be viewed in graphs and trees, allowing observation of various types of connections and relationships.

This tool consists of two parts; client and server. The client is a desktop application written in Java. It serves as a graphical interface to whole process of information collection and interpretation of results. A client communicates with a Maltego server which performs actual tasks connected with collection of information (web searches, DNS queries...). It is possible to use public servers provided by Paterva or to purchase them in form of VMWare images and run them privately on one's own infrastructure. The client comes in two versions. The community version is for free, it must not be used for commercial use and has some limitations (slower servers, limited number of results, communication with servers is not encrypted, no end user support...). The paid version has no limitations of the community version, public servers are more powerful and it can be used with purchased private servers [37].

Unfortunately, I could not try this tool, because its graphical interface is not accessible for people using screen readers.

2.1.2 Recon-NG

Recon-NG is a web reconnaissance framework written in Python 2. It provides a console interface similar to Metasploit and it has even similar commands. Every reconnaissance task is accomplished by a certain module. Custom modules written in Python 2 may be easily added. It does not offer any graphical interface, but it has a command line interface and an RPC interface which can be used for remote control through commands in JSON or XML format. Results are stored in an SQLite database. The framework contains 79 reconnaissance modules and 13 modules for other purposes (report generation) [45].

Recon-NG's set of modules is comparable to Maltego's basic documented set of transforms, although there are some differences. Both tools use several search engines which require API keys to be accessed. The company behind Maltego takes care of assigning all needed API keys to Maltego servers, whereas when using Recon-NG, it is up to you to get and eventually renew needed keys. Recon-NG does not provide any graphical interpretation of results. Recon-NG plugins are aimed mainly at reconnaissance, but Maltego offers beside reconnaissance other sets of transforms, which are aimed at exploitation of vulnerabilities.

2.1.3 Whatweb

Whatweb is a website scanner written in Ruby. This tool visits a given URL and tries to extract all possible information about web technologies, content management systems, web server software, embedded devices, analytics packages and JavaScript libraries. Currently, it offers 1742 plugins. Every plugin looks for certain information in HTTP responses received from the web server and extracts it. This tool has a simple command line interface and it can generate several log formats which may be suited for further processing by other tools for penetration testing such as Magictree [22].

I have tried this tool and it works as expected. You can view a sample output in section A.1.

2.1.4 Stompy

Stompy is a tool for evaluation of randomness of session tokens. In case of web applications, those tokens are usually session cookies, but Stompy can evaluate randomness of any tokens which are meant to be unpredictable and secure against statistical analysis or brute force attacks [44, pp. 210–213]. Stompy is fully automatic. It can obtain cookies by sending HTTP requests to a web server, or it can be provided with a file with already collected tokens. It automatically detects alphabet used in tokens. It decomposes tokens into bit streams and observes how bit structure changes when a new token is generated. Stompy performs NIST FIPS-140-2 PRNG evaluation tests [23] and n-dimensional phase analysis on bit streams. In the end, it performs spatial correlation checks to identify dependencies between neighbouring bits. A final report contains number of correct and anomalous bits, as well as human-readable rating of untainted entropy (number of bits which are not predicable) [47].

2.2 Tools discovering particular vulnerabilities

This section introduces three tools which discover and exploit particular vulnerabilities present in web applications. It may seem unnecessary to use such tools when there exist complex web application security scanners, but these specialized tools offer more flexibility and

advanced features suited for detection and exploiting of particular vulnerability. There may appear a case when complex web application scanner discovers potential vulnerability but it can not verify it. In this case, it is a good idea to use specialized tool, which may confirm or refute the presence of the vulnerability. As most of these tools are able to exploit the vulnerability, they can demonstrate real impact of such vulnerability which could be caused by potential attacker.

Such tools are sometimes created as quick defence against newly discovered vulnerabilities, because it takes some time to implement detection of such flaws into larger vulnerability scanners. This may be case of Heardbleed SSL vulnerability, Shellshock vulnerability etc.

I chose three tools: OWASP Xenotix, Sqlmap and Tessercap. The table 2.4 compares tools according to general parameters mentioned at the beginning of this chapter.

Table 2.4: Tools discovering specific vulnerabilities - general parameters

Name	Xenotix	Sqlmap	TesserCap
Version	6.1	1.0	1.0
Last updated	December 2014	January 2016	Unknown
Supported OSs	Windows	Windows, Linux, OS X	Windows
Licence	CC BY-SA 3.0	GNU GPLv2	unknown
Price	free	free	free
Interface	GUI	CLI/JSON	GUI
OWASP Top 10	A3	A1	A2
Report format	unknown	CSV, TXT	none

2.2.1 OWASP Xenotix XSS exploit framework

This framework focuses on detection and exploitation of cross-site scripting (XSS) vulnerabilities. I selected this tool because it is a really good example of specialized tool. XSS vulnerability is a type of injection and occupies the third place in OWASP Top 10 2013 edition. This

vulnerability occurs when a web application sends unvalidated data to a browser, which were previously acquired through an untrusted input. If these data are not properly sanitized and contain text-based scripts, they may execute in browser environment and cause a significant damage ranging from insertion of hostile content to hijacking of user's whole browser session. OWASP recognizes three types of XSS; stored, reflected and DOM-based. Every such XSS flaw can occur on a server side or on a client side [29].

In the era of modern browsers and rich internet applications, it becomes harder to detect XSS flaws. Especially client side XSS flaws are hard to detect, because they cannot be spotted in any source code audit of a server application. They occur in user's browser and they have to be searched for right there. Xenotix differs from other tools by using three separate browser cores (Trident used by Internet Explorer, WebKit used by Google Chrome, and Gecko used by Firefox). Thanks to this approach to detection and exploitation of XSS, it gains large advantage over competing tools and frameworks, because it renders all content in real world browsers.

The framework contains scanner modules, including HTTP POST and GET fuzzers, DOM XSS analyser and hidden parameter detector. It contains set of fingerprinting modules, allowing performing fingerprinting of web application firewalls as well as of potential victims. The framework contains impressive database of real world exploits, including keyloggers, web shells, DDoS exploits, exploits used during social engineering attacks etc. There are also numerous helpful tools for further analysis, including several interesting JavaScript encoders, hash calculator, hash detector, JavaScript beautifier and set of developer tools for WebKit browser core. Xenotix also offers its own API for easy scripting [31].

Unfortunately, I was not able to test this tool, because its interface is inaccessible to people using screen readers. This tool also lacks proper documentation. Therefore, I was not able to get more detailed information about its features.

2.2.2 Sqlmap

Sqlmap is a tool aimed at discovering and exploiting of SQL injection flaws. This tool is described in the chapter 3.

2.2.3 Tessercap

This tool utilizes image preprocessing and optical character recognition (OCR) technology to overcome image captchas, therefore assessing their strength. Captcha stands for Completely Automated Public Turing test to tell Computers and Humans Apart [4]. This test can have various forms; images containing text which needs to be rewritten by humans, sound recordings containing spoken characters which need to be rewritten, simple mathematical tasks, interactive chess tasks etc. The first mentioned type of captcha is today the most used type, although there exist disputes over its efficiency and accessibility [20].

This tool has a simple graphical interface. The tool collects defined amount of captchas from provided URL. A user is able to create image preprocessing template to make text within images more readable. This template consists of several image modifications: inverting of RGB value of chosen pixels, application of smoothing or sharpening filter to the image, removal of granular noise, modification of border colour scheme etc. After modification of image, it is passed to Tesseract OCR engine which performs actual text recognition and tries to extract textual content from the image. When enough captchas are tested, the user can mark correct or incorrect results and receive statistical results. It is possible to limit characters of alphabet used during text recognition to improve quality of results. The tool supports communication through web proxy and modification of HTTP headers during obtaining captcha images, so that user can supply valid cookies and perform other necessary modifications [15].

I have not tested this tool, as I am not able to distinguish its right guesses from wrong ones. I can only tell that the interface of the tool is accessible for people using screen readers.

2.3 Intercepting web proxies

Intercepting proxy is an important tool, which enables detailed analysis of web traffic flowing between our computer and target web application. A user can inspect traffic with network protocol analysers such as Wireshark, but specialized proxies are designed to process web traffic, and therefore they are more comfortable to use. Moreover, they often offer specialized features not shipped with generic network protocol analysers, such as detailed inspection and modification of HTTP/S traffic, replay of modified traffic, passive or active vulnerability scanners, detailed graphs and reports. It is important to note here that intercepting proxies or tools built around such proxies are different from automatic web vulnerability scanners mentioned in the section 2.4. Intercepting proxies are meant to be used in manual security testing. Although they offer some automation to ease auditor's work, they cannot perform audits automatically without human intervention.

I selected three intercepting proxies: Burpsuite, Mitmproxy, and OWASP Zed Attack Proxy. General parameters are compared in the table 2.5, relevant features of proxies are compared in the table 2.6.

Table 2.5: Intercepting web proxies - general parameters

Name	Burp Suite	Mitmproxy	ZAProxy
version	1.6.32	0.16	2.4.3
Last updated	January 2016	February 2016	December 2015
Supported OSs	Windows, Linux, OS X	Linux, OS X, partially Windows	Windows, Linux, OS X
Licence	custom	MIT	Apache
Price	free edition or $349 per user per year for professional edition	free	free
Interface	GUI	CLI, console	GUI
OWASP Top 10	all except A9	not applicable	all except A9
Report format	HTML, XML	not applicable	HTML

Table 2.6: Intercepting proxies - features

Feature	BurpSuite	Mitmproxy	ZAProxy
Dynamic SSL certificates	yes	yes	yes
Real time rendering of web content	yes	no	no
Web spider/crawler	yes	no	yes
support for complex authentication schemas	yes	no	yes
Integrated vulnerability scanner	yes	no	yes
On-the-fly modification of traffic	yes	yes	yes
Manual vulnerability scanner	yes	no	no
Random token analyzer	yes	no	no
Support for X.509 client certificates and smartcards	no	no	yes
Forced browsing	no	no	yes
Transparent TCP proxy	no	yes	no

2.3.1 Burp Suite

I selected this tool because it is a well-known tool in this category. The key piece of whole framework is an intercepting proxy, which captures HTTP and HTTPS traffic, displays it to a user with automatic syntax highlighting and web content rendering. It serves as a hub from which requests can be sent to other parts of the tool. Requests can be modified while flowing through the proxy based on defined rules. Next, the framework contains a web spider, which collects all URLs and functional paths of the application, presenting them in form of trees or tables. It can work in passive mode, building a picture of a web application's structure only through analysing requests flowing through the proxy. It can also work in active mode, automatically following links and submitting forms. It can deal with complex applications employing AJAX and authentication schemes.

The framework features a vulnerability scanner. It can perform passive scanning of request passing through the Burp proxy, or it can issue additional requests, actively probing for flaws in the application. According to the information on Burp Suite website, the scanner uses unique feedback-driven logic, therefore reproducing behaviour of methodical penetration tester and delivering high rate of vulnerability detection and at the same time low rate of false positives.

Another feature is called Intruder. It is a tool created for performing of customized attacks. It is useful when testing some non-standard authentication schemes, fuzzing, performing brute force attacks etc. It is actually a complement to the scanner, which works in fully automatic way, whereas intruder requires a user to mark parts of an HTTP requests where a payload will be inserted. It performs a number of requests while changing the payload to discover potential vulnerability. Intruder subsequently displays received responses and it is up to a user to decide, if there exists a vulnerability or not.

Burp suite contains two more tools: a repeater and a sequencer. The repeater may seem to be a little similar to the intruder, but there is a significant difference. The intruder submits large number of requests and automatically makes small changes in inserted payload, searching for a vulnerability. The repeater sends only one request and displays its response, allowing making sophisticated manual tests. The sequencer evaluates randomness of various tokens, which should not

be predictable or easily guessable (antiCSRF tokens, session cookies). It contains a large battery of statistical tests including those meeting FIBS specification. It can automatically collect required amount of samples and continuously perform tests at character or bit level of tokens [39].

The framework is written in Java and can be extended with plugins written in Java, Python or Ruby. It has a complex graphical interface. I found that the interface is not prepared to be used with screen readers. Therefore, I was not able to fully test this tool. The tool has two editions; free and professional. Free edition does not contain some features (scanner, target analyser, support for save and resume) and has other limitations (limited usage of intruder, slower update cycle). The professional edition does not have any limitations.

2.3.2 Mitmproxy

Mitmproxy is a simple but powerful intercepting web proxy. Its main purpose is to intercept and eventually manipulate HTTP/S traffic. Mitmproxy can display and modify requests and responses based on regular expression filters. It supports on-the-fly generation of SSL certificates, allowing intercepting HTTP as well as HTTPS traffic. It is possible to replay saved client requests and server responses. Traffic flows can be exported into files for later analysis or replay. If a client does not support proxy configuration, Mitmproxy can work in so called transparent mode, receiving and redirecting traffic on the network layer. It does not contain any scanners, fuzzers or other advanced security tools.

Mitmproxy can be controlled in several ways. It has an interactive terminal interface where everything can be controlled and displayed interactively. The interface uses Ncurses library. It can be also run in unattended mode. This is useful if we want to use Mitmproxy in some automated process. It is possible to extend it with Python scripts. As it is written in Python2, its features can be easily included into custom projects [9]. I used its library Libmproxy in my simple Python framework for capturing of HTTP traffic needed for Sqlmap. You can read about it in section 3.6.

2.3.3 OWASP Zed Attack Proxy

I selected this tools, because it represents an open-source alternative to BurpSuite. This tool includes an intercepting web proxy, which displays detailed information about traffic flowing in both directions and allows on-the-fly modification of data. It is a central part of the whole tool. The proxy can deal with encrypted traffic by automatically creating SSL certificates, effectively employing man-in-the-middle attack on the client. It can also use X.509 certificates stored in files or on a smartcard. Traffic flowing among websockets can be intercepted and modified as well. Another feature is a spider, which recursively crawls through the web application and collects URLs of found resources. Traditional spider searches for URLs in HTML tags, comments and any textual responses. AJAX spider can crawl modern applications which utilize AJAX.

The tool contains two scanners; active and passive. The active scanner uses set of scanning rules to search actively for vulnerabilities like cross-site scripting, SQL injection, remote file inclusion, command injection, buffer overflow etc. The passive scanner inspects and parses flows of traffic going through the proxy and searches for vulnerabilities without actively creating new requests. It searches for disclosure of private IP addresses, improper display of mixed content, proper clickjacking protection and more.

ZAProxy can perform forced browsing. It is a process during which it tries to locate files and directories, which are not directly linked from other sites, but they may exist based on predictable file names. This technique is also known as predictable resource location.

ZAProxy supports running of scripts written in ECMAScript, Zest, JavaScript, Groovy, Python, Ruby and more. They may access internals of ZAProxy and help in non-standard situations. The tool can be extended with add-ons, which can be downloaded from ZAProxy marketplace. Add-ons contain for example new rules for scanners or fuzzers, support for Selenium IDE, list of directories used within forced browsing etc. [5], [32]

ZAProxy is written in Java and it has a rich graphical interface. It does not have any command line interface, but it offers REST-based API, currently available through JSON, HTML or XML messages.

I have not worked extensively with this tool, because it has very complex interface and I have encountered some accessibility problems while working with it.

2.4 Complex web vulnerability scanners

Vulnerability scanners usually scan whole applications or whole parts of an application and they try to detect a wide scale of flaws. I selected four scanners: Acunetix web vulnerability scanner, W3af, Netsparker, and Arachni. The table 2.7 compares general parameters of tools. The table 2.8 compares scanners according to offered features.

I encourage readers to look at [8] for interesting comparison of vulnerability scanners. It includes results of additional free and commercial vulnerability scanners.

Table 2.7: Complex web vulnerability scanners - general parameters

Name	Acunetix WVS	Arachni	Netsparker	W3af
Version	10	1.4	Desktop 4.5.7, Cloud 20160129	1.7.6
Last updated	November 2015	February 2016	January 2016	February 2016
Supported OSs	Windows	Windows, Linux, OS X	Windows	Windows, Linux, OS X
Licence	custom	Apache licence	custom	GNU GPLv2
Price	from $2495 to $5495 per year for the On-premise solution, from $345 to $10275 per year online service	free	from $1950 to $5950 per year desktop version, cloud version is priced per scanned site or per number of scans	free
Interface	desktop, web	CLI, web, RPC	desktop, web	CLI, GUI, console, RPC
OWASP Top 10	all	all except A9	all	all except A9
Report format	PDF, TXT, DOC, BMP, HTML	HTML, JSON, XML, YAML, Marshal, TXT	HTML, PDF, CSV, XML	CSV, HTML, TXT, XML

Table 2.8: Complex web vulnerability scanners - features

Feature	Acunetix WVS	Arachni	Netsparker	W3af
Crawling of dynamic applications	yes	yes	yes	no
Source code analysis	yes	no	no	no
Login sequence recorder	yes	yes	yes	no
Compliance reports	yes	yes	yes	no
Network scanner	yes	no	no	no
Manual penetration testing tools	yes	no	yes	no
WSDL support	yes	no	yes	no
Exploiting of found vulnerabilities	no	no	yes	yes
Ability to abort and resume scans	no	yes	yes	no

2.4.1 Acunetix Web Vulnerability Scanner

This is a commercial web vulnerability scanner. According to its brochure [1], it is fully prepared to scan modern dynamic applications employing AJAX as well as applications written in HTML5. It can detect over 500 web vulnerabilities [2]. They include vulnerabilities in actual application as well as vulnerabilities in back end technologies and third-party libraries. It uses several unique technologies. Its AcuSensor technology combines black box scanning with source code analysis. The scanner looks into source code of target application while performing regular vulnerability scan. It means that when a vulnerability is found, the scanner can report its exact location in source code and other additional information which can help with problem remediation. AcuSensor works with applications written in PHP and .NET

and it can intercept SQL queries between the web application and a back end database for better detection of SQL injection flaws.

Acunetix DeepScan technology specializes in crawling of HTML5 applications which make heavy use of JavaScript libraries (AngularJS, Backbone.js etc.). It can scan single page applications(SPAs). These applications use only one web page and dynamically load and change content to resemble desktop applications. Scanning of web services (WSDL) is supported as well.

Scanning is important, but reporting of found vulnerabilities is too. The scanner can generate extensive reports for security professionals as well as for busyness managers. It can report state of compliance to following specifications.

- CVE/SANS top 25 most dangerous software errors
- The Health Insurance Portability and Accountability Act(HIPAA)
- ISO 27001
- NIST Special Publication 800-53
- OWASP Top10 2013
- Payment Card Industry Data Security Standard (PCI DSS)
- Sarbanes Oxley Act
- Defence Information System Agency Security Technical Implementation Guide (DISA STIG)
- Web Application Security Consortium (WASC) thread classification

This tool also incorporates open-source network vulnerability scanner OpenVAS and there provides complex scan of web application and related network components such as HTTP servers, FTP servers, name servers, mail servers etc. Acunetix web vulnerability scanner contains specialized part for scanning for vulnerabilities in Wordpress content management systems. In the end, it offers several tools for manual penetration testing including HTTP fuzzer, request editor and sniffer [3].

The scanner can be purchased as a cloud application which runs on Acunetix servers. The second option is to buy On-Premise licence.

A 14 days trial version of both variants is available for free. The trial version limits target to Acunetix's own deliberately insecure website.

While evaluating desktop version of the scanner I found several problems related to accessibility of user interface for people using screen readers. I was not able to configure scan properly.

2.4.2 Arachni

Arachni is a free open-source web vulnerability scanner written in Ruby. It is described in the section 5.

2.4.3 Netsparker

Netsparker is a commercial web vulnerability scanner. It declares support of dynamic web applications written in HTML5 and using rich JavaScript technologies. Support for WSDL and SOAP web services is implemented. However, it offers interesting concepts and features not present in other solutions. If a vulnerability is found during scan, Netsparker reports it and also tries to exploit it. This method confirms the vulnerability and Netsparker even tries to extract useful information through exploited flaw (for example SQL injection) for further scanning. The scanner also offers manual penetration testing tools for custom scanning and confirming of vulnerabilities. These tools include encoder and decoder (Base64, URL, HTML and more schemes), a view state monitor for ASP.NET and several features for manual scanning of applications.

Netsparker can operate together with other security tools including Metasploit, Burp Suite, Threadfix Vulnerability Manager, Dradis Framework and several others. Another feature which may increase productivity is a possibility to automatically create tickets in bug tracking systems such as Jira or Github. Automated security scans can be ran as a part of a build process. When scanning a large application, the scan can take several hours or days. In these cases, it is possible to abort the scan, save the session and resume the scan later. Netsparker offers broad reporting capabilities. It generates standard reports for security professionals, reports for developers which include list of scanned URLS and other technical information and the scanner reveals its reporting API for further customization of reports [25]. The

tool can generate compliance reports according to HIPAA, PCI dss and OWASP Top 10 standards.

Netsparker can be purchased as a cloud service or a desktop application. It offers a command line interface for automation.

I tried to test capabilities of Netsparker, but graphical user interface had serious accessibility issues and I was not able to configure it properly and investigate results. I wanted to use command line interface but proper functionality requires a scanning profile. This profile can be created only through graphical interface.

2.4.4 W3af

W3af is an open-source vulnerability scanning framework. Please see the section 4 which is dedicated to this tool.

3 Sqlmap

This chapter describes SQL injection vulnerability and its various types. Then it describes `Sqlmap`: a tool for detection and exploitation of SQL injection. Eventually, it summarizes methodology and results of audit performed with `Sqlmap`. I used `Sqlmap` to audit open-source deliberately vulnerable web applications.

3.1 SQL injection vulnerability

This kind of vulnerability allows an attacker to inject malicious data into an SQL statement performed by the web application. It is caused by improper sanitization of user input, which is subsequently used in a SQL query. Successful exploitation of SQL injection may have various unpleasant consequences. Although SQL injection vulnerability has been well known for more than 19 years, it still belongs among the most exploited web application vulnerabilities [33].

3.1.1 How does an SQL injection look?

The following URL contains a parameter *id*, which is further processed.

```
http://example.com/view.php?id=1
```

This parameter is consequently inserted into the following SQL query.

```
SELECT number,title,description FROM products WHERE id = 1
    AND hidden = 0;
```

Here we can see that argument of parameter id is directly inserted into the SQL query without any further processing or input validation. This creates an SQL injection vulnerability. If a potential attacker wants to view hidden products as well, they can visit the following URL.

```
http://example.com/view.php?id=1 OR 3 = 3--
```

This will display data returned by the following statement, effectively bypassing the simple restriction because of the inserted comment character.

```
SELECT number,title,description FROM products WHERE id = 1
OR 3 = 3--AND hidden = 0;
```

This example was really simple. Unvalidated input may be inserted into more complicated statements, returning more complex data structures.

3.1.2 SQL injection in numbers

SQL injection vulnerability is still very popular among attackers and it represents a big risk for today's businesses, as they use web technologies more and more often. SQL injection together with other injection flaws took the first place in OWASP Top 10 2013 report [30]. According to statistics [24] the best year for SQL injection vulnerabilities was 2008 with 1096 vulnerabilities. Then it continuously dropped down to 145 identified vulnerabilities, but it rose again to 296 in 2014 and in November 2015, there has been identified 203 vulnerabilities so far.

3.1.3 SQL injection impacts

Impact of successful exploitation of an SQL injection vulnerability can range from unauthorized access to data stored in a database, to gaining control over a remote server. It strongly depends on a point in an SQL query, where a malicious input is inserted, and specific type of the Query. Furthermore, it depends on back end database engine and its version, access rights of a database user executing the query, back end operating system and its version etc.

SQL injection vulnerability can break all three key concepts of information security: confidentiality, integrity, and availability. If an attacker is able to exploit the vulnerability, they are usually able to read information from the database and this breaks confidentiality. Sometimes they may be even able to modify or delete data, therefore breaking integrity and availability. And if we take into account the worst case, taking control over a whole server, the security is completely defeated.

3.2 Sqlmap in detail

`Sqlmap` is an automatic SQL injection and database takeover tool [12]. It is controlled mainly from the command line, but it also ships with REST-JSON server and client. There exist a web based graphical user interface which uses the API [14]. It can be run in batch mode, which does not require any interaction. It can be also easily extended or included in larger frameworks such as `W3af` [40].

`Sqlmap` is able to detect SQL injection vulnerabilities in 11 different database back ends through 5 different detection techniques. Although it is an automatic tool, it allows modification of almost every aspect of detection and exploitation phase. It can work within authenticated parts of a web application or send traffic through a proxy. It ships with a set of so called tamper scripts, which can bypass weakly configured web application firewalls.

Another interesting feature is its ability to import targets from the third party web application scanners. Currently it recognizes logs created by BurpProxy and Webscarab. If you want to know what traffic flows between `Sqlmap` and the target web application, `Sqlmap` can store all traffic for later analysis. And least but not last feature is its ability to simulate a real attack by exploitation of vulnerable database with help of user defined functions or Metasploit payloads.

3.2.1 Supported detection techniques

`Sqlmap` supports five different techniques for detecting an SQL injection flaw. These techniques try to use various vectors to detect if malicious queries can be inserted into a database and if it is possible to retrieve returned data.

Boolean based blind detection technique

This technique is called *blind* because `Sqlmap` really does not know and cannot know in which part of an SQL query it tries to inject its payload. It is also called *boolean* because `Sqlmap` differentiates only between two results: true and false [26].

`Sqlmap` retrieves result of the input without any modification and pronounces this a true page. Then it tries to modify the input in a way,

which should intentionally force the query to return different result without causing an SQL error. Sqlmap pronounces this page as false. If Sqlmap is convinced that this technique can be used, it starts retrieving data from the back end database as shown in following example.

Let's assume that the following URL is a target.

```
http://example.com/getuser.php?id=2
```

The parameter of the URL is inserted in an SQL statement.

```
SELECT name,surname,signature from USERS where id = '2';
```

Let's assume that this query returns following data.

```
name='John'; surname='Doe'; signature='Lorem ipsum';
```

For now, Sqlmap supposes that this return value is valid, no error has occurred. Therefore, it calls this page a *true* page. Sqlmap now tries to induce a *false* response.

```
http://example.com/getuser.php?id=2'
```

This request leads to execution of the following statement.

```
SELECT name,surname,signature from USERS where id = '2'';
```

And this in turn results in an error, because syntax of our query is invalid. Let's assume that this application does not display any error, but returns this.

```
name = ''; surname = ''; signature='';
```

Sqlmap sees a difference in returned data and pronounces this result as *false* page. It suspects that there might be a vulnerability, so it will try to confirm it.

```
http://example.com/getuser.php?id=2' AND 1 = 1--
```

The sequence of two dashes at the end of the query denotes an SQL comment, so that the apostrophe hard coded in the statement is ignored.

```
SELECT name,surname,signature from USERS where id = '2'
AND 1 = 1--';
```

This request returns information which Sqlmap has already seen.

```
name='John'; surname='Doe'; signature='Lorem ipsum';
```

Now it injects a query which should be always false.

```
http://example.com/getuser.php?id=2' AND 1 = 3--
```

And this is the *false* page because

```
SELECT name,surname,signature from USERS where id = '2'
AND 1 = 3--';
```

never evaluates true.

Sqlmap blindly inserts malicious inputs and makes assumptions based on returned outputs. When Sqlmap is sure about true page and false page, it can try to exploit discovered vulnerability by reading some data from the database. Let's assume that it can use function *substring(string, start, end)* which returns a substring of the *string* parameter, spanning characters from *start* to *end* position. It is also going to use an *ascii(char)* function, which returns an ASCII value of the character *char*. The following URL returns the *true* page if the first character of John's password is A.

```
http://example.com/getuser.php?
id=2' AND ASCII(SUBSTR(password, 1, 1)) = 65--
```

If Sqlmap does not receive the expected result, it continues testing for character B, C, etc. Sqlmap implements the bisection algorithm which distinguishes between false and true pages and can fetch one character in a maximum of seven requests. A user can help Sqlmap by providing strings present on true or false pages.

Time based blind technique

This is again a blind based SQL injection technique, so Sqlmap cannot directly read data returned by a malicious query, but it can inference them. However, it uses artificially induced time delays to detect if the malicious statement evaluated true or false [26]. Continuing with the hypothetical vulnerable web application, imagine what happens if this URL is accessed.

```
http://example.com/getuser.php?id=2' UNION SELECT
IF(ASCII(SUBSTRING(password,1,1)) = 50,
BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')),null)
FROM users WHERE user_id = 2;
```

This is a little more complicated. Function *benchmark(count,function)* gets introduced. It performs supplied *function count* times. If the first character of John's password is 2, server response should be somehow delayed. Otherwise, the response should not be delayed. This technique requires some knowledge about back end database and its version. This technique uses the same algorithm as the previous one.

Error based SQL injection technique

Notice that this is not a blind technique any more. This technique requires a web application to display database error message to the user [12]. `Sqlmap` submits special statements provoking error messages for specific database back ends and searches within returned error message for injected characters and result of maliciously injected subquery. The technique is specific for every database back end. Here is a modified example based on [34]. Suppose that we have the following statement.

```
select username from users where name = username;
```

This query is performed by a script. Parameter is passed by the following URL.

```
http://www.example.com/getuser.php
?id=username
```

Let's suppose that an application uses Oracle 10g as its back end database engine. Visiting following URL may generate an error which contains information stored in the database.

```
http://www.example.com/getuser.php?id=1
||UTL_INADDR.GET_HOST_NAME((SELECT user FROM DUAL))--
```

The appended query uses GET_HOST_NAME() function to resolve a host name, but it passes an information stored in a database which

is certainly not a valid host name. The function fails and returns following error together with information which should not be visible to potential attacker.

```
ORA-292257: host JOHN unknown
```

Union based SQL injection technique

When Sqlmap uses this technique, it tries to append a valid UNION ALL statement to the original query. The original query has to be inserted into a SELECT statement. If the original statement is executed in a for loop, therefore returning multiple results, Sqlmap tries to retrieve some additional information in this way [12]. Sqlmap must include the same number of columns in the UNION statement as in original statement, otherwise it would cause an error. Sqlmap is also able to detect partial union SQL injection vulnerabilities, where the original statement is not executed in a for loop. Sqlmap allows user to supply custom values for number and position of columns, if they are known.

Let's return to the vulnerable web application at example.com and look at the following URL.

```
http://example.com/getuser.php?
id=2' UNION SELECT name,surname,password FROM users--
```

This results in a query returning also user's password.

```
SELECT name,surname,signature FROM users
WHERE id = '2' UNION SELECT name,surname,password
FROM users--';
name='John'; surname='Doe'; signature='Signature';
name='John'; surname='Doe'; password='secret';
```

Stacked queries SQL injection vulnerability

Some web application frameworks and back end databases support stacked queries. It means that an attacker can end an existing query with a semicolon and append another query. This is dangerous, because a successful attacker is able to easily manipulate data or access

underlying file system very easily [12]. Sqlmap requires this vulnerability for access to Windows registry and it simplifies some other exploitation cases.

A simplified example follows. If an attacker wants to run some malicious query, they can try following.

```
http://example.com/getuser.php?id=2'; MALICIOUS_QUERY--
SELECT name,surname,signature FROM users WHERE id = '2';
MALICIOUS_QUERY--';
```

3.2.2 Sqlmap usage

It is not my goal to explain every command line switch and argument of Sqlmap, this has been already done by the author on Sqlmap's wiki page. But I would like to summarize and demonstrate Sqlmap's useful features which may be attractive for companies wishing to integrate Sqlmap into their testing environment. I mention appropriate command line switches and provide some examples, but I encourage reader to look at the wiki [12] for more information.

Specifying target

Sqlmap needs a target to audit. Sqlmap can be provided with an URL, a list of URLs, a log file created by WebScarab or Burp proxy, a sitemap.xml file, a Google dork or a raw HTTP request file. Passing an URL with the *u* switch is probably the most straightforward way. Sqlmap detects possible injection points and immediately starts auditing. If the parameter is not recognized by Sqlmap, it can be specified manually by surrounding it with asterisks.

```
sqlmap -u http://example.com/get/user/*42*
```

Sqlmap can automatically crawl provided website and search for more potentially vulnerable URLs. It can also automatically detect presence of forms and submit them, obtaining another possible injection points.

However, while trying to audit an application which communicates through HTTP POST requests instead of GET request, it might be difficult or even impossible to supply Sqlmap with correct data. This communication method is often encountered in modern web

applications. A URL of a website stays the same while HTML5 or JavaScript code communicates with a server in background. There exist two solutions to this problem: to use the *–data* switch and supply POST data manually, or to use the *-r* switch and provide file with a HTTP request. Such file contains whole HTTP request as it is sent to a server. It includes all HTTP headers and request body. It may look like the example in listing 3.1.

Listing 3.1: Sample HTTP request file

```
POST /users/UserManager HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive
Content-Length: 48
Accept: application/json, text/javascript, */*; q=0.01
Origin: http://127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome
    /39.0.2171.95 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=
    UTF-8
Referer: http://127.0.0.1/index.jsp
Accept-Encoding: gzip, deflate
Accept-Language: cs-CZ,cs;q=0.8,en;q=0.6
Cookie: JSESSIONID=85FB869D9F18B70D668318FA53FC5807; lang
    =en

get=userinfo&id=42
```

I suggest the second option, because it offers the most faithful representation of traffic including HTTP headers which may also be a target of an SQL injection attack.

`Sqlmap` provides further options, which allow to fully customize data being sent to the application. A user can add, modify or remove HTTP headers, load cookies from a file, randomly change User-Agent header, randomly change value for specified parameters, or even evaluate Python code which creates values for parameters. `Sqlmap` allows sending of all traffic through proxy, which may be helpful while per-

forming authenticated scans or using a proxy to capture HTTP traffic for later analysis. Authenticated proxies are supported.

Performing authenticated scan

Lots of today's application offer full functionality only to authenticated users. Content management systems, forums, and eshops are several example cases. Auditing authenticated sites poses no problem for `Sqlmap`. It can send traffic through authenticated proxies or utilize HTTP authentication including Basic, Digest, and NTLM. `Sqlmap` can be provided with an anti-CSRF token or URL to obtain such token.

A user can provide `Sqlmap` with so called safe URL. Some applications may terminate user sessions after some time passes, or after certain amount of invalid requests is detected. Visiting safe URL should not destroy current user session but rather keep it active. A good example of safe URL is user's profile page. `Sqlmap` visits provided safe URL once per configurable amount of requests and ensures that our session with the application stays active.

Improving detection

`Sqlmap` offers interesting and simple to use optimization features, which may speed up the process of finding a vulnerability. It can use persistent HTTP/S connections. It allows using one TCP connection for sending and receiving multiple HTTP requests and responses. A new connection is created for every HTTP request and response when not using this option. Obviously, this option cannot be used while using a proxy. If back end HTTP server supports null connections, `Sqlmap` can utilise them. Null connections allow `Sqlmap` to retrieve page length without receiving actual response body. `Sqlmap` can perform queries in up to ten threads. While retrieving output from database character by character (see blind techniques in subsection 3.2.1), `Sqlmap` uses inference algorithm for sequential statistical prediction. It builds a statistical table based on most common sequences of characters. This may drastically speed up the detection, but the optimization technique cannot be used with multiple threads. A user can provide his or her own statistical tables if they have some knowledge about common table names or other values which can help in output prediction.

Another feature offered by `Sqlmap` are so called tamper scripts. They are simple Python2 scripts, which perform certain transformations on malicious data just before they are sent. They may help to evade some simple web application firewalls. For example, they perform base64 encoding, conversion to the upper or the lower case, placement of random blank characters etc.

It is possible to specify range of performed tests. `Sqlmap` has two main criteria for this: level and risk. Higher level means that more thorough testing will be performed. For example, levels higher than 3 test also Cookie, Host, User-Agent and Referer headers. On the other hand, risk determines potential danger of performed tests. Higher risk value means higher risk of data corruption or modification.

As stated earlier, a user can help `Sqlmap` to distinguish between true and false pages while using blind detection techniques by specifying a string or regular expression, which can be found on true or false page. A user can also force `Sqlmap` to process only text part of a page, without JavaScript and other embedded objects. This can help while auditing a site with lots of dynamic content, e.g. adds or banners.

Enumeration of a database

`Sqlmap` offers a broad range of ways of exploiting found SQL injection vulnerabilities. Please understand that some exploitation and enumeration features require certain conditions to be executed successfully, for example specific back end database engine, insertion into a specific SQL query or sufficient permissions for database user. `Sqlmap` can perform thorough fingerprinting of back end database engine, enumerate columns, tables, schemas and databases. It can even dump hashes of passwords and start a dictionary attack against them. It is possible to dump parts of the database, search for values, execute arbitrary SQL statements or launch a real SQL shell including history and TAB completion. Advanced features include loading of user defined functions, execution of commands on back end operating system, downloading and uploading of files, access to Windows registry or spawning of Meterpreter shell through payloads created with Metasploit.

Other useful features

Sqlmap can be run in fully automatic mode thanks to *––batch* and *––answers* switches. The first one tells Sqlmap not to ask any interactive questions and go with predefined defaults, the second switch allows a user to redefine answers to certain questions. Sqlmap can also save all HTTP traffic into a file for later analysis. There is no need to type all command line switches every time you run Sqlmap, it is possible to store them in configuration file and load them with just one switch. For every site, Sqlmap maintains a session file, which keeps track of testing. This is useful if a testing process is interrupted and later resumed.

3.3 Integration of Sqlmap with Atlassian Bamboo

Sqlmap can be integrated into Atlassian Bamboo environment. However, the integration process is not so straightforward as with other tools mentioned in this thesis. It is caused by the way in which Sqlmap receives input vectors (see 3.2.2).

The biggest limitation is Sqlmap's crawling mechanism which parses only regular links and forms represented by <a> and <form> HTML tags respectively. This method is insufficient for contemporary web applications. Links and forms are often replaced by more sophisticated controls written in JavaScript or other languages. If a user needs to probe such application for SQL injection vulnerabilities, it is needed to perform additional steps before running Sqlmap.

If Sqlmap is not able to find desired input vectors on its own, they need to be supplied either in form of file containing list of URLs or a set of files containing HTTP requests (see listing 3.1). It might be helpful to consult this part with developers, because it is important to cover every case of user input being passed to an SQL engine and at the same time ignore irrelevant inputs.

If the problem with supplying of desired input is solved, it is advised to create a configuration file for better management of Sqlmap options. Sqlmap is run from a script which in turn is run as a Bamboo script task. The script takes care of running Sqlmap, monitoring its progress and passing correct return value to Bamboo, because Sqlmap always returns 0 even if it detects a vulnerability. If Sqlmap gets input

vectors from HTTP request files, the script has to launch separate `Sqlmap` instance for every file. `Sqlmap` has to be ran in batch mode.

It is a good idea to let `Sqlmap` generate a traffic file containing HTTP traffic generated during the scan and keep this file as a task artifact for later analysis. Standard and error output may be redirected into a file for analysis as well.

An example script which can be run as bamboo task can be found in section B.2. The script performs all actions mentioned above. Additionally, it creates one output file called `results` which contains numbers of requests which led to discovery of an SQL injection flaw. It also contains a minimalistic HTTP client which can authenticate against the application and provide `Sqlmap` with session cookie.

3.4 Real usage

This section briefly describes my methodology for auditing a web application with `Sqlmap` and contains an example of `Sqlmap` usage as well as my latest acquired results.

3.4.1 Collecting data

Firstly, it is needed to collect enough data for `Sqlmap`. One way of doing this is to supply `Sqlmap` with hand-picked URLs and let it do its work. Another possibility is to use already mentioned crawling feature of `Sqlmap`, which recursively walks through all links present on the supplied site up to specified depth and searches for possible injection points. But this method would miss a lot of input points present in HTTP POST requests. Contemporary Java-based web applications often use this method to pass information between client and the application. Therefore, the only way which I found useful while auditing such an application, is to collect all HTTP traffic while performing normal actions with the application. It is important to walk through all available functional paths present in the application.

Data can be collected in several ways. I used two applications: Wireshark and Mitmproxy. Wireshark is a well-known tool for capturing and analysing network traffic. Mitmproxy is an interactive man-in-the-middle proxy, which can intercept and eventually modify HTTP

traffic (see subsection 2.3.2). I developed a small framework, which extracts HTTP traffic as well as individual HTTP requests from pcap files and Mitmproxy logs. It is described in section 3.6. At first I used Mitmproxy running on my laptop running Linux. But I encountered situation when I needed to use a computer running Windows operating system. It is quite difficult to run Mitmproxy on Windows and therefore I decided to use Wireshark in this case.

At the end of the collection phase, I had one file with all HTTP conversations including requests and responses, as well as collection of small files, where every file contained one HTTP request. These individual requests were ready to be used by `Sqlmap` as input data. I collected several thousands of requests and I needed to separate those containing possible injection points. As I did this for the first time, I did it manually, but I believe this process can be automated up to certain point based on knowledge of application logic.

3.4.2 Performing tests

At the beginning, an active session with the application has to be established, if needed. This can be done manually or automatically. The important piece of information is the cookie, which is going to be used further. This step is required only at the beginning of testing, or after the session is destroyed.

Next, `Sqlmap` is run with desired parameters, including valid session cookie and desired file with HTTP request. This repeats for every request, which has been selected for testing. This part can be fully automated with a script, which runs `Sqlmap` with desired parameters, checks for result, prepares the next request and so forth.

I used following command to run `Sqlmap` on a single request.

```
sqlmap -r request, --cookie="JSESSIONID=XXXXXXXXXXXXXXXXXX"
--level=5 --risk=3 --threads=8  --dbms=postgresql
--keep-alive --answers="result=n,potential=n" --beep
--invalid-bignum --skip=Cookie --batch -t
sqlmap.out -v3 | tee sqlmap.log
```

-r specifies a file with HTTP request

--cookie specifies a valid session cookie

--level specifies how thorough testing is going to be performed

--risk specifies risk of possible data corruption during the testing because of dangerous queries

--threads specifies number of concurrent threads to be used during the testing

--dbms gives `Sqlmap` a hint about back end database management system

--keep-alive is an optimization technique which uses persistent HTTP connections

--answers specifies preconfigured answers to some questions in the form `"text_contained_in_question=reply"`, where reply is y or n

--beep beeps whenever `Sqlmap` finishes or an interactive response is required

--invalid-bignum instructs sqlmap to use big numbers for invalidating of parameters, e.g. 18 = 999989

--skip prevents some parameters from being tested, because testing of session cookie invalidated session in this case

--batch configures `Sqlmap` to run in batch mode, so no interaction is needed

-t instructs `Sqlmap` to output all HTTP traffic generated during testing into a file

-v sets a verbosity level, level 3 outputs debugging information including injected payloads

tee sqlmap.log uses classic Unix command to save the standard output into a file

An example shortened output is available in section A.2.

3.4.3 Analysis

If `Sqlmap` finds positive results for SQL injection, it may be worth to further investigate generated files, or rerun the interesting part with different configuration options. This part is probably quite hard to automatize, except for parsing of logged `Sqlmap` output. `Sqlmap` issues a warning when it thinks it found an injection but cannot verify it.

3.5 Results

I chose two deliberately vulnerable open-source web applications with known SQL injection vulnerabilities and I let `Sqlmap` detect them. Following sections summarize my results.

In both cases, I used method described in the subsection 3.4.1.

3.5.1 Mutillidae

For this evaluation, I used Mutillidae version 2.6.16. Mutillidae is a PHP based deliberately vulnerable web application used for educational purposes [11]. I used `Sqlmap` version 1.0 cloned from git repository at the time of test (December 2014).

This version of mutillidae contains 16 SQL injection vulnerabilities [10]. The list of vulnerabilities does not distinguish among particular types of SQL injection vulnerabilities. `Sqlmap` was able to detect some vulnerabilities with multiple techniques. `Sqlmap` detected all of them. The following list shows various types of injectable input vectors contained in the application.

- URL parameters (3)
- form fields (7)
- cookies (2)
- HTTP referer header (1)
- HTTP User-agent header (1)
- REST web service parameter (1)
- SOAP web service parameter (1)

3.5.2 WebGoat

As I cooperated with YSoft while researching tools for this thesis, we decided to use deliberately vulnerable application based on Java technology.

I chose OWASP WebGoat version 6.0.1. WebGoat is deliberately vulnerable web application containing lessons demonstrating various vulnerabilities including SQL injection [27]. I chose this application

because lessons are clearly separated and it is easy to evaluate individual lessons with security tools without unwanted interference with other parts of the application.

WebGoat includes 9 lessons containing SQL injection vulnerabilities. The injection point is always a parameter submitted through a form. Two lessons present blind SQL injection vulnerabilities. `Sqlmap` detected all vulnerabilities. However, it used boolean-based blind detection technique in all cases.

While performing tests on WebGoat, I encountered a bug in `Sqlmap`. `Sqlmap` incorrectly recognized HSQL database used in WebGoat as MySQL database. This was fixed by the developer [43].

3.6 Data collection framework

While collecting input data for `sqlmap`, I created a small framework for extraction of HTTP requests from PCAP files and Mitmproxy logs [38]. PCAP files are generated by network traffic analysers such as Wireshark. I selected those formats because I used Wireshark and Mitmproxy for data collection. The framework is written in Python2. It consists of 3 parts: PCAP parser, Mitmproxy parser and Content identifier.

PCAP parser and Mitmproxy parser produce two types of output: traffic file and individual requests. The traffic file (`traffic.log`) contains HTTP requests and responses as they were parsed from an input file. Every request and response is assigned a number depending on its order in a traffic flow. This file is used mainly by a user to search for responses to particular requests. Individual request files are prepared to be used as input for `Sqlmap`. They are named `request#` where `#` signifies a request number which corresponds to a number in the traffic file.

3.6.1 PCAP parser

The PCAP parser uses the Scapy library for processing of PCAP files. Scapy is an interactive packet manipulation program which can decode, capture, and manipulate network packets [6]. My framework

currently does not support live capturing of data. It can parse files in PCAP format produced for example by Wireshark or Tcpdump.

3.6.2 Mitmproxy parser

The Mitmproxy parser can process logs created by Mitmproxy and extract HTTP requests. It also contains an option to perform live capture of traffic but its configuration is very limited and therefore I recommend creating logs with Mitmproxy and parse them later. It is actually built on top of Mitmproxy so it requires it to be installed.

3.6.3 Content identifier

While using my framework I encountered HTTP requests and responses containing binary payloads (images, icons, PDF files…). This content was not useful for me and it prevented me from viewing the traffic file because text editors could not interpret it properly. This problem let to creation of Content identifier module. The module is designed to be used as a helper module. It keeps two lists of Content-type HTTP headers: accepted, and rejected. It accepts a HTTP request or response as input. Firstly it examines its Content-Type header and tries to find it in the list of accepted or rejected types. If it finds a match, it accepts or rejects the message respectively. If there is no match, it displays found Content-Type header to a user and asks if they wish to accept it, reject it, or try to decode and display it. If a user decides to display it, the module tries to decode body content, replacing characters which could cause problems while being displayed with the U+FFFD Unicode replacement character. Then it again asks a user what to do. The module accepts custom function for decoding of HTTP messages.

4 W3af

W3af stands for Web application audit and attack framework. This chapter summarizes its notable features, covers available plugins categorized according to OWASP Top 10 and presents my results of auditing open-source applications with W3af.

4.1 Features

W3af is written in Python2. It can run on Linux and OS X . It is also distributed in form of a Docker image, which is currently the only solution for running W3af on Windows. It can be controlled through interactive console interface, graphical GTK-based interface or a REST API. Running W3af in non interactive mode is an option.

4.1.1 Highlights

W3af contains 165 plugins divided into 10 categories. The following list shows categories, a number of available plugins, and short description of the category.

- Attack (8) - plugins try to exploit discovered vulnerabilities
- Audit (32) - plugins try to discover vulnerabilities
- Authentication (2) - plugins allow establishing an authenticated session with audited application
- Brute force (2) - plugins try to guess authentication information through brute force
- Crawling (31) - plugins try to harvest possible input vectors
- Evasion (11) - plugins modify HTTP requests sent by W3af to evade web application firewalls
- Grep (45) - plugins passively analyse HTTP traffic, searching for interesting information and signs of vulnerabilities
- Infrastructure (26) - plugins probe the application for information about its infrastructure (used technologies, load balancers etc.)

- Mangle (1) - this category contains one plugin, which serves as a stream editor for HTTP requests
- Output (7) - plugins generate reports in different formats

`W3af` can audit query strings, POST data, headers, cookie values, forms, URLs and URL parts. Unfortunately, compared to Arachni (see subsection 5.1.1), `W3af` is not able to audit some elements such as UI inputs, JSON or XML input data etc.

Although the Crawling category contains many plugins, not all of them are suitable for internal audits. Several plugins use third-party services (Google, Bing, Google hacking database, Archive.org) and therefore they are of no use during audits of internal applications. The framework can use automatic and manual crawling methods. Manual crawling is realized with help of a proxy. A user walks through the application while HTTP traffic passes through the proxy and `W3af` extracts input vectors.

`W3af` can exploit some vulnerabilities. It uses payloads written in Python2 or generated with Metasploit. Exploitable vulnerabilities include SQL injection, file inclusion vulnerabilities, WebDAV vulnerabilities, OS commanding vulnerabilities and XPath vulnerabilities.

Generation of reports is accomplished through Output plugins. Currently, it supports output to a console, HTML, plain text, CSV and XML. It is possible to send the reports via email.

4.1.2 Audit plugins

The table 4.1 lists audit plugins along with the classification of detected vulnerabilities according to OWASP Top 10.

Table 4.1: W3af audit plugins

Name	OWASP Top 10 mapping	Name	OWASP Top 10 mapping
Blind SQL injection	A1	OS commanding	A1
Buffer overflow	A1	Phishing vectors	A10
CORS origin	A5	Preg_replace	A1
CSRF	A8	ReDos	
DAV	A5	Response splitting	A1
Eval	A1	RFD	A1
File upload	A5	RFI	A1
Format string	A1	Shell shock	A9
Frontpage	A9	SQL injection	A1
Generic	depends on case	SSI	A1
Global redirect	A10	SSL certificate	A5
Htaccess methods	A5	UnSSL	A5
LDAP injection	A1	Websocket hijacking	A8
LFI	A1	XPath	A1
memcache injection	A1	XSS	A3
MX injection	A1	XST	A3

4.2 Integrating W3af into Atlassian Bamboo

W3af can be integrated with Atlassian Bamboo. Again, the best solution is to use a Bamboo task script. The script ensures that W3af receives a configuration profile and returns an appropriate value depending on scan results.

The configuration profile can be created through console or graphical interface. The profile should contain at least the following information:

- a base URL of an audited application
- enabled appropriate crawling plugins with necessary options (excluded URL patterns)
- enabled authentication plugin with necessary options (if performing authenticated scans)
- enabled Console output plugin and at least one other output plugin to generate a report file
- enabled other desired plugins
- scan time-out (if needed)

If the Manual spider plugin is enabled, W3af creates a proxy server listening on configured address and port. Please note that the scan will not start before this plugin is terminated by visiting http://127.7.7.7/spider_man?terminate. It is also important to configure the Bamboo task to keep generated report as a task artifact for later analysis.

There exist two plugins which may save some scanning time. The Export request plugin from Output category can output all requests sent by W3af during a scan to a file. This file can be later processed by Import results plugin from the Crawling category. This may save crawling time in future scans.

4.3 Results

Because I had a good experience with WebGoat from previous Sqlmap audit, I decided to use it as a primary tool for the evaluation of three main tools selected for this thesis. I wanted to evaluate W3af's abilities

to detect vulnerabilities, and therefore I had performed this scan. I used WebGoat version 6.0.1, which was the latest stable version at that time (November 2015).

WebGoat application is structured into lessons. As mentioned earlier, `W3af`'s crawling mechanism has its limitations. WebGoat uses menus generated with JavaScript to move among lessons and `W3af` was not able to follow these links. Therefore, I had to use manual spider plugin [see 42, Performing authenticated scans]. For every lesson, I performed the following steps:

1. I logged into WebGoat.
2. I configured `W3af` to accept input through its Manual spider plugin and disabled all other crawling plugins.
3. I enabled all auditing plugins except those which were not suitable for WebGoat's technology and back end operating system.
4. I launched `W3af` and walked through the particular lesson. I performed actions which were related to the particular vulnerability (submitting of forms, clicking on links).
5. After I had walked through all related functional paths, I terminated the Manual spider plugin and let `W3af` perform audit of potential vulnerabilities.
6. I analysed generated HTML report and compared `W3af`'s results to expected results mentioned within WebGoat's lessons.

The table 4.2 summarizes achieved results. If `W3af` reported a vulnerability which was not present in the lesson or not related to the lesson, I marked it as false positive. If `W3af` did not report a vulnerability present in the lesson, I marked it as false negative. Table does not include lessons in which I was not able to exactly decide if `W3af` was successful or not.

Table 4.2: Results of W3af audit performed on WebGoat

Vulnerability	False negatives	False positives	True positives
Ajax – Dangerous use of eval	1	3	0
Ajax – DOM based XSS lab	1	4	0
Ajax – DOM injection	1	3	0
Ajax – Insecure client storage	1	3	0
Ajax – JSON injection	1	2	0
Ajax – Same origin policy	1	3	0
Ajax – Silent transactions	1	5	0
Ajax – XML injection	1	6	0
Buffer overflow – off by one	1	3	0
Code quality – hidden information	1	3	0
Injection – Add data with SQL injection	0	3	1
Injection – blind numeric SQL injection	1	3	0
Injection – Blind string sql injection	0	2	1
Injection – Command injection	0	3	1
Injection – Database backdoor	0	2	1
Injection – Modify data with SQL injection	0	1	1
Injection – Numeric SQL injection	0	5	1
Injection – String SQL injection	1	4	0
Injection – Xpath injection	1	4	0
Malicious execution of code	1	2	0
XSS – CSRF	1	3	0
XSS – CSRF with prompt	1	3	0
XSS – CSRF with token	1	2	0
XSS – Reflected XSS	0	2	1
XSS – Stored XSS	1	3	0
XSS – XST	1	2	0

The previous table does not include 26 lessons. In those lessons it was not clear what vulnerability should `W3af` search for or W3af did not include a plugin for evaluation of the vulnerability. I excluded following lessons.

- AJAX – Client side filtering
- Improper error handling – Fail-open authentication scheme
- injection – Log spoofing
- Injection – SQL injection lab
- XSS - Phishing with XSS
- General – HTTP basics
- Access control flaws – Using an access control matrix
- Access control flaws – Bypass a path based control scheme
- Access control flaws – Role based access control lab
- Authentication flaws – Password strength
- Authentication flaws – forgot password
- Authentication flaws – Multi level login
- Concurrency – Thread safety problems
- Concurrency – Shopping cart concurrency problem
- Denial of service – Denial of service from multiple logins
- Insecure communication - Insecure login
- Insecure configuration – Forced browsing
- Parameter tampering – Bypass HTML field restrictions
- Parameter tampering – Exploit hidden fields
- Parameter tampering – Exploit unchecked email
- Parameter tampering – Bypass client side JavaScript
- Session management flaws – Spoof an authentication cookie
- Session management flaws – Hijack a session
- Session management flaws – Session fixation
- Web services – WSDL scanning
- Web services – Web service SQL injection

- Web services - Web service SAX injection

While evaluating part of Y Soft products with `W3af`, I encountered a problem which prevented me from further evaluation. Number of HTTP requests sent by W3af continuously dropped during the scan up to a point when no requests were sent at all and the scan would probably have run indefinitely. I consulted this problem with the author of `W3af` but I could not provide him with relevant information because of NDA between me and Y Soft corporation. According to [41] the problem is difficult to detect. Therefore, I decided to move away from `W3af` to `Arachni` for evaluation of Y Soft products.

5 Arachni

This chapter looks at Arachni vulnerability scanner in greater detail. It describes its features, categorizes its checks according to OWASP Top 10, analyses methods of integrating Arachni into Atlassian stack and in the end, presents results of Arachni scanning open-source vulnerable web application.

5.1 Features

Arachni is an open-source web application vulnerability scanner written in Ruby. It is actively developed. Packages are available for Linux, OS X and Windows operating systems. It can be controlled through a command line, a web interface or a REST interface.

The following subsections provide overview of Arachni's features, checks and plugins. For detailed information please see [17].

5.1.1 Highlights

Arachni comprises 65 checks: 30 active and 35 passive. The first category of checks actively probes for vulnerabilities by submitting requests and reading responses. Passive checks either look for interesting files and folders or passively monitor traffic flowing between Arachni and the application. Arachni also contains 19 additional plugins, which enhance its functionality. A combination of enabled checks, plugins and additional configuration can be saved as a profile, which may be later used to perform an actual scan.

Arachni uses its own browser environment utilizing PhantomJS library [13]. Therefore, it can audit modern web applications which make use of JavaScript, HTML5, AJAX etc. This is accomplished by monitoring DOM transitions, JavaScript data and execution flows. Arachni actually behaves like a JavaScript and DOM debugger. It even contains specific hooks for popular JavaScript frameworks JQuery and AngularJS for easier data collection. To be more specific, Arachni can audit forms, user-interface forms, user-interface inputs, cookies, headers, generic client-side elements, AJAX request parameters and request data formatted as JSON or XML.

Authenticated parts of an application do not cause serious problems to Arachni. A user can configure Session check URL, which is periodically checked for supplied text pattern. If the pattern is not present, Arachni decides that the session has been destroyed and tries to re-establish it, if possible.

Arachni is also prepared to be ran as a grid for better scaling of resources and bandwidth. It is possible to split the audit among several Arachni instances. Every instance runs a browser cluster with several browser jobs. The grid automatically and completely transparently spreads jobs evenly among all instances. Communication among instances is realized through RPC calls and it is encrypted. Arachni supports line aggregation, allowing grouping together instances which are using the same line bandwidth. This prevents unnecessary line congestion.

The framework allows detailed customization of scan parameters. They include HTTP headers, cookies, predefined input to be used with forms, HTTP request concurrency, screen resolution of a browser etc. Other notable features include SOCKS and HTTP proxy support, site authentication (form-based, NTLM, digest, Kerberos…), full SSL support, ability to suspend and resume scans and more.

Arachni generates reports in its native AFR format. Such report is prepared to be transformed into other formats by so called reporters. Currently, Arachni contains reporters for HTML, XML, plain text, JSON, Marshal, and YAML. The HTML reporter generates a Zip archive containing HTML pages, CSS styles and JavaScript. Resulting report is interactive and contains summary of all vulnerabilities including graphs and OWASP Top 10 classification, detailed information about every vulnerability including remediation guidance, discovered site map etc.

5.1.2 Arachni checks

The table 5.1 and the table 5.2 list Arachni active and passive checks respectively. They also show matching category of vulnerability classified according to owasp Top 10 which is detected by the check. Note that active checks mostly search for injection, XSS and CSRF flaws, whereas passive checks mostly cover security misconfiguration and sensitive data exposure flaws.

Table 5.1: Arachni active checks

Name	OWASP Top 10 mapping	Name	OWASP Top 10 mapping
Code injection	A1	SQL injection	A1
Code injection (php://input wrapper)	A1	SQL injection (differential analysis)	A1
Code injection (timing)	A1	SQL injection (timing attack)	A1
CSRF	A8	Trainer	
File inclusion	A1	Unvalidated redirect	A10
LDAP injection	A1	Unvalidated DOM redirect	A10
NoSQL injection	A1	Blind XPath injection	A1
Blind NoSQL injection (differential analysis)	A1	Blind XSS	A3
OS Command injection	A1	DOM XSS	A3
OS command injection (timing)	A1	DOM XSS in script context	A3
Path traversal	A1	XSS in HTML element	A3
Response splitting	A1	XSS in path	A3
Remote file inclusion	A1	XSS in script context	A3
Session fixation	A2	XSS in HTML tag	A3
Source code disclosure	A5	XXE	A1

Table 5.2: Arachni passive checks

Name	OWASP Top 10 mapping
Allowed methods	
Backdoors	A5
Backup directories	A5
Backup files	A5
Captchas	
Common administration interfaces	A5
Common directories	A5
Common files	A5
Cookie set for parent domain	A2
Credit card number disclosure	A6
CVS/SVN users	A6
Directory listing	A5
E-mail address disclosure	A6
Form-based file upload	
.htaccess limit misconfiguration	A5
HTML objects	
HTTP HSTS	A6
HTTPOnly cookies	A2, A5
HTTP PUT	A5
Insecure client-access policy	A5
Insecure cookies	A2, A5
Insecure CORS policy	A5
Insecure cross-domain policy (allow-access-from)	A5
Insecure cross-domain policy (headers)	A5
Interesting responses	
Localstart.asp	A2
Mixed resource	A6

Table 5.2: Arachni passive checks continued

Name	OWASP Top 10 mapping
Origin spooff access restriction bypass	A5
Password field with auto-complete	A6
Private IP address finder	A6
SSN	A6
Unencrypted password forms	A6
WebDAV	
Missing X-Frame-Options header	A5
XST	A3

5.1.3 plugins

Following paragraphs describe several plugins shipped with `Arachni`. I chose plugins which may be helpful while integrating `Arachni` into a continuous integration process.

The Proxy plugin launches a HTTP/S proxy. This proxy can be used to supply `Arachni` with input vectors by browsing specified application through the proxy. This method can be used to audit highly specific parts of an application or input vectors which cannot be detected by `Arachni` crawling mechanism. The proxy also contains login sequence recorder. While performing authenticated scan, this feature is used to record authentication process. If an authenticated session is destroyed, `Arachni` uses this recorded process to establish a new session.

Beside the login sequence recorder, `Arachni` contains two other plugins which are used for establishing an authenticated session. The Autologin plugin can perform form-based authentication. The Autologin_script plugin expects a Ruby script and it can actually perform any action resulting in an active session e.g. going through a multi-factor authentication process or gaining authentication tokens from external source.

The EmailNotify plugin sends notification or complete scan report through SMTP at the end of an audit. The Exec plugin can run external executable programs at different stages of an audit.

5.2 Integration of `Arachni` into Atlassian Bamboo

`Arachni` can be integrated with Atlassian Bamboo. Firstly, a proper profile has to be created. The following important parameters should be defined in the profile:

- required active and passive checks
- required part of an application to be audited (can be limited through excluded or included vector patterns, URL patterns etc.)
- information required for establishing authenticated session (if auditing authenticated part of an application)
- a format of a report generated by `Arachni`
- an optional scan time-out to prevent extremely long running Bamboo task (there is a configuration setting which suspends scan after reaching the time-out and the scan can be resumed later if needed)

It is important to configure the task to keep `Arachni` report as a task artifact so that it can be analysed later. Please note that if the Proxy plugin is enabled, the scan will not start before the plugin is terminated by visiting http://arachni.proxy/shutdown.

`Arachni` contains a feature which may save some crawling time during a scan. The Vector collector plugin can collect all discovered input vectors and save them in a YAML file. The Vector feed can import this file and use input vectors contained in it. This method can save time, because `Arachni` does not need to crawl whole application every time a new scan is run.

5.3 Results

As with two previous tools, I evaluated `Arachni`'s features with help of OWASP WebGoat application. I used similar process as with W3af. The description follows.

1. I logged into WebGoat.
2. I enabled the Proxy plugin and set scope page limit to 0, effectively disabling automatic crawling.
3. I enabled all active and passive checks.
4. I walked through the lesson, performing necessary steps (submitting forms, activating links).
5. I terminated the proxy and let `Arachni` perform the audit.
6. I compared generated HTML report with vulnerabilities mentioned in a description of the particular lesson.

I audited only lessons which were audited with `W3af` to provide exact comparison results (see section 4.3 for list of lessons).

The results were very surprising for me. `Arachni` did not manage to detect any vulnerability connected with any particular lesson. For every lesson, it announced two CSRF vulnerabilities and warned about a missing X-Frame-Options HTTP header. However, this means that for every tested lesson (see the table 4.2), `Arachni` detected 1 false negative and 3 false positives.

I contacted the author of `Arachni` and notified him about this strange behaviour. -we are currently investigate this problem. Unfortunately, neither me nor the author haven't found a cause of this anomaly yet.

While I was performing the audit, I noticed that scans take extremely long time and they even failed to finish. If I tried to interrupt a scan prematurely, it failed to finish and therefore I was not able to get even partial results. I notified the author about this problem. He discovered that `Arachni` tried to capture website snapshot even if requesting of that site had failed. This prevented scans from finishing. The bug was fixed in the development version [18].

6 Conclusion

The thesis researched and compared a broad range of tools for dynamic security analysis of web applications. Features of relevant tools were compared, as well as their general parameters. Three tools underwent deeper analysis including their evaluation on open-source deliberately vulnerable web applications. During the evaluation I notified authors of `Sqlmap` and `Arachni` about problems which arose during the evaluation. During evaluation of `Sqlmap` I created a framework for extraction of HTTP requests which are used as input for `Sqlmap`.

The thesis demonstrated that approach of vulnerability scanners has to change according to modern trends in web technologies. This can be clearly seen in case of `W3af` which failed to crawl application written in Java and it needed to be guided through it. This makes the tool unsuitable for regular vulnerability scanning.

However, tools researched in this thesis occupy only a part of the market and there remain other tools which perform dynamic analysis and have not been evaluated yet (IBM AppScan, WebInspect, Tinfoil security, Skipfish, Wapiti…) [8]. That means that it is certainly possible to continue in this research. It is encouraged to include other tools and compare their results. Research can also take into account different deliberately vulnerable applications (The bodge IT store, Wavsep…).

Bibliography

[1] Acunetix. (2015). Acunetix web vulnerability scanner brochure, [Online]. Available: https://www.acunetix.com/resources/wvsbrochure.pdf (visited on 02/01/2016).

[2] Acunetix. (2016). Vulnerabilities. List of web vulnerabilities detected by Acunetix WVS, [Online]. Available: https://www.acunetix.com/vulnerabilities/web/ (visited on 05/03/2016).

[3] Acunetix. (2015). Web application security with acunetix web vulnerability scanner, [Online]. Available: https://www.acunetix.com/vulnerability-scanner/ (visited on 02/01/2016).

[4] L. Ahn, M. Blum, N. J. Hopper, and J. Langford, "Advances in cryptology — eurocrypt 2003: International conference on the theory and applications of cryptographic techniques, warsaw, poland, may 4–8, 2003 proceedings", in, E. Biham, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ch. CAPTCHA: Using Hard AI Problems for Security, pp. 294–311, ISBN: 9783540392002. DOI: 10.1007/3-540-39200-9_18. [Online]. Available: http://dx.doi.org/10.1007/3-540-39200-9_18 (visited on 12/15/2015).

[5] S. Bennetts. (2015). Zed attack proxy wiki, [Online]. Available: https://github.com/zaproxy/zaproxy/wiki (visited on 02/01/2016).

[6] P. Biondi. (2016). Scapy, [Online]. Available: http://www.secdev.org/projects/scapy/ (visited on 05/05/2016).

[7] BlackArch Linux. (2016). Blackarch linux | penetration testing distribution. List of tools in BlackArch Linux, [Online]. Available: http://www.blackarch.org/tools.html (visited on 02/01/2016).

[8] S. Chen. (Jul. 1, 2015). The prices vs. features of web application vulnerability scanners,

[Online]. Available: http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html (visited on 04/16/2016).

[9] A. Cortesi. (2015). Introduction - mitmproxy 0.16 documentation,
[Online]. Available: http://docs.mitmproxy.org/en/stable/ (visited on 02/01/2016).

[10] J. Druin. (Sep. 2014). List of vulnerabilities in mutillidae version 2.4.16,
[Online]. Available: https://sourceforge.net/p/mutillidae/git/ci/cfb44365eab6e259286547f74bf693fa0e0a013e/tree/documentation/vulnerabilities.php (visited on 04/15/2016).

[11] J. Druin. (Mar. 2016). Owasp mutillidae 2, [Online]. Available: https://sourceforge.net/projects/mutillidae/ (visited on 04/15/2016).

[12] B. D. A. Guimaraes and M. Stampar. (2015). Sqlmap wiki,
[Online]. Available:
https://github.com/sqlmapproject/sqlmap/wiki (visited on 12/04/2015).

[13] A. Hidayat. (2016). Phantomjs, [Online]. Available:
http://phantomjs.org/ (visited on 05/09/2016).

[14] Hood3dRob1n. (Mar. 20, 2015). Sqlmap-web-gui, Php frontend to work with the sqlmap json api server (sqlmapapi.py) to allow for a web gui to drive near full functionality of sqlmap!,
[Online]. Available:
https://github.com/Hood3dRob1n/SQLMAP-Web-GUI (visited on 02/16/2016).

[15] G. S. Kalra, *Tessercap - a visual captcha solving tool*,
TesserCap user guide found inside the installation package,
Foundstone Professional Services, 25 pp. [Online]. Available:
http://www.mcafee.com/us/downloads/free-tools/tessercap.aspx (visited on 02/01/2016).

[16] T. Laskos, *Arachni user guide*, 2016. [Online]. Available:
https://github.com/Arachni/arachni/wiki/User-guide (visited on 02/01/2016).

[17] T. Laskos. (Feb. 2016). Arachni: Web application security scanner framework,
[Online]. Available: https://github.com/Arachni/arachni (visited on 04/09/2016).

[18] T. Laskos. (Apr. 13, 2016). Browser#trigger_event: Don't capture snapshot on failure. a commit into Arachni experimental branch as a solution to problem encountered while auditing a web application,
[Online]. Available: https://github.com/Arachni/arachni/commit/9c92c4ccb1950ccf54f8e6726dfebbad7dcb360b (visited on 04/16/2016).

[19] N. MacDonald. (Jan. 19, 2011). Static or dynamic application security testing? both!, [Online]. Available: http://blogs.gartner.com/neil_macdonald/2011/01/19/static-or-dynamic-application-security-testing-both/ (visited on 11/03/2015).

[20] M. May, Ed. (Nov. 23, 2005). Inaccessibility of captcha, Alternatives to visual turing tests on the web. W3C Working Group Note, W3C,
[Online]. Available: https://www.w3.org/TR/turingtest/ (visited on 02/01/2016).

[21] A. Mesbah, A. van Deursen, and S. Lenselink, "Crawling ajax-based web applications through dynamic analysis of user interface state changes",
ACM Trans. Web, vol. 6, no. 1, 3:1–3:30, Mar. 2012,
ISSN: 1559-1131. DOI: 10.1145/2109205.2109208. [Online]. Available: http://doi.acm.org/10.1145/2109205.2109208 (visited on 12/10/2015).

[22] MorningStar Security, *Whatweb*, 2015. [Online]. Available: http://www.morningstarsecurity.com/research/whatweb (visited on 02/01/2016).

[23] National Institute of Standards and Technology,
FIPS PUB 140-2, Security Requirements for Cryptographic Modules. Gaithersburg, MD 20899-8900: Information Technology Laboratory, National Institute of Standards and Technology, May 21, 2001. [Online]. Available: http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf (visited on 02/01/2016).

[24] National Institute of Standards and Technology. (Dec. 4, 2015). Nvd - statistics result. Statistics results showing numbers of SQL injection vulnerabilities since January 2005 till November 2015., [Online]. Available: https://web.nvd.nist.gov/view/vuln/statistics-results?adv_search=true&cves=on&cwe_id=CWE-89&pub_date_start_month=0&pub_date_start_year=2005&pub_date_end_month=10&pub_date_end_year=2015 (visited on 12/04/2015).

[25] Netsparker Ltd. (2015). Netsparker web application security scanner benefits overview, [Online]. Available: https://www.netsparker.com/web-vulnerability-scanner/overview/ (visited on 02/01/2016).

[26] Open Web Application Security Project. (Aug. 26, 2013). Blind sql injection, [Online]. Available: https://www.owasp.org/index.php/Blind_SQL_Injection (visited on 02/16/2016).

[27] Open Web Application Security Project. (Feb. 5, 2016). Category:owasp webgoat project, [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project (visited on 02/17/2016).

[28] Open Web Application Security Project. (Dec. 29, 2015). Category:vulnerability scanning tools, [Online]. Available: https://www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools (visited on 02/01/2016).

[29] Open Web Application Security Project. (Feb. 2, 2016). Cross-site scripting (xss), [Online]. Available: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) (visited on 04/06/2016).

[30] Open Web Application Security Project, "Owasp top 10 - 2013", Open Web Application Security Project, Tech. Rep., Jun. 12, 2013, 21 pp. [Online]. Available: http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf (visited on 12/04/2015).

[31] Open Web Application Security Project. (2015). Owasp xenotix xss exploit framework, [Online]. Available: https://www.owasp.org/index.php/OWASP_Xenotix_XSS_Exploit_Framework#tab=Features (visited on 02/01/2016).

[32] Open Web Application Security Project. (2016). Owasp zed attack proxy project. list of features, [Online]. Available: https://www.owasp.org/index.php/ZAP#tab=Functionality (visited on 02/01/2016).

[33] Open Web Application Security Project. (Aug. 14, 2014). Sql injection, [Online]. Available: https://www.owasp.org/index.php/SQL_injection (visited on 12/04/2015).

[34] Open Web Application Security Project. (Apr. 26, 2016). Testing for sql injection (otg-inpval-005), [Online]. Available: https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005) (visited on 05/03/2016).

[35] Paterva, *Maltego transforms, A reference guide*, version 3.0, Paterva, Jan. 2011, 97 pp. [Online]. Available: http://www.paterva.com/web6/documentation/M3GuideTransforms.pdf (visited on 02/01/2016).

[36] Paterva, *Maltego version 3 user guide, Using the gui*, version 3.0, Paterva, Jan. 2011, 87 pp. [Online]. Available: http://www.paterva.com/web6/documentation/M3GuideGUI.pdf (visited on 02/01/2016).

[37] Paterva. (). Paterva / maltego. Information about Maltego pricing and editions, [Online]. Available: http://www.paterva.com/web6/sales/buy.php (visited on 02/01/2016).

[38] V. Polášek. (May 18, 2016). Httprequestextractor, [Online]. Available: https://github.com/vojtapolasek/httprequestextractor (visited on 05/18/2016).

[39] PortSwigger Ltd. (2016). Burp suite, [Online]. Available: https://portswigger.net/burp/ (visited on 02/01/2016).

[40] A. Riancho. (Jun. 15, 2015). w3af/sqlmap.py. W3af attack plugin using Sqlmap to exploit SQL injection vulnerability, [Online]. Available: https://github.com/andresriancho/w3af/blob/master/w3af/plugins/attack/sqlmap.py (visited on 02/16/2016).

[41] A. Riancho, *Re: [w3af-users] incredibly slow crawling and auditing*, conversation between me and Andres Riancho about problems encountered while scaning a web application which uses

Javascript, Jan. 13, 2016. [Online]. Available: https://sourceforge.net/p/w3af/mailman/message/34763858/ (visited on 02/16/2016).

[42] A. Riancho, *W3af documentation*, 2016. [Online]. Available: http://docs.w3af.org/en/latest/ (visited on 02/01/2016).

[43] M. Stampar. (Oct. 15, 2015). Further fixes for sqlmap to work properly with hsqldb (webgoat). a commit into Sqlmap fixing detection of HSQL databases submitted as a result of Sqlmap audit of WebGoat, [Online]. Available: https://github.com/sqlmapproject/sqlmap/commit/570562369b0ba1636dc733c102340e50b053f106 (visited on 04/16/2016).

[44] D. Stuttard and M. Pinto, *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*, 2nd ed. Indianapolis, Indiana: John Wiley & Sons, Inc., 2011, 912 pp., ISBN: 9781118026472.

[45] T. Tomes, *Recon-ng usage guide*, Jan. 21, 2016. [Online]. Available: https://bitbucket.org/LaNMaSteR53/recon-ng/wiki/Usage%20Guide (visited on 02/01/2016).

[46] Wikipedia contributors. (May 7, 2016). Continuous integration, [Online]. Available: https://en.wikipedia.org/w/index.php?title=Continuous_integration&oldid=719045551 (visited on 05/09/2016).

[47] M. Zalewski. (2007). Stompy readme file, [Online]. Available: http://lcamtuf.coredump.cx/soft/stompy.tgz (visited on 02/01/2016).

A Tool outputs

A.1 Sample Whatweb output

Here follows a verbose output of Whatweb scanner while performing scan against www.muni.cz. The scan was run with aggression level set to 3. It means that all plugins were not run, but if the initial request contained indications of information which could be confirmed with other plugins, they were run as well.

```
URL    : http://www.muni.cz
Status : 200
    ASP_NET
        Description: ASP.NET is a free web framework that enables
     great Web
                     applications. Used by millions of developers,
     it runs some
                     of the biggest sites in the world. -
    homepage:
                     http://www.asp.net/
        Version    : 4.0.30319

    Cookies
        Description: Display the names of cookies in the HTTP
    headers. The
                     values are not returned to save on space.
        String     : w3mu.global
        String     : w3mu.prev-page

    Country
        Description: Shows the country the IPv4 address belongs to
    . This uses
                     the GeoIP IP2Country database from
                     http://software77.net/geo-ip/. Instructions
    on updating the
                     database are in the plugin comments.
        Module     : CZ
        String     : CZECH REPUBLIC
```

Frame
Description: This plugin detects instances of frame and iframe HTML
elements.

HTTPServer
Description: HTTP server header string. This plugin also attempts to
identify the operating system from the server header.
String : Microsoft−IIS/7.5 (from server string)

IP
Description: IP address of the target, if available.
String : 147.251.5.231

JQuery
Description: A fast, concise, JavaScript that simplifies how to traverse
HTML documents, handle events, perform animations, and add
AJAX. − Homepage: http://jquery.com/

Meta−Author
Description: This plugin retrieves the author name from the meta name
tag − info:
http://www.webmarketingnow.com/tips/meta−tags−uncovered.html
#author

String : Masaryk University

Microsoft−IIS
Description: Microsoft Internet Information Services (IIS) for Windows

```
                    Server is a flexible, secure and easy-to-
manage Web server
                    for hosting anything on the Web. From media
 streaming to
                    web application hosting, IIS's scalable and
open
                    architecture is ready to handle the most
demanding tasks. -
                    homepage: http://www.iis.net/
    Version     : 7.5

OpenSearch
    Description: This plugin identifies open search and
extracts the URL.
                    OpenSearch is a collection of simple formats
for the
                    sharing of search results.
    String      : http://www.muni.cz/psearch_en.xml,http://
www.muni.cz/search_en.xml

Script
    Description: This plugin detects instances of script
HTML elements and
                    returns the script language/type.
    String      : text/javascript

Title
    Description: The HTML page title
    String      : Masaryk University (from page title)

X-Powered-By
    Description: X-Powered-By HTTP header
    String      : ASP.NET (from x-powered-by string)
```

A.2 Sample sqlmap output

The following listing show output of one run of Sqlmap against WebGoat. Methodology and configuration are described in the section 3.4. The output is shortened but it shows key parts of an Sqlmap audit.

```
[!] legal disclaimer: Usage of sqlmap for attacking targets
    without prior mutual consent is illegal . It is the end user's
    responsibility to obey all applicable local, state and federal
    laws. Developers assume no liability and are not responsible
    for any misuse or damage caused by this program

[*] starting at 13:31:07

[13:31:07] [INFO] parsing HTTP request from 'request'
[13:31:07] [DEBUG] not a valid WebScarab log data
[13:31:07] [DEBUG] cleaning up configuration parameters
[13:31:07] [DEBUG] checking for WebSocket
[13:31:07] [DEBUG] setting the HTTP timeout
[13:31:07] [DEBUG] setting the HTTP Cookie header
[13:31:07] [DEBUG] creating HTTP requests opener object
[13:31:07] [DEBUG] setting the HTTP Referer header to the target
    URL
[13:31:07] [DEBUG] setting the HTTP Host header to the target
    URL
[13:31:07] [DEBUG] resolving hostname '127.0.0.1'
[13:31:07] [INFO] testing connection to the target URL
[13:31:07] [DEBUG] declared web page charset 'iso-8859-1'
[13:31:07] [INFO] checking if the target is protected by some
    kind of WAF/IPS/IDS
[13:31:07] [PAYLOAD] NjYD=4961 AND 1=1 UNION ALL
    SELECT 1,2,3,table_name FROM information_schema.tables
    WHERE 2>1-- ../../../etc/passwd
[13:31:07] [INFO] testing if the target URL is stable
[13:31:08] [INFO] target URL is stable
[13:31:08] [INFO] testing if POST parameter 'account_number' is
    dynamic
[13:31:08] [PAYLOAD] 2857
```

```
[13:31:08] [DEBUG] setting match ratio for current parameter to
    0.960
[13:31:08] [INFO] confirming that POST parameter '
    account_number' is dynamic
[13:31:08] [PAYLOAD] 1632
[13:31:08] [INFO] POST parameter 'account_number' is dynamic
[13:31:08] [PAYLOAD] 101(..''(,",)
[13:31:08] [PAYLOAD] 2458-2357
[13:31:08] [WARNING] reflective value(s) found and filtering out
[13:31:08] [INFO] heuristic (basic) test shows that POST
    parameter 'account_number' might be injectable
[13:31:08] [PAYLOAD] 101'lVfB<'">uFSx
[13:31:08] [INFO] testing for SQL injection on POST parameter '
    account_number'
[13:31:08] [INFO] testing 'AND boolean-based blind - WHERE
    or HAVING clause'
[13:31:08] [PAYLOAD] 101) AND 4348=6875
[13:31:08] [DEBUG] setting match ratio for current parameter to
    0.930
[13:31:08] [PAYLOAD] 101) AND 8661=8661
[13:31:08] [PAYLOAD] 101) AND 7675=4473 AND (3653=3653
[13:31:08] [DEBUG] setting match ratio for current parameter to
    0.930
\ldots output ommitted \ldots
[13:31:08] [PAYLOAD] 101 AND 8661=8661
[13:31:08] [PAYLOAD] 101 AND 3808=1570
[13:31:08] [INFO] POST parameter 'account_number' seems to be '
    AND boolean-based blind - WHERE or HAVING clause'
    injectable
\ldots output ommitted \ldots
[13:31:10] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[13:31:10] [PAYLOAD] 101;SELECT SLEEP(5)
[13:31:10] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy
    query - comment)'
[13:31:10] [PAYLOAD] 101;SELECT BENCHMARK(5000000,MD5
    (0x69637458))#
\ldots output ommitted \ldots
```

[13:31:11] [INFO] testing 'Generic UNION query (NULL) − 1 to 20 columns'
[13:31:11] [WARNING] using unescaped version of the test because of zero knowledge of the back−end DBMS. You can try to explicitly set it using option '−−dbms'
[13:31:11] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[13:31:11] [PAYLOAD] 101 ORDER BY 1−− −
[13:31:11] [PAYLOAD] 101 ORDER BY 4237−− −
[13:31:11] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[13:31:11] [PAYLOAD] 101 ORDER BY 10−− −
[13:31:12] [PAYLOAD] 101 ORDER BY 6−− −
[13:31:12] [PAYLOAD] 101 ORDER BY 8−− −
[13:31:12] [PAYLOAD] 101 ORDER BY 7−− −
[13:31:12] [INFO] target URL appears to have 7 columns in query
[13:31:12] [WARNING] applying generic concatenation with double pipes ('||')
[13:31:12] [PAYLOAD] 101 UNION ALL SELECT 'qzkzq'||' nwMEmHVwIYPJaTYtMBorEUaVCtuLAEOybOjRAgaz'||' qzkbq',NULL,NULL,NULL,NULL,NULL,NULL−− −
[13:31:12] [PAYLOAD] 101 UNION ALL SELECT NULL,NULL, NULL,NULL,NULL,'qzkzq'||' eepITKlQdJJVvoZiEsUbHzVhRMvWswBTbvLtNGBu'||'qzkbq ',NULL−− −
\ldots output ommitted \ldots
[13:31:12] [PAYLOAD] −2952 UNION ALL SELECT NULL,NULL, NULL,NULL,'qzkzq'||'bTZwwdDVXJ'||'qzkbq',NULL,NULL −− −
[13:31:12] [PAYLOAD] −2110 UNION ALL SELECT NULL,NULL, NULL,'qzkzq'||'SOyPhjrpxj'||'qzkbq',NULL,NULL,NULL −− −

```
injection not exploitable with NULL values. Do you want to try
    with a random integer value for option '--union-char'? [Y/n]
     y
[13:31:16] [PAYLOAD] 101 UNION ALL SELECT 51,51,'qzkzq'||'
    yGJOuJUymGjLZxASAPcoefewUralMfqTKoYhZcRw'||'qzkbq
    ',51,51,51,51-- -
[13:31:16] [PAYLOAD] 101 UNION ALL SELECT 'qzkzq'||'
    dazjtlHTbeinfbrDBoBUWHYDjXYJdbogRnsQIeHn'||'qzkbq
    ',51,51,51,51,51,51-- -
[13:31:16] [PAYLOAD] 101 UNION ALL SELECT 51,51,51,'qzkzq
    '||'TzZoPRzGdVkOJaGtGokPTzLZOpLkgbzGRKYfgNhk'||'
    qzkbq',51,51,51-- -
[13:31:16] [PAYLOAD] 101 UNION ALL SELECT 51,51,51,51,51,'
    qzkzq'||'AGVsnAvvBSWDRZBPJrVXlZKzaLfdJfaAASMrpCJD
    '||'qzkbq',51-- -
\ldots output ommitted \ldots
[13:31:18] [PAYLOAD] -9683 UNION ALL SELECT
    51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,
  51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,
  51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,
  51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,
 51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51,51#
[13:31:18] [DEBUG] skipping test 'MySQL UNION query (NULL)
    - 82 to 100 columns' because the user provided a specific
    character, 51
[13:31:18] [DEBUG] skipping test 'MySQL UNION query (random
     number) - 82 to 100 columns' because the user provided a
    specific character, 51
[13:31:18] [INFO] checking if the injection point on POST
    parameter 'account_number' is a false positive
[13:31:18] [PAYLOAD] 101 AND 41=41
[13:31:18] [PAYLOAD] 101 AND 41=83
[13:31:18] [PAYLOAD] 101 AND 83=53
[13:31:18] [PAYLOAD] 101 AND 53=53
[13:31:18] [PAYLOAD] 101 AND 49=49[13:31:19] [PAYLOAD] 101
    AND 87=87
[13:31:19] [DEBUG] checking for filtered characters
```

```
[13:31:19]  [PAYLOAD] 101 AND (9476)=9476
[13:31:19]  [PAYLOAD] 101 AND 9477>9476

POST parameter 'account_number' is vulnerable. Do you want to
    keep testing the others (if any)? [y/N] sqlmap identified the
    following injection point(s) with a total of 443 HTTP(s)
    requests:
---
Parameter: account_number (POST)
    Type: boolean-based blind
    Title : AND boolean-based blind - WHERE or HAVING
    clause
    Payload: account_number=101 AND 8661=8661&SUBMIT=Go!
    Vector: AND [INFERENCE]
---
[13:31:21]  [INFO] testing MySQL
[13:31:21]  [PAYLOAD] 101 AND QUARTER(NULL) IS NULL
[13:31:21]  [INFO] confirming MySQL
[13:31:21]  [PAYLOAD] 101 AND SESSION_USER() LIKE USER()
[13:31:21]  [WARNING] the back-end DBMS is not MySQL
[13:31:21]  [INFO] testing Oracle
[13:31:21]  [PAYLOAD] 101 AND ROWNUM=ROWNUM
[13:31:21]  [WARNING] the back-end DBMS is not Oracle
\ldots output ommitted \ldots
[13:31:22]  [INFO] testing IBM DB2
[13:31:22]  [PAYLOAD] 101 AND 8148=(SELECT 8148 FROM
    SYSIBM.SYSDUMMY1)
[13:31:22]  [WARNING] the back-end DBMS is not IBM DB2
[13:31:22]  [INFO] testing HSQLDB
[13:31:22]  [PAYLOAD] 101 AND CASEWHEN(1=1,1,0)=1
[13:31:22]  [INFO] confirming HSQLDB
[13:31:22]  [PAYLOAD] 101 AND ROUNDMAGIC(PI())>=3
[13:31:22]  [INFO] the back-end DBMS is HSQLDB
[13:31:22]  [PAYLOAD] 101 AND (SELECT 5869 FROM (VALUES(0)
    ))=5869
[13:31:22]  [WARNING] running in a single-thread mode. Please
    consider usage of option '--threads' for faster data retrieval
```

```
[13:31:22] [PAYLOAD] 101 AND ASCII(SUBSTR((IFNULL(CAST("
    org.hsqldbdb.Library.getDatabaseFullProductVersion"() AS
    LONGVARCHAR),CHAR(32))),1,1))>64
[13:31:22] [PAYLOAD] 101 AND ASCII(SUBSTR((IFNULL(CAST("
    org.hsqldbdb.Library.getDatabaseFullProductVersion"() AS
    LONGVARCHAR),CHAR(32))),1,1))>32
[13:31:22] [PAYLOAD] 101 AND ASCII(SUBSTR((IFNULL(CAST("
    org.hsqldbdb.Library.getDatabaseFullProductVersion"() AS
    LONGVARCHAR),CHAR(32))),1,1))>16
[13:31:22] [PAYLOAD] 101 AND ASCII(SUBSTR((IFNULL(CAST("
    org.hsqldbdb.Library.getDatabaseFullProductVersion"() AS
    LONGVARCHAR),CHAR(32))),1,1))>8
[13:31:22] [PAYLOAD] 101 AND ASCII(SUBSTR((IFNULL(CAST("
    org.hsqldbdb.Library.getDatabaseFullProductVersion"() AS
    LONGVARCHAR),CHAR(32))),1,1))>4
[13:31:22] [PAYLOAD] 101 AND ASCII(SUBSTR((IFNULL(CAST("
    org.hsqldbdb.Library.getDatabaseFullProductVersion"() AS
    LONGVARCHAR),CHAR(32))),1,1))>2
[13:31:22] [PAYLOAD] 101 AND ASCII(SUBSTR((IFNULL(CAST("
    org.hsqldbdb.Library.getDatabaseFullProductVersion"() AS
    LONGVARCHAR),CHAR(32))),1,1))>1
[13:31:22] [INFO] retrieved:
[13:31:22] [DEBUG] performed 7 queries in 0.09 seconds
back-end DBMS: HSQLDB >= 1.7.2 and < 1.8.0


[*] shutting down at 13:31:22
```

B Atlassian Bamboo integration scripts

This chapter contains sample Python scripts which might be used for integration of three researched tools into Atlassian Bamboo continuous integration server. Continuous integration is the practice of merging of all source code copies into one repository [46]. This is performed several times a day. Continuous integration servers such as Bamboo automate this task by providing environment for building, running unit and integration tests, and committing source code into the repository.

B.1 Atlassian Bamboo

Bamboo server can handle continuous integration and deployment of software projects. Projects are not restricted to any particular programming language. It is available for Windows, Linux, Solaris and OS X operating systems. It is written in Java. It offers good integration with other Atlassian products e.g. Jira, Bitbucket.

Bamboo uses a hierarchical structure to define desired workflow. The largest unit is a project which usually represents whole application or a discrete part of an application. A project can have several plans. Every plan has one or more stages, which are run sequentially. Every stage contains one or more jobs which are processed in parallel and they may be processed on different agents depending on job requirements (operating system, software, CPU architecture…). Every job contains one or more tasks which are run sequentially on the same agent. Tasks represent the smallest units of work (pulling from a repository, compilation, performing of a unit test…). AGents are Bamboo processes capable of running a given tasks. They may be run on the same machine as Bamboo server, on a remote machine or in Amazon Elastic Compute Cloud.

B.2 Sqlmap task

This section contains two example scripts which can be used to integrate `Sqlmap` into Atlassian Bamboo.

B.2.1 sqlmap.py

Following Python script performs actual automation of `Sqlmap`.

```
#!/usr/bin/env python


import os, subprocess, sys, getopt
import client

#configuration options
SQLMAP_PATH = '/opt/sqlmap-git/sqlmap.py' #path to Sqlmap.
    py script
CONF_FILE = './sqlmap.conf' #path to Sqlmap config file
SQLMAP_ARGS = ['/usr/bin/python2', SQLMAP_PATH, '-c',
    CONF_FILE] #default Sqlmap arguments
COOKIE_REQUEST_FILE = 'cookierequest' #file name of HTTP
    request file which is used to get authentication cookie
RESULTSFILE = 'results' #filename of file with summary of results
OUTPUTDIR = 'outputs' #path to directory which will contain all
    output files
REQUESTDIR = 'requests' #path to directory containing requests
use_cookies = False

def getRequestNumber(filename):
        '''
        extracts a number from filename (request1 = 1)
        '''
        result = None
        for i in range (0, len(filename)):
                if not filename[-1 -i:].isdigit():
                        result = filename[-1 -i +1:]
                        return result

def getHostFromFile(file):
        '''
        extracts a host name from a request file
        '''
        hostname = None
```

```
        reqfile = open(os.path.join(REQUESTDIR, file), 'r')
        for line in reqfile:
                if line.startswith('Host'):
                        if line.find(':') == line.rfind(':'):
                                hostname = line[6:]
                        else:
                                hostname = line[6:line.rfind(':')]
                        break
        reqfile.close()
        hostname = hostname.rstrip() #remove trailing line endings
        return hostname



opts, args = getopt.getopt(sys.argv[1:], 'c')
for o, a in opts:
        if o == '-c':
                use_cookies = True
                print('using cookierequest')


resfile = open(os.path.join(OUTPUTDIR, RESULTSFILE), 'w')
for file in os.listdir(REQUESTDIR):
        if not file.startswith('request'):
                print('invalid filename {0} encountered'.format(
    file))
                continue
        reqnumber = getRequestNumber(file)
        print('processing request #{0}'.format(reqnumber))
        hostname = getHostFromFile(file) #needed to acquire
    information from Sqlmap output folder
        outfile = open(os.path.join(OUTPUTDIR, '{0}.out'.format(
    reqnumber)), 'w')
        args = SQLMAP_ARGS
        if use_cookies:
                c = client.Client()
```

```
                c.postFile(os.path.join(REQUESTDIR,
    COOKIE_REQUEST_FILE))
                for cookie in c.cookies:
                        args.append('--cookie=\'{0}={1}\''.
    format(cookie.name, cookie.value))
        args.append('-r')
        args.append(os.path.join(REQUESTDIR, file))
        retcode = subprocess.call(args, stdout = outfile, stderr =
     subprocess.STDOUT)
        print ('returned {0}'.format(retcode))
        outfile.close()
        logfile = open(os.path.join(OUTPUTDIR, hostname, 'log'),
    'r')
        if logfile.read() != '':
                resfile.write('Request #{0} may contain an SQL
    injection.\n'.format(reqnumber))
        logfile.close()
resfile.close()
sys.exit(0)
```

B.2.2 client.py

Following script is a helper script. It is a really minimalistic HTTP client supporting cookie storage and it is used to acquire authentication cookie by script in subsection B.2.1.

```
'''
Created on 1. 8. 2015

@author: vojta
'''


from requests import Session
import cookielib
import os
import getopt
import sys
from Cookie import SimpleCookie
```

```
class Client(Session):
    def __init__(self, *args, **kvargs):
        self.secure = False
        self.ignored_headers = {"Content-Length", "Cookie"}
        Session.__init__(self, *args, **kvargs)
        self.cookies = cookielib.MozillaCookieJar(None)

    def postFile(self, filename):
        f = open(filename, "ru")
        lines = f.readlines()
        f.close()
        headers = {}
        data = ""
        reading_headers = True
        url2 = lines[0].split(" ",)[1]
        for l in lines[1:]:
            if reading_headers:
                if l == "\n":
                    reading_headers = False
                    continue
                else:
                    name = l.split(": ")[0]
                    value = l.split(": ")[1][:-1] #removing line
    endings
                    if name in self.ignored_headers: #ignore
    certain headers
                        continue
                    headers[name] = value
                    if name == "Host":
                        url1 = value
            else:
                data += l
        if data.endswith("\n"): #remove blank line from POST
    data
            data = data[:-1]
        if self.secure == True:
```

```
            url = "https://"+url1+url2
        else:
            url = "http://"+url1+url2
        if lines[0][:3] == "GET":
            response = self.get(url, headers = headers, data =
    data, verify = False)
        elif lines[0][:4] == "POST":
            response = self.post(url, headers = headers, data =
    data, verify = False)
        return response

    def loadCookies(self, filename):
        if os.path.exists(os.path.abspath(filename)):
            print ("Cookiejar found")
            self.cookies.load(filename, ignore_discard=True)
        else:
            print ("Cookiejar not found!")
        return

    def saveCookies(self, filename):
        self.cookies.save(filename, ignore_discard=True)
        print ("Cookies saved.")
        return

def viewResponse(response):
    while True:
        ans = raw_input("Received url {0} with status code of {1}.
     Do you want to see the response? y/n".format(response.url,
    response.status_code))
        if ans.lower() == "y":
            print (response.text)
            break
        elif ans.lower() == "n":
            break
    return
```

```
if __name__ == "__main__":
    quiet = False #will ask for every request to vieq it?
    save_cookies = False #will save cookies into cookie jar?
    cookiefile = None #file for exporting of cookies
    c = Client()
    opts, args = getopt.getopt(sys.argv[1:], "c:C:sqx:")
    for o, a in opts:
        if o == "-C":
            c.loadCookies(a)
        if o == "-c":
            name = a.split("=")[0]
            value = a.split("=")[1]
            cook = SimpleCookie({value:name})
            c.cookies.set_cookie(cook)
        if o == "-s":
            c.secure = True
        if o == "-q":
            quiet = True
        if o == "-x":
            save_cookies = True
            cookiefile = a
    for r in args:
        response = c.postFile(r)
        if quiet == False:
            viewResponse(response)
    if save_cookies == True:
        c.saveCookies(cookiefile)
```

B.3 W3af task

An example script written in Python follows. It allows integration of `W3af` into Atlassian Bamboo continuous integration server.

```
#!/usr/bin/python2
```

```
import os, subprocess, sys

#configuration
W3AFPATH = '/opt/w3af/w3af_console' #path to w3af_console
    executable
PROFILENAME = 'test' #name of the W3af profile
OUTPUTDIR = 'outputs' #output directory for results
args = ['/usr/bin/python2', W3AFPATH, '-P', PROFILENAME, '|
    tee', os.path.join(OUTPUTDIR, 'output.log')]

outfile = open(os.path.join(OUTPUTDIR, 'w3af.out'), 'w')
subprocess.call(args, stdout = outfile, stderr = subprocess.
    STDOUT)
outfile.close()
sys.exit(0)
```

B.4 Arachni task

Following script written in Python2 enables integration of `Arachni` into Atlassian Bamboo.

```
#!/usr/bin/python2

import os, subprocess, sys

#configuration options
ARACHNIPATH = '/opt/arachni/bin/arachni' #path to Arachni
    executable
PROFILEPATH = './arachni.afp' #path to Arachni profile
TARGETURL = 'http://10.0.0.1' #target URL
OUTPUTDIR = 'outputs' #output directory for storing of results
ARGS = [ARACHNIPATH, '--profile-load-filepath={0}'.format(
    PROFILEPATH), TARGETURL]


outfile = open(os.path.join(OUTPUTDIR, 'arachni.out'), 'w')
```

```
subprocess.call(ARGS, stdout = outfile,  stderr  = subprocess.
    STDOUT)
outfile.close()
sys.exit(0)
```

C glossary

This chapter provides brief description of some terms used in this thesis.

AJAX (Asynchronous JavaScript and XML) is a term used for set of web client-side technologies which create asynchronous web applications.

anti-CSRF token is a sequence of characters tied to a session of a user with a web application. It is sent with every HTTP request and it should prevent CSRF attacks.

API (Application Programming Interface) is a special interface offered by an application to be used by another software or a programmer.

Basic HTTP authentication is avery simple form of authentication against web applications. User name and password are sent in unencrypted form, only Base64 encoding is applied.

Digest HTTP authentication is an authentication method used with web applications. User name and password are not sent directly. MD5 digest is sent instead.

Docker is an open-source project which allows quick deployment of software through software containers.

fuzzer is a piece of software performing fuzzing. Fuzzing involves providing unexpected, invalid, or random data to an input of an application and analysing its behaviour.

JSON (JavaScript Object Notation) is a data representation independent on underlying platform. It is used for data transfer.

Magictree is a data management and reporting tool used by penetration testers.

Marshal is a serialization format used with Python programming language.

Meterpreter shell is an executable used as a payload by Metasploit framework. This shell is specially designed to ease post exploitation of a target.

MongoDB is an open-source and free document-oriented NoSQL database.

NTLM HTTP authentication is an authentication method which uses NT Lan Manager (NTLM) authentication protocol developed by Microsoft.

SSL (Secure Sockets Layer) is an encryption and authentication layer inserted between transport and application layer.

TLS (Transport Layer Security) is an SSL successor.

WebDAV (Web-based Distributed Authoring) is a protocol for cooperation and remote management of files stored on an HTTP server.

WSDL (Web Services Description Language) is an XML based language. It is used to describe functionality of web services (how to call it, expected data, returned data…).

XPath (XML Path Language) is a language allowing addressing specific parts of an XML document.

YAML (YAML Ain't Markup Language) is a data serialization language. It can be easily interpreted by computers as well as humans.