Standardization of Bug Validation

Viktor Austli, Elin Hernborg

Bachelor's thesis in technology, 15 credits

Halmstad 2017-07-03
Abstract

The usage of the Internet is widely implemented all over the world in a number of concepts. This generates a demand of establishing security as to sustain the integrity of data. In this thesis a service will be presented which can be used to identify various web vulnerabilities in order to regulate these and therefore prevent exploitation. As the world is today the increase of technical implementation provides with a growing amount of security flaws, this affect the organizations which may have to increase their resource financing in an effort to counter these. But what if a tremendous amount of work could be automated and avoid organizations having to spend an enormous amount of finances validating security flaws reported to them? What if these flaws could be validated in a more effective manner? With this tool being establish an individual will no longer require advanced technical knowledge in order to identify whether a web vulnerability is present or not but instead have an automated test perform the procedure for them.
Acknowledgements

We would like to express our greatest gratitude towards Detectify for their unlimited support in the process of developing the standardized language, parser and tests. We are especially thankful for the support received by Johan Norman and Frans Rosén which has put their interest and time into this project.

None the less, we would also like to acknowledge Eric Järpe from whom we have received valuable guidance supporting us when composing this thesis.
Definitions

Command A word, or phrase, executed in order to achieve what the specified command represent.

Data structure This is a method of organizing information within a system in order to rationalize the wielding of it.

DNS Domain Name System, this is a system which simplify the addressing process of a computer within an IP-network such as the Internet. It works in the manner of an IP-address being connected with a domain name which may be used throughout the connection instead of the IP-address itself.

Domain A domain is a way of organizing a group of systems in order for them to share a common communication address.

HTTP Hypertext Transfer Protocol, is a protocol handling communication to transfer websites within the information network WWW, World Wide Web, on the Internet.

IDE Integrated Development Environment is a computer program which usually contain a text editor, compiler and a debugger together with various functions regarding programming.

JavaScript This is a scripting language which is dynamic, high-level and the use is mainly client-oriented regarding web applications.

Open source The definition of an open sourced program is that the source code of the software is accessible for any user to inspect, modify and enhance.

OWASP The Open Web Application Security Project is a global organization specialized in security regarding software applications, most commonly web applications.
Parser  A parser is a program, which commonly is a part of a compiler, which receive input and sort it into data structures in order to simplify the translation between different programming languages.

Regex  This is used to describe the value of strings and is composed by syntax rules. It is commonly used in programming languages and text editors as a method to search for, and manipulate, text.

Security flaw  A security flaw is a complication which pose a threat to the security of a program, system, file etcetera.

Standardized language  A standardized language is developed to create a structured process to perform a given task.

Source code  Computer instructions written in an human readable format.

Trouble-shooting  An act performed with the purpose of identifying a complication.

URI  Uniform Resource Identifier, this identifies a resource, file location within a uniform format.

Web Application  A web application is software accessible by the communication with a web browser.

XML  Extensible Markup Language, this is a markup language which both the human eye and the machine will be able to read.
Contents

Abstract i
Acknowledgements iii
Definitions v

1 Introduction 1
   1.1 Related articles 2

2 Background 4
   2.1 Motivation 5
   2.2 Purpose and Project Goal 6
   2.3 Limitations 7
   2.4 Tools 7
   2.5 Problematization 7

3 Method 9
   3.1 Theoretical studies 9
3.2 Development .................................................. 10
3.3 Validation ..................................................... 10
3.4 Problematization of Method ................................. 11
3.5 Ethical Considerations ....................................... 13

4 Theory 14
4.1 Selection of Vulnerability Coverage ......................... 14
4.2 Vulnerability Coverage ........................................ 14
  4.2.1 SQL-injection ............................................. 15
  4.2.2 LFI-vulnerability .......................................... 15
  4.2.3 IDOR-vulnerability ......................................... 15
  4.2.4 Subdomain takeover ........................................ 16
  4.2.5 .git- directory disclosure ................................. 16
  4.2.6 Stored XSS- & Blind Stored XSS-Injection ............ 16
  4.2.7 Reflected XSS .............................................. 17

5 Experiment 18
5.1 Selection of tools ............................................ 18
5.2 Development .................................................. 19
  5.2.1 Standardized language ................................. 19
  5.2.2 Parser ..................................................... 20
  5.2.3 Vulnerability tests ....................................... 21
5.2.3.1 SQL-Injection .................. 21
5.2.3.2 LFI-vulnerability .............. 22
5.2.3.3 IDOR-vulnerability .......... 22
5.2.3.4 Subdomain takeover .......... 22
5.2.3.5 .git-directory disclosure .... 23
5.2.3.6 Stored XSS- & Blind Stored XSS-Injection 23
5.2.3.7 Reflected XSS ............... 23

5.2.4 XML-specification ............... 24

5.2.5 General code ................... 24

5.2.5.1 SQL-injection ............... 27
5.2.5.2 IDOR-vulnerability .......... 27
5.2.5.3 .git-directory disclosure .... 28
5.2.5.4 Stored XSS-, Blind Stored XSS-Injection & Reflected XSS ............... 29

6 Results .......................... 30

7 Discussion ........................ 32

8 Conclusion ........................ 35

8.1 Future Work ..................... 36

Bibliography ....................... 39
A Pseudo code

A.1 Main .................................................. 44
A.2 XMLParser ........................................... 45
A.3 Test .................................................... 45
A.4 HeaderCall .......................................... 47

B Examples

B.1 SQL-Injection ......................................... 48
B.2 LFI-vulnerability ..................................... 54
B.3 .git- directory disclosure ........................... 60
B.4 Stored XSS-Injection ................................. 64
B.5 Reflected XSS ......................................... 69
1. Introduction

Along with the evolvement of technology occurrence of security flaws will be a certainty. If these security flaws are overlooked, or omitted, they may pose a threat to an entire IT-infrastructure since there is a risk that they can be exploited through e.g. cyber attacks. There have been major security flaws reported in different applications throughout time which have affected a great deal of users. One well-known incident concerning this was The Heartbleed Bug which enabled exploitation of user information regarding credit card numbers, passwords etc [1]. There are major organizations such as Adobe, Linkedin, Sony and Microsoft that have fallen victim to exploitation of security loopholes which also affected a wide range of their users [2]. To be able to patch these security loopholes they have to be discovered. There are multiple methods in order to achieve detection of security flaws, for instance the use of a Bug Bounty can be applied. Microsoft is one of the organizations which have implemented the use of a Bug Bounty-program, this allow users to hack their organization in order to expose the security flaws and if the users are successful they may receive a payment as compensation [3].

When a security flaw is discovered it is critical to adjust it, in order to guarantee that this process is executed in a proper manner an established approach is in need. This is of high significance since the individual reporting the identified security flaw may possess a higher level of knowledge than the individual receiving the material. Greater organizations are usually in possession of a well established security de-
partment with the knowledge essential to administer these findings in contrast to smaller organizations which may not. Smaller organizations are in need to retain the security as well but may not value this as highly, this does not mean that it is not crucial. This issue is brought up in the thesis *Investigation the current state of security for small sized web applications* by Lundberg [4].

1.1 Related articles

Along with the development of technology web applications continues to be targeted for security attacks, this is a major issue yet to be resolved. Since the technology evolve the number of approaches to employ grow along with it, this create a high demand on security updates to withstand the high number of security attacks possible to perform. This issue is brought up in [5] where different hacking techniques and a vulnerability analysis is presented. This is where the vulnerability scanners may come in handy for an individual, or organization with or without bad intentions. By using these a security flaw may be detected and either exploited, reported or patched. There are different ways of detecting security flaws in web applications by using vulnerability scanners, this is brought up in [6]. A description of the process which is carried out when an individual identifies security flaws with a bug bounty program is found in [7], in this thesis the deficient process regarding validation of the reported loophole is evident. As mentioned earlier different approaches to exploit applications are discovered constantly which is brought up in [8], therefore procedures to correct these security flaws is of crucial manner. This is also discussed in [9], the issue regarding lack of knowledge among individuals to accommodate security flaws is also mentioned. There have been a numerous methodologies and tools constructed to perpetuate web security but as mentioned earlier the lack of knowledge remains an issue. In [10] the fact that the diversity
in tools and methodologies accessible for assessment of security is not fully utilized is brought up, this may be a consequence due to absence of sufficient knowledge. In order to detect the security flaws which may pose a threat to an entire organization there are multiple methods to achieve this, penetration testing is a well established method and this procedure is expounded in [11]. To avoid the possibility of exploitation of applications maintenance shall be of high priority, the issue regarding this is brought up in [12]. Deficiency concerning detection of security flaws may be costly for organizations due to the absence of adequate security which enables exploitation of various nature, in [13] this potential situation is mentioned. Since human interaction is a common necessity when detecting web vulnerabilities sufficient education and knowledge may be crucial. This is brought in [14] where the demand of the participation of an individual to ensure the accuracy when performing a web vulnerability test is pointed out. As noted in [15] critical security flaws are frequently detected in web applications and one of the most dire grade of these vulnerabilities are SQL-injections. These are commonly exploited but could be obviated which is explained in [16].

Web application security is a prominent topic in the world of technology since this may affect an immense quantity of users and organizations. Projects are initiated frequently to increment security measurements although this may not unfailingly be prioritized. In [17] HTML5 is reviewed to resolve whether it has been constructed to make web applications more secure or not.
2. Background

This project has been developed through collaboration with an organization named Detectify which specializes in web security.

Throughout time the development in technology has peaked in a tremendous matter during the 21st century since it was first created, almost 40 years earlier, in the 1960s. In 1988 one of the first web security threats was spread across the Internet, the Morris worm named after its creator Robert Tappan Morris. This is considered as a significant milestone in the development of malware [18].

The Internet has been downright implemented in our society and every individual, or organization, is in many ways highly dependent on the functionality of the technology to survive. Both categories are reliant on the technology in manners of communication, economy, health care etcetera [19]. Along with the technical evolvement threats towards the industry arise, these may target the individual, organization or a wider range of users. Web applications are subjects to the attackers and by studying data presented by the SANS institute this is easy to cognize. SANS institute estimate that more than 80% of all cyber security incidents are due to already known web vulnerabilities, this is confirmed by similar data presented by the Verizon Data Breach Investigation report [20], [21]. This generate a high demand on appropriate security being advanced to attain a standard reliable to undertake these attacks. On the Internet there are no national borders which may provoke difficulties
2.1. Motivation

in how to counter the constant security threats, this create a requirement on the individuals and organizations to institute their own security measurements. If these security faults are not administrated the loss for an organization may be devastating. The Computer Security Institute has performed a survey presenting the casualties of security breaches which resulted in $445 million loss, these catastrophic numbers were presented after a review of only 223 computer professionals [22]. These facts all point to one major necessity, the implementation of sufficient security measurements and a structured process to administrate these. The web vulnerabilities has to be managed before an attack occur to ensure that the integrity of data is being sustained.

2.1 Motivation

There is no structured process of reporting a security flaw detected by an external individual to the organization involved. IT-resources and knowledge in the range of Microsoft is not a possibility for all organizations which make the phase of reporting a flaw complex in some situations. This problem may result in a security issue being neglected, if even understood. The communication between the penetration tester and the organization need to be adequate in order to guarantee the resolution of a reported security issue. There have been experiments concerning the accuracy of a vulnerability scan for example Alexander Norström examine this in his thesis *Measuring Accurancy* [sic] of Vulnerability Scanners: An Evaluation with SQL Injections although he does not cope with the complication in validating the reported flaws for an organization [15]. Since the current approaches regarding administration of reports is a major issue, concerning discoveries of security flaws, an automated method of conducting this may be of great interest for a numerous organizations which does not possess the resources essential to perform this process themselves. There is no method for performing
this process in an automated manner during the present time which may generate a demand for this service.

2.2 Purpose and Project Goal

The purpose of the thesis is to establish a tool with the potential to evaluate whether a reported web vulnerability exists or not. This should be able to test for specific vulnerabilities of different variety. It also has to be user friendly since an advanced technical knowledge should not be necessary in order to utilize the tool. The test which will be accessible for the user to execute will be an automated test, therefore the user itself does not have to possess any knowledge concerning security flaws or programming etc.

To ensure that the product developed will reach the target of being user friendly a standardized language will be established, this code will have to cover a numerous assortment of previously known web vulnerabilities. The standardized language will collaborate with a parser created to enable the evaluation of web vulnerabilities, this parser should be able to read parameters from the standardized language and therefore evaluate whether a vulnerability is present or not.

The following is to be designed and evaluated during this thesis:

- A standardized language-independent specification for reporting web vulnerability.
- A parser that can interpret the specifications into automated security tests.
- Evaluate the coverage of the parser towards a set of common categories of web vulnerabilities.
2.3 Limitations

The product developed will only detect the web vulnerabilities defined in standardized language, these test may or may not require additional information specified by the test conductor, therefore this is not a general vulnerability identifier.

2.4 Tools

This project does not require any huge assortment of tools but primarily a tool to enable the communication between the parser and any web application, defined in the standardized language, which toward the test will be conducted. An IDE, Integrated Development Environment, will be used to develop the standardized language and parser. The IDE should be efficient and effortless for the user to facilitate the process of developing the codes.

2.5 Problematization

In order to achieve the project goal of this thesis the development of a parser is crucial but to accomplish this the standardized language is essential. The conclusion of the complications concerning this thesis is that each part is necessary for a successful result. If the parser is not customized properly for the standardized language it will not be of any functionality, this represent a considerable issue therefore it will have to be developed meticulously to avoid time loss and other potential obstacles. Major issues can also occur regarding the source code, these are unpredicted and may result in time loss which can therefore pose a threat to the thesis being completed within the set time frame. The
quality of the application is in need to be verified since it will handle
the validation of security flaws and therefore it is critical that it will
be accurate. The biggest component of this thesis experiment is the
development of the standardized language and this is also crucial in
order to complete the other objectives of the project goal. The language
has to be accurate in order to assure valid results and furthermore be
compatible with the parser. Therefore the process of the standardized
language will require a larger amount of time for development, evaluation
and potentially trouble-shooting and corrections. If this is not performed
accurately the entire project goal may be at stake.
3. Method

The thesis will be accomplished by performing three steps; research to acquire knowledge, develop a standardized language and a parser and finally validate the functionality of the finalized product. The project will be performed with the structure of research, implementation and evaluation based on a similar structure of [23] and [6].

3.1 Theoretical studies

Theoretical studies will occur to receive knowledge concerning the area to enable the process of creating the standardization which needs to be readable for both the human eye and a machine. These will be performed by reading related articles to retrieve knowledge within the specific area, furthermore contact with an organization which possess relevant expertise will also occur. Regarding the theoretical studies they will be focused on theses with a higher standard rather than news articles etc. Platforms such as DiVa and IEEE Xplore will be used to acquire knowledge from previous completed theses concerning web vulnerabilities, therefore search phrases such as ”Web vulnerabilities”, ”Automatic vulnerability check”, ”standardized language”, ”XML”, ”SQL-Injection” etcetera will be applied.

The knowledge obtained will be processed to determine the approach
used for the development of the standardized language and parser. This will also be helpful while performing the validation of the functionality and if found necessary complementary research will be carried out.

Collaboration with an organization specialized in web security will occur throughout the project and will be of high value regarding acquisition of necessary knowledge to fulfill the ambitions stated.

3.2 Development

The standardized language will be developed with the requirement of being user friendly in the aspect of the user not necessarily possessing advanced knowledge regarding programming but also the demand of covering a numerous assortment of known web vulnerabilities. This is an important factor since the ambition of this project is to create a tool helpful, to identify web vulnerabilities, for any individual with or without technical knowledge.

The prerequisite of the parser is the functionality regarding parameters and structure. The parser is required to enable the receiving of parameters originating from the standardized code. These parameters should afterwards be structured in a manner suitable to enable the initiation of the vulnerability test. This is of high importance in order for the test to be accurate and avoid complications regarding the programming of the test itself.

3.3 Validation

Throughout the establishment of the tool there will be continuously validation issued to ensure its functionality in order to minimize the risk of an error being overlooked. The process of guaranteeing that
the code being precise will be followed up when the code is complete with a validation procedure. To evaluate the accuracy of the tests executed, a numerous variation of tests containing web vulnerabilities will be performed which is of significant importance in order to confirm the functionality of the tool. Since the tests will be issued by a user to identify if a vulnerability is present or not the validation process has to be authentic and executed meticulously.

An open sourced program will be used as a communication tool between the parser and the web application to increase the effectivity of certain tests. This will avoid any complications regarding the development of the parser since it otherwise would have to be able to execute JavaScripts and handle sessions. If no such tool would be accessible an enormous amount of extra programming would be essential.

\textbf{3.4 Problematization of Method}

When gathering intelligence regarding the subject of this project the demand of high quality references is in need. This may be time-consuming since various theses and other projects often collect their knowledge from varied sources, therefore it is required to perform this process meticulously to ensure the authenticity of the information acquired.

The requirement of a user friendly standardized language has to be achieved or the complete project is compromised, therefore this has to be established scrupulously with precaution.

If the parser is not developed scrutinizingly this may result in a numerous complications, these could be time-consuming and lead to delay of the entire project. Therefore it is of high importance to assure that this is fulfilled throughout the project development.

When carrying out the validation of the finalized product this has to
be done with precision, otherwise this may result in the entire project being inaccurate and therefore not useful for any individual. When validating the different tests some may be slightly harder to validate, it will although be possible to achieve but it might require a greater deal of work.

If there is no possibility to acquire access to an open sourced tool, which can handle the communication between the parser and the web application, it may be devastating for the time plan. This is due to the fact that it would require a great deal of extra code to enable various functions in order to establish this sort of communication.

Since the source code of this project will be confidential any reader of this thesis may not be able to reconstruct the tool created. An experience individual may however, from reading about the XML-code and requirements, create something similar. This is a disadvantage considering validation of the tool actually working, however it is an advantage when controlling the release of the tool and the possibilities to extend the coverage before the implementation of external use.

As for the approach of developing the vulnerability scanner there are many different methods at hand but in order to optimize the selection a method proposed in an earlier work of similar character should be in use. In [6] they propose the usage of obtaining knowledge from OWASP which also will be applied in this project. The main difference between the projects proposed here is that this concern vulnerability scanners detecting unknown vulnerabilities while the tool which will be developed throughout this project will be used to identify and report a known or supposed security flaw. Although the project conducted in the thesis is evaluated by testing different scenarios to ensure the creditability of the results, this is something of high value in a project of this nature. Therefore this procedure will be applied to this project to ensure that the veracity of the results achieve a high accuracy and credibility. In order
3.5 Ethical Considerations

to develop the scanner itself a method equivalent to the one presented in [24] will be applied, this is due to the characteristics of the product being developed in this thesis is incredibly similar to the one which will be developed in this experiment. The program presented is very similar in the sections of going through entire web pages in order to scan them for specific issues. Therefore this method, that has been shown successful, will suit the approach of the project presented in this thesis.

3.5 Ethical Considerations

Although this tool will be established with the intention of helping an organization or individual to prevent security flaws being exploited it may be used in a scenario driven by bad intentions. This tool will however not be a general vulnerability scanner and therefore a new vulnerability most likely will not be identified but rather already known vulnerabilities. A user with bad intentions may however use the tool to determine whether a vulnerability has been patched or if it is still available for exploitation. On the other hand the organization, or individual, containing the security flaw may also use the tool to identify whether it has been administrated correctly. With these potential scenarios the organization, which contain the security flaw, has every possibility to patch it against security attacks and will not be left without any opportunities to secure their web application.
4. Theory

4.1 Selection of Vulnerability Coverage

In the process of coverage selection there are a number of different attributes to take into account. Web vulnerabilities which pose a crucial threat to the administration of a web application is of high interest to confront. The security flaws presented below are considered to hold a high threat status towards web applications, some of them are also well-known and rated highly upon the list of web vulnerabilities by OWASP. The most common web vulnerabilities are not necessarily the most crucial, some of the selected security flaws may not have been prominent but still constitute a high threat status. Therefore a mixture of common web vulnerabilities with the threat they may pose towards a web application will be applied to achieve a satisfying web vulnerability coverage.

4.2 Vulnerability Coverage

There will be a various number of vulnerabilities possible to validate using the automated test developed. In this section these vulnerabilities will be listed and elucidated.


4.2. Vulnerability Coverage

4.2.1 SQL-injection

SQL-injection is a security flaw on account of unsanitized user input and is rated in the top ten security flaws by OWASP, as late as 2013 it was considered the leading vulnerability in web security. The security flaw issue a threat where a unauthorized user may insert SQL statements by exploiting user input data through a web-based application. Through exploitment of this security flaw a user can access the entire backbone database [25], [26].

4.2.2 LFI-vulnerability

LFI-vulnerability, Local File Inclusion, is a security flaw within a web application where various files may be included by an unauthorized user. These files can be included on the server machine, within the web application. The security flaw occur when there is insufficient programming to regulate the built-in methods and functions which therefore enable a user to pass selected parameters with these not being validated to ensure the privileges of the user [27], [28].

4.2.3 IDOR-vulnerability

IDOR-vulnerability, Insecure Direct Object References, may arise in a situation where the developer of an application endanger a reference of the internal implementation such as a file, directory or database key. This occur when the user is able to circumvent the authorization process by manipulating the references and may therefore acquire access to data to which they have no proper authorization [29], [30].
4.2.4 Subdomain takeover

When a domain lack validation it pose a threat to the website if it has a DNS entry pointing towards a domain that has not been claimed. If the domain has not yet been claimed there is a possibility that an outsider, rather than the organization itself, claims it and therefore acquire the possibility of serving their own content [31].

4.2.5 .git- directory disclosure

This security flaw is present if the access to .git-folders is not denied, if a production server has directory listing enabled it is also susceptible to exploitation. An unauthorized user may acquire data by issuing only one command, this is a major security issue due to the entire source code of a website would be accessible for the attacker. If the directory listing would be disabled there is a way for the more knowledgeable user to access the repository but this demand further cunning regarding how git manage this [32], [33].

4.2.6 Stored XSS- & Blind Stored XSS-Injection

This vulnerability is exploited in a method where a web application store potentially malicious input from a user. If the input that is stored is not filtered thoroughly it can lead to disastrous consequences since the data will occur to be part of the web site and will therefore be run as if it would be by the web browser of a user. Since the data appears to be part of the web site the privileges of this will be set to the same as the web application. The blind stored XSS-vulnerability is basically the same as the classic XSS-flaw, the major difference is that the attacker will not possess any knowledge of when, or where, the payload is executed.
4.2. Vulnerability Coverage

The attacker instead has to wait until the administrator of the website execute the malicious script. [34], [35].

4.2.7 Reflected XSS

This security flaw occur when an unauthorized user inject code which is browser executable within a HTTP response. The exploit pose a threat to users opening a malicious link or third-party web page exclusively due to the injection not being stored within the web application. The attacker include the attack string as a component of the URI or HTTP parameters and is subsequently processed erroneous by the web application and finally returned to the user [34], [36], [37].
5. Experiment

The ambition of this experiment is to establish a tool which can validated the given web vulnerabilities presented below. This project should be user friendly and may be evolved in the future to cover a wider range of web vulnerabilities.

5.1 Selection of tools

During the development of the standardized language XML-coding will be implemented, this due to the fact that XML is a well-known language managed by a wide range of users. This yield the opportunity to gather information, concerning the language, in order to assist the development of the project during a possible phase where additional knowledge may have to be acquired to fulfill the requirements of the code. XML-code is also a widely implemented language among projects concerning web applications and such.

The programming will be issued in IntelliJ IDEA which is an IDE, Integrated Development Environment, where the language XML, Extensible Markup Language, will be applied. The reason for this choice pedicate in the benefits of this IDE which primarily are the user-friendly environment established for the developer. During the development of the code IntelliJ IDEA will support the user by finishing inaugurated commands
5.2. Development

To increase the effectiveness of the project. The structure will be managed in an apparent manner which includes coloring and auto-indenting of the code and the program will assist the developer in debugging of the code.

To enable the establishment between the parser and the web application to conduct the necessary tests an open sourced program will be used, the tool of choice is called Selenium. Selenium allow the parser to send the commands through the web browser in order to perform the specified tests. The primary argue to use Selenium, during the development of this project, is because the program is well implemented. This contribute with a higher security of an accurate program due to the higher number of users and bugs being detected and managed in a wider range compared to a supposedly newly created program.

5.2 Development

Throughout the development phase the tests created to validate the given web vulnerabilities will remain confidential and no unauthorized individual will gain access to the programming code, this include the presentation of the project in this thesis. The tests developed will be discussed in a manner where their functionality and outcome is explained in the situation of being executed by a test performer and which aspects that has been kept in mind.

5.2.1 Standardized language

When developing the standardize language the basics regarding the URL, Uniform Resource Locator, and the port number was the first line of action. These will be used in the manner of deciding which web application the test will be conducted within and therefore needed to be
established as a ground pillar for the project. As a second approach in developing the language the possibility of including information such as login name and password was essential which will be inserted into the web application. In this step the location of the web application, where the login in will be handled, has to be specified by the test performer in order to enable the establishment a connection. When the user has specified the basic parameters the option of performing tests within the application will be possible. This may be accomplished by defining the parameters such as files and paths of the web application which shall be included in the test conducted. If necessary the forms need to be specified with further information, to ensure that the test executes correctly, which will later on be sent to the web application although this may not be essential in every test. In order to determine whether the test was successful or not the user has to compare the code expected to be faulty with the outcome, this may be performed using regex or plain text. This comparison is performed by specifying the expected outcome in the standardized language. If the outcome present that the expected code is faulty as suspected this will be displayed with the expected flawed code and a notification reporting to the user that the security flaw do exist within the web application. If the situation is the opposite and the security flaw does not exist nothing will be displayed by the outcome other than a notification that will inform the user of the vulnerability not being active within the web application.

5.2.2 Parser

In order to enable the execution of the tests the parser has been developed with the possibility of retrieving the input from the XML-file containing the standardized language given by the user. The parameters specified by the user in the XML-file will be sorted into data structures to create an unequivocal architecture which improve the management
of the parameters. Certain tests require the usage of Selenium in order to be executed to make the process more effective. The communication with Selenium works in a way where the parser contact the tool and present the given parameters. These parameters are received by Selenium which forwards it to the web application specified in the XML-file. When the web application has been approached Selenium initiate the tests given and the parser scan the outcome which it afterwards present to the user. For the tests not using Selenium only a request is specified in the XML-file which the parser manage by connecting to the web application and then compare the given information with the response retrieved from the application.

5.2.3 Vulnerability tests

For any given vulnerability the outcome of the test will indicate whether a security flaw is present or not. The user which perform the test should have no administrator privilege but instead act as an unauthorized user in order for the results of the tests to be accurate. The received outcome, which the test performer receives from each individual test, will give an unambiguous response to whether the security flaw is present or not by the lines ”The security flaw is present” or ”The security flaw is not present”.

5.2.3.1 SQL-Injection

To discover if a web application is susceptible to a SQL-Injection attack the test developed will navigate to the website where a suspected vulnerability exist. When in association with the website a SQL-request will be initiated to determine whether the security flaw is present or not. The outcome depends on whether the executed SQL-request will initiate the SQL-code given or not, if this code is executed and consummated
correctly the security flaw is present and has to be managed.

5.2.3.2 LFI-vulnerability

This vulnerability is rather straightforward when it comes to determine whether it is present or not. To conclude if this security flaw is active within a web application the performer of the test has to specify any file, within the test, which an unauthorized user normally should not gain access to and if the user may acquire permission of this file the security flaw is present.

5.2.3.3 IDOR-vulnerability

The test conducted in order to identify if this vulnerability exist or not is very similar to the solution of performing a test for the LFI, Local File Inclusion, -vulnerability. In this case the test is merely evolved around the reference of the web application rather than the files. The user performing the test solely has to specify a reference within the test that an unauthorized user should not have admission to, if the test is successful and the user acquire access to the specified reference the IDOR-vulnerability is present.

5.2.3.4 Subdomain takeover

When performing the test to identify whether a takeover is possible to conduct towards a subdomain the entire content will be reviewed. This is to determine if the content within the subdomain correspond to the specifics inserted within the test by the user. If the two sections conform with one another the subdomain is recipient for takeover.
5.2. Development

5.2.3.5 .git-directory disclosure

When determining whether this security flaw exist within a web application or not a test which perform a check has to be executed. This check has to determined if the file ".git/HEAD" exist within the route of the web application to enable further analysis to detect the authenticity of the file, the analysis is only feasible to perform if the file is present. If a file would not be discovered by the check the security flaw is not present, therefore further scrutiny would not be necessary. If the file would be found by the check regex will be used to determine if the file is authentic or not, if so the vulnerability is present and has to be adjusted by an administrator. The correction of a vulnerability of this extent is critical due to the possibility of an attacker being able to access and download the entire source code of the website, this is rated as one of the most crucial security flaws.

5.2.3.6 Stored XSS- & Blind Stored XSS-Injection

To investigate the possibility of these vulnerabilities being present a test has to be executed within the web application where these are suspected to exist. Within this web application a JavaScript will be inserted, if this script is not initiated the security flaw is not present. On other hand, if the script is executed correctly the vulnerability exist and has to be dealt with by the administrator.

5.2.3.7 Reflected XSS

The reflected XSS-vulnerability is discovered by a test with the same characteristics as when identifying a Stored, or Blind Stored, XSS-Injection. The test has to be performed within the web application, where the security flaw is expected to be contained, in order to be able
to identify it. When located in the area of the web application where the security flaw is suspected to exist a JavaScript will be executed. The script will however not be initiated if the security flaw is not present, the test performer will in this situation be notified about the absence of this sort of vulnerability.

5.2.4 XML-specification

In this section the XML-code developed to enable vulnerability detection of the listed security flaws will be elucidated.

5.2.5 General code

This section list all the general code which is included within all the tests. This is code which state the language in use, the paths etcetera.

Sets XML version and text encoding used.

```xml
<?xml version="1.0" encoding="UTF-8"?>
```

This is the root node of the xml file.

```xml
<webvup>
```

This define the web application which the test will be run towards and which port the server is listening on.

```xml
<site url="http://localhost/" port="80">
```

This is the first step node, the steps run in an ascending order.

```xml
<step>
```
5.2. Development

If the step would solely perform a header request, a header parameter can be set with requested a method.

```xml
<step header="GET"/>
```

If it is required for a user to be logged in to be able to execute a test the user can define on which page the login form is located. The user is also obligated to state the name of the form, username- and password field. Afterwards the user has to insert additional data to define which username and password will be in use.

```xml
<form url="login.php" name="login">
  <field name="username">admin</field>
  <field name="password">password</field>
</form>
```

This is the end of the first step node.

```xml
</step>
```

The second step node.

```xml
<step>

This is the start of the variable node which define the variables which are to be used in this step.

```xml
<variables>

This is the path which will be used in the URL. The data currently inserted will set the URL as http://localhost/fistdir/seconddir/.

```xml
<paths>
  <path>firstdir</path>
  <path>seconddir</path>
```
Chapter 5. Experiment

This define which parameters that are to be used in the URL. The present data will set the URL as http://localhost/fistdir/seconddir/?id=5&color=green.

```xml
<parameters>
    <parameter name="id">5</parameter>
    <parameter name="color">green</parameter>
</parameters>
```

This form define a form which should be specified by the user with name and method, POST/GET can also take additional data from the user. The desirable fields may be specified by the individual performing the test.

```xml
<input>
    <form name="" method="POST">
        <field name="id">input data</field>
    </form>
</input>
```

The end of the variable node.

```xml
</variables>
```

This is the expected response if the execution of the test is performed with success. This can be defined by the type TAG, ALERT or HEADER. If the tag is of the sort HTML it may be defined by the type of tag and name/id of the tag.

```xml
<expected_response type="tag" container="div" name="payload">"Cleartext or REGEX"</expected_response>
```
5.2. Development

This is the end of the second step and this close the last nodes.

```
</step>
</site>
</webvup>
```

### 5.2.5.1 SQL-injection

To perform a test which evaluate whether a SQL-Injection vulnerability is present or not a SQL-command has to be entered at the position within a web application where the vulnerability is suspected to exist.

```
<variables>
    <input>
        <form>
            <field name="id">%' or '0'='0'; SELECT @@version;--'</field>
        </form>
    </input>
</variables>
```

This is the expected outcome from the SQL-injection test which should give the version and name of the database the test was executed towards if the test is successful.

```
<expected_response type="tag"
    container="pre">MariaDB</expected_response>
```

### 5.2.5.2 IDOR-vulnerability

In this case the *page* parameter is used to fetch specified files to present them in the web browser. By traversing back three steps and entering the *etc* folder the passwd file of the server is available to be fetched.
<parameters>
  <parameter name="page">../../../etc/passwd</parameter>
</parameters>

If the string root:x:0:0 exist on the page the passwd file was sucessfully found and displayed.

<expected_response type="tag" name="content"
  container="div">root:x:0:0</expected_response>

5.2.5.3 .git-directory disclosure

To detect a .git-directory disclosure solely one header step has to be performed.

<step header="GET">

This will perform a search for the file HEAD in the folder .git.

<variables>
  <paths>
    <path>.git</path>
    <path>HEAD</path>
  </paths>
</variables>

If the returned header acquired a successful code 200, the requested HEAD file and the regular expression is found the threat is definite.

<expected_response type="header" code="200">.*([a-f0-9]{40}|ref:.*).*/</expected_response>
5.2.5.4 Stored XSS-, Blind Stored XSS-Injection & Reflected XSS

To perform a test to evaluate whether a XSS-Injection is present or not a small JavaScript is posted through a form.

```html
<form name="XSS">
  <field name="name"><script>alert ("The script has been stored and executed");</script></field>
</form>
```

If the JavaScript is executed with the outcome where the alert message shows the vulnerability exist.

```html
<expected_response type="alert">The script has been stored and executed</expected_response>
```
6. Results

When developing this tool to enable automated tests for validation of web vulnerabilities a variety of characteristics had to be taken into account. For example the standardized language should be user friendly, the parser should be able to handle different parameters and the accuracy of the tests performed has to be validated and insured. In order for the standardized language fulfilling the function of being user friendly this has been programmed with a clear and straightforward structure in order for the user to effortlessly understand and have the ability to utilize this. Solely by overlooking the language a user can locate the sections where additional information has to be inserted in an uncomplicated manner and also distinguish which sort of information. The test performed may be saved as a XML-file and therefore sent to an individual, or organization, which the security flaw concerns. The receiver can afterwards execute the file and the process of administrating the vulnerability may begin. This function makes the process very easy and no technical knowledge is demanded in order to retrieve the evidence but may instead just be passed on to the individual which will resolve the issue. This function and possibility makes the tool extremely user-friendly and easy to use and may save an organization a tremendous amount of resources. More to the point a handbook will be created to support any user feeling insecure or confused due to the lack of knowledge during the performance of the tests. The handbook will cover the entire range of the available tests which can be performed with the tool
presented in this thesis. This is of crucial importance due to the process performing these tests being possible to execute for any user possessing no certain level of technical knowledge. This handbook will be enclosed for any individual with authority to access this tool. The insurance of the parser fulfilling its functionality when managing the parameters is a crucial matter, this due to the test not being performed accurately if at all. This has been validated through a number of different tests and have an impeccable result. The tests performed have been issued using Selenium and have been successful and a more detailed view of the process is given in the Appendix B and the pseudo code of the project is given in Appendix C. Every parameter has reached the web application in order to perform the given tests and has also been rendered accurately. As mentioned earlier in this thesis it is of highest priority to guarantee the validity of the outcome and this has been conducted by performing the tests in an environment where the web vulnerabilities being present and where they are not. Depending on the web application where the tests will be conducted the demands of specified input within the standardized language will vary. For example will some web applications require the input of login information, parameters, forms and paths which will depend on which test, or tests, that will be performed and towards which web application. These aspects are something the user performing the test, or test, has to handle which may include acquiring further knowledge of the web application being investigated. For the tests which demand navigation and forms Selenium will be used but for tests solely examine whether a file or page exist requests will be used instead, for one example the .git-vulnerability is one of these situations.
7. Discussion

The technology of today might be out-dated tomorrow, this due to the
evolvement constantly taking place. This contribute to the necessity of
up-to-date security measurements to prevent security flaws from arising
along with the technological evolvement. To ensure that this does not
occur the usage of the product presented in this thesis can be used, the
maintenance of the security system is reckoned to be equally important
as the implementation. Alatalo and Vallgren agree with these thoughts
according to their thesis, which has been mentioned earlier, Anpass-
ningar till standardisering inom Software Configuration Management:
En fallstudie om standardisering inom mjukvarukonfigurationshantering
[12].

In order to execute the web vulnerability scanning selected by the user
human interaction is in need. This create a possibility for the user
to specify the test which will be executed but also demand interaction
to complete the test which may be seen as a disadvantage. As noted
by Ahlberg in his thesis Generating web applications containing XSS
and CSRF vulnerabilities the interaction of an individual is critical to
detect vulnerabilities in web applications to ensure the validity and this
is something we agree upon [14]. The code presented in this thesis is,
as stated earlier, not a general vulnerability scanner but an option to
perform specified tests within a web application. Therefore management
by a user is crucial to execute these tests since the code is dependent on
the data inserted by this individual to ensure that it has been performed accurately.

Since the interaction of an individual is in need to perform the test the handbook is crucial to maintain the goal of this process being user-friendly. The reason why we have prioritized this highly is due to the fact that web security may be badly affected by the insufficient knowledge concerning the individuals which are dependent on these test to detect and confirm the web vulnerabilities. This is a major issue which is presented in Web security and common shortfalls: the state of knowledge among developer by Strandberg and Lyckne [9]. Their thesis show devastating results in the lack of knowledge among developers which may affect the necessity of detection of vulnerabilities being available. In their thesis they have performed tests among different developers with different levels of education and work experience and some of these perform no better than 1 correct answer among 19 questions concerning web vulnerabilities. We find these results startling due to the fact that educated individuals lack the possession of the knowledge necessary concerning a variety of different web vulnerabilities, some of these individuals possess more than 20 years of work experience. Their results make it clear to us that the handbook, which we will create to enable the usage of the code presented in this thesis, will be of crucial matter. This will support any individual, with or without any technical knowledge, to execute the tests accurately.

The web vulnerability scanner produced and presented in this thesis may prevent organizations from incidents with catastrophic consequences due to lack of security. It is of common knowledge that security flaws within web applications are exploited and many individuals all around the world have some time been affected. We find that these security flaws does not only pose a threat to an organization but to individuals as well. If an organization, which possess sensitive data about individuals, is affected all the data at their disposal may be at risk which also will pose a threat
towards the entire clientele with data within their network. As stated by Easttom in *Computer Security Fundamentals* security flaws within web applications may contribute with sky high bills for an organization if these are not detected and managed [22]. This confirm the need of a vulnerability scanner and in some cases it might even be a matter of survival for an organization to have a possibility to adjust the present security flaws within their web applications.

We agree upon that the web applications are in crucial need of new tools and methods in order to sustain the demand of sufficient security measurements. There are new security flaws constantly presented to the world of technology, therefore vulnerability scanners will be an optimal tool for an organization in need of assistance concerning vulnerability detection.
8. Conclusion

Throughout this project the goal has been to create a web vulnerability scanner which enable a user to specify the test towards the web application which the individual want to detect the specified vulnerability within. The vulnerability scanner would have to be user-friendly and contain a numerous different vulnerability test which would execute in an accurate manner to ensure the validity of the results. The web vulnerability scanner presented in this thesis has achieved all the desirable characteristics and is therefore considered to be satisfying in an operational perspective. It would be beneficial if the individual, which will perform these test, would not have to interact with the tool to conduct the vulnerability scans but this is, at this moment, not possible since this would compromise the assurance of an accurate result.

The tool will make the process of reporting security flaws much less complicated due to the possibility of just sending a XML-file with the parameters set to perform the test which point towards the vulnerability. The receiver can execute the file and watch the test be performed and identifying the alleged security flaw. This eliminate the necessity of a middle-man normally having to search and find the bug, report back to the authority that it does exist and afterwards wait for instructions. With this step being eliminated the individual receiving this bug report immediately can instruct the administrators to patch this security flaw and also refer to the XML-file for exact information about the issue.
The tool will be presented to the users with a handbook which will support them in the usage of the tool. This will prevent mis-usage and therefore it will not require an immense amount of resources being spent on additional support for the users.

Since the code is compiled in a manner which enable it to be expanded additional test may be added, this is beneficial since various security flaws most certain will occur in the future. These web vulnerabilities may be crucial for an individual or organization to detect and it will be optional to add these as features.

Web vulnerabilities are often expensive to manage for organizations, many of these may not possess necessary resources to manage these. This service open up an opportunity for anyone to perform the detection themselves, this will prevent expensive investigations. Instead the organizations can focus on the management of the security flaws in order to avoid devastating exploitations.

The product developed throughout this thesis allow an organization to invest an inferior amount of resources such as time when validating whether a web vulnerability reported is present or not. With the aspects of user friendly, effective, accurate and with a coverage of multiple web vulnerabilities the tool developed has reached beyond the requirements stated earlier in this thesis. Therefore this project has fulfilled the goal of the development and may contribute tremendously in an organizations effort in securing their web environment.

8.1 Future Work

The web vulnerability scanner presented in this thesis may be evolved in the future to cover a wider range of vulnerabilities. Every day there are new security risks discovered, these threats may be included in this
scanner to enable detection depending on the demand.
Bibliography


A. Pseudo code

Figure A.1: Dependencies diagram

A.1 Main

DEFINE BOOLEAN answer AS false
DEFINE LIST stepList

CALL XMLParser WITH xmlfile INTO stepList

CREATE Test OBJECT WITH URL and Port FROM stepList
CALL Test.Connect
A.2 XMLParser

FOR all testParameters IN stepList
    IF is header step THEN
        CALL HeaderCall WITH testParameters INTO answer
    ELSE
        IF login information exist THEN
            CALL Test.LogIn WITH login information
        END IF
        CALL Test.TraverseSite WITH paths and parameters
        CALL Test.postForms WITH form information
        CALL test.checkResponse WITH expected value INTO answer
    END IF
    IF expects response AND answer EQUALS true THEN
        vulnerability exist
    END IF
END FOR

A.2 XMLParser

FUNCTION XMLParser ( xmlfile )
    CREATE testParameters OBJECT
    CREATE XMLReader OBJECT WITH xmlfile
    FOR all xml nodes in XMLReader
        INSERT current node INTO testParameter.currentNode
    END FOR
END FUNCTION

A.3 Test
FUNCTION Test ( URL and Port )
    DEFINE STRING url AS URL
    DEFINE INTEGER port AS Port

    CREATE Selenium OBJECT

FUNCTION connect ()
    CALL Selenium.Connect WITH url and port
END FUNCTION

FUNCTION logIn ( login information )
    CALL Selenium.goTo WITH login page FROM login information
    CALL Selenium.post WITH name and password FROM login information
END FUNCTION

FUNCTION TraverseSite ( paths and parameters )
    CALL Selenium.goTo WITH paths and parameters
END FUNCTION

FUNCTION postForms ( form information )
    CALL Selenium.post WITH form information FROM login information
END FUNCTION

FUNCTION checkResponse ( expected value )
    IF expected value is in a tag THEN
        RETURN true IF expected value is in tag ELSE false
    ELSE IF expected value is in a alert THEN
        IF alert is present AND alert got the expected text THEN
            RETURN true
        ELSE
            RETURN false
        END IF
    END IF
END IF
FUNCTION HeaderCall (testParameters)
    DEFINE STRING path AS paths and parameters FROM testParameters
    DEFINE RESPONSE response

    CREATE httpConnection OBJECT WITH URL, paths and port FROM testParameters

    CALL httpConnection.connect WITH headermethod FROM testParameters INTO response

    IF response.code EQUALS testParameters.expectedcode AND response.text EQUALS testParameters.expectedtext THEN
        RETURN true
    ELSE
        RETURN false
    END IF
END FUNCTION
B. Examples

B.1 SQL-Injection
Figure B.1: Inputs login-form fields names, username and password.
Figure B.2: Inputs path to the vulnerability.
Figure B.3: Input a valid SQL-injection.
Figure B.4: Input expected response that should exist if vulnerability exist.
Figure B.5: Vulnerability found.
B.2 LFI-vulnerability

Figure B.6: Inputs login-form fields names, username and password.
Figure B.7: Inputs path to the vulnerability.
Figure B.8: Input vulnerable URL parameter.
Figure B.9: Input expected response that should exist if vulnerability exist.
Figure B.10: Full XML code.
Figure B.11: The test is executed and vulnerability is found.
B.3 .git- directory disclosure

Figure B.12: Create a header step.
B.3. .git- directory disclosure

Figure B.13: Inputs path to the vulnerability.
Figure B.14: Input the expected response if the file exists.
Figure B.15: The test is executed and vulnerability is found.
B.4 Stored XSS-Injection

Figure B.16: Inputs login-form fields names, username and password.
Figure B.17: Inputs path to the vulnerability.
Figure B.18: Fill in the vulnerable form with an exploitable JavaScript.
Figure B.19: Input expected response that should exist if vulnerability exist, in this case an alert pop-up.
Figure B.20: The alert is executed and the vulnerability exist.
B.5 Reflected XSS

Figure B.21: Inputs login-form fields names, username and password
Figure B.22: Inputs path to the vulnerability
Figure B.23: Fill in the vulnerable form with an exploitable JavaScript.
Figure B.24: Input expected response that should exist if vulnerability exist, in this case an alert pop-up.
Figure B.25: The alert is executed and the vulnerability exist.
Elin Hernborg

Viktor Austli