İSTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS

ADVANCED ZERO FALSE POSITIVE AUTOMATED CROSS-SITE SCRIPTING VULNERABILITY DETECTION

Graduation Project Final Report

Cemal Türkoğlu
150140719

Department: Computer Engineering
Division: Computer Engineering

Advisor : Asst. Prof. Dr. Şerif Bahtiyar

June 2018

Statement of Authenticity

I/we hereby declare that in this study

1. all the content influenced from external references are cited clearly and in detail,
2. and all the remaining sections, especially the theoretical studies and implemented software/hardware that constitute the fundamental essence of this study is originated by my/our individual authenticity.

Istanbul, 01.06.2018

Cemal Türkoğlu

ADVANCED ZERO FALSE POSITIVE AUTOMATED CROSS-SITE SCRIPTING VULNERABILITY DETECTION

(SUMMARY)

Concept of Internet of Things, increases the number of the devices that is connected to the networks and internet day by day. With that fact security related issues is getting more important. Having every device including mobile phones, tablets, house and cars as smart devices brings huge risk that is being hacked and damaged besides its big advantages.

It is obvious that biggest attack surface in this whole connected networks are web applications. And it is usually the gate to the servers that users and attackers can reach and interact. Therefore it is highly important to make web servers and web applications secure. Making these test manually would obviously take a lot of time even though it would be the best way to examine. Because of the huge amount of these servers and applications it is impossible to do those test without help of the computers automated properties.

There are these kind of applications which can make security tests automatically, however they are either commercial applications or when it comes to open source ones, there are very few applications which is useful, successful and stable. For such applications trends are changing day by day, because of the errors, instability and not satisfying the needs of users. For the security testers a tool which is not working properly can be very time consuming. However it is also a fact that can not be ignored that some of those applications are dealing with some specific problems in the most proper or optimal way.

The project is focused on Cross-Site Scripting attacks that are one of the most important web application attacks. XSS is type of vulnerability that is still widely encountered even by biggest companies as Google or Facebook. Drowbacks of available applications are that being superficial, not being accurate, roboust or reliable. Besides this fact, mostly these applications have some strong ways to follow. Therefore the aim is combining those application’s best properties, best mechanism and algorithms and creating a new application that can deal with these problems with the best well known approach.

One of the biggest problem of automated vulnerability testing applications is misleading and wrong results, also known as false-positives. In this project testing Javascript injection with a real browser gives hundred percent correct results. The reason behind the idea is that Javascript can only be observed after page is loaded to the browser. There is also proxy layer in this project that intercepts the traffic and injects the payload. Proxy layer brings strong functionality for manipulating possible input values.Project consists of 3 indipended module for the sake of flexibility; Scanning/Crawling module, Browsing and Testing Module and Proxy module. The experiments made on test environments gives zero false postitive, in other words totally correct results and application is able to detect all targeted vulnerabilities.

GELİŞMİŞ SİTELER ARASI BETİK ÇALIŞTIRMA AÇIKLIĞININ ATOMATİK TESPİT EDİLMESİ

(ÖZET)

Nesnelerin interneti konsepti ile internete bağlı olan cihaz sayısı her geçen gün artmaktadır. Bu gerçekle beraber siber güvenliğin önemi gittikçe artmaktadır. Bütün network güvenliği önem arzetse de genellikle web uygulamarı saldırı yüzeyinin en geniş olduğu alandır. Ayrıca en çok açıklık bulunan ve saldırı alan uygulamalar yine web uygulamalarıdır. Bu sebeple web sunucularını güvenli tutmak ve mevcut açıklıkları önceden tespit edip önlem almak önemlidir.

Mevcut web uygulama sayısı göz önüne alındığında, hatta tek bir sitenin tüm haritası ve test edilmesi gereken parametreleri düşünüldüğünde bu testleri el ile yapmak pek mümkün değildir. Bu sebeple geliştirilmiş test araçlarının avantajları ve dezavantajları bulunmaktadır. Projede bu problemin çözümüne yönelik avantaj sağlayan yaklaşımlar bir araya toplanmaya çalışılırken dezavantaj veya ekstra maliyet gerektiren çözümler için alternatifler üretilmeye çalışılmıştır.

Bu çalışmanın temel amacı bir sitede mevcut olabilecek siteler arası betik çalıştırma (Cross-Site Scripting, XSS) açıklıklarını, hiç bir hatalı positif sonuç içermeden yani aslında var olmayan bir açıklığın bulunmuşçasına sonuca yansıması gibi bir durum söz konusu olmadan ve tüm XSS açıklık türlerini tespit edebilecek bir otomatik test uygulaması geliştirmektir. XSS açıkları genel web uygulama güvenliği açısından günümüzde hala en önemli ve tehlikeli açıklıklardan sayılmakta, Google veya Facebook gibi büyük şirketler bile bu açıklıktan zarar görmektedir.

Proje birbirinden bağımsız 3 farklı modül olarak gerçeklenmiştir. Bu modüllerin ilki sitenin tüm linklerini Önce Enine Tarama algoritması ile gezinerek, algoritmaya verilen derinliğe göre sitenin bir haritasını çıkarmaktadır. İkinci mödül ise Selenium sürücüsü vaıstasıyla Google Chrome tarayıcısını çalıştırıp site linkindeki parametreleri ve form girdilerini belirleyip üçüncü modül olan vekil sunucuya göndermekte ve gelen cevabı tarayı üstünde gözlemleyip Javascript komutunun çalışıp çalışmadığını inceleyerek açıklığı tespit etmektedir. Bu çözümün getirisi yanlış yönlerdirmelere sebep olmamak ve sonuç listesinde hiç bir hatalı sonuç, yada yanlış positif olmamasını sağlamaktır. Javascript komutları gelen cevabın tarayıcı tarafından yorumlanıp yüklenmesinden sonra çalıştığı için herhangi bir programlama dilinin istek/cevap mekanizmaları veya sanal tarayıcı gibi ortamlarla açıklığı tespit etmeye çalışmak hem çok zorlu hemde yanlış sonuçlar elde etmeye çok müsaittir. Bunun yerine gerçek tarayıcıda javascript komutlarının çalışıp çalışmadığını gözlemlemek çok daha güçlü bir yaklaşımdır. Bu yaklaşımın tek dezavantajı ise safyanın yüklenme süreleri ve tarayıcının işlem süresinden dolayı performansı negatif yönde etkilemesidir.

Diğer yandan başka alanlarda yapılan performans geliştirmeleri çalışma süresini kısaltmaya yetmiştir. Özellikle Go dilinin eş zamanlı çalışma performansı ve güçlü kendine has thread mekanizması ile performansa ciddi katkı sağlamıştır. Ayrıca Go dili yazımı

açısından Python gibi kolay bir sözdizimine sahip olmakla beraber düşük seviye programlama dilleri gibi derlenen bir dil olduğu için hız konusunda önemli artıları vardır. İlk iki modül dilin bu avantajlarından faydalanmak üzere Go ile geliştirilmiş olup üçüncü modül ise esneklik ve stabil olması açısından avantajlı mitmproxy ismindeki vekil sunucuyu yönetmek için Pyhton3 dili ile geliştirilmiştir.

Üçüncü mödül olan Vekil sunucu ise tarayıcı ile istemci arasındaki istekleri kesip, zaralı Javascript kodlarını isteklerin başlıklarına, sorgu değişkenlerine ve form değişkenlerine yerleştirerek sorguyu manipüle etmekte gelen cevabı ise doğrudan tarayıcıda gözlemlenmesi için yönlendirmektedir. Vekil sunucunun sağladığı katkı ise gerçek browser üzerinde veya herhangi bir programlama dili ile değiştirilmesi çok zor veya bazı durumlarda imkansız olan bazı kullanıcı girdilerini araya girip paketin içeriğini değiştirebildiği için sorunsuz bir şekilde manipüle edebilmesidir. Ayrıca bazı sayfalarda form girdileri istemci taraflı kontroller ile kısıtlanırken vekil sunucusu böyle bir engel ile karşılaşmadan değişiklik yapabilmektedir.

Yapılan test saldırıları göstermiştir ki gerçek bir tarayıcı ile yapılan test sonucunda hatalı sonuçlar sıfıra indirgenmekte ve uygulama her 3 tip XSS açıklarınıda tespit edebilmektedir. Hatalı sonuç olmamasına rağmen “Web For Pentester” test sisteminde mevcut olan iki açıklığın bulunamadığı gözlemlenmiştir. Bulunamayan bu iki açıklığın sebebi dizin testlerinin implement edilmemesi ve sayfa haritalama yaparken bütün butonlara tıklanıp yeni çıkabilecek linklerde göz önünde bulundurulmalıdır. Örnek 9 da link bulunur ancak # parametresi tespit edilemediği için oradaki açık bulunamamıştır. Bu geliştirmeler performansı da kötü yönde etkileyeceği için şuan bu çalışmanın kapsamına dahil edilmemiş olup sonraki işler listesine alınmıştır.

Contents

1 Introduction and Project Summary

Thanks to the concept of Internet of things, number of the devices that is connected to the networks and internet is getting increased day by day. With that fact security related issues is getting more important. Having every device including mobile phones, tablets, house and cars as smart devices brings huge risk that is being hacked and damaged besides its big advantages.

It is also obvious that biggest attack surface in this whole connected networks are web applications. And it is usually the gate to the servers that users and attackers can reach and interact. Therefore it is highly important to make web servers and web applications secure. Making these test manually would obviously take a lot of time even though it would be the best way to examine. Because of the huge amount of these servers and applications it is impossible to do those test without help of the computers automated properties.

There are these kind of applications which can make security and penetrations tests automatically, however they are usually commercial applications. And when it comes to open source ones, there are very few applications which is useful, successful and stable. Usually with this kind of open source applications trends are changing day by day, because of the errors, instability and not satisfying the needs of users. For the security testers a tool which is not working properly can be more time consuming than manual tests. However it is also a fact that can not be ignored that some of those applications are dealing with some specific problems in the most proper or optimal way. Therefore the aim is combining those applications best properties, best mechanism and algorithms and creating a new application that can deal with these problems with the best well known approach.

As it is mentioned above, web application security is one of the most important attack surface and to scan web application vulnerabilities there are commercial and open source vulnerability scanning tools to scan lots of common weaknesses. Even thought the idea of this project was creating a comprehensive scanning tool, it is obvious that to make an advanced tool even for only one specific vulnerability it needs way more complicated project. Some of drowbacks of these comprehensive applications are that being superficial, not being accurate, roboust or reliable. Besides this fact, mostly these applications have some strong ways to follow. Therefore all types of Cross Site Scripting (XSS) is chosen as topic to be focused.

Because of the importance and criticality of web technologies OWASP organisation is founded at 2001 and according to their 2017 Top 10 Vulnerability Report ; “XSS is the second most prevalent issue in the OWASP Top 10, and is found in around two-thirds of all applications.” [1]

XSS is another type of code injection attacks and attacker is able to run arbitrary javascript code in the browsers of users who visit the victim website. At first any Javascript code may not seem harmful, because Javascript has control over very restricted areas. It has very limited access operating system. Therefore one can thing that consequences of XSS attack can not be so harmful. However Javascript has the control over the browser and also combining it with other attacks can cause disastrous scenarios. Aim of XSS attacks is usually known as stealing cookie from user, however as it is mentioned controlling the

browser, being able to send HTTP Requests via using XMLHttpRequest, and also controlling the DOM and manipulating it is stronger aspects of Javascript.

Possible consequences of XSS can be cookie theft via accessing the document.cookie object, keylogging via running an event listener such as addEventListener, and phishing via changing the apperance of website, in other words attacker can insert a fake authentication form inside the page and it will end up with stealing user credentials. Being able to change DOM of webpage gives unlimited different possible attack scenarios.

When it comes to cyber security attacks, especially attacks targeting web applications, user input points are almost always the main source of problems. This interaction with the outside is inevitable because software is for human usage and it is impossible to isolate the software from external world.

Cross site scripting is also a vulnerabilty suffers from user inputs. It occours when a user input is included into the page without sanitizing it. As it is visible from the name of vulnerability it is related with injecting javascript command into original code. Besides HTML and CSS, javascript is one of the mostly used programming language for web technologies. It provides interactivity and dynamics for the pages. Because of new trends as Angular, React, NodeJs, javascript is getting more and more popular and it is in the first place of mostly used programming langueges in Github list as it can be seen in Figure 1.1

# Ranking	Programming Language	Percentage (Change)	Trend
1	JavaScript	22.947% (-2.606%)	
2	Python	16.127% (+0.848%)	
3	Java	9.960% (+0.001%)	
4	Go	7.236% (+1.238%)	^
5	Ruby	6.732% (+0.237%)	
6	C++	6.423% (+0.779%)	^
7	PHP	6.094% (-1.242%)	v
8	TypeScript	4.807% (+1.999%)	^
9	C#	3.375% (-0.647%)	v
10	C	2.890% (-0.442%)	v

Figure 1.1: Github Programming Languages Rank [2]

Because of this popularity and being mosty used language javascript injections, in other words XSS, is more important threat anymore. Even biggest companies have been suffering from this vulnerability. Including companies as Google, Facebook, Yahoo, Ebay, Microsoft, Github, Wordpress, Uber and many more, are announced a bug bounty program and after resolving bugs usually it is made public and possible to see where was the vulnerability. From this type of examples it is visible to see that including all this big companies most of webpages are still endengerad about XSS and new evidences are still published.

Document Object Model abbreviated as DOM is another important subject to understand the nature of this vulnerability. It is kind of data structure to store all elements of a page when the page is loaded to browser. It contains the whole structure of webpage. As it is said JS is client side scripting language to change the behavior or appearance of web page via using DOM.

Examining all types of XSS vulnerability, detecting intrusion points, testing these points and trying to exploit can be said as main goal of this project. Example projects are also examined, weak and strong parts of them will be shown later on Comparative Evaluation and Discussion part. Most proper and stronger ways of approachs for solving typical problems are collected in the project.

There are 3 types of XSS Injection types; Reflected XSS, where the malicious string arise from victim's request, Stored XSS, where malicious code is stored inside the webpage persistently and runs anytime page is visited, DOM Based XSS, where the malicious code is activated after page is loaded in client's browser and browser is rendering the DOM.

The project is implemented to find all 3 types of XSS attacks successfully. Project consist of 3 modules; traversing the web page for finding possible links, dealing with query parameters and form input variables and finally sending it to proxy server to change parameters with payloads and test if javascript is working or not.

2 Terminology

In introduction section basics of the project and fundemantals of XSS vulnerability is mentioned. It is also important to understand how XSS occours and possible scenarios for different XSS vulnerability types.

Possible stored XSS attack scenario can be seen in Figure 1.2. Attacker first fills message form with malicious javascript code. Server inserts it to database without sanitizing it and also when a user opens message box, all messages from database are retrieved and listed in html as response. Because the response is coming from a trusted webpage, everything including this malicious code is interpreted as part of this webpage. When this command is executed by victim's browser, code sends a HTTP request to attacker's server with its cookie information. So at the end attacker is able hijacke all sessions of users who visit this page.

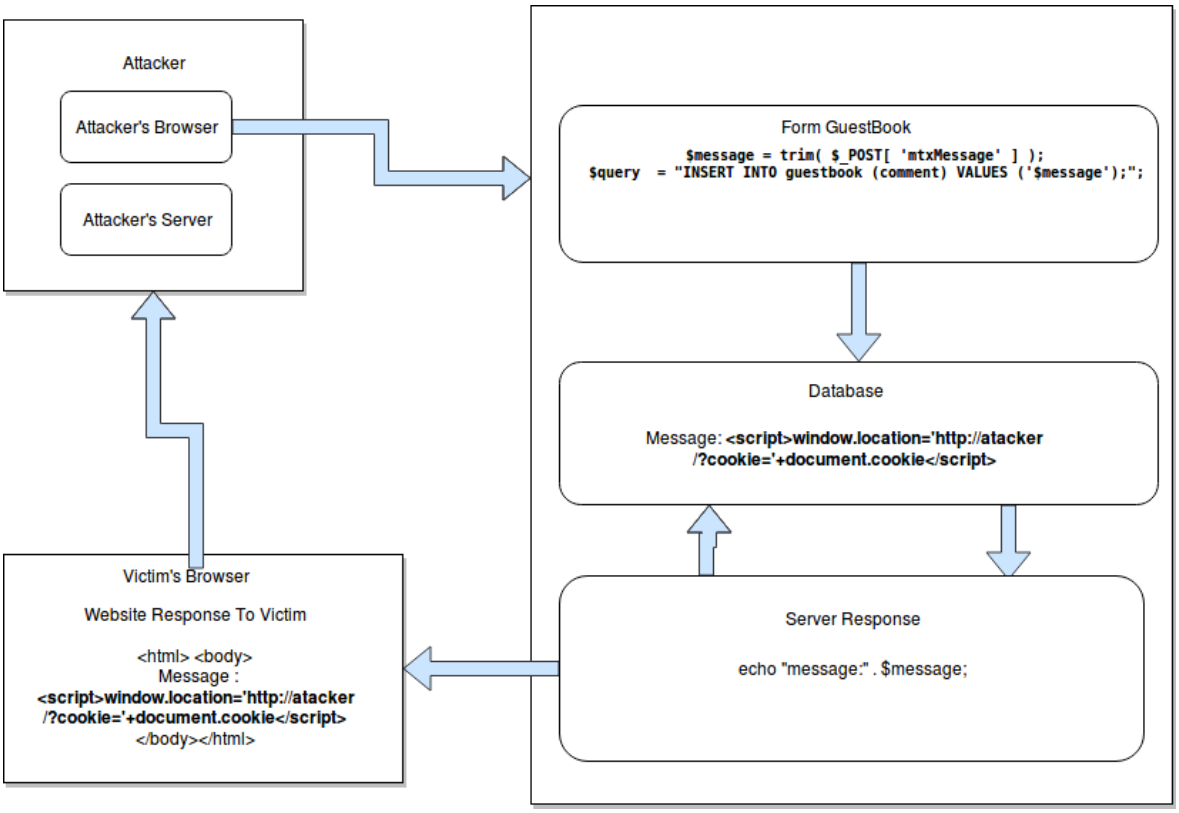

Figure 2.1: Stored XSS Attack Scenario

Example request and response can be seen in Figure 2.2 and Figure 2.3. HTTP connection is interrupted with Burp Suite Proxy and query parameter that is expected as a string to be searched is changed with the malicious Javascript payload. And in the response it is visible to see that target web page does not control user inputs and use it directly inside the HTML. Therefore when it is loaded to browser one can observe that Javascript would run and pop up the alert text.

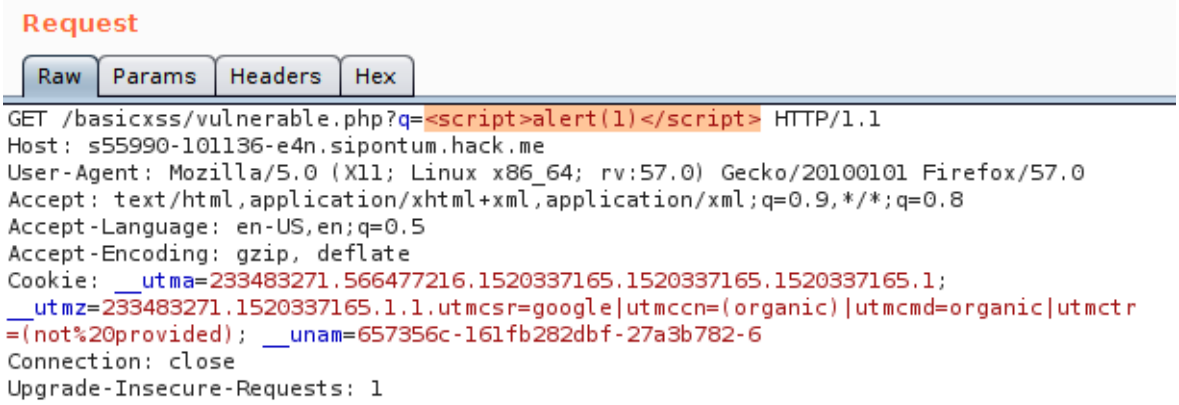

Figure 2.2: Example request tampered with proxy

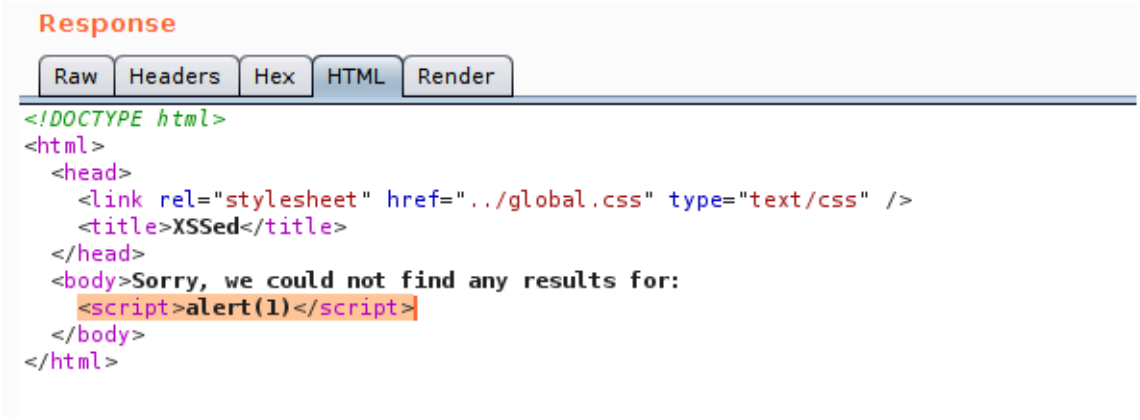

Figure 2.3: Example response

After finding security vulnerabilities, it is also important to show some statistical results and understandable risk assessment reports for the users or the administrators of the systems. This part will be more innovative and it has been seen that it is kind of new research area. Also these reports are the way that tester or security experts are expressing themselves to the clients or the companies. Therefore it is getting more and more important

2.1 Types Of XSS

2.1.1 Reflected XSS

Unvalidated and unsanitazied user input is rendered and included directly into the page. Successful attack provides attacker to run arbitrary Javascript or HTML command inside the page. To exploit this vulnerability attacker needs to make victim open the link with malicious payload. In other words attacker has to send malicious link to attacker and victim is needed to click it to be exploited. Example inaccurate input handling can be as given:

```
// Whether there is any input
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ] != NULL ) {
    // Feedback for end user
    echo '<pre>Hello ' . $_GET[ 'name' ] . '</pre>';
}
```

Attacker can make arbitrary xss run in the code as follows.

http://vulnerable?name=<script>alert(1)</script>

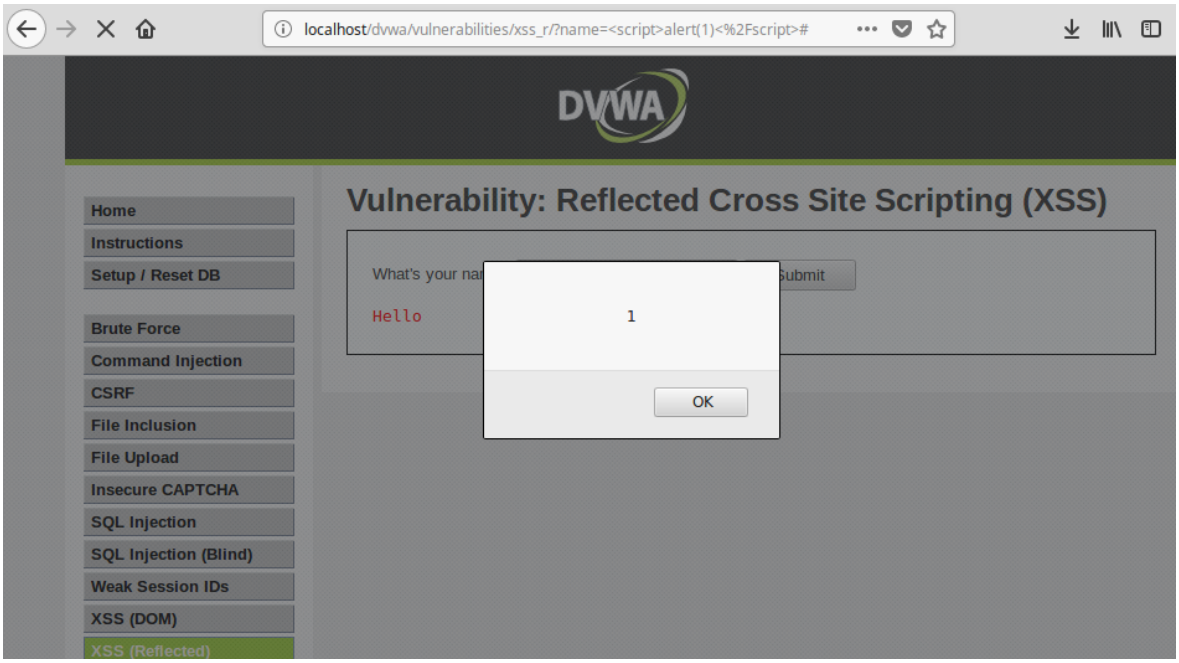

Figure 2.2: Reflected XSS Attack Scenario

2.1.2 Stored XSS

The application does not only handle the unescaped input from user but also stores it in database or other storage areas. Therefore injected script is permanently stored by the target system and anytime page is loaded and data is retrieved from storage, malicious code will be loaded to page and any client who is visiting this page will suffer from this malicious code. Stored XSS is considered as the most dangerous XSS type and consequences are critical.

Possible attack scenario is explained in Introduction section. Also you can see a message box filled with malicious payload and running that Javascript code any time page is loaded.

2.1.3 DOM Based XSS

Dom based XSS occours via changing DOM environments at client side. Contrary to Stored and Reflected XSS, DOM based XSS is not visible in the response, in other words in HTML content but it is executed after page is loaded and DOM elemnts are rendered. Therefore to experiment DOM Based XSS page must be loaded a real browser and should be observed on runtime after browser renders the DOM. Possible attack scenario will be given, in a language form as in figure 2.4.

```
…
Select your language:
<select><script>
document.write("<OPTION value=1>"+document.location.href.substring(document.location.href.indexOf("default=")+8)+"</OPTION>");
document.write("<OPTION value=2>English</OPTION>");
</script></select>
…
```

Figure 2.4: Language choice form

The language can be choosen as: **page.php?default=English**

However if a malicious user interupts the HTTP Request and change the input with a javascript code, after page is rendered it will be inserted inside the dom element. Prof of concept that attacker can reach the cookie of user is shown in Figure 2.5.

Payload: ?default=<script>alert(document.cookie)</script>

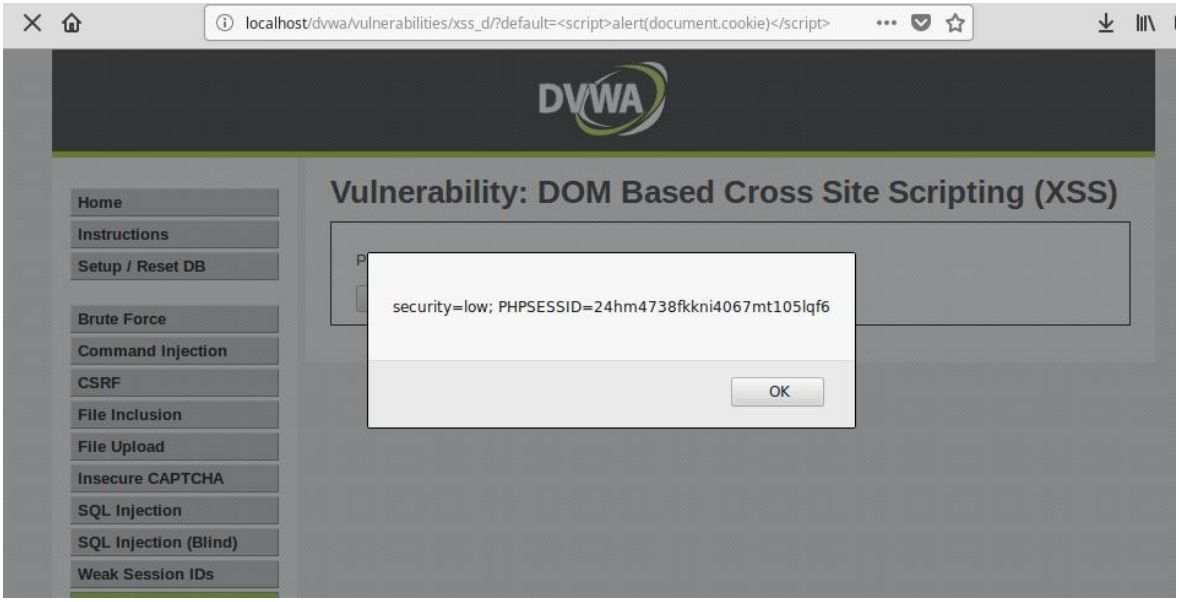

Figure 2.5: Dom Based XSS Injection Example Attack

2.2 Xss Attack Points

User inputs are not only taken by form but also there are a lot possible ways that software is interacting with the user. There are 4 mostly encountered injection point for javascript commands. To make a comrehensive test all injection points are taken into considiration in the project.

- **URLs**: https://target.com/PAYLOAD/

It is possibly dangerous when target software is using the current URL or current file of itself. For example in PHP it is possible to reach current filename with **$_SERVER['PHP_SELF']**. Taking this as a parameter and using it somewhere inside the code without escaping from special characters would cause an xss vulnerability. An example usage could be as follows.

<form action=”{$_SERVER['PHP_SELF']}”> ..</form>

In this page location is taken directly from link. An attacker can exploit it with changing the file name with and javascript or html command.

http://target.com/form.php”><script>alert(document.cookie)</script>

- **Queries**: https://target.com/page.php?name=PAYLOAD

This type of requests works with GET Requests and if the developer takes name parameter without sanitizing it and include inside the page attacker may exploit it with a malicious payload as follows.

https://target.com/page.php?name=<script>alert(document.cookie)</script>

- **HTTP Headers**: Referer, User Agent, Cookie

It is common to came across information taken from HTTP Headers and inserted to the webpage. Developer must be very careful while doing it, all information that is going to be included inside the page. Referers are very useful to go back buttons, or a lot of information is stored inside the cookies to keep track of client and show kind of history. Normally from browser it is not possible to change this information but with a proxy or HTTP request tamper tool it is easy to alter this information. Example scenario:

** Go Back **

If the attacker alter the referer header inside HTTP Request as:

Referer: <script>alert(document.cookie)</script>

2.3 XSS Injection Points

- Html attribute

If user input determines an attribute of html element, it is endangered unless it is not validated. For example choosing a color of an element:

<tagname style="*property*:{*value};">

Here the value is the place to insert the payload. An example payload for this type of location can be:

abc” onload=alert(1)>;

- Javascript Section

Because of it’s nature javascript reachs elements of webpage and use and alter them. Therefore if an element or parameter is taken by javascript such as via document.getElementsByName, and used in javascript code without sanitizing it may cause xss injection.

<script> var Name = ‘</script><svg onload=alert(1)>’;

3 Comparative Literatre Survey

The article in Cross Site Scripting Attacks Journal,2007, is one of the essential study and one of the first paper explaining the fundementals of Cross Site Scripting vulnerability in a wide range.[3]

In 2007, Jose Fonseca, Marco Vieira and Henrique Madeira take into consider these automated tools about XSS and SQL Injection Vulnerabilities, their performances and drawbacks in their article [4]. It is visible that they are also expressing the false positives as the biggest disadvantage and the time consuming part for this type of tools.

M. K. Gupta, M. C. Govil, G. Singh, and P. Sharma are published a conference paper focused on XSS basics and they proposed a context-sensitive approach based on static taint analysis and pattern matching techniques to detect and mitigate the XSS vulnerabilities in the source code of web applications in 2015.[5]

Another defensive coding approach and Cross-Site Scripting removal technique is studied by G. Shanmugasundaram, S. Ravivarman, and P. Thangavellu in 2015 and published as conference paper.[6] However this study is also looking at the vulnerability from different perspective while in this project offensive and black box tecniques are studied.

The study made by H. Shahriar and M. Zulkernine in 2011 brough a fully new approach for defensive tecniques via injecting comment to detect Javascript code injection attacks[7].

In 2017, S. K. Mahmoud, M. Alfonse, M. I. Roushdy, and A.-B. M. Salem created a comperative study that examines most widely used tecniques and compares the studies and techniques which is applied until that time.[8] It is one of the most comprhenesive paper so far created to compared advantages and disadvantages of common approachs. It gives the fundementals of the vulnerability and types of it. Also it shows many wide examples about Xss vulnerability findings on global companies and consequences of the bug. The applications developed with papers are also examined with their weaknesses and strengths.

DEXTERJS developed by Parameshwa ran et. al., is one of the most robost and successful tool so far developed. Authors explained the algorithms and tecniques they are using in the paper that they published in 2015[9]. However they decided to not open the tool to the public. Strong side of the study is that, it is totally focused on DOM Based Cross Site Scripting and their experiments shows that it is such a successful approach.

S2XS2 is also another tool developed by H. Shahriar and M. Zulkernine in 2011[10]. Authors developed an automated framework to detect XSS attacks at the server side based on the notion of boundary injection and policy generation.It can be also considered as defensive approach and the time needed to check policies are main drawback of this tool. However it is robost detecting possible intrusions.

The work of S. Gupta and B. B. Gupta in 2015 is also important to show relevant statistics and incident reports for the vulnerability.[11] They published many statistical details about the attack including organizations,incidents of global companies, categories and taxonomy of attact. It also focuses on available studies and tools.

4 Developed Approach and System Model

One of the most important challange of Xss automation tools is after sending user input to the target system, to be able to find out if the javascript is injected and working in target page or it is not allowed. The payload sent may apper in html of the page however it does not mean that it is interpreted as javascript code. It may be escaped, validated and after inserted as text to html page as well. Because of this reason it is hard to determine if the malicious code is interpreted as javascript code or standart text by the target system. Most of the tools designed for this purpose are trying to test if attack is succussful or not according to the raw http response. However javascript is not meaningful in this phase, it is only possible to detect it after seeing the result rendered by a browser kind client.

XssGo project is designed to have totally correct results. As it is mentioned also in the Introduction section, biggest problem of vulnerability scanning tools is having false positives in the result. When it comes to javascript injection, there are developed methods for testing if the page is vulnerable or not after injection, however the most reliable way is to run it on a real browser, after injecting examine if javascript alert function works properly or not.

Therefore the project is splitted into 3 indipendent module. All of the modules are using best practice methods and strongest approach of regarding applications. These modules are illastured as in Figure 4.1

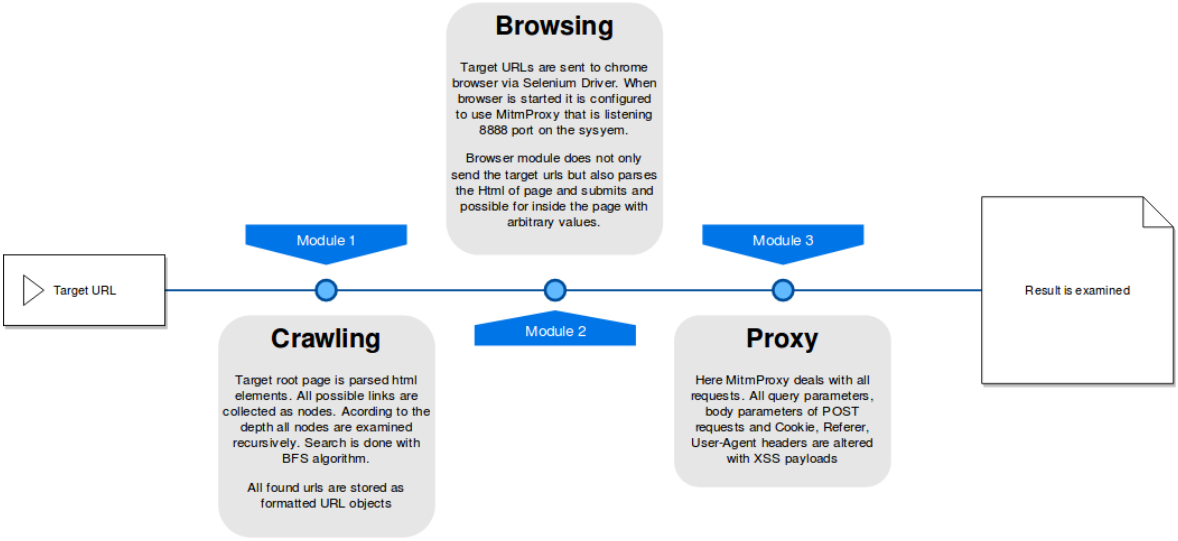

Figure 4.1: Modules of XssGo

In the next subsections details of the application is going to be discussed. Details of each module is explained in Section 3.3 Dynamic Model. Also the details of algorithms are explained in section 2.2 Structural Model.

4.1 Data Model

4.1.1 Important Data Structures

- **Empty{Payload string}**

Empty object has a payload attribute to store which payload succeded.

- **var TargetURLsORIGINAL map[string]Empty**
- **var TargetURLsALTERED map[string]Empty**

When Crawling module is working URLs are requested and examines the response. All links are included firstly to TargetURLsORIGINAL map, secondly if there is query parameter in the link it is changed with and arbitrary unique value. For instance considering following 2 urls:

http://target/?language=English
http://target/?language=French

These urls are included to TargetURLsORIGINAL dictionary but also value of parameter is changed as

http://target/?language=1

and appended to TargetURLsALTERED. Reason of this change is to not duplicate same url and query parameters. Because testing module is iterating over TargetURLsALTERED map, otherwise the test for language payload would be done (2 * number of payloads) time. It is experineced in a test case that a parameter may have many different values and if they are included as different target URLs, there would be (number of values-1) * (number of payloads) times unnecessary work, and waste of time.

- **var VulnerableURLs map[string]Empty**

This map is to store urls that is determined as vulnerable. While vulnerable URL is the key of this dictionary, payload which triggered the alert is also stored as the value of dictionary.

- **var BadUrls []string**

While traversing all possible pages, submitting all encountered forms, pressing all buttons may cause some harm on the webpage or authenticated user profile. For instance deleting profile, deleting some users or posts, logging out from a session. For such sensitive pages user may include this pages to badURLs array, which can be delivered as command line parameter or a file. Another reason of using such list is about performance issues. Loading some pages or files has no purpose for the test and also decreasing the speed of running time, for instance pdf, doc, jpg, mp4 and some similar files. Therefore altought some of these file types are by default already in this BadURLs list, it can be be appended by user as well. To sum up during test such URLs that is in this list are skipped and not tested.

- **var Jar *cookiejar.Jar**

As it is known, HTTP is stateless protocol and it is needed to send some authentication information in every request after getting authenticated once. In XssGo project it is possible to log in with user credentials, or directly importing cookies of an authenticated user from browser. Therefore after authentication cookie jar is created and used for all requerst during runtime. Cookie jar mechanism allows program to change/update cookies according to responses anytime it is delivered.

- **var Payloads []string**

Payloads are the malicious codes to be used for injection. During test payload is needed to be proper acording to place that code is included. Therefore there is a text file including 50 different type of strong payloads. It is possible to extend the list. When program first starts these payloads are stored in this array to use later in the test one by one.

4.2 Structural Model

After the presenting entities the key algorithms, and developed approach is covered in this section. As it is mentioned earlier the project consist of 3 main modules. Modules are indipendent from each other. Basics of modules are mentioned in the beginning of section 4. Also it is possible to see some details of modules in Figure 4.1

Program take input parameters from user as comman line flags and makes necessary initialization according to these parameters and necessary functions are determined also with this parameters. These user inputs can be listed as follows:

Options:

--Help
Display this page

--BlackList
Forbid some links to be visited via giving comma seperated list

--CookieFile
Authenticate via cookies. Give path of cookie file as json

--Headless
Browser can to run as headless

--Level int
Scan Depth Level (default 3)

--LoginPage string
Authenticate via login page. (default empty)

--PayloadFile string
Set Payloads from file (default "payload.txt")

--URL string
Target Url to be scanned (default "http://localhost/dvwa/")

4.2.1 Crawling/Scanning Module

First 2 modules of the project are implemented by Go language, also known as Golang, and the proxy module is developed by python version 3. Go language is relatively new language developed by Google company. Even though Go is very young and not widely used, it caught attention by many developers and companies with its advantages. The idea of the syntax is easy to read and write as normal English text. Therefore it is similar to Python language from that perspective. On the other hand, the drawback of Python like high level languages is being intrepreted language and it causes performance reductions. Considering that perspective Go language is also compiled language as C or C++. Therefore it is easy to read/write but also fast when it is running. Another point is compiled code can be distributed among different systems and ready to run.

Biggest advantage of Go language is seen as having go routines as different form operating system threads or processes. Having this special threads and good orginaziton of the language about concurrency is one of the mostly admired side of the language.

There are many studies about performance of Go language. Mutex and concurrency mechanism of the language is main reason of its good performance. It is possible to see popularity of the language in Github statistics. Also when it comes to performance there is an example work showing the performance of the language. It is proved in the study that Go is way better than Java language about performance.[12]

Therefore it was the one of the most effecient way to implement such iterative process with Go language. It is implemented in a way that program runs different threads as many as number of the CPU cores of the device that it is running. Therefore if program runs on computer with 4 CPU cores, there will be 4 different concurrent threads running and dividing the whole work to 4.

Another important issue about sending HTTP Requests are that this communication works asynchronously. Therefore one need to be sure that the response is fully got by requester before sending another request. Interrupting one operation and starting second one in the middle causes wrong answers or loosing the data. Therefore it was also proper choice to implement it by Go language to solve this problem with Go's advanced Mutex lock mechanism and Go channels to synchronization.

Crawling module starts taking first path to scan, sends a get request to URL and parses HTML response. After having HTML elements, all a tags which are used to create and anchor link are examined and if the URL belongs to same domain it is included to target URLS list. To crawl over the whole web page and create a site map Breadth First Search Algorithm is used as a base. Depth is taken from user, and it is important parameter about execution time. It is not because BFS algorithm would run longer, this module is the fastes module in the entire project, but if the number of targets increases, the process time of 2. Module, browsing module is also increases.

There is native HTTP package and NET package of Go language to deal with connections and HTTP Requests. Surf library is also designed upon Go native libraries to make such operations in more user friendly way. Some of these operations can be listed as browsing pages with requests, cookie management, having a session to store history, dealing with HTTP headers such as User-Agent, submitting forms, creating DOM structure of page, anaylzing HTML elements so on so forth.

The pseude code of the algorithm of the module is as follows:

```
Initialize all flag parameters including URL and Level taken from user
CrawlURL(url)
If  Level > 1{
        For each url in TargetURLs
                CrawlURL(url)
        Done
}
```

```
CrawlURL(url string) {
        Response := Get(url)
        HTML := ParseHTML(Response)
        Links := HTML.links()
        For each link in links array {
                alteredUrl := handle_query_parameters(link)
                ifContains := TargetURLsAltered[alteredUrl]
                whitelisted := true
                if url contains any element from BadURLs
                        whitelisted := false
                if Target.Host == link.Host && !ifContains && whitelisted {
                                TargetURLsOrignal[link] = Empty{}
                                 TargetURLsAltered[alteredUrl] = Empty{}
                }
        }
}
```

The reason of having 2 different list for target links is as explained in Section 4.1.1 that to get rid of repeating the test for same page having different values in a query parameter. In other words, if the crawler came across with an almost same url but only having different value for query parameter, it causes multiple unnecessary repeats. To avoid it and increase the performance all query parameters are given a single format and stored in altered data structure.

An example result can be seen in figure x.x . Scanning is done over the facebook home page and any authentication mechanism is not chosen, therefore scanner module is traversing the page as anonymous user.

```
1 https://www.facebook.com/
2 https://www.facebook.com/recover/initiate?lwv=110
3 https://www.facebook.com/policies/cookies/
4 https://www.facebook.com/about/privacy
5 https://www.facebook.com/legal/terms
6 https://www.facebook.com/pages/create/?ref_type=registration_form
7 https://www.facebook.com/r.php
8 https://www.facebook.com/login/
9 https://www.facebook.com/lite/
10 https://www.facebook.com/mobile/?ref=pf
11 https://www.facebook.com/find-friends?ref=pf
12 https://www.facebook.com/directory/people/
13 https://www.facebook.com/directory/pages/
14 https://www.facebook.com/places/
15 https://www.facebook.com/games/
16 https://www.facebook.com/directory/places/
17 https://www.facebook.com/directory/celebrities/
18 https://www.facebook.com/directory/marketplace/
19 https://www.facebook.com/directory/groups/
20 https://www.facebook.com/recipes/
21 https://www.facebook.com/sport/
22 https://www.facebook.com/look/directory/
23 https://www.facebook.com/local/lists/245019872666104/
24 https://www.facebook.com/facebook
25 https://www.facebook.com/ad_campaign/landing.php?placement=pflo&campaign_id=402047449186&extra_1=auto
26 https://www.facebook.com/pages/create/?ref_type=sitefooter
27 https://www.facebook.com/careers/?ref=pf
28 https://www.facebook.com/privacy/explanation
29 https://www.facebook.com/help/568137493302217
30 https://www.facebook.com/policies?ref=pf
31 https://www.facebook.com/help/?ref=pf
```

Figure 4.2: Part of the result of crawling Facebook

4.2.2 Authentication Mechanism

Nowadays most of the web pages are dynamic and interacting with users. Therefore functionality of page needs sessions and authentication by each user. In the XssGo it is possible to scan the web page without authenticating if login page or cookie flag is empty. However if it is already known that most of the functionality of target page is after logging in, program need be granted as a regular user to have an access to these parts. Therefore program offers to user 2 different authentication way.

First way to be authenticated is supplying a login page as a parameter as follows.

```
./XssGo --LoginPage http://target.com/login.php
```

By this way program opens this page and after parsing its html elements it tries to find possible login form. Keywords as login, username, password, mail has priority for it, but in case of absense of such html tags with attribute, every form found is iterated and all input parameters are asked to user to be filled. After taking the credentials form values are filled and the form is submitted. After submitting the cookie sent by target server as response is attached to cookie jar structure. Cookie jar structure is used to keep track of any update on cookies for any current request.

Second way to have a session is delivering a cookie file. User can supply the cookies as file taken from browser that user already logged in and having a session. The way to supply a cookie file is with the following flag:

```
./XssGo --CookieFile /path/cookies.json
```

Cookie file is expected in json format for logical reasons. Nowadays it is easy to extract cookies from browser, by directly options of the browser or it is also possible and easy to use some browser add-ons. Mostly browsers or such tools extract cookies in json format. Therefore file is taken as json object, parsed and related cookie values are attached to cookie jar.

Therefore user can choose to scan whether with authentication or not, and authentication can be done by delivering login page and user credentials, or also delivering a cookie file taken from a session that already exist.

4.2.3 Browsing and Testing Module

The core of browsing module consist of selenium web driver. Here is the solution for false-positives comes. Testing XSS vulnerability by virtual browsers or HTTP request/response mechanism of programming language gives a huge possibility of having wrong results. By these techniuqes it is so hard ot catch wheter malicious code is included in page or not without being sanitazed. Therefore it is the most accurate way to examine the results of testing that using a real browser and observing if Javascript is really run after page is loaded to the browser.

To interact with a real browser Selenium web driver is one of the most advanced and common application. Therefore main program runs the google chrome browser after crawling phase is done. Google chrome is invoked with proxy parameters to let the

browser send request to the middle man.Google chrome might be run optionally as headless or with user interface. Headless browser is a term refering that a real browser is running in the background, all functionality of Selenium driver is similar but user interface is not shown to the user. Using it with the interface is the recommended way because of possible javascript problems of headless browser technique.

After crawling module targetURLs are already filled in TargetURLsAltered list. The module first loads the page and check if there is any alert pop up alread to catch stored XSS vulnerabilities. However one problem with this step is that if the page has some built in pop up alerts it may mislead the program. To solve it one approach is checking the alert text and if alert text contains any kind of payload texts then it is because of XSS injection, otherwise it is intentional notification mechanism of the webpage.

All links in the target url array is tested from the perspective of query parameters and forms. Actual replacement is done in proxy module but this module makes necessary manipulations and prepare the request for proxy module.

Another functionality of this module is setting payloads. As it is mentioned payloads are replaced with input values in proxy module but the application written in Go, and the proxy script implemented in python has to interract with each other and share the payload text. Therefore this module reads the payload file and sends set payload command to the proxy server for each test. Proxy server replaces values with this payloads sent by Go program. The algorithm of the module is as follows:

```
Initialize google chrome and run the process with setting proxy server
// ./google-chrome --proxy-server=http://localhost:8888

for u := range TargetURLsALTERED {
        SendToProxy(u)
}

SendToProxy(url string){
        if there is query parameter:
                for each payload:
                        send query set command to proxy server
                        send the request to url
                        // proxy server will handle the request
                        // and change the necessary areas

                        wait page to be loaded
                        if there is alert pop up
                                // xss found, add url with related payload
                                VulnerableURLs[u] = Empty{Payload:payload}

        request the url and parse html
        if there is any form tag:
                for each form:
                        fill form inputs with arbitrary value
                        submit the form
                        // proxy server will handle the request
```

```
                    // and change values with payload

                    wait page to be loaded
                    if there is alert pop up
                        // xss found, add url with related payload
                        VulnerableURLs[u] = Empty{Payload:payload}
}
```

4.2.4 Proxy Module

Adding the proxy module was inevitable to increase the functionality and access maximum number of attack vector. While observing a webpage on browser it is not possible to change some user inputs. These inputs can be hidden input tags, http headers so on so forth. Another important thing is that for selection tags or radio buttons it is not possible to send arbitrary value for the input parameter. Also another problem one might face that some forms are developed in a way that user has restricted options because of client side controls. Characher limitation for an input area, or expecting and integer value for a year inpur can be such examples. Therefore not forcing the browser to inject payloads but interrupting the requests on fly and injecting payload to all necessary areas are way better solution and it increases the functionality of the test tool. After all it is a fact that it is impossible to change some input points via browser.
Proxy module is written in python3 because of good library support and its relatively better performance than other alternatives. The term called as man in the middle refers interrupting a connection having the control over the communication. Here in this module mitmproxy proxy tool that is written in python3 is used. The proxy can be imported directly to python project or dumb executable of the project can be run with supplying parameters as port number or specific script determines the rules of proxy.

In the project delivering a script file to executable of mitmproxy is chosen. The way to run executable with necessary parameters is as follows:

```
./mitmdump –p 8888 –s script.py
```

The script.py file contains the main functionality of this module. The script is used, as it is said, to determine the rules of proxy. Main idea of this module is intercepting connections made by browser, tampering the query parameters, some HTTP headers and the body part if it exist, in other words if it is a POST request and the module 2 is submitting a form.

The proxy is able to support HTTPS connections via decrypting and encrypting the packets on fly. But it comes with a requirement for the client that needs to have self signed or built in SSL certificate. This can be done by installing the CA certificate to the client browser. The proxy has an easy way to do it. Visiting the page mitm.it webpage with proxy gives the following result.[13]

Figure 4.3: Installing SSL Certificate

Downloading and adding the pem file to chrome browser authorised certificates is enough to have this operation done.

The script file, which is the body part of proxy module, has 3 functionalities. First one is not changing anything but just forwarding the packets. It is needed because first time when page is loaded nothing on the page should be changed to give correct HTTP response to browsing module. Second function of it is to set payload. Script starts with a default payload but payload file is readed by Go program and when a parameter is tested all payloads are tried one by one. Therefore browsing module should notify the proxy script for which payload needed to be tested. Final and core part of the script is changing query parameters, POST data parameters and HTTP headers that are Referer and User-Agent. The pseudo code of the algorithm is as follows.

```
Payload := <script>alert(1);</script>
Listen_HTTP_Requests(req){
        if req.url == "localhost/payload$":
                payload := url.split('$')
                return

        if req.url == "$FORWARD"
                return

        # set headers
        req.headers["Referer"] := payload
        req.headers["User-Agent"] := payload

        # set form parameters
        form := req.urlencoded_form
                for k,v in form.items():
                        req.urlencoded_form[k] := payload

        # set query parameters
        parsed := urlparse(req.url)
        query := parsed.query()
        for k,v in query.items():
                        req.query[k] := payload
}
```

5 Experimentation Environment and Experiment Design

For testing this project it is not possible to attack live systems because of not having consent of the owner of system. Therefore during the development of the project not any live system is tested because it would be illegal and problematic from law perspective. It is a considerable idea to get permission to attack and test some of ITU web pages, however penetration testers are signing official contracts to make such tests. Therefore this bureaucracy was an obstacle for actualizing this idea. However 2 test systems that are developed for educational purposes are installed to local system and tested to experiment the results.

First test system is DVWA webpage that contains some vulnerabilities to be discovered with different difficulty levels.[14] There are low,medium,and impossible levels for XSS challenges. For XSS injection attacks there exists 3 challenge covering 3 different types of XSS; stored, reflected and dom based xss . With depth 3 all possible pages were traversed by the crawler module. For testing a payload file contains 50 strong payload is collected from different sources. Payloads are to implement different cases such as different injection points or alternative versions to bypass from filter evasion. For both low and medium levels the tool was successful to discover 3 types of XSS injections.Final result of the test over this system for security level medium is as follows:

total number of pages found after traversal . 19

form inputs tested in:
http://localhost/dvwa/security.php
http://localhost/dvwa/vulnerabilities/weak_id/
http://localhost/dvwa/vulnerabilities/sqli_blind/

vulnerable urls:

1. http://localhost/dvwa/vulnerabilities/xss_r?name=test
   Payload: <body onload=alert(“XSS”)>

2. http://localhost/dvwa/vulnerabilities/xss_s/
   Payload: <img src=x onError=alert(‘xss’)>

3. http://localhost/dvwa/vulnerabilities/xss_d/?default=English
   Payload: <script>alert(1)</script

   Time Elapsed: 1m17.955466174s

It is visible that application succeed finding 3 possible XSS vulnerabily in the target page. When security level is low the most basic payloads are even enough to inject javascript inside the page successfully, but for medium level simple payloads are not enough and more advanced one and location specific payloads are needed. Therefore my payload list is comprehensive enough to catch all this 3 vulnerabilities in also medium level.

Second vulnerable application that is used in tests are virtual machine called “Web For Pentester” which is intalled to local system. [15] This test platform is also developed for

educational purposes by PentesterLab team. There are 9 different XSS challenges and XssGo is able to find 9 of them. Tricky part with this test system is that in any next example it is needed to use a better payload. Or in other words, for each example payload has to be changed to fit the place that is going to be placed.. The result of test on this system as follows:

total number of pages found after traversal. 28

form inputs tested in:
http://192.168.56.101/upload/example2.php

vulnerable URLs:

1. http://192.168.56.101/xss/example2.php?name=1
   payload: <BODY ONLOAD=alert('XSS')>

2. http://192.168.56.101/fileincl/example1.php?page=1
   payload: <script>alert('XSS');</script>

3. http://192.168.56.101/xss/example5.php?name=1
   payload: <script>alert('XSS');</script>

4. http://192.168.56.101/xss/example7.php?name=1
   payload: ';alert(1);//

5. http://192.168.56.101/xss/example3.php?name=1
   payload: <script>alert('XSS');</script>

6. http://192.168.56.101/xss/example6.php?name=1
   payload: </script><script>alert('XSS');</script>

7. http://192.168.56.101/xss/example4.php?name=1
   payload: <BODY ONLOAD=alert('XSS')>

8. http://192.168.56.101/fileincl/example2.php?page=1
   payload: <script>alert('XSS');</script>

9. http://192.168.56.101/sqli/example3.php?name=1
   payload: <script>alert('XSS');</script>

10. http://192.168.56.101/xml/example1.php?xml=1
    payload: <BODY ONLOAD=alert('XSS')>

11. http://192.168.56.101/xss/example1.php?name=1
    payload: <script>alert('XSS');</script>

Time Elapsed: 2m47.808894758s

From test result it is visible that not only xss challenges but application found also other pages that are vulnerable to XSS. It was not probably the aim of the developer or test system but couple more pages suffer from XSS.

Example number 9 could not be found by the application. The reason not to find this example, is that while traversing if the url is not encontered as in the form that having # sign already written in link. So it is seen that it is not totally capable of finding all possible URLs.

Example 7 could not be succeeded as welli The reason of it is not testing directory names to inject javascript. It is relatively rare case and testing all directory names in a path would increase the running time considerable amount. Therefore it is not impelemented in the project. However finding an optimum solution for such cases is noted as future work..

6 Conclusion and Future Work

To sum up, the project is scanning target web pages for all types of XSS attacks. Biggest advantages of the project is having no wrong result, also known as false positive, fast running time with concurrent threads and giving flexibility to user. First phase of project scans the entire web page according to user supplied depth level by using Breath First Search Algoritm. Second phase sets payloads and tests all query parameter and form parameters. In the third face all possible input vectors are replaced with payload sent from second phase. While second phase is browsing the page with real browser via using Selenium driver and also observing the result and deciding whether it is vulnerable or not, at third phase there is proxy server that replaces all possible input vectors. Tests results are promising with not having any false-positive or true-negative. Future work or possible development steps could be listed:

- When there is query parameter,all payloads are tested to be injected with the loop. However if there is not vulnerability this approach makes running time slower. To increase the performance it is a better idea to first try with a locator. Locator can be chosen as **';!--"<XSS>=&{()}**. First attempt before starting to try all payloads this payload can be used to check how special characters are handled by web application. As a result of that more advanced and specialized payloads can be generated. Considering extreme cases this approach can even be supported with machine learning algortihms.
- When there is already a pop up by default inside the web application, it is ambiguous to handle it whether it is a stored xss or it is logical part of the application and not a vulnerability. In real scenarios XSS attacks especially discovered on big company's applications such as Google,Facebook, are usually more complicated. It is possible to inject the payload in one page but result can be seen in another page. Therefore to solve this ambiguity the text inside the payload can be another locator to detect if alert text is this locater or another text written by application developer.
- Another possible input vector of XSS is directory names in path. One can include the path name of a link. For example the developer can use the name of path in the link target.com/main/index.php. Here if developer is taking the name of directory which is main from current url and including it to HTML without sanitazing it, it also causes XSS injection attack. Testing all directory names would be costy and it is relativelty a rare case compared to other input points. Therefore to not decrease the performance it is not covered in this project however finding an approach for the balance between performance and increasing number of possible inputs could be another possible future work.

7 References

[1] Owasp,“ OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks”
https://docs.mitmproxy.org/stable/concepts-certificates/

[2] Github, “Github Programming Languages Rankings”
https://madnight.github.io/githut/#/pull_requests/2018/1

[3] J. Grossman, R. Hansen, P. Petkov, A. Rager, and S. Fogie, “Cross-site Scripting Fundamentals,” Cross Site Scripting Attacks, pp. 1–13, 2007.

[4] Hobbs, M. (2007). 13th Pacific Rim International Symposium on Dependable Computing: PRDC 2007: proceedings: 17-19 December, 2007. Los Alamitos, CA: IEEE Computer Society Press.

[5] M. K. Gupta, M. C. Govil, G. Singh, and P. Sharma, “XSSDM: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications,” 2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2015.

[6] G. Shanmugasundaram, S. Ravivarman, and P. Thangavellu, “A study on removal techniques of Cross-Site Scripting from web applications,” 2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC), 2015.

[7] H. Shahriar and M. Zulkernine, “Injecting Comments to Detect JavaScript Code Injection Attacks,” 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops, 2011.

[8] S. K. Mahmoud, M. Alfonse, M. I. Roushdy, and A.-B. M. Salem, “A comparative analysis of Cross Site Scripting (XSS) detecting and defensive techniques,” 2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS), 2017.

[9] Inian Parameshwaran, Enrico Budianto and Shweta Shinde, “DEXTERJS: robust testing platform for DOM-based XSS vulnerabilities”, 10th Joint Meeting on Foundations of Software Engineering(August 30-September 4), pp. 946-949, Bergamo, Italy, 2015.

[10] H. Shahriar and M. Zulkernine, “S2XS2: A Server Side Approach to Automatically Detect XSS Attacks,” 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, 2011.

[11] S. Gupta and B. B. Gupta, “Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art,” International Journal of System Assurance Engineering and Management, vol. 8, no. S1, pp. 512–530, 2015.

[12] Togashi, N. and Klyuev, V. (2014). Concurrency in Go and Java: Performance analysis. In: Information Science and Technology (ICIST), 4th IEEE International Conference. Shenzhen: IEEE, p.1.

[13] Mitmproxy,“About Certificates”
https://docs.mitmproxy.org/stable/concepts-certificates/

[14] DVWA (Damd Vulnerable Web Application)
http://www.dvwa.co.uk/

[15] Web For Pentester
https://pentesterlab.com/exercises/web_for_pentester