SOFTWARE TRAINING AND DEVELOPMENT CENTRE

C-DAC, Thiruvananthapuram

A PROJECT REPORT ON

"Enhancing Security Posture: Altoro Mutual & Mutillidae II Penetration Testing"

SUBMITTED TOWARDS THE

PG-DCSF SEPTEMBER 2023

BY
GROUP: 12

AKSHAY PRADIP DESHMUKH	230960940003
BHANGALE GUNJAN RAJU	230960940009
NAJMA P I	230960940030
PRATHAM GHOSH	230960940040
SHENDE ANKIT NARENDRA	230960940048

Under The Guidance Of

Mr. Jayaram P.   **Dr. Sreedeep A L**

Centre Co-ordinator   **Project Guide**

TABLE OF CONTENTS

ABSTRACT

This project report presents the results of security testing conducted on two web applications, Altoro Mutual and Mutillidae II, to assess their vulnerability to OWASP Top 10 threats. Penetration testing methodologies were applied systematically, employing various tools and techniques to emulate real-world attack scenarios. The focus was on identifying vulnerabilities such as Injection, Broken Authentication, Cross-Site Scripting (XSS), and others, which pose significant risks to the security of the applications and their data. The findings highlight specific vulnerabilities discovered in each application, their potential impact, and recommendations for mitigation. By addressing these vulnerabilities, organizations using Altoro Mutual and Mutillidae II can strengthen their security posture and reduce the likelihood of exploitation by malicious actors.

Keywords: Security Auditing, Web Application Security, Vulnerabilities, DNS Reconnaissance, OWASP Top 10.

<u>INTRODUCTION</u>

In today's interconnected world, where web applications serve as the cornerstone of modern business operations and personal interactions, the stakes for cybersecurity have never been higher. As organizations digitize their processes and embrace the convenience of online platforms, they also expose themselves to a myriad of potential threats, ranging from data breaches to malicious attacks.

Against this backdrop, this project endeavors to shed light on the critical importance of conducting penetration testing on web applications, using the widely recognized OWASP Top 10 vulnerabilities as a benchmark for assessment. By focusing on two prominent web applications, Altoro Mutual and Mutillidae II, the project seeks to provide a comprehensive analysis of their security posture, uncovering potential vulnerabilities that could compromise their integrity and confidentiality.

Through meticulous testing methodologies, including reconnaissance, vulnerability scanning, and exploitation, the project aims to simulate real-world attack scenarios and evaluate the resilience of these web applications against common threats. The findings of this assessment will not only serve to highlight areas of concern but also provide actionable insights and recommendations for strengthening their security defenses.

Moreover, by emphasizing the proactive nature of penetration testing and its role in mitigating risks before they escalate, this project underscores the importance of fostering a culture of cybersecurity awareness and diligence within organizations. Ultimately, by equipping stakeholders with the knowledge and tools to safeguard their web applications, this project aims to contribute to a safer and more resilient digital ecosystem for businesses and individuals alike.

LITERATURE SURVEY

- **OWASP Top 10 (2021)**

  - We rely on the official OWASP Top 10 documentation to understand the most prevalent web application security risks. This comprehensive resource provides us with insights into common vulnerabilities such as Injection, Broken Authentication, and Cross-Site Scripting (XSS), shaping our understanding of the security landscape and guiding our approach to the project.

- **OWASP Testing Guide (Version 4.0.1)**

  - The OWASP Testing Guide serves as a foundational reference for our project, offering guidance on effective security testing methodologies and techniques. By consulting this authoritative source, we gain valuable insights into penetration testing approaches, vulnerability identification, and exploitation, empowering us to conduct thorough assessments of web applications for our group project.

These resources, endorsed by the OWASP community, provide us with a robust framework for understanding and addressing web application security risks. They serve as authoritative references, guiding our efforts to evaluate and mitigate vulnerabilities in Altoro Mutual and Mutillidae II as part of our group project.

SCOPE AND OBJECTIVES

Our project entails a comprehensive evaluation of the web applications "Altoro Mutual" and "Mutillidae II," focusing on assessing their digital infrastructure, applications, and data protection mechanisms. We aim to identify vulnerabilities, weaknesses, and potential threats that could compromise the confidentiality, integrity, and availability of these applications' resources. The security audit will cover both technical and operational aspects, including software assessment, network architecture, user access controls, and adherence to relevant security standards and best practices. Our assessment will leverage a combination of manual and automated approaches, utilizing the latest legitimate tools available in the field. Additionally, the evaluation will extend to user authentication mechanisms, encryption practices, and incident response procedures, given the online nature of the project.

The primary objective of our project is to conduct a thorough assessment of the web applications "Altoro Mutual" and "Mutillidae II" to identify potential security vulnerabilities, such as SQL injection, cross-site scripting (XSS), file upload vulnerabilities, and weaknesses in password policies. Through this assessment, we aim to provide actionable recommendations and best practices to address identified vulnerabilities and enhance the overall security posture of the applications. By implementing these recommendations, we seek to bolster the reputation of the applications, enhance user trust, and safeguard user data. Ultimately, our goal is to contribute to a safer online environment by strengthening the security of these web applications and mitigating potential risks to their users.

METHODOLOGY

Static application security testing (SAST) is a white box security testing technique where the tester approaches the basic source code. In SAST, the application is tested for vulnerabilities from the backend to the frontend.

SAST doesn't require a deployed application. It analyzes the sources code or binary without executing the application.

In any case, since SAST apparatuses filter static code, it can't discover run-time weaknesses.

Dynamic application security testing (DAST) is an application security arrangement in which the analyzer has no information on the source code of the application or the advancements or structures the application is based on. In DAST, the application is tested by running the application and interfacing with the application.

It empowers the security expert to recognize security weaknesses in the application in a run-time condition i.e. once the application has been sent.

Dynamic testing recognizes potential weaknesses incorporating in the frontend interface of web applications.

DAST gives insights into web applications once they are sent and running, empowering an organization to address potential security weaknesses before an attacker abuses them to dispatch a cyberattack.

Testing Environment:

1. Altoro Mutual [demo.testfire.net]

2. Mutillidae II [Self Hosted]

Tools Used:

1. Burp Suite
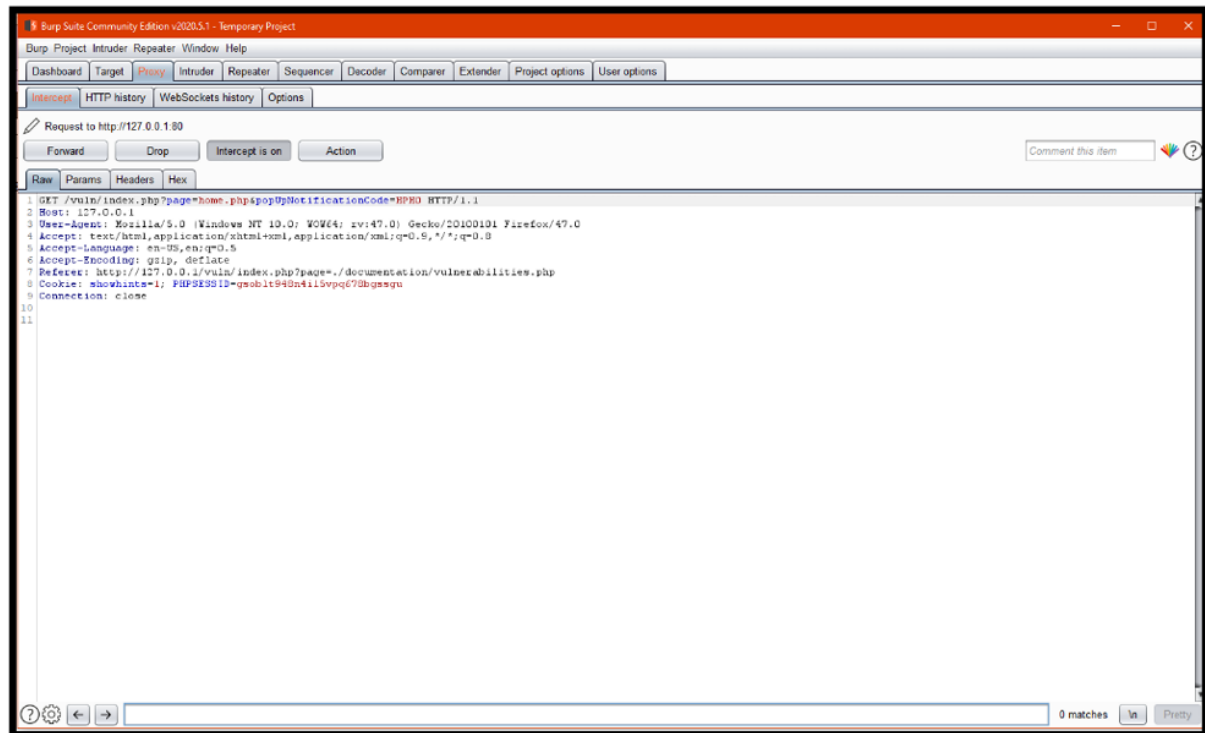
2. SQL Map
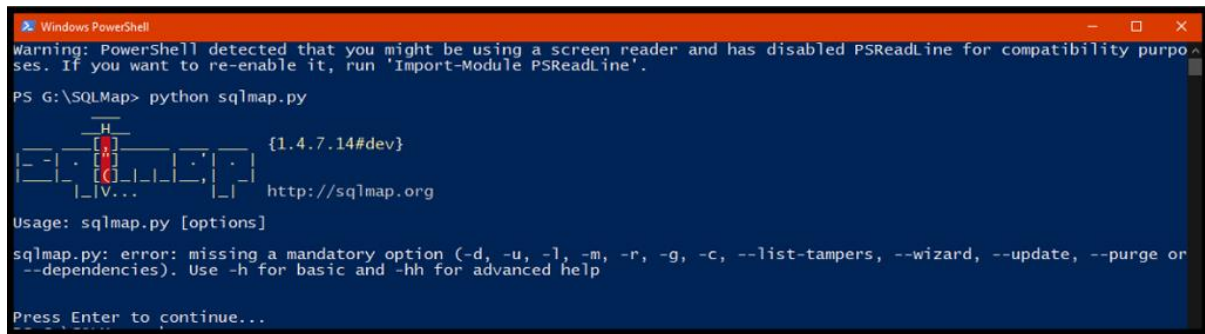
3. OWASP Zap Vulnerability Scanner
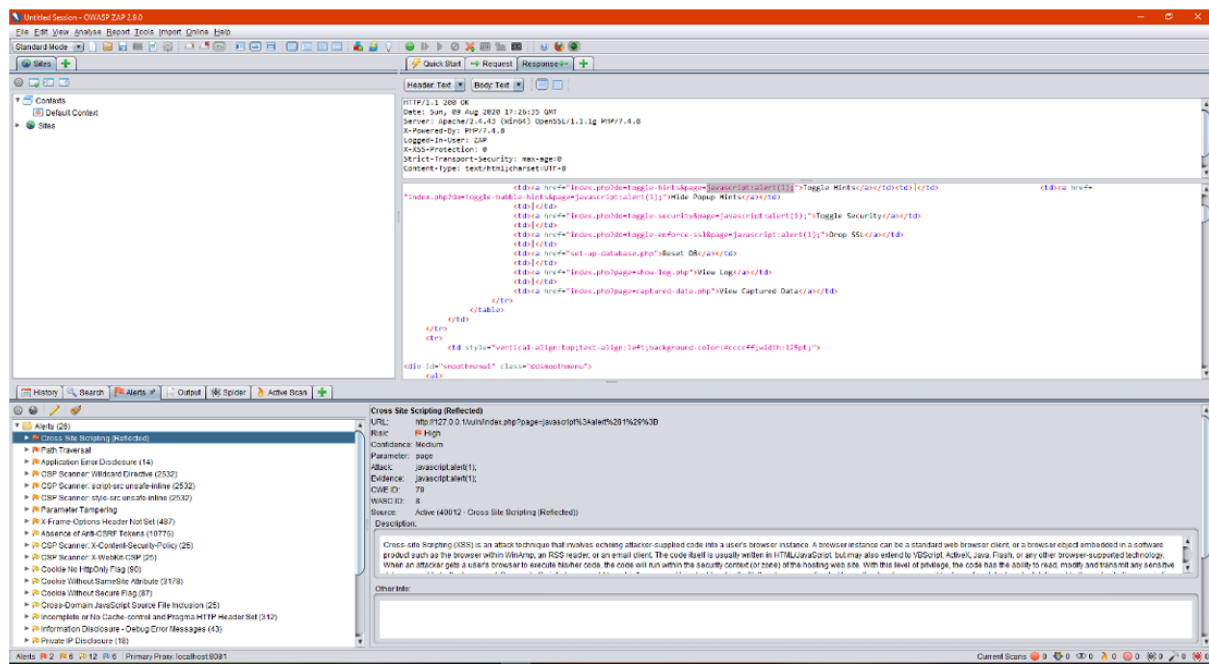
ALTORO MUTUAL VULNERABILITIES

OWASP TOP 10 SECURITY RISKS 2021

1. Broken Access Control

2. Cryptographic Failures

3. Injection

4. Insecure Design

5. Security Misconfiguration

6. Using Components with known vulnerabilities

7. Broken Authentication

8. Software and Data Integrity Failures

9. Security Logging and Monitoring Failures

10. Server-Side Request Forgery

DAST Report:

1. SQL injection.

SQL Injection is a mainstream assault, where hackers embed notorious codes so as to access the application's database.

(a) Risk/ Undesirable impact if exploited.

SQL injection attacks represent an extreme security danger to associations. A successful SQL injection assault can bring about confidential and important information being erased, edited or taken out for malicious uses.

Other risks are sites being ruined, defaced or unapproved access to frameworks or accounts and, eventually, compromised machines or whole systems.

(b) How to fix the defect.

SQL injection attacks possibly work when an application is tricked into executing code since it gets client contribution to a structure it isn't anticipating.

That implies a crucial SQL injection security measure is input disinfection and character wise approval. This adequately adds an assessment layer to guarantee that any submitted information isn't a malicious code and may represent a SQL injection hazard.

Sterilization for the most part includes running any submitted information through a scripted wall, to guarantee that any hazardous characters, like ' are not passed to a SQL question in input fields.

Character approval is somewhat helpful in saving the web application from getting injected by SQL codes.

Approval includes adding code that guarantees that any information submitted is in the structure that is normal in that specific occasion.

At the most essential level this incorporates guaranteeing that email addresses contain an "@" sign, and that numeric information, (for example, a mobile number) is accepting only numeric characters.

Also check that the length of a bit of information submitted isn't longer than the greatest anticipated length, for example a phone number field must be restricted to 10 digits.

Finally, I would recommend whitelisting legal characters and blacklisting illegal characters in each input field is the best possible way to keep a web application safe from SQL injection.

Attack on Altoro Mutual:

 # Location / URL - http://demo.testfire.net/login.jsp
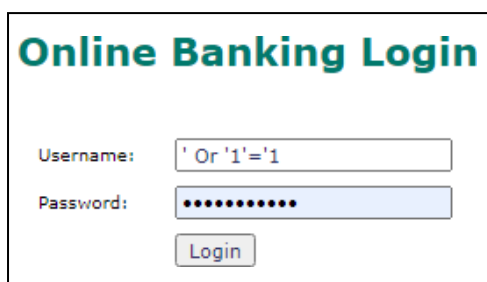
Steps and Description:

I was able to easily get through the login page of demo.testfire.net, accessible at http://demo.testfire.net/login.jsp by using the payload ' Or '1'='1.

(Ignored the ' after the last character of the payload, as it was found as already being added by the web app.)

Hello Admin User

Welcome to Altoro Mutual Online.

View Account Details: [800000 Corporate ▾] [GO]

Congratulations!

You have been pre-approved for an Altoro Gold Visa with a credit limit of €8799.02!

Click Here to apply.

Attack on Multillidae 2:

 # Location / URL – Multillidae -> OWASP 2021 -> A03 - Injection (SQL) -> SQLi - Extract Data -> **User Info (SQL)**

Steps and Description:

1.   Enter a random input to get an SQL error.

This vulnerability represents how dangerous showing an SQL error to the client can prove to be.

2.   We can see (red outlined) that the web application adds '' to the entered query and uses AND operator for password. We can also see other crucial information. We can use a universally true SQL query as a payload.

3. Entering 1' or '5' = '5 can help, and to get rid of the "AND password='password'" we can comment it out using **-- -**

4. Using the payload in step 2 we can fetch all the usernames and passwords.
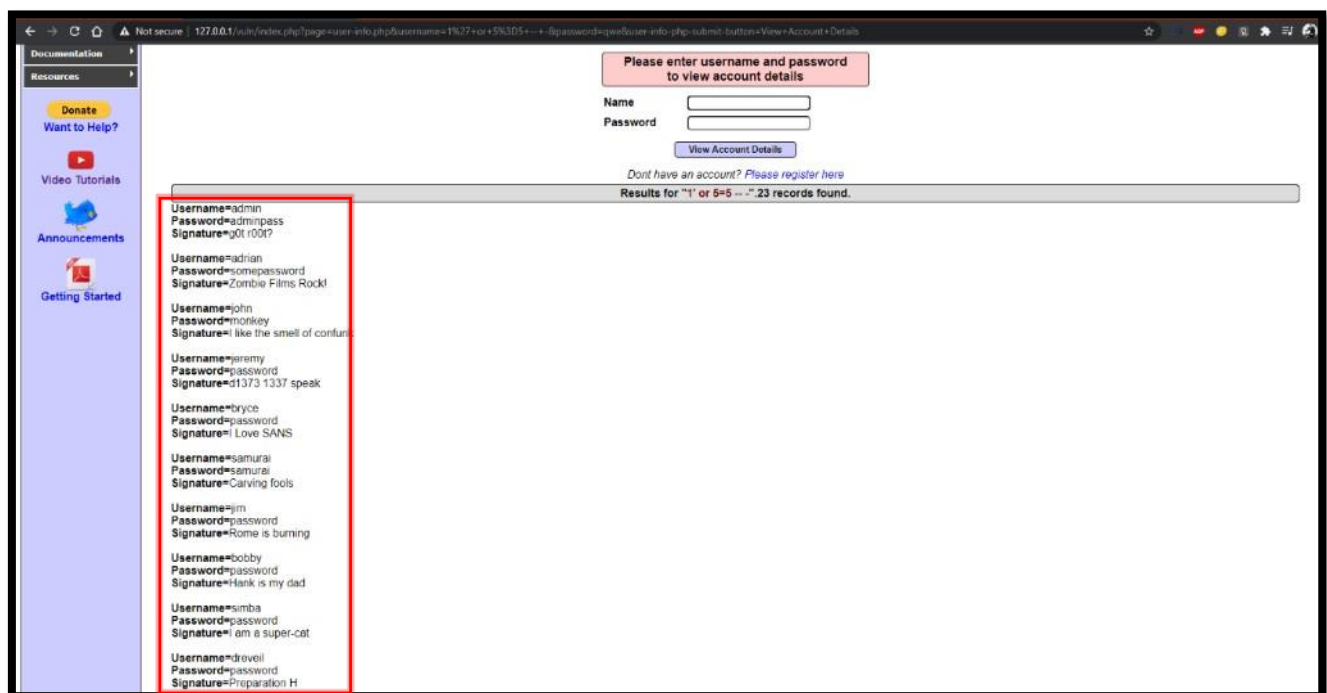
Location / URL – OWASP Top 10 --> A03 - SQL Injection --> SQLi - Extract Data --> **User Info**

Steps and Description:

1.   We type just to check what kind of error we get.

2.   We can see an error statement "The used SELECT statements have different number of columns." MySQL should not show us this information. We can exploit this vulnerability can be exploited.

3.   Tried an impractical way of guessing the no. of columns. Decided to try upto 5 null statement. In input I typed:

' union select null,null -- ,

' union select null,null,null --   ,
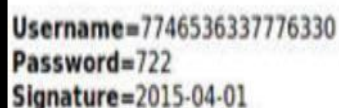
' union select null,null,null,null – and

' union select null,null,null,null,null --

When used null 5 times, got output username, password and signature.

4.   Proceeded further to exploit this by ' union select 1,2,3,4,5 --.

5.   The username field populated with 2, password with 3 and signature field got value 5. Indicating that Username is the second table field, Password is the third table field, and Signature is the fourth table field.

6.   We can use this vulnerability to fetch all details from other tables of database, like credit card details etc.

```
Username=77465363337776330
Password=722
Signature=2015-04-01

Username=8242325748474749
Password=461
Signature=2016-03-01

Username=7725653200487633
Password=230
Signature=2017-06-01

Username=1234567812345678
Password=627
Signature=2018-11-01
```
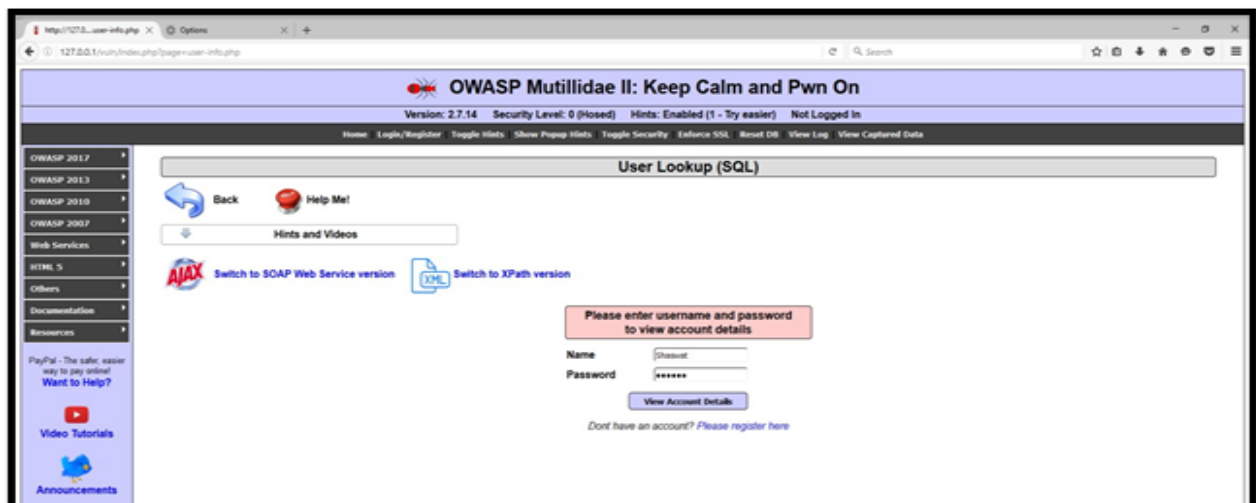
Location / URL – OWASP Top 10 --> A03 - SQL Injection --> SQLi - Extract Data --> **User Info**
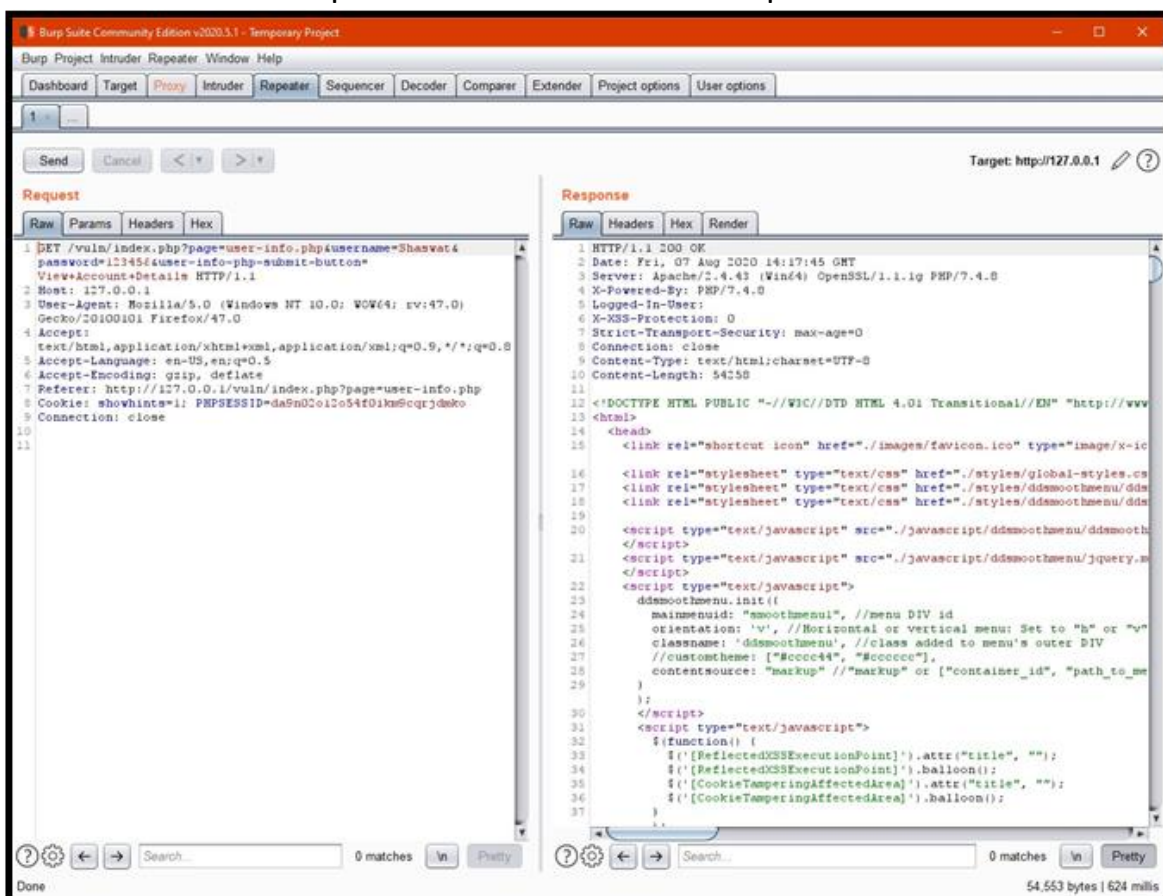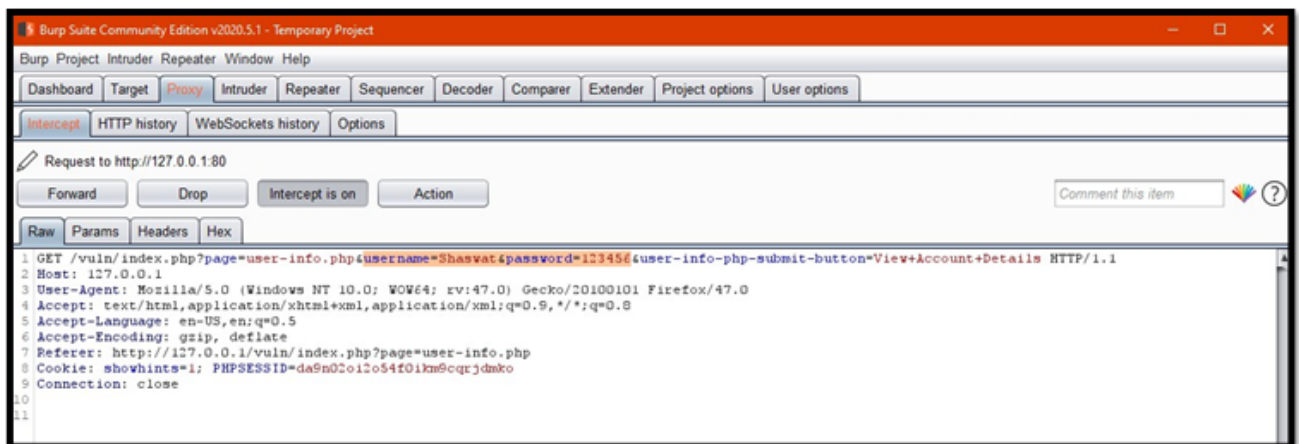
Steps and Description:

SQL Injection with SQLMap through Burp Suite.

1.   On the User Info page, we capture a normal request using Burp Suite.

2.   We take the request and forward it to the repeater.

3. Save the repeater request to a file, and send it through SQLMap.
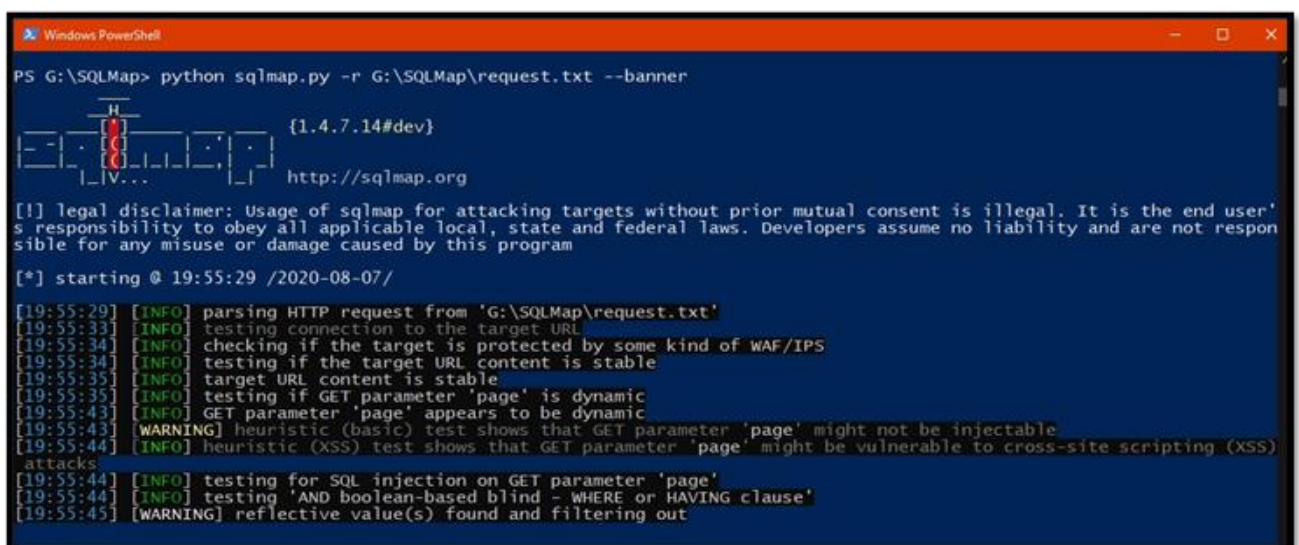
4. SQLMap finds the backend DBMS to be MYSQL.

5. We can pass this detail to further exploit the database.
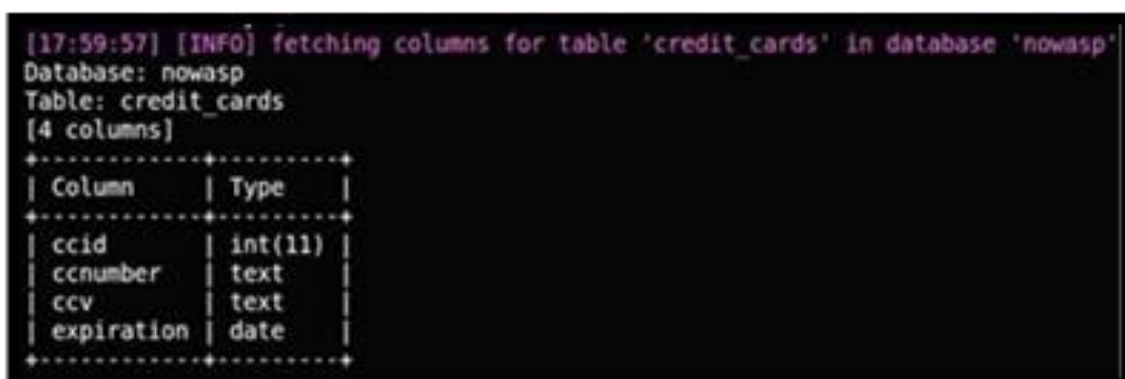
 # Location / URL – OWASP 2017 --> A03 - SQL Injection --> SQLMap Practice -->
View Some ones Blog

Steps and Description.

1.   View blog entries of a random person. And intercept that request using
Burp Suite.
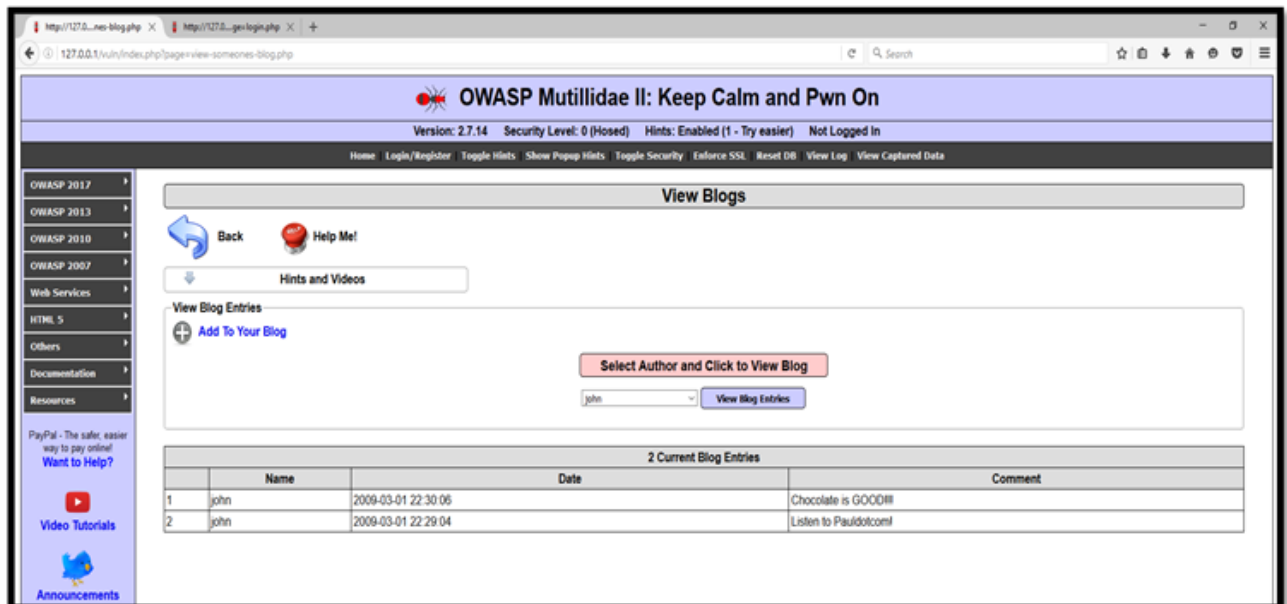
2.   Add ' in place of user to check for SQLi Vulnerability.

3.   After forwarding the intercepted packet, the error shown below proves
the existence of SQLi vulnerability.

4. We can see the SQL Query used. [The last sentence in Message field]

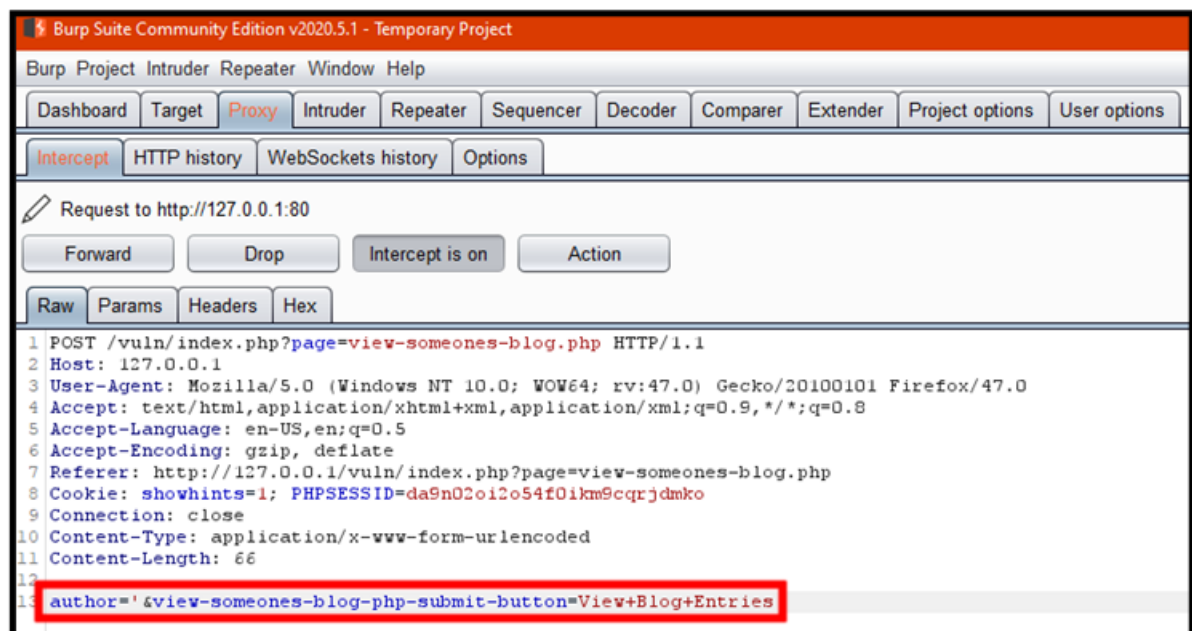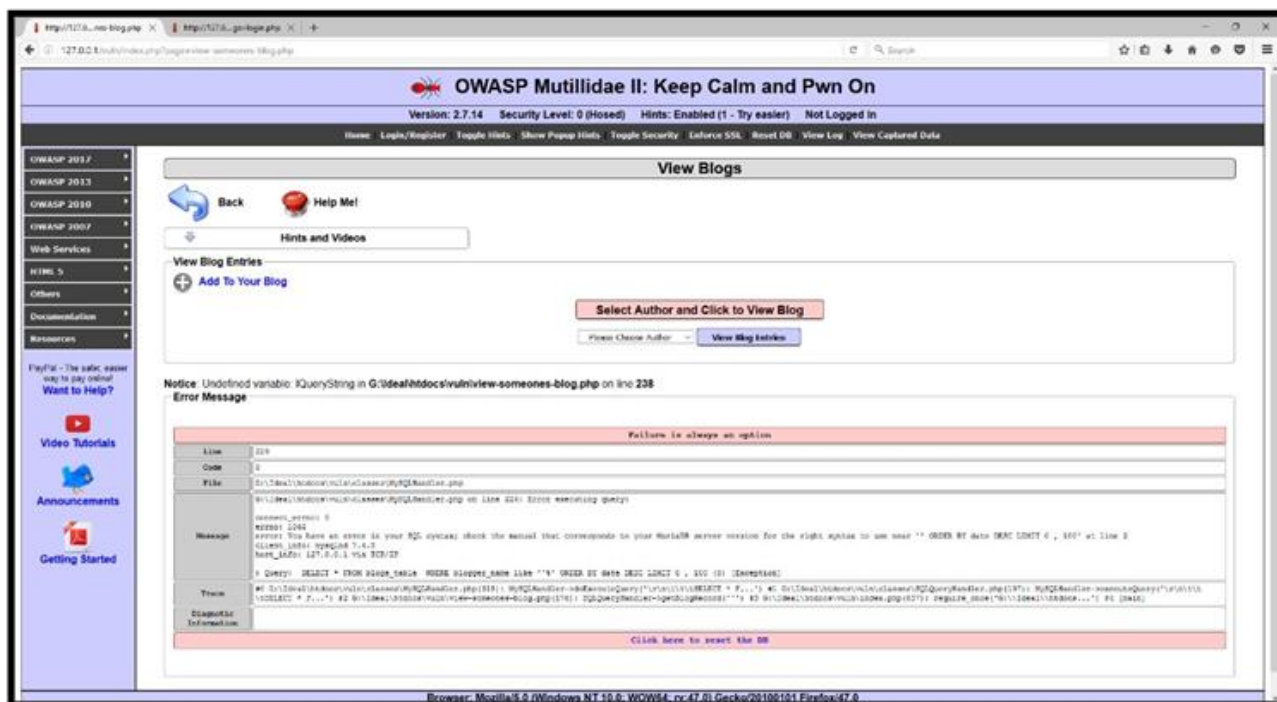|	
	Failure is always an option
Line	229
Code	0
File	G:\Ideal\htdocs\vuln\classes\MySQLHandler.php
Message	G:\Ideal\htdocs\vuln\classes\MySQLHandler.php on line 224: Error executing query: connect_errno: 0 errno: 1064 error: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '' ORDER BY date DESC LIMIT 0 , 100' at line 2 client_info: mysqlnd 7.4.8 host_info: 127.0.0.1 via TCP/IP) Query: SELECT * FROM blogs_table WHERE blogger_name like ''%' ORDER BY date DESC LIMIT 0 , 100 (0) [Exception]
Trace	#0 G:\Ideal\htdocs\vuln\classes\MySQLHandler.php(315): MySQLHandler->doExecuteQuery('\r\n\t\t\tSELECT * F...') #1 G:\Ideal\htdocs\vuln\classes\SQLQueryHandler.php(197): MySQLHandler->executeQuery('\r\n\t\t\tSELECT * F...') #2 G:\Ideal\htdocs\vuln\view-someones-blog.php(176): SQLQueryHandler->getBlogRecord('') #3 G:\Ideal\htdocs\vuln\index.php(637): require_once('G:\\Ideal\\htdocs...') #4 {main}
Diagnostic Information	
	Click here to reset the DB

5. Save the request from Burp Suite HTTP history, to a file. And run SQLMap, and try to grab the database using –dbs command.
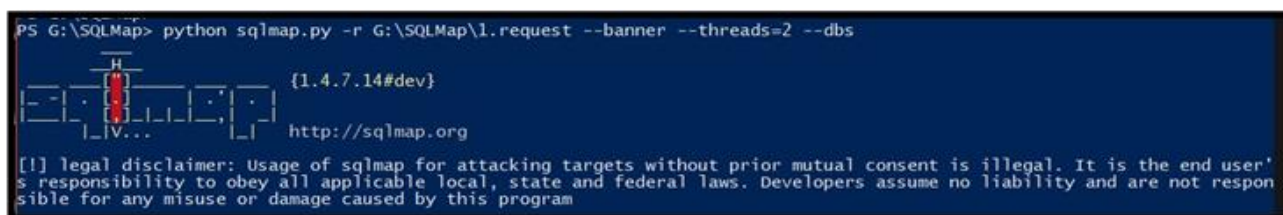
6. We get a list of databases when the scan finishes.

```
available databases [8]:
[*] cdcol
[*] information_schema
[*] mysql
[*] owasp10
[*] performance_schema
[*] phpmyadmin
[*] test
[*] webauth
```

2.   Cross Site Scripting [XSS]

(a)   Risk/ Undesirable impact if exploited.

Cross-site scripting (XSS) is a web security weakness that allows an attacker to compromise the interaction that the clients have with an insecure web application.

Cross-site scripting weaknesses regularly permit an attacker to act as a victim client, to complete any activities that the client can perform, and to get to any of the client's information.

XSS is presumably the most common high risk web application vulnerability these days, but then it is as yet one of the most disregarded by engineers and security experts.

There are three primary sorts of XSS assaults. These are:

•   Reflected XSS, where the malicious script comes from the current HTTP request.

•   Stored XSS, where the malicious script comes from the website's database.

•   DOM-based XSS, where the vulnerability exists in client-side code rather than server-side code.

One of the most widely recognized XSS assault vectors is to seize real client accounts by taking their session cookies. This permits attackers to mimic

victims and access any touchy data or usefulness for their benefit. How about we analyze how this can be accomplished.

A down to earth assault vector for XSS is to utilize HTML and JavaScript so as to take client credentials, rather than their cookies. This should be possible by cloning the login page of the web application and afterward utilizing the XSS weakness so as to serve it to the victims.

Another amazing assault vector for XSS is to utilize it so as to exfiltrate private information or to perform unapproved tasks. Some other well-known manners by which XSS could be weaponized incorporate the accompanying:

•   Keylogger: utilizing JavaScript it is conceivable to log every single key stroke that a client enters in a weak page.

•   Port output: XSS is additionally an unforeseen source to start port sweeps against the inner system of a customer that gets to a weak site.

•   Website Defacing: one of the most straightforward but then viable ways for aggressors to target organizations or government establishments is to change the visual appearance of a site helpless against XSS. Either utilizing humiliating pictures or hacktivism messages this can carry associations to the spotlight for an inappropriate reason.

(b)   How to fix the defect.

So as to limit the dangers related with XSS, developers ought to encode all fields while showing them in the web browser. Furthermore, guarantee that client input is appropriately filtered on account of extraordinary characters. A typical wellspring of XSS are obsolete 3rd party libraries incorporated in the code, and all things considered, update these to the most recent stable renditions. As a component of a protection in-depth system, guarantee that cookie properties, (for example, HttpOnly) and security headers, particularly (Content Security Policy) CSP, are set in like manner.

On a more significant level, guarantee that security is appropriately coordinated in all periods of the improvement procedure and that developers know about normal web application weaknesses. At last, customary infiltration

tests would help recognize such defects and improve the security position of the web applications.

Most by far of XSS weaknesses can be found rapidly and dependably utilizing Burp Suite's web weakness scanner.

Physically testing for reflected and stored XSS regularly includes injecting a short alphanumeric string into each passage point in the application; recognizing each area where the submitted input is returned in HTTP reactions; and testing every area independently to decide if reasonably created information can be utilized to execute subjective JavaScript.

Physically testing for DOM-based XSS emerging from URL boundaries includes a comparative procedure: putting some basic interesting contribution to the boundary, utilizing the program's engineer instruments to scan the DOM for this info, and testing every area to decide if it is exploitable. Be that as it may, different sorts of DOM XSS are more earnestly to identify. To discover DOM-based weaknesses in non-URL-based info, (for example, document.cookie) or non-HTML-based sinks (like setTimeout), there is not a viable alternative for investigating JavaScript code, which can be very tedious. Burp Suite's web weakness scanner joins static and dynamic examination of JavaScript to dependably computerize the location of DOM- based weaknesses.

XSS weaknesses can be fixed by including a blend of the following measures:

•   Channel contribution on appearance. At where client input is gotten, channel as carefully as conceivable dependent on what is normal or substantial info.

•   Encode information on yield. At where client controllable information is yield in HTTP reactions, encode the yield to keep it from being deciphered as dynamic substance. Contingent upon the yield setting, this may require applying mixes of HTML, URL, JavaScript, and CSS encoding.

•   Utilize fitting reaction headers. To forestall XSS in HTTP reactions that aren't expected to contain any HTML or JavaScript, you can utilize the Content-Type and X-Content-Type-Options headers to guarantee that programs decipher the reactions in the manner you plan.

- • Content Security Policy. As a last line of safeguard, you can utilize Content Security Policy (CSP) to diminish the seriousness of any XSS weaknesses that despite everything happen

Attack on Altoro Mutual:

 # Location / URL – demo.testfire.net/search.jsp

Steps and Description.

Enter "<script>alert(1)</script>" in the search box returns with the output programmed by us. This shows the search box is vulnerable to Reflected XSS.
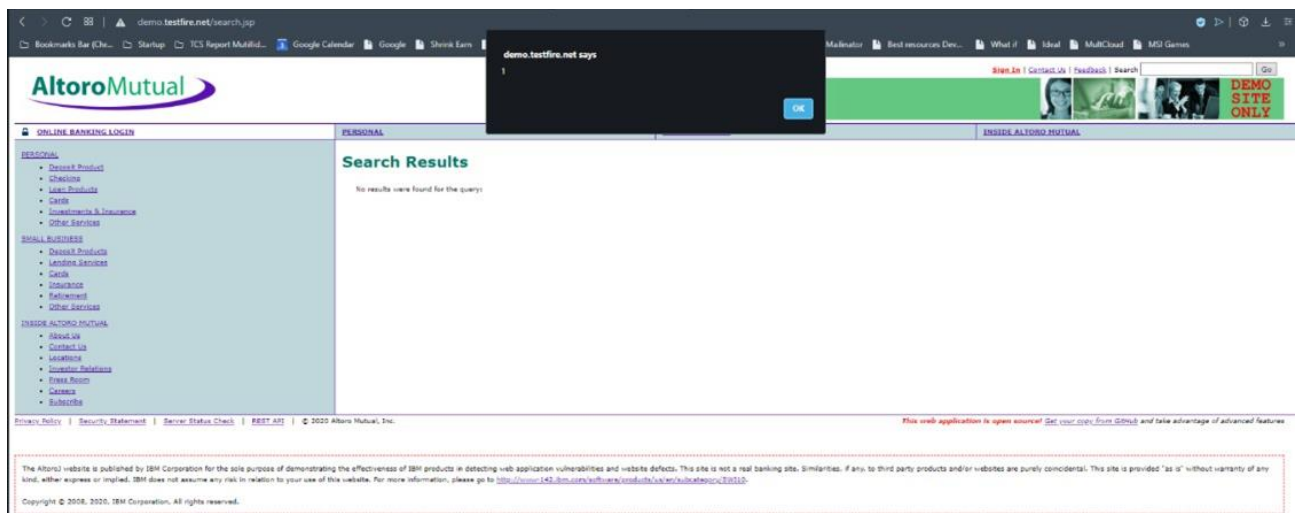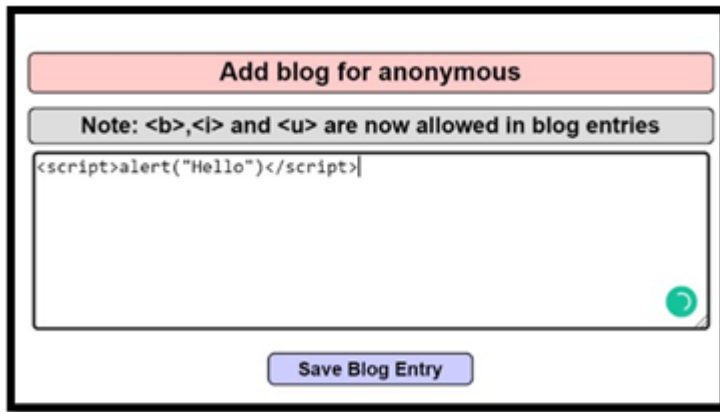

Attack on Multillidae 2:

Steps and Description.

This is a persistent XSS in Multillidae. Whenever a user reaches or refreshes this page, the malicious code is run.

We have a single step,

Simply inject a javascript code in the "Add to your blog"

input field.

Add blog for anonymous

Note: <b>,<i> and <u> are now allowed in blog entries

```
<script>alert("Hello")</script>
```

Save Blog Entry

We get the output as programmed. One thing to notice is that whenever this page is refreshed the code runs again, this shows that the injected code has been embedded in to the source of the web application.

Hence making it a stored / persistent XSS vulnerability.

127.0.0.1 says

Hello

OK
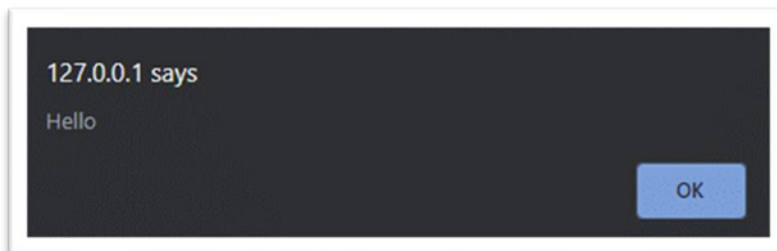
 # Location / URL – OWASP Top 10 --> A03 – SQl injection(Cross Site Scripting (XSS)) --> Reflected (First Order) --> **DNS Lookup**

Steps and Description.

In the DNS Lookup input field place the following string "<script>alert(document.cookie)</script>".

The goal here is to determine:

1.   if this webpage contains a cookie.

2.   if we can display the cookie in a JavaScript alert box.
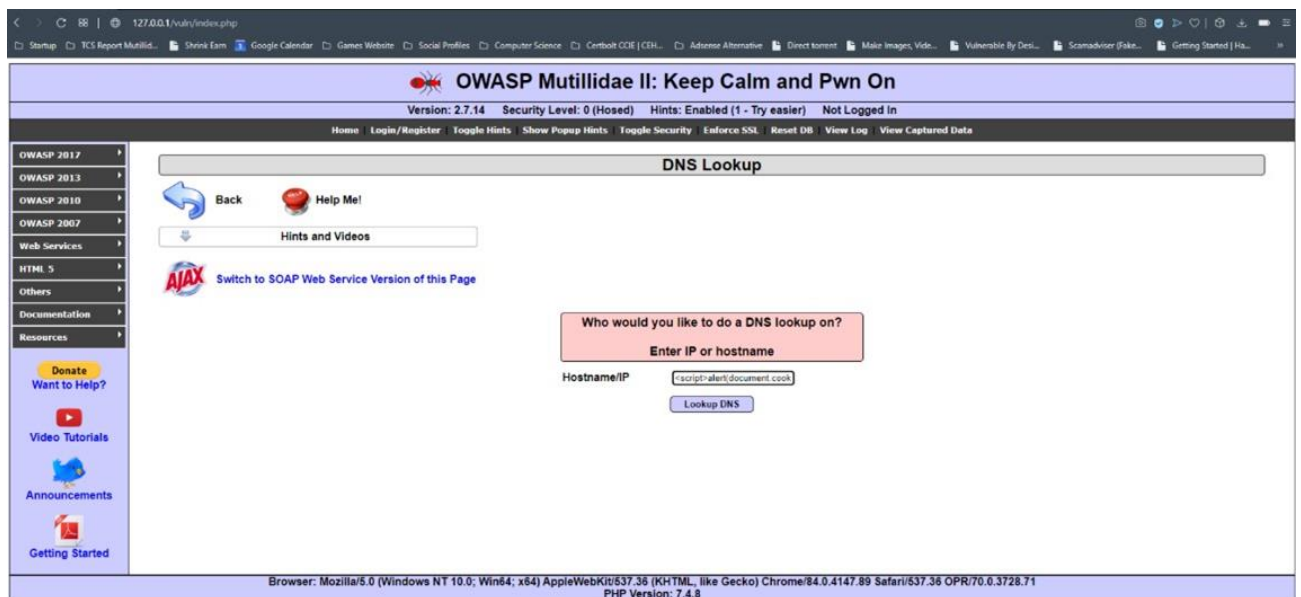
We get the following output.
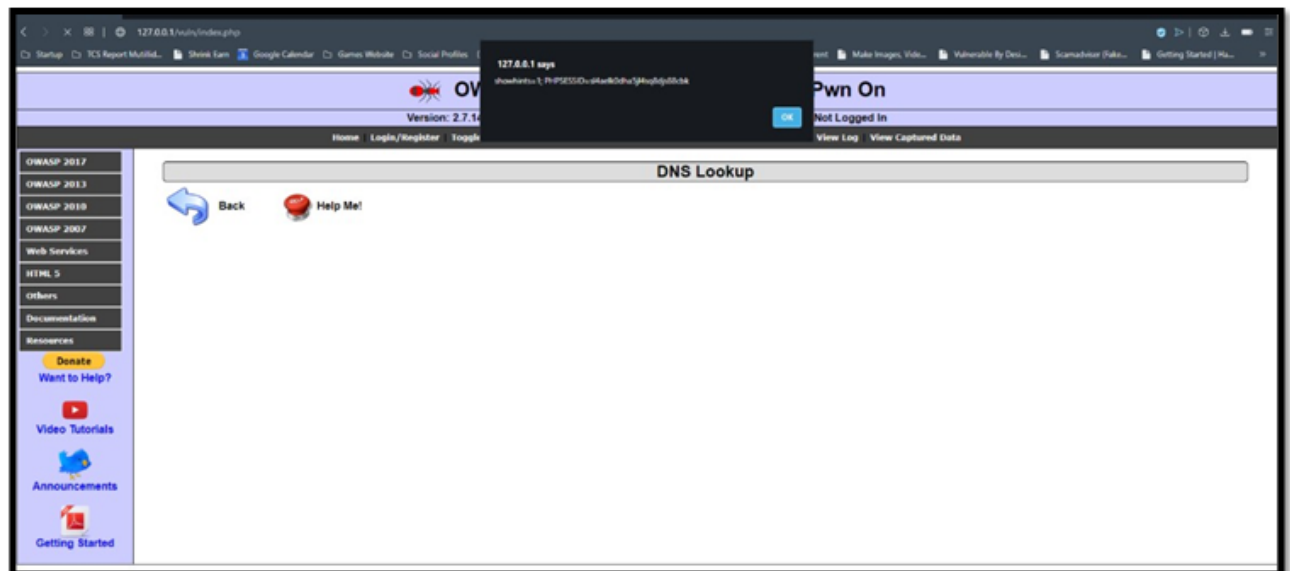
We can clearly see the cookie displays the username and the PHP Session ID.

127.0.0.1 says

showhints=1; PHPSESSID=sl4aelk0dha5jl4sq8djs88cbk

OK
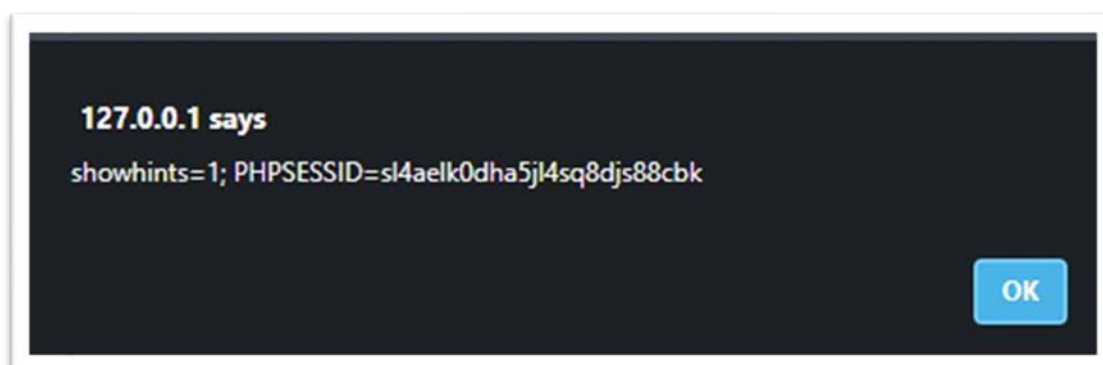
3.   In the manual testing of demo.testfire.net, found some extremely high risk issues related to security misconfiguration.

•   debugger is enabled

•   incorrect directory permissions

•   using default accounts and passwords

•   setup/config pages enabled

3.   Cryptographic Failure (Information Disclosure or Sensitive Data Exposure)

(a)   Risk/ Undesirable impact if exploited.

Sensitive Data Exposure happens when an application doesn't enough secure touchy data. The information can fluctuate and anything from passwords, meeting tokens, charge card information to private wellbeing information and more can be uncovered. As the finding just applies to delicate information, the potential effect is constantly viewed as high. What the information comprises of shifts thus does the effect. The peril lies in the information being uncovered, and the potential effect mirrors the information's affectability.

Touchy information presentation happens because of not enough securing a database where data is put away. This may be a consequence of a huge number of things, for example, powerless encryption, no encryption, programming blemishes, or when somebody erroneously transfers information to an inaccurate database.

Various sorts of information can be uncovered in a delicate information introduction. Banking account numbers, Mastercard numbers, human services information, meeting tokens, Social Security number, street number, telephone numbers, dates of birth, and client account data, for example, usernames and passwords are a portion of the sorts of data that can be left uncovered.

Most weaknesses inside this classification can't be filtered for because of two principle reasons:

•   To decide chance, it must be chosen what data is viewed as delicate, which can be a hard undertaking to complete consequently.

• An outer pentester can't know whether inner information is scrambled or not as that isn't uncovered.

(b)   How to fix the defect.

The initial step is to make sense of what information can be viewed as delicate and in this manner essential to secure. At the point when that is done, turn out every one of these information focuses and ensure that:

• The information is never put away in clear content.

• The information is never sent in clear content. Model among database and worker, or over the web.

• The calculations used to scramble the information are viewed as sufficient.

• The age of the keys is secure.

• Program headers are set to not store when the delicate information is introduced to end-client.

• Here are a few hints that can help.

 Utilize a remarkable and complex secret key for every one of your online records. Monitoring each one of those passwords can be troublesome, yet there are items, for example, Norton Password Manager, that can help make this assignment simpler to oversee.

Utilize just secure URLs. Be certain that you are visiting a notable site that you trust. For the most part, respectable destinations start with https://. The "s" is vital. This is particularly significant when entering charge card or other individual data.

 # Location / URL – http://127.0.0.1/vuln/index.php?page=phpinfo.php

Steps and Description.

Multillidae has provide access to PHP info for all end users, this can be a high risk of sensitive data exposure.

PHP Version 7.4.8

System	Windows NT MRSHASH_PC 10.0 build 19041 (Windows 10) AMD64
Build Date	Jul 9 2020 11:24:06
Compiler	Visual C++ 2017
Architecture	x64
Configure Command	cscript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	G:\Ideal\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,TS,VC15
PHP Extension Build	API20190902,TS,VC15
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar
Registered Stream Socket Transports	tcp, udp, ssl, tls, tlsv1.0, tlsv1.1, tlsv1.2, tlsv1.3
Registered Stream Filters	convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, zlib.*, bzip2.*

This program makes use of the Zend Scripting Language Engine:
Zend Engine v3.4.0, Copyright (c) Zend Technologies

zend engine

Security Configuration must be defined and deployed for the application, frameworks, application server, web server, database server, and platform. If these are properly configured, an attacker can have unauthorized access to sensitive data or functionality.

Sometimes such flaws result in complete system compromise. Keeping the software up to date is also good security.

 # Location / URL – http://demo.testfire.net/index.jsp?content=inside_jobs.htm

Steps and Description.

The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.

Screenshot of the rendered page containing this vulnerability.

Location / URL – OWASP 2021 -> A05 - **Security Misconfiguration** -> **Unrestricted File Upload**

Steps and Description.

This page lets the client transfer any sort of file. This must be confined; in any case this can prompt the transfer of malicious php codes into the server which can prompt trade off of the database.

Also, an attacker can

• Use this vulnerability to upload a php backdoor shell into the server to get complete access to the files and the databases within.

• Do great damages by uploading huge files and hence causing file space denial of service.

• Or can upload file using malicious path or name to overwrite a critical file.
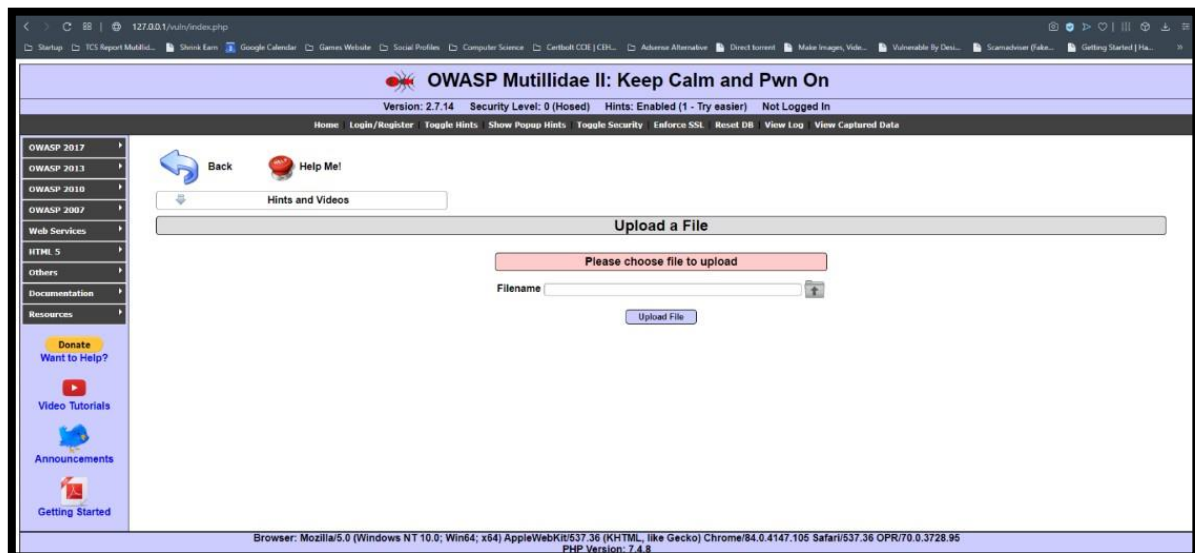
Proof of a php backdoor getting accepted as an upload.
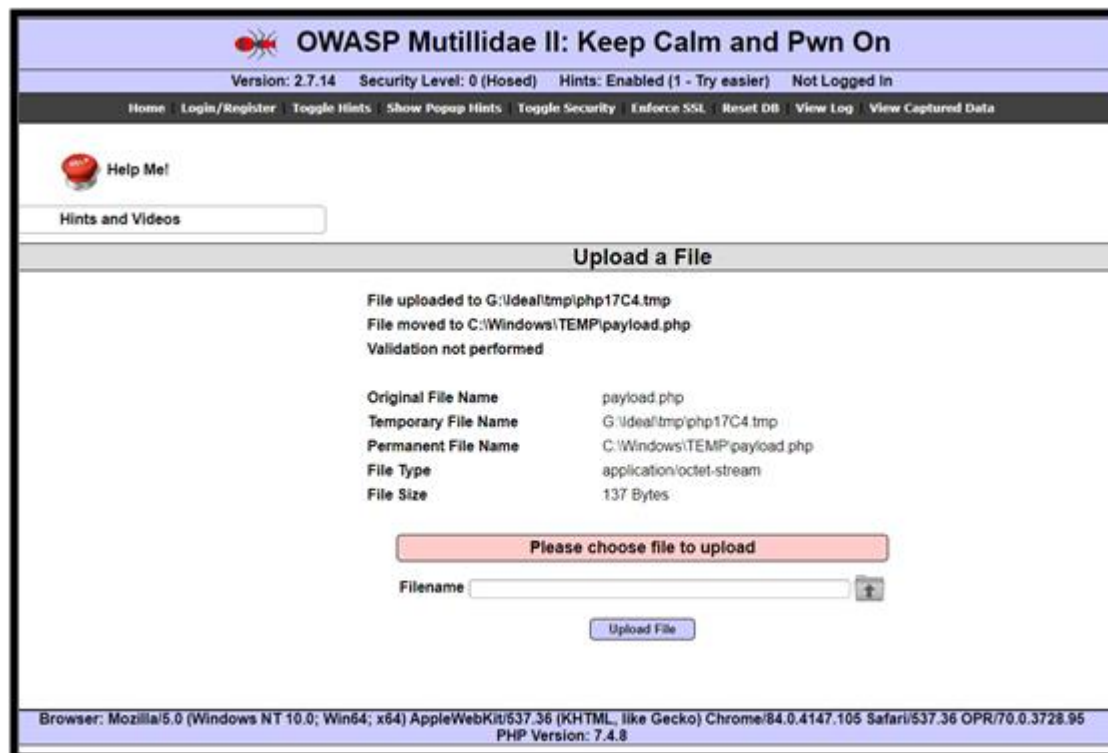
To exploit this further, I uploaded an HTML file which was accepted and was accessible on the server. It can replace the original index.html file too. Causing a website defacement.

We can navigate to the file on the server.

We can see a complete HTML file that is stored on the server and if this replaces index.html, this is the file every visitor sees.
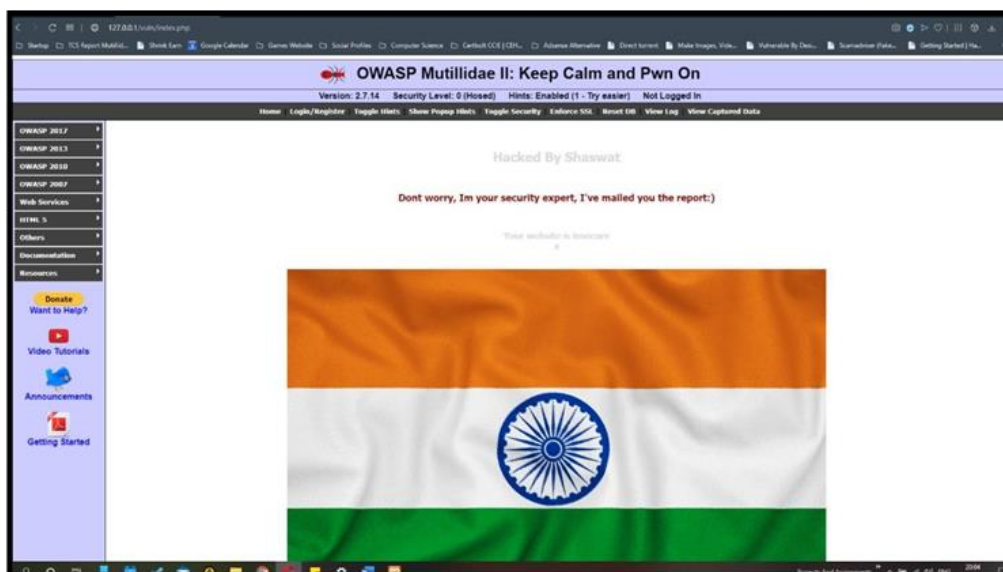
4.   Broken Authentication

(a)   Risk/ Undesirable impact if exploited.

Broken authentication is on the most recent (2021) OWASP Top 10 list. Broken authentication is commonly brought about by inadequately implemented authentication and session management.

Broken authentication attacks intend to assume control more than at least one records giving the aggressor indistinguishable benefits from the victim client.

Because of helpless plan and usage of character and access controls, the pervasiveness of broken validation is broad.

 Normal hazard factors include:

•   Unsurprising, easy to guess login credentials

•   Client confirmation certifications that are not secured when put away

•   Meeting IDs uncovered in the URL (e.g., URL modifying)

•   Meeting IDs powerless against meeting obsession assaults

•   Meeting esteem that doesn't break or get nullified after logout

•   Meeting IDs that are not turned after effective login

•   Passwords, meeting IDs, and different qualifications sent over decoded associations

(b)   How to fix the defect.

The most possible fixes have been listed into bullet points below:

•   Where conceivable, execute multifaceted verification to forestall computerized, accreditation stuffing, animal power, and taken qualification re-use assaults.

•   Try not to transport or send with any default accreditations, especially for administrator clients.

•   Actualize powerless secret word checks, for example, testing new or changed passwords against a rundown of the best 10000 most exceedingly terrible passwords.

• Adjust secret phrase length, unpredictability and turn arrangements with NIST 800-63 B's rules in area 5.1.1 for Memorized Secrets or other present day, proof based secret key strategies.

• Guarantee enrollment, certification recuperation, and API pathways are solidified against account count assaults by utilizing similar messages for all results.

• Breaking point or progressively delay fizzled login endeavors. Log all disappointments and ready managers when qualification stuffing, beast power, or different assaults are recognized.

• Utilize a worker side, secure, worked in meeting chief that creates another irregular meeting ID with high entropy after login. Meeting IDs ought not be in the URL, be safely put away and nullified after logout, inert, and outright breaks.

Location / URL – Mutillidae -> OWASP 2021 -> /index.php?page=**register.php**
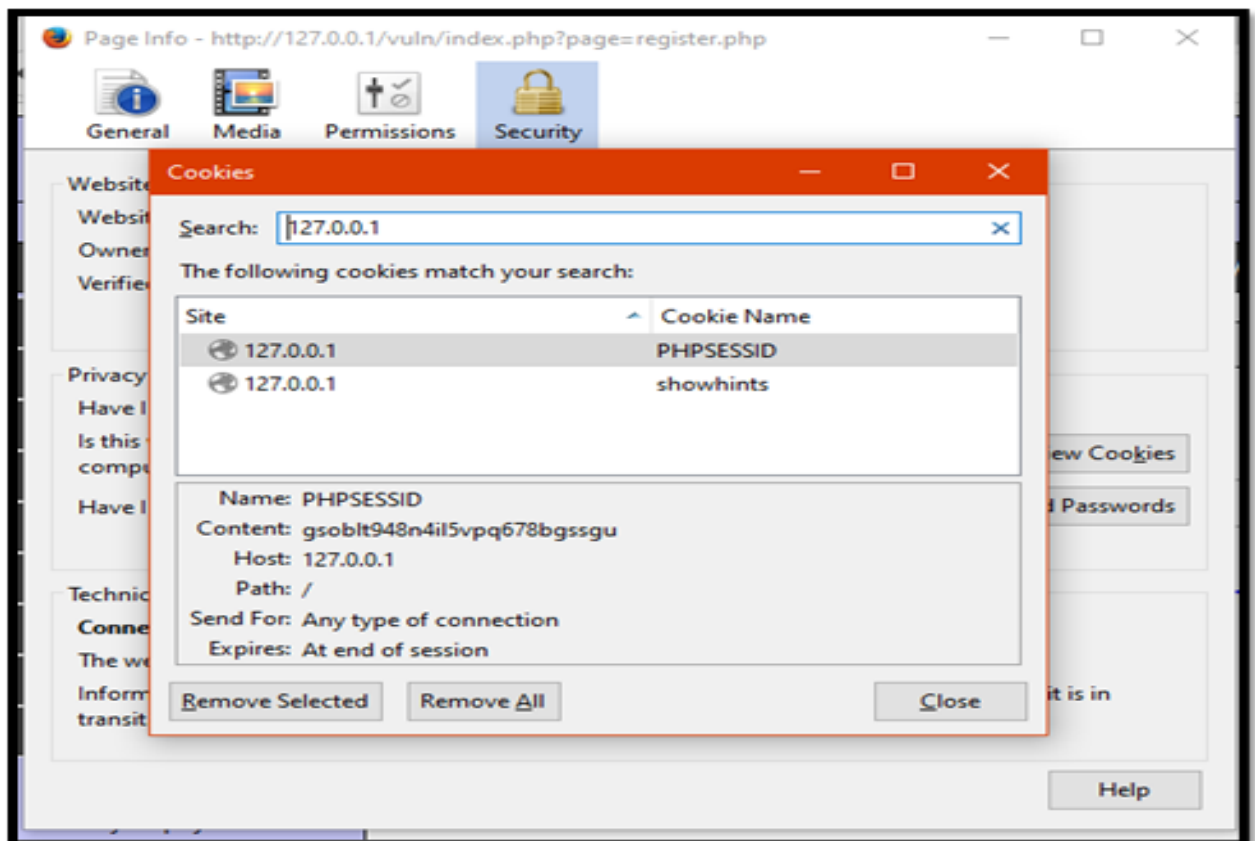
Steps and Description.

This page contains a register from to create a new user.

I will try to bypass authentication via cookie manipulation.
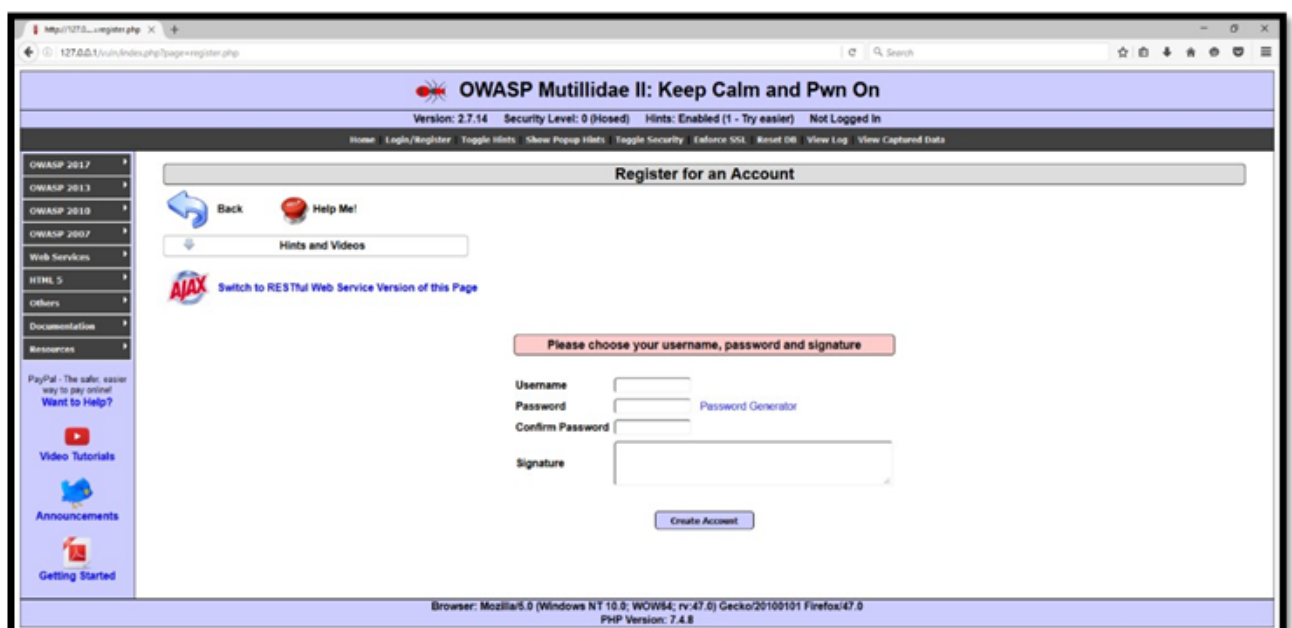
We can see the page has two active cookies. The PHPSESSID cookie can help us.

Create a new user, to find out how the cookie changes.

Login to the newly created account. We can see two new cookies.

We can see the cookies mention

• Username - Shaswat

• uid - 26

I created a new user and noticed the uid cookie increases by 1 each time, and hence we can manipulate the cookie to bypass authentication and visit other user's dashboard.
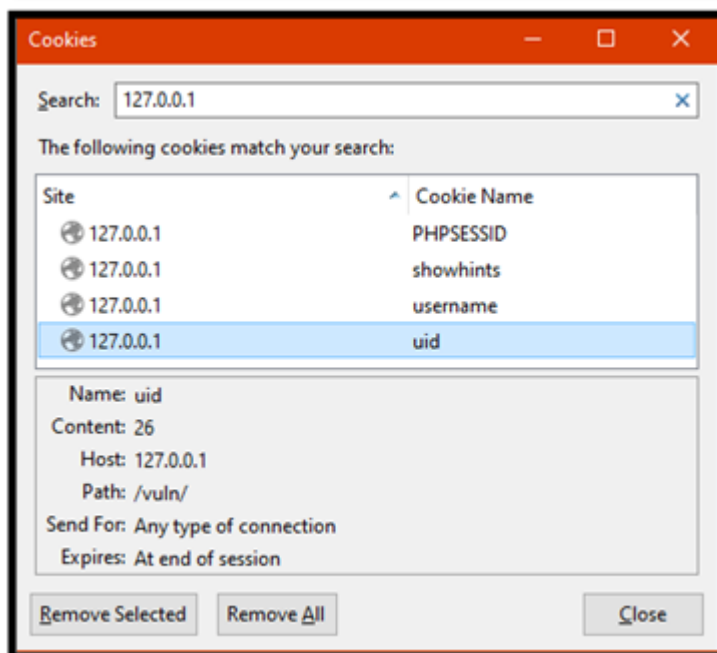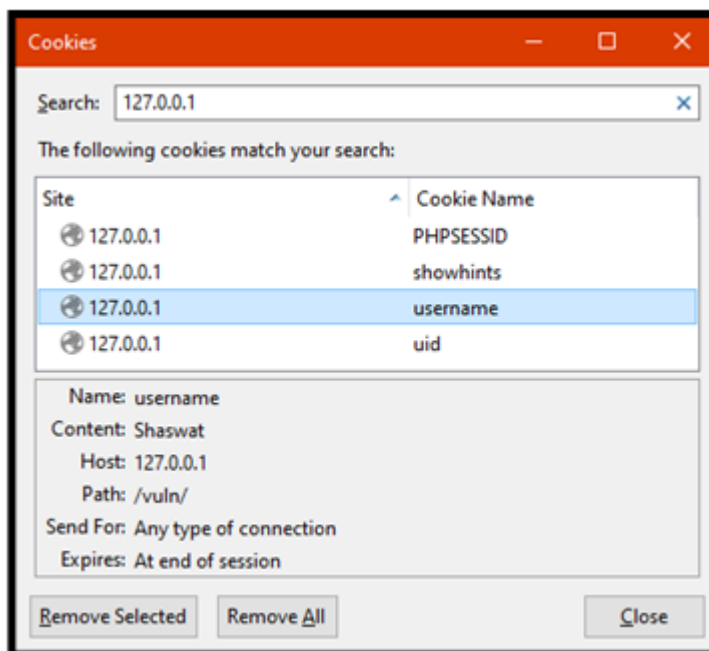

Editing the uid cookie to any other number simply logs me into the respective profiles with the input uid.

Location / URL – Mutillidae -> OWASP 2021 -> Broken Authentication and Session Management -> **Authentication Bypass-> Via Brute Force**

Steps and Description.

We can brute force into the login here.

First we need to know some usernames, which we can find on the "View Someone's Blog" page.
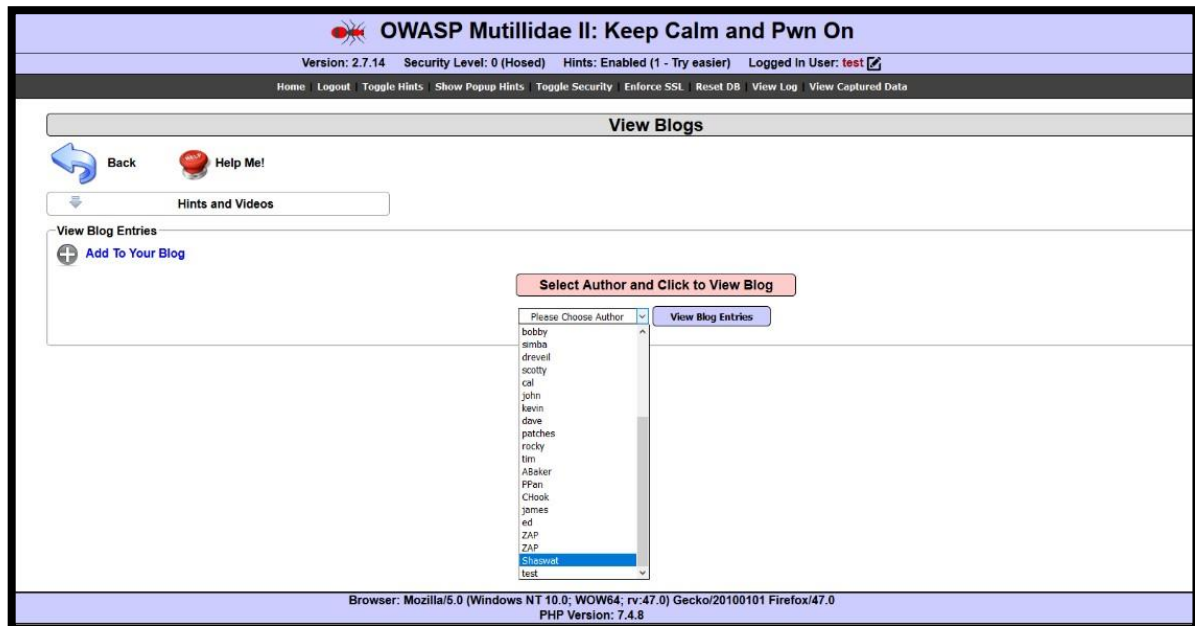
Now that we know all the usernames, we can try to brute force into one of them. Go to the login page and intercept the login request using Burp Suite.
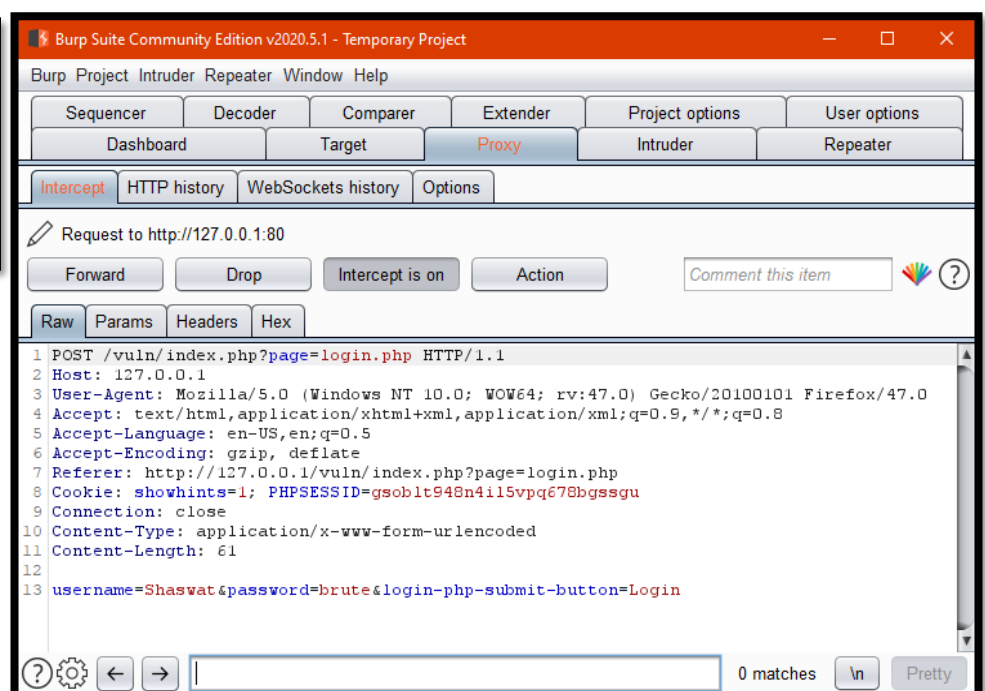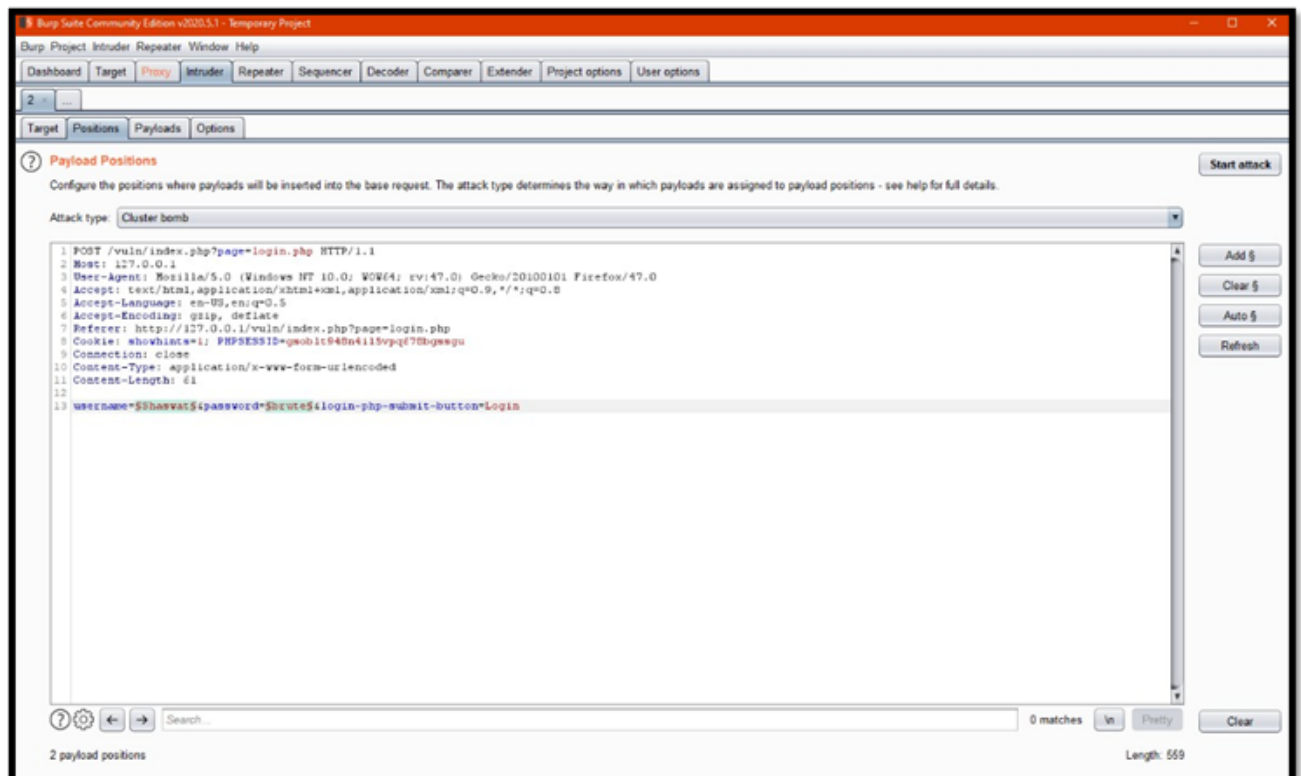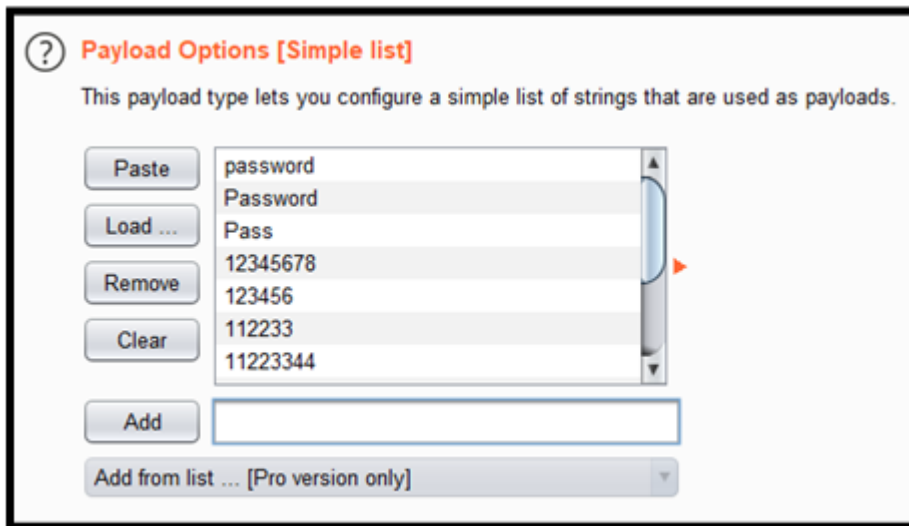
Send the intercepted request to the intruder, and select the username and password fields only. Use Cluster Bomb Attack Type
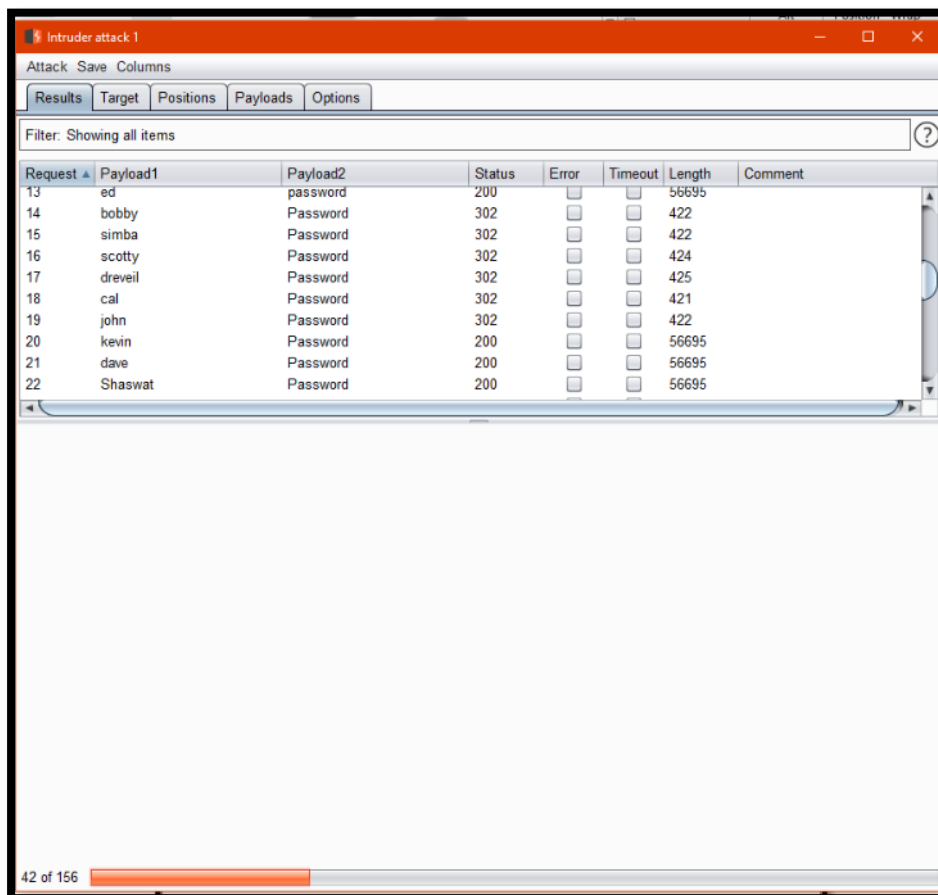

In the list 1 of payloads use a manually input usernames list.
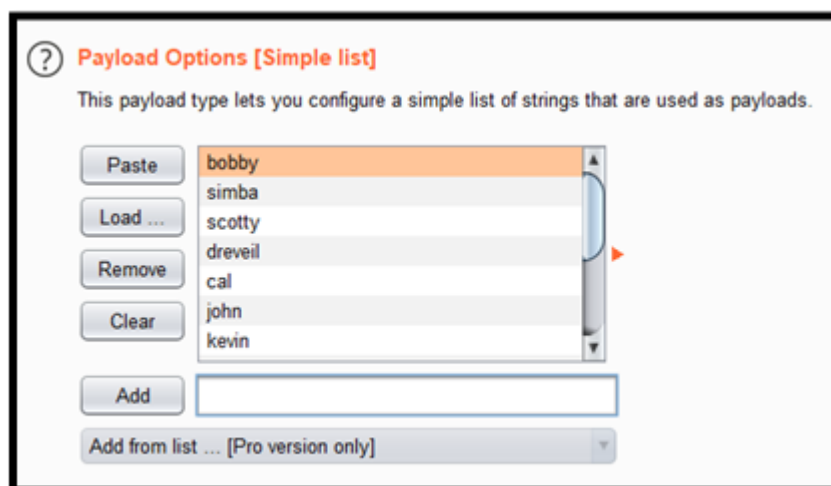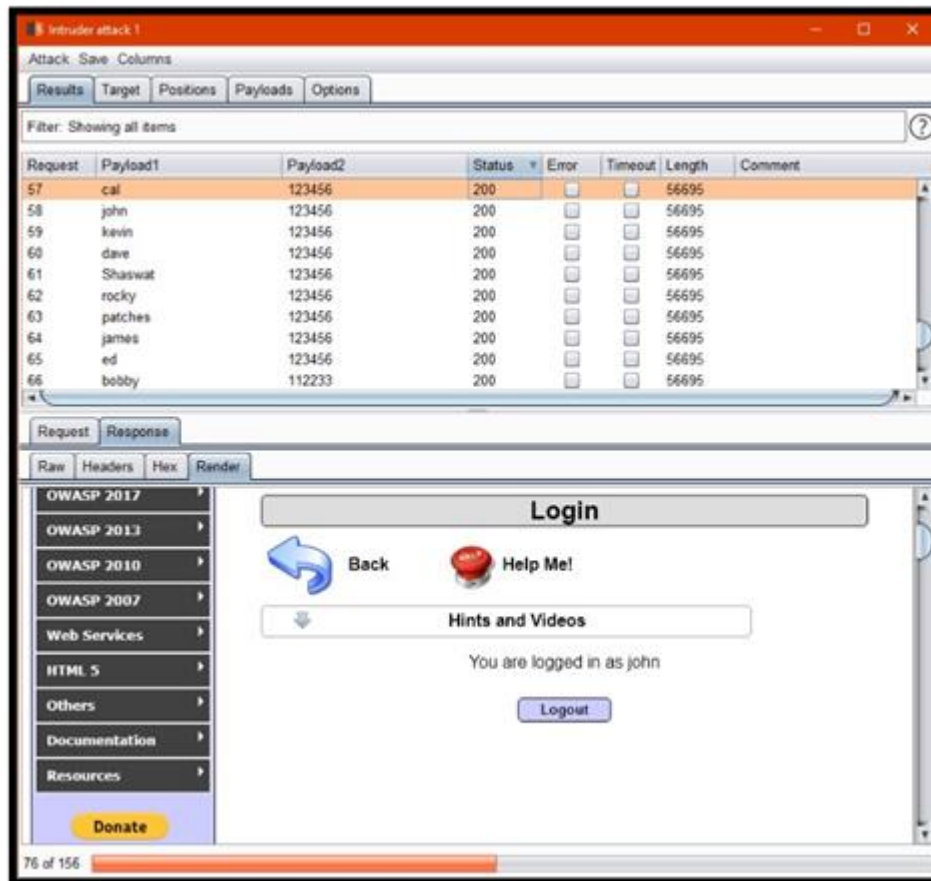

And for the list 2 use a list of most common passwords, we can get most common password list online.

Start intruder brute force.

We can see, many users have poor passwords, already listed in popular lists online.

I recommend using a minimum of 12 character password which includes a special character, a capital alias character, a small alphabet and a numeric character.

**Moving forward to OWASP ZAP automated testing tool.**

Here are the results I got about the vulnerabilities found in demo.testfire.net.

➢ demo.testfire.net Automated Scan Using OWASP ZAP
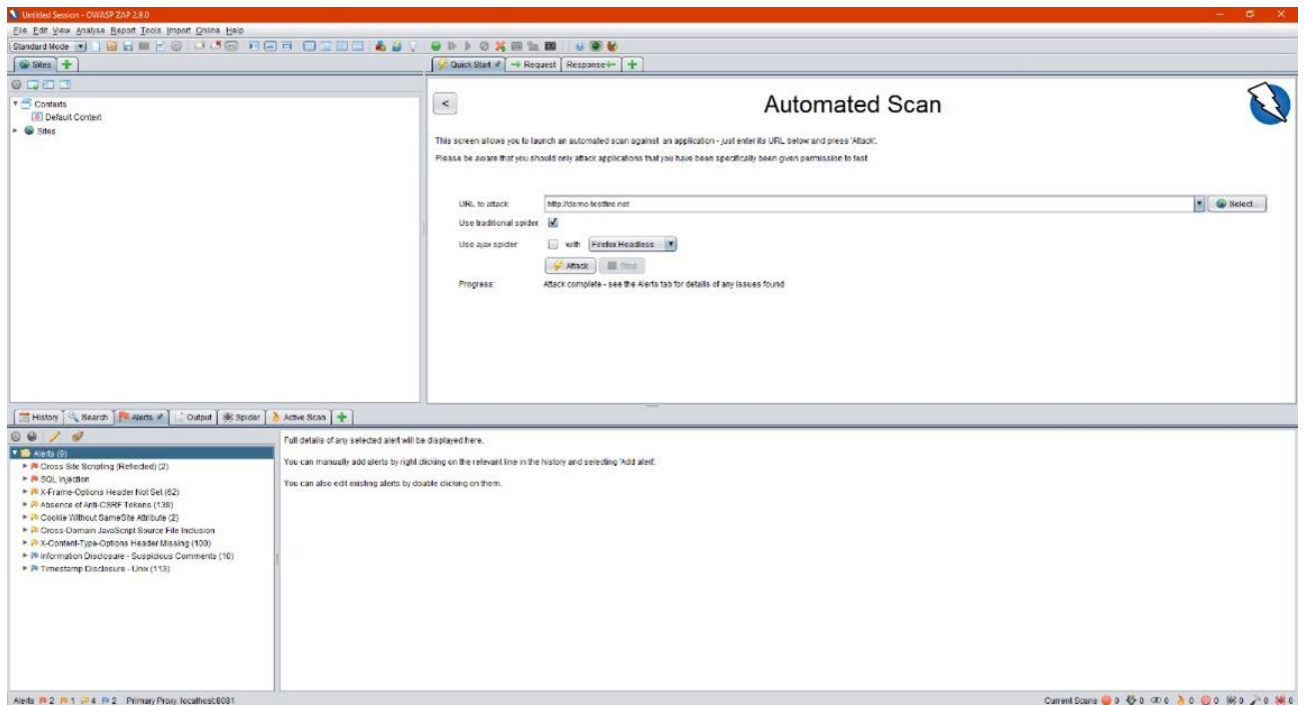
Some of these found vulnerabilities were already found in manual testing while some where found to be false positives. This was one of the errors when we try to open a found vulnerability from ZAP in system browser.

This indicates a false positive report.
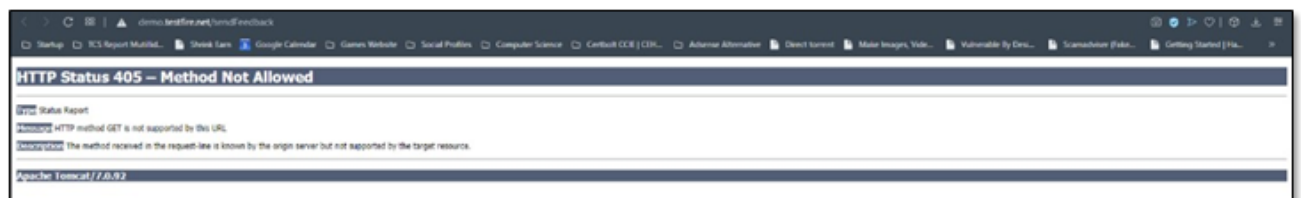
Here are the results I got about the vulnerabilities found in Mutillidae

➢ Mutillidae Automated Scan Using OWASP ZAP
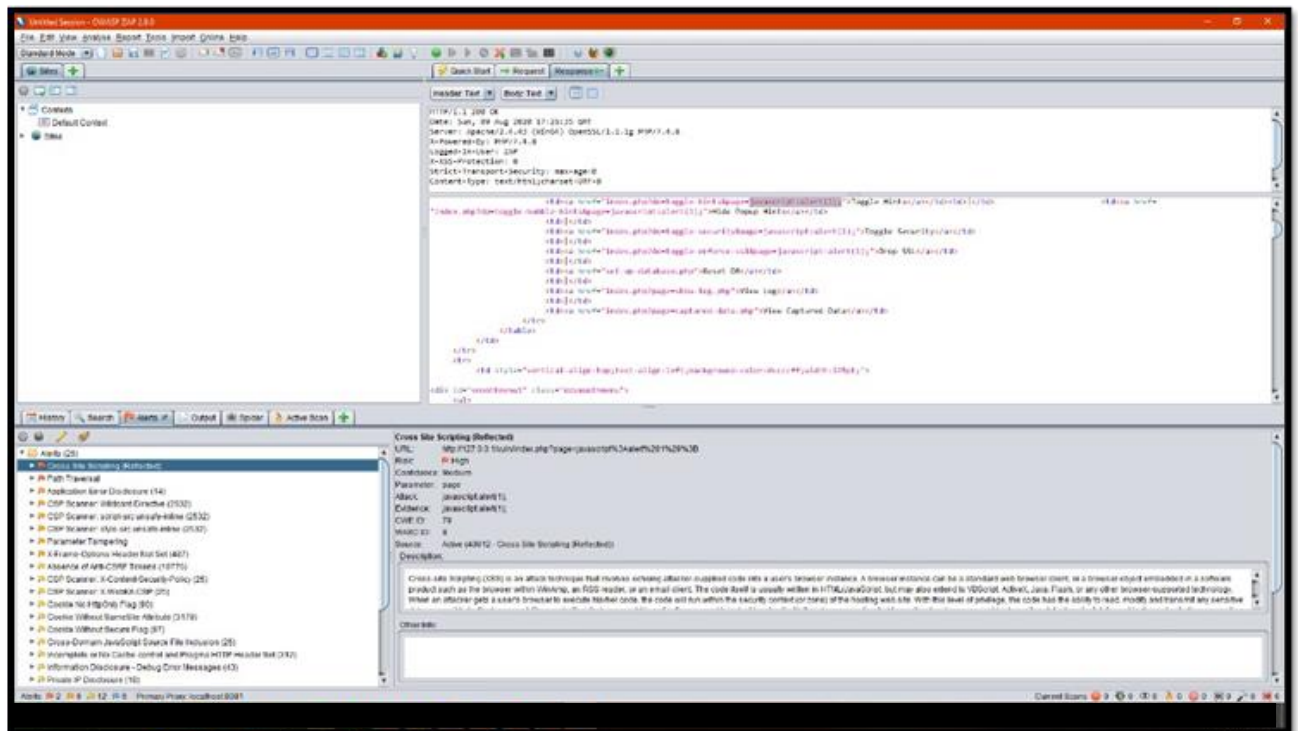
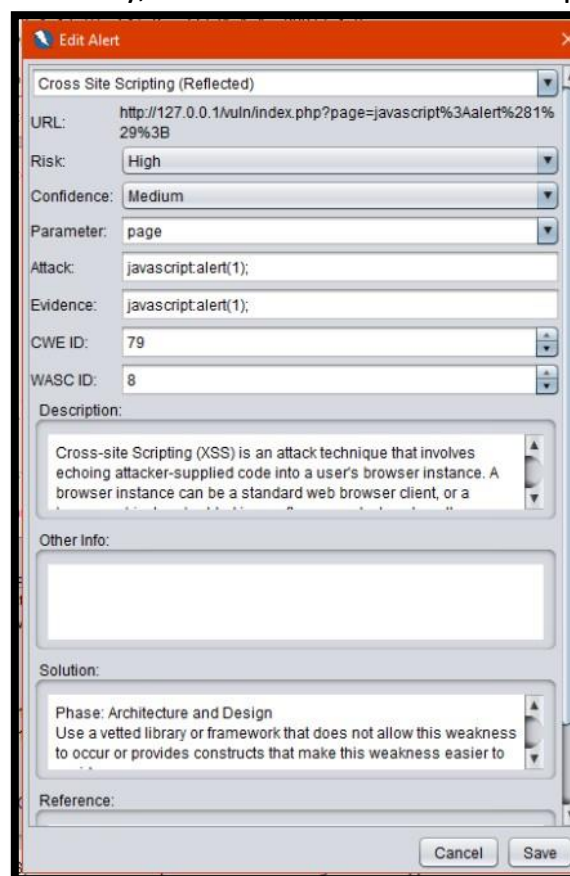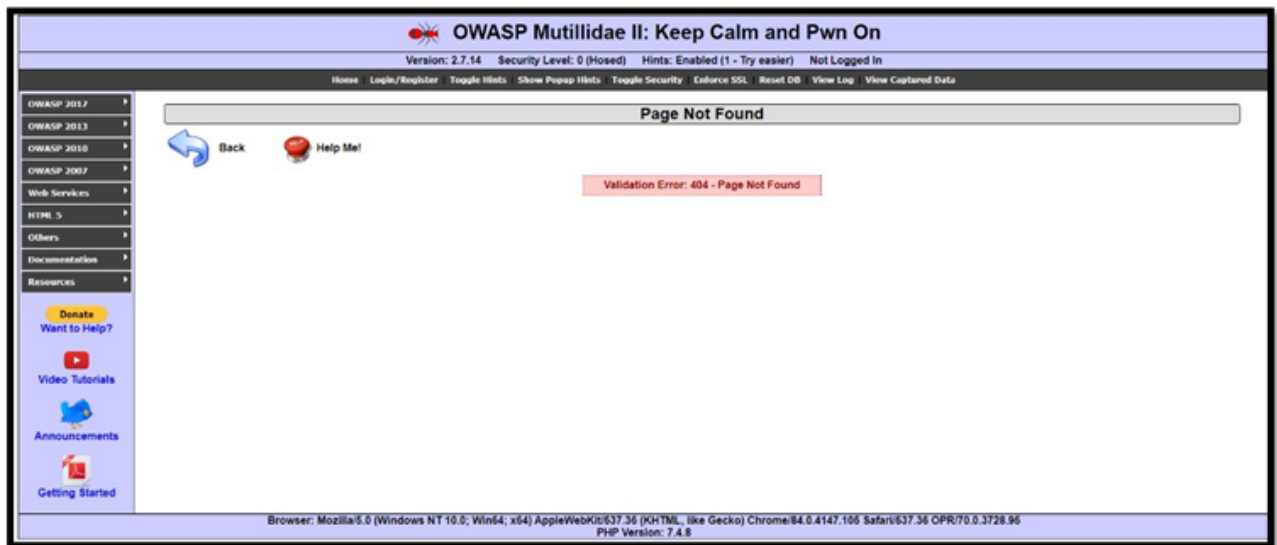**#** ZAP is confident with this vulnerability, but this doesn't work as expected. It

indicated false positive results.

While trying to exploit this reported vulnerability, we get an error.

Project Outcome:

After completion of this internship, we can state the outcome as.

The Mutillidae II has many vulnerabilities that we have to find deep inside, though we had list of vulnerabilities that Mutillidae has, but keeping in mind the black box nature of the project, we decided not to look out for any pre listed vulnerabilities listed by Mutillidae itself.

All vulnerabilities were searched by project members, one by one manually and then by automated testing tools. Throughout this project my mentors have helped me through active support and we never felt left alone.

We feel confident about Advanced Dynamic Application Security Testing, and all latest OWASP Top 10 vulnerabilities. We have researched and learnt the maximum about the following topics.

- SQL Injection

- XSS

- Broken Authentication

- Security Misconfiguration

- Sensitive Data Exposure

We have learnt about many types of vulnerabilities found in web applications, some of the most common of them are.

• Injection Attacks

• Broken Authentication Attacks

• Sensitive Data Exposure

• XML External Entities

• Broken Access Control

• Security misconfiguration

• Cross Site Scripting

• Insecure Deserialization

• Using Components with Known Vulnerabilities

• Insufficient Logging and Monitoring

demo.testfire.net (Altoro Mutual) was found to be highly vulnerable and at extreme risk, although it was made that way. We have got a sound understanding of top vulnerabilities found in web applications listed in OWASP Top 10 and have got an exposure to manual DAST techniques to find security defects.

We have got an expertise in using OWASP tool to find security defects rapidly and with comprehensive coverage. We can now create professional DAST report of all defects in covered in an insecure web-based application.

Steps followed by me in completion of the project include.

• Manual exploring of application.

• Looking for details about the application on public resources.

• Looking for information disclosure by the application itself.

• Try basic SQL queries.

• Use JavaScript queries in input fields and comment boxed to detect persistent and non-persistent XSS vulnerabilities.

- Use vulnerability scanners to scan for website vulnerabilities.

- Exploiting and reporting each vulnerability one by one.

Other Techniques and Skills that I Gained:

- Basic HTML knowledge for decoding the GET and POST types of forms and buttons.

- Basic JavaScript to check for cross site scripting.

- Basic PHP knowledge to figure out the way the form is connected to database.

- Basic SQL queries to check for possible injections and also to extract data from database after compromising security.

Enhancement Scope:

The Web Application Penetration testing project was perfectly carried out by the project members under the guidance of project mentor A.L.Sreedip sir.

High level SQL and PHP is a needed skill for white hat hackers, this is where we can enhance.

Conclusion

This pentesting report serves as a comprehensive repository of insights garnered from the security assessment conducted on the Altoro Mutual and Multillidae 2 platform. The intention is to facilitate a holistic understanding of web application vulnerabilities, encourage hands-on learning, and equip professionals with the knowledge to safeguard applications against potential threats. Through this report, we endeavour to contribute to the collective effort in fortifying web applications' security, ensuring a safer digital environment for users and organizations alike.