



## INTERNSHIP: PROJECT REPORT

Internship Project Title	TCS RIO-210 - Advanced Dynamic Application Security Testing to Find Defects in Web Applications
Project Title	TCS RIO-210 - Advanced Dynamic Application Security Testing to Find Defects in Web Applications
Name of the Company	TCSiON
Name of the Industry Mentor	Vineet Kannoly / Shreya Masurkar
Name of the Institute	Madhyanchal Professional University, Bhopal

**SUBMITTED BY :** Shaswat Manoj Jha

Start Date	End Date	Total Effort (hrs.)	Project Environment	Tools used
18th June '20	1st August '20	210	<b>Mutillidae and Altoro Mutual</b>	Burpsuite, SQLMap, ZAP.

 **Title of The Project:** Advanced Dynamic Application Security Testing to Find Defects in Web Applications  
This was a project on DAST, based on the Top 10 Vulnerabilities releases every few years via [OWASP.ORG](https://owasp.org).  
In this project I have learnt about various kinds of techniques that hackers use to compromise web applications and also learnt about how to secure them.

 **Learning Content:**  
Throughout this project my mentors have helped me through active support and I never felt left alone.  
I 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

In this internship I have learnt about OWASP Top 10.

OWASP represents the Open Web Application Security Project, an online network that produces articles, strategies, documentation, instruments, and innovations in the field of web application security.

OWASP Top 10 is the rundown of the 10 most popular web application weaknesses. It additionally shows their dangers, effects, and countermeasures. Refreshed each three to four years, the most recent OWASP weaknesses list was discharged in 2018.

During this internship I also found out about the contrasts among DAST and SAST. I am confident about how the SAST and DAST types of testing works.

**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.

### **Solution Approach:**

Mutillidae II and demo.testfire.net are web applications that are purposefully made vulnerable to OWASP Vulnerabilities.

My solution approach was simple, I assumed that I don't know these applications are vulnerable.

I started exploring these applications one by one manually. And as I proceeded further, I found new vulnerabilities which have already been listed by me in the activity report, submitted daily throughout the project.

As, in real world I will never get the vulnerability information from the organization.

The internship, was carried out in 3 phases.

Which I have elaborated below.

1. In **"Phase 1"** of the internship project, I was manually able to find some vulnerabilities related to the following.
  - SQL Injection
  - XSS
  - Broken Authentication
  - Broken Access Control
  - Security Misconfiguration
  - Sensitive Data Exposure
  - Known Vulnerabilities that applications use.
2. In **"Phase II"** of the internship I used OWASP ZAP, Burp Suite etc tools to do automated testing and found new SQL, XSS and Security Misconfiguration vulnerabilities.
3. In **"Phase III"**, I searched for additional vulnerabilities listed below.
  - Broken Authentication
  - Broken Access Control
  - Sensitive Data Exposure.

## INTERNSHIP: PROJECT REPORT

---

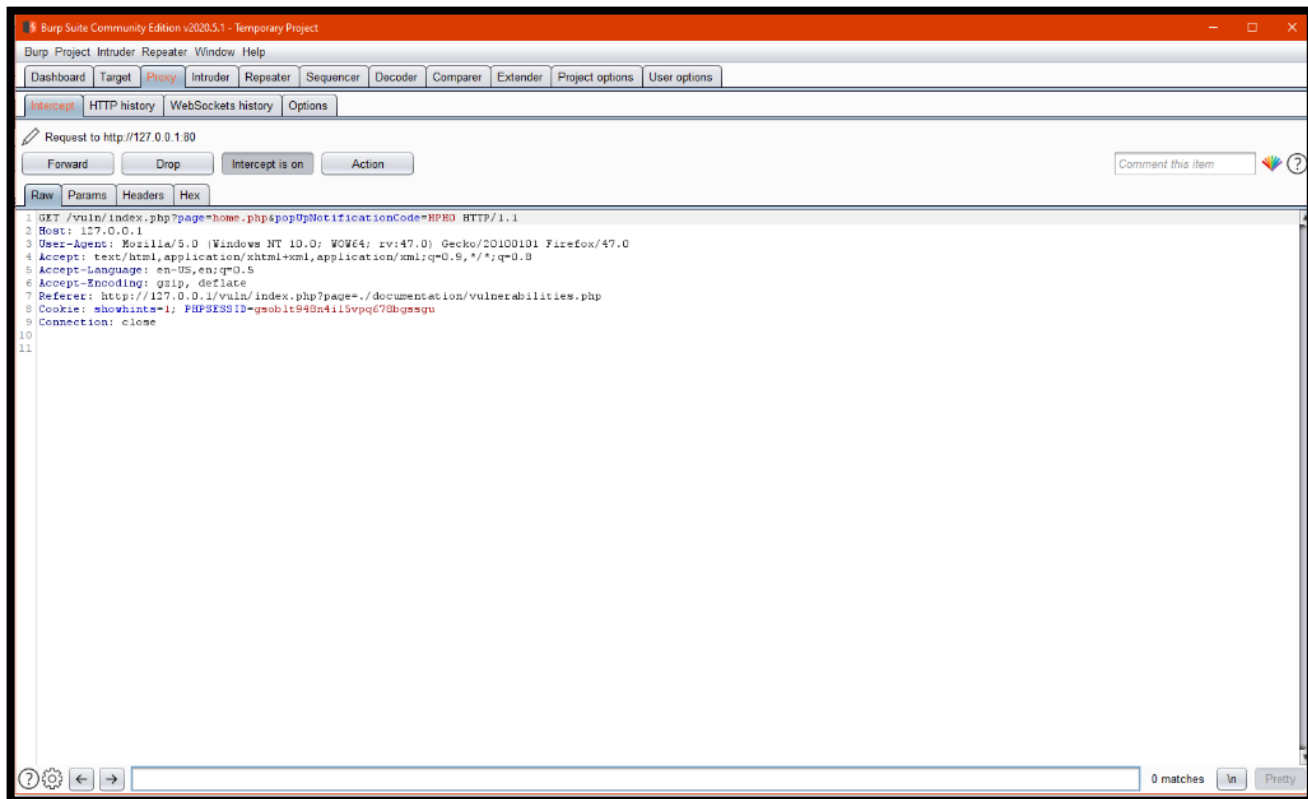
My aim was to know about how exactly malicious hackers exploit vulnerabilities and how can we as white hats secure them. The second aim was to get familiar with all types of vulnerabilities that web apps can have. During this project I have used the following tools most frequently.

### Testing Environment:

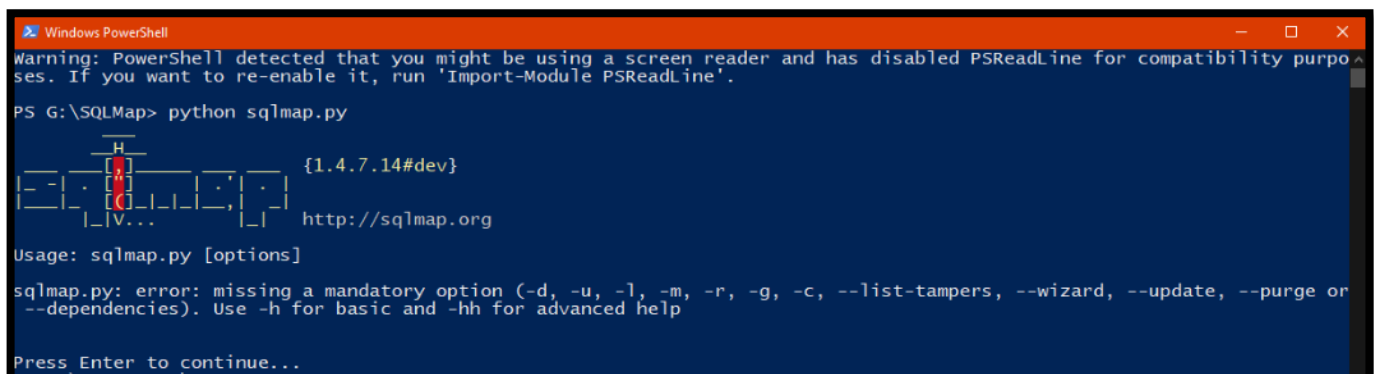
1. Altoro Mutual [demo.testfire.net]
2. Mutillidae II [Self Hosted]

### Tools Used:

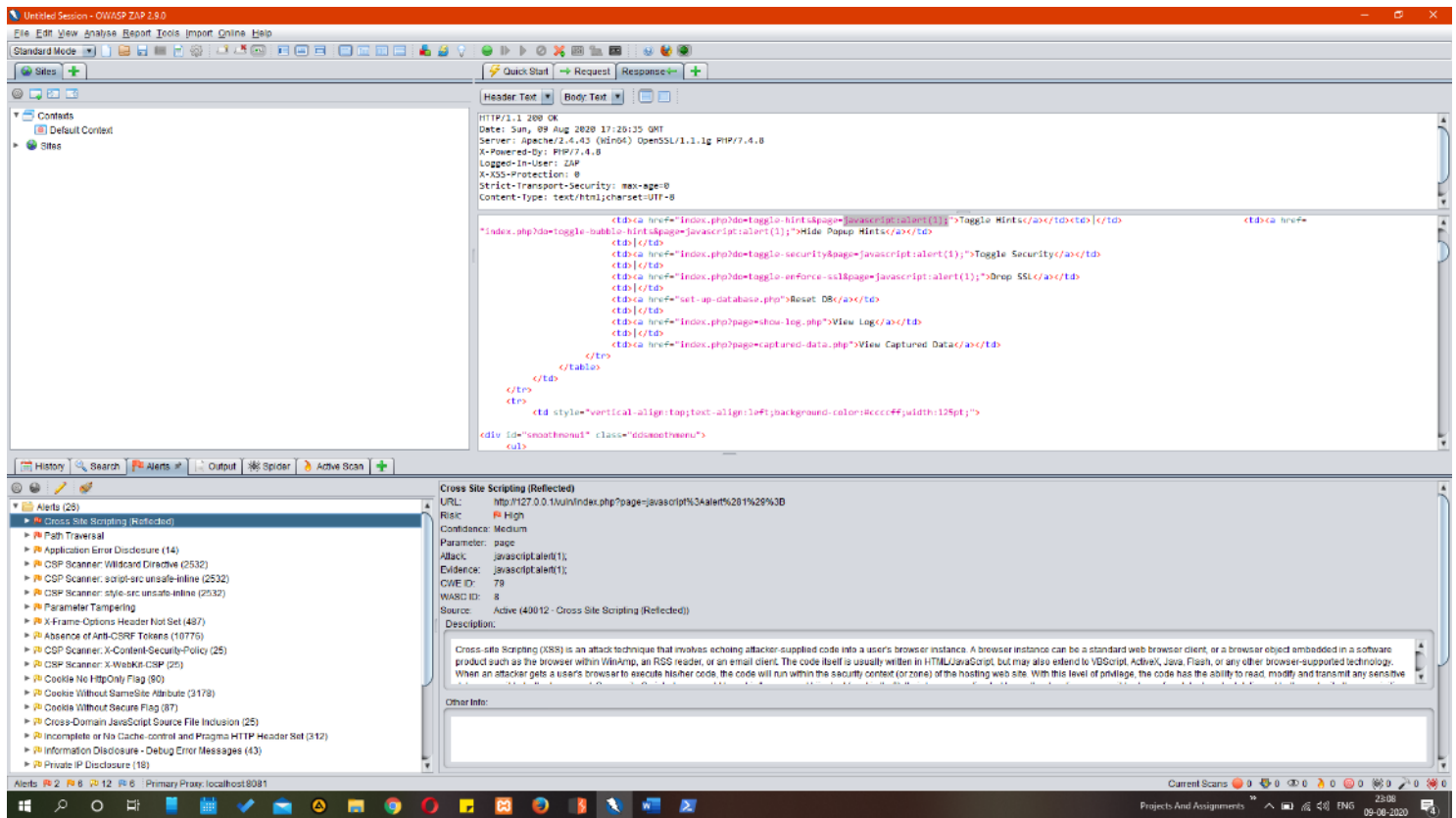
#### 1. Burp Suite



#### 2. SQL Map



## 3. OWASP Zap Vulnerability Scanner



### 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.

## INTERNSHIP: PROJECT REPORT

---

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.

### (c) Authoritative reference links, and explanatory examples

- W3Schools – SQL Injection.
- Mathew Setter – How to intercept request and modify response in Burp Suite.

### [#1]

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.)

A screenshot of a web form titled 'Online Banking Login'. It contains two input fields: 'Username:' with the value ' Or '1'='1' and 'Password:' with a masked password represented by dots. Below the password field is a 'Login' button.A screenshot of a web page titled 'Hello Admin User'. It displays a welcome message 'Welcome to Altoro Mutual Online.' and a 'View Account Details:' section with a dropdown menu showing '800000 Corporate' and a 'GO' button. Below this, it says 'Congratulations!' and provides information about a pre-approved Altoro Gold Visa with a credit limit of €8799.02!, along with a link to apply.

### [#2]

Location / URL – Mutillidae -> OWASP 2013 -> A1 - 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.

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

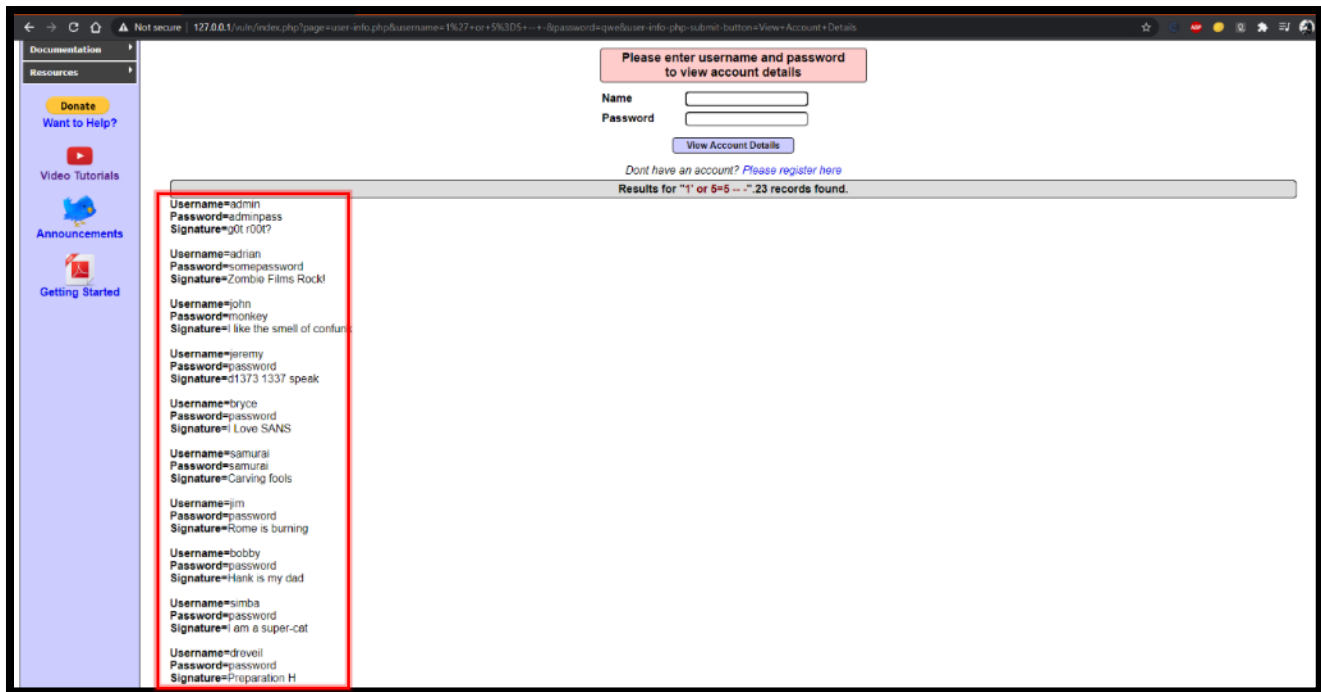
#### Entered random input to get an SQL error.

The screenshot shows the 'User Lookup (SQL)' page of the Mutillidae application. It features a login form with fields for 'Name' and 'Password', and a 'View Account Details' button. Below the form is an 'Error Message' section. The error message is a red box containing the following text:

```
Failure is always an option
Line 229
Code 0
File G:\Ideal\htdocs\vu\classes\MySQLHandler.php
Message connect_error: 0
error: 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 'password' at line 2
client_info: MySQLnd 7.4.8
host_info: 127.0.0.1 via TCP/IP
} Query: SELECT * FROM accounts WHERE username='sheswet' AND password='password' () [Exception]
Trace #0 G:\Ideal\htdocs\vu\classes\MySQLHandler.php(315): MySQLHandler->doExecuteQuery('SELECT * FROM a...') #1 G:\Ideal\htdocs\vu\classes\SQLQueryHandler.php(358): MySQLHandler->executeQuery('SELECT * FROM a...') #2
G:\Ideal\htdocs\vu\user-info.php(191): SQLQueryHandler->getUserAccount('sheswet', 'password') #3 G:\Ideal\htdocs\vu\index.php(637): require_once('G:\Ideal\htdocs...') #4 (main)
Diagnostic Information Error attempting to display user information
Click here to reset the DB
```

The error message is highlighted with a red box. The bottom of the page shows the browser and PHP version information: 'Browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.105 Safari/537.36' and 'PHP Version: 7.4.8'.

- Using the payload in step 2 we can fetch all the usernames and passwords.



### [#3]

Location / URL – OWASP Top 10 --> A1 - SQL Injection --> SQLi - Extract Data --> User Info

#### Steps and Description.

- We type 'union select null – just to check what kind of error we get.

#### Error Message

Failure is always an option	
Line	229
Code	0
File	G:\Ideal\htdocs\vuln\classes\MySQLHandler.php
Message	<pre>G:\Ideal\htdocs\vuln\classes\MySQLHandler.php on line 224: Error executing query: connect_errno: 0 errno: 1222 error: The used SELECT statements have a different number of columns client_info: mysqlnd 7.4.8 host_info: 127.0.0.1 via TCP/IP  ) Query: SELECT * FROM accounts WHERE username='' union select null -- ' AND password='' (0) [Exception]</pre>
Trace	<pre>#0 G:\Ideal\htdocs\vuln\classes\MySQLHandler.php(315): MySQLHandler-&gt;doExecuteQuery('SELECT * FROM a...') #1 G:\Ideal\htdocs\vuln\clas: G:\Ideal\htdocs\vuln\user-info.php(191): SQLQueryHandler-&gt;getUserAccount('' union select ...', '') #3 G:\Ideal\htdocs\vuln\index.php(6:</pre>
Diagnostic Information	Error attempting to display user information
<a href="#">Click here to reset the DB</a>	

## INTERNSHIP: PROJECT REPORT

---

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=7746536337776330
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

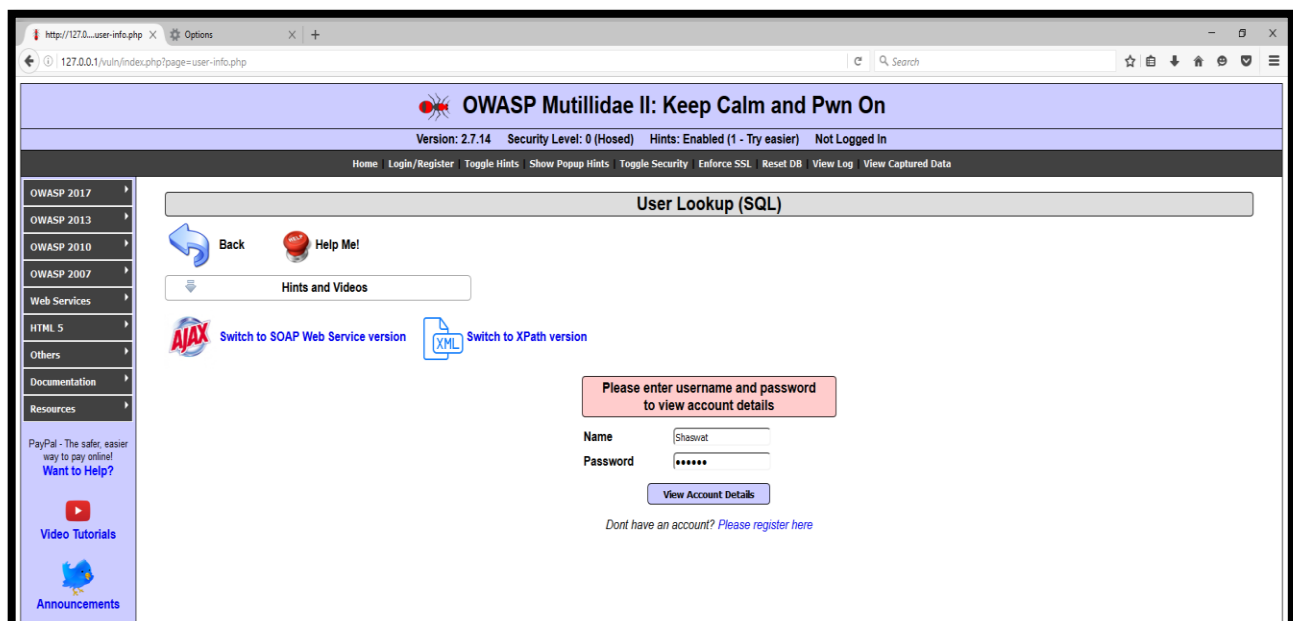
### [#3]

Location / URL – OWASP Top 10 --> A1 - SQL Injection --> SQLi - Extract Data --> User Info

#### Steps and Description.

SQL Injection with SQLMap through Burp Suite.

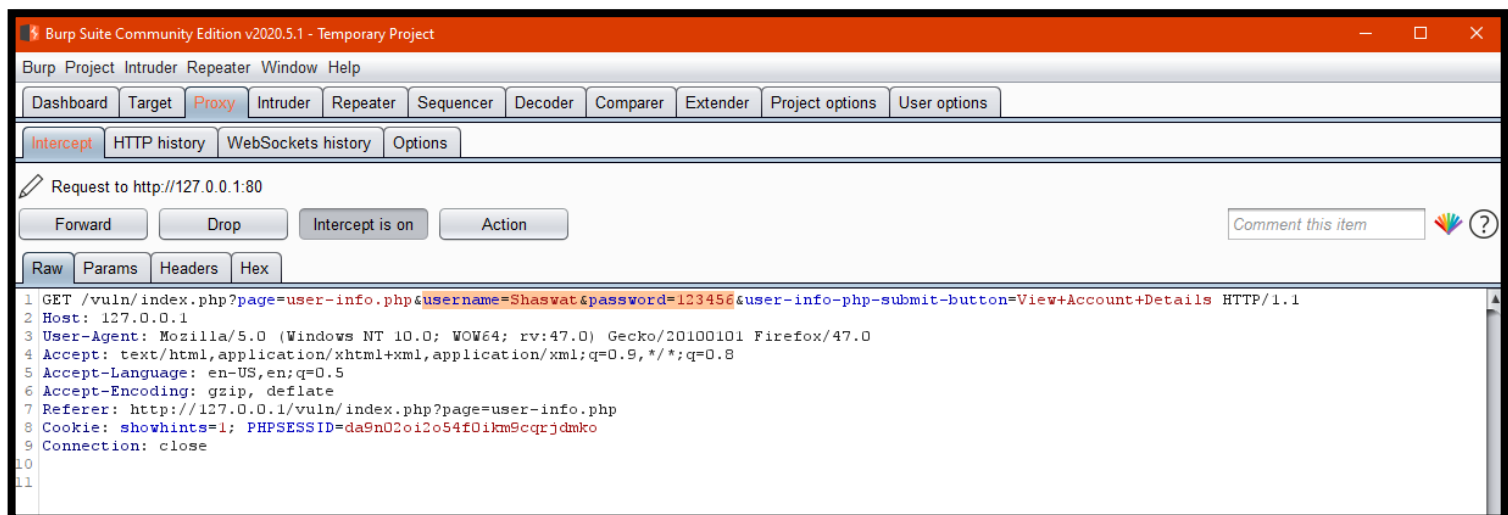
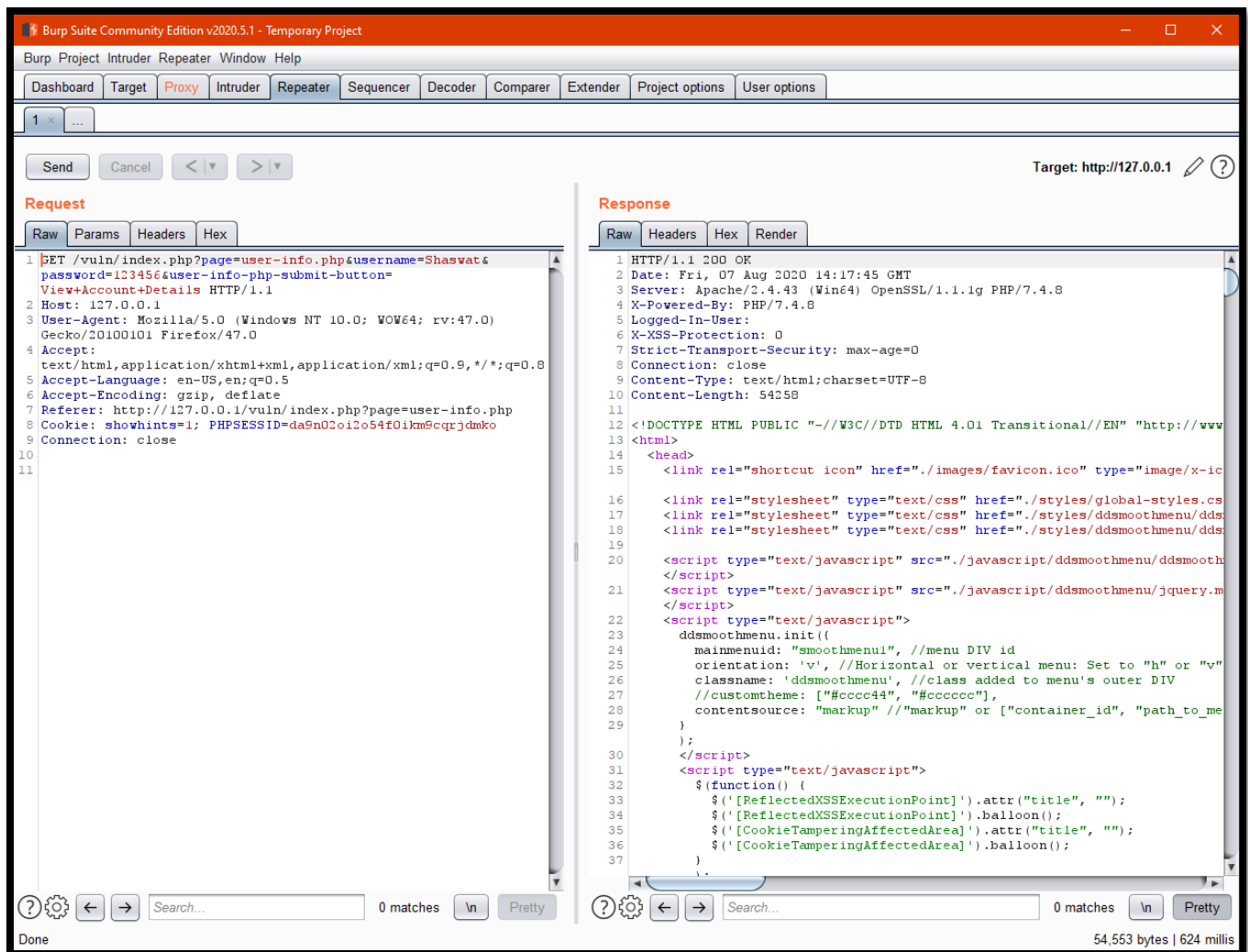
1. On the User Info page, we capture a normal request using Burp Suite.





## INTERNSHIP: PROJECT REPORT

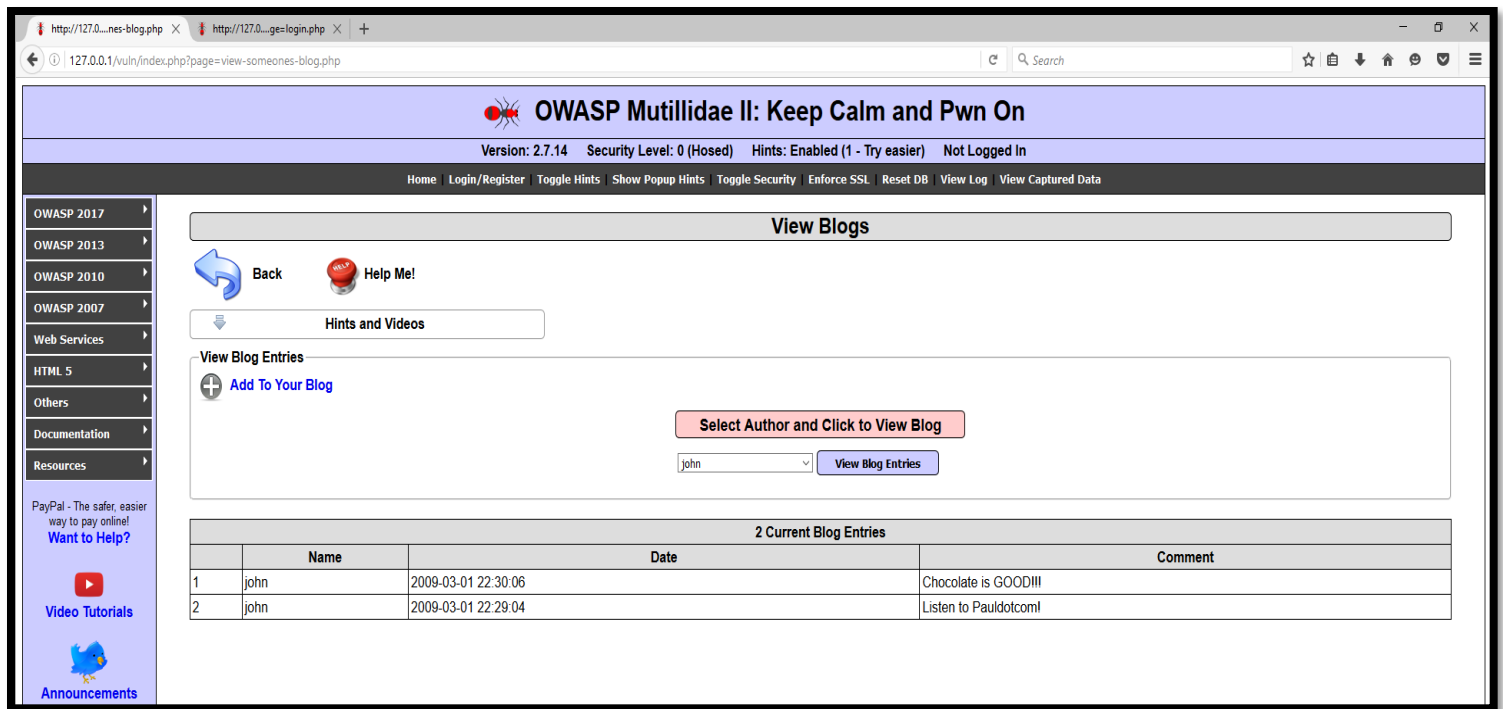
2. We take the request and forward it to the repeater.



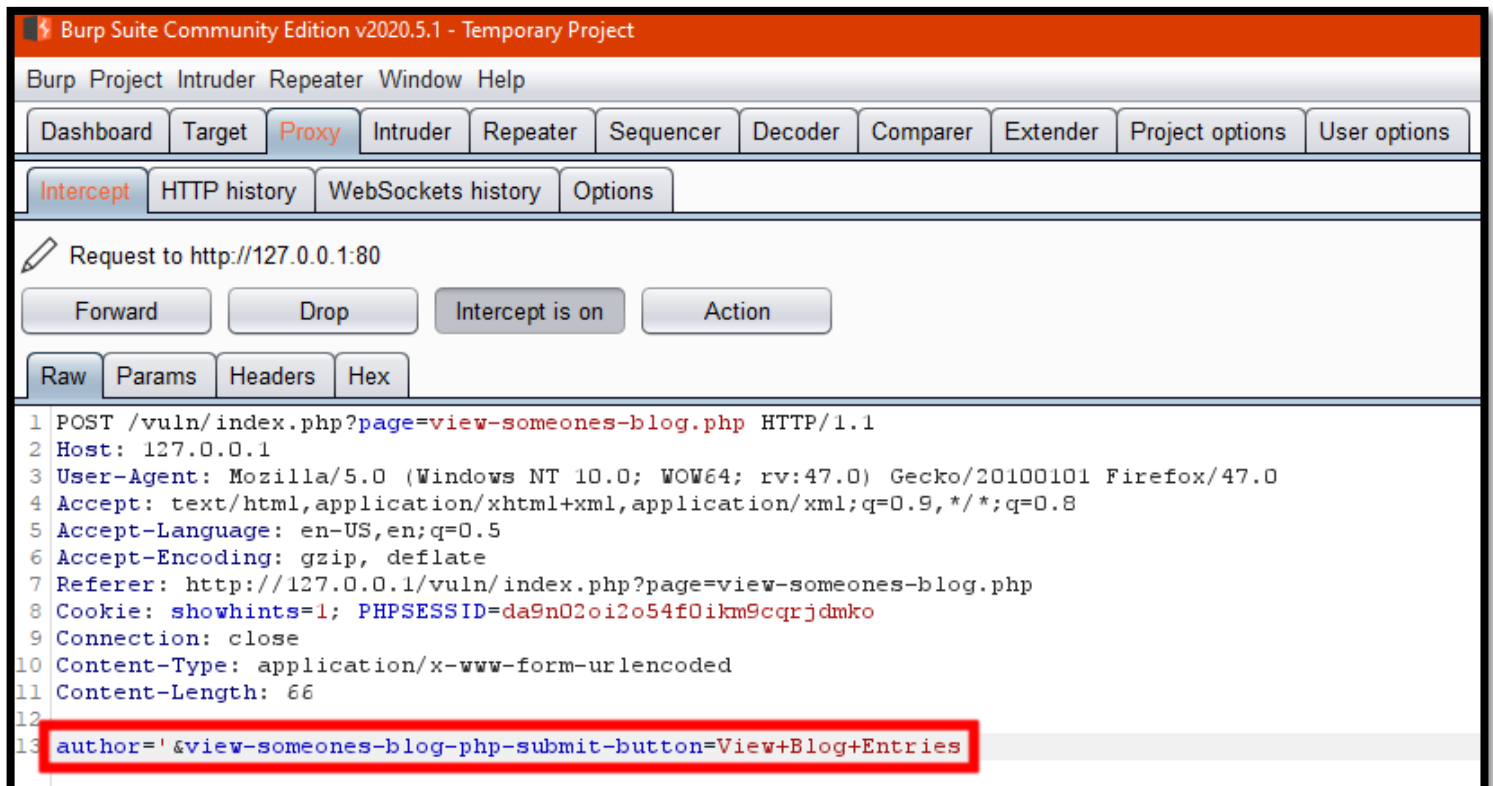


## INTERNSHIP: PROJECT REPORT

And intercept that request using Burp Suite.



2. Add ' in place of user to check for SQLi Vulnerability.



## INTERNSHIP: PROJECT REPORT

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

The screenshot shows the OWASP Mutillidae II web application interface. The browser address bar displays the URL `http://127.0.0.1/vuln/index.php?page=view-someones-blog.php`. The application header includes the title "OWASP Mutillidae II: Keep Calm and Pwn On" and navigation links such as "Home", "Login/Register", "Toggle Hints", "Show Popup Hints", "Toggle Security", "Enforce SSL", "Reset DB", "View Log", and "View Captured Data".

The main content area is titled "View Blogs" and contains a "View Blog Entries" section with a button labeled "Add To Your Blog". Below this, a notice states: "Notice: Undefined variable: IQueryString in G:\Ideall\htdocs\vuln\view-someones-blog.php on line 238".

An "Error Message" section displays the following details:

Line	Code	File	Message
229	0	G:\Ideall\htdocs\vuln\classes\MySQLHandler.php	connect_error: 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 7.0.6 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]

The "Trace" section shows the following stack trace:

```
#0 G:\Ideall\htdocs\vuln\classes\MySQLHandler.php(315): MySQLHandler->doExecuteQuery('')  
#1 G:\Ideall\htdocs\vuln\classes\SQLQueryHandler.php(197): MySQLHandler->executeQuery('')  
#2 G:\Ideall\htdocs\vuln\view-someones-blog.php(174): SQLQueryHandler->getBlogRecords('')  
#3 G:\Ideall\htdocs\vuln\index.php(437): require_once('G:\Ideall\htdocs...')  
#4 [main]
```

The browser status bar at the bottom indicates: "Browser: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0".

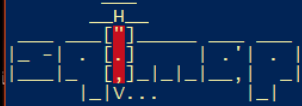
4. We can see the SQL Query used. [The last sentence in Message field]

## INTERNSHIP: PROJECT REPORT

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\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	
<a href="#">Click here to reset the DB</a>	

5. Save the request from Burp Suite HTTP history, to a file. And run SQLMap, and try to grab the database using `--dbs` command.

```
PS G:\SQLMap> python sqlmap.py -r G:\SQLMap\1.request --banner --threads=2 --dbs
```



```
{1.4.7.14#dev}
http://sqlmap.org
```

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

6. We get a list of databases when the scan finishes.

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

## 4. Cross Site Scripting [XSS]

(a) Risk/ Undesirable impact if exploited.

## INTERNSHIP: PROJECT REPORT

---

**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

**(c) Authoritative reference links, and explanatory examples.**

- [Wikipedia](#)
- [Owasp.org](#)
- [Portswigger](#)

**[#1]**

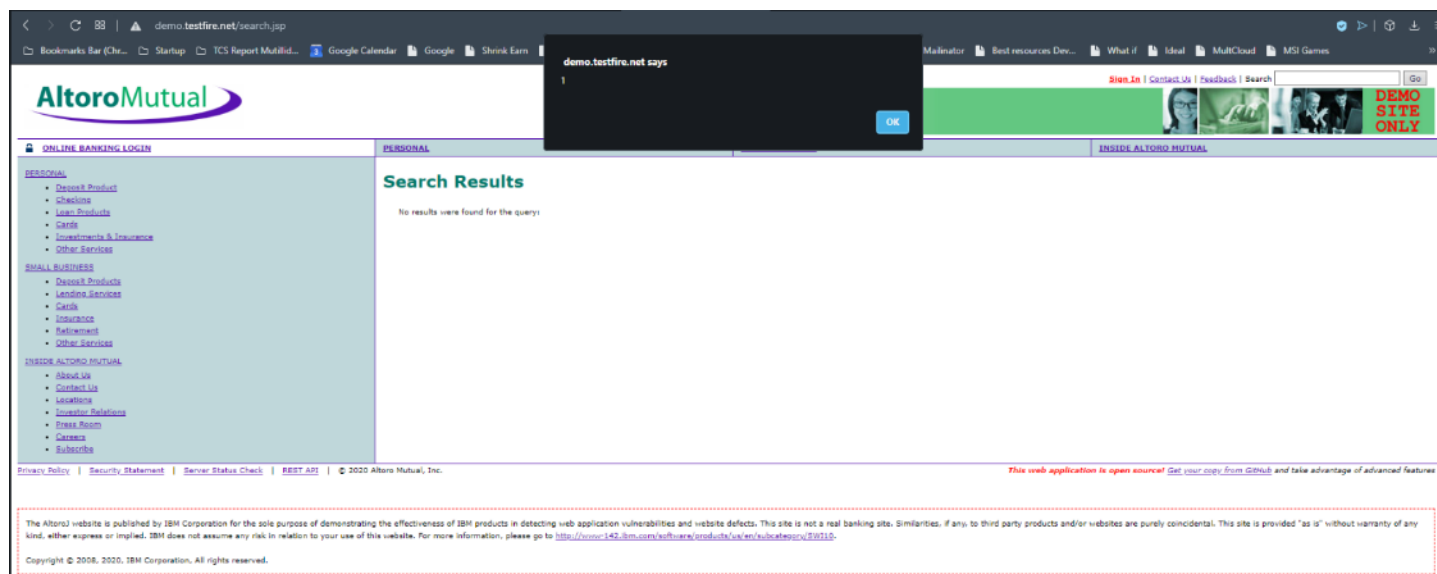
**Location / URL – demo.testfire.net/search.jsp**

## INTERNSHIP: PROJECT REPORT

### 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.



## [#2]

Location / URL – OWASP Top 10 --> A2 - Cross Site Scripting(XSS) --> Persistent(Second Order) --> Add to your blog

### Steps and Description.

This is a persistent XSS in Mutillidae. 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.

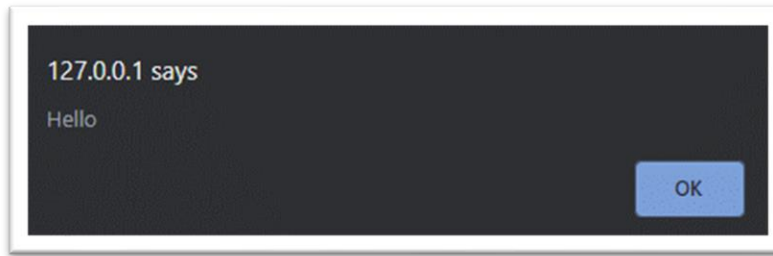
We get the output as programmed. One thing to notice is that whenever this page is refreshed the code runs again, this shows that the injected code has been embedded in to the source of the web application.



## INTERNSHIP: PROJECT REPORT

---

Hence making it a stored / persistent XSS vulnerability.



### [#3]

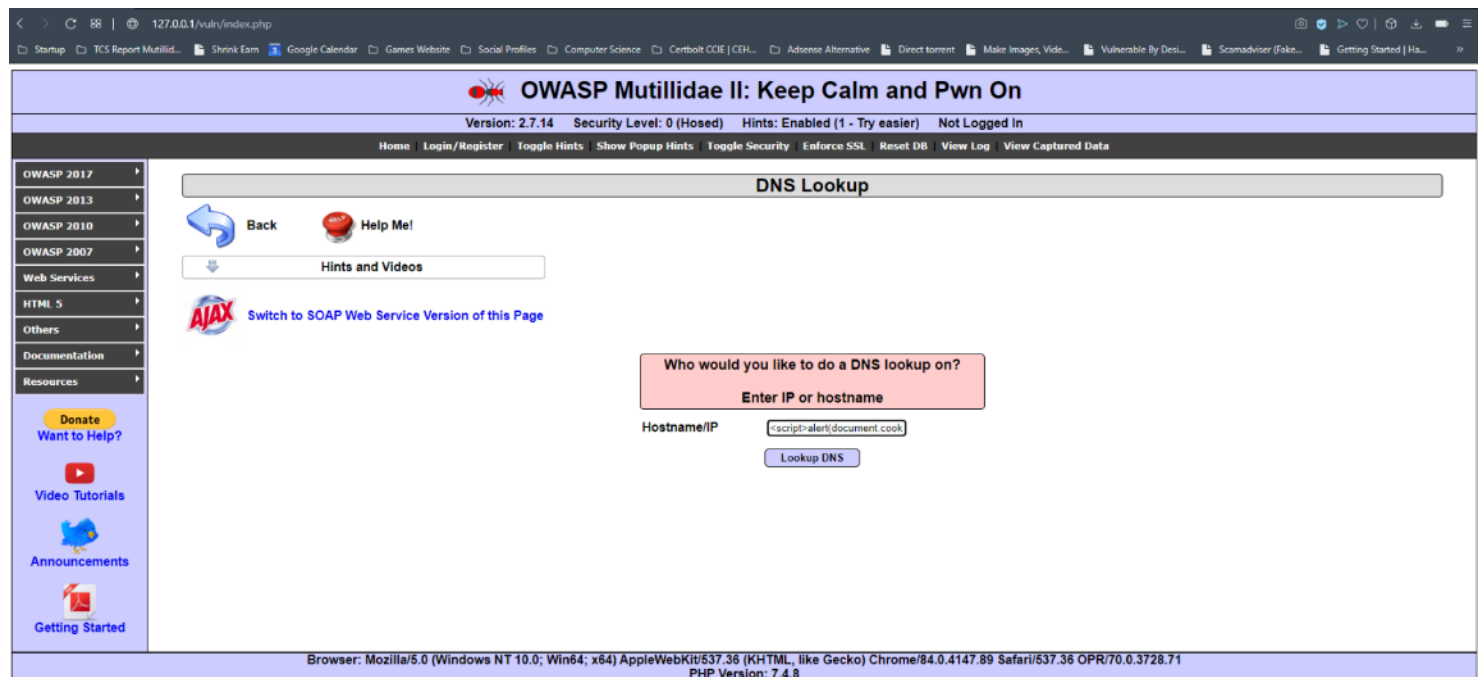
**Location / URL** – OWASP Top 10 --> A2 - Cross Site Scripting (XSS) --> Reflected (First Order) --> **DNS Lookup**

#### Steps and Description.

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

The goal here is to determine:

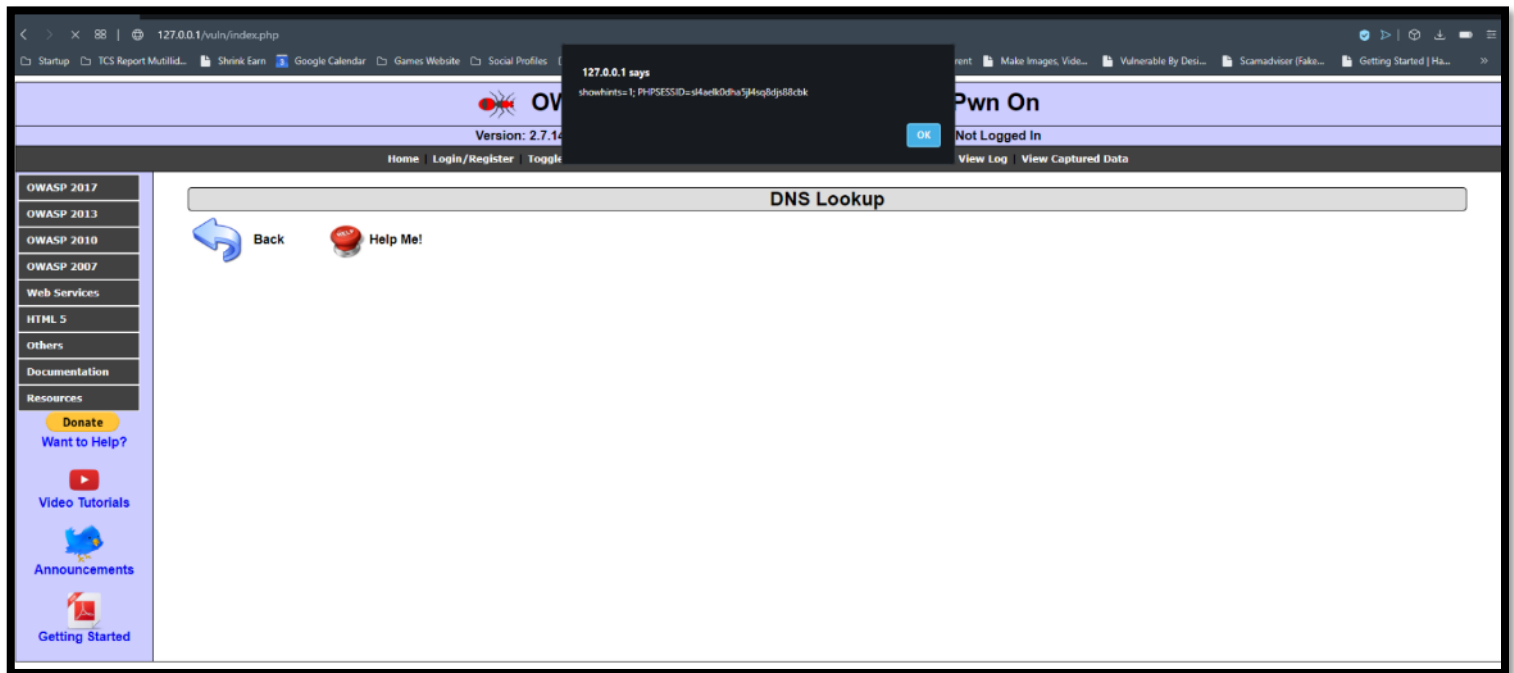
1. if this webpage contains a cookie.
2. if we can display the cookie in a JavaScript alert box.



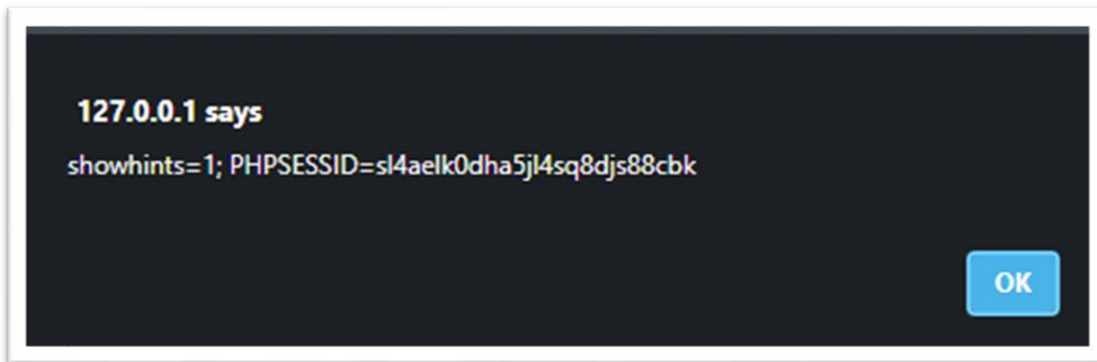
## INTERNSHIP: PROJECT REPORT

---

We get the following output.



We can clearly see the cookie displays the username and the PHP Session ID.



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

### 4. 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.

## INTERNSHIP: PROJECT REPORT

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.

### (c) Authoritative reference links, and explanatory examples.



- [wiki.net](https://www.wiki.net)
- [hdivsecurity](https://hdivsecurity.com)

## [#1]

**Location / URL** – <http://127.0.0.1/vuln/index.php?page=phpinfo.php>

### Steps and Description.

Mutillidae 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 MRSASH_PC 10.0 build 19041 (Windows 10) AMD64
Build Date	Jul 9 2020 11:24:06
Compiler	Visual C++ 2017
Architecture	x64
Configure Command	csript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\64instantclient_12_1sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\64instantclient_12_1sdk,shared" "--enable-object-out-dir=.obj" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgsql"
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	
	

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.

## INTERNSHIP: PROJECT REPORT

### [#2]

**Location / URL** – [http://demo.testfire.net/index.jsp?content=inside\\_jobs.htm](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.

**AltoroMutual**

Sign In | Contact Us | Feedback | Search  Go

**ONLINE BANKING LOGIN**

**PERSONAL**

- Deposit Products
- Checking
- Loan Products
- Cards
- Investments & Insurance
- Other Services

**SMALL BUSINESS**

- Deposit Products
- Lending Services
- Cards
- Insurance
- Retirement
- Other Services

**INSIDE ALTORO MUTUAL**

- About Us
- Contact Us
- Locations
- Investor Relations
- Press Room
- Careers
- Subscribe

### Current Job Openings

We update our job database daily so that you can find the most up-to-date career opportunities within Altoro Mutual.

Group	Date Posted	Title
Administration	Oct-23-2006	<a href="#">Executive Assistant</a>
Consumer Banking	Oct-19-2006	<a href="#">Teller</a>
Customer Service	Oct-26-2006	<a href="#">Customer Service Representative</a>
Marketing	Oct-25-2006	<a href="#">Locality Marketing Program Manager</a>
Risk Management	Oct-17-2006	<a href="#">Operational Risk Manager</a>
Sales	Oct-24-2006	<a href="#">Mortgage Lending Account Executive</a>

Altoro Mutual and its affiliates recruit and hire qualified candidates without regard to race, religion, color, sex, sexual orientation, age, national origin, ancestry, citizenship, veteran or disability status or any factor prohibited by law, and as such affirms its policy and practice to support and promote the concept of equal employment opportunity and affirmative action, in accordance with all applicable federal, state and municipal laws. Candidates must possess the right to work in the United States, as it is not the practice of Altoro Mutual to sponsor individuals for work visas.

[Privacy Policy](#) | [Security Statement](#) | [Server Status Check](#) | [REST API](#) | © 2020 Altoro Mutual, Inc.

*This web application is open source! Get your copy from [GitHub](#) and take advantage of advanced features*

The Altoro website is published by IBM Corporation for the sole purpose of demonstrating the effectiveness of IBM products in detecting web application vulnerabilities and website defects. This site is not a real banking site. Similarities, if any, to third party products and/or websites are purely coincidental. This site is provided "as is" without warranty of any kind, either express or implied. IBM does not assume any risk in relation to your use of this website. For more information, please go to <http://www-167.ibm.com/software/products/us/en/subcategory/89710>.

Copyright © 2008-2020, IBM Corporation. All rights reserved.

### [#3]

**Location / URL** – OWASP 2017 -> A6 - Security Misconfiguration -> **Unrestricted File Upload**

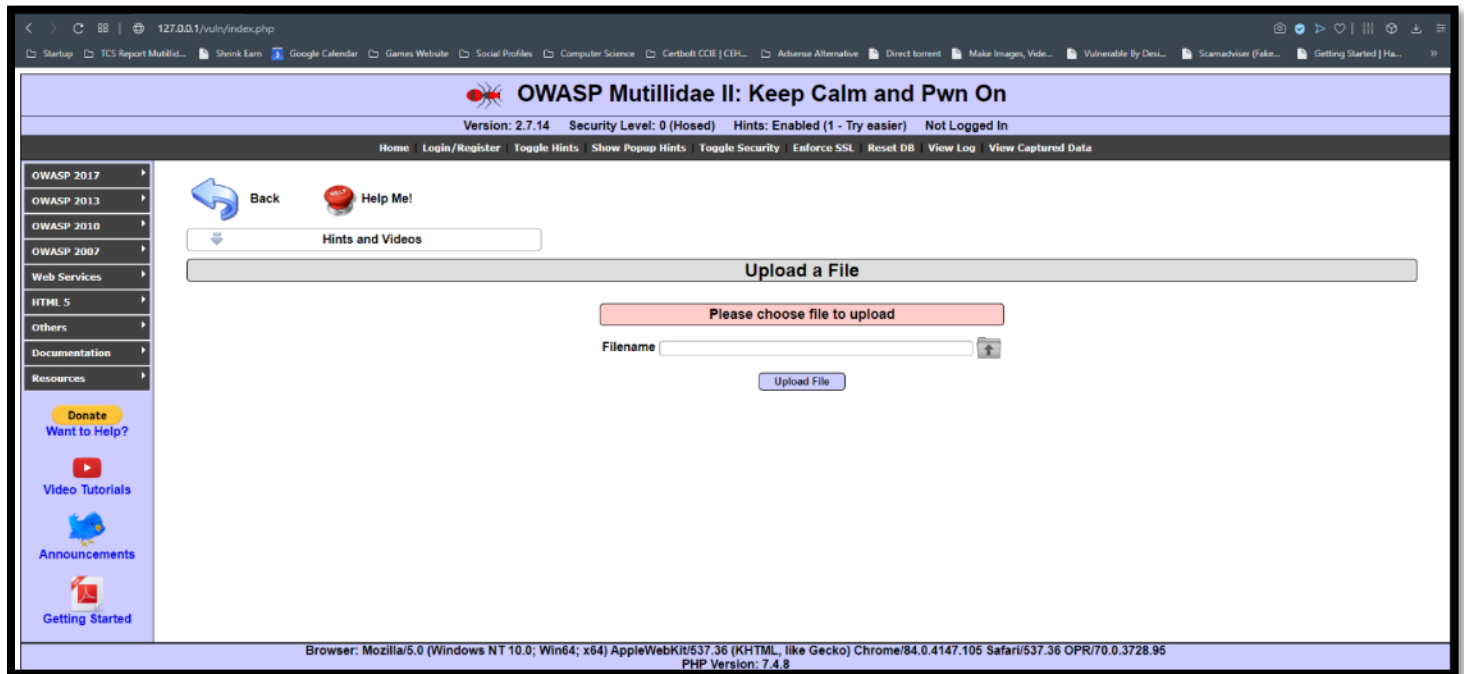
#### Steps and Description.

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.

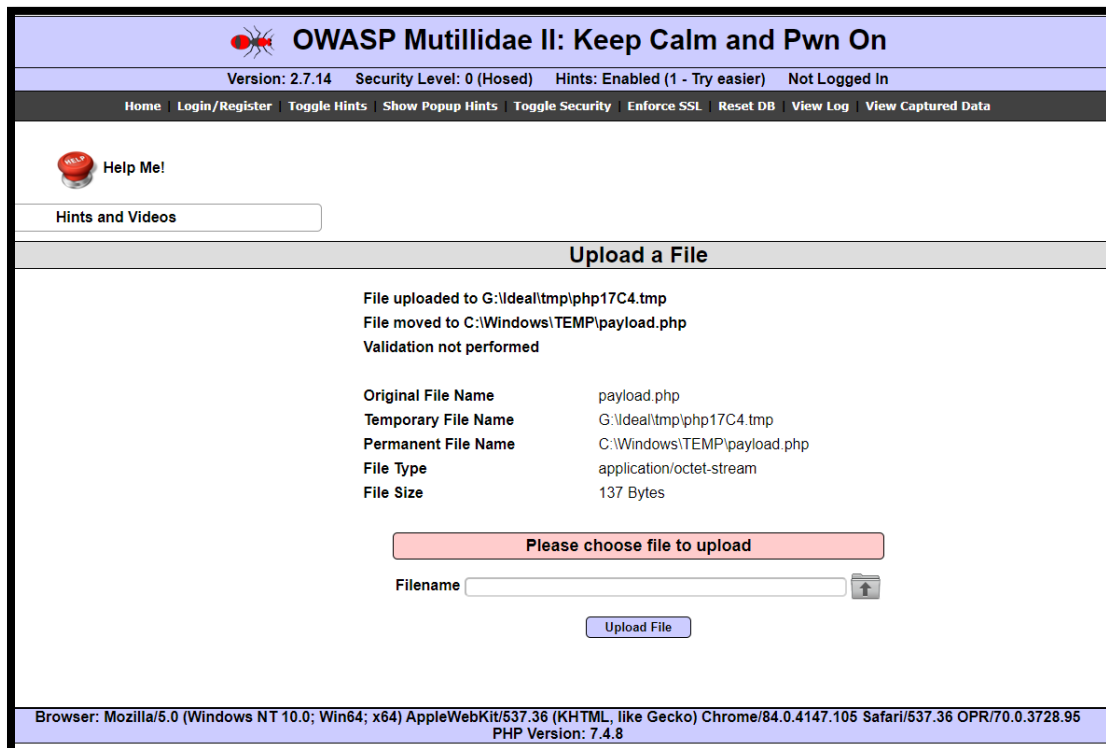
Also, an attacker can

- 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.
- etc.

## INTERNSHIP: PROJECT REPORT



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.

Upload a File

Please choose file to upload

Filename

Upload File

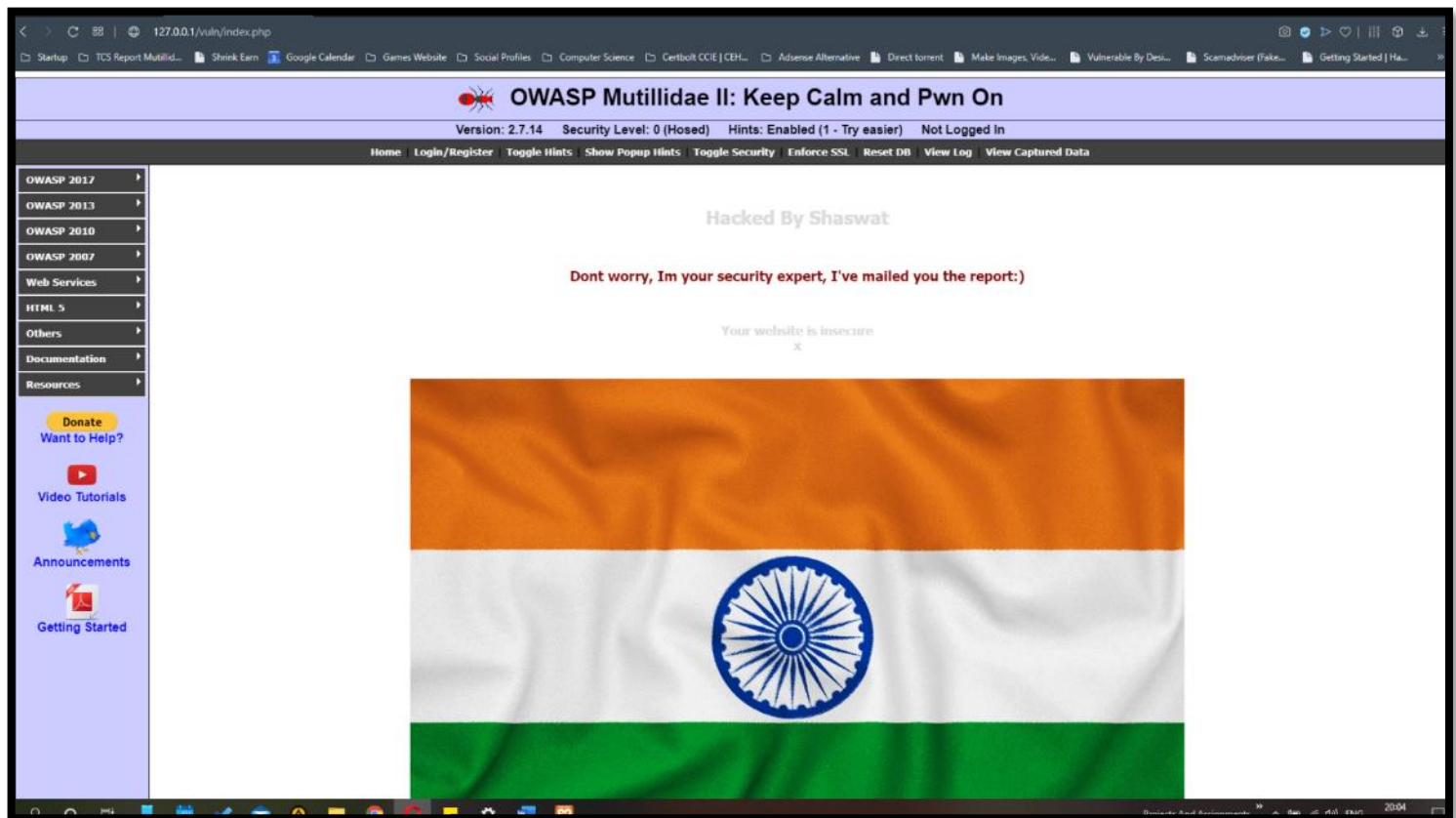
We can navigate to the file on the server.

Upload a File

File uploaded to G:\Ideal\tmp\php501A.tmp  
File moved to C:\Windows\TEMP\payload.html  
Validation not performed

Original File Name	payload.html
Temporary File Name	G:\Ideal\tmp\php501A.tmp
Permanent File Name	C:\Windows\TEMP\payload.html
File Type	text/html
File Size	1 KB

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.



## 5. Broken Authentication

### (a) Risk/ Undesirable impact if exploited.

Broken authentication is #2 on the most recent (2017) 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.

(c) Authoritative reference links, and explanatory examples.

- [tutorials point](#)
- [hdivsecurity](#)

### [#1]

Location / URL – Mutillidae -> OWASP 2017 -> /index.php?page=register.php

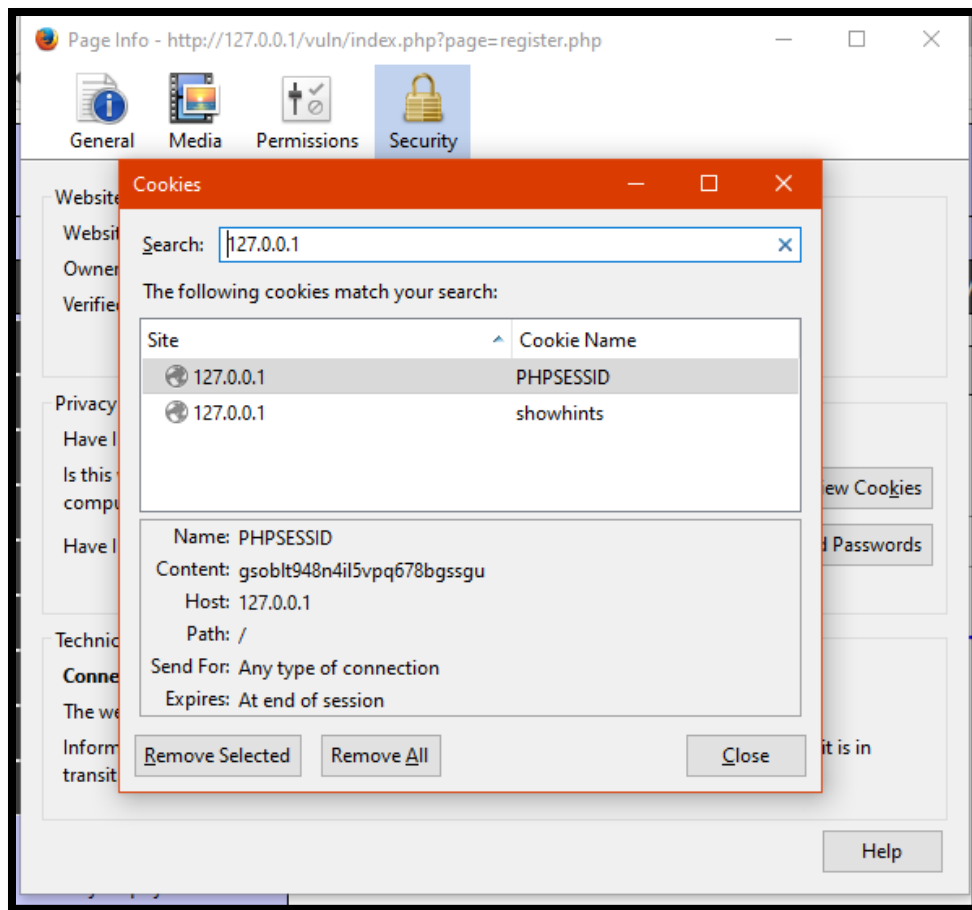
#### Steps and Description.

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

I will try to bypass authentication via cookie manipulation.

The screenshot shows a web browser window displaying the OWASP Mutillidae II registration page. The browser's address bar shows the URL `http://127.0.0.1/vuln/index.php?page=register.php`. The page has a purple header with the title "OWASP Mutillidae II: Keep Calm and Pwn On" and a navigation bar with links like "Home", "Login/Register", "Toggle Hints", etc. A left sidebar contains a menu with categories like "OWASP 2017", "OWASP 2013", "OWASP 2010", "OWASP 2007", "Web Services", "HTML 5", "Others", "Documentation", and "Resources". The main content area is titled "Register for an Account" and includes a "Back" button, a "Help Me!" button, a "Hints and Videos" dropdown, and an "AJAX Switch to RESTful Web Service Version of this Page" link. The registration form itself is titled "Please choose your username, password and signature" and contains input fields for "Username", "Password", "Confirm Password", and "Signature". A "Password Generator" link is next to the password field. A "Create Account" button is at the bottom of the form. The footer of the page displays browser and server information: "Browser: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:47.0) Gecko/20100101 Firefox/47.0" and "PHP Version: 7.4.8".

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.

Please choose your username, password and signature

Username

Password  [Password Generator](#)

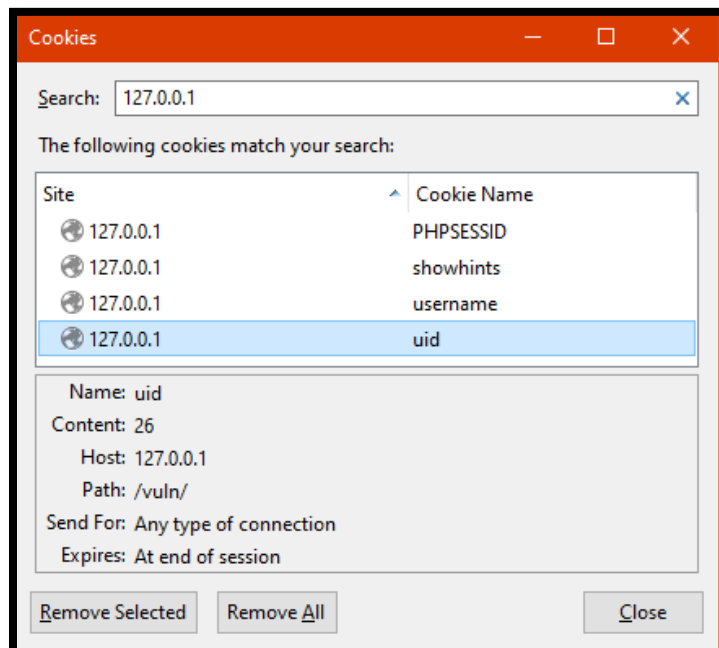
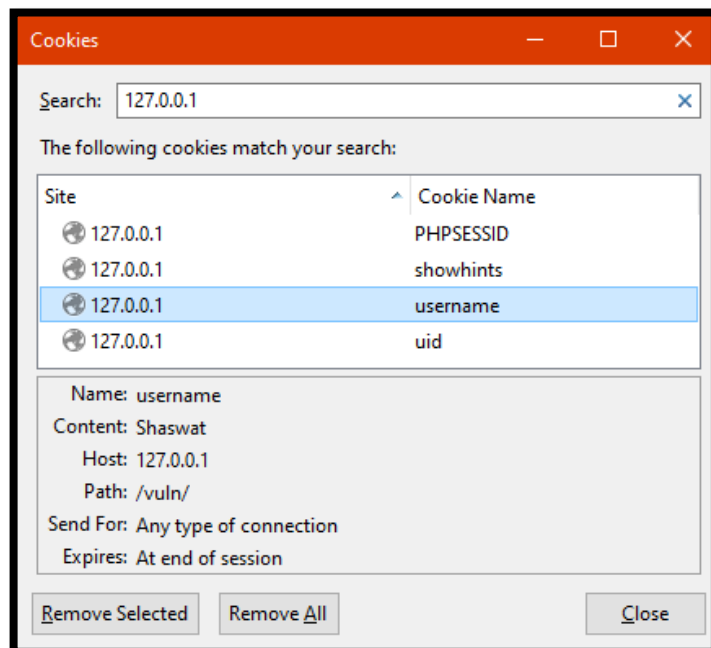
Confirm Password

Signature

[Create Account](#)

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.

## [#2]

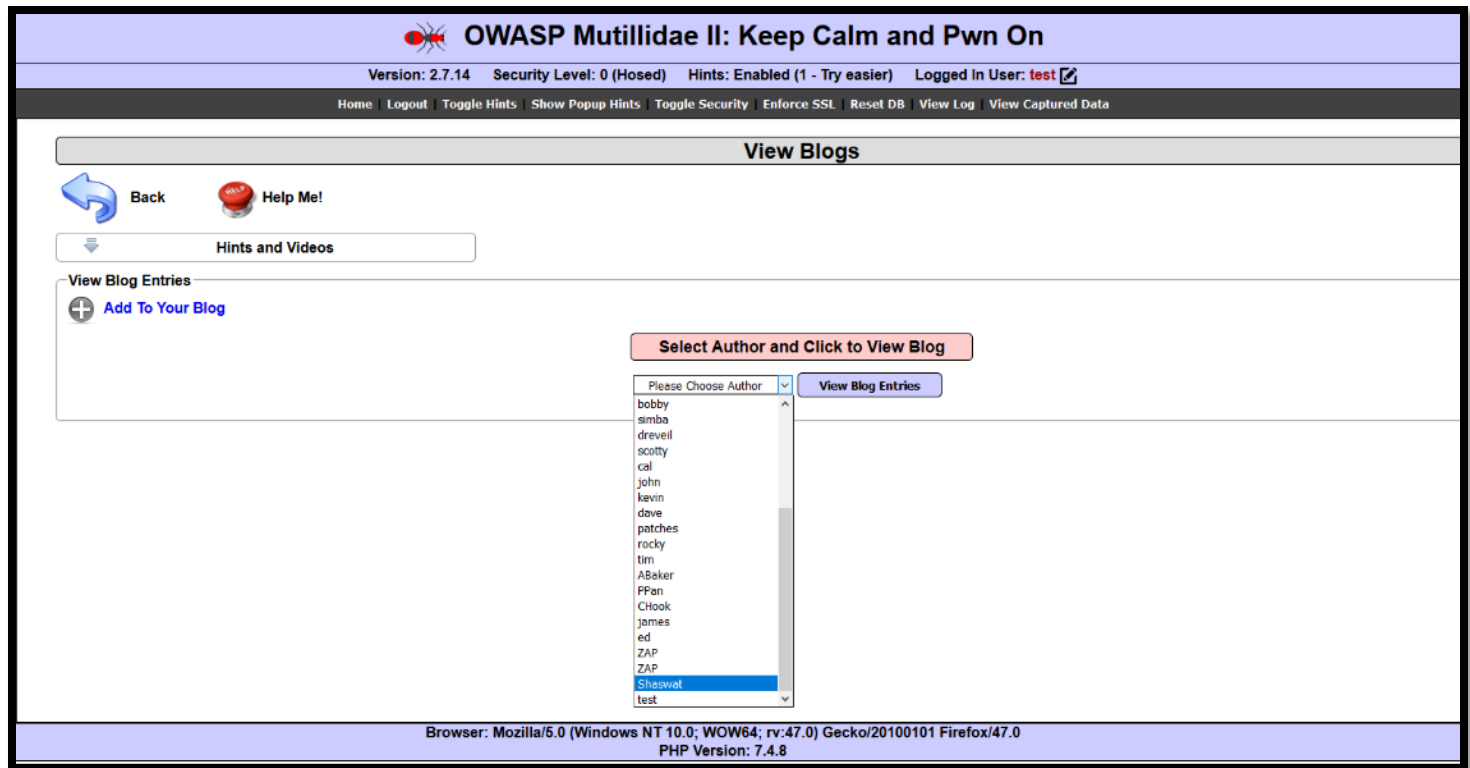
**Location / URL – Mutillidae** -> OWASP 2017 -> Broken Authentication and Session Management -> Authentication Bypass

-> **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.

Please sign-in

Username

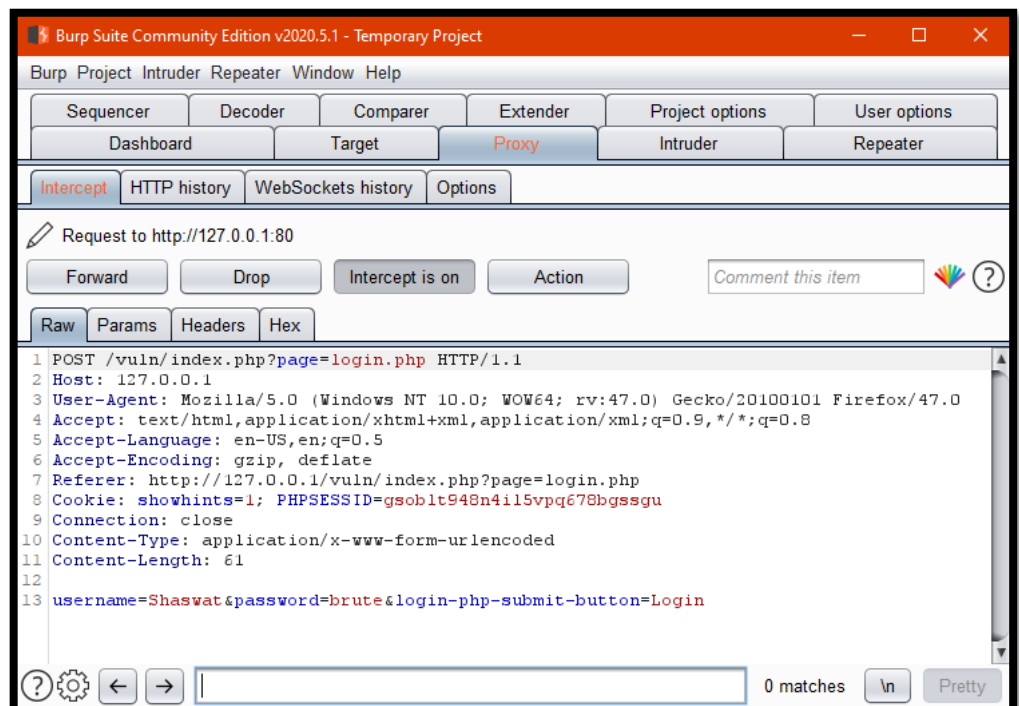
Shaswat

Password

.....

Login

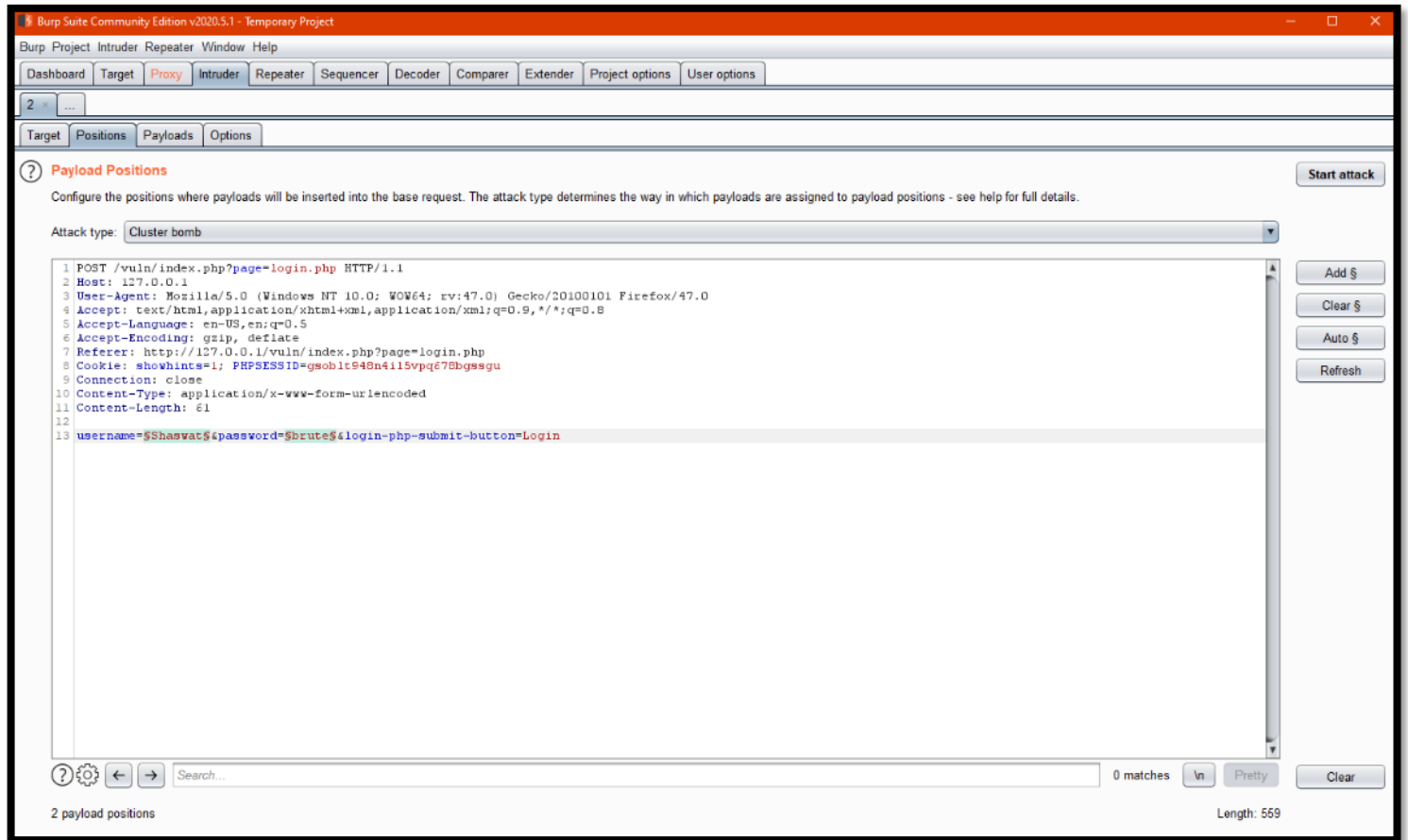
Dont have an account? [Please register here](#)



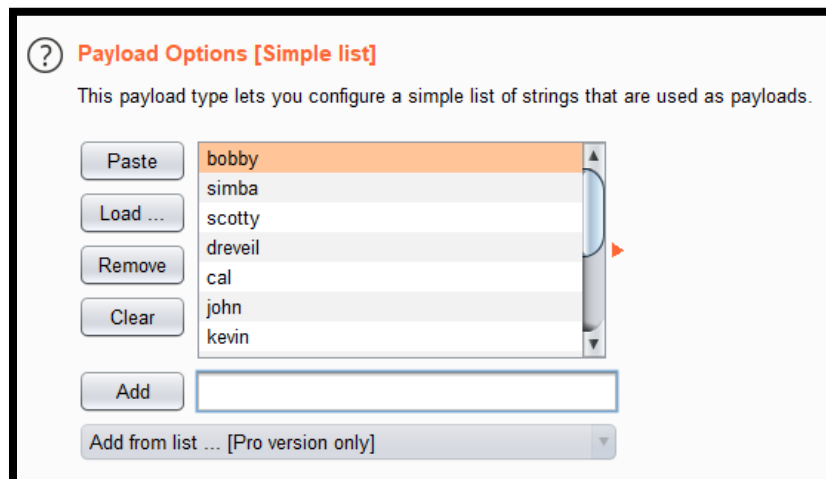
## INTERNSHIP: PROJECT REPORT

Send the intercepted request to the intruder, and select the username and password fields only.

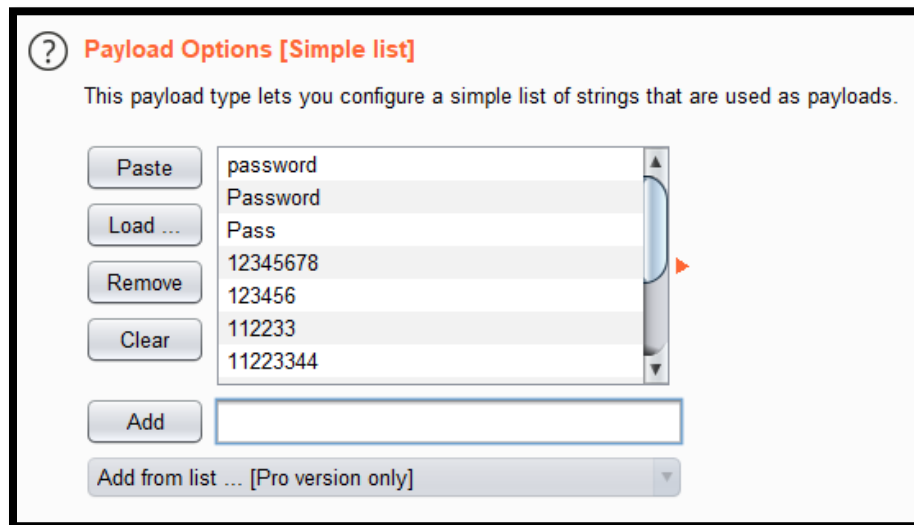
Use Cluster Bomb Attack Type.



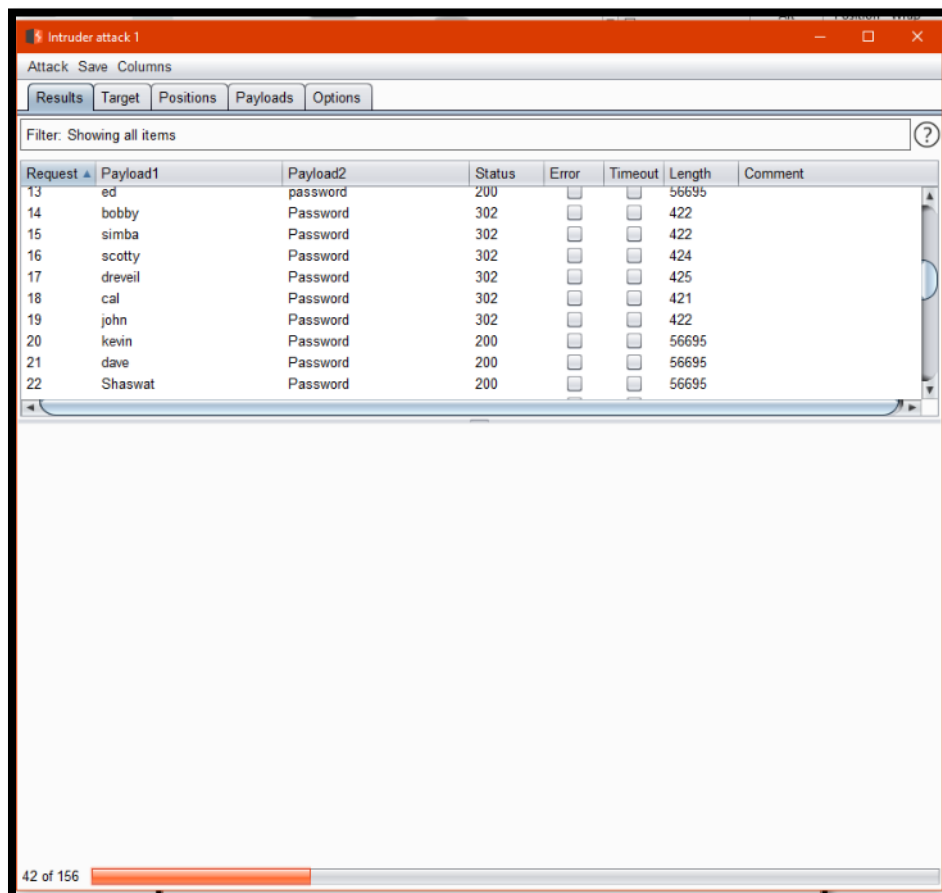
In the list 1 of payloads use a manually input usernames list.



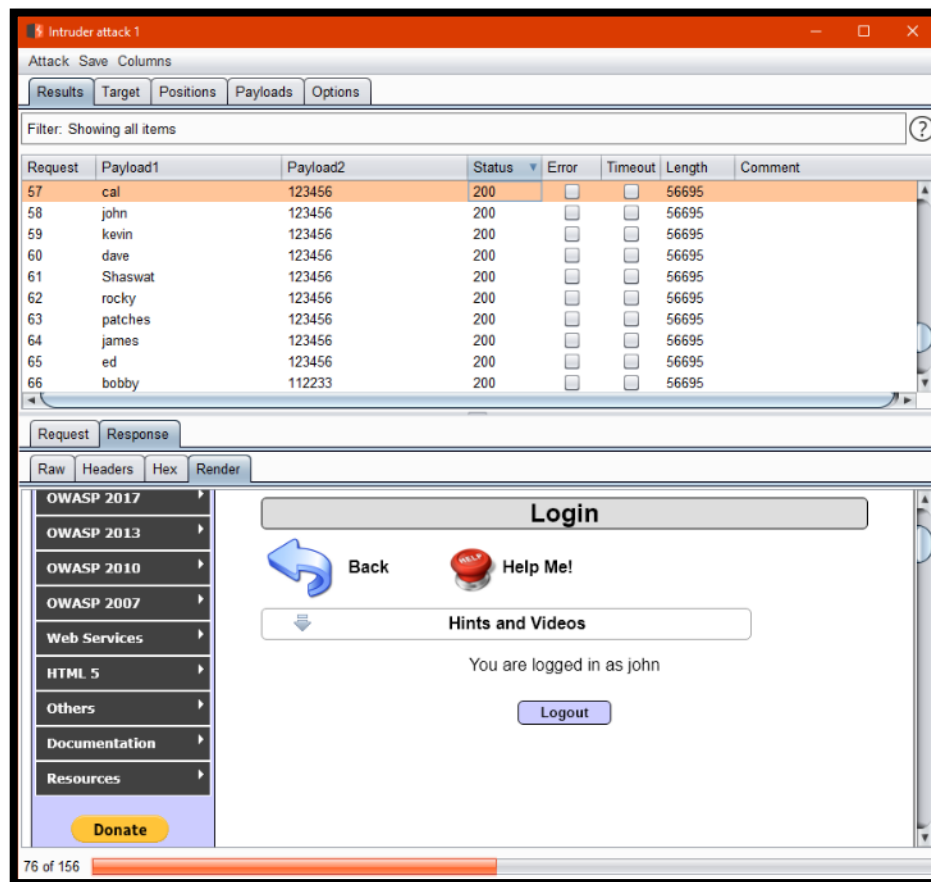
And for the list 2 use a list of most common passwords, we can get most common password list online.



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.

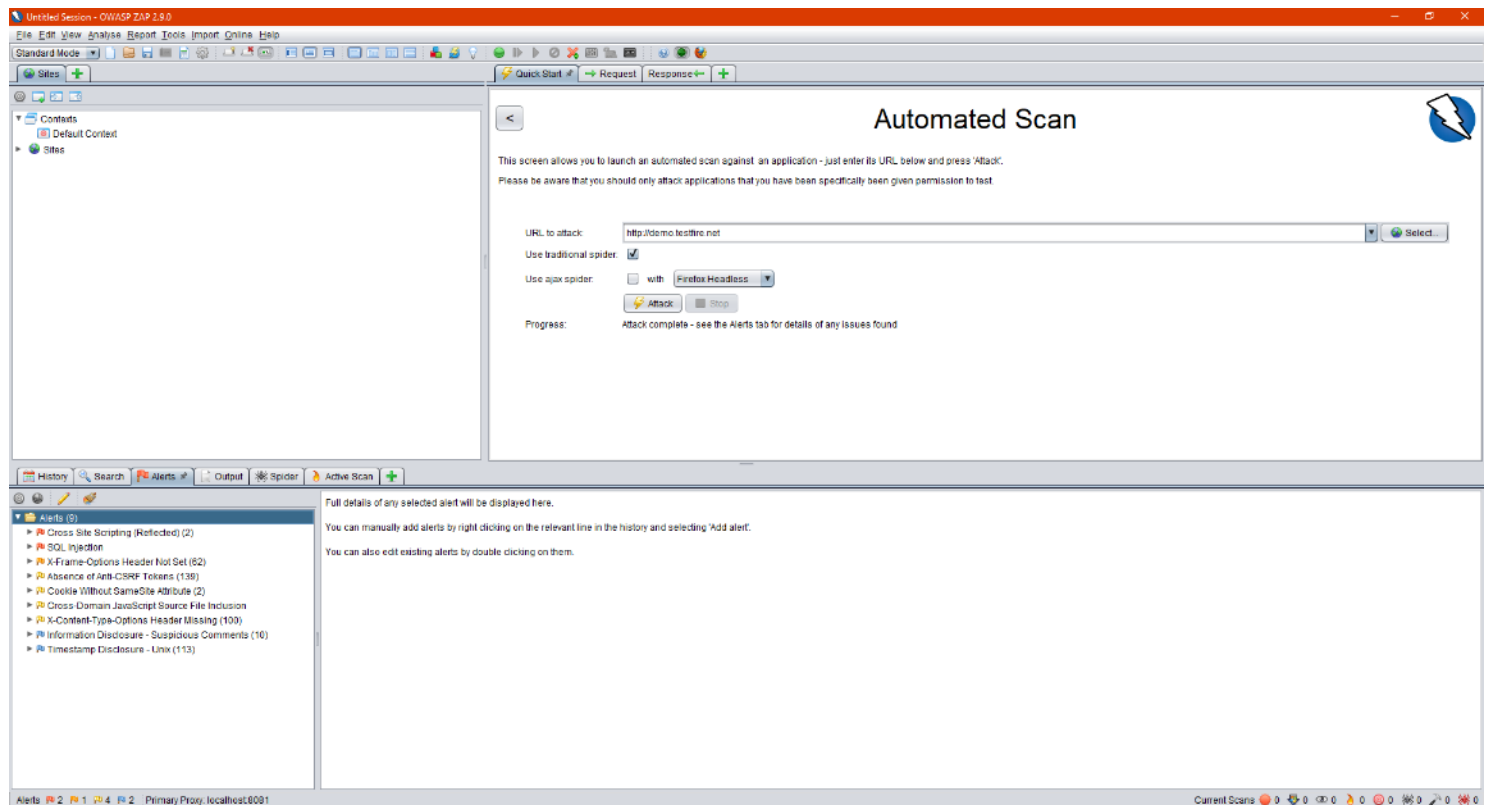
Here are the results I got about the vulnerabilities found in demo.testfire.net.

➤ demo.testfire.net Automated Scan Using OWASP ZAP



# INTERNSHIP: PROJECT REPORT

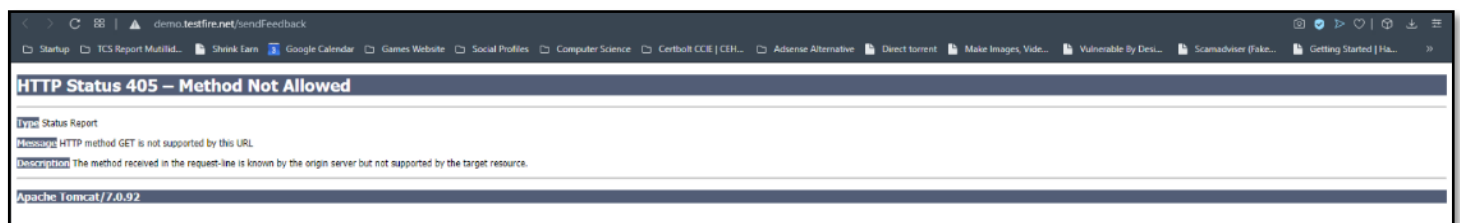
[#1]



Some of these found vulnerabilities were already found in manual testing while some were found to be false positives.

This was one of the errors when we try to open a found vulnerability from ZAP in system browser.

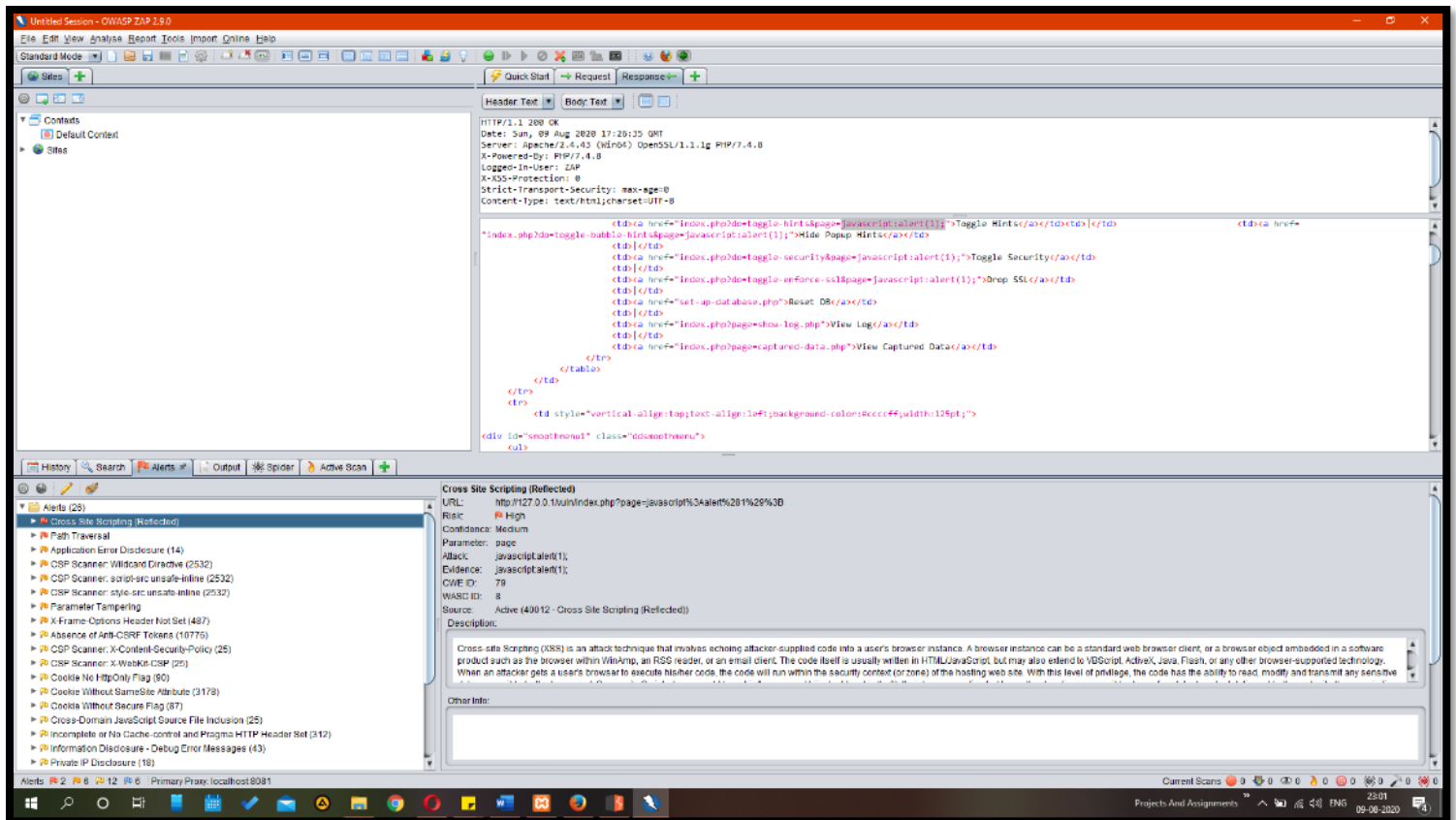
This indicates a false positive report.



## INTERNSHIP: PROJECT REPORT

Here are the results I got about the vulnerabilities found in Mutillidae

### ➤ Mutillidae Automated Scan Using OWASP ZAP



## [#1]

ZAP is confident with this vulnerability, but this doesn't work as expected. It indicated false positive results.

Edit Alert

Cross Site Scripting (Reflected)

URL:  
http://127.0.0.1/vuln/index.php?page=javascript%3Aalert%281%29%3B

Risk:  
High

Confidence:  
Medium

Parameter:  
page

Attack:  
javascript:alert(1);

Evidence:  
javascript:alert(1);

CWE ID:  
79

WASC ID:  
8

Description:  
Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a

Other Info:

Solution:  
Phase: Architecture and Design  
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to

Reference:

CancelSave

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

OWASP Mutillidae II: Keep Calm and Pwn On

Version: 2.7.14 Security Level: 0 (Hosed) Hints: Enabled (1 - Try easier) Not Logged In

Home Login/Register Toggle Hints Show Popup Hints Toggle Security Enforce SSL Reset DB View Log View Captured Data

OWASP 2017 OWASP 2013 OWASP 2010 OWASP 2007 Web Services HTML 5 Others Documentation Resources

Donate Want to Help? Video Tutorials Announcements Getting Started

Page Not Found

Back Help Me!

Validation Error: 404 - Page Not Found

Browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.105 Safari/537.36 OPR/70.0.3728.95 PHP Version: 7.4.8



### Outcome:

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

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

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

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

- SQL Injection
- XSS
- Broken Authentication
- Security Misconfiguration
- Sensitive Data Exposure

I 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. I 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.

I have got an expertise in using OWASP tool to find security defects rapidly and with comprehensive coverage. I 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.

## INTERNSHIP: PROJECT REPORT

---

- 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 internship project was perfectly carried out by the mentors.

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

---