INTERNSHIP REPORT

Intern : Suvhankar Dutta

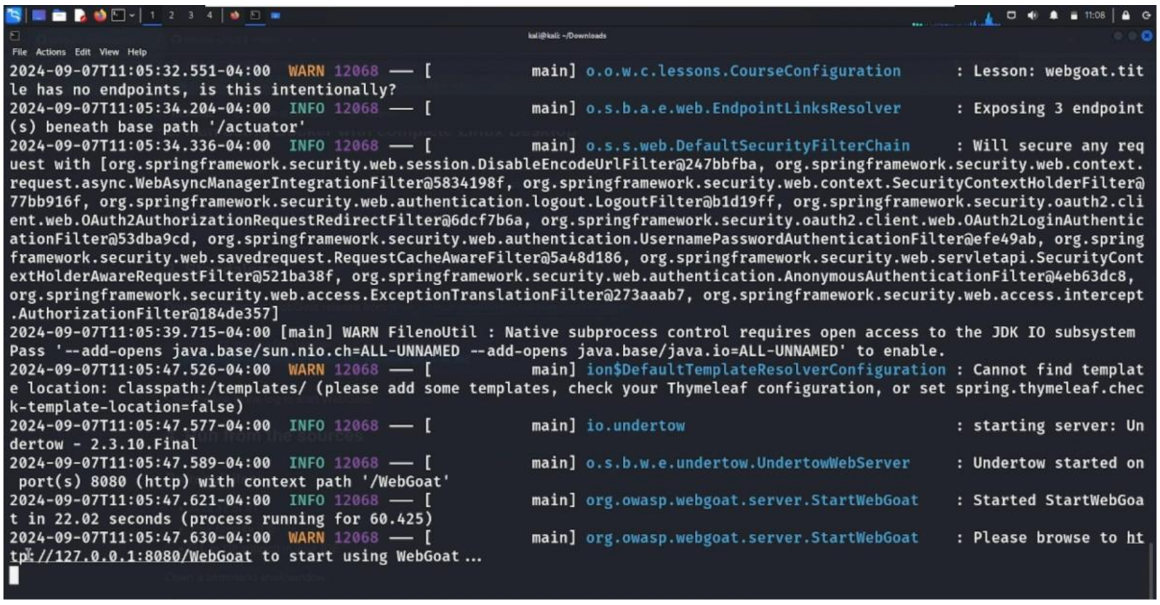
Internship Organization : Redynox

Duration: 1 Month

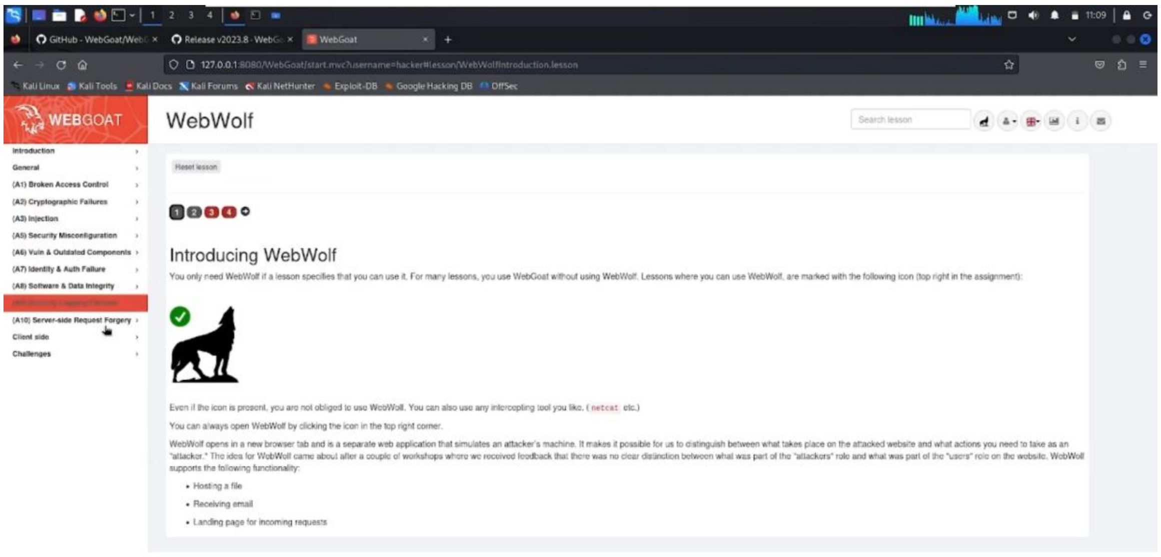
2. Task 2: Introduction to Web Application Security

2.1 Task 1: [Installing WEBGOAT]

first open up your browser go to Google and type OS web goat and press enter go to the first link where it says OS webcode when you're on this site on the right side you will see a link called Standalone jars under downloads click on the link and you will be prompted to a GitHub page scroll down under the latest version you will see a jar file so for me it is web code dash 2023.4.jar click on the link and your download of the jar file will start wait for the download to get finished after the download is finished go back to your terminal and change to the directory where your file got downloaded to CD downloads and I can now confirm that my file has been downloaded successfully next you need to have Java installed on your system Java comes pre-installed on Kali Linux type in the command javaspace dash version and I can see Java open jdk 17 is



```
2024-09-07T11:05:32.551-04:00  WARN 12068 --- [           main] o.o.w.c.lessons.CourseConfiguration      : Lesson: webgoat.tit
le has no endpoints, is this intentionally?
2024-09-07T11:05:34.204-04:00  INFO 12068 --- [           main] o.s.b.a.e.web.EndpointLinksResolver      : Exposing 3 endpoint
(s) beneath base path '/actuator'
2024-09-07T11:05:34.336-04:00  INFO 12068 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any req
uest with [org.springframework.security.web.session.DisableEncodeUrlFilter@247bbfba, org.springframework.security.web.context.
request.async.WebAsyncManagerIntegrationFilter@5834198f, org.springframework.security.web.context.SecurityContextHolderFilter@
77bb916f, org.springframework.security.web.authentication.logout.LogoutFilter@b1d19ff, org.springframework.security.oauth2.cli
ent.web.OAuth2AuthorizationRequestRedirectFilter@6dcf7b6a, org.springframework.security.oauth2.client.web.OAuth2LoginAuthentic
ationFilter@53dba9cd, org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter@efe49ab, org.spring
framework.security.web.savedrequest.RequestCacheAwareFilter@5a48d186, org.springframework.security.web.servletapi.SecurityCont
extHolderAwareRequestFilter@521ba38f, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@4eb63dc8,
org.springframework.security.web.access.ExceptionTranslationFilter@273aaab7, org.springframework.security.web.access.intercept
.AuthorizationFilter@184de357]
2024-09-07T11:05:39.715-04:00 [main] WARN FilenoUtil : Native subprocess control requires open access to the JDK IO subsystem
Pass '--add-opens java.base/sun.nio.ch=ALL-UNNAMED --add-opens java.base/java.io=ALL-UNNAMED' to enable.
2024-09-07T11:05:47.526-04:00  WARN 12068 --- [           main] ion$DefaultTemplateResolverConfiguration : Cannot find templat
e location: classpath:/templates/ (please add some templates, check your Thymeleaf configuration, or set spring.thymeleaf.chec
k-template-location=false)
2024-09-07T11:05:47.577-04:00  INFO 12068 --- [           main] io.undertow                              : starting server: Un
dertow - 2.3.10.Final
2024-09-07T11:05:47.589-04:00  INFO 12068 --- [           main] o.s.b.w.e.undertow.UndertowWebServer     : Undertow started on
 port(s) 8080 (http) with context path '/WebGoat'
2024-09-07T11:05:47.621-04:00  INFO 12068 --- [           main] org.owasp.webgoat.server.StartWebGoat    : Started StartWebGoa
t in 22.02 seconds (process running for 60.425)
2024-09-07T11:05:47.630-04:00  WARN 12068 --- [           main] org.owasp.webgoat.server.StartWebGoat    : Please browse to ht
tp://127.0.0.1:8080/WebGoat to start using WebGoat ...
```


GitHub - WebGoat/WebG
Release v2023.8 - WebG
WebGoat
127.0.0.1:8080/WebGoat/start.mvc?username=hacker#lesson/WebWolfIntroduction.lesson
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
WEBGOAT
WebWolf
Search lesson
Introduction
General
(A1) Broken Access Control
(A2) Cryptographic Failures
(A3) Injection
(A5) Security Misconfiguration
(A6) Vuln & Outdated Components
(A7) Identity & Auth Failure
(A8) Software & Data Integrity
(A10) Server-side Request Forgery
Client side
Challenges
Reset lesson
1 2 3 4

Introducing WebWolf

You only need WebWolf if a lesson specifies that you can use it. For many lessons, you use WebGoat without using WebWolf. Lessons where you can use WebWolf, are marked with the following icon (top right in the assignment):

Even if the icon is present, you are not obliged to use WebWolf. You can also use any intercepting tool you like. (`netcat` etc.)

You can always open WebWolf by clicking the icon in the top right corner.

WebWolf opens in a new browser tab and is a separate web application that simulates an attacker's machine. It makes it possible for us to distinguish between what takes place on the attacked website and what actions you need to take as an "attacker." The idea for WebWolf came about after a couple of workshops where we received feedback that there was no clear distinction between what was part of the "attackers" role and what was part of the "users" role on the website. WebWolf supports the following functionality:

- Hosting a file
- Receiving email
- Landing page for incoming requests

2.2 Task 2: Perform Basic Vulnerability Analysis:

Objective of the Task:

The main goal of this task was to explore how web applications can be vulnerable to common attacks. I used **OWASP ZAP**, a popular security tool, to scan a demo website and identify potential threats like:

- **SQL Injection**
- **Cross-Site Scripting (XSS)**
- **Cross-Site Request Forgery (CSRF)**

This task helped me understand how attackers might exploit these flaws and how developers can defend against them.

Tools and Methods Used:

For this task, I used:

- **OWASP ZAP** for scanning and detecting vulnerabilities.
- A browser (Chrome/Firefox) set up with ZAP as a proxy.
- **DVWA (Damn Vulnerable Web App)** running on XAMPP as the test site.
- **Kali Linux** to host and manage the scanning environment.

Steps I Followed:

1. **Environment Setup**:
   I first installed DVWA on my local machine using XAMPP and ensured it was running properly. I launched Kali Linux and opened OWASP ZAP.

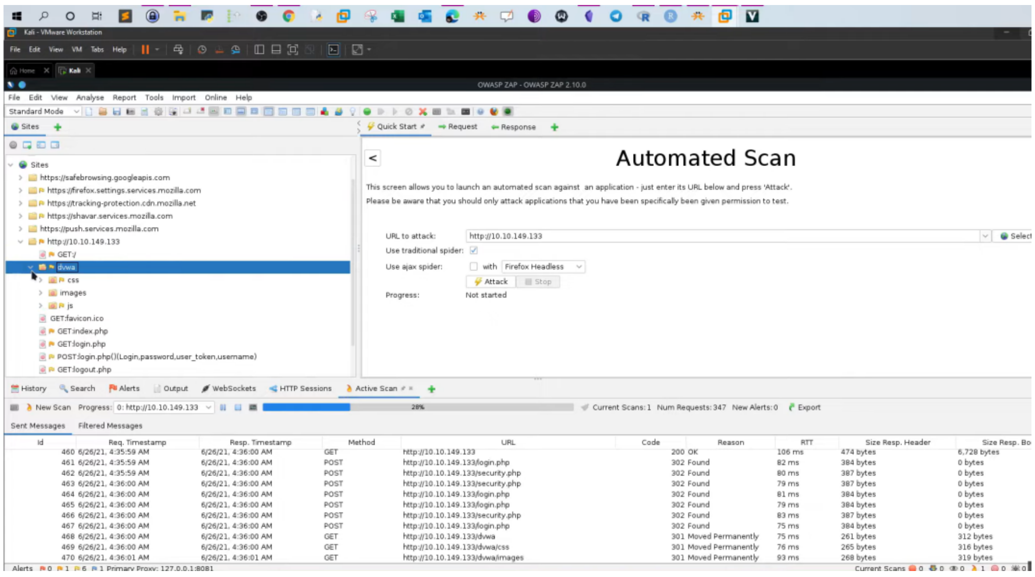
2. **Connecting ZAP with the Web App**:
   I configured my browser to route all traffic through ZAP's proxy. This allowed ZAP to monitor and capture every interaction I had with the test website.

3. **Scanning the Web App**:
   I used the **Spider** tool to crawl through all the pages of the site and then ran an **Active Scan** to automatically test for vulnerabilities.

4. **Finding SQL Injection**:
   ZAP flagged a login form where it injected test inputs like ' OR '1'='1—which could allow an attacker to log in without a password.

5. **Finding XSS**:
   In the site's search box, ZAP injected a script: <script>alert(1)</script>. When this script ran, it showed a popup, confirming the site was vulnerable to XSS.

6. **Finding CSRF**:
   ZAP also detected forms (like the user update form) that didn't have CSRF protection, meaning an attacker could trick someone into submitting that form unknowingly.


Kali - VMware Workstation
File Edit View VM Tabs Help
Home Kali
OWASP ZAP - OWASP ZAP 2.10.0
File Edit View Analyse Report Tools Import Online Help
Standard Mode
Sites
Quick Start
Request
Response
History Search Alerts Output WebSockets HTTP Sessions Active Scan
New Scan
Sent Messages Filtered Messages
Export

Sites
- https://safebrowsing.googleapis.com
- https://firefox.settings.services.mozilla.com
- https://tracking-protection.cdn.mozilla.net
- https://shavar.services.mozilla.com
- https://push.services.mozilla.com
- http://10.10.149.133
  - GET:/
  - dvwa
    - css
    - images
    - js
    - GET:favicon.ico
    - GET:index.php
    - GET:login.php
    - POST:login.php()(Login,password,user_token,username)
    - GET:logout.php

Automated Scan

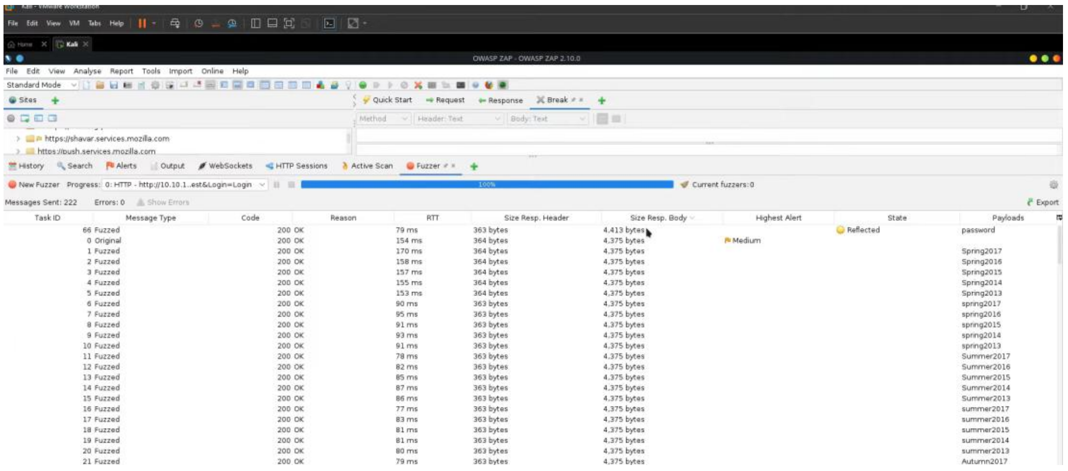
This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://10.10.149.133
Use traditional spider: ✓
Use ajax spider: ☐ with Firefox Headless
Attack Stop
Progress: Not started

Progress: 0: http://10.10.149.133 28% Current Scans: 1 Num Requests: 347 New Alerts: 0

Id	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Bo
460	6/26/21, 4:35:59 AM	6/26/21, 4:36:00 AM	GET	http://10.10.149.133	200	OK	106 ms	474 bytes	6,728 bytes
461	6/26/21, 4:35:59 AM	6/26/21, 4:36:00 AM	POST	http://10.10.149.133/login.php	302	Found	82 ms	384 bytes	0 bytes
462	6/26/21, 4:35:59 AM	6/26/21, 4:36:00 AM	POST	http://10.10.149.133/security.php	302	Found	80 ms	387 bytes	0 bytes
463	6/26/21, 4:36:00 AM	6/26/21, 4:36:00 AM	POST	http://10.10.149.133/security.php	302	Found	79 ms	387 bytes	0 bytes
464	6/26/21, 4:36:00 AM	6/26/21, 4:36:00 AM	POST	http://10.10.149.133/login.php	302	Found	81 ms	384 bytes	0 bytes
465	6/26/21, 4:36:00 AM	6/26/21, 4:36:00 AM	POST	http://10.10.149.133/login.php	302	Found	79 ms	384 bytes	0 bytes
466	6/26/21, 4:36:00 AM	6/26/21, 4:36:00 AM	POST	http://10.10.149.133/security.php	302	Found	83 ms	387 bytes	0 bytes
467	6/26/21, 4:36:00 AM	6/26/21, 4:36:00 AM	POST	http://10.10.149.133/login.php	302	Found	75 ms	384 bytes	0 bytes
468	6/26/21, 4:36:00 AM	6/26/21, 4:36:00 AM	GET	http://10.10.149.133/dvwa	301	Moved Permanently	75 ms	261 bytes	312 bytes
469	6/26/21, 4:36:00 AM	6/26/21, 4:36:00 AM	GET	http://10.10.149.133/dvwa/css	301	Moved Permanently	76 ms	265 bytes	316 bytes
470	6/26/21, 4:36:01 AM	6/26/21, 4:36:01 AM	GET	http://10.10.149.133/dvwa/images	301	Moved Permanently	93 ms	268 bytes	319 bytes

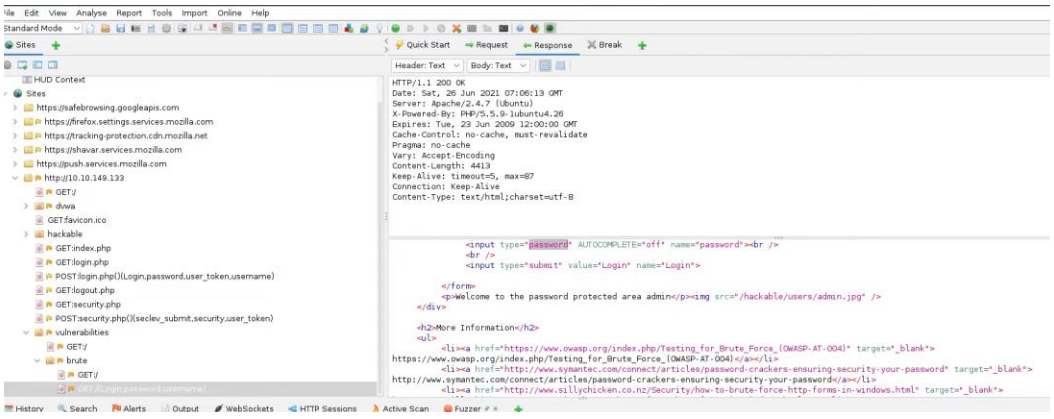
Alerts 0 1 6 1 Primary Proxy: 127.0.0.1:8081 Current Scans 0 0 0 1 0 0



Messages Sent: 222 Errors: 0

Progress: 0: HTTP - http://10.10.1..est&Login=Login 100% Current fuzzers: 0

Task ID	Message Type	Code	Reason	RTT	Size Resp. Header	Size Resp. Body	Highest Alert	State	Payloads
66	Fuzzed	200	OK	79 ms	363 bytes	4,413 bytes		Reflected	password
0	Original	200	OK	154 ms	364 bytes	4,375 bytes	Medium		
1	Fuzzed	200	OK	170 ms	364 bytes	4,375 bytes			Spring2017
2	Fuzzed	200	OK	158 ms	364 bytes	4,375 bytes			Spring2016
3	Fuzzed	200	OK	157 ms	364 bytes	4,375 bytes			Spring2015
4	Fuzzed	200	OK	155 ms	364 bytes	4,375 bytes			Spring2014
5	Fuzzed	200	OK	153 ms	364 bytes	4,375 bytes			Spring2013
6	Fuzzed	200	OK	90 ms	363 bytes	4,375 bytes			spring2017
7	Fuzzed	200	OK	95 ms	363 bytes	4,375 bytes			spring2016
8	Fuzzed	200	OK	91 ms	363 bytes	4,375 bytes			spring2015
9	Fuzzed	200	OK	93 ms	363 bytes	4,375 bytes			spring2014
10	Fuzzed	200	OK	91 ms	363 bytes	4,375 bytes			spring2013
11	Fuzzed	200	OK	78 ms	363 bytes	4,375 bytes			Summer2017
12	Fuzzed	200	OK	82 ms	363 bytes	4,375 bytes			Summer2016
13	Fuzzed	200	OK	85 ms	363 bytes	4,375 bytes			Summer2015
14	Fuzzed	200	OK	87 ms	363 bytes	4,375 bytes			Summer2014
15	Fuzzed	200	OK	86 ms	363 bytes	4,375 bytes			Summer2013
16	Fuzzed	200	OK	77 ms	363 bytes	4,375 bytes			summer2017
17	Fuzzed	200	OK	83 ms	363 bytes	4,375 bytes			summer2016
18	Fuzzed	200	OK	81 ms	363 bytes	4,375 bytes			summer2015
19	Fuzzed	200	OK	81 ms	363 bytes	4,375 bytes			summer2014
20	Fuzzed	200	OK	80 ms	363 bytes	4,375 bytes			summer2013
21	Fuzzed	200	OK	79 ms	363 bytes	4,375 bytes			Autumn2017

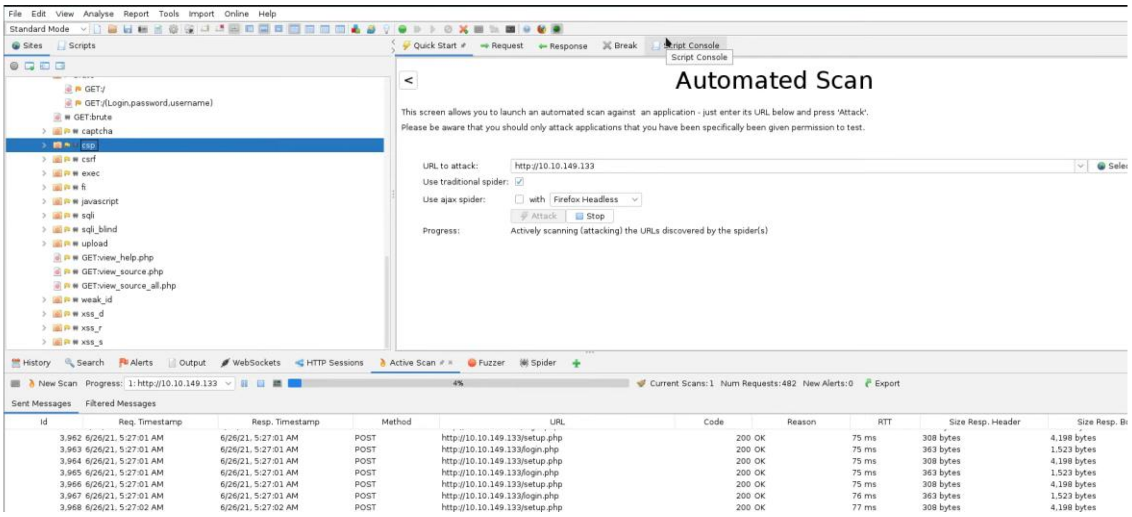


```
HTTP/1.1 200 OK
Date: Sat, 26 Jun 2021 07:06:13 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.26
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 4413
Keep-Alive: timeout=5, max=87
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
```

```
                <input type="password" AUTOCOMPLETE="off" name="password"><br />
                <br />
                <input type="submit" value="Login" name="Login">

            </form>
            <p>Welcome to the password protected area admin</p><img src="/hackable/users/admin.jpg" />
        </div>

        <h2>More Information</h2>
        <ul>
            <li><a href="https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)" target="_blank">
https://www.owasp.org/index.php/Testing_for_Brute_Force_(OWASP-AT-004)</a></li>
            <li><a href="http://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password" target="_blank">
http://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password</a></li>
            <li><a href="http://www.sillychicken.co.nz/Security/how-to-brute-force-http-forms-in-windows.html" target="_blank">
```



Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://10.10.149.133
Use traditional spider: ✓
Use ajax spider: with Firefox Headless
Progress: Actively scanning (attacking) the URLs discovered by the spider(s)

Progress: 1: http://10.10.149.133 4% Current Scans: 1 Num Requests: 482 New Alerts: 0

Id	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
3,962	6/26/21, 5:27:01 AM	6/26/21, 5:27:01 AM	POST	http://10.10.149.133/setup.php	200	OK	75 ms	308 bytes	4,198 bytes
3,963	6/26/21, 5:27:01 AM	6/26/21, 5:27:01 AM	POST	http://10.10.149.133/login.php	200	OK	75 ms	363 bytes	1,523 bytes
3,964	6/26/21, 5:27:01 AM	6/26/21, 5:27:01 AM	POST	http://10.10.149.133/setup.php	200	OK	75 ms	308 bytes	4,198 bytes
3,965	6/26/21, 5:27:01 AM	6/26/21, 5:27:01 AM	POST	http://10.10.149.133/login.php	200	OK	75 ms	363 bytes	1,523 bytes
3,966	6/26/21, 5:27:01 AM	6/26/21, 5:27:01 AM	POST	http://10.10.149.133/setup.php	200	OK	75 ms	308 bytes	4,198 bytes
3,967	6/26/21, 5:27:01 AM	6/26/21, 5:27:01 AM	POST	http://10.10.149.133/login.php	200	OK	76 ms	363 bytes	1,523 bytes
3,968	6/26/21, 5:27:02 AM	6/26/21, 5:27:02 AM	POST	http://10.10.149.133/setup.php	200	OK	77 ms	308 bytes	4,198 bytes

Challenges I Faced and How I Solved Them:

- **HTTPS Traffic Wasn't Showing Up**:
  At first, I couldn't see any HTTPS traffic in ZAP. To fix this, I installed ZAP's security certificate into my browser to allow it to decrypt secure traffic.

- **DVWA Security Level Was Too High**:
  Some vulnerabilities didn't show up initially. I later found out DVWA was set to "high" security. Switching it to "low" made it easier to test.

- **Understanding the Alerts**:
  ZAP shows a lot of alerts, which was confusing at first. But after going through them carefully and testing them manually, I understood how each attack worked.

Results and What I Learned:

By the end of this task, I was able to successfully:

- Find an **SQL Injection** in the login form.
- Confirm a **Cross-Site Scripting (XSS)** vulnerability in the search bar.
- Identify a **CSRF vulnerability** in the profile update form.

This hands-on experience taught me how automated tools like OWASP ZAP can find real security flaws and how developers can secure their applications by addressing these issues early. It also gave me a clearer idea of how ethical hackers think and operate during real-world security assessments.

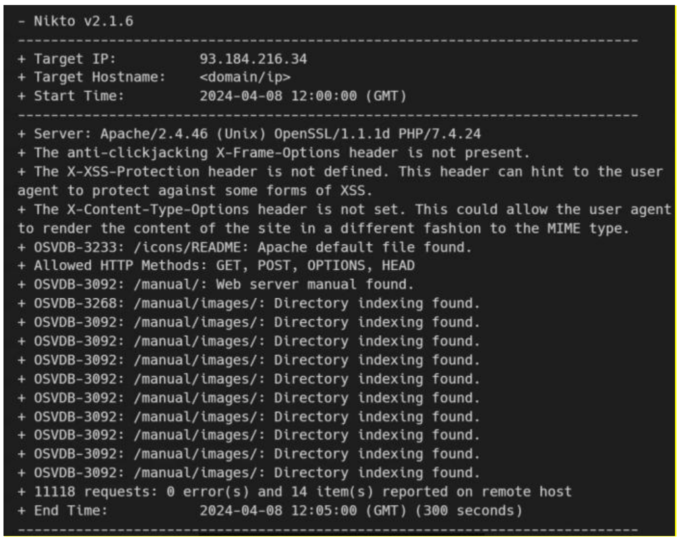
1.3 Task 3 Exploring Vulnerabilities

Explore Vulnerabilities Using OWASP ZAP

Understanding How Vulnerabilities Work:

As part of this task, I used OWASP ZAP not only to scan for vulnerabilities but also to understand the technical details behind each issue. ZAP provides descriptions, risk levels, and recommended solutions for every detected vulnerability, which helped me gain a deeper understanding of how attacks are performed and how they can be prevented.

Here are three key vulnerabilities I focused on:



```
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          93.184.216.34
+ Target Hostname:    <domain/ip>
+ Start Time:         2024-04-08 12:00:00 (GMT)
---------------------------------------------------------------------------
+ Server: Apache/2.4.46 (Unix) OpenSSL/1.1.1d PHP/7.4.24
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user
agent to protect against some forms of XSS.
+ The X-Content-Type-Options header is not set. This could allow the user agent
to render the content of the site in a different fashion to the MIME type.
+ OSVDB-3233: /icons/README: Apache default file found.
+ Allowed HTTP Methods: GET, POST, OPTIONS, HEAD
+ OSVDB-3092: /manual/: Web server manual found.
+ OSVDB-3268: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ OSVDB-3092: /manual/images/: Directory indexing found.
+ 11118 requests: 0 error(s) and 14 item(s) reported on remote host
+ End Time:           2024-04-08 12:05:00 (GMT) (300 seconds)
---------------------------------------------------------------------------
```

SQL Injection (SQL):

- **How it works**: SQL Injection occurs when user input is improperly filtered, allowing attackers to execute arbitrary SQL commands on a database. This can lead to unauthorized access, data leakage, or even deletion of entire databases.

- **ZAP Alert**: ZAP highlighted a vulnerable login parameter. It included a test payload like ' OR '1'='1.
- **Manual Exploitation**: I tested this manually by entering ' OR '1'='1 -- in the username field of DVWA's login form. It bypassed authentication and granted access without valid credentials.
- **Learning Outcome**: This showed how poor input validation can compromise authentication mechanisms.

Cross-Site Scripting (XSS):

- **How it works**: XSS lets attackers inject malicious scripts into webpages viewed by other users. It can be used to steal session cookies, deface websites, or redirect users.
- **ZAP Alert**: The tool flagged a search input field as vulnerable to reflected XSS.
- **Manual Exploitation**: I manually entered <script>alert("XSS Test")</script> into the search box. The script executed on the same page, confirming the vulnerability.
- **Learning Outcome**: I learned how script injection can hijack browser sessions if output is not sanitized properly.

Cross-Site Request Forgery (CSRF):

- **How it works**: CSRF tricks an authenticated user into unknowingly submitting a request to a web app, such as changing account details or passwords.
- **ZAP Alert**: ZAP identified forms that lacked anti-CSRF tokens.
- **Manual Exploitation**: Using ZAP's "Generate Anti-CSRF Test Form" feature, I crafted a fake HTML form that could submit a request on behalf of the user. When executed, it successfully updated the account details without user consent.
- **Learning Outcome**: This showed the importance of CSRF tokens and how simple HTML forms can exploit missing protections.

  Summary:

  By combining automated scanning with manual testing, I was able to clearly understand how these vulnerabilities work in real-world applications. Reading the detailed explanations provided by OWASP ZAP helped me bridge the gap between theoretical knowledge and practical exploitation. This exercise improved my ability to recognize insecure coding practices and understand how attackers think.

3. Conclusion

This internship was a transformative experience that equipped me with both theoretical knowledge and hands-on cybersecurity skills. I learned how to:

- Secure a small network using industry best practices.
- Analyze and understand network traffic with Wireshark.
- Scan web applications and find critical vulnerabilities using OWASP ZAP.
- Improve my professional branding through strategic LinkedIn engagement.

The experience also strengthened my passion for ethical hacking and system defense.

4. Closing Remarks

I sincerely thank the Redynox team for the opportunity and guidance throughout the internship. The structured tasks, real-world simulations, and self-paced learning approach made this journey enriching. I recommend future interns to:

- Stay curious and try manual testing along with tools.
- Document everything with screenshots and notes.
- Engage in the cybersecurity community for continuous learning.

Looking forward to applying these skills in future cybersecurity roles.