OWASP ZAP a Vulnerability Scanner

Cybersecurity Internship – Phase 1, Task 4

Name: Parmar Rakesh
Date:16,AUG,2025
Internship Program: Cybersecurity Internship – UJAR TECH

What is OWASP Zed Attack Proxy?

Zed Attack Proxy (ZAP) is a free, open-source tool developed by OWASP that is used to test the security of websites and web applications. It is one of the most popular tools among penetration testers, bug bounty hunters, and developers because it helps them find security weaknesses before attackers can exploit them. ZAP is especially useful during the development and testing stages of an application, as it makes it easier to identify problems early on.ZAP works like a **middle-man (proxy)** between your browser and the web application. Every time you visit a website through ZAP, the tool records all the requests your browser sends and all the responses the website sends back. This allows you to see the exact data being exchanged, and even modify it if you want to test how the application reacts under different conditions. For example, you could change a request to see if the application is vulnerable to attacks like SQL injection or cross-site scripting (XSS).

The tool offers both **passive and active scanning**. Passive scanning means it quietly watches the traffic between your browser and the website, checking for common issues without sending any extra requests. Active scanning, on the other hand, is more aggressive-it actively sends test requests to the website to look for vulnerabilities. Besides this, ZAP also has features for **API testing**, which is important as modern applications often rely on APIs to communicate.

In short, ZAP is like a **security guard for your website**. It lets you monitor, intercept, and manipulate the traffic to understand how your site behaves and where it might be weak. By using ZAP, you can identify potential attack paths and fix them before hackers have the chance to take advantage.

OWASP ZAP vs BurpSuite

Similar Purpose

Both **OWASP ZAP** and **Burp Suite** are used for web application security testing:

- Intercept requests
- Find vulnerabilities
- Perform fuzzing & scanning

Price & Accessibility

- **ZAP** ⇒ 100% Free & Open-source
- **Burp Suite** ⇒ Free (limited), **Professional (Paid)** for full features

Speed & Performance

- **ZAP** ⇒ Lighter, faster, less resource-heavy
- **Burp** ⇒ Heavier, more polished

Feature Comparison

1. Auto Scan

- Burp ⇒ Only in *Pro* (Professional) version
- ZAP ⇒ Free (Attack Mode)

2. Intruder vs Fuzzer

- Burp ⇒ *Intruder* (limited in free, single-threaded)
- ZAP ⇒ *Fuzzer* (fully free)

3. Extra Features in ZAP

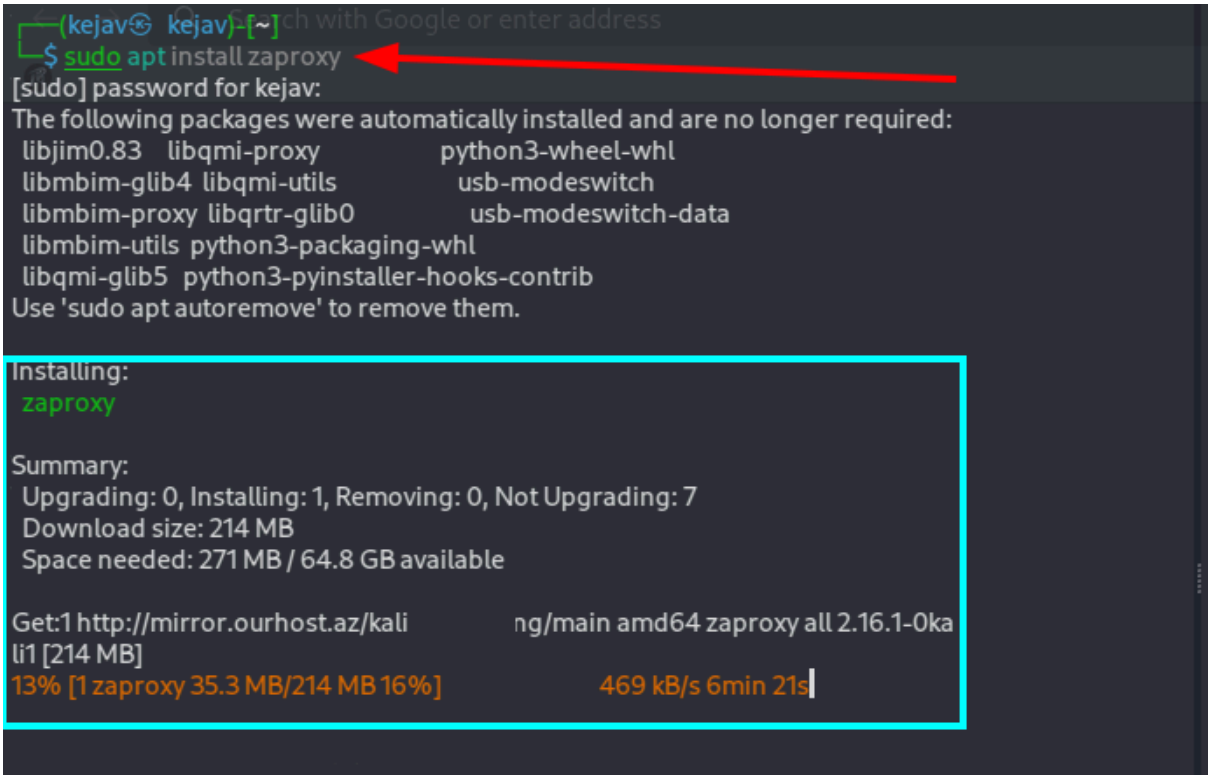
- **Automation Framework** ⇒ Run scans via YAML configs
- **HUD (Heads-Up Display)** ⇒ Test directly inside the browser

Feature Mapping (Burp ⇒ ZAP Equivalent)

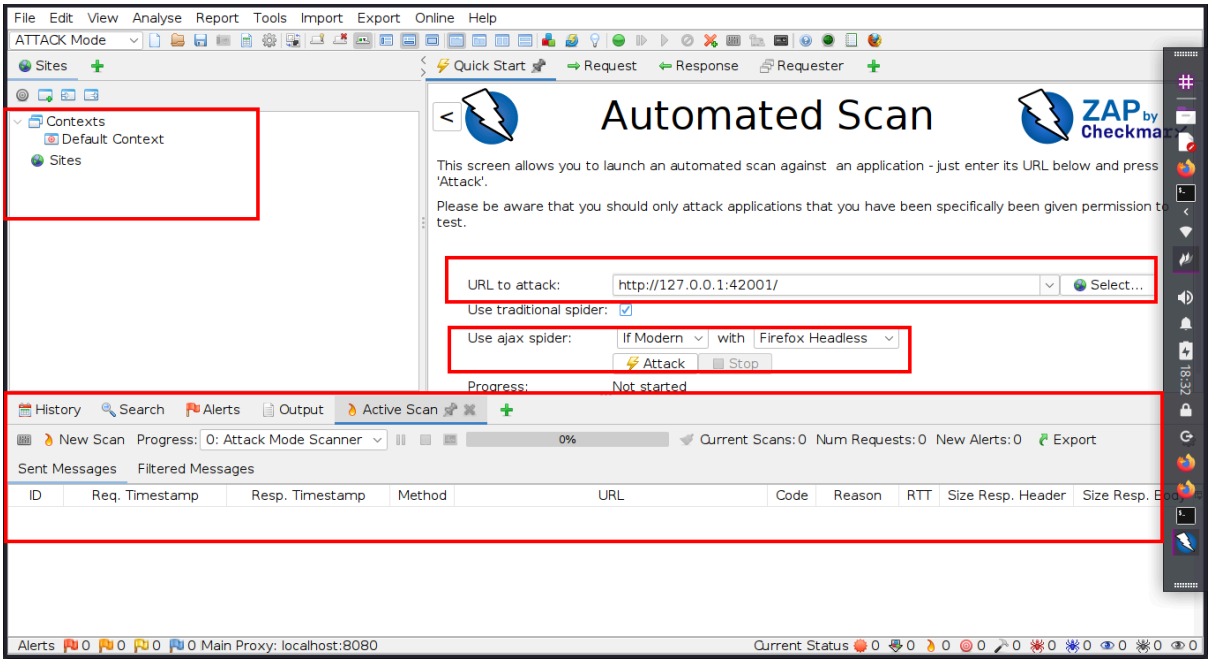
Burp Suite Feature	ZAP Equivalent
Burp Collaborator	ZAP OAST Support Add-on
Decoder / Encoder	ZAP Compare/Diff tools + Add-ons
DOM Invader	ZAP Eval Villain Add-on
Burp Extender	ZAP Marketplace (Add-ons + Scripts)
Intercept	ZAP Breakpoints
Intruder	ZAP Fuzzer
Live Scan (Community)	ZAP Attack Mode
Project Files	ZAP Session Files
Proxy	Proxy (same feature)
Repeater	Manual Request Editor (Requestor)
Scanner (Community)	Active Scanner
Sequencer	Token Analysis Tool

How to setup ZAP in Kali Linux?

ZAP is not installed in the current version of Kali, which is 2025.2 at the time of this writing. However, it can be easily installed.To setup ZAP you need to just run **sudo apt install zaproxy** in the terminal and that's it. As we set up Proxy in Burpsuite same at it is we can configure ZAP proxy[13].

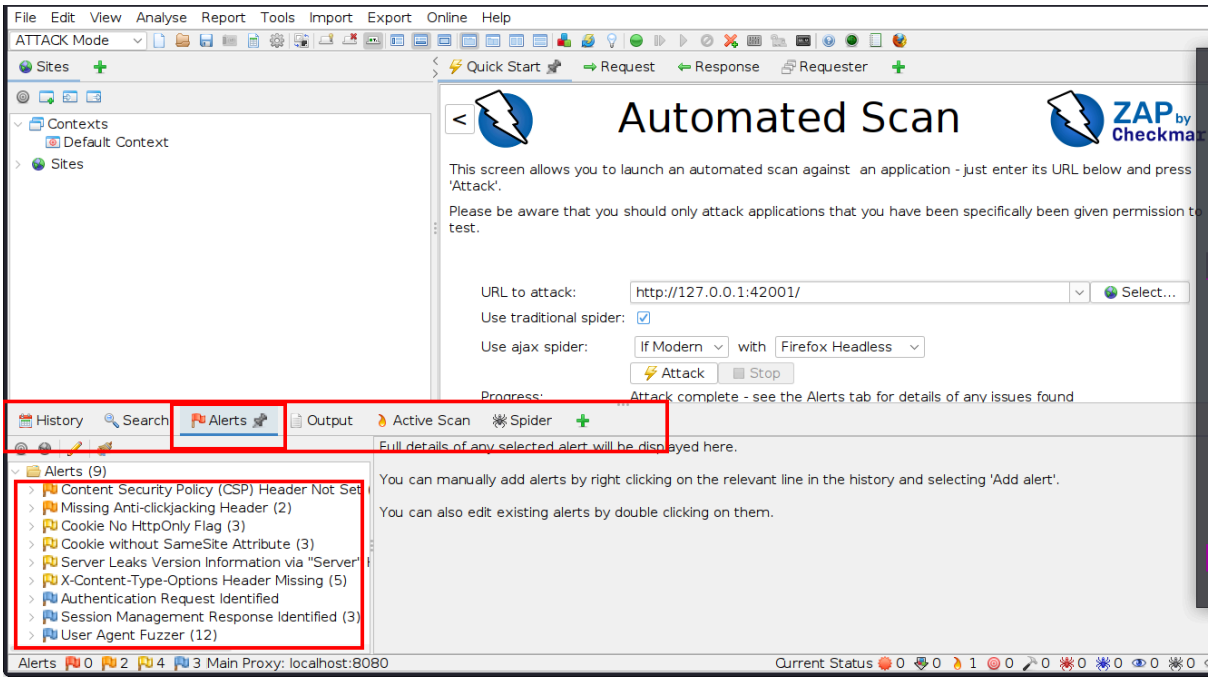


Automatic Scan on ZAP



According to your requirement you can fill your ip or domain url in **URL to attack field.**

I got 6 alert which you can see in image

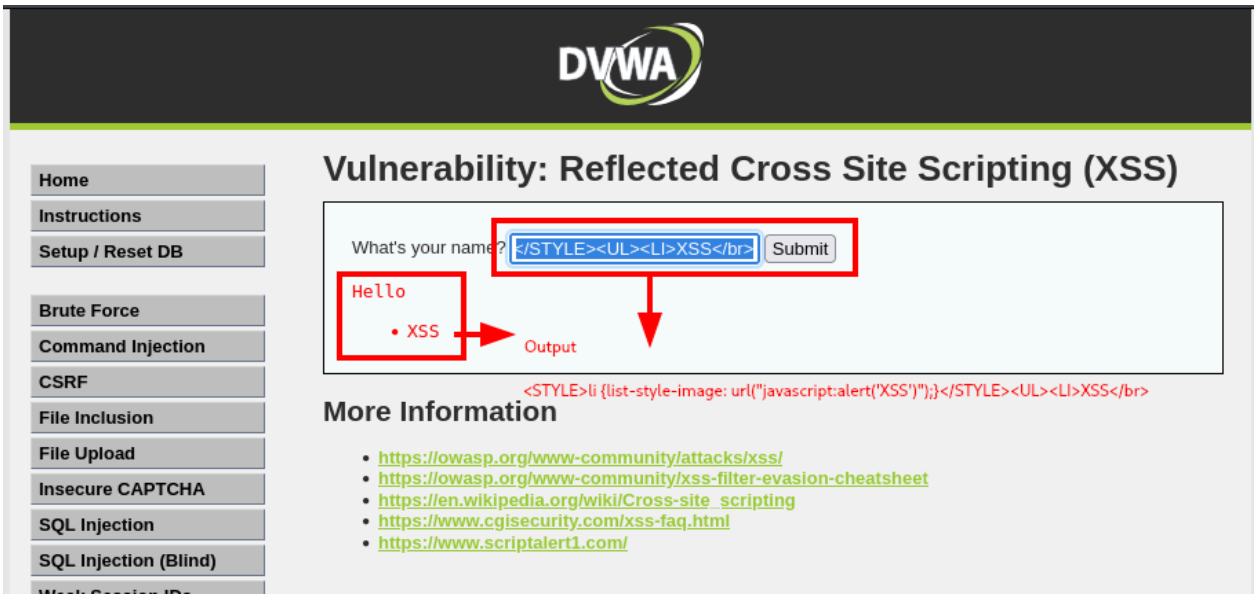


Let's check if it is vulnerable or not.

1.Content Security Policy (CSP) Header Not Set

URL:http://127.0.0.1:42001/vulnerabilities/xss_r/?name
Risk:Medium
CWE ID:693
WASC ID:15
Description:Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.
Solution:Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header[14].

Payload used:
<STYLE>li {list-style-image: url("javascript:alert('XSS')");}</STYLE><UL><LI>XSS</br>
POC of CSP:



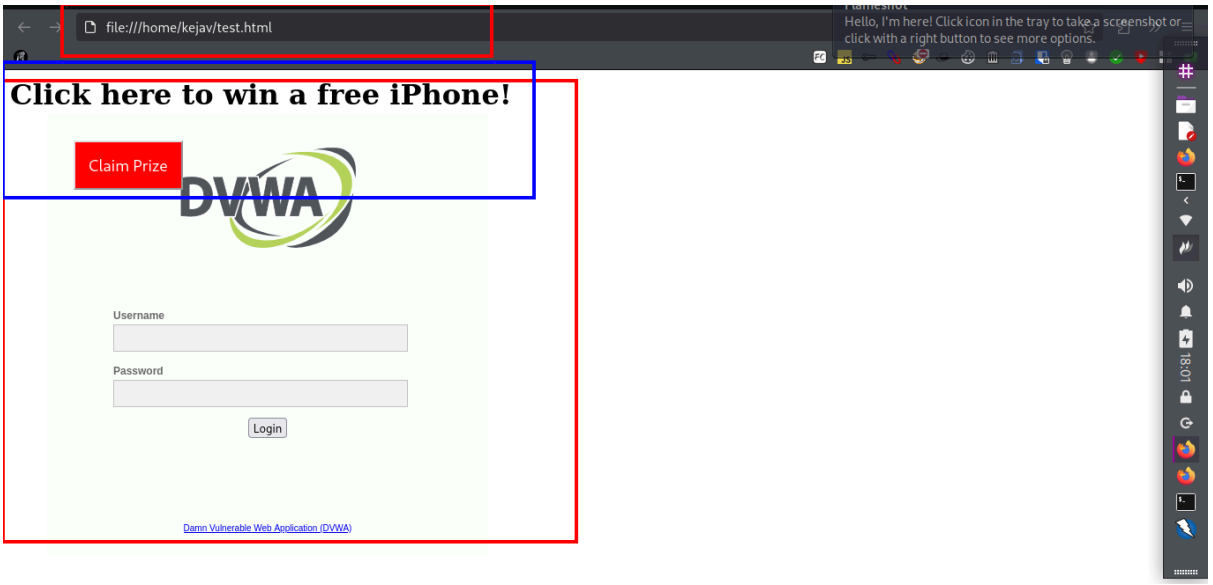
2.Missing Anti-clickjacking Header

URL:http://127.0.0.1:42001/login.php
Risk:Medium
CWE ID:1021
WASC ID:15
Description:The response does not protect against 'ClickJacking' attacks. It should include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options.
Solution:Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app.
If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive[15].

Payload:

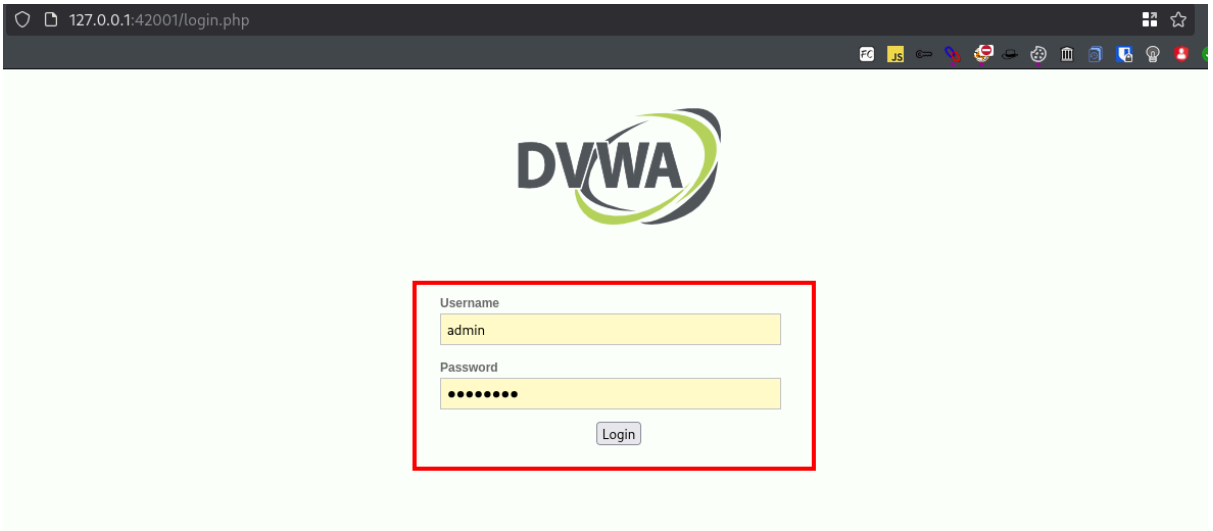
```
<html><!DOCTYPE html>
<html>
<head>
 <title>Clickjacking Demo</title>
 <style>
  iframe {
   position: absolute;
   top: 50px;
   left: 50px;
   opacity: 1; /* almost invisible */
   z-index: 1;
   width: 500px;
   height: 500px;
  }
  button {
   position: absolute;
   top: 80px;
   left: 80px;
   z-index: 2;
   padding: 15px;
   font-size: 18px;
   background: red;
   color: white;
  }
 </style>
</head>
<body>
 <h1>Click here to win a free iPhone!</h1>
 <button>Claim Prize</button>

 <iframe src="http://127.0.0.1:42001/login.php" frameborder="0"></iframe>
</body>
</html>
```

POC Of Clickjacking:

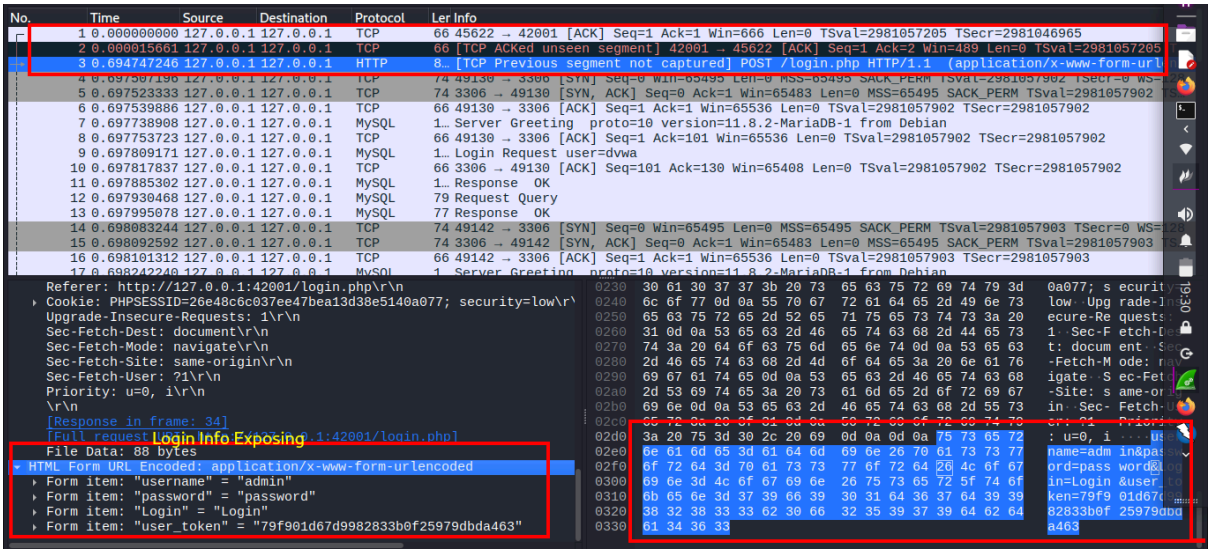


3.Cookie No HttpOnly Flag

URL:http://127.0.0.1:42001/login.php
Risk:Low
CWD ID:1004
WASC ID:13
Description:A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.
Solution:Ensure that the HttpOnly flag is set for all cookies[16].
Payload:I just opened Wireshark and captured the incoming HTTP requests. When I logged into the DVWA website, that's it - the login credentials were clearly visible in plain text.



POC of Cookie No HttpOnly:

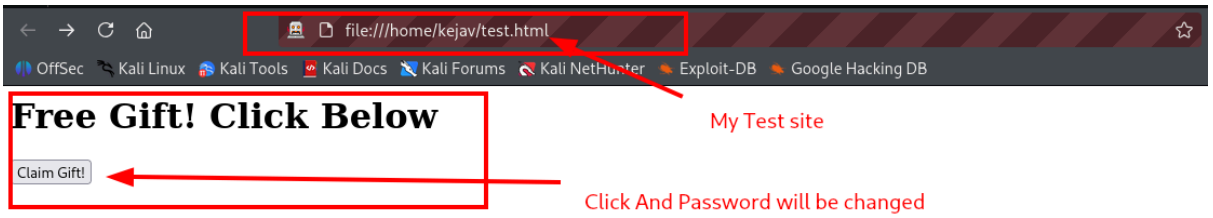


4.Cookie without SameSite Attribute

URL:http://127.0.0.1:42001/vulnerabilities/csrf/
Risk: Low
CWE ID:1275
WASC ID:13
Description:A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks.
Solution: Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies[17].
Payload: This is the simple code of exploiting CSRF.

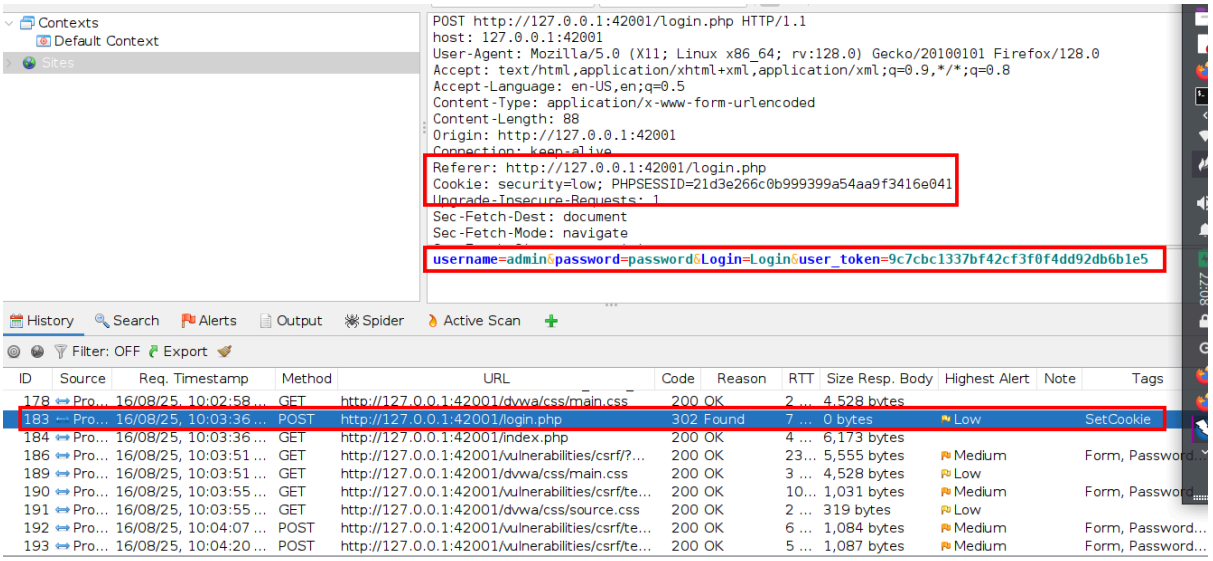
```
GNU nano 8.4                test.html
<html>
<body>
 <h1>Free Gift! Click Below</h1>
 <form action="http://127.0.0.1:42001/vulnerabilities/csrf/" method="GET">
  <input type="hidden" name="password_new" value="hacked123">
  <input type="hidden" name="password_conf" value="hacked123">
  <input type="hidden" name="Change" value="Change">
  <input type="submit" value="Claim Gift!">
 </form>
</body>
</html>
```

This is the preview of My test page payload.

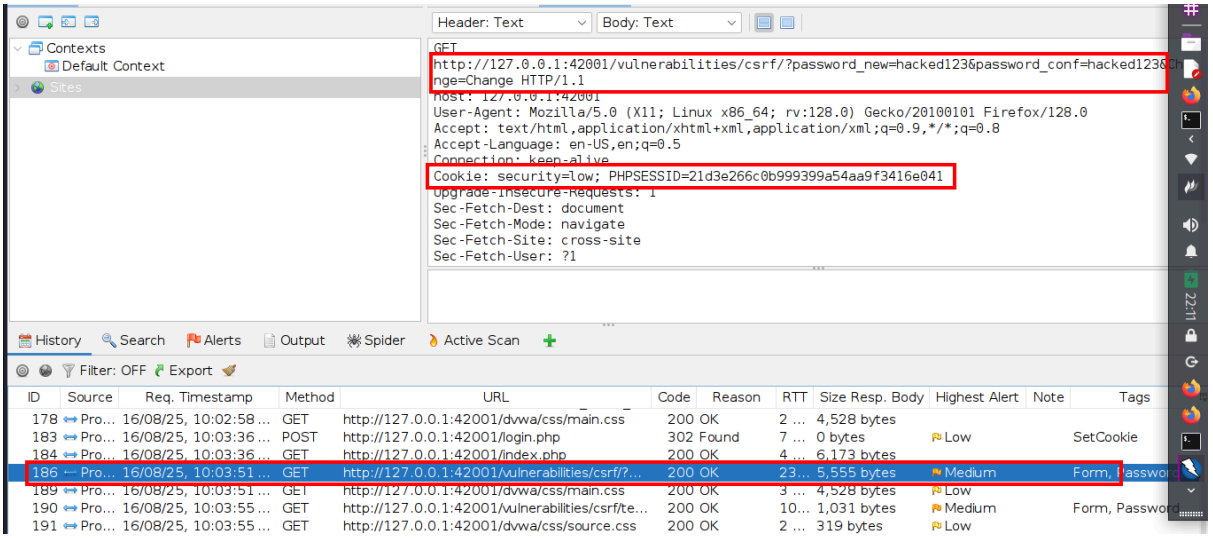


POC of Misconfiguration SameSite Attribute

Before Clicking on Claim gift Password and Cookie info:



After Clicking on Claim gift Password and Cookie info:

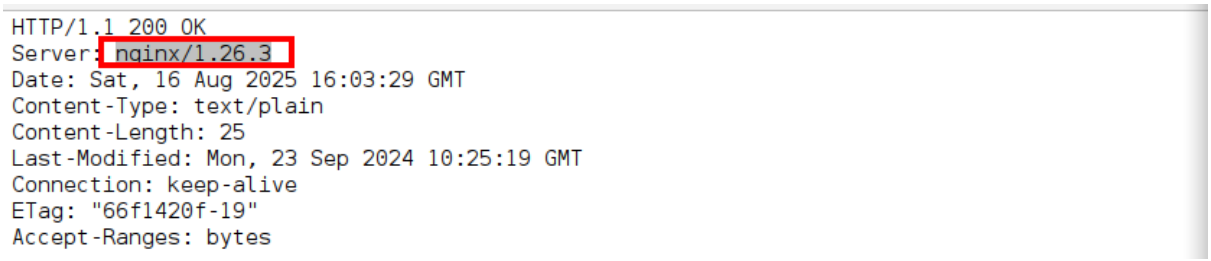


Note:This vulnerability known as Cross Site Request Forgery(CSRF) which is occurs by misconfiguration SameSite Attribute

5.Server Leaks Version Information via "Server" HTTP Response Header Field

URL:http://127.0.0.1:42001/robots.txt
Risk: Low
CWE ID:497
WASC ID:13
Description: The web/application server is leaking version information via the "Server" HTTP response header. Access to such information may facilitate attackers identifying other vulnerabilities your web/application server is subject to.
Solution: Ensure that your web server, application server, load balancer, etc. is configured to suppress the "Server" header or provide generic details[18].

Payload: This Info is just getting by checking Headers
POC of Information Discloser:



6.X-Content-Type-Options Header Missing

URL:http://127.0.0.1:42001/robots.txt
Risk: Low
CWE ID:693
WASC ID:15
Description: The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.This issue still applies to error type pages (401, 403, 500, etc.)[19] as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.At "High" threshold this scan rule will not alert on client or server error responses.
Solution: Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.
If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.

Conclusion:

The findings highlight how OWASP ZAP helps identify real-world misconfigurations and security weaknesses in web applications. Even common issues like missing headers or improper cookie attributes can open doors to attacks such as XSS, CSRF, clickjacking, or information disclosure. By using ZAP during development and testing, organizations can detect and patch these vulnerabilities early, strengthening overall application security.