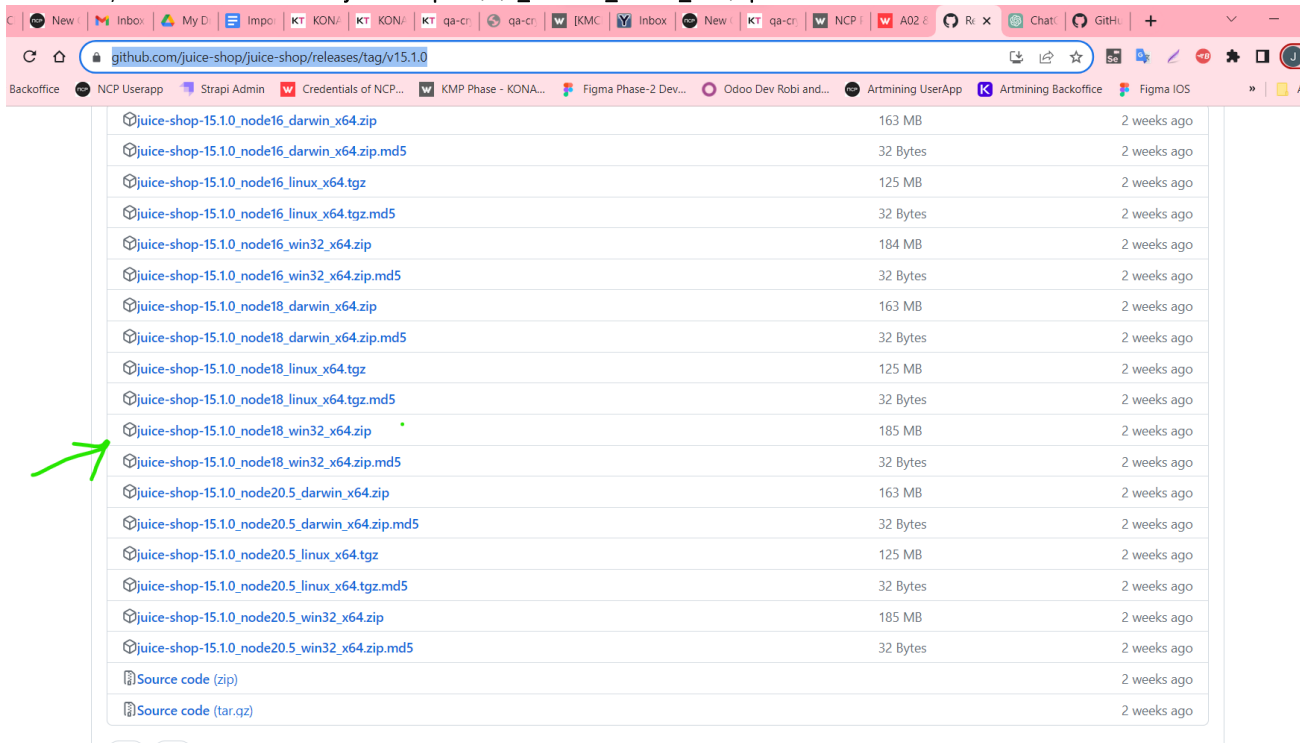


Top 1: Hands-on

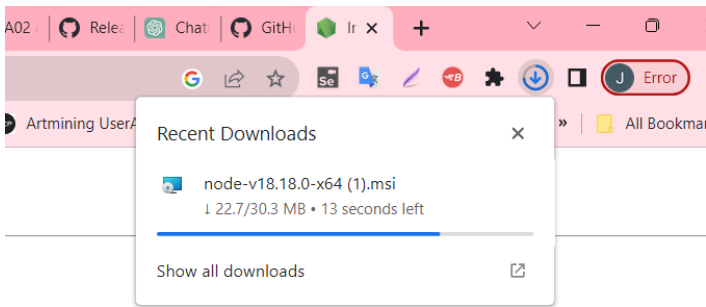
Step1: Deploy a vulnerable site on windows pc: (<https://github.com/juice-shop/juice-shop#packaged-distributions>)

1. Download juice-shop-<version>_<node-version>_<os>_x64.zip (or .tgz) attached to latest release <https://github.com/juice-shop/juice-shop/releases/tag/v15.1.0>.
In our case, We have downloaded **juice-shop-15.1.0_node18_win32_x64.zip**



github.com/juice-shop/juice-shop/releases/tag/v15.1.0		
juice-shop-15.1.0_node16_darwin_x64.zip	163 MB	2 weeks ago
juice-shop-15.1.0_node16_darwin_x64.zip.md5	32 Bytes	2 weeks ago
juice-shop-15.1.0_node16_linux_x64.tgz	125 MB	2 weeks ago
juice-shop-15.1.0_node16_linux_x64.tgz.md5	32 Bytes	2 weeks ago
juice-shop-15.1.0_node16_win32_x64.zip	184 MB	2 weeks ago
juice-shop-15.1.0_node16_win32_x64.zip.md5	32 Bytes	2 weeks ago
juice-shop-15.1.0_node18_darwin_x64.zip	163 MB	2 weeks ago
juice-shop-15.1.0_node18_darwin_x64.zip.md5	32 Bytes	2 weeks ago
juice-shop-15.1.0_node18_linux_x64.tgz	125 MB	2 weeks ago
juice-shop-15.1.0_node18_linux_x64.tgz.md5	32 Bytes	2 weeks ago
juice-shop-15.1.0_node18_win32_x64.zip	185 MB	2 weeks ago
juice-shop-15.1.0_node18_win32_x64.zip.md5	32 Bytes	2 weeks ago
juice-shop-15.1.0_node20.5_darwin_x64.zip	163 MB	2 weeks ago
juice-shop-15.1.0_node20.5_darwin_x64.zip.md5	32 Bytes	2 weeks ago
juice-shop-15.1.0_node20.5_linux_x64.tgz	125 MB	2 weeks ago
juice-shop-15.1.0_node20.5_linux_x64.tgz.md5	32 Bytes	2 weeks ago
juice-shop-15.1.0_node20.5_win32_x64.zip	185 MB	2 weeks ago
juice-shop-15.1.0_node20.5_win32_x64.zip.md5	32 Bytes	2 weeks ago
Source code (zip)		2 weeks ago
Source code (tar.gz)		2 weeks ago

2. Setup Node JS. Version should be similar which mentioned in the zip file. Our zip file name was juice-shop-15.1.0_node18_win32_x64.zip so we have downloaded the node v18.18.0. Verify the node version after setup. Command: node -v



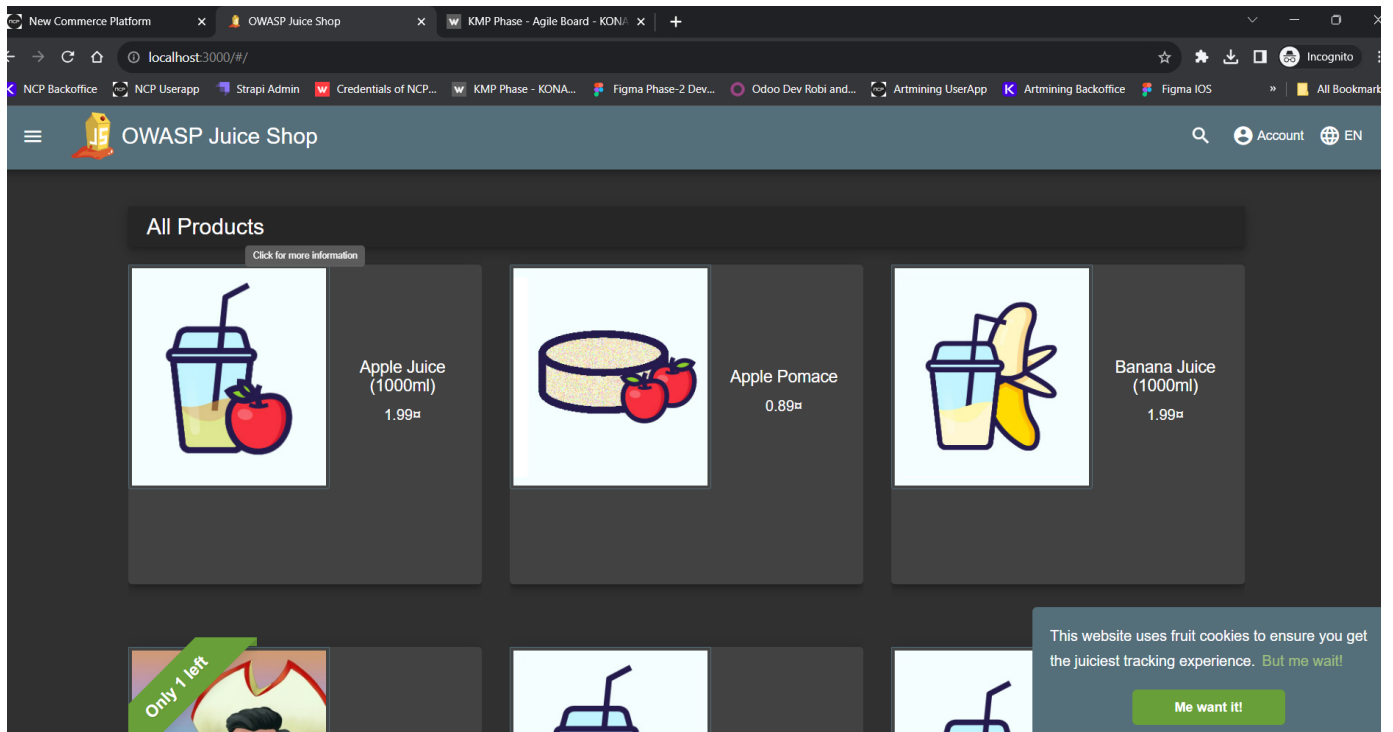
```
Microsoft Windows [Version 10.0.22000.2416]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jayed.ibrahim>node -v
v18.18.0

C:\Users\jayed.ibrahim>npm -v
9.8.1

C:\Users\jayed.ibrahim>
```

3. Go into the unzipped folder with `cd juice-shop`
4. Run `npm install` (only has to be done before first start or when you change the source code)
5. Run `npm start`
6. Browse to <http://localhost:3000>



Or

Docker Steps:

step1: Install Docker

step2: Run `docker pull bkimminich/juice-shop`

step3: Run `docker run --rm -p 3000:3000 bkimminich/juice-shop`

step4: Browse to <http://localhost:3000> (on macOS and Windows browse to <http://192.168.99.100:3000> if you are using docker-machine instead of the native docker installation)

Step 2: Scan with ZAP

2.1 Manual Scan


1. Download ZAP from <https://www.zaproxy.org/download/> and setup in your windows pc.



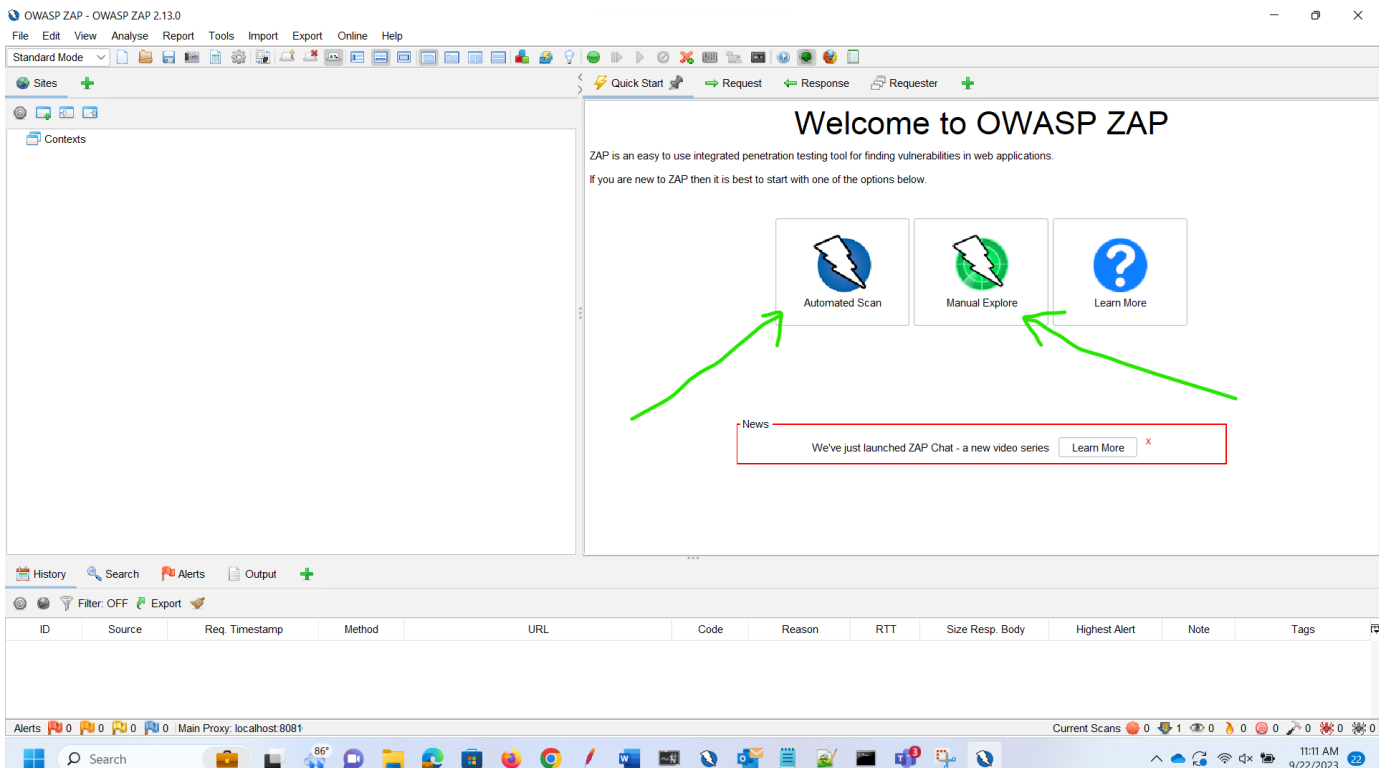
Download ZAP

- Checksums for all of the ZAP downloads are maintained on the [2.13.0 Release Page](#) and in the relevant [version files](#).
- As with all software we strongly recommend that ZAP is only installed and used on operating systems and JREs that are fully patched and actively maintained.

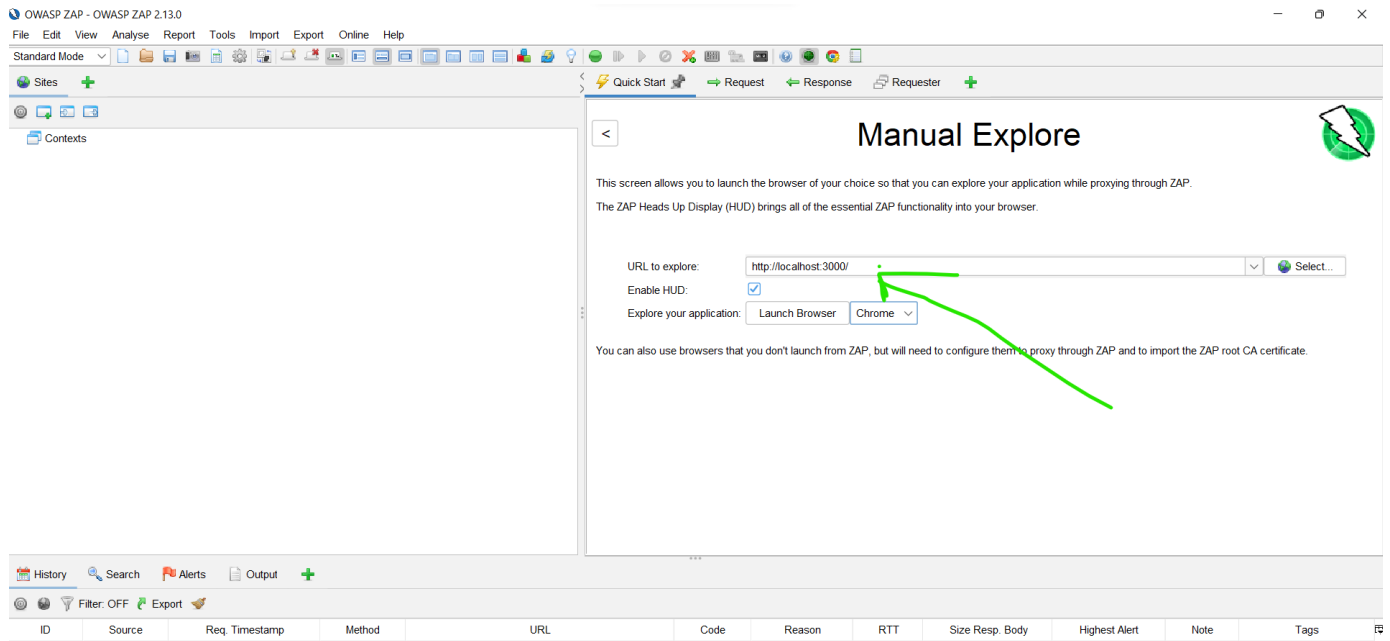
ZAP 2.13.0

 Windows (64) Installer	195 MB	Download
Windows (32) Installer	195 MB	Download
Linux Installer	199 MB	Download
Linux Package	196 MB	Download
macOS (Intel - amd64) Installer	224 MB	Download

2. Open ZAP and try to Manual Scan first



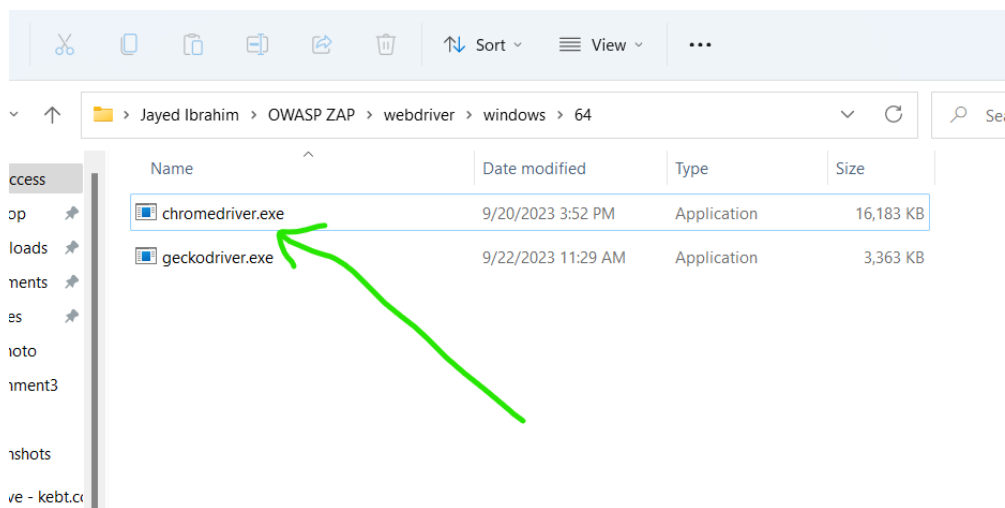
3. Click on Manual Explorer button and point juice shop URL there.



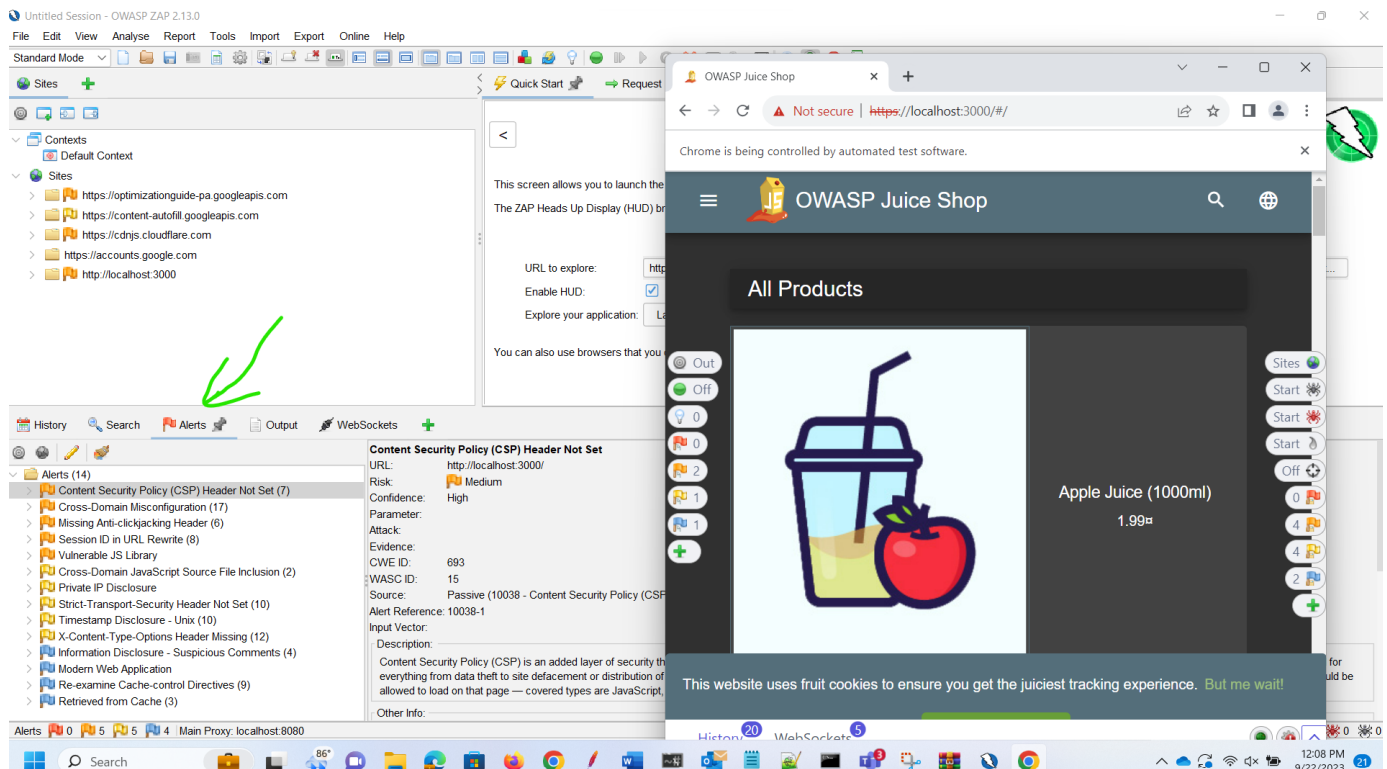
Problem: If got any error for launching the browser then cross check your browser version and browser driver version.

Solution:

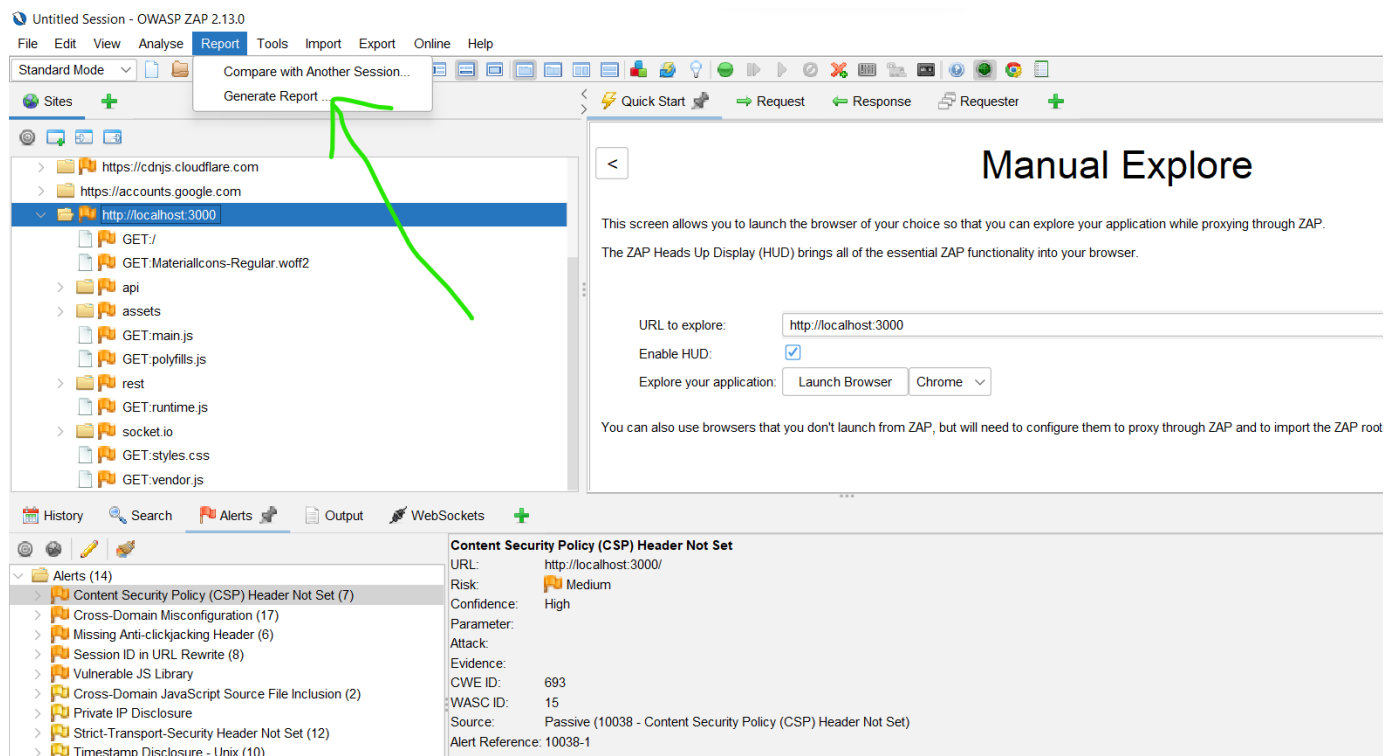
1. In ZAP go to Help > Support Info ...
2. Click on the Open ZAP Home button
3. Close ZAP (the Explorer directory should stay open)
4. Go to the webdriver directory and replace the correct version of webdriver there (Download correct version of webdriver from internet)
5. Restart ZAP



4. After successfully launching the browser, now browse different pages, ZAP will be scan in the background. Go to Alert Section for checking the vulnerabilities.



5. Now Generate the report from the top-menu section.



ZAP Scanning Report

Generated with ZAP on Fri 22 Sept 2023, at 12:10:50

ZAP Version: 2.13.0

Contents

- [About this report](#)
 - [Report parameters](#)
- [Summaries](#)
 - [Alert counts by risk and confidence](#)
 - [Alert counts by site and risk](#)
 - [Alert counts by alert type](#)
- [Alerts](#)
 - [Risk=Medium, Confidence=High \(2\)](#)
 - [Risk=Medium, Confidence=Medium \(3\)](#)
 - [Risk=Low, Confidence=High \(1\)](#)
 - [Risk=Low, Confidence=Medium \(3\)](#)
 - [Risk=Low, Confidence=Low \(1\)](#)
 - [Risk=Informational, Confidence=Medium \(2\)](#)

2.1 Automatic Scan

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 given permission to test.

URL to attack:

Use traditional spider: ☒

Use ajax spider: ☒ with

Progress:

Actively scanning (attacking) the URLs discovered by the spider(s)

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
4.762	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%3Bsleep+1.0%3B&M	400	Bad Request	20 ms	230 bytes	41 bytes
4.763	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%3Bsleep+1.0%3B&M	400	Bad Request	15 ms	230 bytes	41 bytes
4.764	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%22%2Bsleep+1.0%3B&M	400	Bad Request	0 ms	230 bytes	41 bytes
4.765	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%22%2Bsleep+1.0%3B&M	400	Bad Request	0 ms	230 bytes	41 bytes
4.766	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%22%2Bsleep+1.0%3B&M	400	Bad Request	5 ms	230 bytes	41 bytes
4.767	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%22%2Bsleep+1.0%3B&M	400	Bad Request	5 ms	230 bytes	41 bytes
4.768	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%27%2Bsleep+1.0%3B&M	400	Bad Request	0 ms	230 bytes	41 bytes
4.769	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%27%2Bsleep+1.0%3B&M	400	Bad Request	0 ms	230 bytes	41 bytes
4.770	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%27%2Bsleep+1.0%3B&M	400	Bad Request	0 ms	230 bytes	41 bytes
4.771	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=4%27%2Bsleep+1.0%3B&M	400	Bad Request	0 ms	230 bytes	41 bytes
4.772	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=type+%26SYSTEMERR	400	Bad Request	5 ms	230 bytes	41 bytes
4.773	9/22/23, 12:19:03 PM	9/22/23, 12:19:03 PM	POST	http://localhost:3000/socket.io/?IO=type+%26SYSTEMERR	400	Bad Request	5 ms	230 bytes	41 bytes

We can scan also scan in browser headless mode and get the report from ZAP.

Alert counts by alert type

This table shows the number of alerts of each alert type, together with the alert type's risk level.

(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

Alert type	Risk	Count
Cloud Metadata Potentially Exposed	High	1 (6.2%)
SQL Injection - SQLite	High	1 (6.2%)
Content Security Policy (CSP) Header Not Set	Medium	54 (337.5%)
Cross-Domain Misconfiguration	Medium	53 (331.2%)
Missing Anti-clickjacking Header	Medium	42 (262.5%)
Session ID in URL Rewrite	Medium	169 (1,056.2%)
Vulnerable JS Library	Medium	1 (6.2%)
Cross-Domain JavaScript Source File Inclusion	Low	6 (37.5%)
Private IP Disclosure	Low	1 (6.2%)
Strict-Transport-Security Header Not Set	Low	2 (12.5%)
Timestamp Disclosure - Unix	Low	5 (31.2%)
X-Content-Type-Options Header Missing	Low	169 (1,056.2%)
Information Disclosure - Suspicious Comments	Informational	4 (25.0%)
Modern Web Application	Informational	4 (25.0%)
Retrieved from Cache	Informational	93 (581.2%)
User Agent Fuzzer	Informational	127 (793.8%)
Total		16

Cross-Domain Misconfiguration:

The screenshot displays the OWASP ZAP 2.13.0 interface. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Import, Export, Online, and Help. The main toolbar contains icons for Standard Mode, Sites, Quick Start, Request, Response, and Requester. The left sidebar shows a tree view with Contexts (Default Context) and Sites. The central pane displays an HTTP response with headers: HTTP/1.1 200 OK, Access-Control-Allow-Origin: *, X-Content-Type-Options: nosniff, X-Frame-Options: SAMEORIGIN, Feature-Policy: payment 'self', X-Recruiting: /#/jobs, Accept-Ranges: bytes, Cache-Control: public, max-age=0, and Last-Modified: Wed, 20 Sep 2023 04:41:08 GMT. A green arrow points to the 'Access-Control-Allow-Origin: *' header. The bottom pane shows a list of alerts, with 'Cross-Domain Misconfiguration (63)' selected. The details pane for this alert shows the URL 'http://localhost:3000/runtime.js', Risk 'Medium', Confidence 'Medium', and a description: 'Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server'. Other information includes the attack type 'Access-Control-Allow-Origin: *', CWE ID '264', WASC ID '14', and a reference to 'https://vuln.cat.fortify.com/en/detail?id=desc.config.dotnet.html5_overly_permissive_cors_policy'.

Problem Details:

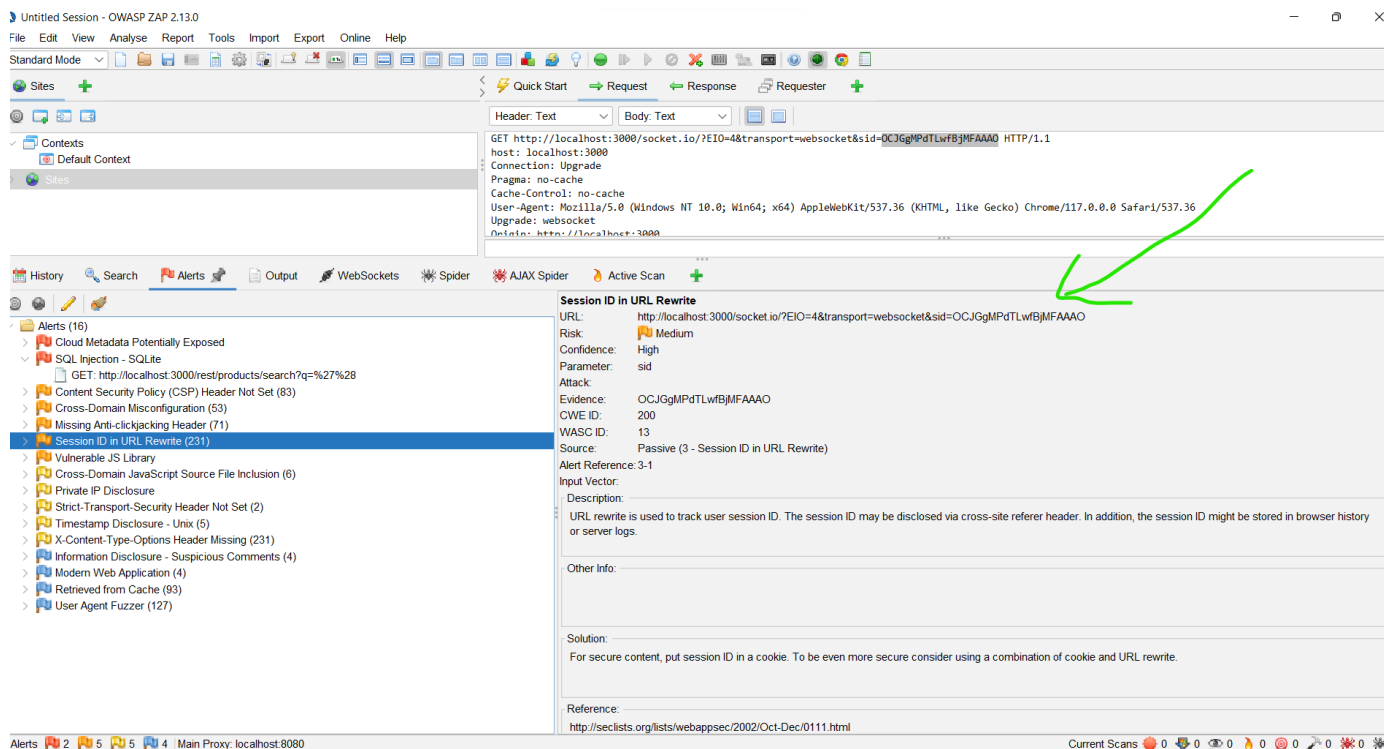
Web browser data loading may be possible, due to a Cross Origin Resource Sharing (CORS) misconfiguration on the web server.

The CORS misconfiguration on the web server permits cross-domain read requests from arbitrary third party domains, using unauthenticated APIs on this domain. Web browser implementations do not permit arbitrary third parties to read the response from authenticated APIs, however. This reduces the risk somewhat. This misconfiguration could be used by an attacker to access data that is available in an unauthenticated manner, but which uses some other form of security, such as IP address white-listing.

Solution:

Ensure that sensitive data is not available in an unauthenticated manner (using IP address white-listing, for instance). Configure the "Access-Control-Allow-Origin" HTTP header to a more restrictive set of domains, or remove all CORS headers entirely, to allow the web browser to enforce the Same Origin Policy (SOP) in a more restrictive manner.

Session ID Re-write:



Problem Details:

Storing session IDs in the URL can introduce security risks and should generally be avoided unless you have strong reasons for doing so. Here are some of the risks associated with using session IDs in URLs:

1. **Session Fixation Attacks:** Session fixation is a type of attack where an attacker can force a user to use a specific session ID. If session IDs are exposed in URLs, they become easier for attackers to manipulate. An attacker could trick a user into using a session ID that the attacker controls, potentially compromising the user's session.
2. **Information Leakage:** URLs are often logged in various places, including server logs, browser history, and referrer logs. If session IDs are included in URLs, they may be exposed in these logs, leading to the potential exposure of sensitive user data.
3. **Bookmarks and Shared Links:** Users may bookmark URLs that contain session IDs, and these bookmarks may include session-specific data. If users share these URLs, the session data can be inadvertently shared as well, potentially leading to information leakage.
4. **Search Engines:** Search engine crawlers can follow and index URLs with session IDs, leading to the unintentional indexing of session-specific content. This can expose sensitive data in search engine results.
5. **User Experience:** Including session IDs in URLs can make URLs less user-friendly and aesthetically pleasing. It can also lead to issues with copy-pasting URLs, as users might accidentally include session IDs, resulting in broken links.
6. **Session Hijacking:** If session IDs are exposed in URLs, they become easier to steal. Malicious actors who gain access to a user's URL, either through browser history or other means, can use that session ID to impersonate the user and gain unauthorized access to their session.

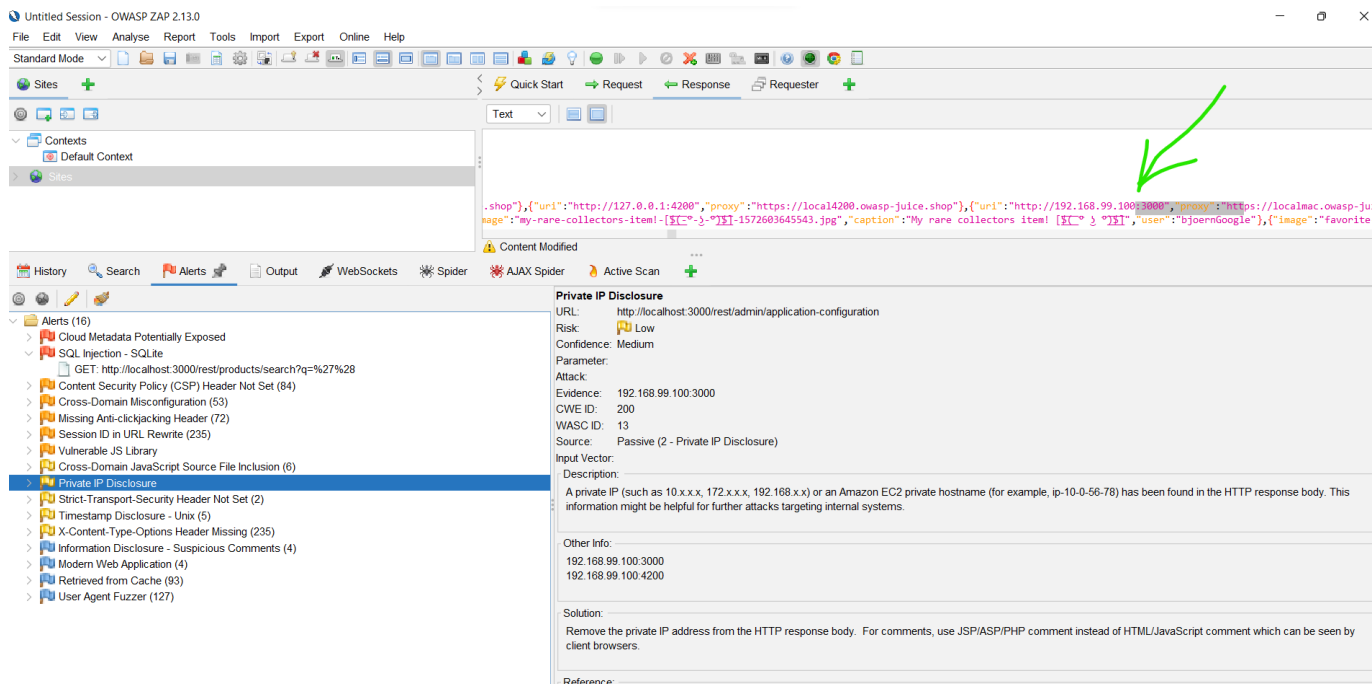
Solution Details:

Given these risks, it's generally recommended to use more secure methods for managing session state, such as:

1. **Cookies:** Storing session IDs in cookies is a common and more secure approach. Cookies can be marked as "HttpOnly" and "Secure" to prevent JavaScript access and ensure that they are transmitted over secure connections (HTTPS).
2. **Server-Side Session Management:** Store session data on the server rather than in URLs. Most web frameworks and programming languages provide built-in mechanisms for managing sessions securely.
3. **Token-Based Authentication:** If you need to maintain user state for APIs or single-page applications, consider using token-based authentication, such as JSON Web Tokens (JWTs), instead of session IDs in URLs.

While there may be cases where using session IDs in URLs is unavoidable or necessary, it's crucial to implement strict security measures and consider the associated risks carefully. Security should always be a top priority when handling user sessions and sensitive data in web applications.

Private IP Disclosure:



Problem Details:

A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.

Solution:

To address private IP disclosure issues and mitigate potential security risks, you can take several steps:

1. Implement Proper Error Handling and Logging:
 - Ensure that error messages and debug information do not include private IP addresses. Review and sanitize error messages before they are exposed to users or logged.
 - Implement comprehensive error handling to prevent stack traces or internal information from being displayed to users.
2. Web Application Firewalls (WAFs):
 - Use a Web Application Firewall to filter and block requests or responses that contain private IP addresses.
 - Configure the WAF to detect and block any attempts to exploit private IP address disclosures.
3. Regular Security Audits and Testing:
 - Conduct security audits and penetration testing on your web applications to identify and rectify any instances where private IP addresses are exposed.
 - Use automated security scanning tools that can identify potential vulnerabilities related to IP disclosure.
4. Security Headers:
 - Implement security headers such as Content Security Policy (CSP) to control which domains and sources are allowed to load content on your web pages. This can help prevent third-party scripts from accessing private IP addresses.
5. Review Third-Party Services:
 - If your application relies on third-party services or APIs, ensure that they do not expose private IP addresses in their responses. Review their documentation and security practices.
6. Data Sanitization:
 - When processing user input or data from untrusted sources, ensure that any IP addresses are properly sanitized and validated to prevent the inclusion of private IPs.
7. Reverse Proxy and Load Balancer Configuration:
 - If you use a reverse proxy or load balancer in your infrastructure, configure it to ensure that private IP addresses are not exposed in HTTP headers or responses.

Timestamp Disclosure:

