# 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.

## What is the difference between HTTP and WebSockets?

Most communication between web browsers and web sites uses HTTP. With HTTP, the client sends a request and the server returns a response. Typically, the response occurs immediately, and the transaction is complete. Even if the network connection stays open, this will be used for a separate transaction of a request and a response.

Some modern web sites use WebSockets. WebSocket connections are initiated over HTTP and are typically long-lived. Messages can be sent in either direction at any time and are not transactional in nature. The connection will normally stay open and idle until either the client or the server is ready to send a message.

WebSockets are particularly useful in situations where low-latency or server-initiated messages are required, such as real-time feeds of financial data.

## How are WebSocket connections established?

WebSocket connections are normally created using client-side JavaScript like the following:

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

#### Note

The wss protocol establishes a WebSocket over an encrypted TLS connection, while the ws protocol uses an unencrypted connection.

To establish the connection, the browser and server perform a WebSocket handshake over HTTP. The browser issues a WebSocket handshake request like the following:

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

If the server accepts the connection, it returns a WebSocket handshake response like the following:

HTTP/1.1 101 Switching Protocols

Connection: Upgrade

Upgrade: websocket

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

At this point, the network connection remains open and can be used to send WebSocket messages in either direction.

#### Note

Several features of the WebSocket handshake messages are worth noting:

* The Connection and Upgrade headers in the request and response indicate that this is a WebSocket handshake.
* The Sec-WebSocket-Version request header specifies the WebSocket protocol version that the client wishes to use. This is typically 13.
* The Sec-WebSocket-Key request header contains a Base64-encoded random value, which should be randomly generated in each handshake request.
* The 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. This is done to prevent misleading responses resulting from misconfigured servers or caching proxies.

## What do WebSocket messages look like?

Once a WebSocket connection has been established, messages can be sent asynchronously in either direction by the client or server.

A simple message could be sent from the browser using client-side JavaScript like the following:

ws.send("hello mrwebsecure");

In principle, WebSocket messages can contain any content or data format. In modern applications, it is common for JSON to be used to send structured data within WebSocket messages.

For example, a chat-bot application using WebSockets might send a message like the following:

{"user":"Hal Pline","content":"I wanted to be a Playstation growing up”}

## Manipulating WebSocket traffic

Finding WebSockets security vulnerabilities generally involves manipulating them in ways that the application doesn't expect. You can do this using Burp Suite.

You can use Burp Suite to:

* Intercept and modify WebSocket messages.
* Replay and generate new WebSocket messages.
* Manipulate WebSocket connections.

### Intercepting and modifying WebSocket messages

You can use Burp Proxy to intercept and modify WebSocket messages, as follows:

* Open Burp's browser.
* Browse to the application function that uses WebSockets. You can determine that WebSockets are being used by using the application and looking for entries appearing in the WebSockets history tab within Burp Proxy.
* In the Intercept tab of Burp Proxy, ensure that interception is turned on.
* When a WebSocket message is sent from the browser or server, it will be displayed in the Intercept tab for you to view or modify. Press the Forward button to forward the message.

### Replaying and generating new WebSocket messages

As well as intercepting and modifying WebSocket messages on the fly, you can replay individual messages and generate new messages. You can do this using Burp Repeater:

* In Burp Proxy, select a message in the WebSockets history, or in the Intercept tab, and choose "Send to Repeater" from the context menu.
* In Burp Repeater, you can now edit the message that was selected, and send it over and over.
* You can enter a new message and send it in either direction, to the client or server.
* In the "History" panel within Burp Repeater, you can view the history of messages that have been transmitted over the WebSocket connection. This includes messages that you have generated in Burp Repeater, and also any that were generated by the browser or server via the same connection.
* If you want to edit and resend any message in the history panel, you can do this by selecting the message and choosing "Edit and resend" from the context menu.

### Manipulating WebSocket connections

As well as manipulating WebSocket messages, it is sometimes necessary to manipulate the WebSocket handshake that establishes the connection.

There are various situations in which manipulating the WebSocket handshake might be necessary:

* It can enable you to reach more attack surface.
* Some attacks might cause your connection to drop so you need to establish a new one.
* Tokens or other data in the original handshake request might be stale and need updating.

You can manipulate the WebSocket handshake using Burp Repeater:

* Send a WebSocket message to Burp Repeater.
* In Burp Repeater, click on the pencil icon next to the WebSocket URL. This opens a wizard that lets you attach to an existing connected WebSocket, clone a connected WebSocket, or reconnect to a disconnected WebSocket.
* If you choose to clone a connected WebSocket or reconnect to a disconnected WebSocket, then the wizard will show full details of the WebSocket handshake request, which you can edit as required before the handshake is performed.
* When you click "Connect", Burp will attempt to carry out the configured handshake and display the result. If a new WebSocket connection was successfully established, you can then use this to send new messages in Burp Repeater.

## WebSockets security vulnerabilities

In principle, practically any web security vulnerability might arise in relation to WebSockets:

* User-supplied input transmitted to the server might be processed in unsafe ways, leading to vulnerabilities such as SQL injection or XML external entity injection.
* Some blind vulnerabilities reached via WebSockets might only be detectable using out-of-band (OAST) techniques.
* If attacker-controlled data is transmitted via WebSockets to other application users, then it might lead to XSS or other client-side vulnerabilities.

### Manipulating WebSocket messages to exploit vulnerabilities

The majority of input-based vulnerabilities affecting WebSockets can be found and exploited by tampering with the contents of WebSocket messages.

For example, suppose a chat application uses WebSockets to send chat messages between the browser and the server. When a user types a chat message, a WebSocket message like the following is sent to the server:

{"message":"Hello mrwebsecure"}

The contents of the message are transmitted (again via WebSockets) to another chat user, and rendered in the user's browser as follows:

<td>Hello mrwebsecure</td>

In this situation, provided no other input processing or defenses are in play, an attacker can perform a proof-of-concept XSS attack by submitting the following WebSocket message:

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

### Manipulating the WebSocket handshake to exploit vulnerabilities

Some WebSockets vulnerabilities can only be found and exploited by manipulating the WebSocket handshake. These vulnerabilities tend to involve design flaws, such as:

* Misplaced trust in HTTP headers to perform security decisions, such as the X-Forwarded-For header.
* Flaws in session handling mechanisms, since the session context in which WebSocket messages are processed is generally determined by the session context of the handshake message.
* Attack surface introduced by custom HTTP headers used by the application.

### Using cross-site WebSockets to exploit vulnerabilities

Some WebSockets security vulnerabilities arise when an attacker makes a cross-domain WebSocket connection from a web site that the attacker controls. This is known as a cross-site WebSocket hijacking attack, and it involves exploiting a cross-site request forgery (CSRF) vulnerability on a WebSocket handshake. The attack often has a serious impact, allowing an attacker to perform privileged actions on behalf of the victim user or capture sensitive data to which the victim user has access.

# Cross-site WebSocket hijacking

In this section, we'll explain cross-site WebSocket hijacking (CSWSH), describe the impact of a compromise, and spell out how to perform a cross-site WebSocket hijacking attack.

## What is cross-site WebSocket hijacking?

Cross-site WebSocket hijacking (also known as cross-origin WebSocket hijacking) involves a cross-site request forgery (CSRF) vulnerability on a WebSocket handshake. It 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. This means that, unlike regular CSRF, the attacker gains two-way interaction with the compromised application.

## What is the impact of cross-site WebSocket hijacking?

A successful cross-site WebSocket hijacking attack will often enable an attacker to:

* **Perform unauthorized actions masquerading as the victim user.** As with regular CSRF, the attacker can send arbitrary messages to the server-side application. If the application uses client-generated WebSocket messages to perform any sensitive actions, then the attacker can generate suitable messages cross-domain and trigger those actions.
* **Retrieve sensitive data that the user can access.** Unlike with regular CSRF, cross-site WebSocket hijacking gives the attacker two-way interaction with the vulnerable application over the hijacked WebSocket. If the application uses server-generated WebSocket messages to return any sensitive data to the user, then the attacker can intercept those messages and capture the victim user's data.

## Performing a cross-site WebSocket hijacking attack

Since a cross-site WebSocket hijacking attack is essentially a CSRF vulnerability on a WebSocket handshake, the first step to performing an attack is to review the WebSocket handshakes that the application carries out and determine whether they are protected against CSRF.

In terms of the normal conditions for CSRF attacks, you typically need to find a handshake message that relies solely on HTTP cookies for session handling and doesn't employ any tokens or other unpredictable values in request parameters.

For example, the following WebSocket handshake request is probably vulnerable to CSRF, because the only session token is transmitted in a cookie:

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

#### Note

The Sec-WebSocket-Key header contains a random value to prevent errors from caching proxies, and is not used for authentication or session handling purposes.

If the WebSocket handshake request is vulnerable to CSRF, then an attacker's web page can perform a cross-site request to open a WebSocket on the vulnerable site. What happens next in the attack depends entirely on the application's logic and how it is using WebSockets. The attack might involve:

* Sending WebSocket messages to perform unauthorized actions on behalf of the victim user.
* Sending WebSocket messages to retrieve sensitive data.
* Sometimes, just waiting for incoming messages to arrive containing sensitive data

## 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.