Security aspects of full-duplex web interactions and WebSockets

Yang Fu
iTEAM Research Institute
Universitat Politècnica de Valencia
València, Spain
yanfu@docto.upv.es

Marisol García-Valls
Department of Communications
Universitat Politècnica de Valencia
València, Spain
mgvalls@dcom.upv.es

***Abstract*—Integrating full-duplex low-latency communication middleware inside an HTTP based client-server interaction enables the development of powerful and highly-interactive web applications. Among the existing client-server communication protocols, there is WebSockets that allows the browser and the server backend to open a lively session interaction independent from the main HTTP application communication. There are different implementations of this full-duplex protocol, though per se, they do not offer the security protections required in cyber-physical systems (CPS). As a result, low-latency comes at the price of explicitly adding the needed security hooks to protect the application. This paper contributes an analysis of WebSockets technology from the point of view of its architecture and vulnerabilities and its applicability to CPS. We present the some selected implementations of the protocol that have become de-facto standard implementations given their broad usage; and we describe the vulnerabilities that they integrate –given their architecture– and how to overcome them. The paper provides an exemplification of a step by step communication between server and client on a full duplex WebSocket communication and shows how the associated vulnerabilities are exploited in a real use case scenario.**

***Index Terms*—Web, web communication, WebSockets, security, time sensitive systems**

I. Introduction

As new and more powerful application paradigms populate the society, their demand over the acquisition and processing of data in real time is constantly increasing. Solely using a data base that contains a rather static view on the used data is no longer sufficient in the modern world. However, HTTP (HyperText Transfer Protocol) did a great job in fulfilling the needs of the static unidirectional request-response mechanism of traditional communications from server to client over the Internet. This model, however, implies that each client request always gets a server response, even if there is no data to be sent back to the client. The advantage of this model is that it adjusts to the long-used stateless transaction model: under a static connection, the communication could be easily restored if it would break for whatever reason.

However, the Internet of Things (IoT) and Cyber-Physical Systems (CPS) applications have much more strict requirements, especially in what concerns real-time behaviour [1]. When it comes to the development of real-time applications, a pure HTTP-like connection will incurr in large overheads and lead to longer latencies; the excess of request-response messages and accompanying headers traveling back and forth makes this protocol quite inefficient for time-sensitive sytems.

To overcome this problem, researchers brought to the table a solution named *long polling*. Long polling reduces the interactions and associated exchange messages as much as possible; it allows the TCP socket connection to last longer to compile more data packets that can be sent back to the client all together in a single response. Meanwhile, it also helps to mitigate the return of empty response that yields missing data, since the server can respond when there are some actual data that need to be sent back to the client.

Although the long polling technique helps to ease the situation, it does not fully solve the problem: long latencies are still considerable. As a result, time-sensitive applications having strict requirements over the speed of data transfer and processing are not well addressed by this technology; as a result, it is likely that user-related time deadlines are missed, yielding the system to react late to the user requests for action.

As the web is progressively increasing its popularity as a means to establish easy and self-explainable client-server interactions, a number of contributions have appeared to improve the communication latencies. Contributions have searched to provide a real-time communication scheme that supports two-side exchanges and reduces the development learning curve. As a result, more efficient communication channels were designed to enable `direct` communication between client and server in a web context. This yielded to the appearance of Websockets as a new communication protocol designed especifically for web interactions.

This paper focuses on web-based full-duplex technologies by describing their inner workings; and then studying a subset of the most widely used implementations that enable the client and server sides of a web application to establish real-time parallel communication channels a part from the mainstream

This research was funded by: "*Design of services for resilient and real-time execution of social dispersed computing applications in cyber-physical domains*" funded by Generalitat Valenciana (Conselleria de Innovación, Universidades, Ciencia y Sociedad Digital), Spain, under grant No. AICO/2021/138; by grant PID2021-123168NB-I00 funded by ERDF *A way of making Europe* and MCIN/AEI/10.13039/501100011033; and by *Green Networks: Towards environmental-friendly 6G Networks* under grant TED2021-131387B-I00 funded by MCIN/AEI/10.13039/501100011033 and by the European Union NextGenerationEU/RTRP.

2023 20th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA) | 979-8-3503-1943-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/AICCSA59173.2023.10479302

HTTP conversation. The paper also contributes a discussion of the vulnerabilities that these technologies have as the price to pay for the real-time behavior. The paper lays down the fact that developing safe real-time web applications is not an easy task, as the potential vulnerabilities are extense. Nevertheless, this paper analyses how to overcome these to protect the communication, and therefore, the endpoints.

II. Background

In their origings, web protocols were long latency ones and offered a strict unidirectional communication channel between the endpoints.

Real-time technology for the web appeared long after the first web technologies were brought to life. Real-time web applications involve clients that receive constant and immediate data updates from the server side, as soon as these data modifications occur [2]. Examples of real-time web-based applications are multi-stackeholder video gaming and document editing; in these domains, whatever is done by a client arrives immediaty to the rest of clients involved in the application. First approaches to enable such a real-time model depended on technologies like AJAX that appeared in 2005 supporting bidirectional connections between server and client. Although AJAX augments the responsiveness of web applications, it was still far from properly addressing all the needs of real-time environments as it still provides a heavy latency communication model [3] also needing some plug-in installation that causes the server to incur in overheads. Moreover, the requirements of applications with respect to the amount of data that is consumed and produced has changed incredibly since the times of first AJAX appearance.

The most significant fact that enabled real-time communication in the Web was the release of HTML5 [4] and the WebSockets protocol [5]. Since then, a number of popular applicatons with real-time requirements have used this technology and some have even documented it, e.g., financial stock market applications [6]; timing display applications [7]; [9] and video streaming [10]. Also, in the recent years a number of web security contributions are appearing such as [12] on web server security or [11] for application level security [11]. All the above works have focused on the functional aspects of real-time web application design and development; whereas the security aspect initially remained in a second line.

As inherent to a highly connected domain like Internet and the Web, web applications are particularly vulnerable to cyberattacks. They are affected by the vulnerabilities introduced at different points of the software layers, from the application level down to the operating system drivers and networking software. Across the software tower, one finds a number of interpreted code that can access application level data, database information [18], or monitor the network traffic. By checking the top 10 security risks [13] of web environments, it can be seen that the potential effects of these risks horizontally threaten all application domains.

Since the revelations of some masive Intenet-traffic surveillance programs in 2015 [14], the perception of security in the Internet changed and the big technology companies became more aware of the importance of security in the actual design and deployment of web applications. Although large repositories of vulneralities are publically available and constantly updated, web contributions are mostly target functional-driven implementations where the analysis of their inherent security problems and solutions is left on a second line. Even though some studies have contributed analysis of the performance characteristics of using the different versions of HTTP and WebSockets [21] [23], their security vulnerabilities and the implications of these have been seldom considered. This paper is a contribution towards filling this gap.

III. WebSocket protocol

A. Direct web communication

In its conception, HTTP protocol was used as request-response mechanism working as follows. Firstly, a connection is opened; then, the request message is sent from client to server. After, the server returns a response message to the client; and eventually, the server sends an indication to the client that the connection is closed.

Obviously, HTTP protocol is a one direction communication. This implies that a client request gets a server response. This mechanism works well for maintream applications such as those involving exchange of text documents with some additional rich data like images, etc. However, when it comes to real time communication, e.g., video streaming or interactive web-based games, there is a considerable overhead problem that cannot be fulfilled solely with HTTP communication.

As a result, there are three main versions of HTTP: *polling*, *long polling*, and *streaming*. Here is how they work:

- *Polling* implies that each client request gets a server response. Each interaction requires that the connection is terminated.
- *Long polling* consists of holding the connection and the response message, such that the response message is returned whenever there are new data. Still the client needs to poll the server whenever it wishes to receive more data. The server always signals the client when the request is complete, even if there is no data to return.
- *Streaming* was incorporated in an additional header of HTTP version 1.1 that can be added to the request namely `Keep-Alive`. This header instructs to open an indefinite connection between client and server, and the server does not signal the client when the connection is complete so that the connection is left open. Still in this version, the header information is sent to the client in each message, and this may cost kilobytes of bandwidth due to the HTTP long headers.

B. The WebSockets protocol

WebSockets (WS) [5] were first introduced in 2010 and soon they gained a considerable support from the existing browsers. WS protocol helps with the former lack of real-time web communication capabilities of pure HTTP channels.

WebSockets introduced new ways of dealing with actual real-world applications. One of the major characteristics was the introduction of *full-duplex communication* channels over a single TCP connection. Moreover, WS also offered *compatibility with various programming languages* such as JavaScript or Python, among others, that was fundamental to gain popularity among different programming communities.

In 2011, the Internet Engineering Task Force (IETF) standardized WebSockets protocol as RFC 6455 [5]. Meanwhile, the World Wide Web Consortium (W3C) also standardized WebSockets API in Web IDL which helped to define the interfaces.

WebSockets versus HTTP. There are some similarities and differences between both protocols. First of all, both work at the application layer (layer 7) in OSI model. Also, they are above the TCP protocol at the transport layer (layer 4). To ensure compatibility between WebSockets and HTTP, the WebSockets protocol was required to work over the same ports as HTTP in RFC 6455. Moreover, the handshake in WebSockets uses the *upgrade header* derivative from HTTP protocol.

Unlike HTTP, WebSockets can achieve full duplex communication. Besides, WebSockets open the streams of messages based on TCP connection, while TCP handles the streams of data alone without the intrinsic concept of messages. The target of WebSockets protocol is to solve the problem of utilizing the message inefficiently without damaging the security of web.

Also, unlike HTTP/1.1 (synchronous) and HTTP/2 (asynchronous, while the server is only able to initiate streams in response to requests), the connection under WebSocket protocol is fully asynchronous. With this characteristic, both of the server and client sides can send data at any time without restrictions.

WebSockets connection protocol. The connection protocol is initiated upon a request, and followed by a handshake protocol that leaves the floor to the data exchange phase. The connection is initiated by either client or a server, and then it is established permanently, only terminating when one of the endpoints decides to end it. As long as the connection stays active, both of the terminals will keep communicating through the same channel. In general, WebSockets is known as a stateful protocol. If the connection between client and server has been established, this connection will keep active until it is closed by either client or server. Upon a connection termination is called for by either side, this connection is ready for stopping both terminal points.

Additionally, in WebSocket protocol, two different uniform resource identifier (URI) schemes are defined for distinct usage of unencrypted and encrypted connections, which are: WS (WebSocket) and WSS (WebSocket Secure).

A WebSocket connection starts with an HTTP handshake; then, it changes to use the WebSocket wire protocol. Therefore, some of the HTTP security mechanisms also apply to a WebSocket connection [5].

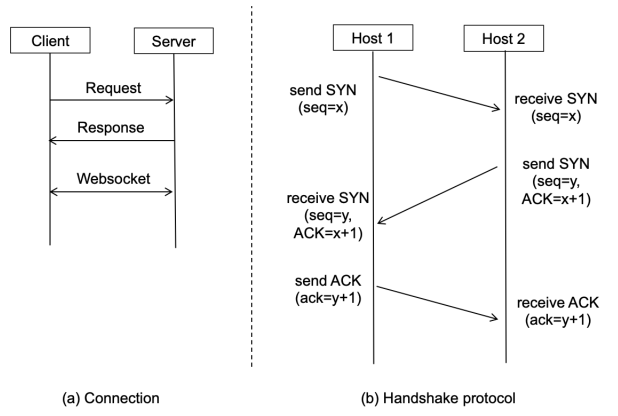


Fig. 1. Protocol phases

There are some implementations of WebSocket tunnels can be configured to use TLS/SSL HTTPS protocol. Also, some implementations identify WebSocket endpoints with a URL that means that security policies like SOP (Same Origin Policy) can be applied.

C. Application domains

Deciding when to use and when not to use WS protocol is key to the responsiveness and usability of the application. Applications where using WebSockets is a clear added value have the following characteristics:

- Client front-end and web server backend.
- Client-initiated and server-initiated communications.
- Timely requirements.
- Resource efficiency: connections are typically reused to decrease resource consumption in terms of memory, processor cycles, and reduce latency.

Typical examples of such applications are [22] distributed transactive systems such as trading systems; gaming; or conversational applications like chats.

Bitcoin trading is typically provided as a real-time web application where the data packets are continuously transmitted via the open connection to show the fluctuation of price.

Gaming applications naturally adopt the WebSockets protocol. Data are constantly flowing from the sever to the client machine and displayed on screen. During this process, the user interface retrieves data automatically and continuously refreshes the screen with new information. The same connection is used; no additional ones are needed.

Conversational applications are also prone to using Websockets. After establishing the connection via WebSocket, the client and server (or even peers) reuse the same WebSocket connection for sending and receiving the messages and one to one message transfers. It is worth mentioning that some conversational applications may also use peer-to-peer UDP-based communications.

However, it is not always recommended to use WebSocket. For instance, if a user wishes to retrieve old messages; or receive a data piece that s/he will only use once, HTTP protocol is a more suitable protocol to use. In unfrequent requests for stored data (as they are a simple and one-time occurrence), it is more efficient to request via HTTP.

D. Technology differences

This section briefly outlines some of the technology competitors to WebSockets, and highlights the pros and cons of this technology.

1) Socket, WebSockets and Socket.io: Essentially, *socket* is a transmission control layer protocol; *WebSocket* is an application layer protocol; and *Socket.IO* is a framework, essentially a wrapper to WebSocket protocol.

Sockets are the middleware abstraction layer for communication between the application layer and the TCP/IP protocol suite. It is a set of interfaces whose design lies on a facade mode, which hides the complex TCP/IP protocol family behind the Socket interface. The interface is a simple one enabling users to organize data conforming to the specified and agreed communication protocol.

WebSockets is a communication protocol of HTML5, allowing the server to actively push information to the client; and allowing the client to actively send information to the server.It becomes a true two-way, equal dialogue and falls within the category of a server push technology.

Socket.IO [8] is a framework that provides real-time, full-duplex communication between client and server. Version 4.0 has been released in March 2021, and it keeps to be a popular framework for low-latency web communication. Underneath Socket.IO architecture, there is a WebSocket protocol. As a result, the client will first try to establish a WebSocket connection; if it is not possible, the communication falls back to an HTTP long polling scheme.

Therefore, the properties of Socket.IO stem from the WebSocket protocol: it is a low-latency channel and full-duplex communication between client and server. Websocket is supported by virtually all modern browsers [17], so this backs up Socket.IO as well.

Socket.IO can be implemented by JavaScript, based on Node.js, supports WebSocket protocol for real-time communication, cross-platform open source framework, it includes client-side JavaScript and server-side Node.js. This means that Socket.io encapsulates WebSocket, a polling mechanism and other real-time communication methods into common interfaces and implements these real-time communication mechanisms on the server side.

2) Advantages and disadvantages: There is a constant growth in the number of users of WebSockets due to the significant benefits its offers:

- Full-duplex communication among server and client.
- The speed of sending or receiving data is much higher than that of the HTTP protocol.
- Communications under WebSocket allows requesting by different origins.
- WebSocket is compatible between different platforms such as website, local computer and mobile terminal.
- Compared with HTTP or HTTP long polling, WebSocket has significantly reduced the overhead (i.e. for HTTP might have 2000 bytes overhead while WebSocket only cost 2 bytes) [21].

In contrast, some disadvantages do not advise for using WebSockets in some application domains:

- It does not follow the same-origin policy (SOP) [15], so it can suffer security violations. SOP is a fundamental web application security model that controls the mutual access of two contexts depending on their origin.
- The web browser is required to use fully HTML5-compliant version coding.
- There is no edge caching provided in WebSocket.
- For the applications that do not require continous interactions, HTTP is much easier to be employed and implemented.

IV. Security analysis

A. Vulnerabilities

DoS Attacks. Denial-of-service (DoS) is a major attack that exhausts server resources, leaving the server useless as it cannot process further requests. The most common framework to DoS is to flood the server with a bulk of packets that request service of some kind. The server response becomes unusually slow and turns out to be useless. A common path to these attacks is to exploit machines infected with malware or botnet software. TCP SYN flood attack leverages the three-way handshake of TCP. The client will send SYN packets to the server before the handshake; and it will obtain a SYN/ACK response from the server for synchronization. The client, then, responds with an ACK to let the server know the successful reception and full establishment of the connection. Then, the client has the chance to send traffic to the server through such a connection, e.g., large numbers of SYN requests can be sent to the server with faked origin addresses. Each communication will require server memory and, in the end, the server will run out of it. As WebSockets use TCP to establish the communication, DoS attacks can occur by the same token. In WebSockets, there is no upper bound on the number of incoming connections reaching the server side; so it may be high (potentially unlimited). The consequence is that either the web server reponse latency in greatly slowed down or the server becomes unavailable.

No authentication in the handshake protocol a.k.a. *Cross-Site WebSocket Hijacking*. In WebSockets protocol, the server does not authenticate the client during the handshake process, i.e., only the HTTP mechanism for connection establishment is available (not HTTPS). HTTP sends the authentication information directly to the WebSocket part. This is a vulnerability that can be exploited [20]. If the protocol solely relies on cookies, an attacher might trap a client user to initiate a request with its session on the attacker's server. Once the trap is successful, the attacker then will be able to communicate with the server via WebSocket and perform actions on behalf of the user. For solving this kind of problem, it is necessary to add a unique identifier per session which is not able to be guessed as a parameter in a handshake request.

The absence of a standard authentication procedure translates into development complexity as the corresponding safety problems are transfered to the developers. Moreover, putting in place different levels of authorization is also needed. In a WebSocket scenario, vertical and horizontal privilege escalation are possible.

Unencrypted TCP channels. As in HTTP, WebSocket also runs over TCP protocol, leading to similar security issues than those faced by HTTP; examples are man-in-the middle attacks or sensitive data exposure, among others listed in OWASP Top 10 [13]. Encryption methods like using TLS to provide origin authentication may help to mitigate the attack, but it still has weakness such as the TLS is not automatically enabled. The handshake process is unverified, i.e., keys are exchanged between client and server at the handshake to set up a secure connection; so, the attacker can gain a middle man position and steal credentials from user and server. Additionally, masking with WebSocket frames cannot avoid man-in-the-middle attacks to access data, as the key for masking the data is part of WebSocket [5] frame.

Input-data attacks. As SOP security policy is not enforced in WebSockets protocol, cibercriminals can inject malicious scripts, modifying the website in some sort and yield cross-site scripting attacks [19]. For example, an attacker can impersonate a privileged user, gaining access grants to run some code and modify the contents on the website. Later, when clients receive the modified server data, they have the risk of disclosing their credentials to an ilegitimate server. Under this vulnerability, attackers can exploit it to spread malware, steal user credentials, and more. Determined by the target categories, attacks can be divided into two types: attacks targeted user and attacks targeted server.

SQL injection attacks have some of the most dangerous effects on the server side. These can happen due to an improper data cleanning by the server (e.g. when receiving data from a form) and/or a malicious data input from a client to an unprotected server. Some other similar injection attacks on HTML code may also occur if malicious data are entered and stored in the server. The client that downloads this code can be redirected to a malicious URL with a valid client cookie previously used as URL parameter. Then, the attacker may collect cookie values from the client and may impersonate it.

Cache poisoning attack. Using transparent proxies, a.k.a. *intercepting* can be a common practice [27]. Transparent proxies cannot sometimes correctly deal with the upgrading mechanism used by the WebSocket handshake protocol. Actually, in HTTP there are two different fways to use the upgrade mechanism: (1) *connection* that contains the string `upgrade`; (2) *upgrade* header that indicates the protocol that the client would like to switch to. The tricky part here is that the *upgrade* header is typically not used; consequently, the handshake request is quite similar to a *post* request that also makes use of *upgrade* header for posting data [28]. In this scenario, an attacker can construct a server that utilizes WebSockets to establish a connection with a legitimate proxy server. The malicious payload from the attacker WebSocket server will be handled as valid HTTP request by the proxy server. As a result, any client requesting the infected file from the attacked server will obtain the malicious file.

Data masking. Data masking is a helpful feature used by WebSocket protocol to avoid attacks like proxy-cache poisoning in HTTP [13]. Web-cache poisoning takes place when attackers use the server and cache to send harmful HTTP responses to the victims. However, its down side is that it difficults the normal operation of security tools when trying to detect certain traffic pattens. Even more, WebSocket protocol is transparent to software like DLP (Data Loss Prevention); as a consequence, this software cannot perform data analysis on WebSocket traffic, and it cannot detect data leakage or malicious JavaScript code.

Tunneling. Most public networks have some form of firewall or proxy to enforce participants to use only certain protocols. Since the most commonly used protocol nowadays is HTTP, an attacker can use the WebSocket protocol (that is compatible with HTTP) to bypass firewalls and proxies. Also, there are some tools that provide tunneling for any type of traffic like [16] that established a TCP socket tunnel over WebSocket connection for bypassing firewalls.

Openning a connection with WebSockets from an insecure context. WebSockets should not be employed in mixed context environments. This means that there is a security hazard if a WebSocket connection is opened on a page loaded with HTTP.

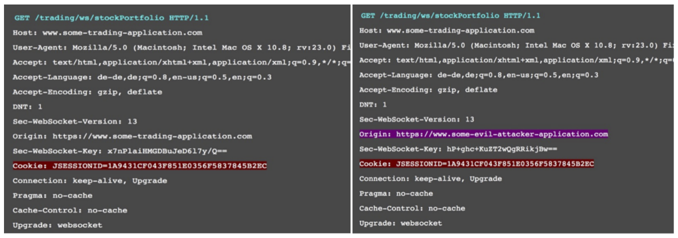
The solution here is to always protect the connection using secure WebSockets instead, i.e., WSS (WebSocket Secure protocol). In fact, some browsers only allow secure WebSocket connections; and do not support WebSockets in insecurte contexts.

B. Some exemplification

1) Handshake without authentication: As discussed before, client-server communication still uses HTTP protocol during the handshake process, and, at this step, there is no authentication. As a result, there is no SOP protection. A cross-site WebSocket hijacking attack might happen if the communication only counts on cookies.

Figure 2 shows how an attacker can easily construct a fake cookie that mimics the cookie from real client (shown in highlighted part) and send a request to server with such fake cookie. Once the trap is successful, the attacker then will be able to communicate with the server via WebSocket and perform actions on behalf of the user.

2) Unencrypted channel: As TCP is the protocol underneath WebSockets, there is an inherent risk to suffer a man-in-the-middle attack. Such attack consists of a cybercriminal creating an independant connection with the two communication endpoints and exchanging the data that they receive. By some means, the attacker gains a man-in-the-middle position between the two endpoints such that the endpoints believe that



```
GET /trading/ws/stockPortfolio HTTP/1.1
Host: www.some-trading-application.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:23.0) Fi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Sec-WebSocket-Version: 13
Origin: https://www.some-trading-application.com
Sec-WebSocket-Key: x7nPlaiHMGDBuJeD6l7y/Q==
Cookie: JSESSIONID=1A9431CF043F851E0356F5837845B2EC
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
```

(a) Original client request

```
GET /trading/ws/stockPortfolio HTTP/1.1
Host: www.some-trading-application.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:23.0) Fi
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
DNT: 1
Sec-WebSocket-Version: 13
Origin: https://www.some-evil-attacker-application.com
Sec-WebSocket-Key: hP+ghc+KuZT2wQgRRikjBw==
Cookie: JSESSIONID=1A9431CF043F851E0356F5837845B2EC
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
```

(b) Fake server response

Fig. 2. Problem due to lack of authentication

they are communicating directly with each other through a private connection; however, the fact is that the entire conversation is hacked by the attacker. In this situation, the attacker can intercept the conversation between the communicating parties and inject new content.

A man-in-the-middle attack is caused by the lack of mutual authentication. Most encryption protocols have added special authentication methods to prevent these attacks from happening. For example, using WebSocket protocol with SSL or TLS can verify whether the certificate used by the involved parties is issued by a legitimate and trusted digital certification authority, whereas the two-way identity authentication is performed.

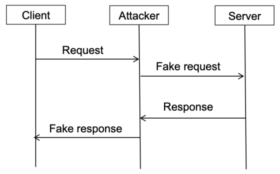


Fig. 3. Man-in-the-middle attack

Figure 3 illustrates a man-in-the-middle attack. A client will connect to the server, while there is an attacker waiting to intercept the communication; the attacker intercepts the user connection attempt and delivers a false message to the server. In the initial request, the client asks the server for some data. As the attacker is already monitoring the conversation, s/he may confront the actual server that actually wishes to send data to client. The attacker will provide the response to the client request; the attacker respose to the client will mimic that of the server including the needed hooks to make its appeal as real as possible. If the client believes that the other end of the conversation is the legitimate server, it may continue to send further requests and data, events her/his credentials. In this situation, both of the client and the legitimate server have no idea that they are not talking to each other. Thus, this type of attack might cause severe security breaches such as the leaking of keys.

V. USE CASE SCENARIO

The target scenario addresses an IoT ecosystem that monitors a smart home and supports actuation on the house hold to fine tune the comfort conditions, among other aspects. This scenario (shown in figure 4) includes a number of IoT devices and sensors as well as server nodes that control the devices and provide a gateway to some remote human user. The server node runs a web application that processes user requests for reading sensor values and provides means to set parameters like monitoring periods and other actuation values. Some nodes have a software stack that includes WebSocket to illustrate the particular vulnerabilities and the communication locations where these take place [24].

The data is obtained through sensors and external data sources (e.g., temperature and humidity sensors, cameras, etc.). Such devices are monitored and controlled by a single board embedded computer that processes the raw data. The collected data and processing results will be sent to a remote site in Internet via a gateway. The Internet will connect to a database server and via some IoT framework (examples can be FIWARE IoT platform, among many others) for further processing the data. After formatting these data and generating information based on their analysis, the information will be displayed on a user platform based on web technology for accessibility through different devices such as mobile phone, laptops, etc. Through the data shown on the website, users can monitor the conditions remotely and they can introduce actuation commands. In this scenario, data will be sent from the sensor to client (web page) at fixed time intervals (e.g., 60 seconds). Communication between information provider and server uses WebSockets to achieve full duplex communication: monitor the environment and actuate on it. In the smart home scenario, the parameters to be monitored are temperature and humidity; also, the safety conditions will be analyzed through cameras. In this context, WebSockets allow actors to access and exchange data in real-time via the web interface; but this introduces new challenges in privacy and security.

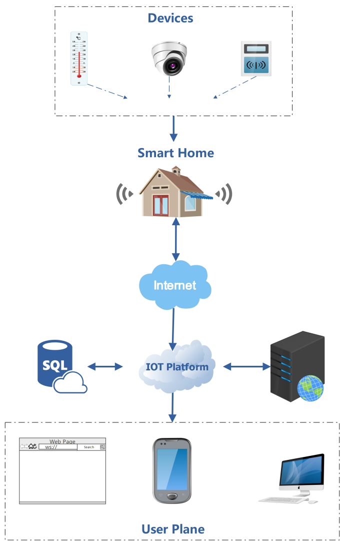


Fig. 4. Smart home scenario

In this use case, the parameters could be intercepted by third parties (i.e., attackers) for non-authorized uses. Figure 5 shows the communication steps between client and server nodes in this scenario. In the very beginning, client and server will perform handshake process under TCP protocol. Then firstly, the client sends a request to server for connection. Server will send ACK to client to show it has received request from client, and then send a rich HTML page to client via HTTP. This page will include JavaScript and WebSockets code and also display a login form to the client. The client then fills in its credentials and sends them back to the server. Once server receives these data, it starts to verify the credential to check if the message is from a legitimate user. If the client is validated by the server, the server will send another rich HTML page to the client side, which contains data readings from the sensors and has some form fields to let the user at the client node to select what parameters s/he wants to view or monitor from the smart home and/or what parameters s/he wants to control. After the client fills in the form with these requirements and clicks *submit* to server, an event that triggers the WebSocket protocol is generated: it will send an *upgrade* request for the server to switch the protocol to WebSockets for full duplex communication. The server receives these data and generates the corresponding return information with the sensor data requested by the user that will be sent to the client via WebSockets in the form of an *event* and its embedded data. For the communication steps described above, step 1 to 7 will be communicated under HTTP, while after step 8, the session will be carried on via WebSockets. At this point, different attacks can take place. For example, in the beginning process of TCP handshake is vulnerable to DoS attacks if the attacker tries to congest the communication with huge amount of request packet without response. Since the resources of server are occupied processing these unuseful packets, the actual packets with data retrieved by sensors are not able to be analyzed by server. Also, in the meantime, from the user plane, the clients cannot access to the website either. Thus, this whole system is no longer in due operation. As for the attack during WebSocket communication after step 8, it is likely that man-in-the-middle attack takes place; then, the credential can be stolen by an attacker. Moreover, since this connection under WebSocket does not use the encrypted transmission, the communication will be in high risk of attackers inserting a malicious script for modifying the website. As a result, the attacker will be able to log in or control the sensors at home on behalf of the actual user, resulting in security incidents where private data may be intercepted.

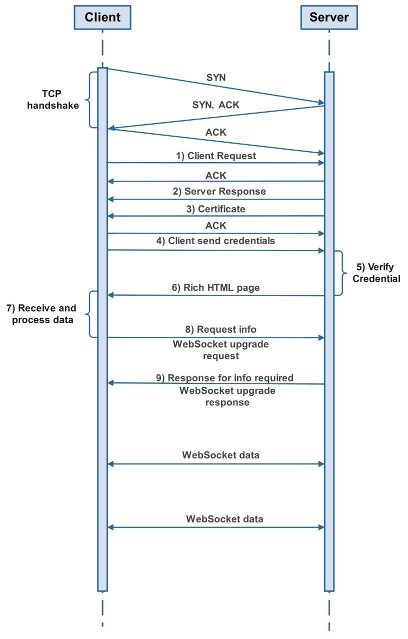


Fig. 5. Interaction diagram

VI. Discussion

Solutions to face vulnerabilities and avoid security hazards in web applications have significantly improved over time; but programming them is not always an easy task. In the particular case of WebSockets, after analyzing the vulnerabilities, a number of guidelines and recommendations follow to protect applications that use this technology.

Among the most effective and recommended solutions is to always use WebSocket Secure (WSS). Similar to what HTTPS means to HTTP, there is WSS to improve WebSockets. WSS helps in providing a secure transport as it consists of openning WebSockets connections over TLS/SSL therefore encrypting the channel. Then, a number of attacks that target the socket connection will be infeasible if the transport is secured, e.g., main-in-the-middle attacks.

To avoid input data attacks, it is needed to employ client-input validation techniques. The input data or any other requests from clients should be validated before being processed. With this validation, injection attacks via WebSocket are prevented. Besides client-side data validation, also validation at server side is important to ensure that it is indeed the data returned from a legitimate server.

For authorization or authentication, there are a few techniques like *ticket-based authentication* [25] [26]. It is based on the idea that when the client-side code tries to open a WebSocket connection, the HTTP server gathers some information for allowing the client-side code to obtain an authorization ticket. The server generates this ticket integrating in it information such as a user ID, the IP address of the client requesting the ticket, a timestamp, and other data for internal record purposes. Then, the ticket will be stored on the server or database and be returned to the client. At this moment, a client will be able to open the WebSocket connection and send this ticket together with the initial handshake. In that case, the server will have the chance to compare the ticket information, evaluating source IP and verifying the safety of this particular ticket origin. Then, the server will decide on the authenticated WebSocket connection.

Handling tunneling mechanisms can also be done via the usage of WSS to protect the transport channel. Nevertheless, using other secured and verified protocols on top of WebSockets is highly recommended.

Denial of Service (DoS) attacks can be addressed by rate limiting the connection request pace, that will prevent abuse of the web application. This helps preventing the website from malicious bots, scraping attacks, and small-scale denial of service (DoS) attacks. To implement rate limiting, a technique based on *buckets* can be employed: a *bucket* will be assigned to every user, and determine some parameters like: (1) limit over the WebSocket traffic that is sent by a user per second; (2) limit over the traffic that a server can safely process per second; or (3) timeout period for clients, among others.

Origin Header is recommended to be used in WebSocket.The origin header is similar to the AJAX X-Requested-With header. It determines which host a WebSocket connection is coming from. Otherwise the client will have the chance to communicate with any host over the WebSocket protocol. Although the origin might be faked by attackers, still this would provide an extra layer of protection for users as it requires the attackers to change the origin header on the client browser, which is blocked by modern browsers in most circumstances. Setting the origin field is a good security practice, but programmers should not only rely on it for authentication; instead, it should be combined with cookies or another authentication mechanism.

VII. Conclusions

Full-duplex communication is a major advance in web technologies that is available for over a decade now. It introduced a lower latency communication model with more efficient interaction than that of HTTP for applications that require constant information exchange. One of the technologies that made this possible was WebSockets protocol, and other frameworks such as Socket.IO. In this paper, we have analyzed some of these technologies with a focal point on WebSockets. It paves the way for developing time-sensitive web applications and overcomes the latency issues of half-duplex protocols. Its benefits have been presented as well as the drawbacks concerning the security issues that may be raised. Attacks like DoS and cross-site WebSocket hijacking have been analyzed, among others. In order to overcome these problems, we have also discussed some solutions like using a secure protocol, checking origin headers during handshake, employing a web token to perform authentication or validating input data at both client and server side. Although it is a hard task to provide a fully shielded web application, it is closer to possibble if proper mechanisms and programming schemes are employed.

References

[1] M. García-Valls, A. Dubey, V. Botti. Introducing the new paradigm of Social Dispersed Computing: Applications, Technologies and Challenges. Journal of Systems Architecture, vol. 91, pp- 83–102. Nov. 2018.

[2] J. Cristian. Real time web applications, comparing frameworks and transport mechanisms. UIO Dep. of Informatics, vol. 1(1), p. 15, 2014.

[3] S. Loreto ,P. Saint-Andre, S. Salsano, G. Wilkings. Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP. RFC 6202. IETF. ISSN: 2070-1721. April 2011.

[4] W3C. HTML5 A vocabulary and associated APIs for HTML and XHTML. W3C Recommendation. 28 October 2014. https://www.w3.org/TR/2014/REC-html5-20141028/

[5] Ian Fette, Alexey Melnikov. Relationship to TCP and HTTP. RFC 6455 The WebSocket Protocol. IETF. sec. 1.7. December 2011.

[6] Yinka-Banjo Chika, Ogundeyi Kehinde Esther. Financial stock application using WebSockets in real-time application. International Journal of Informatics and Communication Technology (IJ-ICT) Vol.8, No.3, December 2019, pp. 139 151 ISSN: 2252-8776, DOI: 10.11591/ijict.v8i3.pp139-139

[7] C. K. Rajak, U. Soni, B. Biswas and A. K. Shrivastava. Real-time web based Timing display Application for Test Range Applications. In Proc. of the 2^{nd} International Conference on Range Technology (ICORT), 2021, pp. 1-6, doi: 10.1109/ICORT52730.2021.9581663.

[8] Socket.IO. https://socket.io/docs/v4/ (Last Accessed January 2022)

[9] A. Rodas Vásquez, A. Valencilla Carrasquilla. Development and implementation of a prototype for a technological platform for the transmission of text and video (streaming) in real-time using technology websocket. Ingenierías USBMED, vol. 9, pp. 2-10, July-December 2018.

[10] X. Wu, C. Zhao, R. Xie, L. Song. Low Latency MPEG-DASH system over HTTP 2.0 and WebSocket. In Proc. of 14^{th} Int'l Forum of Digital TV and Wireless Multimedia Communication, pp. 357–367. 2018.

[11] V. Sharma, M. Mondal. Understanding and Improving Usability of Data Dashboards for Simplified Privacy Control of Voice Assistant Data In Proc. of 31^{st} USENIX. Boston, USA. August 10-12, 2022.

[12] L. Song; M. García-Valls. Improving Security of Web Servers in Critical IoT Systems through Self-Monitoring of Vulnerabilities. Sensors, vol. 22, 5004. December 2022.

[13] OWASP Top 10: 2021. https://owasp.org/Top10/

[14] K. Wahl-Jorgensen. The normalization of surveillance and the invisibility of citizenship: Media debates after the Snowden revelations. International Journal of Communications, vol. 11, pp. 740–762. 2017.

[15] MDN Web Docs. Same Origin Policy. `https://developer.mozilla.org/es/docs/Web/Security/Same-origin_policy` (Last Accessed January 2022)

[16] WSTunnel. https://github.com/erebe/wstunnel (Last Accessed Jan. 2022)

[17] CanIUse. https://caniuse.com/?search=websockets (Last Accessed January 2022)

[18] S. Kumar, R. Mahajan, N. Kumar and S. K. Khatri. A study on web application security and detecting security vulnerabilities. 2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2017, pp. 451-455, doi: 10.1109/ICRITO.2017.8342469.

[19] Admir Dizdar. What is XSS? Impact, Types, and Prevention https://www.neuralegion.com/blog/xss/ June 2021. (Last Accessed January 2022)

[20] W. Mei, Z. Long. Research and Defense of Cross-Site WebSocket Hijacking Vulnerability. In Proc. of the IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA 2020). Dalian, China. June 2020. DOI: 10.1109/ICAICA50127.2020.9182458

[21] D. Skvorc, M. Horvat and S. Srbljic. Performance evaluation of Websocket protocol for implementation of full-duplex web streams. In Proc. of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2014, pp. 1003-1008, doi: 10.1109/MIPRO.2014.6859715.

[22] Paul Murley, Zane Ma, Joshua Mason, Michael Bailey, and Amin Kharraz. 2021. WebSocket Adoption and the Landscape of the RealTime Web. In Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3450063 https://kharraz.org/publications/www21.pdf

[23] Ramus Appelqvist, Oliver Ornmyr. Performance comparison of XHR polling, Long polling, Server sent events and Websockets. Blekinge Institute of TEchnology, Sweden. 2017. http://www.diva-portal.org/smash/get/diva2:1133465/FULLTEXT01.pdf

[24] B. Soewitoa, Christiana, WebSocket based Smart home applications. 4th International Conference on Computer Science and Computational Intelligence 2019, (ICCSCI), 12-13 September 2019

[25] M. E. Hajj, A. Fadlallah, M. Chamoun and A. Serhrouchni. A Survey of Internet of Things (IoT) Authentication Schemes. Sensors, vol. 19, no. 5, March 2019.

[26] T. Qiu, N. Chen, K. Li, M. Atiquzzaman and W. Zhao. How can heterogeneous Internet of Things build our future: a survey. IEEE Trans. Comm. Surveys & Tutorials, vol. 20, no. 3, pp. 2011-2027, Feb 2018.

[27] L. Huang, E. Chen, A. Barth. Talking to yourself for fun and profit. Proceedings of W2SP, 1–11. 2011.

[28] J. Karlström. The WebSocket Protocol and Security- Best Practices and Worst Weaknesses. Master's Thesis. Department of Information Processing Science, University of Oulu.