ESKISEHIR OSMANGAZI UNIVERSITY
FACULTY OF ENGINEERING ARCHITECTURE
COMPUTER ENGINEERING

SECURITY OF WEBSOCKET

Orhan Özkerçin	152120151008
Nur Sultan Bolel	152120151022

DECEMBER 2018

Abstract

Websocket has emerged to provide real-time and two-way connectivity needs.WebSocket technology comes with HTML5 and developed with HTML5. WebSocket has many advantages but over time security vulnerabilities were noticed by attackers. On the other hand this protocol gives so many advantages that we can't easily give it up. There are some threats about Websockets and some solutions about it. This paper starting from basic information about connection protocols and continue with websocket and threats about it. Also there are some solutions about this threats. In short in this paper we have tried to examine security of Websocket.

1. Introduction

WebSocket is a technology that provides a bi-directional real-time communications channel over a Transmission Control Protocol (TCP) connection[1]. The WebSocket technology covers JavaScript API (Application Programming Interface) that is specified by World Wide Web Consortium (W3C), and a protocol that is specified by Internet Engineering Task Force (IETF)[2].WebSockets provide a persistent connection between a client and server that both parties can use to start sending data at any time.The client establishes a WebSocket connection through a process known as the WebSocket handshake. This process starts with the client sending a regular HTTP request to the server. An Upgrade header is included in this request that informs the server that the client wishes to establish a WebSocket connection[3].

2. Background

This chapter includes the WebSocket technology in more detail and has two section. 2.1 section is about advantages and abilities of WebSocket.2.2 section is about functionality of WebSocket. This section includes some basic code from WebSocket JavaScript API and their explanation.

2.1 Features of WebSockets

If we want to understand advantages and what is the ability that webSocket can give us, we need to look difference of the usual network traffic over HTTP[4]. The main difference is web socket protocol does not follow the traditional request, you just connect once. Once a client and a server have opened a web socket connection, both endpoints may asynchronously

send data to each other. The connection never ends until one of the client or a server close.[4]

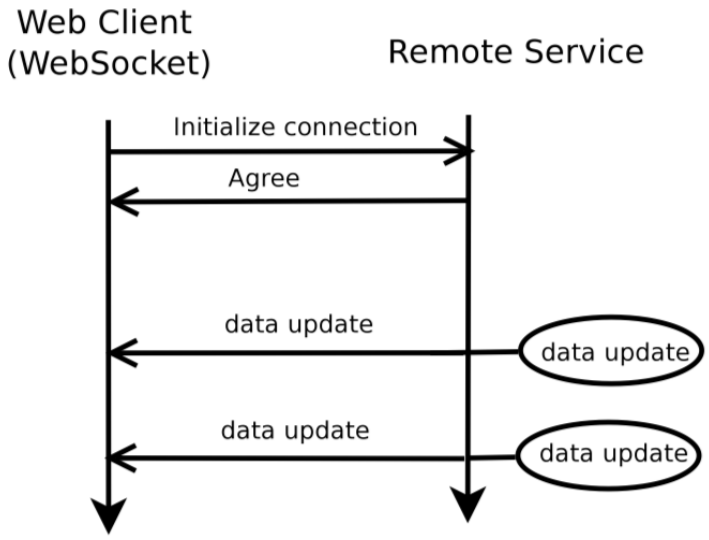

Fig.1. Retrieving real-time data with HTTP

Opposite way of this the traditional approach relies on polling. Polling means client opens a new TCP connection and makes HTTP request to receive data from server but server can not send data if client doesn't check. Also for every separate data transmission need this loop. If clients wants to know updates need to check server every time. Therefore, compared to WebSockets ,the traditional HTTP request-response convention results in high latency and high amount of network traffic.

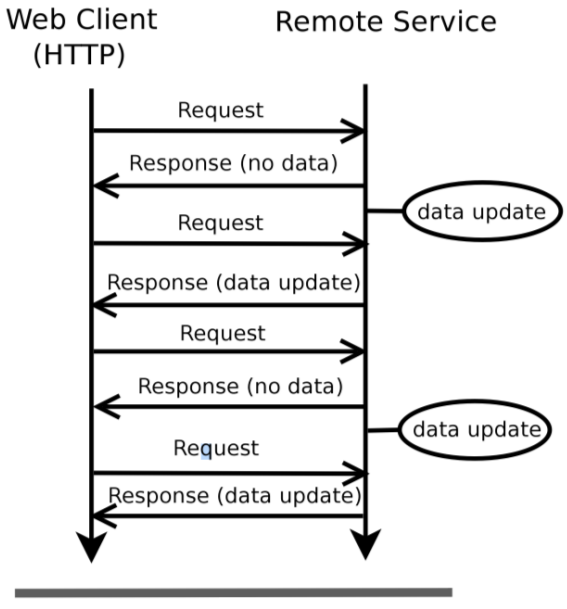

Fig.2. Retrieving real-time data with Web- Socket

2.2 Functionality of WebSocket

A client sends a classic http-request to a server only once to establish a WebSocket channel and server accepts the request, opens a connection to the client via WebSocket. As a result, a full-duplex / two sided connection is WebSocket protocol. When the server has accepted the request, the subsequent messages are sent using the WebSocket protocol.
The WebSocket API is an interface that enables developers to use the WebSocket protocol in web applications.[5] The WebSocket API can be used in a very similar way to JavaScript in XmlHttpRequests.
Creating WebSocket connections is simple. WebSocket constructor is called and URL is passed of your server.

```
var socket = new WebSocket
  ("ws://<address>:<port>");
socket.onopen = function(e) {
  // socket opened
};
socket.onmessage = function(e) {
  alert("data sent by server: " + e.data);
};
socket.onclose = function(e) {
  // socket closed
};
```

a

lizing a WebSocket connection

In the Fig.3 first argument of WebSocket(),address, specifies the URL to which to connect. The second attribute, port is optional, and if present, specifies a sub-protocol that the server must support for the connection to be successful.

```
socket.send("hello world");
socket.close();
```

Fig.4. Sending data and closing the WebSocket connection

Once you get a WebSocket connection with the web server, you can send data from browser to server by calling a send() method, and receive data from server to browser by an onmessage event handler.[6]

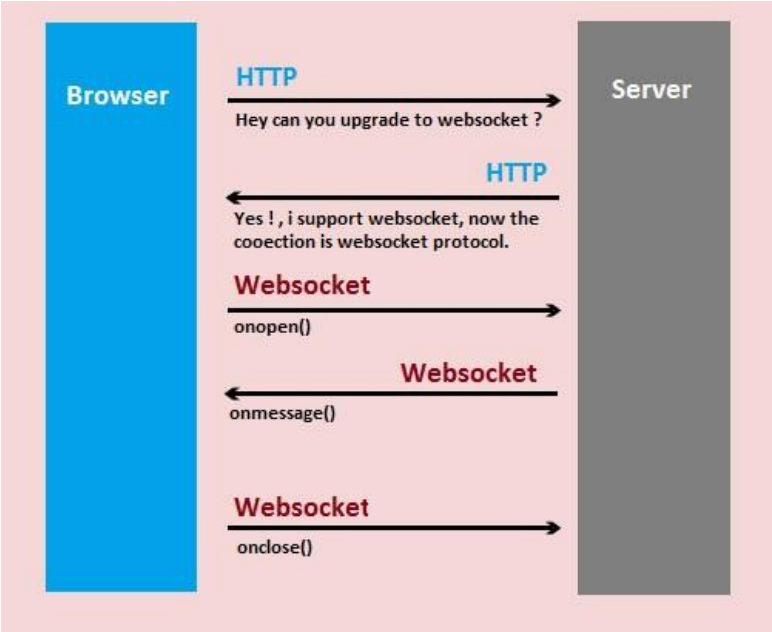

Fig.5. One TCP connection and upgraded http protocol

3. Security Issues of WebSocket

This chapter includes the WebSocket's security issues and a few attacks types. The WebSocket's security issues are cross-domain requesting, traffic encryption, authentication and authorization. The some attacks which are about the WebSocket's security issues are Cross Site WebSocket Hijacking (CSWSH), Man-in-the-Middle (MitM).

3.1 Cross-Domain Requesting

The WebSocket Protocol enables two-way communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code. The security model used for this is the origin-based security model commonly used by web browsers. [3-2] The purpose of origin-based security model to know the scope of authority or privilege by user agents. When a client wants to open a WebSocket channel sends an http-request and web browser adds an HTTP header.The HTTP header field, named "Origin", that indicates which origins are associated with an HTTP request.[7].

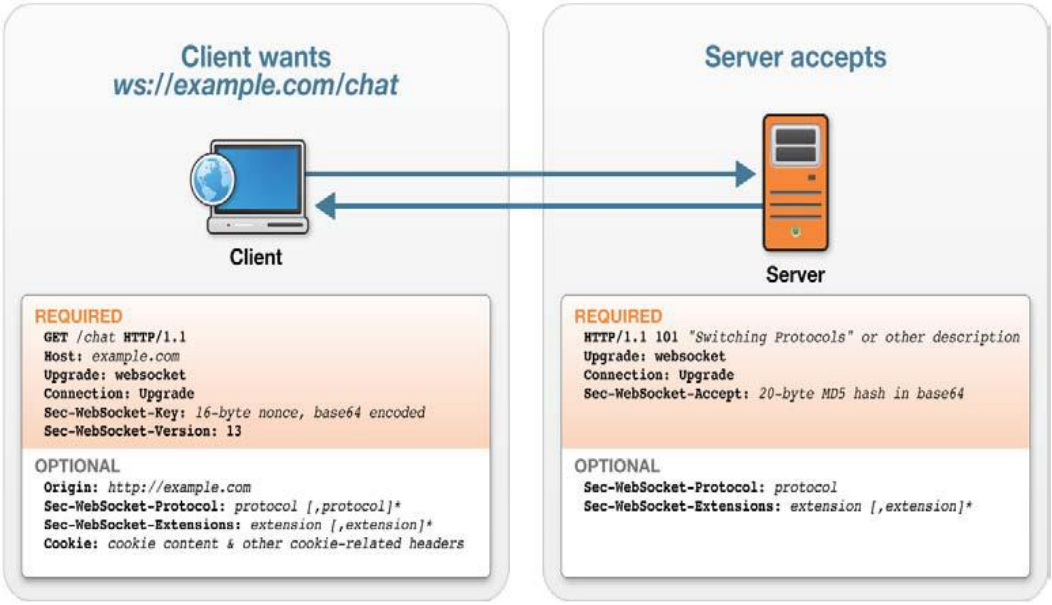

Fig.6. Protocol Overview

In Fig.6. illustrates the protocol which has two parts: a handshake and the data transfer. A web document makes a cross-origin access when it requests a resource from a different domain, protocol, or port. As it can be seen from the figure, the |Origin| header field is used to protect against unauthorized cross-origin use of a WebSocket server by scripts using the WebSocket API in a web browser. The server is informed of the script origin generating the WebSocket connection request. If the server does not wish to accept connections from this origin, it can choose to reject the connection by sending an appropriate HTTP error code. This header field is sent by browser clients; for non-browser clients, this header field may be sent if it makes sense in the context of those clients.[4]

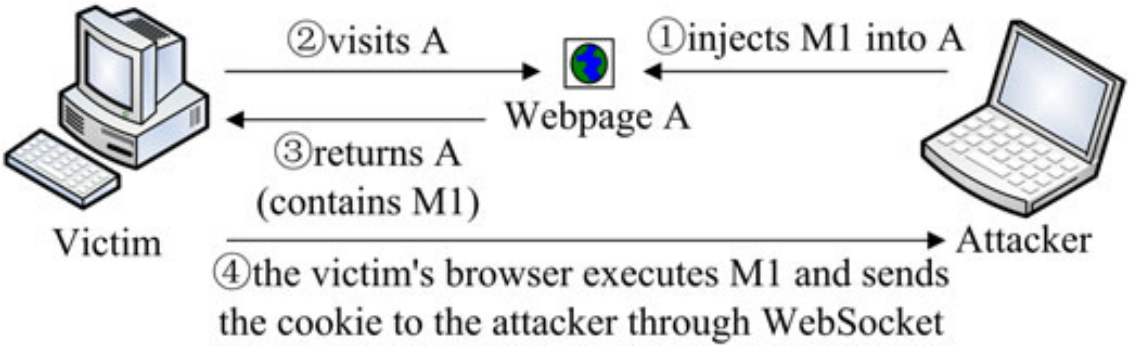

Fig.7. Attack scenario of stealing privacy via WebSocket.

```
<scirpt>
var cookie = document.cookie;
var ws = new WebSocket("ws://attacker.com:12345");
ws.send(cookie);
</scirpt>
```

Fig.8. Example script code for injection operation (M1)

Same-origin policy is a critical security mechanism that restricts how a document or script loaded from one origin can interact with a resource from another origin.[8-8]The same-origin policy does not limit the WebSocket APIs and allows the cross-domain communications. Thus, attackers can easily use the send() method to steal the cookies or other sensitive data. As the data are sent through the WebSocket, the traditional HTTP-based firewall may fail to detect these new privacy thefts. Fig.7. shows a realistic attack scenario. The attacker uses the WebSocket to steal the victim's privacy. First, the attacker injects script code into the web page A. The content of the script code is to read the cookie and send it to the attacker. When the victim visits A, the victim's browser will execute the script code and send the cookie to the attacker.[9] Attacks with this way is called as Cross Site WebSocket Hijacking (CSWSH) attack which is illustrated in Fig.8.

3.2 Traffic Encryption

Traffic encryption is known as hypertext transfer protocol secure (HTTPS).It ensures the security of website traffic by encrypting the information being transmitted, and by using security certificates to identify and authenticate the website.[10] When we don't use the encryption , an attacker can listen to the original connection and attacks with this way is called as Man-in-the-Middle (MitM) attack which is illustrated in Fig.9.

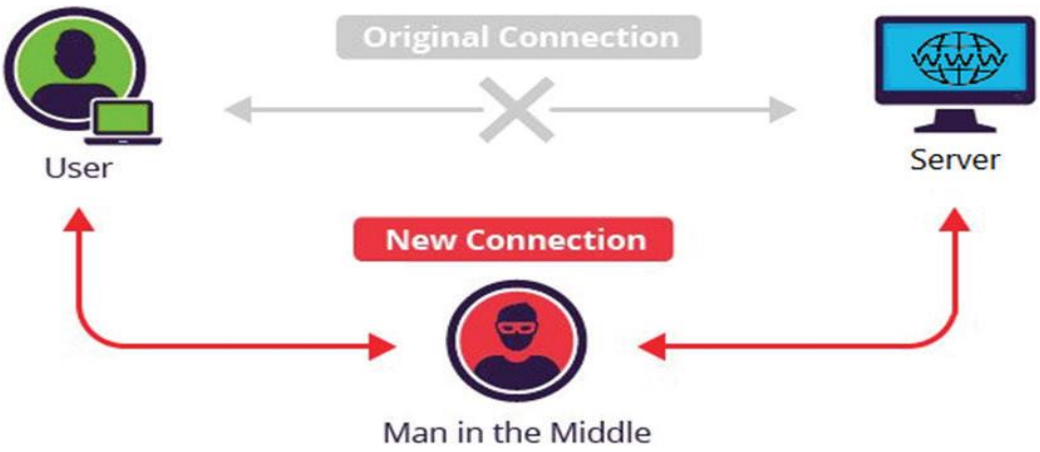

Fig.9. Common MitM architecture.

A common MitM attack, in which the attacker 'sits' between two parties connecting to both ends. As shown in Fig.9., in the common MitM

architecture, when one party (the client) sends the data, the MitM attacker acts as the receiving end (the server) and, vice versa, when the server sends the response back, the attacker acts as the client. The most useful protocol to accomplish MitM is WebSocket. [11] BLE component on the left side of the Fig.10. acts as the smart device,BLE component on the right side of the Fig.10. acts as the Mobile App. According to the figure two different protocol is required for MitM attack by using WebSocket.

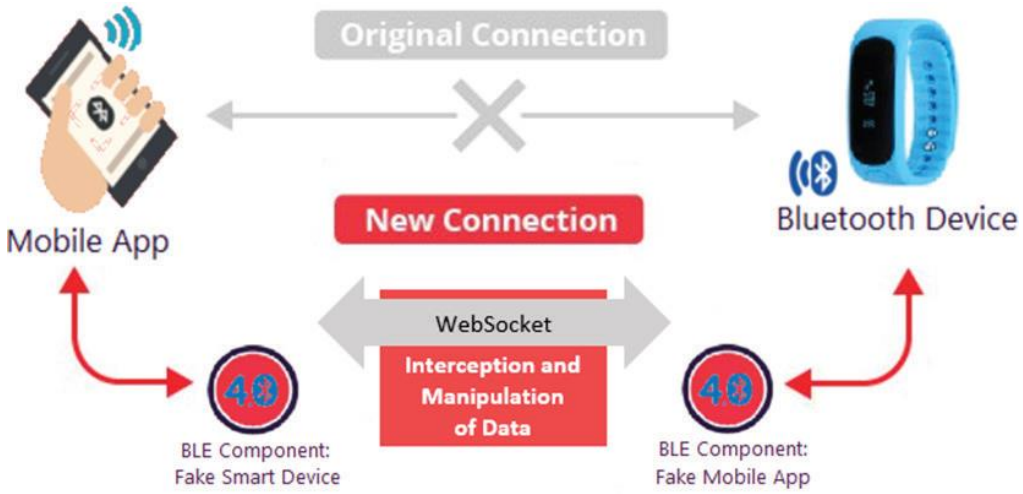

Fig.10. An example of MitM attack by using WebSocket.

3.3 Authentication

This protocol doesn't prescribe any particular way that servers can authenticate clients during the WebSocket handshake. The WebSocket server can use any client authentication mechanism available to a generic HTTP server, such as cookies, HTTP authentication, or TLS authentication.[12]

Since you cannot customize WebSocket headers from JavaScript, you're limited to the "implicit" auth (i.e. Basic or cookies) that's sent from the browser. Further, it's common to have the server that handles WebSockets be completely separate from the one handling "normal" HTTP requests. This can make shared authorization headers difficult or impossible.[13]

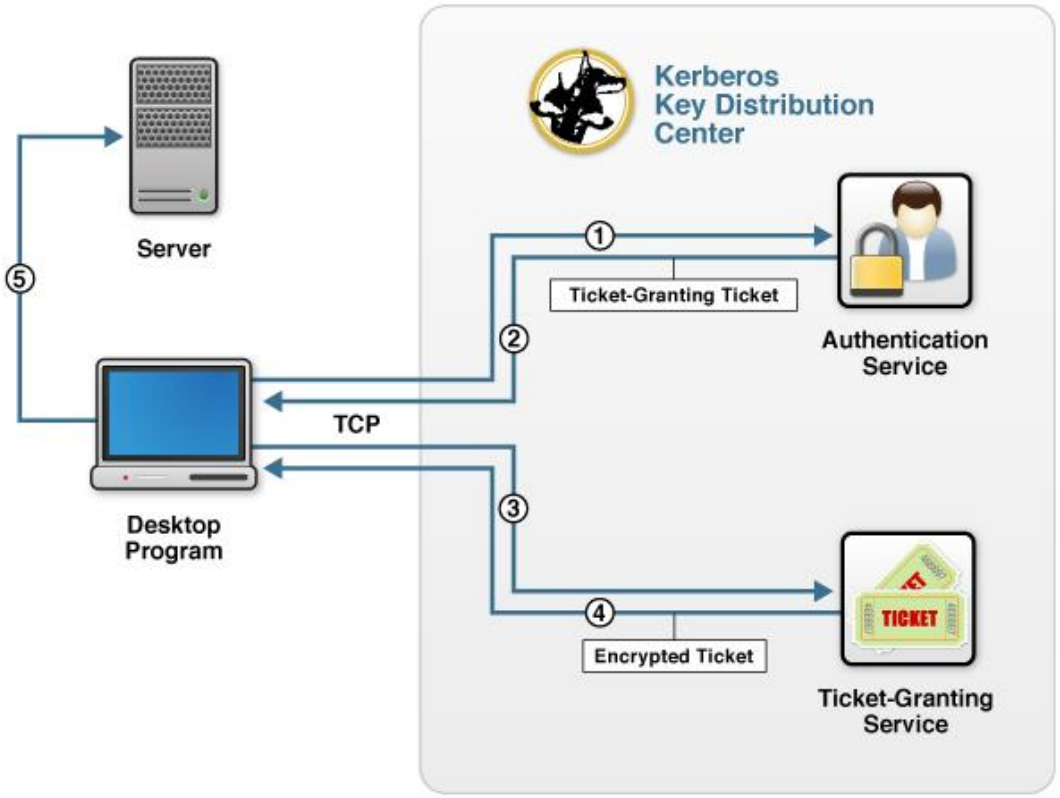

Fig.11. An example of authentication system

So, one pattern we've seen that seems to solve the WebSocket authentication problem well is a "ticket"-based authentication system. Broadly speaking, it works like this:

- When the client-side code decides to open a WebSocket, it contacts the HTTP server to obtain an authorization "ticket".
- The server generates this ticket. It typically contains some sort of user/account ID, the IP of the client requesting the ticket, a timestamp, and any other sort of internal record keeping you might need.
- The server stores this ticket (i.e. in a database or cache), and also returns it to the client.
- The client opens the WebSocket connection, and sends along this "ticket" as part of an initial handshake.
- The server can then compare this ticket, check source IPs, verify that the ticket hasn't been re-used and hasn't expired, and do any other sort of

permission checking. If all goes well, the WebSocket connection is now verified.[13]

3.4 Authorization

Authorization ensures that the authenticated user has appropriate privileges to access resources. The resources the user have access usually depends on the user's role in the service.With authorization user should only have access to resources, that the user is able to perform an assigned job task, nothing else.
In case of WebSocket, authorization is heavily application context dependent. There are three possible scenarios that could occur:

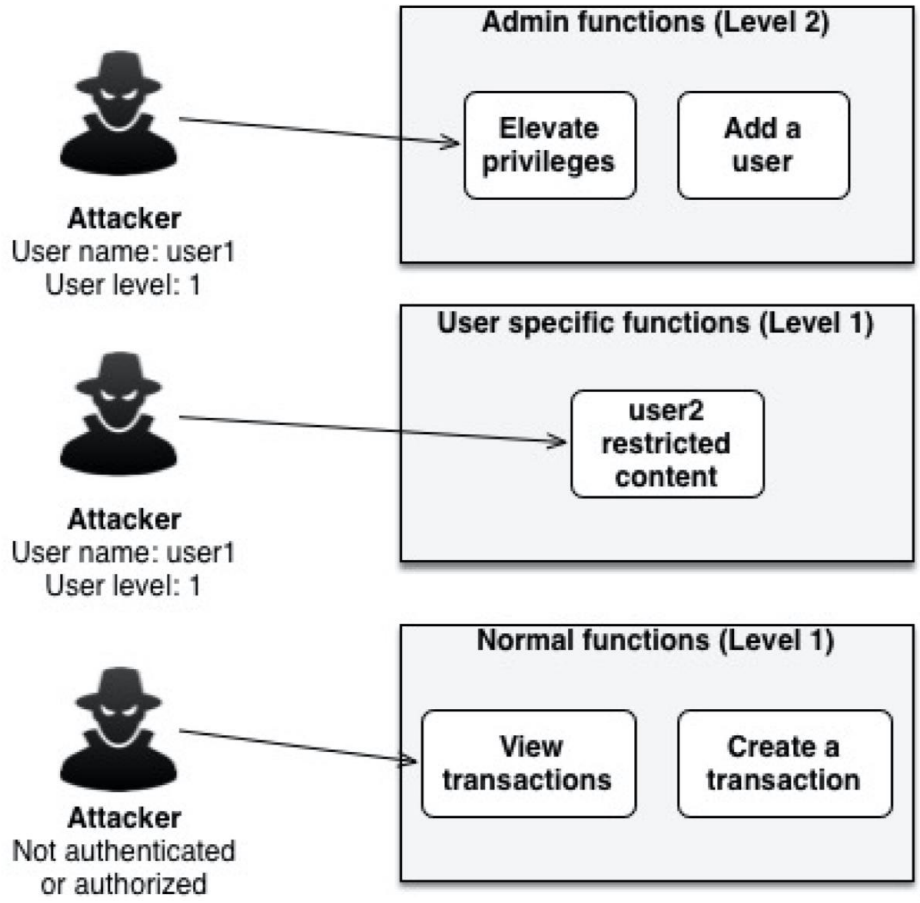

Fig.12. Attacks against authorization levels.

- An attacker is able to access a function that requires higher-level authorization to the WebSocket service than originally assigned for the user.

- An attacker is able to access a function that shows other user's restrictedcontent.
- An attacker is able to access the WebSocket service that requires a user authorization, without any authorization.[14]

4. Discussion and Solution

The WebSocket protocol is one of the new HTML5 protocols.There are several articles about HTML5 and their security. WebSocket technology comes with HTML5 and developed with HTML5. WebSocket has many advantages but over time security vulnerabilities were noticed by attackers. In this article some of security issues are specified in chapter 3.

When analyzing possible solutions for the security issues involving the WebSocket technology,we can divide the problem into two separate parts.
1. Deploying the WebSocket technology in trustworthy web services, while preserving the security of the service and privacy of users.
2. Shielding web browsers and end-users against malicious usage of WebSockets.[15]

Detecting for the attack can be very difficult. In this case, prevention is better than cure.[16] Below are the general guidelines for the safe deployment of a web application utilizing WebSockets. The list is based on the findings of this paper and the "HTML5 Security Cheat Sheet":[15]

- Both a service and a client should be implemented carefully, strictly following the latest specifications.
- Developers should be aware that XSS vulnerabilities exist, and they can be used to compromise WebSocket based communication.
- To achieve confidentiality and integrity, TLS encryption should be used. If the website initiating the WebSocket connection is delivered over HTTPS, TLS should be used anyway.
- Attention should be paid to implementing the authentication and the session management.
- Endpoints should not trust each other. Thus, both a client and a server should validate all the input received through WebSocket connections.[17]

5. Conclusion

This paper provides lots of headlines and topics about Websocket. Also paper dealt with answers of questions such as 'What is the Websockets', 'What is the differences with HTML and Websockets'.Also vulnerabilities of

websockets and solutions of threats are given. There are examples about Websocket issues and attacks and also how to block this attacks.
In conclusion the websocket technology gives us some opportunities of real time web applications. Also websocket has some vulnerabilities.Websocket technology provides us some opportunities of real time web applications.

References

[1] R. R. Ganji, M. Mitrea, B. Joveski, and F. Preteux.HTML5 as an application virtualization tool. In Consumer Electronics (ISCE), 2012 IEEE 16th International Symposium on, pages 1 –4, June 2012.

[2] Internet Engineering Task Force (IETF), "The WebSocket Protocol",RFC 6455

[3] https://blog.teamtreehouse.com/an-introduction-to-websockets

[4] A. Wessels, M. Purvis, J. Jackson, and S. Rahman. Remote Data Visualization through WebSockets. *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 1050 –1051, April 2011.

[5] J. Karlström, "The WebSocket Protocol and Security: Best Practices and Worst Weaknesses", *WebSocket API, pages 17-32,* Oulu Finland, 14.12.2015

[6] https://www.tutorialspoint.com/html5/html5_websocket.htm (Site Access Date: 12.12.2018)

[7] Internet Engineering Task Force (IETF), "The Web Origin Concept",RFC 6454

[8] https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy (Site Access Date: 13.12.2018)

[9] J. Bai, W. Wang1, M. Lu, H.Wang, J. Wang, "TD-WS: a threat detection tool of WebSocket and Web Storage in HTML5 websites",*Client-side WebSocket privacy leaks, pages 5433-5434,*Changsha USA,27.11.2016

[10] https://gethelp.wildapricot.com/en/articles/178-securing-your-site-using-traffic-encryption#about (Site Access Date: 15.12.2018)

[11] T. Melamed, "An Active MAN-IN-THE-MIDDLE Attack on Bluetooth Smart Devices", *Practical MitM Attack For BLE, pages,* İsrail, 2018

[12] https://github.com/cassiomolin/chat-websockets (Site Access Date: 19.12.2018)

[13] https://devcenter.heroku.com/articles/websocket-security#authentication-authorization (Site Access Date: 20.12.2018)

[14] H.Kuosmanen, “Security Testing of WebSockets”, *Authorization, page 29,* JAMK University of Applied Sciences, May 2016

[15] Jussi-Pekka Erkkilä,”WebSocket Security Analysis”,*Analysis of possible solutions, page 5,* Aalto University School of Science, Autumn 2012

[16] https://www.computerweekly.com/tip/Man-in-the-middle-attack-prevention-strategies(Site Access Date: 25.12.2018)

[17] https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet (Site Access Date: 25.12.2018)