HOGESCHOOL ROTTERDAM
TECHNISCHE INFORMATICA

How to properly secure the connection between a server and an embedded device

Student name: *Niek van Leeuwen 0967267*

Course: *Project 78 (2020) Sensetable* – Professor: *S.M. Hekkelman, H. Kroeze*
Due date: *21 June, 2020* – First submission

Contents

1. Introduction

For project 7/8 I am working with my team on the *Senstable*. The table consists of a dynamic combination of different test set-ups to easily test different sensors. The *Senstable* will be made available to our fellow students in the *Stadslab Rotterdam*. This is a place that offers easy access to new manufacturing technologies such as 3D printers and laser cutters. It is also possible for students to buy parts and follow workshops here.

During our project we need to connect an embedded device to our server, so we can display data in real-time on our website. To achieve this, we have used WebSockets. We have written code for the NodeMCU to connect to our NodeJS server using this protocol.

2. The problem

Although no important data is sent between the embedded device and the server, it is important to make our systems as secure as possible to prevent abuse. For example, if we don't add security to the server, anyone can log on a server and pretend to be a test setup. The data that a test setup sends is passed directly to the users of the website. If the server (a trusted source) sends unauthorized data to the user, this is a major security problem.

Also we can't offer the website via HTTPS because of the mixed content [1] policy. Mixed content means that a website sent over HTTPS also has a connection to a unsecured server. This means that no unsecured WebSocket connection may be established on a site sent over HTTPS. However, it is a good idea to implement HTTPS on our site [2]

I am going to answer the main question with a few partial questions: which form of encryption is appropriate, how do I authenticate a connection, is input validation necessary and finally, is origin-based security possible.

3. Background

Before we start I would like to make clear how the data flows within our project. We have 6 different test setups, each with its own sensor. Each test setup contains a NodeMCU, this is a microcontroller based on the inexpensive WiFi-microchip ESP8266. We connect this microcontroller to the WiFi network. Finally, the NodeMCU starts a WebSocket connection with our server.

After a connection to the server has been established, the microcontroller sends new data to the server at 300ms intervals. On of the advantages of this option compared to, for example, sending data to an API is that the client does not have to poll the server for new data, but that new data is only plotted as soon as new data has been received. Another advantage is that the WebSocket protocol has less overhead than, for example, HTTP and is therefore better suited for use to display real-time data. [3]

4. Methods

In order to be able to answer the main question, I first need to research the current solution to connect embedded devices to the internet. I also need to investigate how a server can be secured. Next, it is important to look separately at the actual connection between the embedded device and the server. I am mainly going to use literature research during this research, but I am also going to use our access to embedded devices and server to test some parts myself.

5. Results

In this chapter I will describe the results of my earlier described research.

5.1. Encryption methods.

During a data transmission, the sensor ID, the sensor value and the key (more on this subject later), is sent over a WS (WebSocket) from the client to the server. If WS is used instead of WSS (WebSocket Secure), all requests and answers can be read by anyone following the session. Essentially, a malicious individual can simply read the text in the request or response and know exactly what information someone is asking for, sending, or receiving. To secure this connection, encryption must be used. Encryption is the only option here because the data does not need to be irreversibly encrypted, the data the client sends to the server needs to be read by the server. This is why the data is encrypted by the client and sent over the network, and then decrypted by the server.

Because WS is not encrypted, the WSS protocol was created. WSS is based on the WS protocol with an encryption layer on top. Nowadays the term Secure Sockets Layer (SSL) is still very common. SSL is no longer secure [4]. The successor of SSL is the Transport Security Layer, or TLS. TLS is the encryption layer on the WebSocket protocol. WebSocket over TLS, (same as HTTPS is HTTP over TLS), the transport security layer encrypts the data at sender and decrypts at the receiver.

5.2. Authentication of an API call.

Our server, which accepts the WebSocket connections, is publicly available. This makes it possible for a test-setup to establish a connection from any WiFi network to the server, but it also means that anyone can connect to the server. To make sure that no changes are made to the data that does not originate from one of the test setups, authentication is required. A common way is to use keys, for example a secret access key, password, or access token. The advantage of using the WS protocol over other stateless protocols such as HTTP is that we only need to authenticate a test setup once. To authenticate a client it is therefore possible to send a key while setting up the WS connection and then the server knows that this connection can be trusted.

It is also good practice to generate a separate key for each client. This is an important detail as it makes it possible, in case of a breach, to detect abuse by linking the key used to a client. It is also important to store the API keys safely. For example, it is important not to save the keys anywhere in plain text [5].

5.3. Input validation.

Another important part to secure the server is 'input validation'. This means that all data sent to the API is extensively checked for unexpected characters or string length, for example. This prevents incorrectly shaped data from being entered, which could cause the server to crash. It also prevents other security risks, such as SQL injections [6], but in this case, our NodeJS server is not connected to any database.

To test our own server and input validation I set up the server and started sending data. At first I started testing a best-case scenario, in which there was only a typo in the supplied JSON. I have removed a character from a correct JSON string as shown below:

```
{
   "sensor":{
      "id":6,
      "value":165"
   }
}
```

To my surprise, or maybe not, the server crashed immediately. And actually this is a good thing, imagine if the JSON string was malicious, then it couldn't do any further damage since the parsing didn't succeed. If input validation had been present, the server would not only have been safer but also more robust, because the server would have caught this parse error with the measure in place.

5.4. Origin-based Security.

When using Origin-based Security, the Origin header, that the client needs to send to the server, is used to to filter malicious clients. This also is related to Same-Origin policy [7]. This policy ensures that only content is loaded from the same origin. For example, if we use the same server for the website as the web sockets, we may use this policy to make our system more secure. To make the WebSocket connection more secure, we can enable the server to only accept data from a certain origin. This has disadvantages such as the origin changing the client's public IP, or the public IP being changed by the provider of the internet. If the origin of the client changes, it is therefore necessary to modify the server code.

6. Conclusion

During this research it was deemed necessary to use encryption to secure the connection between an embedded device and a server. In order to secure our test setups we have chosen to use the WSS protocol because of the multiple benefits it brought as described in this study. This research also shows that a key needs to be generated to authenticate the WS connection. Subsequently, it is important to have good input validation to avoid wrong/harmful data in the system,among other advantages. Finally, origin-based security is not a good option because, during our project, we cannot guarantee that the origin of our client will remain the same.

References

[1] MDN contributors. *Mixed content*, Last modified: Apr 1, 2020 (accessed June 19, 2020).

[2] Michael Rodriguez. Https everywhere: Industry trends and the need for encryption. *Serials Review*, 44(2):131–137, 2018.

[3] V. Pimentel and B. G. Nickerson. Communicating and displaying real-time data with websocket. *IEEE Internet Computing*, 16(4):45–53, 2012.

[4] Ankita R Chordiya, Subhrajit Majumder, and Ahmad Y Javaid. Man-in-the-middle (mitm) attack based hijacking of http traffic using open source tools. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*, pages 0438–0443. IEEE, 2018.

[5] H. K. Lu. Keeping your api keys in a safe. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 962–965, 2014.

[6] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda. Preventing input validation vulnerabilities in web applications through automated type analysis. In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pages 233–243, 2012.

[7] MDN contributors. *Same-origin policy*, Last modified: Feb 22, 2020 (accessed June 19, 2020).