Chapter 7

7 Building a Fortress: Strategies for Securing Communication Channels

The main challenge for network architects is not setting up communication between different hosts, LANs, VLANs, and WANs, but rather securing that communication. Even in a closed environment, there are potential threats that could compromise the security of the communication channels and the data they transmit. Therefore, network architects must implement a range of security measures to protect communication channels and ensure the confidentiality, integrity, and availability of the data being transmitted. These measures include techniques such as encoding, encryption, hashing, authentication, and authorization, network segmentation, access control, and other creative solutions.

7.1 Basic concepts

7.1.1 Encoding vs Encryption vs Hashing

Encoding:

Encoding is the process of converting data from one format to another. This can be done for a variety of reasons, such as to make the data compatible with a particular system or to make it easier to transmit over a network. Common examples of encoding include Base64 encoding and URL encoding.

Encoding is not a form of encryption, as the encoded data can easily be decoded back to its original form. Encoding is not intended to provide security, but rather to facilitate the transfer or storage of data.

Encryption:

Encryption is the process of converting data into a format that is unreadable without a key or password. The goal of encryption is to protect the confidentiality of the data so that it cannot be read by unauthorized users.

Encryption can be either symmetric or asymmetric. With symmetric encryption, the same key is used to encrypt and decrypt the data. With asymmetric encryption, a public key is used to encrypt the data, and a private key is used to decrypt it.

Encryption is commonly used to protect sensitive data such as financial information, login credentials, and personal identification data. It's important to note that encryption does not ensure data integrity or authenticity, meaning that encrypted data can still be modified by unauthorized users if they are able to decrypt it.

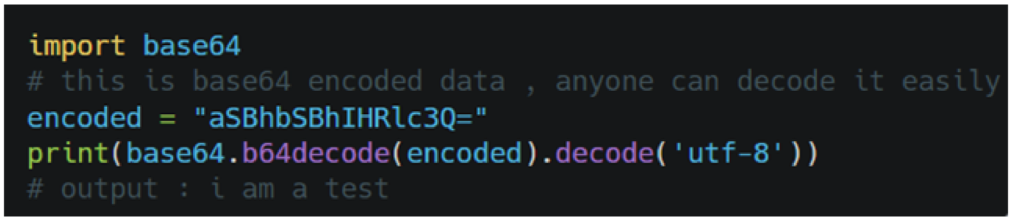
Hashing:

Hashing is a process of converting data of any size into a fixed-size string of characters, known as a hash. Hash functions are designed to be one-way, meaning that it's easy to generate a hash from a given input, but it's extremely difficult (if not impossible) to generate the original input from a hash.

Hashing is often used for data integrity verification, where the goal is to ensure that the data has not been tampered with. For example, a website might store the hash of a user's password instead of the actual password. When the user enters their password, the website generates a hash and compares it to the stored hash. If the hashes match, the password is considered valid.

Examples

As we can see the decoding process for an encoded text is quite easy as follows, also encoding formats are well-known and easy to identify

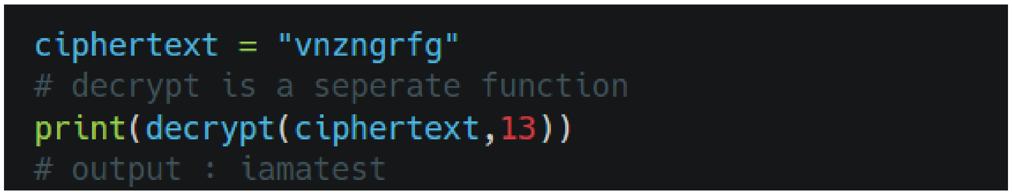


```python
import base64
# this is base64 encoded data , anyone can decode it easily
encoded = "aSBhbSBhIHRlc3Q="
print(base64.b64decode(encoded).decode('utf-8'))
# output : i am a test
```

However for encryption, it is difficult to decrypt an encrypted string without knowing the encryption algorithm applied and the key used, For example, if we have the following string we will have no idea what is this

Encrypted = "vnzngrfg"

After having the information of the cipher used is Caesar cipher and the key is 13 we can decrypt it now After having the information of the cipher used is Caesar cipher and the key is 13 we can decrypt it now   Now for hashing,



```python
ciphertext = "vnzngrfg"
# decrypt is a seperate function
print(decrypt(ciphertext,13))
# output : iamatest
```

there are dozens of hashing algorithms that can be used, the difference between them lies in the hash length, how the algorithm is applied, speed, and other factors. The most common one is MD5 Hash which is 32 Length, as you can see in the following example just because spaces are added in the same string we got entirely different hash values!

Hashing is not a reversible process (Although Attackers can bruteforce it from a wordlist until they find a word that has the same hash of the given hash

```
"iamatest"    : "0bd9b89127d7034515caf5f9508da6c7"
"iam a test ": "79c55c87e090157fecd657dc9a7bf4f9"
```

and figure out the original word in this way). Files, Folders, games, videos, and any data can have a hash value and the hash value is unique for the same data as we have discussed.

7.1.2 Authentication, Authorization, Accountability

Authentication:

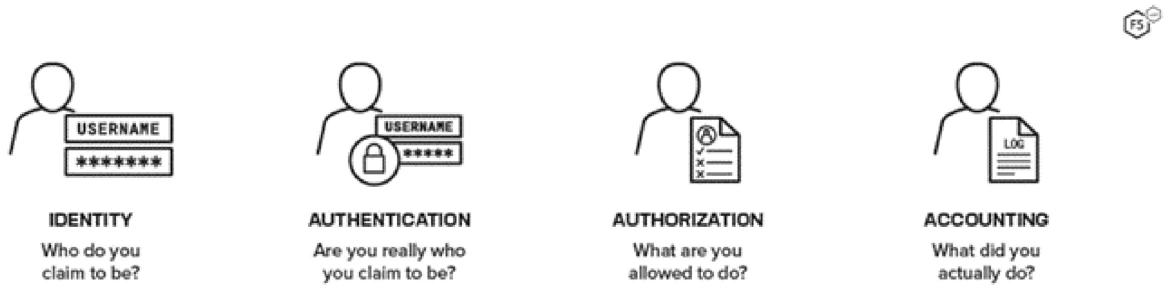
Authentication refers to the process of verifying the identity of a user, device, or application. Authentication is typically achieved by requiring a user to provide some form of credential, such as a username and password, a security token, or a biometric identifier. Once a user has been authenticated, they are typically granted access to some set of resources or functions based on their authorization.

Authorization:

Authorization is the process of granting or denying access to a resource or function based on a user's identity and associated permissions. Authorization is typically determined by comparing the user's identity and associated permissions to a set of access control rules that define which users are allowed to perform which actions on which resources. Authorization controls are often implemented using access control lists (ACLs) or role-based access control (RBAC) systems.

Accountability:

Accountability is the ability to trace the actions of a user or system to a specific identity or entity. Accountability is typically achieved through the use of audit logs, which record a detailed history of events and actions taken within a system. Audit logs can be used to identify the source of security breaches, track user activity, and provide evidence for legal or regulatory compliance.

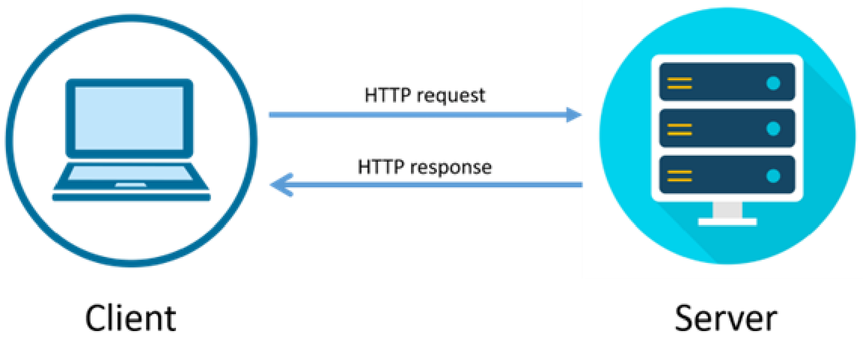


7.1.3 HTTP vs HTTPs

- **HTTP (Hyber-text transfer Protocol)**

HTTP is an application layer protocol based on the TCP/IP Model, it is a standard protocol used for transmitting data over the internet. It is an unencrypted protocol, which means that any data transmitted using HTTP can be intercepted and read by anyone who has access to the network.
This makes HTTP insecure for transmitting sensitive data such as passwords, credit card information, and other personal data. For more details about HTTP inner work refer to Chapter-1.

Remember that HTTP is a stateless protocol, that is why Web applications use sessions and cookies to keep track of the User. also, remember it is by default running on port 80 while HTTPs runs by default on 443



- **HTTPs**

HTTPS is a secure version of HTTP. It uses HTTP with SSL/TLS (Secure Socket Layer/Transport Layer Security), to encrypt data being transmitted between the client (web browser) and the server. This encryption makes it difficult for hackers to intercept and read the data being transmitted, ensuring the confidentiality and integrity of the data.

The SSL/TLS protocol involves a handshake process, which allows the client and server to authenticate each other's identities, negotiate the encryption algorithm and keys to be used, and establish the secure connection. During the handshake process, the client and server exchange digital certificates, which are used to verify each other's identities and establish trust. The client and server then agree on a shared secret key, which is used to encrypt and decrypt data being transmitted between them.

7.1.4 Self-signed certificate vs CA signed certificate

- **Self Signed Certificate**

A self-signed certificate is one that is signed by the same entity that issued it, without involving any third-party Certificate Authority. Self-signed certificates are typically used in testing and development environments, where security requirements are not as stringent.

- **Certificate Authority (CA) Signed Certificate**

This certificate is issued by a trusted third-party certificate authority (CA), which verifies the identity of the website owner before issuing the certificate. And it is trusted by default by your operating system and browser the user doesn't need to configure anything

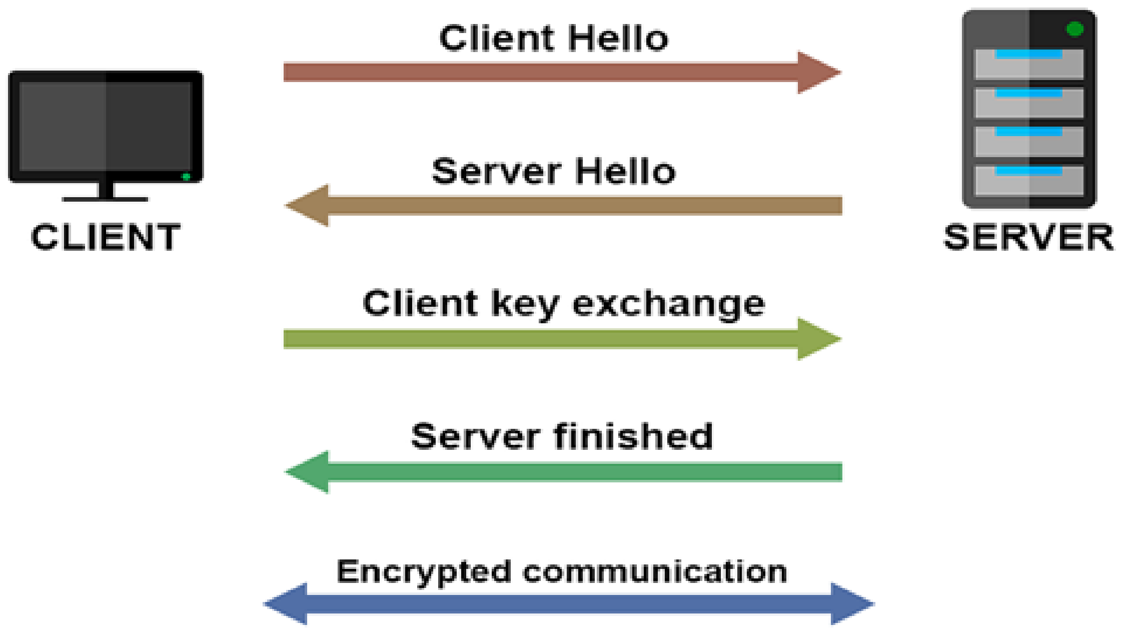
HTTPs Communication Steps

1. The client sends a "Client Hello" message to the server, which includes a list of supported cryptographic algorithms and a random number (called the "client random").
2. The server responds with a "Server Hello" message, which includes the chosen cryptographic algorithm, a random number (called the "server random"), and the server's SSL/TLS certificate (which contains the server's public key).
3. The client verifies the server's SSL/TLS certificate to ensure it was issued by a trusted authority and that the server's identity can be verified.
4. The client generates a random session key (called the "pre-master secret"), encrypts it with the server's public key obtained from the certificate and sends it to the server.
5. The server decrypts the pre-master secret using its private key, and then uses it to generate the shared secret key.
6. The client and server use the shared secret key to encrypt and decrypt all subsequent communication in the session.

HTTP VS HTTPs In Action

We will discuss the Wireshark tool latter, But in short, it shows us traffic in and out on a given interface on our device, here I am trying to log in to a web application it uses HTTP, and we can see the traffic is not encrypted and anyone who is between me and the Server hosting the web app can theoretically sniff the traffic and see the username and password provided as following

If we made the same web application uses HTTPs instead HTTP here is the traffic



To generate a Self-signed certificate for local testing follow these steps:

1. generates Certificate and keys using openssl library

   ```
   openssl req -newkey rsa:2048 -nodes -keyout server.key -x509 -days 365
   -out server.crt
   ```

2. start python HTTPs server using this code

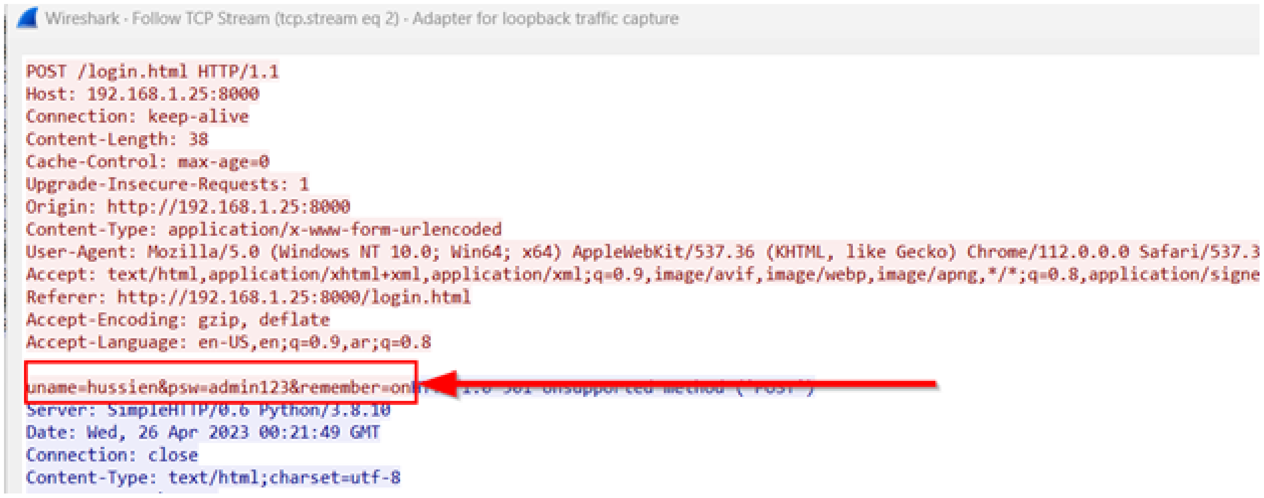
```
import http.server
import ssl
httpd = http.server.HTTPServer(('localhost', 443), http.server.SimpleHTTPRequestHan
httpd.socket = ssl.wrap_socket(httpd.socket, keyfile='server.key', certfile='server.crt',
server_side=True)
httpd.serve_forever()
```

And in case you have a web application and want to start an HTTP Server for local testing and development reasons you can do this easily in python using

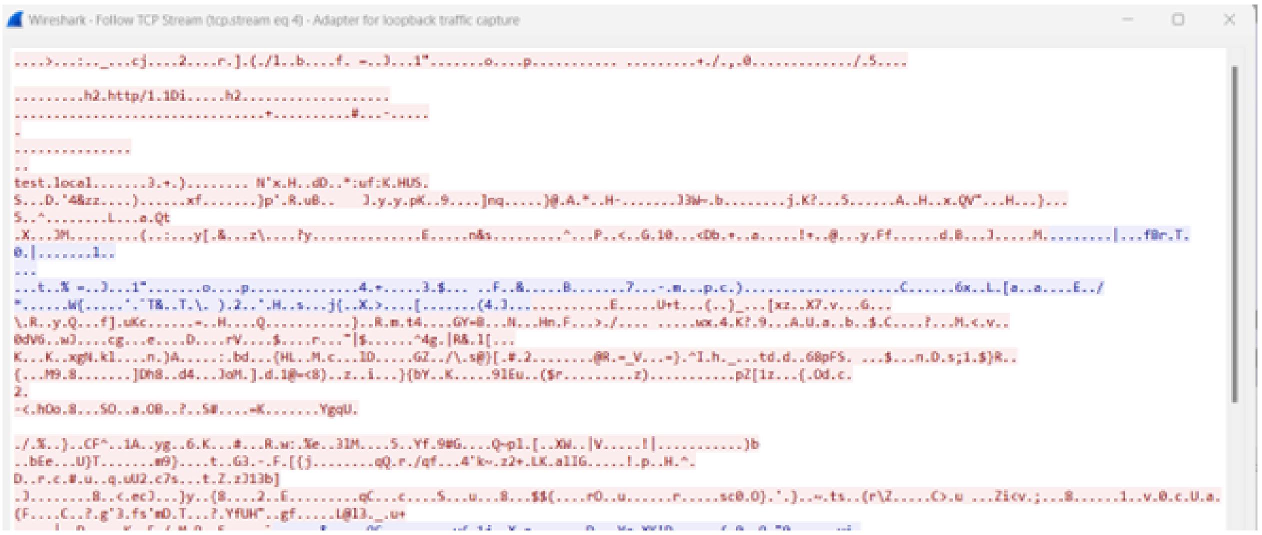
```
python3 -m http.server
```

7.1.5 JSON Web tokens (JWT)

Before we discuss what JWT is, we need to understand what a token is. Let's take an example to clarify it. As we have discussed HTTP is a stateless protocol


Wireshark · Follow TCP Stream (tcp.stream eq 2) · Adapter for loopback traffic capture

```
POST /login.html HTTP/1.1
Host: 192.168.1.25:8000
Connection: keep-alive
Content-Length: 38
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.1.25:8000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signe
Referer: http://192.168.1.25:8000/login.html
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,ar;q=0.8

uname=hussien&psw=admin123&remember=onHTTP/1.0 501 Unsupported method ('POST')
Server: SimpleHTTP/0.6 Python/3.8.10
Date: Wed, 26 Apr 2023 00:21:49 GMT
Connection: close
Content-Type: text/html;charset=utf-8
```


Wireshark · Follow TCP Stream (tcp.stream eq 4) · Adapter for loopback traffic capture

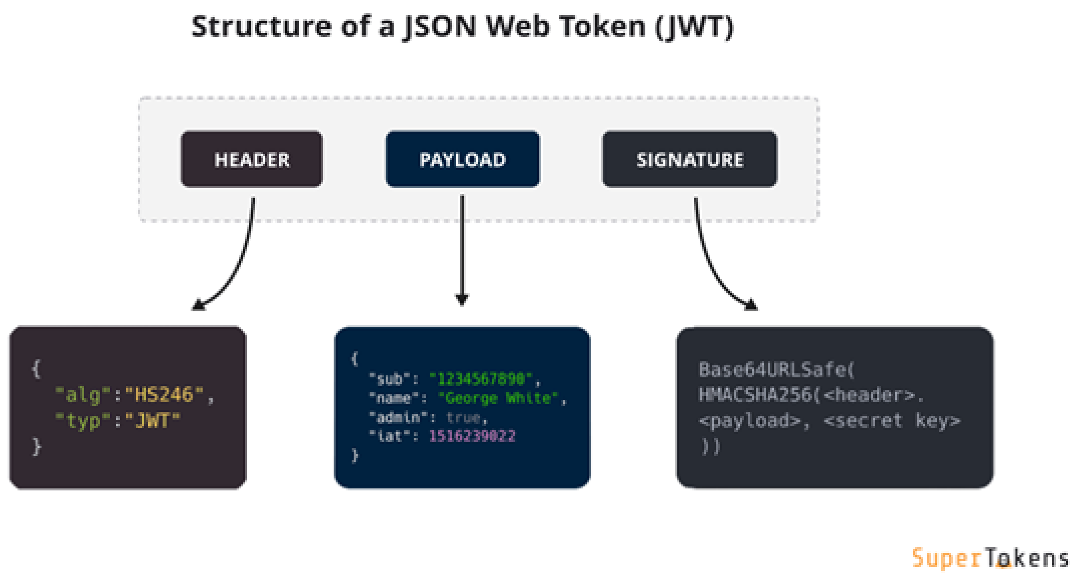
```
....>...:.._...cj....2....r.].(./1..b....f. -..}...1"........o....p........... ..........+./.,.0............./.5....

..........h2.http/1.1Di.....h2....................
.................................+...........#...-.....
.
................
..
test.local.......3.+.}........ N'x.H..dD..*:uf:K.HUS.
S...D.'4&zz....).......xf........}p'.R.uB..    ].y.y.pK..9....]nq.....}@.A.*..H-.......J3W-.b........j.K?...5.......A..H..x.QV"...H...}...
S..^........L...a.Qt
.X...JM..........(..:....y[.&...z\....?y..............E.....n&s.........^...P..<..G.10...<Db.+..a......!+..@...y.Ff.......d.B...J.....M.........|...f8r.T.
0.|........1..
...
...t..% =..}...1".......o....p...............4.+......3.$... ..F..&.....B.......7...-.m...p.c.)...................C.......6x..L.[a..a.....E../
*.......W(.....'.'T&..T.\. ).2..'.H..s...j(..X.>....[........(4.}...........E.....U+t....(..)_...[xz..X7.v...G...
\.R..y.Q...f].uKc.......=..H....Q............}..R.m.t4....GY=B...N...Hn.F....>./.... .....wx.4.K?.9...A.U.a..b..$.C.......?....M.<.v..
0dV6..wJ....cg...e....D....rV....$.....r...'|$......^4g.|R&.1[...
K...K..xgN.kl....n.)A......:.bd...(HL..M.c....1D.....GZ../\.s@}[.#.2........@R.=_V...-}.^I.h._...td.d..68pFS. ...$...n.D.s;1.$}R..
{...M9.8........]Dh8..d4...JoM.].d.1@=<8)..z..i...}{bY..K.....91Eu..($r..........z)...........pZ[1z...{.0d.c.
2.
-c.hOo.8...SO..a.OB..?..S#.....K........YgqU.

./.%..}..CF^..1A..yg..6.K...#...R.w:.%e..31M....5..Yf.9#G....Q-p1.[..XW..|V.....!|............)b
..bEe...U}T........m9}.....t..G3.-.F.[{j.........qQ.r./qf....4'k=.z2+.LK.a1IG......!.p..H.^.
D..r.c.#.u..q.uU2.c7s...t.Z.z]13b]
.J.........8..<.ec}...}y..{8....2..E..........qC...c.....S...u....8...$$(.....r0..u.......r......sc0.0}.'.}..-.ts..(r\Z......C>.u ...Z1<v.;....8.......1..v.0.c.U.a.
(F....C..?.g'3.fs'mD.T....?.YfUH"..gf......L@13._.u+
```

so it doesn't recognize users. if a user logged in to the web application, then navigates to another page in the same web application, it is not the smartest approach to ask him again for his credentials!

That is where sessions come to play, sessions are used when a user logged in to a website he will be assigned a session cookie so when he visits another web page he will make an HTTP request that has the session cookie in it so the web app will recognize that user. sessions are stored on the server side to track the users activity and maintain the user session at the front end.

Sessions are known to be a random string that will be assigned for a user and that is true, but as we have mentioned it will be stored on the Server side to track users, So why JWT?

JWT is more complex than creating a session, it contains 3 parts as follows:



```
{
  "alg":"HS246",
  "typ":"JWT"
}
```

```
{
  "sub": "1234567890",
  "name": "George White",
  "admin": true,
  "iat": 1516239022
}
```

```
Base64URLSafe(
HMACSHA256(<header>.
<payload>, <secret key>
))
```

Here is an example of such tokens

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 . eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ . SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

The first part is the base64 encoded version of the algorithm used and the type of the token, second part is the base64 encoded version of the payload also known as claims and it contains data about a user for example

{
"name": "John Doe",
"iat": 1516239022
}

Note that it field refers to the issued at the time and the value is the Epoch timestamp, this can value can be helpful in tracking tokens and expiry

The third part is the applying of the algorithm and it is most often signed by a strong complex secret key that is hard to guess by attackers.

The beauty of JWT is that it is not stored on the Server which allows more scalability and faster communication, less storage, and more importantly better security because attackers will not be able to tamper with the claims/data because they don't know the secret key.

Now we need to know How the Server is validating the tokens if it is not stored there?

- Decode the first 2 parts of the token.
- Apply the algorithm on the server side with the secret key.
- Compare the result with the third part of the token.

7.1.6 What is API

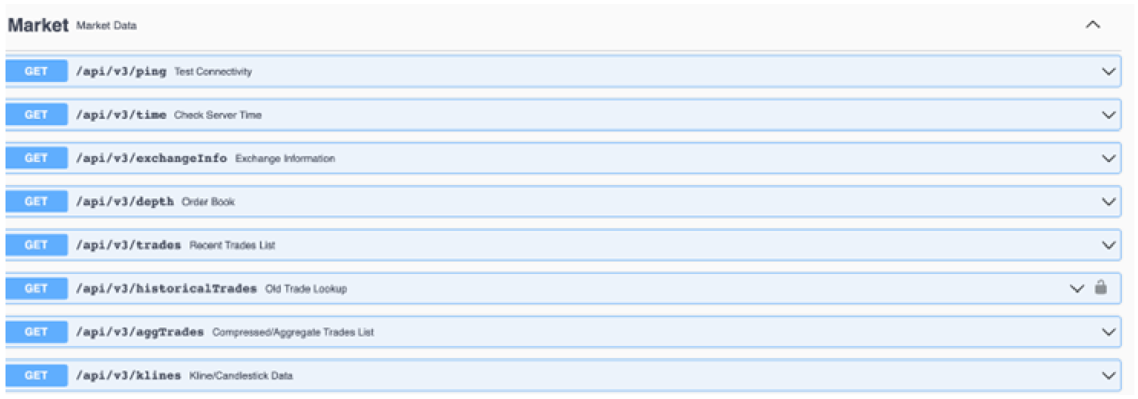
API stands for Application Programming Interface. An API is a set of protocols, routines, and tools for building software applications. It defines how different software components should interact with each other.

In other words, an API acts as an interface between different software applications, allowing them to communicate with each other and exchange data. APIs can be used to access or manipulate data from a third-party application or service, such as social media platforms, payment gateways, or weather APIs.

APIs can be classified into different categories based on their functionality and level of access. For example, public APIs are open to everyone and allow developers to build applications that access a service's data or functionality. Private APIs, on the other hand, are only accessible to specific users or organizations.

APIs can be accessed through different methods, such as HTTP requests, RESTful APIs, or GraphQL. They can return data in various formats, including JSON, XML, and CSV.

For Example, a web application can use some public API to retrieve recent books released and then display it inside the app, and there are way more complex examples where API can play a critical role in the web application



Market Market Data

GET /api/v3/ping Test Connectivity
GET /api/v3/time Check Server Time
GET /api/v3/exchangeInfo Exchange Information
GET /api/v3/depth Order Book
GET /api/v3/trades Recent Trades List
GET /api/v3/historicalTrades Old Trade Lookup
GET /api/v3/aggTrades Compressed/Aggregate Trades List
GET /api/v3/klines Kline/Candlestick Data

7.1.7 Tools

Wireshark

Wireshark is a popular network protocol analyzer tool used to capture, analyze, and troubleshoot network traffic in real time. It is an open-source tool that runs on multiple platforms, including Windows, Linux, and macOS.

It is important to understand that Wireshark capabilities end at the device you are running it from, it simply sniffs the traffic that passes through your network card interface, and the traffic will pass your interface in 2 cases whether this traffic is being sent explicitly to you or a broadcasted traffic, so if a user-x communicating with user-y in normal cases you will not be able to sniff this kind of traffic

Snort

Snort is an open-source network intrusion detection and prevention system that is widely used for detecting and preventing attacks on computer networks.

Snort operates by analyzing network traffic in real time and comparing it against a set of rules to identify potential security threats. It can be configured to alert network administrators or take automated action when an attack is detected.

Postman

Postman is a popular API development and testing tool used by developers to design, build, test, and document APIs. It is available as a standalone desktop application, as well as a browser extension.

With Postman, developers can easily make HTTP requests to an API endpoint and view the response, as well as set request headers, parameters, and authentication details. It also supports automated testing, allowing developers to write test scripts that can be run to ensure that the API is functioning correctly.

Flask Framework

Flask is a lightweight and flexible Python web framework used for building web applications. Flask is designed to be simple and easy to use, while still providing developers with the tools they need to build robust web applications. It is popular among developers due to its minimalistic approach, which allows for rapid development and customization.

Flask provides features such as routing, request handling, and template rendering, as well as support for extensions that can add functionality such as authentication, database integration, and form handling. It also provides a built-in development server for testing and debugging.

Ryu controller

During this chapter, we will obtain the RYU controller case study because :

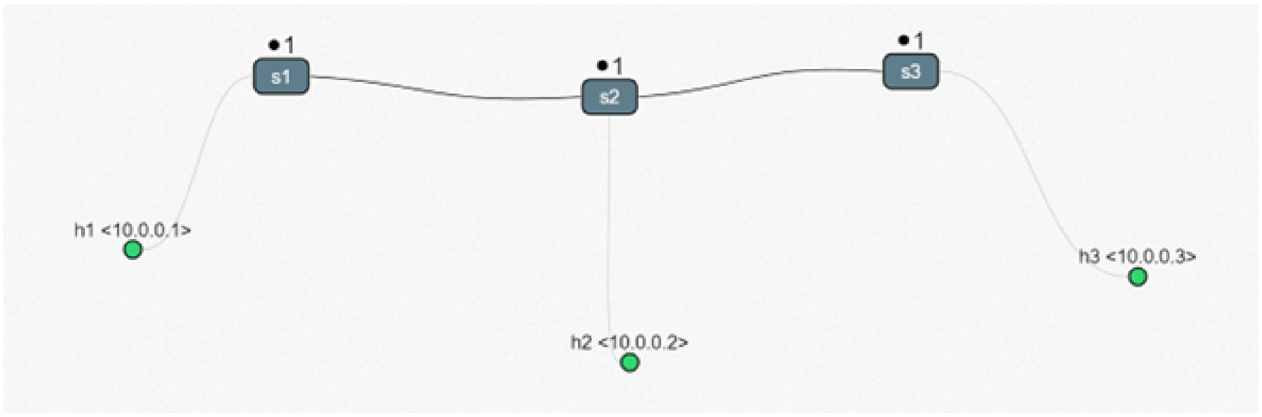
- Ryu is an open source controller with a large community.
- It is widely used for building and managing SDN networks.
- It is maintained by developers who are always introducing new features.
- Ryu is highly customizable, providing flexibility to users.
- It can be integrated with other security tools like Snort.
- Applications are relatively easy to implement compared to others.
- It is the best choice for experimentation as it is easy to reinstall and debug errors.

7.2 SDN Network Problems Showcase

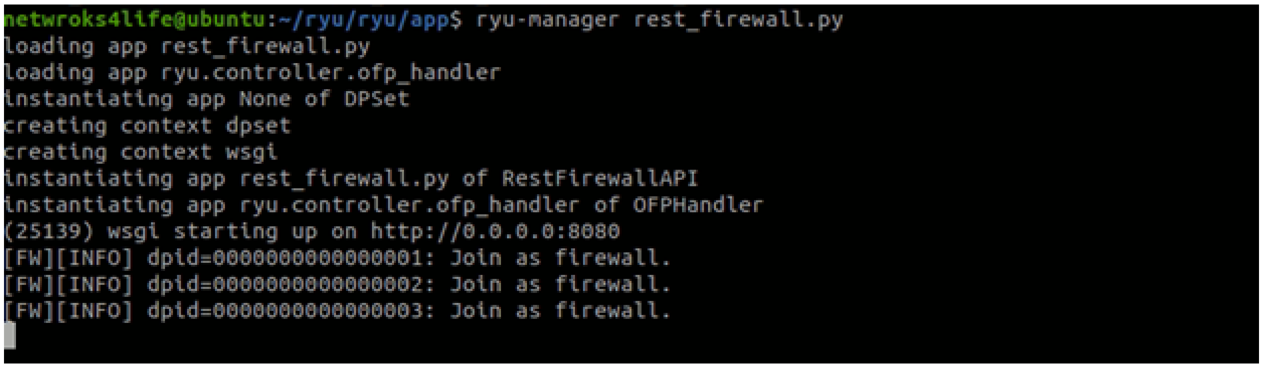
7.2.1 Setting up the environment for Testing

To showcase the problems we will work with the most basic setup, we will work on the same machine for now (we are working on an Ubuntu 20.04.5 machine, The IP of this machine is **192.168.38.130**)

```
sudo mn –topo linear,3 –controller remote
```



We can now run the RYU controller on the same machine, we can start it with any application we want (From the ones we have discussed in chapter 5), Let's use the Firewall one because it shows the problem more clearly.



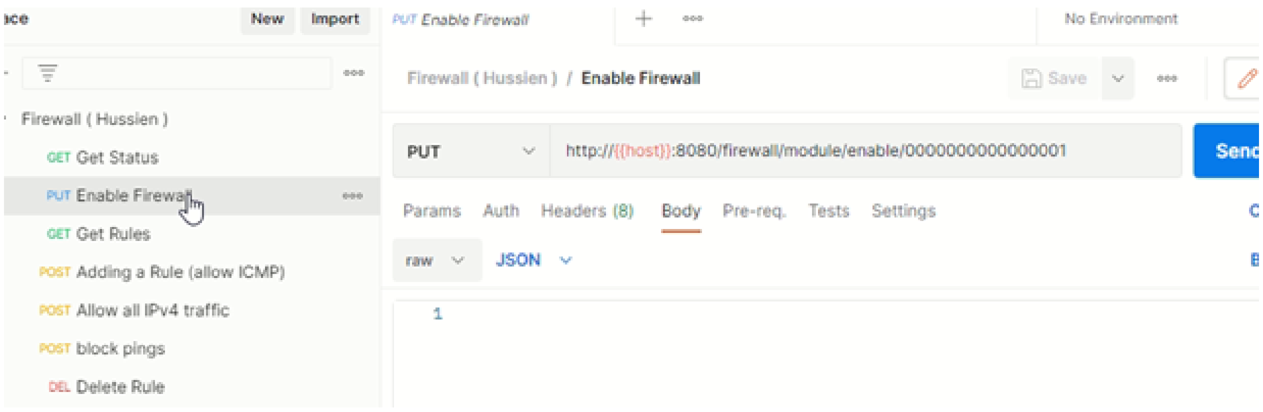
```
netwroks4life@ubuntu:~/ryu/ryu/app$ ryu-manager rest_firewall.py
loading app rest_firewall.py
loading app ryu.controller.ofp_handler
instantiating app None of DPSet
creating context dpset
creating context wsgi
instantiating app rest_firewall.py of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(25139) wsgi starting up on http://0.0.0.0:8080
[FW][INFO] dpid=0000000000000001: Join as firewall.
[FW][INFO] dpid=0000000000000002: Join as firewall.
[FW][INFO] dpid=0000000000000003: Join as firewall.
```

The default Firewall from the Ryu developers can be Found Here and that is the version we will test with (initially)

Now we need to set up the way we are going to communicate with the firewall rest API, Firewall is working based on HTTP Requests being sent to it to take actions on the switches, for examples if we want to enable the firewall on switch-1 we will issue an HTTP Request with the method PUT to this endpoint

http://{{host}}:8080/firewall/module/enable/{{switch_id}}

This instruction can be found in the Rest Firewall API documentation Here, as we have mentioned earlier in Chapter 5 we have created our collection in POSTMAN so instead of re-writing the commands each time we have saved them also POSTMAN provides more convenient way to interact with them APIs



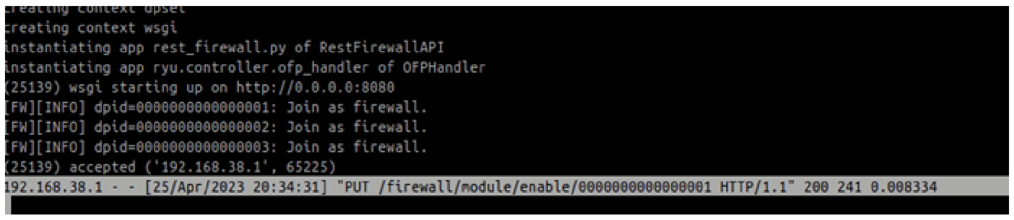
The last thing we want to have it running is Wireshark to monitor traffic flow

7.2.2 Identifying Flaws

After setting up our environment, Let's assume a normal case where an administrator wants to take actions on the firewall. What will he do?

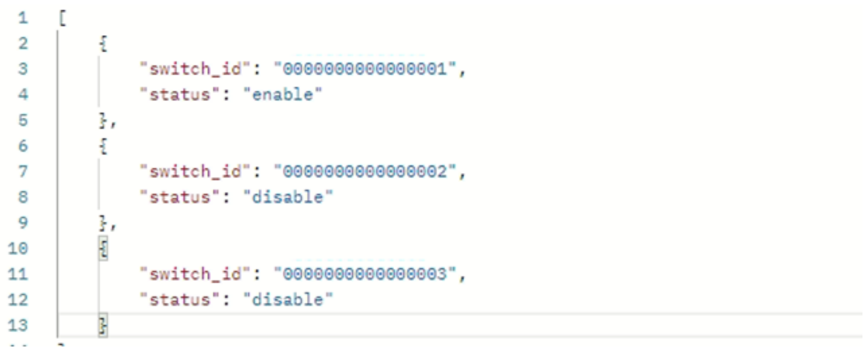
- He will issue the HTTP requests we have discussed earlier to take action.

- For example, let's enable the firewall.



```
creating context dpset
creating context wsgi
instantiating app rest_firewall.py of RestFirewallAPI
instantiating app ryu.controller.ofp_handler of OFPHandler
(25139) wsgi starting up on http://0.0.0.0:8080
[FW][INFO] dpid=0000000000000001: Join as firewall.
[FW][INFO] dpid=0000000000000002: Join as firewall.
[FW][INFO] dpid=0000000000000003: Join as firewall.
(25139) accepted ('192.168.38.1', 65225)
192.168.38.1 - - [25/Apr/2023 20:34:31] "PUT /firewall/module/enable/0000000000000001 HTTP/1.1" 200 241 0.008334
```

- And in the Ryu rest Firewall application we can see in the Logs that a request has been issued and the Firewall is enabled indeed , we can confirm that by issuing a request to get the Firewall Status, we have only enabled switch 1 that's why it is the only one being enabled



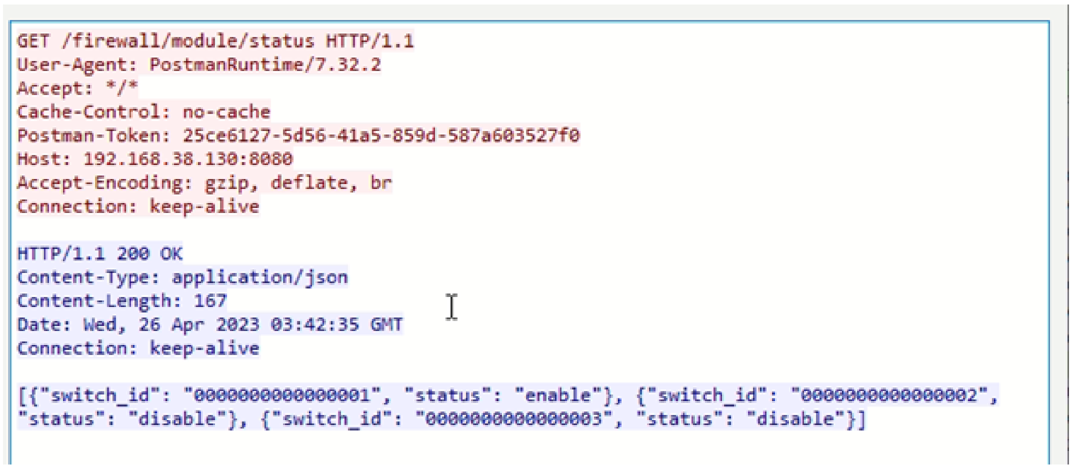
```
[
    {
        "switch_id": "0000000000000001",
        "status": "enable"
    },
    {
        "switch_id": "0000000000000002",
        "status": "disable"
    },
    {
        "switch_id": "0000000000000003",
        "status": "disable"
    }
```

Problem (1): if you focused enough on the logs above you will see the IP Address issuing the request is not the Ryu Machine IP address, it is another machine in the LAN which means any machine in the LAN can control the Firewall As long As it has reachability over the Ryu machine!

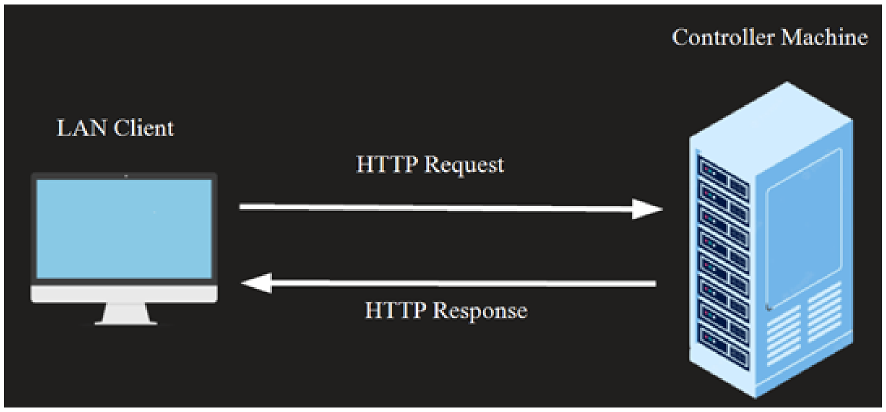
Impact: We can take this to another level and not only enable or disable the firewall But setting rules to allow/disallow traffic between users in the SDN Network!

Problem (2): Taking a look at wireshark to see the traffic being transmitted, we can see it is HTTP Traffic and the command/Results are clear text!
Impact: transmitting such sensitive information in clear text could open the door for Man in The Middle attacks which in turn can allow attackers to intercept the traffic and have access to it.

Problem (3): The current configuration is as follows, it is a client-server system architecture, the client is directly communicating with the controller machine. It is not a big issue but a lot of things can go wrong Like



```
GET /firewall/module/status HTTP/1.1
User-Agent: PostmanRuntime/7.32.2
Accept: */*
Cache-Control: no-cache
Postman-Token: 25ce6127-5d56-41a5-859d-587a603527f0
Host: 192.168.38.130:8080
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 167
Date: Wed, 26 Apr 2023 03:42:35 GMT
Connection: keep-alive

[{"switch_id": "0000000000000001", "status": "enable"}, {"switch_id": "0000000000000002",
"status": "disable"}, {"switch_id": "0000000000000003", "status": "disable"}]
```



ImpactClients are often reliant on the server for all processing and storage, which can result in a slower user experience if the client is on a slow or unreliable network connection.

As The number of clients increases, the server may become overwhelmed and unable to handle all the requests, leading to slower response times or even server crashes.

The client and server communicate directly with each other, which means that the client needs to have some level of trust in the server. If the server is compromised, the attacker may gain access to sensitive data stored on the server.

7.3 Securing SDN Network Methodology

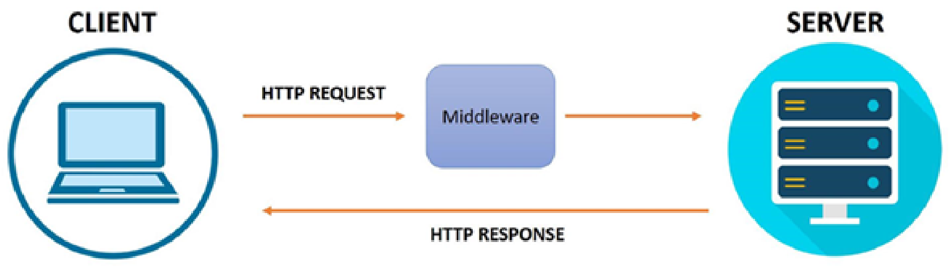
7.3.1 Approach summary

In the next sections we will go through the details of the implementation, But in summary, we are going to do the following

- Implement our new version of the Firewall REST API that supports Authorization (i.e., will not issue commands on the firewall unless a JWT

Token is in place).

- Implement a middleware between controller and Clients by using a Flask web application that applies Authentication.
- Add TLS/SSL over HTTP for the web application to protect against MITM attacks.

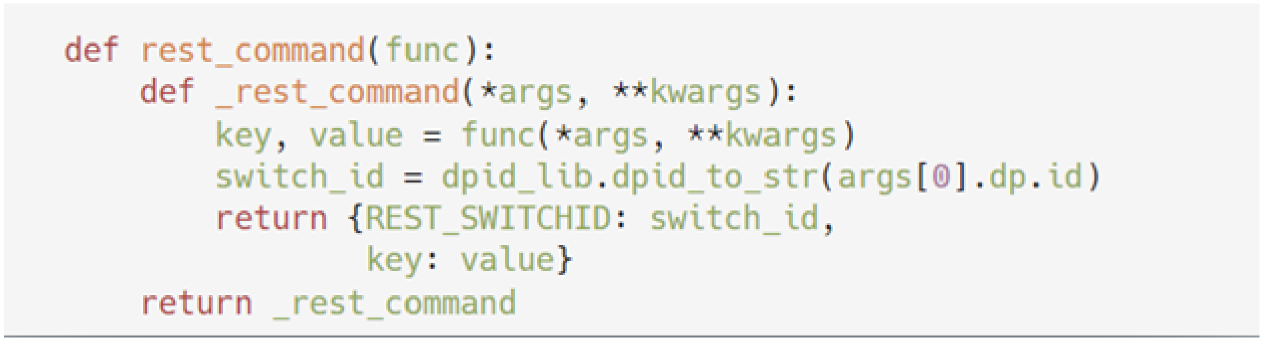


7.3.2 Secure Firewall communication implementation

In This Part we are going to make our own version of the rest firewall script provided by RYU itself, The original can be found here. We want to achieve the following eventually:

- Do not execute commands on the firewall unless a token is in place.
- Verify that this token is a valid JWT token.

Analyzing the Firewall's original source code we found this part which is basically a decorator that when we use any function in the firewall (for example get_status, set_disable_flow, etc) this function is responsible for mapping the request to the corresponding function with the arguments.



```
def rest_command(func):
    def _rest_command(*args, **kwargs):
        key, value = func(*args, **kwargs)
        switch_id = dpid_lib.dpid_to_str(args[0].dp.id)
        return {REST_SWITCHID: switch_id,
                key: value}
    return _rest_command
```

We can confirm as we find the @rest_command will be executed before any function in the firewall

```
    @rest_command
    def get_status(self, waiters):
        msgs = self.ofctl.get_flow_stats(self.dp, waiters)

        status = REST_STATUS_ENABLE
        if str(self.dp.id) in msgs:
            flow_stats = msgs[str(self.dp.id)]
            for flow_stat in flow_stats:
                if flow_stat['priority'] == STATUS_FLOW_PRIORITY:
                    status = REST_STATUS_DISABLE

        return REST_STATUS, status

    @rest_command
    def set_disable_flow(self):
        cookie = 0
        priority = STATUS_FLOW_PRIORITY
        match = {}
        actions = []
        flow = self._to_of_flow(cookie=cookie, priority=priority,
                                match=match, actions=actions)

        cmd = self.dp.ofproto.OFPFC_ADD
        self.ofctl.mod_flow_entry(self.dp, flow, cmd)
```

As the rest_command function is somehow a centralized function that will be called before any function call in the code, we can add our Authorization There, so The Execution Flow for the code will be as following:

- We call some function in the firewall by sending a request to do a certain function.
- Before the function gets executed, it will call the rest_command function.
- The Rest Command will do the Authorization check part, as we will see later.
- If the user is not authorized, the requested function will not take place.

Here is the new rest_command function which will do the following before taking any action

- Check if the function we want to execute needs authorization.
- If so, check if the 'Auth' parameter is passed with the arguments.
- The Auth token contains JWT value that needs to be checked.
- If all these conditions return true, the user will receive the "Successfully Authenticated" message.

For the JWT Function it is pretty standard as we mentioned it will just try to recalculate the Token to check if it is a valid one , also note that the secret used in this Firewall code must be the same secret used in The Flask application

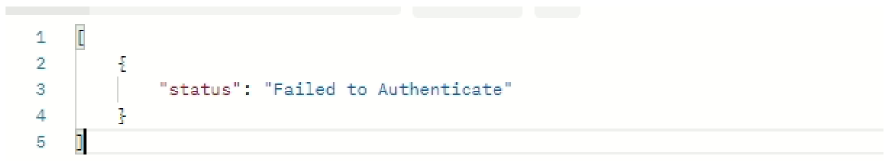
```
    def rest_command(func):
        def _rest_command(*args, **kwargs):
          print(args)
          print(func.__name__)
          if func.__name__ not in ['get_status' , 'get_rules' , 'set_disable_flow'
,'set_log_enable','set_enable_flow','get_log_status'] :
            if len(args) >1 and 'Auth' in  args[1].keys() and check_jwt(args[1]['Auth']) :
              FirewallController._LOGGER.info('Successfully Authenticated')
            else :
              FirewallController._LOGGER.info('no Authenticated')
              return {'status' : 'Failed to Authenticate'}

          key, value = func(*args, **kwargs)
          switch_id = dpid_lib.dpid_to_str(args[0].dp.id)
          return {REST_SWITCHID: switch_id,
                  key: value}
        return _rest_command
```

```
def check_jwt(token):
    secret_key = '[REMOVED]'
    # Define a JWT token to decode
    try:
        # Decode the token using the secret key
        decoded_token = jwt.decode(token, secret_key, algorithms=['HS256'])
        return True
    except jwt.ExpiredSignatureError:
        return False
    except jwt.InvalidSignatureError:
        return False
    except:
        return False
```

(that will send the request to the Controller)

Now we can check our Firewall Functionality by using our new updated code, we can send a request to add a Rule using POSTMAN or using a curl command from the terminal as follows

```
curl 'http://192.168.38.130:8080/firewall/rules/0000000000000001' \
–header 'Content-Type: application/json' \
–data '{
  "nw_src": "10.0.0.2/32",
  "nw_dst": "10.0.0.1/32",
  "nw_proto": "ICMP"
}'
```



```
[
    {
        "status": "Failed to Authenticate"
    }
]
```

So we have achieved our goal by making the firewall REST API Not execute critical functions without an Authorization parameter being sent, we can try for now to add it before implementing the middleware to make sure it is working all right.

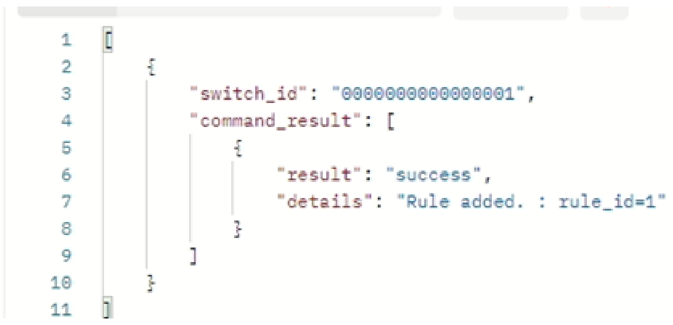
This python code will return a JWT Token that we can use in the firewall request as the value of the Auth parameter

```
mport jwt
token = jwt.encode({}, b'[REMOVED]', algorithm='HS256')
print(token)
```

Trying to send the request with the following format

```
curl 'http://192.168.38.130:8080/firewall/rules/0000000000000001' \
–header 'Content-Type: application/json' \
–data '{
  "nw_src": "10.0.0.2/32",
  "nw_dst": "10.0.0.1/32",
  "nw_proto": "ICMP" ,
 "Auth" :"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.e30.4zpj7GmQwAA-
Crqo1DRhhdNsxcFVeNtU9RwEObRBIHM"
}'
```

The Rule has been added indeed which confirms we are on the right track



```
[
    {
        "switch_id": "0000000000000001",
        "command_result": [
            {
                "result": "success",
                "details": "Rule added. : rule_id=1"
            }
        ]
    }
]
```

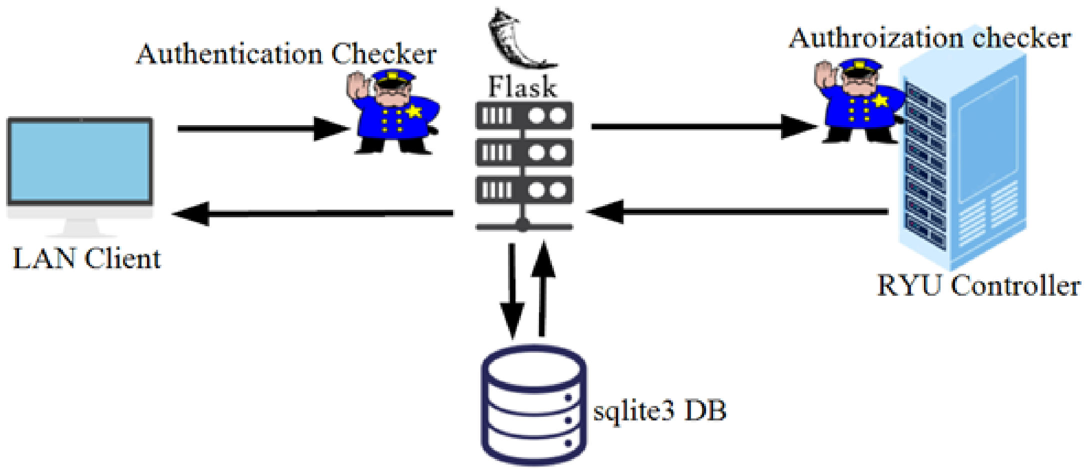
But we can't rely on the previous approach forever. This is just to test the functionality because as you have seen we have sent empty claims/payload meaning there is no data which is not what we want ultimately. But even this testing is better than the default firewall configurations we discussed earlier.

7.3.3 Authentication Implementation in RYU Controller Based networks

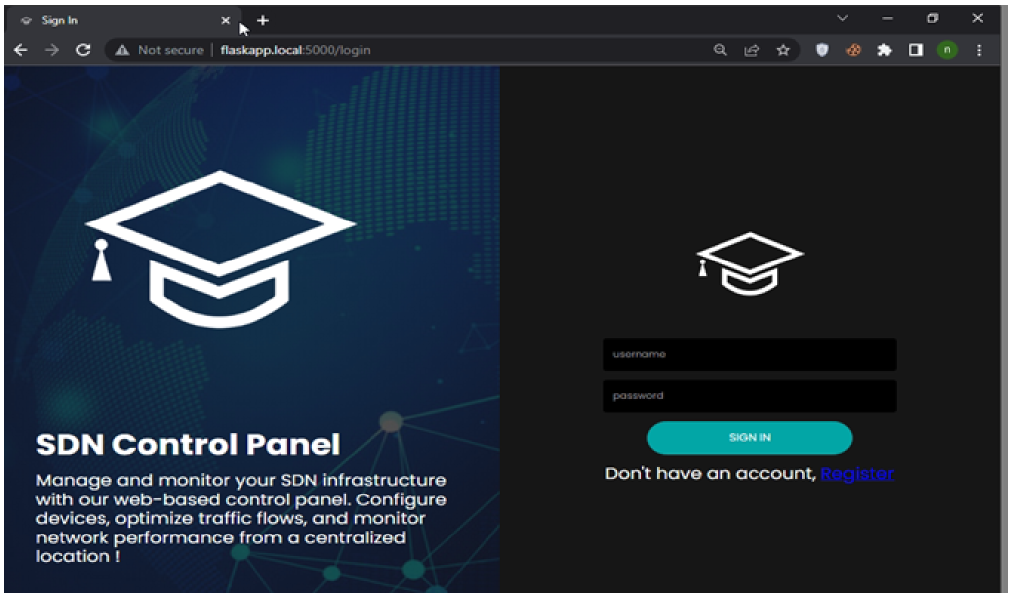
7.3.3.1 Flask Application workflow

We will adopt the following approach, as we have seen earlier we have made it impossible for normal LAN Clients to send commands to the RYU Controller directly, they have to interact with the FLASK application First which we implement in the following part. The application has the following features:

- Login page so only administrators can interact with RYU.
- Register function for registering administrators.
- Firewall management interface (better than Postman and terminal commands).
- JWT Tokens generator.
- Logout to terminate the session.



The Application contains the following login page which performs the authentication guard rule.

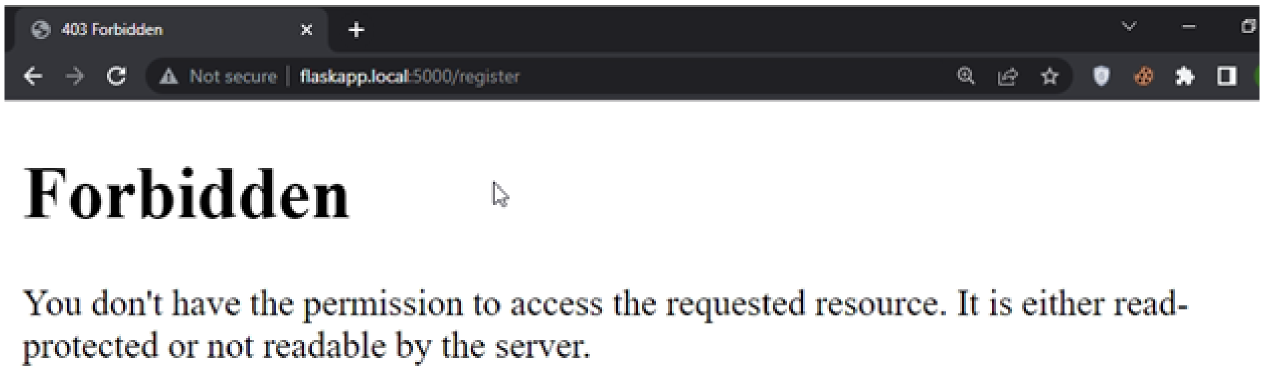


SDN Control Panel

Manage and monitor your SDN infrastructure with our web-based control panel. Configure devices, optimize traffic flows, and monitor network performance from a centralized location !

Don't have an account, Register

We need to register an account first, checking the registration page we got forbidden access



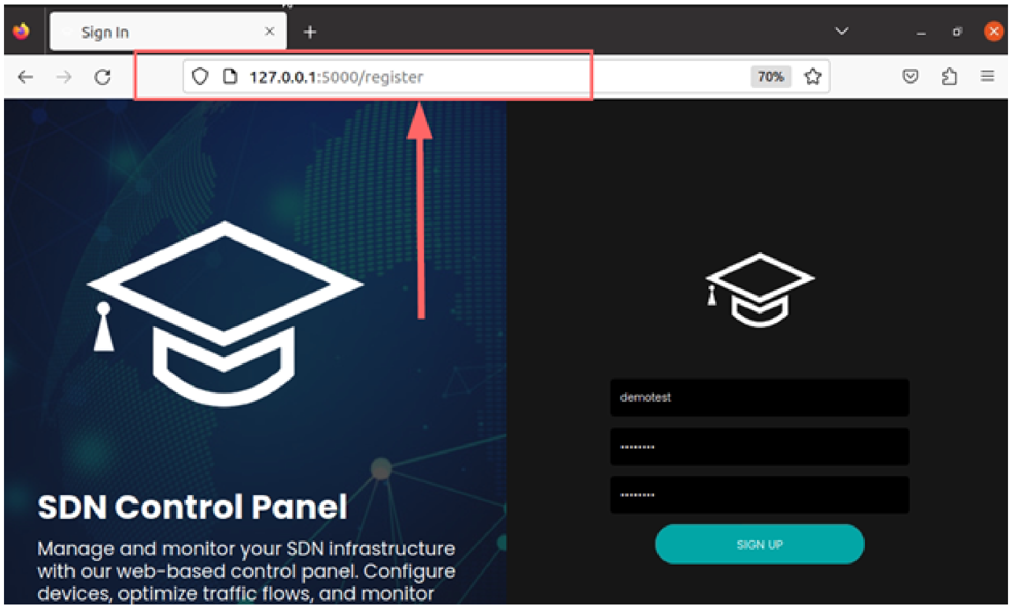
Forbidden

You don't have the permission to access the requested resource. It is either read-protected or not readable by the server.

As the dashboard will allow the logged-in user to perform an action on the network he should be an administrator, and making the register function public to anyone in the LAN will make anyone register himself an admin account and do the action so the whole thing has made will be actually useless. For this, we made the following

```
@app.route('/register', methods=['GET'])

def register_get():
    if request.remote_addr == '127.0.0.1' :
       return render_template('register.html')
    else :
        return abort(403)
```

Only if the user wants to register is accessing the page from localhost (meaning he is in front of the machine running the web application) he will be able to access the registration page, otherwise he will have the forbidden message code 403

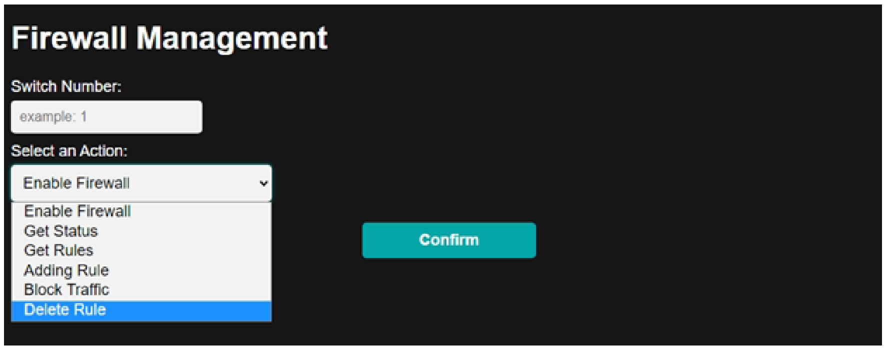

Sign In
127.0.0.1:5000/register
70%
SDN Control Panel
Manage and monitor your SDN infrastructure with our web-based control panel. Configure devices, optimize traffic flows, and monitor
demotest
SIGN UP

After Registering an account, the username, and the hash of the password will be stored in the database Now after registering a user from the localhost ma-

```
conn = sqlite3.connect('users.db')
c = conn.cursor()
c.execute("SELECT * FROM users WHERE username=?", (username,))
result = c.fetchone()
if result:
# if the username already exists, return an error message
   return jsonify({'message': 'Username already exists'}), 409
c.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed_password))
conn.commit()
conn.close()
return jsonify({'message': 'User created successfully'}), 201
```

chine, we can log in from any machine in the network which allows **Remote Administration.**

After logging in we are introduced to the following dashboard

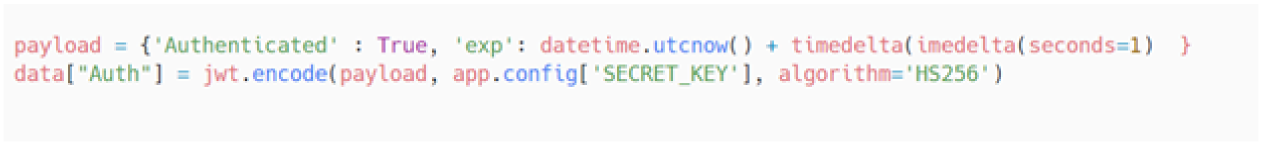


When we select any option following requests will occur:

- Front-end interface will send a request to the Flask application.
- Flask application will:
  - First check if the user is authenticated.
  - Second, add the Auth parameter to the data.
- Send the final request to the Ryu controller.

7.3.3.2 JWT Generating process

We Have learned how to generate and decrypt JWT tokens, and as we have mentioned it contains the claims/payload, in our case we will make the payload describe if the current user is Authenticated and also contains expiration time and in this case, it is one second only, so when the flask app make a request to the controller providing the JWT token this token will be useless after one second leaving no chance for stolen tokens and reuse tokens.



```
payload = {'Authenticated' : True, 'exp': datetime.utcnow() + timedelta(imedelta(seconds=1)  }
data["Auth"] = jwt.encode(payload, app.config['SECRET_KEY'], algorithm='HS256')
```

Every time the administrator makes a request to the FLASK APP it will generate a new JWT token valid for 1 second and use it against the RYU Controller.

Note that in the JWT token is never transmitted in the traffic between the client and the middleware, it is only transmitted between the middleware and the Ryu Controller

7.3.3.3 Secure Firewall in action

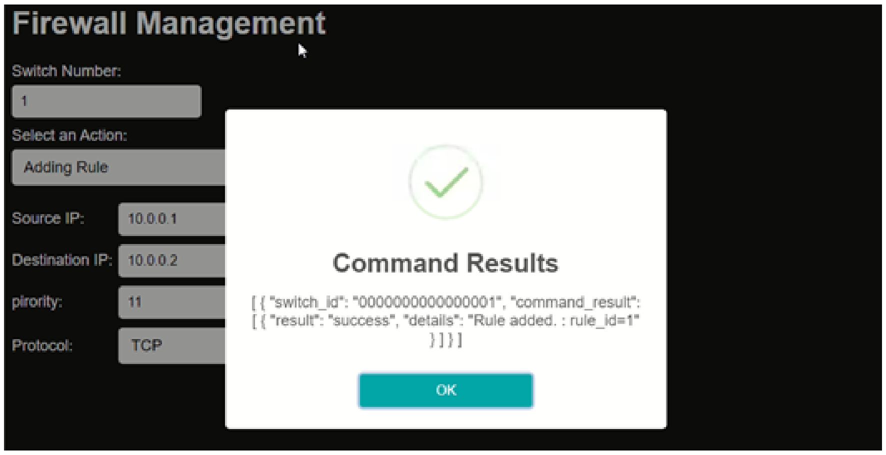
So to recap what we have done before showing an example

- Implemented a dashboard that requires authentication.
- Implemented a Flask application that adds authorization to firewall requests.
- Implemented firewall REST API code that checks authorization before taking actions.
- Isolated the Ryu controller from clients.

Let's now try to add a Rule using the dashboard, we need to set the fields

- Switch number.
- Choosing the action.
- Set source IP address.
- Set destination IP address.
- Set priority.
- Set one of the protocols (TCP, ICMP, ICMPv6, UDP, IPv6).

And after sending the request we will see the result in the alert window as follows

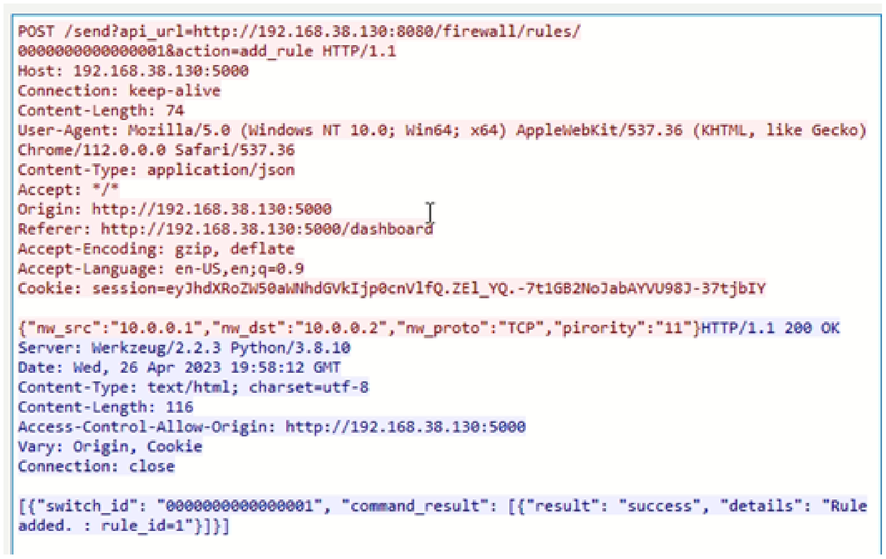


Switch Number: 1
Select an Action: Adding Rule
Source IP: 10.0.0.1
Destination IP: 10.0.0.2
pirority: 11
Protocol: TCP

Command Results

```
[ { "switch_id": "0000000000000001", "command_result":
[ { "result": "success", "details": "Rule added. : rule_id=1"
} ] } ]
```

If we use Wireshark to see how is the request issued and the corresponding response will see the following

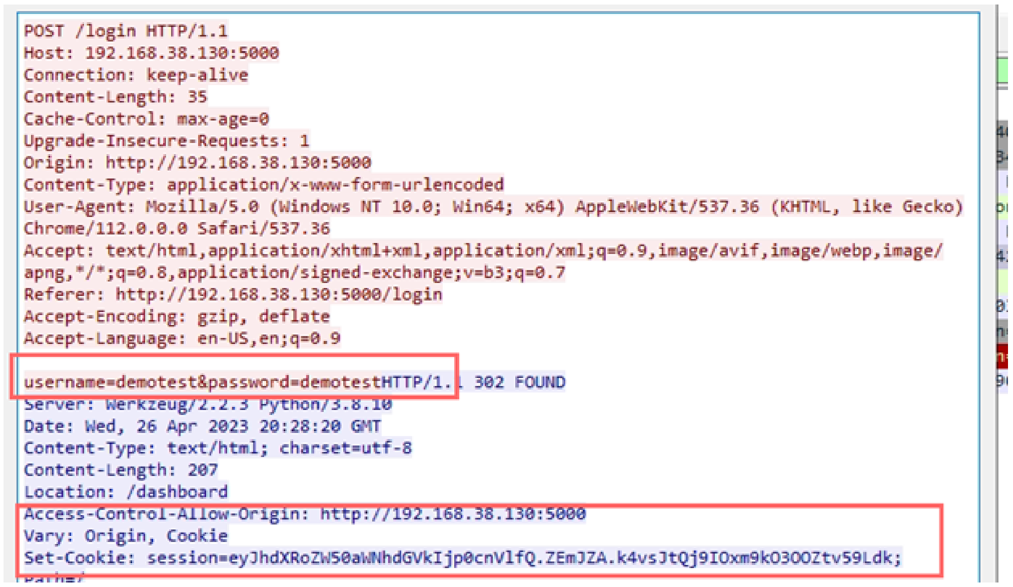


```
POST /send?api_url=http://192.168.38.130:8080/firewall/rules/
0000000000000001&action=add_rule HTTP/1.1
Host: 192.168.38.130:5000
Connection: keep-alive
Content-Length: 74
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/112.0.0.0 Safari/537.36
Content-Type: application/json
Accept: */*
Origin: http://192.168.38.130:5000
Referer: http://192.168.38.130:5000/dashboard
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: session=eyJhdXRoZW50aWNhdGVkIjp0cnVlfQ.ZEl_YQ.-7t1GB2NoJabAYVU98J-37tjbIY

{"nw_src":"10.0.0.1","nw_dst":"10.0.0.2","nw_proto":"TCP","pirority":"11"}HTTP/1.1 200 OK
Server: Werkzeug/2.2.3 Python/3.8.10
Date: Wed, 26 Apr 2023 19:58:12 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 116
Access-Control-Allow-Origin: http://192.168.38.130:5000
Vary: Origin, Cookie
Connection: close

[{"switch_id": "0000000000000001", "command_result": [{"result": "success", "details": "Rule
added. : rule_id=1"}]}]
```

As you see the client request doesn't contain a JWT token, note that a session cookie is different from JWT Tokens, a session cookie is used to maintain user session after the login process. And in the Response, we can see the output From the Firewall API as we have seen before.

7.3.4 Running With HTTPs

What we have done so far is a pretty good approach to secure the environment, However, all of this can be useless because we are using HTTP, let's examine the possible points of attacks:



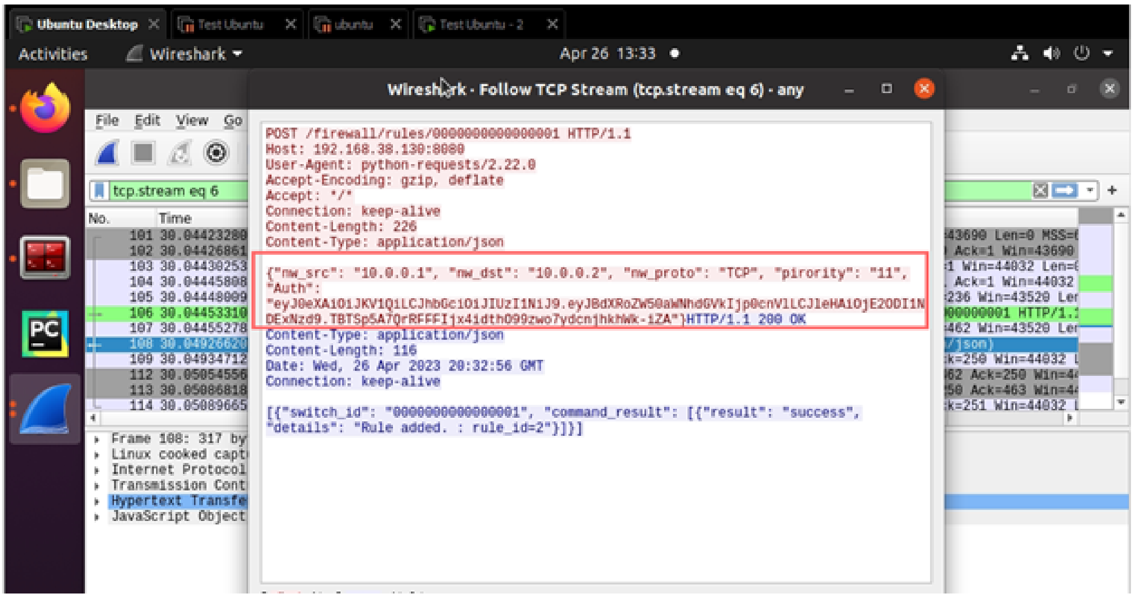
```
POST /login HTTP/1.1
Host: 192.168.38.130:5000
Connection: keep-alive
Content-Length: 35
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.38.130:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/112.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/
apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://192.168.38.130:5000/login
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9

username=demotest&password=demotestHTTP/1.1 302 FOUND
Server: Werkzeug/2.2.3 Python/3.8.10
Date: Wed, 26 Apr 2023 20:28:20 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 207
Location: /dashboard
Access-Control-Allow-Origin: http://192.168.38.130:5000
Vary: Origin, Cookie
Set-Cookie: session=eyJhdXRoZW50aWNhdGVkIjp0cnVlfQ.ZEmJZA.k4vsJtQj9IOxm9kO3OOZtv59Ldk;
```

Man in The Middle attacks will allow malicious actors to obtain our credentials and gain administrator access easily.

In the Webserver machine, if we run Wireshark to see the issued requests when a user makes an action at the dashboard we will see the JWT Tokens are Transmitted in clear text, Although this token is valid only for one second's attackers can still automate the process of stealing the Authorization token value and take an action on behalf of other administrators

As we have mentioned before we can need a certificate to implement the HTTPs and it has 2 types From CA and self-signed ones. We will use a self-signed certificate because this is only within the local area network and not publicly exposed.

To know How to generate the Self-signed Certificate kindly refer to 7.3.1 HTTP vs HTTPs section in this chapter.



```
POST /firewall/rules/0000000000000001 HTTP/1.1
Host: 192.168.38.130:8080
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 226
Content-Type: application/json

{"nw_src": "10.0.0.1", "nw_dst": "10.0.0.2", "nw_proto": "TCP", "pirority": "11",
"Auth":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJBdXRoZW50aWNhdGVkIjp0cnVlLCJleHAiOjE2ODI1N
DExNzd9.TBTSp5A7QrRFFFIjx4idth099zwo7ydcnjhkhWk-1ZA"}HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 116
Date: Wed, 26 Apr 2023 20:32:56 GMT
Connection: keep-alive

[{"switch_id": "0000000000000001", "command_result": [{"result": "success",
"details": "Rule added. : rule_id=2"}]}]
```

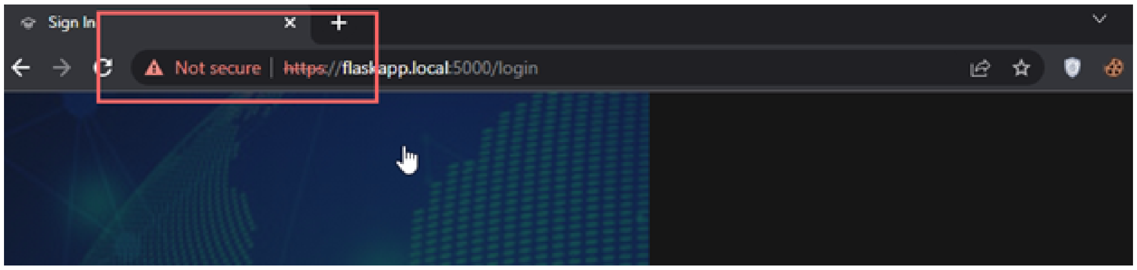
To solve This issue HTTPs comes into play.

After generating the Certificate we need to make a change in the Flask code application, we need to add paths to the key.pem and cert.pem at the run application part in the code

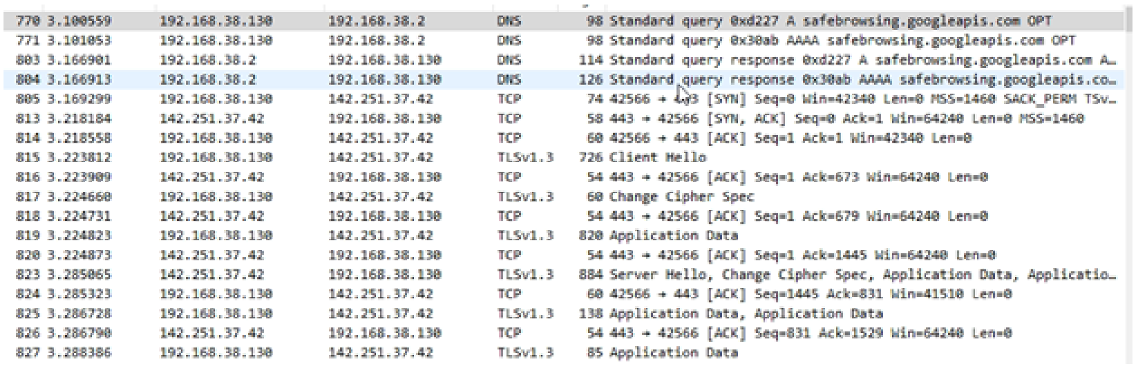
```
if __name__ == '__main__':

    app.run(host="0.0.0.0",debug=True ,  ssl_context=('certs/cert.pem', 'certs/key.pem'))
```

Key.pem contains the private key that the server uses to encrypt/decrypt ,The private key is kept secret and should be protected from unauthorized access. while **Cert.pem** is the digital certificate itself

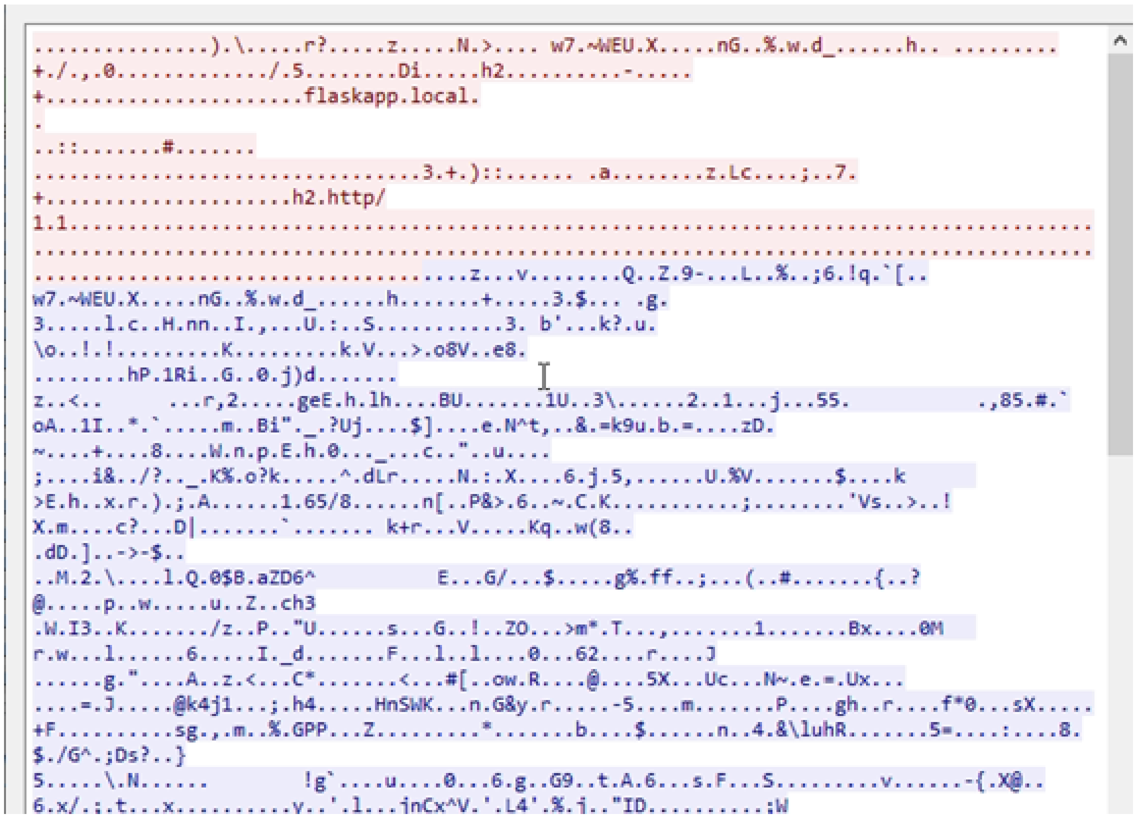
After running the app we can now use HTTPs , it is okay for the Not Secure alert at the browser because the certificate is not signed by the CA.



Now Let's see how the Traffic flows in Wireshark , Trying to Login as we did before we can find the following traffic issued which is the HTTPs communication steps



```
770 3.100559   192.168.38.130   192.168.38.2     DNS       98 Standard query 0xd227 A safebrowsing.googleapis.com OPT
771 3.101053   192.168.38.130   192.168.38.2     DNS       98 Standard query 0x30ab AAAA safebrowsing.googleapis.com OPT
803 3.166901   192.168.38.2     192.168.38.130   DNS      114 Standard query response 0xd227 A safebrowsing.googleapis.com A…
804 3.166913   192.168.38.2     192.168.38.130   DNS      126 Standard query response 0x30ab AAAA safebrowsing.googleapis.co…
805 3.169299   192.168.38.130   142.251.37.42    TCP       74 42566 → 443 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSv…
813 3.218184   142.251.37.42    192.168.38.130   TCP       58 443 → 42566 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
814 3.218558   192.168.38.130   142.251.37.42    TCP       60 42566 → 443 [ACK] Seq=1 Ack=1 Win=42340 Len=0
815 3.223812   192.168.38.130   142.251.37.42    TLSv1.3  726 Client Hello
816 3.223909   142.251.37.42    192.168.38.130   TCP       54 443 → 42566 [ACK] Seq=1 Ack=673 Win=64240 Len=0
817 3.224660   192.168.38.130   142.251.37.42    TLSv1.3   60 Change Cipher Spec
818 3.224731   142.251.37.42    192.168.38.130   TCP       54 443 → 42566 [ACK] Seq=1 Ack=679 Win=64240 Len=0
819 3.224823   192.168.38.130   142.251.37.42    TLSv1.3  820 Application Data
820 3.224873   142.251.37.42    192.168.38.130   TCP       54 443 → 42566 [ACK] Seq=1 Ack=1445 Win=64240 Len=0
823 3.285065   142.251.37.42    192.168.38.130   TLSv1.3  884 Server Hello, Change Cipher Spec, Application Data, Applicatio…
824 3.285323   192.168.38.130   142.251.37.42    TCP       60 42566 → 443 [ACK] Seq=1445 Ack=831 Win=41510 Len=0
825 3.286728   192.168.38.130   142.251.37.42    TLSv1.3  138 Application Data, Application Data
826 3.286790   142.251.37.42    192.168.38.130   TCP       54 443 → 42566 [ACK] Seq=831 Ack=1529 Win=64240 Len=0
827 3.288386   192.168.38.130   142.251.37.42    TLSv1.3   85 Application Data
```



```
...............).\.....r?.....z.....N.>.... w7.~WEU.X.....nG..%.w.d_......h.. .........
+./.,.0............./.5........Di.....h2..........-.....
+.....................flaskapp.local.
.
..::.......#.......
................................3.+.)::...... .a........z.Lc....;..7.
+.....................h2.http/
1.1....................................................................................
.......................................................................................
...................................z...v........Q..Z.9-...L..%..;6.!q.`[..
w7.~WEU.X.....nG..%.w.d_......h.......+.....3.$... .g.
3.....l.c..H.nn..I.,...U.:..S...........3. b'...k?.u.
\o..!.!.........K.........k.V...>.o8V..e8.
........hP.1Ri..G..0.j)d.......
z..<..      ...r,2.....geE.h.lh....BU.......1U..3\......2..1...j...55.          .,85.#.`
oA..1I..*.`.....m..Bi"._.?Uj....$]....e.N^t,..&.=k9u.b.=....zD.
~....+....8....W.n.p.E.h.0..._...c.."..u....
;....i&../?.._.K%.o?k.....^.dLr.....N.:.X....6.j.5,......U.%V.......$....k
>E.h..x.r.).;.A......1.65/8......n[..P&>.6..~.C.K...........;........'Vs..>..!
X.m....c?...D|.......`....... k+r...V.....Kq..w(8..
.dD.]..->-$..
..M.2.\....l.Q.0$B.aZD6^          E...G/...$.....g%.ff..;...(..#.......{..?
@.....p..w.....u..Z..ch3
.W.I3..K......./z..P.."U......s...G..!..ZO...>m*.T...,.......1.......Bx....0M
r.w...l......6.....I._d.......F...l..l....0...62....r....J
......g."....A..z.<...C*.......<...#[..ow.R....@....5X...Uc...N~.e.=.Ux...
....=.J.....@k4j1...;.h4.....HnSWK...n.G&y.r......-5....m.......P....gh..r....f*0...sX.....
+F..........sg.,.m..%.GPP...Z.........*.......b....$......n..4.&\luhR.......5=....:....8.
$./G^.;Ds?..}
5.....\.N......        !g`....u....0...6.g..G9..t.A.6...s.F...S.........v.......-{.X@..
6.x/.;.t...x..........y..'.l...jnCx^V.'.L4'.%.j.."ID..........;W
```