Pentesting Report: <Siemens LOGO! 12/24 RCE>

Daniel Benton (2007018) , Owain Edwards (1924129), Samuel Calvesbert (2474172),
Tobias Neugebauer (2583271), Alexandru-Catalin Radut (1932842)

August 30, 2023

Contents

List of abbreviations

PLC Programmable Logic Controller

IoT Internet of Things

CVSS Common Vulnerability Scoring System

DoS Denial-of-service

ISP Internet service provider

MitM Man-in-the-Middle

RPC remote procedure call

NIST National Institute of Standards and Technology

XSS cross-site scripting

1 Executive summary

In this report we discover several flaws in the security of the examined device. Most of them are not critical and can be easily mitigated by configuring the device properly.
The most interesting discovery of our report is the missing integrity checks for user requests. Therefore, we are able to demonstrate a session token sniffing attack. By obtaining a session token any unauthorised user, of the same local network as the examined device, is able to send requests to the device and take over control.

The report start off with a high-level description of the device in section 2. In the following the investigate the device during setup in subsection 3.1 and try to understand the ongoing services in subsection 3.2. After exploring the different services and functionality of the device we take a look at the possible attacks as a consequence in section 4. In section 5 we analyse the discovered attacks and put them into context of feasibility.

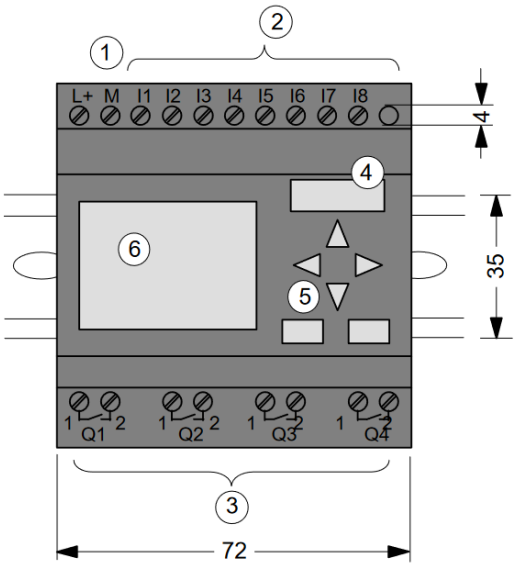
1.1 Recommendations for a hardened device

To summarise our findings, we give the following advice to a user who is not able to influence the release of any fixes in the foreseeable future:

1. Pick unique randomly generated passwords with the max length
2. Disable TDE access
3. If not necessary, deactivate the web interface
4. Assuming the web-interface is used, deactivate HTTP and use HTTPS
5. Do not expose the Programmable Logic Controller (PLC) to the Internet, use VPNs to connect to the PLC instead
6. Set LSC & LWE password
7. Activate the operator mode as soon as the PLC is not being configured anymore

2 High-level description of the device

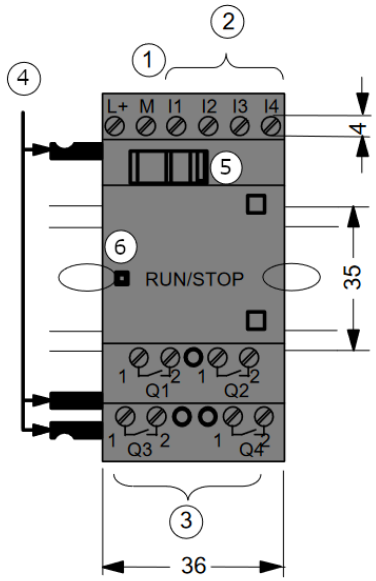
The examined device is a PLC. A PLC is used in industry for controlling industrial machines. The examined model consists of the following components, shown in Figure 1:



1. Power supply input
2. Inputs (i.e. sensors)
3. Outputs (relays with a maximum output of 4A)
4. Module slot with cap
5. Control panel
6. LCD display
7. An Ethernet port (not visible)

Figure 1: Overview of the Siemens LOGO! 12/24 RCE

Additionally, the LOGO! DM8 12/24R expansion module is used, adding four more inputs as well as outputs as shown in Figure 2.



1. Power supply
2. Inputs
3. Outputs
4. Mechanical coding pins
5. RUN/STOP indicator
6. Slide

Figure 2: The LOGO! DM8 12/24R expansion module

The basic idea of a PLC is to receive input data from sensors and decide on which output, a logical one or zero, is sent out. This enables the user to orchestrate factory functionality

via written programs and control machines depending on the inputs received. The user can program the PLC with the provided software "LOGO! Soft Comfort". The program can be started and stopped via the control panel (see Figure 1)or with the programming software and a computer in the same network.
A PLC is usually mounted on rails and located in a control room. It is possible to extend the device with more expansion modules or connect different main modules and let them interact with each other over Ethernet. This way, the user can potentially control complex machinery with great amounts of inputs.

3 Investigating the device

3.1 Analysing the device setup

A PLC, in comparison to other Internet of Things (IoT) devices that are made for home usage, is more difficult to set up and not plug-and-play. The LOGO! 12/24 RCE needs a 12 or 24 V DC power supply. Additionally, all input sensors and outputs/actuators incorporated into the system also require a power supply. The outputs act as a relay and trigger the desired response by controlling the ON/OFF signals perceived by the actuators.

3.1.1 Simulating a real-world environment

To simulate a real environment, we have used an Industrial Control Work-Cell by LJ Create. The so called Industrial Control Trainer is visible in Figure 3. The Work-Cell provides a motor driving a treadmill (1), two motion sensors with different altitudes (2 & 3), three air-pressure controlled pistons (4), the mounted PLC (5), a green light (6), a red light (7), a green button (8), a red button (9) and the air supply container (10).

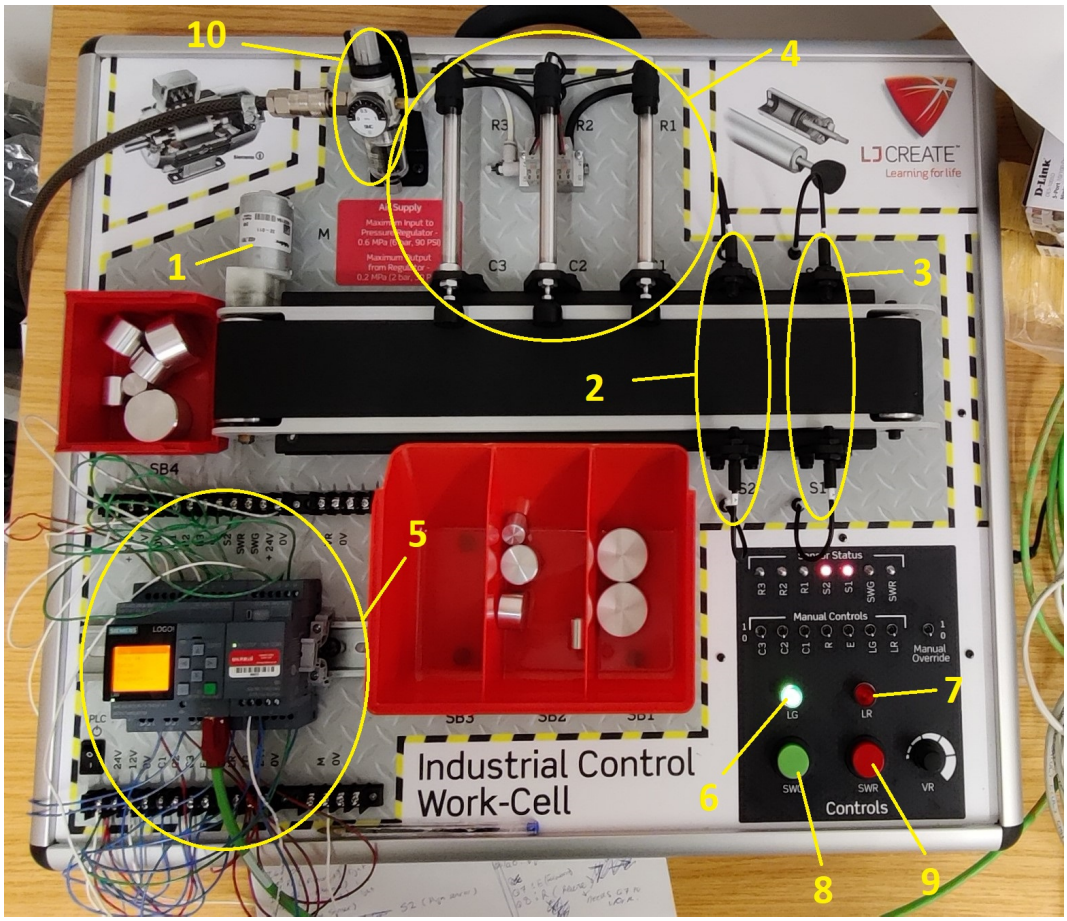


Figure 3: The PLC in the Industrial Control Work-Cell

To use the PLC appropriate wiring is needed. Therefore, we attach the power supply, the

provided sensors and the outputs to the PLC as seen in Figure 4.

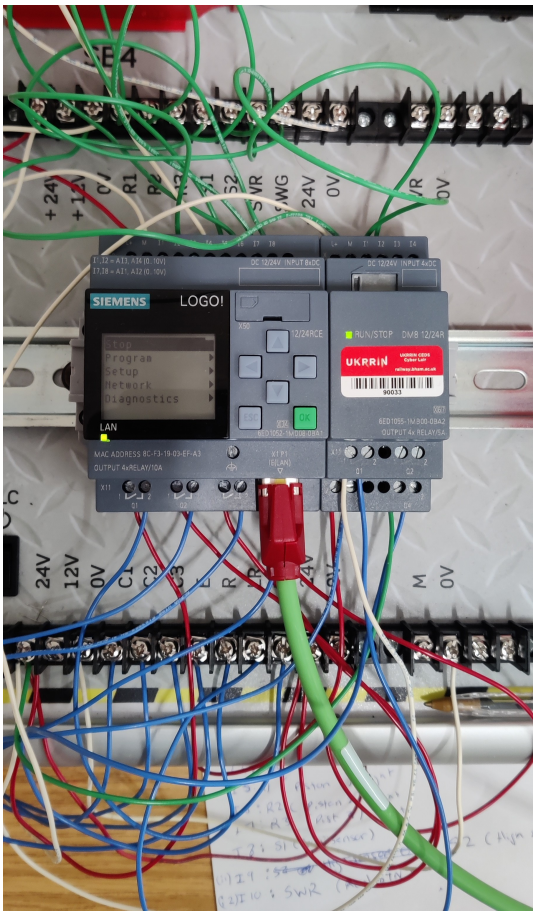


Figure 4: The wiring

3.1.2 The network setup

In the end the Ethernet cable is attached. The Ethernet cable connects the PLC to a switch. We connect both the computer running the LOGO! Soft Comfort software and the computer on which we conduct our measurements and tests, to the switch. We set an initial IP address for the PLC in the menu of the device (these can be changed at a later point via the LOGO! Soft Comfort software once networking is established). Notice that it is crucial to assign a unique IP address and set the IP addresses of the network interfaces of the connected computers accordingly. Duplicate IP addresses must be avoided. We finish the initial setup by turning the Industrial Control Work-Cell on and pinging the PLC from both connected computers. If an answer is received, networking has successfully been established and we can continue on to the programming. PLC.

3.1.3 Creating a program

With the LOGO! Soft Comfort software it is possible to create programs using logic gates. We decide to create a program doing the following:

1. Start the treadmill and light up the green light when the green button is pressed
2. Stop the treadmill, turn off the green light and turn the red light on when the red button is pressed
3. If the lower sensor (marked '3' in Figure 3) is broken but the higher one is not, fire the first piston after a certain delay
4. If both sensors are broken fire the second piston after a certain delay

As a result this program demonstrates the ability to sort the weights visible in Figure 3 by two heights in their respective trays. Figure 5 shows the created program. Additionally a demo of the working program can be seen here. (0:00 to 0:18).

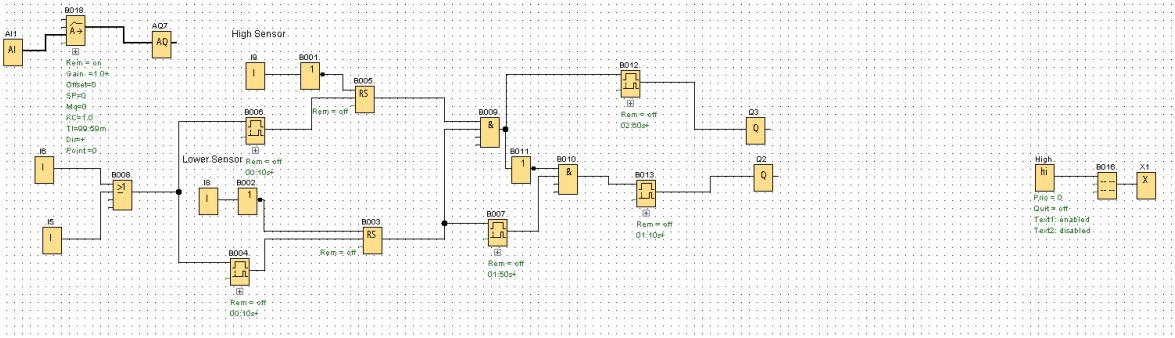


Figure 5: The created program in gate logic

In the program we take advantage of a so called SR latch, which is capable of storing binary values. If a motion sensor is broken it changes the value of its corresponding SR latch. The program waits for a specified amount of time to give the second latch long enough to change its value, assuming the second sensor will be broken as well (meaning a weight with the higher option of altitude is passing along the conveyor belt). If this happens, the second piston is fired. Alternatively, if a weight with the lower option of altitude is travelling along the belt, it will only trigger the lower altitude sensor, this meaning only the first piston is activated. No matter which piston is fired, both SR latches are reset to their initial states.

3.2 Analysing the device in use

Since there are a lot of settings potentially affecting and/or depriving the security of the examined device we start by enumerating characteristics of the device when reset to factory settings. The PLC uses version 1.83.01 of the firmware. There is one newer version, which has been released 08/02/2022. Since the download of the newer version 1.83.02 is restricted and the changelog does not mention security relevant improvements, we stick to version 1.83.01 and can deem any discovered vulnerabilities as remaining prevalent in current versions of the device.

3.2.1 Ports

To examine the open ports by default we reset the PLC to factory settings. After assigning an IP address we conduct a nmap scan using the following parameters:

```
nmap -d -v -A -p- -oA nmapfactorysettings 169.254.161.213
```

With the options `-d` and `-v` we increase the verbose and debugging level to get as much information as possible. `-A` activates OS detection, version detection, script scanning and traceroute. With `-p-` we will probe every port of the examined device. The entire scan can be viewed in A.1.
According to the scan there are three open ports, port 80, 135, and 8443. Port 80 is used by the PLC to provide a readme file in the browser via HTTP. The same site is available on port 8443 using HTTPS. Port 135 is used for communication to TDEs. TDEs are optional text displays to display more information of the PLC. The used protocol is remote procedure call (RPC) which is supported by our discovery of a RPC directory in 3.2.2. This port is considered insecure, giving attackers the ability to modify configuration files. Siemens recommends disabling the port but this is not done by default.[1]
Depending on the settings port 102 may be open as well. It is opened if a program needs S7 access rights. S7 is a proprietary protocol by Siemens to communicate between different PLCs. Siemens itself considers port 102 insecure and warns the user when attempting to open that port. Nonetheless, for our example program we were forced to open port 102 to transfer it onto the PLC.
If the PLC is set to slave mode the ports 102 and 502-510 are opened. They are needed to listen to modbus communication. Modbus is a communication protocol implementing master/slave communications.
If the web-interface (see 3.5) is activated and switched to HTTPS, port 8443 gets replaced by port 443. Port 80 stays open all the time.

3.2.2 Directories

To start further investigation we conduct a directory search. We use `gobuster` with the popular medium list included with Kali Linux.[2]. The results are as follows:

```
/ajax                 (Status: 200)  [Size: 0]
/rpc                  (Status: 405)  [Size: 0]
/AJAX                 (Status: 200)  [Size: 0]
/RPC                  (Status: 405)  [Size: 0]
/Ajax                 (Status: 200)  [Size: 0]
/%3FRID%3D2671        (Status: 200)  [Size: 8254]
/%3F%3F               (Status: 200)  [Size: 8254]
/%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F%3F  (Status: 200)  [Size: 8254]
/%3Fmethod%3Declou3    (Status: 200)  [Size: 8254]
/%3Fmethod%3Dbanner    (Status: 200)  [Size: 8254]
/%3f                  (Status: 200)  [Size: 8254]
```

Obviously, the PLC uses AJAX to dynamically update its web-interface with the current status of the PLC. The directory RPC hints us about the used protocol. Since port 135 is open and we find a directory called RPC we conclude the protocol used to communicate with TDEs is RPC.

[1] `https://cert-portal.siemens.com/productcert/pdf/ssa-817401.pdf`

[2] `https://github.com/daviddias/node-dirbuster/blob/master/lists/directory-list-2.3-medium.txt`

3.3 Access restrictions

The PLC offers different services to avoid unauthorised access. To be precise the following services with password restrictions are offered:

1. LSC & LWE
2. Webserver access
3. Webserver access guest
4. LOGO! APP user
5. TDE

The first password is the one needed to overwrite the software with LOGO! Soft Comfort or the LOGO! Web Editor. It is not used by default.
The web-server passwords are restricting access to the web interface. They are turned on as long as a web interface is activated. The guest is not allowed to modify any values of the variables.
The 'LOGO! APP' is the software for Mac-OS and access can be restricted with a password. The TDE password protects the PLC from unrestricted operation. The PLC offers two modes, the admin and the operator mode. In the admin mode, resets or network changes can be applied. Additionally, the user is able to start or interrupt programs. The TDE password applies no matter if the TDE access is activated or not. Therefore we conclude, that the TDE access is for external TDEs and the password applies to external as well as to usage via the internal menu.

3.4 Known CVEs and attacks

3.4.1 Information gatherer

The German company SySS GmbH developed a proof-of-concept script[3] to demonstrate the possibility of gathering every set password. The attack was disclosed in 2019 and Siemens released a security advisory[4]. The provided script does not work anymore. Therefore, we conclude the disclosed vulnerabilities have been fixed in the meantime.

3.5 Web interface

The PLC hosts different web pages depending on the applied settings. The user can decide if they a) activate the web interface and b) if they activate HTTPS. Assuming they do not activate the web interface a readme page is hosted, as seen in Figure 6. We can access this readme page through port 80 and HTTP or port 8443 and HTTPS.

If the web interface is activated it allows a user to login either from the internal network or remotely. They can change the value of variables and control the in- and output values of the PLC as well as write to the virtual memory of the PLC.

The underlying web server is called `LOS HTTP Server 1.0`. Since we do not find valuable information on Google or Shodan which give us a hint for the usage of this type of server,

[3] `https://github.com/SySS-Research/slig`
[4] `https://cert-portal.siemens.com/productcert/pdf/ssa-542701.pdf`

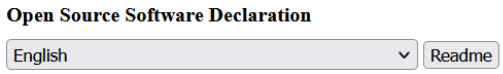


Figure 6: The readme page

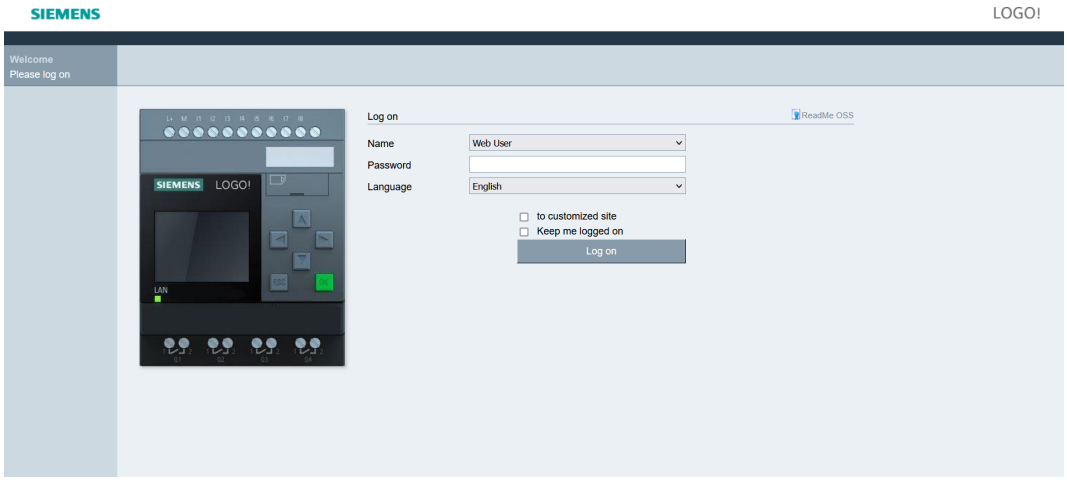


Figure 7: The activated login page

we assume it is a custom web server. But we find other PLCs of the same model on Shodan. Since the query on Shodan searching for devices `LOS HTTP Server 1.0` does give us a lot of devices and devices which are clearly not PLCs we decide to refine our query. We use the search query `ssl.cert.issuer.cn:LOGO Product CA V1.0`. This way we can look at all the devices using the Siemens CA as issuer for their SSL certificate. An excerpt of the results is visible in Figure 8.

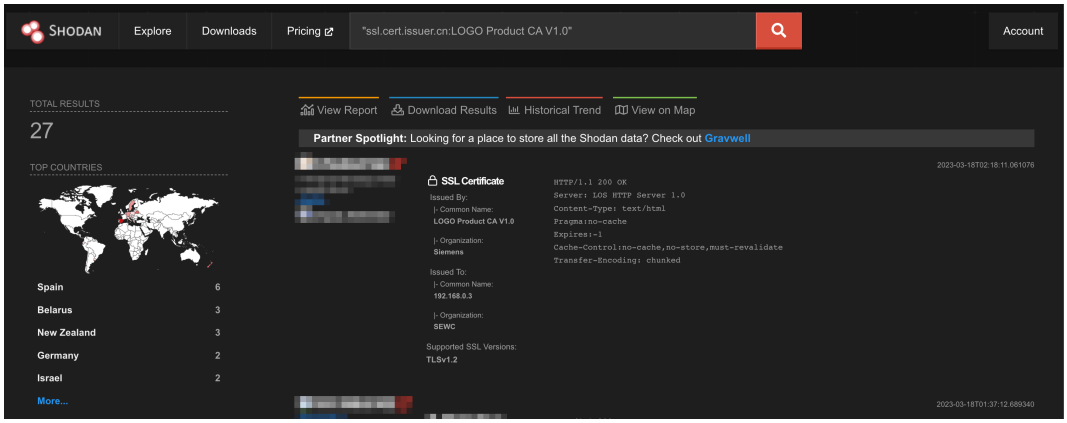


Figure 8: All available devices on the Internet with the same issuer

We find a total of 27 devices. The 27 devices are not necessarily the same model as our examined device. Looking at the given server information of the results we can see `LOS HTTP Server 1.0` appearing. We conclude there are at least 27 potentially vulnerable

PLCs on the Internet at the time being. Due to ethical reasons we do not further investigate the discovered devices and blur their IP addresses.

3.5.1 Authentication

If we login into the web interface we have to authenticate ourselves with a password. There is no minimum requirement for a password, for example the password "1" would be perfectly fine. Nonetheless, there is a maximum of ten characters for a password. According to the Digital Identity Guidelines of the National Institute of Standards and Technology (NIST) a user-generated password shall be at least 8 characters in length and an allowance of at least 64 characters[5]. A maximum of 10 characters and no minimum length requirement is not state-of-the-art. In addition to this, the character restriction implemented into the login section of the web interface is done via JavaScript and hence is run on the client side, meaning the response from the server for this page can be intercepted and altered to remove said character limit. This can prevent the concatenation in point 3 in the list below as the truncated string would only contain the password if the user has changed the character limit to anything above or equal to 32 and has completely filled the password field. While we found no way of taking this discovery further, it is poor design to implement password sanitising and handling on the client side and may present a deeper vulnerability in the future.

For authentication a challenge-response protocol is used which was reversed-engineered[6]. The procedure is as follows, assuming the user's password is '`12345`':

1.: Send a challenge to the server with the cookie `Security-Hint=p`. The body contains `UAMCHAL:3,4,W,X,Y,Z` where W, X, Y, and Z are randomly generated according to the reversed-engineered code. In contradiction to this assumption we always recorded the challenge `UAMCHAL:3,4,1,2,3,4` when testing, as seen in Figure 9.

2.: The server answers with a `statusCode`, a `loginSecurityHint` and a `serverchallenge`. The status code `700` means no error has yet occurred such as in Figure 10.

3.: The client creates a `pwToken` with string concatenation as follows, this is then truncated to a maximum of 32 characters:

```
pwToken = '12345' + '+' + serverChallenge
```

4.: The user creates a `loginPWToken` following this procedure:

```
loginPWToken = crc32(pwToken) XOR serverChallenge
```

5.: Finally they create a `loginServercChallenge`:

```
loginServerChallenge = W XOR X XOR Y XOR Z XOR serverchallenge
```

6.: They send a POST-Request with the following body to the server:

```
UAMLOGIN:Web User,loginPWToken,loginServerChallenge
```

Additionally, the header `Security-Header: loginSecurityHint` is used (Figure 11).

7.: Finally, one last response (as shown in Figure 12) is sent from the server containing:

```
StatusCode, ServerChallengeResponse
```

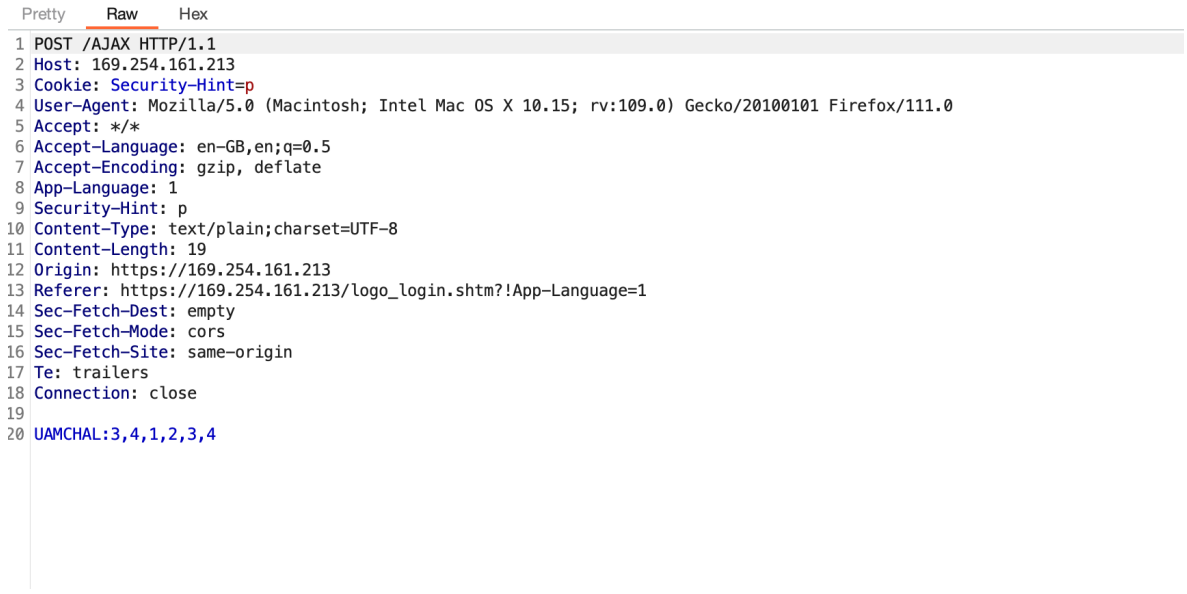


Figure 9: The security challenge

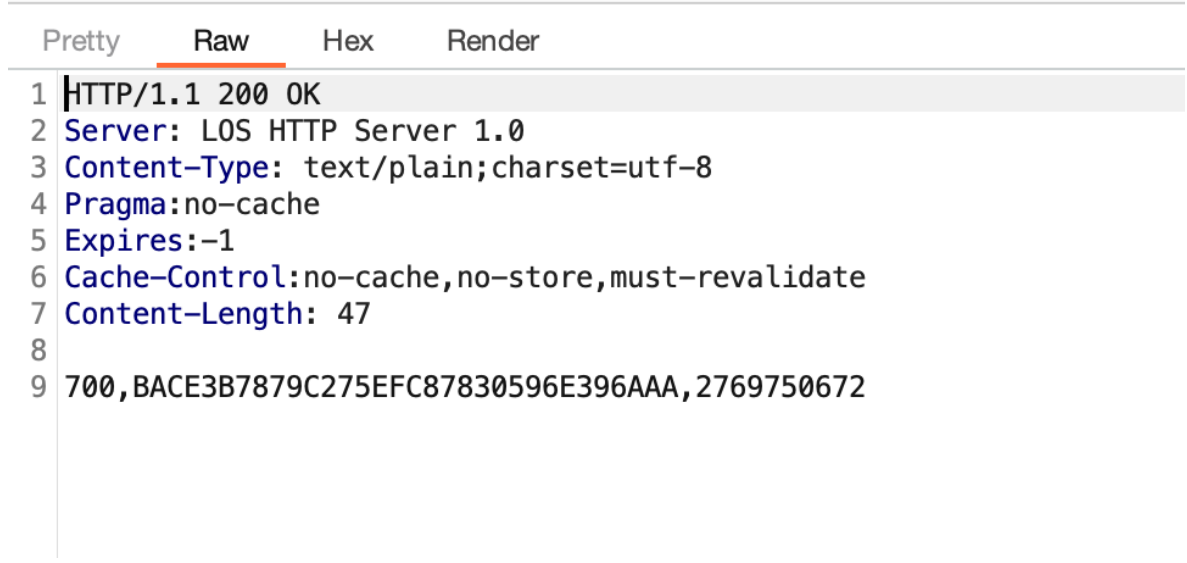
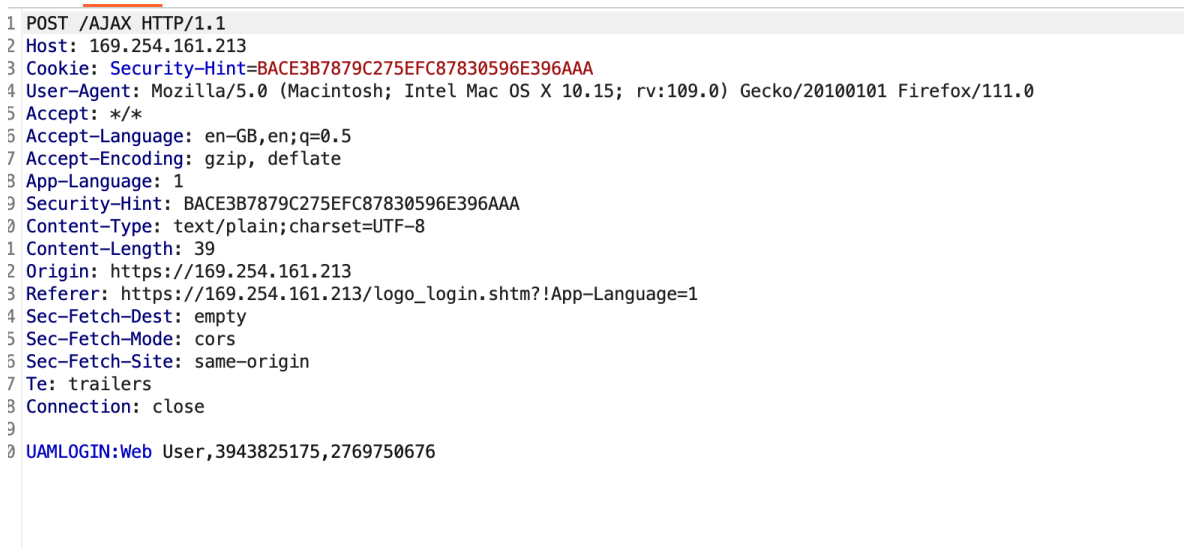


Figure 10: The server response to the challenge containing: StatusCode, LoginSecurityHint and ServerChallenge

The password is never sent in plaintext. However, the security of obtaining the `pwToken` relies on the security of CRC32, the cyclic redundancy check with 32-bit output. CRC32 is not a hash algorithm and is not considered secure. It is quite easy to find collisions for the CRC32 algorithm. Therefore, the CRC32 should not be considered secure, even though it is not exploitable in context of the used challenge-response protocol. Even though we can find different `pwTokens` with the same CRC32 "hash" we are not able to obtain the password

[5] `https://pages.nist.gov/800-63-3/sp800-63b.html`
[6] `https://github.com/jankeymeulen/siemens-logo-rest`



```
POST /AJAX HTTP/1.1
Host: 169.254.161.213
Cookie: Security-Hint=BACE3B7879C275EFC87830596E396AAA
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/111.0
Accept: */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
App-Language: 1
Security-Hint: BACE3B7879C275EFC87830596E396AAA
Content-Type: text/plain;charset=UTF-8
Content-Length: 39
Origin: https://169.254.161.213
Referer: https://169.254.161.213/logo_login.shtm?!App-Language=1
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: close

UAMLOGIN:Web User,3943825175,2769750676
```

Figure 11: The Client Request send corresponding to step 6 in authentication. Contains: Web User, LoginPWToken and LoginServerChallenge

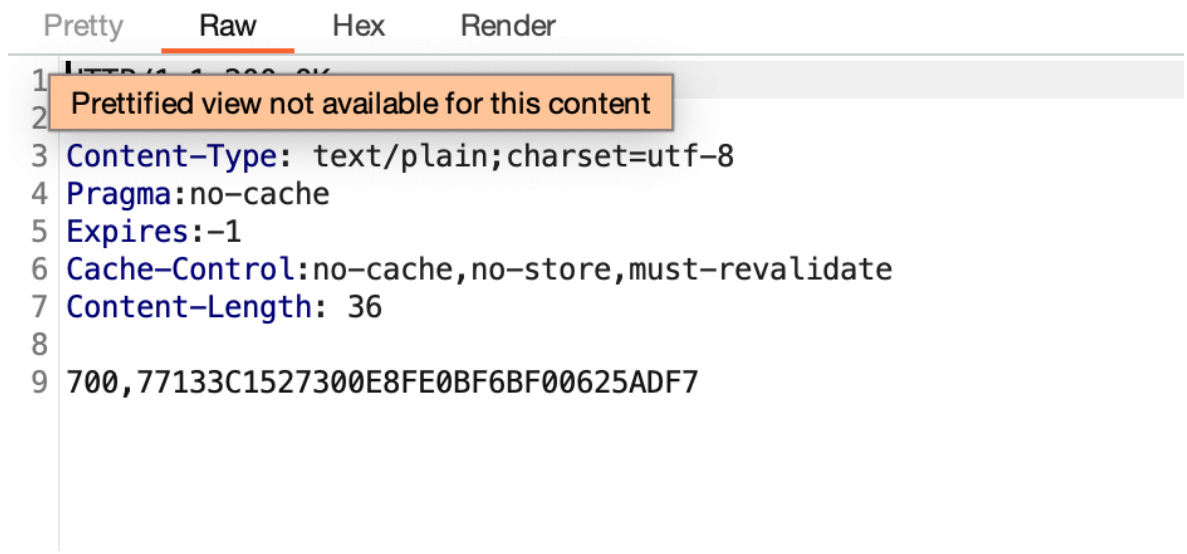


Figure 12: Final response from server before returning the logged in interface. Contains: StatusCode and presumably a serverChallengeResponse

itself. This is due to the fact that the `pwToken` consists not solely of the password and a found collision does not necessarily have the form needed to obtain the password.
After successful authentication the security hint is saved as a cookie and is also passed as a header in all future requests to and from the server. The cookie as saved in a browser is shown in Figure 13.

As visible the `httponly` and the `secure` cookie flags are not set. The `httponly` flag prevents client-side scripts from accessing the data. This mitigates the most common cross-site scripting (XSS) attacks. The `secure` flag prevents a cookie to be sent in plain-text as a

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
Security-Hint	24CB18A95F1FA1354110A268EAC0F6CB	169.254.161.213	/	Session	45	false	false	None	Mon, 20 Mar 2023 12:52:08 GMT

Figure 13: The security hint as cookie

HTTP request. Therefore, the cookie is just sent to a HTTPS site. We want to stress that the shown screenshot is taken when the HTTPS web interface is active. This flag is not just deactivated because HTTPS is deactivated.

3.5.2 SSL certificate

As visible in our port scan A.1 the SSL certificate is by default valid until 01/03/2021 and therefore expired. Once the user activates the web interface the software prompts them to update the certificate. This way the certificate of the PLC is replaced, as visible in A.2.
Anyhow, Siemens prompts the user to install and trust a root certificate, since they sign the certificates of the PLCs by themselves. No matter the certificate it is not trusted by default. Without manually trusting the root certificate the chain of trust is not given and as a consequence the PLC certificate is not trusted as well.

3.5.3 Security header

We use the tool `shcheck`[7] to scan the web-interface for unused or misconfigured security headers. The results are as following:

```
    =======================================================
> shcheck.py - santoru .............................
-------------------------------------------------------
 Simple tool to check security headers on a webserver
=======================================================

[*] Analyzing headers of https://169.254.161.213/
[*] Effective URL: https://169.254.161.213/
[!] Missing security header: X-Frame-Options
[!] Missing security header: X-Content-Type-Options
[!] Missing security header: Strict-Transport-Security
[!] Missing security header: Content-Security-Policy
[!] Missing security header: Referrer-Policy
[!] Missing security header: Permissions-Policy
[!] Missing security header: Cross-Origin-Embedder-Policy
[!] Missing security header: Cross-Origin-Resource-Policy
[!] Missing security header: Cross-Origin-Opener-Policy
-------------------------------------------------------
[!] Headers analyzed for https://169.254.161.213/
[+] There are 0 security headers
[-] There are not 9 security headers
```

[7] `https://github.com/santoru/shcheck`

We look at the missing security headers more in-depth:
The `X-Frame-Options` header decides if a browser is allowed to render pages in `<frame>`, `<iframe>`, `<embed>` or `<object>`. To avoid click-jacking this header shall be set to `DENY` or `SAMEORIGIN`.
`X-Content-Type-Options` avoids MIME type sniffing when set to `nosniff`. MIME type sniffing may lead to JavaScript code being executed.
`Strict-Transport-Security` tells the browser is only allowed to access the site with HTTPS. Since HTTP should no longer be used, this header must be set.
`Content-Security-Policy` can mitigate XSS attacks. Should be set to `default-src 'self'` if all content shall come from the site's own origin.
The `Referrer-Policy` can limit the amount of information shared across different origins in the `referer` header. A privacy-friendly option like `strict-origin-when-cross-origin` should be preferred.
`Permissions-Policy` handles the granted permissions for APIs. For example the site's permission to use the camera or microphone can be restricted. Unnecessary permissions shall be revoked with this header.
`Cross-Origin-Embedder-Policy` prevents loading cross-origin resources from the server that do not permit it. The default value allows access to sensitive information, for instance cookies. For maximum security this header should be set to `require-corp`. This way, all resources loaded from other origins are required to opt into a new context group which isolates them from the rest of the page.
`Cross-Origin-Resource-Policy` controls which origins are allowed to access a resource on a web page. Changing it to `same-origin` only allows resources to be accessed by pages in the same origin.
`Cross-Origin-Opener-Policy` handles the permissions of a page which is opened in a new broswing context. It manages how the original site interacts with the new page. Setting it to `same-origin` will allow the website to be only opened by other sites of the same origin.

A general recommendation for all the security headers is not possible. Some headers need to be chosen individually in context of the specific security requirements of the environment they are deployed in.

3.6 Encryption security

Assuming the user set the web interface to be accessible via HTTPS only, we look at the security of the offered encryption algorithms. We use the script `testssl.sh`[8]. The entire scan is included in the zip file.
The results show that the PLC uses TLS 1.2 and offers no other version. This is fine. For TLS 1.2 it offers the `ECDHE-ECDSA-AES128-GCM-SHA256` cipher suite. This cipher suite is considered secure and is recommended[9]. Additionally, since the root certificate is not trusted by default, a few warnings are enlisted regarding the broken chain of trust. Furthermore, the missing `Strict Transport Security` header is detected, as already mentioned in 3.5.3. No weakness in the encryption was found testing known vulnerabilities like Heartbleed or POODLE. Worth mentioning is the fact that the PLC offers 384 bit ECDH for every simulated

[8] `https://github.com/drwetter/testssl.sh`
[9] `https://ciphersuite.info/cs/TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256/`

browser except Android 7.0 (native). For this browser the 256 bit variant is offered. Since it is still secure this is not a weakness or a flaw.

3.6.1 Change variable section within web interface

There is a section on the web interface which allows an **authenticated** user to send 'variable' changes to the PLC. For example the user can decide whether or not the conveyor belt is running by changing the Boolean variable to 0 or 1. This can similarly be done for any output of the PLC, or within our system specifically, the items labelled 4, 6 and 7 and the conveyor belt itself as shown in Figure 3. Figure 4 shows the wired outputs from the PLC. These outputs labelled Q(_) contain the physical connection necessary to run our **program**. The variables mentioned can be hardwired within the LOGO! Software on the directly connected machine, in this situation this appears to take priority over any attempts to change variables on the web interface. However, this technique offers little flexibility for change in the system, or for co-operative work on a system. Hence, it is possible to set up the system in such a way that the program essentially waits for input from network calls, primarily, via the web interface. We believe this is system is likely to be the more implemented technique in many real world systems as convenience and flexibility at the expense of security is often favoured over an inconvenient system that requires a greater effort when things go wrong or need changing. Also, in a realistic setting, it seems more likely that a manufacturer or company operating machinery will not all be based within a local proximity, the enabling of a web interface would be imperative in allowing them to continue operation of machinery in various locations at once, cutting out substantial travelling time and costs. As a result, we believed the web interfaces ability to change variables live within a running system to be a very promising attack surface.

Figure 14 gives an example of how a variable can be changed in the web interface as described.

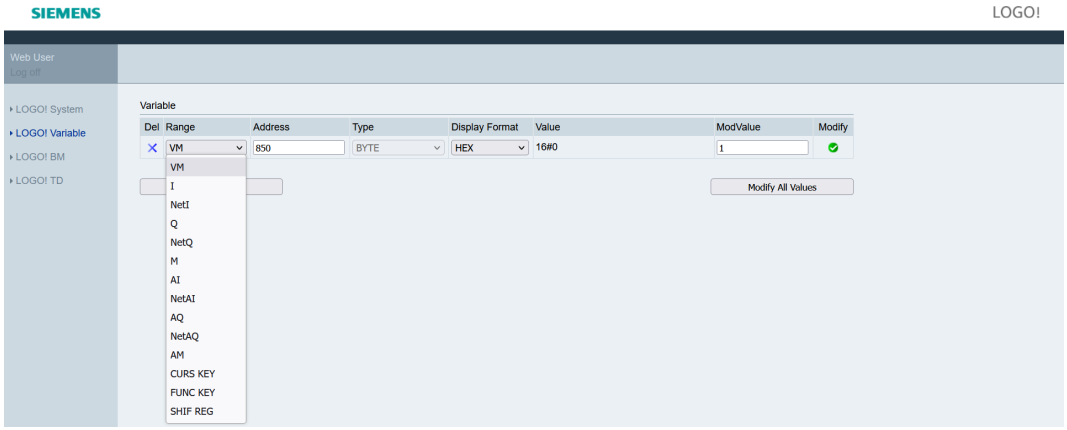


Figure 14: The web interface and changing variables

Just to summarise and give a bit more context on the list of variables in Figure 14 because at first glance these do not give much information (Information from LOGO!Soft Comfort guide:

1. VM: Memory available by remote devices to read or write to.

2. I: Used to address the inputs of the PLC.

3. NetI: You can connect a network input to a block input. You can configure up to 64 network inputs. Network inputs can read values from the following types: VM, remote device.

4. Q: Used to address the outputs of the PLC.

5. NetQ: When the LOGO! Base Module is in slave mode, you can configure a network output on the master to control a digital output on the remote device. You can configure up to 64 network outputs.

6. M: Flag blocks output their input signal.

7. AI: LOGO! PLC can process analog signals. You can use up to eight analog inputs. In your block configuration, you can assign a new input terminal to an input block, provided this terminal is not already used in the circuit program.

8. NetAI: You can connect a network analog input to a block input. You can configure up to 32 network analog inputs. Network analog inputs can read values from the following types: VM, remote device.

9. AQ: Eight analog outputs are available, namely AQ1, AQ2, ... AQ8. You can only set an analog value at the analog output, that is, a function with an analog output or analog flag AM.

10. NetAQ: When the LOGO! Base Module is in slave mode, you can configure a network analog output on the master to control an analog output on the remote device. You can configure up to 64 network outputs.

11. M: Flag blocks output their input signal.

12. AM: The analog flags can be used as markers for analog inputs or analog instruction blocks. The analog flag merely accepts an analog value as input and outputs that value.

13. CURS KEY: You can program cursor keys for the circuit program in the same ways as other inputs. Cursor keys can save switches and inputs, and allow operator control of the circuit program.

14. FUNC KEY: TDE module has four function keys that you can use as digital inputs in your circuit program. You program the function keys in the same way as other inputs in your circuit program. Function keys can save switches and inputs, and allow operator control of the circuit program.

15. SHIFT REG: The LOGO! devices provide eight shift register bits S1 to S8, which are assigned the read-only attribute in the circuit program. The content of shift register bits can only be modified by means of the shift register special function

A video of us exploiting this can be seen here. This video simply shows what happens when you send the variable change to the PLC for the green light. Followed by and interception and change to the instruction which keeps the light in its current state.

Figure 15: Gives an example of what an intercepted call to change variables looks like. On the left is an intercepted call to change a Q output, on the right is a call to change a VM memory value.

Within our system, we relied more on changing the direct output variables (Q) with the web interface. However, in more sophisticated and realistic systems, any interaction with the web interface is more likely to be done via writing to device memory via the VM variables. The program running on the PLC will interact with its memory and certain inputs will rely on memory stored on the device. Any change to the circuit will be implemented by changing the value stored at this address in memory via the web interface. **??** shows two intercepted calls in burp suite that are made when changing variables, first when changing directly the output variable Q7, and setting it to value of 1, as shown by '01' at the end of the string. The second is a call to change a memory address value VM 83 to contain a byte value of 'FF'. Under http communication it is clear how variables are being changed, the request is simply being sent as plaintext to the AJAX server, the 'SETVARS' is presumable filtered to carry out the requested command internally on the server side. This seemed like a promising attack surface. The first port of call is to establish what authentication is taking place on the AJAX server if any. We first intercepted a message via Burp suite and attempted to remove the Security-Hint to see establish whether there is any authority check at all. However this returns a 401 error with the message 'session expired', inferring that the Security-Hint passed to the server was either expired or not authorised. So, as a minimum, the sent request needs to be coupled with a valid `Security-Hint` that has been generated by inputting the correct password during the login process. We developed a shell script that sends curl request, seen in Figure 16, if inputted a valid security-hint, the script will successfully turn on all of the outputs within our system. This proved that someone could directly interact with the entire system, without needing to ever go through the log-in process, all they would need is a valid security-hint, something we explored attacking and elaborated on in a future header.

A video of our attack can be seen here (0:18 to 0:26). Along with a video of the script running at the same time,here.

To elaborate a little also on Figure 16, this script simply filters from Q1-10 and either turns them on or off via boolean 00 or 01 calls. Our headers within the curl request itself simply contain: the security-hint, the change variable command and its accompanying data type, a self-signed certificate the curl required to permit it to make https communications, with `--insecure` telling curl to use this self-signed certificate in any TLS communication attempts. This curl command is received and successfully processed, even when the address of the server is adjusted to incorporate https and the web server setting is changed to https

```
#!/bin/bash

securityHint=$1;
modeInput=$2;
mode=1;

echo $modeInput
echo "off"

if [ "$modeInput" = "off" ];
then
    echo "Turning off.."
    mode=0;
fi

echo "Security Hint entered: $securityHint";

for i in {1..10}
do
    vars="v0,Q..1:$i-1,0$mode"
    echo $vars
    curl -v --header "Security-Hint: $securityHint" --header "Content-Type: text/plain;charset=UTF-8"
        -d "SETVARS:_local_=$vars" --cacert "/Users/¢¢¢¢/####/cacert.pem" --insecure "http://169.254.161.213/AJAX"
done

```

Figure 16: Final shell script developed that successfully turns on all outputs in our system without needing to interact with the web interface.

within the PLC LOGO! software itself. The fact that this curl request still works on https means that appropriate certificate validation is not taking place on the AJAX server. This is reinforced by the fact that https web server communications being enabled on the LOGO! software itself is only met with a warning to update your own certificate (as the software default certificates our outdated and expired), not an invalidation of permission to activate. An implementation on the server side of appropriate certificate validation would significantly help to improve security and authorisation of important requests to the system. Overall this attack, if the correct conditions are met (these are more clearly defined **here**) can be devastating to a system. If an attacker is able to obtain a `Security-Hint` and has the ability to spoof to the original authorised user, the attacker has the potential to take down and ruin the entire system, the only solution would be a whole system reset, this is an unacceptable consequence and can be potentially extremely costly to a company implementing this PLC

3.6.2 Credit to Interface Security

Whilst we have been able to show that changing variables is vulnerable and that the security-hint that is crucial to all variable changes with the PLC can be man-in-the-middled (the details of an attack are elaborated on in **this header**). It is still worth noting what can reliably be deemed secure and good practice by Siemens in this regard. First, the session sniffing itself, whilst this possible via http, trying to carry this out on an external device whilst the web server is running in https mode means that all traffic is encrypted and the security-hint itself is not available to capture at any point.

In addition to this, the ability to alter variables in the web interface is not something that is enabled by default on the PLC, in face the web interface needs to be manually enabled. Whilst this interface is disabled, any attempt to communicate with AJAX web server cannot be processed and hence there is no way to change any variable values or even view any variables via this attack surface. It is also worth noting that the AJAX web server itself is implemented strongly. Whilst the calls to the server are made via Javascript functions that are stored on the client-side, there are no hard-coded vulnerabilities or exploitable bits

of code that could be found to use against the server. It seems strong in its filtering and sanitising of requests, only responding to valid requests that cannot be hijacked in order to carry out anything malicious. This can be further enforced by unsuccessful attempts to use Telnet to connect to the PLC directly. Whilst successfully connecting to the open port 80 running http, the only requests successfully received this way by the server were those that exactly copied those sent by the web interface itself, hence not supporting any new attack surface. In fact, instead suggesting proper security and programming principles on the code of the AJAX server being ran internally. No shell could be ran via Telnet, internal memory accessed, files viewed. The PLC appeared secure to any attempts to access internal workings other than the AJAX server that is hosted on the device itself.

3.6.3 Further Investigation Plan

Within our internal research, due to external restrictions out of our control, we had to treat any attacks of the system internally over a LAN wired connection. Hence there being a higher emphasis on the attacks related to the web interface. However, to take this further, it is possible via the PLC guides and handbooks to set up a WiFi connection to the device, or even interact with a cloud to receive instructions. Both of these would be interesting areas to look into and potentially also devastating attack surfaces. It is possible that the web interface focused on within our network remains the same method of interaction even when WiFi is enabled. In this situation our discovered attack can potentially become more disastrous and the bounds of location no longer apply. In fact, the attack surface would only expand, as victims now become susceptible to powerful phishing attacks that could prompt the user to a fake login for the plc. If the attacker concocts some way to either gain the users password, or an active Security-Hint, they will be able to launch this attack, no matter whether http or https is enabled and appropriate certificates configured.

Enabling the cloud server could also be vulnerable, however, depending on the cloud provider, this may be more difficult to test, as it is not within legal bounds to attempt to attack an external provider. However, if we are able to host our own servers to interact with the PLC's, this could be a potential attack surface hosting more vulnerabilities than were initially found. This would be an interesting area of research if this project were to be taken further. A third interesting attack surface would be the communication possibility between PLC's themselves, as mentioned previous, it is possible to set master and slave modes to give multiple PLC's a hierarchy with one acting as the primary commander. Siemens themselves within the LOGO! software application attempt to persuade you against activating the certain ports required for this sort of communication due to it being 'vulnerable'. The reasons for this being deemed so would be interesting to explore if we had access to additional PLC's in the future.

3.7 LOGO! Soft Comfort

This application is one of the main ways of interacting with the PLC and configuring it, therefore we investigated it for any possible vulnerabilities an attacker might exploit. In order to analyse the software we use Ghidra and check how the executable is structured.

We first take a look at the imports of the executable and we can see three different DLL

files being imported: gdi32.dll, kernel32.dll and user32.dll. While those three DLL files all had CVEs associated with them as some point, the only recent one is CVE-2017-0038 which has been patched already by Microsoft. As long as the Windows 10 installation is up-to-date the security of the DLL files is not a concern. When looking at the functions of the application the first thing we can notice is that the executable is stripped, meaning that all of the unnecessary debug information that was part of the program has been removed. This not only makes the executable itself smaller but it acts as a layer of security by obfuscating the code from disassemblers, like Ghidra, making it more difficult to reverse engineer the application.

In addition, we check the whole codebase for any hard-coded strings that could leak confidential information, like passwords or log messages. While there are a few integer values that are hardcoded, they do not leak any critical information. When it comes to strings the only ones we could find would refer to the name of the application window, or the name of different images that would be imported into the GUI. This is true about log messages as well, the only ones appearing in the executable being errors related to the initialisation of the GUI itself and nothing about the actual protocols, or passwords that are in use.

Because of this our only other option was the analyse the whole binary file. Looking through it gave us more insight on how the application works, like what it uses for the GUI and how it sets up the environment for use, but the inner workings on how it connects to the PLC are heavily obfuscated from the disassembler. While not being able to go in detail through the whole codebase. From what we have analysed so far there are no obvious vulnerabilities in the LOGO! Soft Comfort application.

4 Possible attacks against the device

4.1 Attacks by a local attacker with network access

The following attacks require the attacker to have access to the same network as the PLC.

4.1.1 Session token sniffing

We knew as previously stated that it is possible to launch an attack that manipulates the variable values related to our PLC system from the web interface, this could be done by intercepting any variable requests and manipulating them. We then found that this could be taken a step further and instead of waiting for an authorised user of the system to attempt to change the variables for us to intercept, we can instead intercept their `Security-Hint` (which when the web server is communicating over http is sent every second within the header of idle refresh requests to the AJAX server that the web page seems to make) and use this to our own advantage, which is what was done to create the functioning curl script. Whilst this worked on a local PC, we wanted to simulate this attack as it would happen. To do this, first we launched an ARP poisoning attack between the user who had the intent of logging in, and then PLC itself. This was done via a tool called 'Bettercap'. After poisoning the user and the PLC the attacker can user Wireshark, or Bettercap itself, to intercept all traffic between the two targets. Depending upon whether the web server is being run in HTTP mode (which is done by default) or HTTPS mode, this traffic is either not encrypted or encrypted. In the event it is not encrypted, the attacker is able to easily obtain the `Security-Hint` as it is passed via the header of the response, as shown in Figure 17.

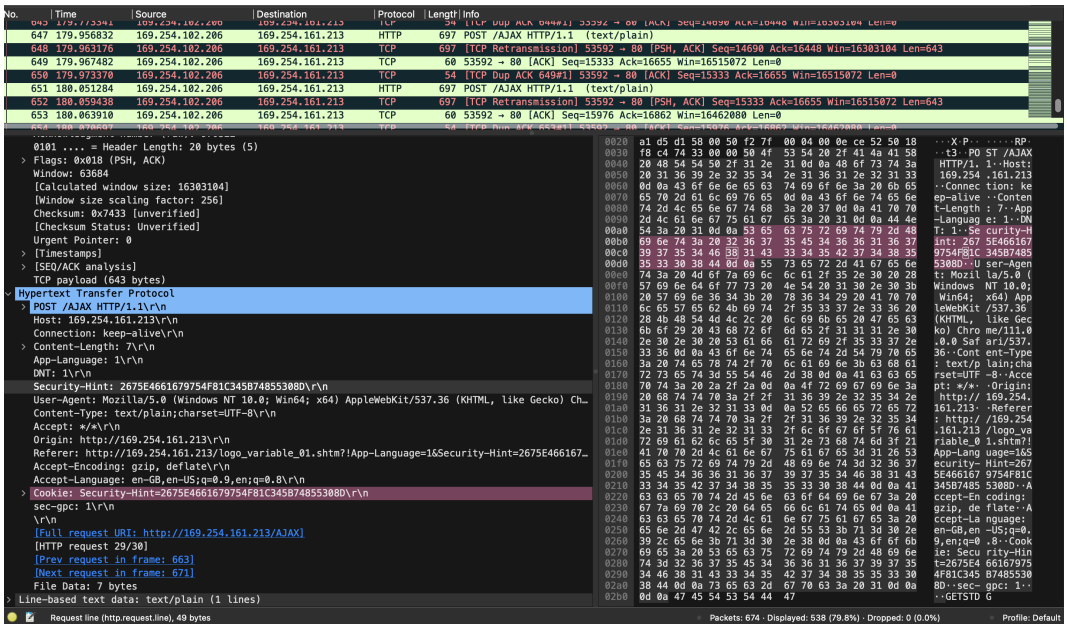


Figure 17: Wireshark interface sniffing one of the packets intercepted between the user and the PLC AJAX web server, can see the crucial Security-Hint variable vulnerable.

This session token is powerful because it appears to be the one of the only verification techniques on the server side of an authorised session. The other is IP address. From what

we can surmise it seems as though the AJAX server will store a list of dictionaries, one for each IP using the web interface, within these dictionaries there are the various session tokens, or `Security-Hint`s that represent valid logged in sessions per user, where a user in this case is identified by their IP address. These session tokens remain valid until either the user: logs out or remains idle long for a certain prolonged period of time (if we had a valid token and did not log out, but returned the next day this was no longer valid). Neither of these two techniques of ensuring user verification are sufficient for a multitude of reasons listed in the following headers.

4.1.2 Insufficient Session Token Cleansing

First focusing on the issues with the conditions that keep a `Security-Hint` token valid. As mentioned, they lose authorisation following a logout call, or by showing a lack of activity for a certain time threshold. As we have shown to obtain the security hint in the first place, it is possible to do an ARP poisoning attack on the ARP cache of both the PLC and the user, within this tool on Bettercap, it is also possible to re-route intercepted traffic via a proxy, this means that an attacker would be able to intercept any logout request and route this to their own proxy. From here, the attacker may choose to simply drop the packet request, although this may seem suspicious since their web-page won't exhibit any indication of a successful logout. A smarter technique would be to drop the packet and return a modified response that routes the user to a spoofed logout page, or back to the legitimate login page without forwarding the logout request. The logout request is easily identifiable as show in Figure 18. It simply contains a call to the 'function' `UAMLOGOUT` which is filtered by the AJAX server (this is recognisable in the client stored JavaScript on the site) and passes in a currently active session token. Presumably the server receives this request and drops the passed in token from the list of authorised '`Security-Hint`'s.

Figure 18: Example logout request. Containing call to function 'UAMLOGOUT' and an example Security-Hint

The second method of cleansing the token by removing its authorisation after a period of inactivity is very easily avoidable by setting up a script that will simply ping the web server with the `Security-Hint` as a parameter and this will keep the session token alive in the AJAX servers eyes. This can be combined with the previous attack of rerouting the logout request and theoretically an attacker has the capability of only needed a single `Security-Hint` token for all subsequent attacks as it will now remain on the server indefinitely in an 'authorised state'. The original user will have no way of purposefully or accidentally removing this token

in the future as any new log-ins will generate a new session token that can coexist with the old one that the attacker has hijacked. Not only is this insecure, it also opens up the potential for overflow attacks. If the attacker intercepts all future logout requests and repeats the process of redirecting the user to the log in page, and then subsequently starting a daemon that frequently pings the web server using this new session token. It allows the attacker to ensure that no valid `Security-Hint` session token is ever dropped, the **??** will only have a limited amount of memory, especially the memory dedicated to this specific function. It may be possible to either: overwrite other memory of the PLC if a sufficient amount of tokens are maintained active or one time, reach an internal limit on the PLC of active session tokens so that any further attempt to log in by the legitimate user will be met with an error, effectively locking them out of web interface access.

4.1.3 The Issue of IP

This second layer of protection is a bit more of an advanced issue but using the similar techniques to those already demonstrated it is again, easy to bypass. Attempting to steal a `Security-Hint` is easy over a HTTP connection. The issue for an attacker when this hint has been obtained is that, when attempting to execute a remote variable change command either individually or via our shell script the server will simply respond with an 'ERROR 401 - Session Expired'. Contrary to the error statement, the session token remains active but is just not usable from any external IPs. It seems the server will pattern match using the IP address of the request sender and the saved IP address it has for the user who generated the token in the first place via a successful login. If they match, execute the request, if not, return a session expired 401 error. We found within our experiments when using a Linux machine, it is possible to simply steal an IP address by manually changing to that of the original authorised user whom the `Security-Hint` initially belongs to, it appears the AJAX server does not check by MAC-address in addition to IP, because doing this simple change allowed us to execute a stolen security hint as if it was our own. This may also cause actual connectivity issues for the original user in an internal LAN network too, so could actually be a form of DOS attack on its own as well as being a part of the larger variable changing vulnerability.

Even if the server did check MAC-Address, this check would be influenced by the aforementioned ARP poisoning and this attack would still be possible. Hence, in our local LAN network, these measures are extremely inefficient in protecting the web server as it currently stands.

4.1.4 The Attack Itself

To summarise this attack using all the previous information: An attacker with access to a LAN network will be able to, from scratch, sniff the traffic between the user and the PLC if the web server is activated. If traffic is not encrypted as the users configured the device with defaults, the attacker will be able to see every time a user logs in, or is active on the web server. Along with this, they will also be able to access their `Security-Hint` for every single session that this user opens. Along with sniffing the network, they are able to proxy it to by using intelligent tools such as Bettercap and Burp suite. This will permit the attacker the ability to totally control and restrict what instructions an internal user is trying to send to the PLC. The attacker can drop requests at will and manipulate responses at will, this allows

them to void any logout requests and keep a session token valid for as long as they desire through simple scripts and daemons that run in the background. The attacker can not only sniff the network, but ARP poison it in order to convince each device that it is the correct one that it should be communicating with. With this, the attacker can execute variable changes on the variables mentioned in section 3.6.1, this, as also shown, can change output signals, virtual device memory, and overall cause catastrophic failure to the entire system that the PLC is controlling.

4.2 Attacks by a remote attacker

4.2.1 Denial-of-service attack

Since the PLC provides a web-interface and an API we can make HTTP-requests. We try two Denial-of-service (DoS)-tools, Slowloris[10] and Goldeneye[11]. Goldeneye does not work as expected and no attack is possible. Slowloris does slow down the loading time of the web-interface for another user but the effect is too small. Therefore, we decide to write an own bash script:

```
!/bin/bash

seq 1 200000 | xargs -n1 -P300
curl "https://169.254.161.213/logo_login.shtm?!App-Language=1" -k
-H "Connection: keep-alive"
```

The script spawns 300 processes and accesses the web-interface up to 200000 in parallel. The `-k` option disables a SSL check, since the certificate of the PLC is not trusted and we want to ignore all SSL errors. `-H` sets a HTTP header. We use the header `keep-alive` to tell the web-server not to close a connection after the request has been made. Therefore, we establish connections and keep them alive.
Starting the script takes down the web-interfaces after a few seconds. During script execution the web-server resets most of the incoming requests, including the requests made by a real browser as shown in Figure 19.

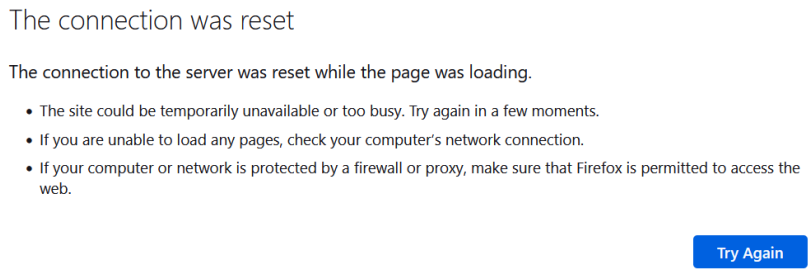


Figure 19: Unreachable webinterface during our DoS attack

Additionally, the AJAX API is not reliable anymore and we are not able to send commands. For example the session sniffing described in 4.1.1 is not possible during the DoS

[10] `https://github.com/gkbrk/slowloris`
[11] `https://www.kali.org/tools/goldeneye/`

attack.
The DoS attack does not effect running programs. The PLC is able to execute the program as usual. This does not surprise us since the attack only targets the web-server and the underlying service of the web-server. The other services are not effected.
As we can not connect the PLC to the Internet, we conduct the DoS attack via the Intranet. Therefore, we only assume this attack is executable by a remote attacker. This attack should be possible as long as the port for the web-interface is accessible through the Internet and no further measurements hinder the attack, for example firewalls or bandwidth of the Internet service provider (ISP).

4.2.2 Default credentials attack

All used passwords, for the web-interface mentioned in 3.5 or the overwrite access, are by default `LOGO`. The software does not prompt the user to change the password immediately. Therefore, only activating the web-interfaces does start the web-interface with the default password. The fact that there is not username to choose from decreases the difficulty for a default credentials attack. An attacker could possibly try the default password on openly accessible web-interfaces.
To mitigate this attack every password must be chosen randomly with the maximum allowed characters. Additionally, every service provided by the PLC must use a unique password. We suggest Siemens to force the user to change the password on setup and not allow them to use default passwords.

4.2.3 Local attacks "remotely"

Assuming the PLC is hardened and it is not possible to access it through the Internet it is possible to deploy malware to obtain access to the PLC. If an attacker gains access to any machine in the network of the PLC they can ARP-spoof the network to Man-in-the-Middle (MitM) communication to the PLC. Since the session token is valid unless a user logs out, the interception of the same in combination of IP spoofing makes it possible to send requests to the PLC.

5 Analysis of the weaknesses found

For the evaluation of the severity of the found vulnerabilities, we are using the Common Vulnerability Scoring System (CVSS) version 3.1.. We use the three main goals for security to categorise the possible consequences of a successful attack. The three security goals are Confidentiality, Integrity, and Availability (CIA goals).
An overview of the attacks is shown in Table 1.

Attack	CVSS	Details	Consequences	Mitigations for users
Session token sniffing	6.3 (medium)	4.1.1	Confidentiality, Integrity	Add a Message Authentication Code or a Digital Signature to ensure the session tokens' integrity
DoS	5.3 (medium)	4.2.1	Availability	Do not make the PLC available on the Internet. Use firewalls and DoS protection
Default credentials	9.1 (critical)	4.2.2	Confidentiality, Integrity	Change the default password during setup and pick it randomly with the max length of 10 chars.

Table 1: Overview of the found weaknesses

5.1 Session token sniffing

Requirements: The session token sniffing requires a lot of effort by the attacker. They need to gain access to the local network the PLC is located in, the web interface must be activated, and the attacker has to conduct an ARP poisoning attack to intercept the traffic of a user, and the PLC. Or situated at the switch/router of the network to monitor non-broadcasted communications. Therefore, they need to conduct the attack while an ongoing web session or a new session is established by another user.
Consequences: Assuming the requirements are met, the attacker gains a lot of rights. They can change variables influencing the behavior of the PLC. This could lead to machinery malfunctioning as long as they are controlled by the PLC. If an attacker gains a valid session token they are able to send arbitrary messages imitating the identity of the attacked user. As a consequence, the integrity of every message of this user is not given anymore. Furthermore, values of set variables of the PLC can be read. The confidentiality gets broken as well.
Mitigation: Activate HTTPS.
Fix: The fundamental flaw which enables this attack is the missing integrity check. To ensure integrity a Message Authentication Code or a Digital Signature must be implemented for every message sent by the authenticated user. Additionally, the certificate should be signed by a trusted source by default without a manual installation. Remove HTTP and force the user and the PLC to use HTTPS.

5.2 Denial-of-service attack

Requirements: Port 80, 443 or 8443 is reachable by the attacker. If the web interface is available on the Internet the attacker does not need local network access, otherwise, access to the local network is necessary.
Consequences: The consequences vary from the deployment scenario of the PLC. If only the readme page is active, the availability of the readme page is impeded but the control functionality of the PLC itself is not. If the web interface is active, the attacker can disturb using the web interface or the AJAX API. This can have a stronger impact on the availability assuming a company relies on controlling the PLC remotely.
Mitigation: Filter any requests on port 80, 443, and 8443 to the PLC with a firewall. Use a VPN to connect to the Intranet and then to the PLC instead of making the PLC reachable through the Internet.
Fix: Use DoS protection to secure the entire network.

5.3 Default credentials attack

Requirements: The attacker has access to any password-protected service of the PLC and the service is active. The user has not yet changed the default password.
Consequences: The attacker can authenticate as a user and gains the same rights as a legitimate one. The confidentiality and the integrity of this service is therefore endangered.
Mitigation: Pick randomly generated unique passwords for each service. Use the maximal length of 10 chars. Change the passwords immediately after setting up the PLC.
Fix: Force the users to change passwords immediately. Remove default passwords. Introduce a minimum length requirement of 10 chars. Remove the maximum limitation or increase it to at least 64 chars.

6 Working as a team

From beginning to end, whether that be: researching, penetration testing or writing the report; our team seemed to demonstrate a high level of competence when working together. From the initial handout of the IoT device our team eagerly created communication channels. We set up a quick means of communication via an instant messaging app to be able to promptly schedule meetings and discuss anything time sensitive related to our project.

Additionally we set up a Discord server, allowing us to manage and collate our project as a team. We had 4 main channels allowing us to write quick notes; Collate any helpful research found that related to the PLC; Paste any information found relevant to the report and a section for detailing what we had left in our list of things we wanted to achieve during the time to produce the report. This can be seen in Figure 20.

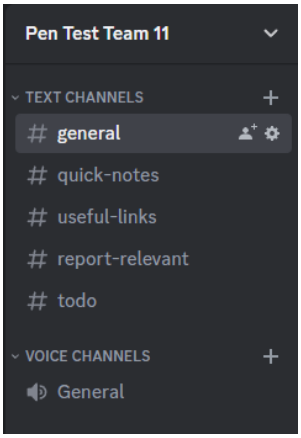


Figure 20: The Basic channel structure used in our Discord server.

6.1 Meeting to work on the device.

Given the nature of our device, We were limited to working on campus and required to make use of the security laboratory. This took away the convenience factor of being able to pass around the device and work on it at home, however we believe it initially forced our hand to schedule a team meet up which made future meetups far more frequent and productive. We initially set meetups for the free time we all shared each week according to our timetables. We then used an instant messaging app to schedule any additional meetups day to day that we might have wanted, and to notify another team member if we decided on a whim to go work on the project and ask if another team member would like to join.

This meant that we would often work on similar areas of the PLC at the same time (researching, pen-testing). However when needed would would play to each others strengths and split the tasks accordingly. A breakdown of the work can be seen in the next subsection. t5

6.2 Teamwork breakdown

Based on the sections previous, We all agree that it's difficult to quantify exact percentages of which team members worked on what areas. We often found that we'd bounce ideas from one

and another and have multiple members working on the exact same thing for a particularly difficult area of the project, this usually resulted in 1 person finding a flaw or strength of the system although the other team member spent around the same amount of time on the same area. Because of situations like this, none of us can comfortably say that they worked on a particular area more than another team member (given some exceptions).

Around half way through the project we had another team member join causing some disruption to the team-working balance we had flowing. We all understand and accept the circumstances which resulted in this disruption and welcomed them with open arms. At this point a lot of the hands on work was done, however we still managed to include them in areas of the project we hadn't quite covered as much or even considered.

6.2.1 Exceptions and Actual breakdown

Due to the for-mentioned inability to comfortably assign exact percentages, we will instead present a table with more general quantifiers to express who worked on what. To make the quantifiers more meaningful, Refer to 6.2.1 below.

.	Report	Research	Device Setup + Programming	Attack and related	
Daniel Benton	two	two	three	two	
Owain Edwards	two	two	three	two	
Samuel Calvesbert	two	two	two	one	
Tobias Neugebaue	one	two	one	two	
Alexandru-Catalin Radu	two	three	four	four	

Table 2: Very Rough Overview of the team working

Although These may not be a completely accurate way of describing our team work, It should generally give an idea of how well we worked together, We would describe our team-working as no less than great; even with the obstacles and disruptions faced along the way.

To further supplement our team-working, we later found trello to be a useful tool to help manage workload when our ideas and research furthered the complexity and length of time needed for the project. We started using this around half way through the time allotted. Figure 21 shows a quick screen-grab of our working area towards the end of our project.

To add more supplimentary detail to the Attack and related section shown in HERE. This includes any attacks one can think of which were attempted on the device. Some examples would be; SQL injection attempts; DoS attacks and MITM. To make it clear; we all didn't work on each attack necessarily, but spread tasks amongst us fairly equally, unless an attack benefited from having multiple team members work on them.

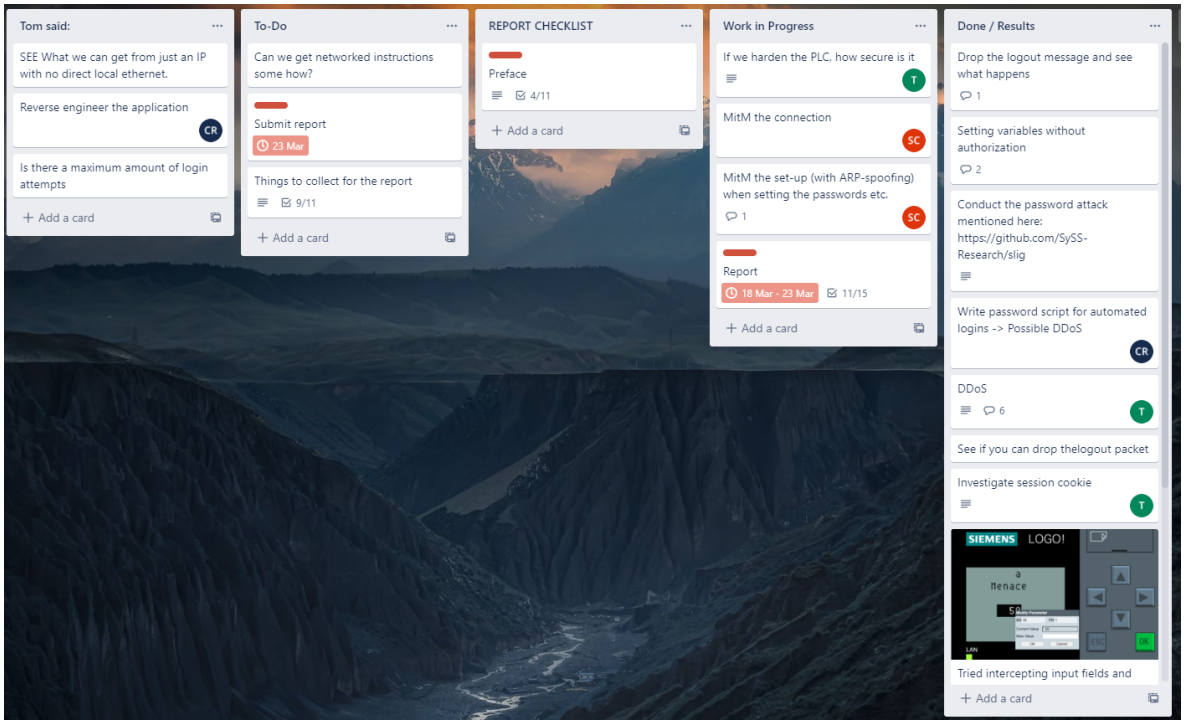


Figure 21: Trello Screen grab

Table Quantifier Descriptions For Table 2

one Large amount of time comparatively to other teammates

two Roughly an equal amount of time comparatively to other team members

three Other team members put a larger amount of time in. But some time was spent

four No time was was spent on this area.

A Scans

A.1 Nmap scan with factory settings applied

```
   # Nmap 7.93 scan initiated Fri Mar 17 07:51:29 2023 as: nmap -d -v -A -p-
-oA nmapfactorysettings 169.254.161.213
--------------- Timing report ---------------
  hostgroups: min 1, max 100000
  rtt-timeouts: init 1000, min 100, max 10000
  max-scan-delay: TCP 1000, UDP 1000, SCTP 1000
  parallelism: min 0, max 0
  max-retries: 10, host-timeout: 0
  min-rate: 0, max-rate: 0
---------------------------------------------
doAnyOutstandingRetransmits took 56ms
Got nsock CONNECT response with status TIMEOUT - aborting this service
Got nsock CONNECT response with status TIMEOUT - aborting this service
Nmap scan report for 169.254.161.213
Host is up, received syn-ack (0.0025s latency).
Scanned at 2023-03-17 07:51:30 EDT for 255s
Not shown: 65532 filtered tcp ports (no-response)
PORT     STATE SERVICE    REASON  VERSION
80/tcp   open  http?      syn-ack
|_http-title: Site doesn't have a title (text/html).
| http-methods:
|_  Supported Methods: GET POST
135/tcp  open  msrpc?     syn-ack
8443/tcp open  https-alt? syn-ack
| http-methods:
|_  Supported Methods: GET
| ssl-date:
|_  ERROR: Unable to obtain data from the target
| http-cisco-anyconnect:
|_  ERROR: Failed to connect to SSL VPN server
| ssl-cert: Subject: commonName=169.254.161.213/organizationName=SEWC/
countryName=CN
| Subject Alternative Name: IP Address:169.254.161.213
| Issuer: commonName=LOGO Product CA V1.0/organizationName=Siemens/
countryName=DE/organizationalUnitName=Copyright (C) Siemens AG 2020
All Rights Reserved
| Public Key type: ec
| Public Key bits: 256
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2020-03-01T00:00:00
| Not valid after:  2021-03-01T00:00:00
| MD5:   d68275a296d983a55c491ec64e628b33
| SHA-1: 8f1a1f2dda295f8b29c7d68a729536c769f0be6e
```

```
| -----BEGIN CERTIFICATE-----
| MIICpzCCAY+gAwIBAgIHA++jAAAAHTANBgkqhkiG9w0BAQsFADB6MR0wGwYDVQQD
| DBRMT0dPIFByb2R1Y3QgQ0EgVjEuMDE6MDgGA1UECwwxQ29weXJpZ2h0IChDKSBT
| aWVtZW5zIEFHIDIwMjAgQWxsIFJpZ2h0cyBSZXNlcnZlZDEQMA4GA1UECgwHU2ll
| bWVuczELMAkGA1UEBhMCREUwHhcNMjAwMzAxMDAwMDAwWhcNMjEwMzAxMDAwMDAw
| WjA2MRgwFgYDVQQDDA8xNjkuMjU0LjE2MS4yMTMxDTALBgNVBAoMBFNFV0MxCzAJ
| BgNVBAYTAkNOMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE3aLIhIzVsIccAo5Y
| YvPmqwjti8BWROoJoAkZ92r+AGiIsz6HAXX4aUaisYNpJLIi4aMBHHJJxO8cynet
| GuDbu6NBMD8wDgYDVR0PAQH/BAQDAgOoMBEGCWCGSAGG+EIBAQQEAwIEUDAJBgNV
| HRMEAjAAMA8GA1UdEQQIMAaHBKn+odUwDQYJKoZIhvcNAQELBQADggEBABaCjwxN
| 4Jcw922sK13vp5pc6axPK3/+meHSU1PxVcTzvWM6z2DGatZ3tO3PvIl9QQZvQyCa
| QjoeziFUOpl4joPy6mZcld7TE8pHhzE9xMrqLpQinzMI29EetMHZSjpYvg7nZmIc
| OnxsTGS7O2DwfEAQOGYHTYdDTn7O6jGfgE9jXdba6hgQik1Lt9hVyslO3pFaiXZO
| 1ZVtJQP7GwHU8NCbiLYlAeyzoQ+DuyoE9wtJZX9C5b3bA9sxRRvDuVLsJrKeFxwZ
| PPiosV+8+q7ImSVDI7h/FlGU27lVIAunS6Svh0R6CpbK+06Q+MhpXyMYYCnVsHi5
| dVk4uAwA/o37U0Q=
|_-----END CERTIFICATE-----

Host script results:
| nbstat:
|_  ERROR: Name query failed: TIMEOUT

Read from /usr/bin/../share/nmap: nmap-service-probes nmap-services.
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
# Nmap done at Fri Mar 17 07:55:45 2023 -- 1 IP address (1 host up)
scanned in 256.53 seconds
```

A.2 Nmap scan with activate HTTPS webinterface

```
    # Nmap 7.93 scan initiated Wed Mar  8 04:32:01 2023 as: nmap -d
-v -A -p- -oA nmaphttpsscan 169.254.161.213
--------------- Timing report ---------------
  hostgroups: min 1, max 100000
  rtt-timeouts: init 1000, min 100, max 10000
  max-scan-delay: TCP 1000, UDP 1000, SCTP 1000
  parallelism: min 0, max 0
  max-retries: 10, host-timeout: 0
  min-rate: 0, max-rate: 0
---------------------------------------------
Packet capture filter (device eth0): dst host 192.168.44.128
and (icmp or icmp6 or ((tcp) and (src host 169.254.161.213)))
Packet capture filter (device eth0): dst host 192.168.44.128
and (icmp or icmp6 or ((tcp) and (src host 169.254.161.213)))
Packet capture filter (device eth0): dst host 192.168.44.128
and (icmp or (tcp and (src host 169.254.161.213)))
```

```
OS detection timingRatio() == (1678268080.408 - 1678268079.906)
* 1000 / 500 == 1.004
Nmap scan report for 169.254.161.213
Host is up, received reset ttl 128 (0.00070s latency).
Scanned at 2023-03-08 04:32:14 EST for 178s
Not shown: 65532 filtered tcp ports (no-response)
PORT    STATE SERVICE   REASON          VERSION
80/tcp  open  http?     syn-ack ttl 128
| http-methods:
|_  Supported Methods: POST
|_http-title: Did not follow redirect to
https://169.254.161.213/
102/tcp open  iso-tsap? syn-ack ttl 128
443/tcp open  https?    syn-ack ttl 128
| ssl-date:
|_  ERROR: Unable to obtain data from the target
| http-cisco-anyconnect:
|_  ERROR: Failed to connect to SSL VPN server
| ssl-cert: Subject: commonName=169.254.161.213/
organizationName=SEWC/countryName=CN
| Subject Alternative Name: IP Address:169.254.161.213
| Issuer: commonName=LOGO Product CA V1.0/organizationName=
Siemens/countryName=DE/organizationalUnitName=Copyright
(C) Siemens AG 2020 All Rights Reserved
| Public Key type: ec
| Public Key bits: 256
| Signature Algorithm: sha256WithRSAEncryption
| Not valid before: 2023-03-01T00:00:00
| Not valid after:  2024-03-01T00:00:00
| MD5:   81db8e43e6664fce8aa1af12b5148a34
| SHA-1: 5755e205860bbcda4a4c96a857a2d7b57329fdc6
| -----BEGIN CERTIFICATE-----
| MIICpzCCAY+gAwIBAgIHA++jAAAAFDANBgkqhkiG9w0BAQsFADB6MR0wGwYDVQQD
| DBRMT0dPIFByb2R1Y3QgQ0EgVjEuMDE6MDgGA1UECwwxQ29weXJpZ2h0IChDKSBT
| aWVtZW5zIEFHIDIwMjAgQWxsIFJpZ2h0cyBSZXNlcnZlZDEQMA4GA1UECgwHU2ll
| bWVuczELMAkGA1UEBhMCREUwHhcNMjMwMzAxMDAwMDAwWhcNMjQwMzAxMDAwMDAw
| WjA2MRgwFgYDVQQDDA8xNjkuMjU0LjE2MS4yMTMxDTALBgNVBAoMBFNFV0MxCzAJ
| BgNVBAYTAkNOMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEmuFd70dTa9Y6Arg4
| UbQPA2fEQ7EQe0foUGtAfRjn2xD7qti4rhrNjG2fbL6iiXw+/0WV6Zl/DCCv6XT+
| SS5OuKNBMD8wDgYDVR0PAQH/BAQDAgOoMBEGCWCGSAGG+EIBAQQEAwIEUDAJBgNV
| HRMEAjAAMA8GA1UdEQQIMAaHBKn+odUwDQYJKoZIhvcNAQELBQADggEBAC4NdyYF
| 3vMC4KmMaWvRvk2TDfsfJttp3Qtf+PugJOlJ0Xgg9sMysVcIsiyO6OGqLrwidXgl
| 9rrsM8DGFUAqhKDRb9UkTbAbGf/TQ0IkAL6s3UE539aHMw4t/HBxzwd40GpY0iy3
| wDhKgB02l5n9B900Aq/psrI7WdMYBC4WlXmFnm6rSQ6ytWTLbEdA6TVQMY7vSWxQ
| pd5KoVoa5FcrshmasbDe2MArifdFpYI3bDcq+9obcF+Pud9yiGg6cp8pkAnzhmtC
| F0bSeThFWfp0XhfrEkh2FUCxxiFyo+pjW8s3TYnAk5koIpde1UhXswq0+/QDSBbX
| 1nqcYG/0veqYOLw=
```

```
|_-----END CERTIFICATE-----
| http-methods:
|_  Supported Methods: GET
Warning: OSScan results may be unreliable because we could
not find at least 1 open and 1 closed port
Device type: WAP|general purpose
Running: Actiontec embedded, Linux 2.4.X|3.X
OS CPE: cpe:/h:actiontec:mi424wr-gen3i cpe:/o:linux:linux_kernel cpe:/o:linux:linux_kernel:2.
OS details: Actiontec MI424WR-GEN3I WAP, DD-WRT v24-sp2
(Linux 2.4.37), Linux 3.2
TCP/IP fingerprint:
OS:SCAN(V=7.93%E=4%D=3/8%OT=80%CT=%CU=%PV=Y%DS=2%DC=T%G=N%TM=640856D0%P=x86
OS:_64-pc-linux-gnu)SEQ(SP=103%GCD=1%ISR=104%TI=I%II=I%SS=S%TS=U)OPS(O1=M5B
OS:4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)WIN(W1=FAF0%W2=FAF0%W3=FAF0%W4
OS:=FAF0%W5=FAF0%W6=FAF0)ECN(R=Y%DF=N%TG=80%W=FAF0%O=M5B4%CC=N%Q=)T1(R=Y%DF
OS:=N%TG=80%S=O%A=S+%F=AS%RD=0%Q=)T2(R=N)T3(R=Y%DF=N%TG=80%W=FAF0%S=O%A=S+%
OS:F=AS%O=M5B4%RD=0%Q=)T4(R=Y%DF=N%TG=80%W=7FFF%S=A%A=Z%F=R%O=%RD=0%Q=)T6(R
OS:=Y%DF=N%TG=80%W=7FFF%S=A%A=Z%F=R%O=%RD=0%Q=)U1(R=N)IE(R=Y%DFI=N%TG=80%CD
OS:=S)

Network Distance: 2 hops
TCP Sequence Prediction: Difficulty=259 (Good luck!)
IP ID Sequence Generation: Incremental

TRACEROUTE (using port 80/tcp)
HOP RTT     ADDRESS
1   0.37 ms 192.168.44.2
2   0.41 ms 169.254.161.213

Read from /usr/bin/../share/nmap: nmap-os-db nmap-service-probes nmap-services.
OS and Service detection performed. Please report any incorrect results
at https://nmap.org/submit/ .
# Nmap done at Wed Mar  8 04:35:12 2023 -- 1 IP address (1 host up)
scanned in 191.56 seconds
```