“Computer & Netwrork Security” Cyber Attack Simulation in the Industrial Control Systems

Ahmad Bashir Usman[†]
University of Padua, Italy
ahmadbashir.usman@studenti.unipd.it

Mauro Conti[‡]
University of Padua, Italy
conti@math.unipd.it

Federico Turrin [‡]
University of Padua, Italy
turrin@math.unipd.it

***Abstract*—The cyber physical system (CPS) is a key infrastructure needed in order to sustain and maintain our society in the future. The growth of CPSs has increased exponentially in the last decades, and our lives rely highly on computrazid networked environments, taking ad- vantage of these physical systems in many different ways, starting from the simplest applications to the most sophisticated devices that are implemented in critical infrastructure. The emergence of several attacks in the Industrial Control Systems (ICS) made the administration monitoring of the ICS a challenging issue to deal with. In this project, i will demonstrate how feasible it's to breach the security of the ICS by simulating the behavior of SCADA and lunch Man In the Middle Attack (MITM) and sniff the ongoing traffic between the Human Machine Interface (HMI) and the Programmable Logic Unite (PLC) in the Industrial Control Systesms.**

***Index Terms*—Industrial Control System, Cyber Physical System, SCADA, Testebed.**

I. Introduction

Industrial control system (ICS) can be defined as a general term which encompasses and used to describe several types of devices of the control systems including networks, systems, controls that are associated with Supervisory control and data acquisition (SCADA), Programmable Logic Controllers (PLC) and Human Machine Interface (HMI), Remote Terminal Units (RTU) and other tools for maintenance and diagnostics of the network protocols.

In order to properly apply security in ICSs and Criticial Infrastructure (CI) environments, it's a vital to have detail knowlaged and be aware of the entire network components in the IT and OT. This will allow us to holistically make an investigation of the process and the system in each part, analyze vulnerabilities, risks and threats and recommend possible solutions. To be able to do so, the Purdue model and ISA 62443 standard are implemented to control the system and secure its components properly.

The Purdue model was first developed by Theodore J. Williams next to other members at Purdue university in the 1990s. Purdue Enterprise Reference Architecture (PERA) model as the name suggests it's maily for enterprises, proves to be a good example defining and distinguishing the five layers/level of critical infrastructure which is deployed in production-lines and provide a better approach to apply security. The implementation of the PERA will resolved the air-wall between ICSs and the main components presented in the IT and the OT.

[†]Department of Information Engineering, Master degree in ICT for Internet and Mutlitimedia (MIME), Track of Cybersystem, University of Padova, Italy

[‡]Part of the Security Through Zeal (SPRITZ) research group in the Department of Mathematics.

II. KingFisher

KingFisher [1]is a modern successful outlier detection mechanism, that was built on Mininet using Machine Learning algorithms. It's the first of its kind that can provide solution and detect attacks in IT and OT traffic in ICS network.

In the experiment, the authors implemented Variational-Autoencoders (VAEs), a branch of an artificial neural- network that classifies the data with no labels provided to the training data. In other words, in an unsupervised manner. The results show that KingFisher has the ability to capture attacks on network and physical layers of TCP/IP models.

After carefully going through the kingfisher paper and understanding how the framework was built, how the architecture designed and the experiment carried out, i noticed that the in the scenario, the duration in which allowed for the attack event to take place (two minutes), the KF-OT IDS can detect anomalies and provide report and determine the communication efficiently in the . While on the other end, the side of the KF-IT IDS, the period in this scenario is not sufficient enough to analyze, predict and provide meaningful information. The second critical perspective suggestion in the implementation of the three Intrusion Detection Systems (KF-IT, KF-OT and KF-PHYS), due to the incompatibility of the data that are arriving to the Correlation Node (CN), leading each of the three to possess a different IDS, resulting asynchronous when collected by CN.

Last but not least, the authors only considered only four types of attacks which are; Man In The Middle attack (MITM), Denial of Service (DoS), Modbus Function Code Modification (MFCM) and Modification of Physical Behavior (MPB).

Finally, The application/paradigm is that it only focuses on a specific IT traffic. in other words, it doesn't identify suspicious activities in the entire system and that include application or transport layers. Also on this paper is that the authors heavily focused on modbus protocol without considering others such as DNP3.

The idea behind presenting and criticizing kingfisher paper on this project is because it simulates exactly the same process and procedure as this paper. Another important reason, by understanding kingfisher properfly, this will simplify my way to extend the paper as it will be delivered as my Masterâ€™s thesis.

III. Simulation & Environmnet

A brief introduction of the mininet environment, pox module and other tools used in this project. The simulation and the architecure of the enviromnet.

Remark III.1. **Mininet:** The Mininet is a public and an open source network emulator that allows researchers to conduct experiments, debug and test networks and allows them to create their own convenient environment for simulating real network behavior. The simulation includes creating virtual switches, hosts, links and controllers. The switches presented in mininet are compatible with OpenFlow which added flexibility to customize the communication as well as Software Defined networking. Implementation of the mininet, the way it works and more details could be found on the mininet original website and also on the paper [2]

Remark III.2. **POX** POX is another networking-software platform that was built using python programming language. When first POX lunch was only functioning as an OpenFlow controller in the earlier version, but it has been extended now it can support and works as an OpenFlow switch, this in general is very important in terms of writing any software which is associated with a network. More details on POX along with its implementation could be found on [3]

Remark III.3. **Environment** The architecture of this networking environment consists of two main hosts (Human Machine Interface, Programmable Logic Unit) connected to one another via switch, and an extra host acts as an attacker (SPRITZER) with an independent switch connected to the switch that links between the HMI and PLC. The figure of this architecture can be vividly seen in the figure 2

IV. MITM

Remark IV.1. **Man in the Middle Attack** Man in the middle attack is a cyber attack, an attack in which the attacker aims to anonymously intercept the communications of two entities/parties who assume that these parties are actively in the communication with each other in the network.

MAC-address flooding attack is a very popular cyber attack. Mainly the MAC-address table stores all the traffic that is started to communicate between switches,these addresses are available on a specific physical port in association with VLAN of the corresponding network. The attack that could possibly arise from MAC-address tables are the so-called flooding attack or overflow attack.

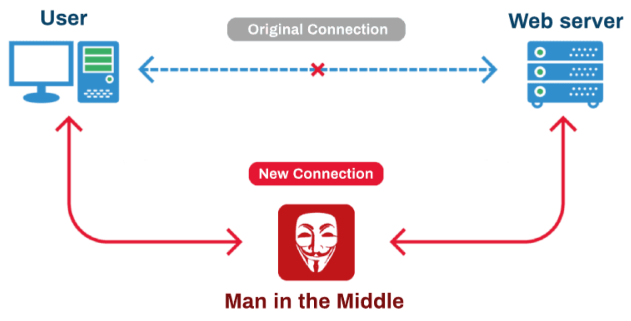


Fig. 1: Typical behavior of MITM attack

In order to comprehensively understand the behavior in which the MAC address-table functioning and the overflow attacks mechanism, we first need to recall firstly how the parties interact with switches in the first place.

MAC-address flooding attack is a very popular cyber attack. Mainly the MAC-address table stores all the traffic that is started to communicate between switches,these addresses are available on a specific physical port in association with VLAN of the corresponding network. The attack that could possibly arise from MAC-address tables are the so-called flooding attack or overflow attack.

In order to comprehensively understand the behavior in which the MAC address-table functions and the overflow attacks mechanism, we first need to recall firstly how the parties interact with switches in the first place.

Before the attack takes place, whenever the switch receives a specific frame forwarding through its port, it takes an action by looking into the MAC-address table to decide where is the proffer destination of the MAC-address. When the frame is forwarded properly, the corresponding MAC-address is therefore, learned from data-link layer (also known as layer2) and stored in the table of MAC-address.

If the forwarded MAC-address previously learned by the computer, then the switch sends the packet to the MAC-address table with designated destination. If on the other hand the MAC-address is not available or doesn't already exist, the switch turns to physical-layer (layer 1) and acts as a hub which results in broadcasting the frame to entire parties belonging to the network. the figure shows the behavior of the man in the middle attack.

V. Attack

Remark V.1. **Overflow** Knowing that the MAC-address table has a limitation size for the recorded address, this is valuable information to bear in mind and understand since the overflow attacks work in such scenarios, here where the attacker comes into play. By flooding the table with an excessively large amount of fake source addresses until it becomes no longer possible to store any MAC-address in the mapping entry.

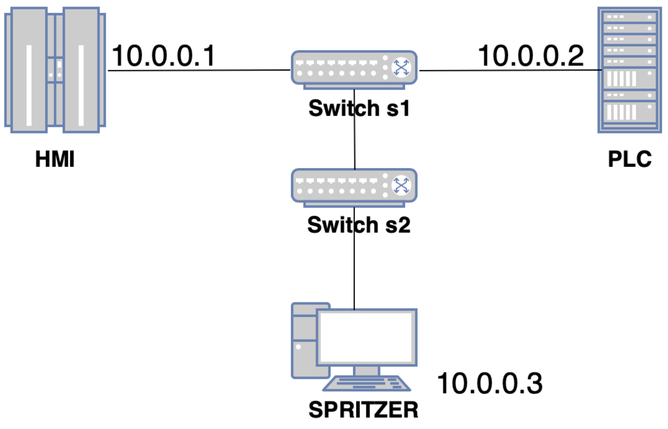


Fig. 2: Architecture of the model

The results of this action forces the switch into fail-open mode, in other words, it acts as a physical layer (layer 1 i,e hub) which gives the attacker the opportunity to view all the ongoing frames between the victim and the second party even with unavailability of the corresponding table which maps the entry

VI. Attack Simulation

In this section, i will demonstrate experiment of this project. This experiment was conducted on Ubuntu.18 Virtual Machine under the MAC-IOS system. Having installed all the experimental's requirements and the dependencies such as mininet, having configured the environment, setted up and simulated the ICS system, it's time to get into work.

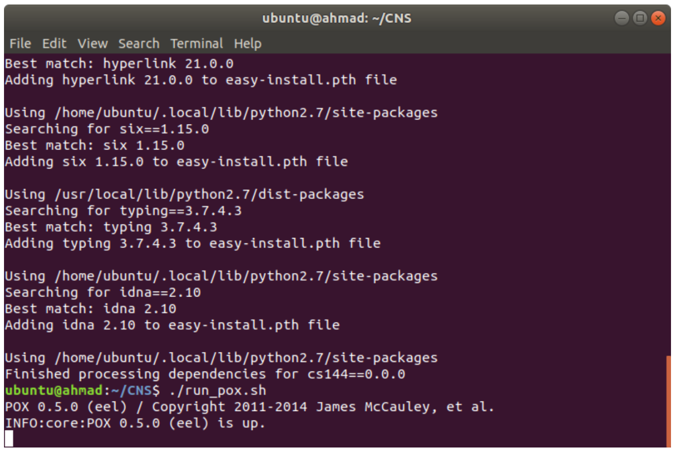
Remark VI.1. Listining Phase
Firstly, we start by launching the POX in the terminal, this will allow the POX to start the networking controller. The purpose of the POX in this scenario is that it will behave and emulate the functionality of the L2 learning switch. Fig. 3.

We leave this terminal open (up), go ahead open another terminal which we can run the main script that contains the entire experiment separately. Fig.4.

This action will result in starting the Mininet network-emulator, and we will see xterm terminals popping up all the nodes in our network. In this case we are more interested in the two hosts (HMI and PLC) and the attacker (SPRITZER). Fig.5.

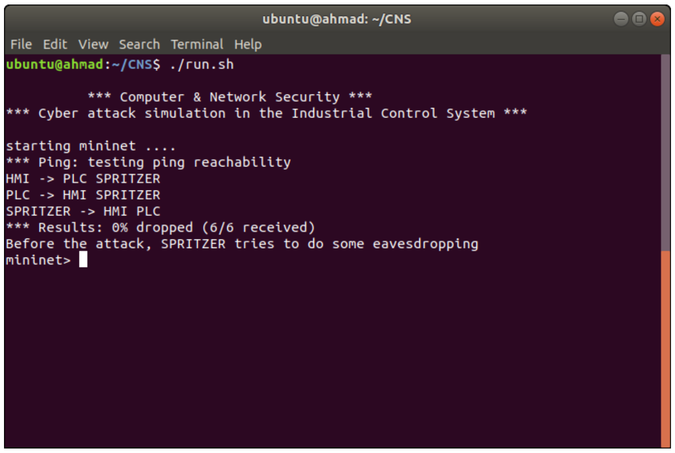
At this point, the PLC will attempt to communicate with the HMI. By pinging to the HMI from the PLC terminal, this action will generate the traffics between the two parties, and we will be able to visualize something like Fig.6.

Since the two devices generate the traffic between each other and are actively reaching each other, now the attacker (SPRITZER) will tcpdump the ongoing traffic in an attempt to eavesdrop the communication from the sender who started the communication (HMI). Itâ€™s premature to embark on the communication at this point. The attacker will only be there listening to nothing. 7 . This is because the switch between the HMI, PL and SPRITZER already learned and stored



```
Best match: hyperlink 21.0.0
Adding hyperlink 21.0.0 to easy-install.pth file

Using /home/ubuntu/.local/lib/python2.7/site-packages
Searching for six==1.15.0
Best match: six 1.15.0
Adding six 1.15.0 to easy-install.pth file

Using /usr/local/lib/python2.7/dist-packages
Searching for typing==3.7.4.3
Best match: typing 3.7.4.3
Adding typing 3.7.4.3 to easy-install.pth file

Using /home/ubuntu/.local/lib/python2.7/site-packages
Searching for idna==2.10
Best match: idna 2.10
Adding idna 2.10 to easy-install.pth file

Using /home/ubuntu/.local/lib/python2.7/site-packages
Finished processing dependencies for cs144==0.0.0
ubuntu@ahmad:~/CNS$ ./run_pox.sh
POX 0.5.0 (eel) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.5.0 (eel) is up.
```

Fig. 3: POX Listining

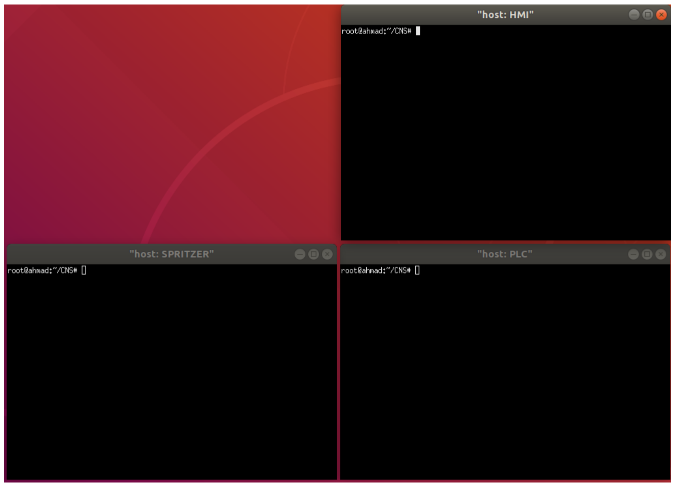


```
ubuntu@ahmad:~/CNS$ ./run.sh

          *** Computer & Network Security ***
*** Cyber attack simulation in the Industrial Control System ***

starting mininet ....
*** Ping: testing ping reachability
HMI -> PLC SPRITZER
PLC -> HMI SPRITZER
SPRITZER -> HMI PLC
*** Results: 0% dropped (6/6 received)
Before the attack, SPRITZER tries to do some eavesdropping
mininet>
```

Fig. 4: Main Mininet architecture

in the MAC-address table, therefore, the communication, in particular, the packet will not be broadcasted, as a result, SPRITZER will not be able to see anything.

Remark VI.2. Attacking Phase Now that we had our

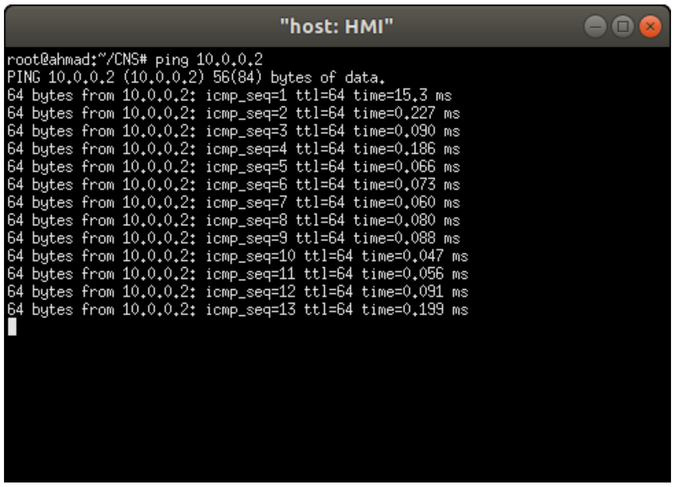


```
root@ahmad:~/CNS#
```

```
root@ahmad:~/CNS#
```

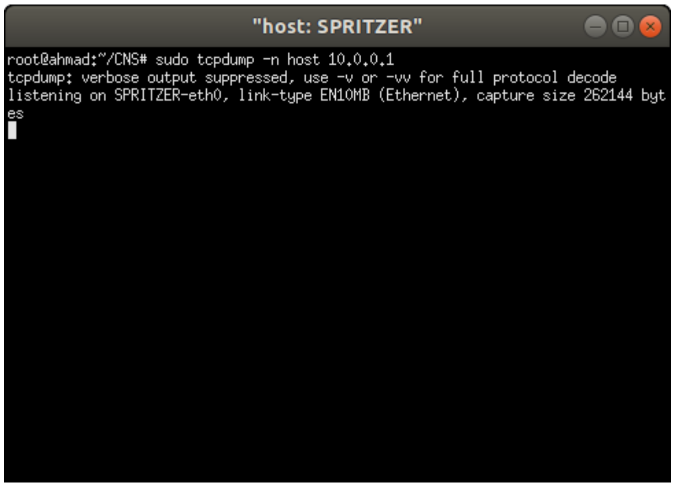
```
root@ahmad:~/CNS#
```

Fig. 5: All the nodes



```
root@ahmad:~/CNS# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=15.3 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.227 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.186 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.066 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.073 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.060 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.088 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.047 ms
64 bytes from 10.0.0.2: icmp_seq=11 ttl=64 time=0.056 ms
64 bytes from 10.0.0.2: icmp_seq=12 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=13 ttl=64 time=0.199 ms
```

Fig. 6: Active Communication between HMI & PLC

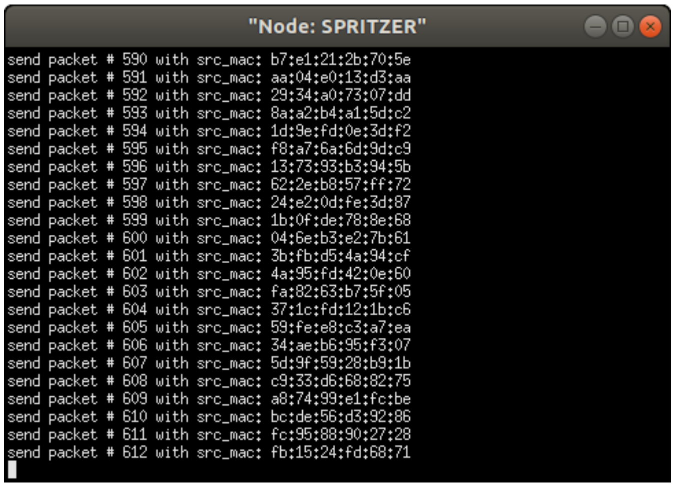


```
root@ahmad:~/CNS# sudo tcpdump -n host 10.0.0.1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on SPRITZER-eth0, link-type EN10MB (Ethernet), capture size 262144 byt
es
```

Fig. 7: SPRITZER in the Listening Phase

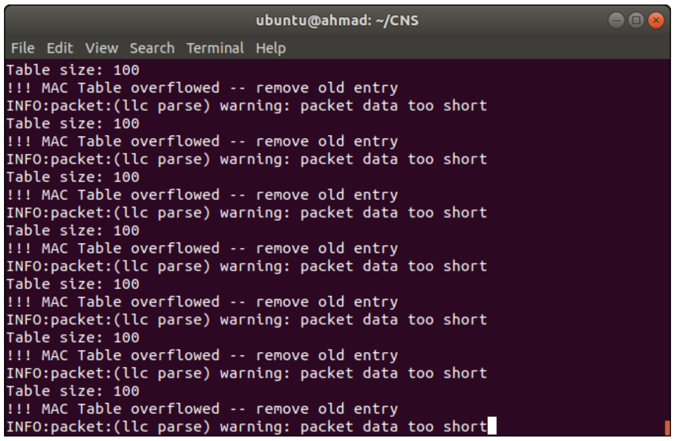
two systems communicating with each other properly with the disability for the attacker to intercept their traffic. We go ahead and open a separate terminal and xterm SPRITZER from the main mininet simulator. From here we launch our attack script which results in the MAC-address table to be overflowed.This command will excessively send huge amount of packets.Fig.8 Going back to our very first terminal in which we executed the POX module command and made it emulate the L2 learning switch, up and in the listening phase. As in figure 9 we will see that it continuously overflows the MAC-addresses.

Now by looking back at the first SPRITZER terminal, we will see that we are successfully able to intercept the traffic by overflowing the traffic which forces the switch to drop all its previous learned addresses and make HMI and PLC to broadcast their MAC-addresses in order to re-establish the connection over and over again which gives the attacker the opportunity to intercept the traffic, beliving that HMI tries to reach PLC. Figure 10 shows the entire system in progress.



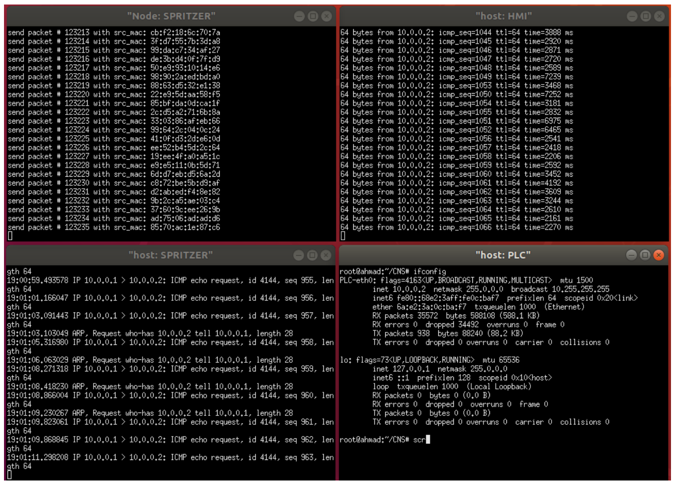
```
send packet # 590 with src_mac: b7:e1:21:2b:70:5e
send packet # 591 with src_mac: aa:04:e0:13:d3:aa
send packet # 592 with src_mac: 29:34:a0:73:07:dd
send packet # 593 with src_mac: 8a:a2:b4:a1:5d:c2
send packet # 594 with src_mac: 1d:9e:fd:0e:3d:f2
send packet # 595 with src_mac: f8:a7:6a:6d:9d:c9
send packet # 596 with src_mac: 13:73:93:b3:94:5b
send packet # 597 with src_mac: 62:2e:b8:57:ff:72
send packet # 598 with src_mac: 24:e2:0d:fe:3d:87
send packet # 599 with src_mac: 1b:0f:de:78:8e:68
send packet # 600 with src_mac: 04:6e:b3:e2:7b:61
send packet # 601 with src_mac: 3b:fb:d5:4a:94:cf
send packet # 602 with src_mac: 4a:95:fd:42:0e:60
send packet # 603 with src_mac: fa:82:63:b7:5f:05
send packet # 604 with src_mac: 37:1c:fd:12:1b:c6
send packet # 605 with src_mac: 59:fe:e8:c3:a7:ea
send packet # 606 with src_mac: 34:ae:b6:95:f3:07
send packet # 607 with src_mac: 5d:9f:59:28:b9:1b
send packet # 608 with src_mac: c9:33:d6:68:82:75
send packet # 609 with src_mac: a8:74:99:e1:fc:be
send packet # 610 with src_mac: bc:de:56:d3:92:86
send packet # 611 with src_mac: fc:95:88:90:27:28
send packet # 612 with src_mac: fb:15:24:fd:68:71
```

Fig. 8: Launching an attack by SPRITZER



```
Table size: 100
!!! MAC Table overflowed -- remove old entry
INFO:packet:(llc parse) warning: packet data too short
Table size: 100
!!! MAC Table overflowed -- remove old entry
INFO:packet:(llc parse) warning: packet data too short
Table size: 100
!!! MAC Table overflowed -- remove old entry
INFO:packet:(llc parse) warning: packet data too short
Table size: 100
!!! MAC Table overflowed -- remove old entry
INFO:packet:(llc parse) warning: packet data too short
Table size: 100
!!! MAC Table overflowed -- remove old entry
INFO:packet:(llc parse) warning: packet data too short
Table size: 100
!!! MAC Table overflowed -- remove old entry
INFO:packet:(llc parse) warning: packet data too short
Table size: 100
!!! MAC Table overflowed -- remove old entry
INFO:packet:(llc parse) warning: packet data too short
```

Fig. 9: Overflow Attack in action



```
send packet # 123213 with src_mac: cb:f2:18:6c:70:7a
send packet # 123214 with src_mac: 3f:d7:55:7b:3d:a8
send packet # 123215 with src_mac: 99:da:c7:34:af:27
send packet # 123216 with src_mac: de:3b:d4:0f:7f:d9
send packet # 123217 with src_mac: 50:e9:93:10:14:e6
send packet # 123218 with src_mac: 98:90:2a:ed:bd:a0
send packet # 123219 with src_mac: 88:63:d5:32:e1:38
send packet # 123220 with src_mac: 22:e9:5d:aa:58:f5
send packet # 123221 with src_mac: 85:bf:da:0d:ca:1f
send packet # 123222 with src_mac: 2c:d5:a2:71:6b:8a
send packet # 123223 with src_mac: 33:03:86:af:eb:66
send packet # 123224 with src_mac: 99:64:2c:04:0c:24
send packet # 123225 with src_mac: 41:0f:d3:2d:e6:0d
send packet # 123226 with src_mac: ee:52:b4:5d:2c:64
send packet # 123227 with src_mac: 19:ee:4f:a0:a5:1c
send packet # 123228 with src_mac: e9:e5:11:0b:5d:71
send packet # 123229 with src_mac: 6d:d7:eb:d5:6a:2d
send packet # 123230 with src_mac: c8:72:be:5b:d9:af
send packet # 123231 with src_mac: d2:ab:ed:f4:8e:82
send packet # 123232 with src_mac: 9b:2c:a5:ae:03:c4
send packet # 123233 with src_mac: 37:60:9c:ee:26:9b
send packet # 123234 with src_mac: ad:75:06:ad:ad:d6
send packet # 123235 with src_mac: 85:70:ac:1e:87:c6
```

```
64 bytes from 10.0.0.2: icmp_seq=1044 ttl=64 time=3888 ms
64 bytes from 10.0.0.2: icmp_seq=1045 ttl=64 time=2920 ms
64 bytes from 10.0.0.2: icmp_seq=1046 ttl=64 time=2871 ms
64 bytes from 10.0.0.2: icmp_seq=1047 ttl=64 time=2720 ms
64 bytes from 10.0.0.2: icmp_seq=1048 ttl=64 time=2589 ms
64 bytes from 10.0.0.2: icmp_seq=1049 ttl=64 time=7239 ms
64 bytes from 10.0.0.2: icmp_seq=1053 ttl=64 time=3468 ms
64 bytes from 10.0.0.2: icmp_seq=1050 ttl=64 time=7252 ms
64 bytes from 10.0.0.2: icmp_seq=1054 ttl=64 time=3181 ms
64 bytes from 10.0.0.2: icmp_seq=1055 ttl=64 time=2832 ms
64 bytes from 10.0.0.2: icmp_seq=1051 ttl=64 time=6975 ms
64 bytes from 10.0.0.2: icmp_seq=1052 ttl=64 time=6465 ms
64 bytes from 10.0.0.2: icmp_seq=1056 ttl=64 time=2541 ms
64 bytes from 10.0.0.2: icmp_seq=1057 ttl=64 time=2418 ms
64 bytes from 10.0.0.2: icmp_seq=1058 ttl=64 time=2206 ms
64 bytes from 10.0.0.2: icmp_seq=1059 ttl=64 time=2592 ms
64 bytes from 10.0.0.2: icmp_seq=1060 ttl=64 time=3452 ms
64 bytes from 10.0.0.2: icmp_seq=1061 ttl=64 time=4192 ms
64 bytes from 10.0.0.2: icmp_seq=1062 ttl=64 time=3609 ms
64 bytes from 10.0.0.2: icmp_seq=1063 ttl=64 time=3244 ms
64 bytes from 10.0.0.2: icmp_seq=1064 ttl=64 time=2610 ms
64 bytes from 10.0.0.2: icmp_seq=1065 ttl=64 time=2161 ms
64 bytes from 10.0.0.2: icmp_seq=1066 ttl=64 time=2270 ms
```

```
gth 64
19:00:59.493578 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 955, len
gth 64
19:01:01.166047 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 956, len
gth 64
19:01:03.091443 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 957, len
gth 64
19:01:03.103049 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
19:01:05.316980 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 958, len
gth 64
19:01:06.063029 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
19:01:08.271318 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 959, len
gth 64
19:01:08.418230 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
19:01:08.866004 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 960, len
gth 64
19:01:09.230267 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
19:01:09.823061 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 961, len
gth 64
19:01:09.868845 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 962, len
gth 64
19:01:11.298208 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 4144, seq 963, len
gth 64
```

```
root@ahmad:~/CNS# ifconfig
PLC-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.2  netmask 255.0.0.0  broadcast 10.255.255.255
        inet6 fe80::68e2:3aff:fe0c:baf7  prefixlen 64  scopeid 0x20<link>
        ether 6a:e2:3a:0c:ba:f7  txqueuelen 1000  (Ethernet)
        RX packets 35572  bytes 588108 (588.1 KB)
        RX errors 0  dropped 34492  overruns 0  frame 0
        TX packets 938  bytes 88240 (88.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@ahmad:~/CNS# scr
```

Fig. 10: Entire system in progress

VII. Concluding & Future work

To summarize, In this project, I have simulated the behavior and architecture of the Industrial Control System. I have shown in practice that it is possible to intrude into the ICS environment and compromise the entire system by placing Man in The Middle between HMI and PLC , launch and attack and intercept the ongoing traffic that are actively in transmission.

In the future work, it's possible to ARP spoof or poison the ongoing traffic by falsifying the ARP messages in the Local-Area-Network. This could be achieved by linking an attacker's (SPRITZER) MAC-address with a specific IP-address of the hosts (HMI or PLC) which legitimately communicate over the network. The moment SPRITZER's MAC-address is able to link the authenticate IP-address, then he can be able to receive any packet directed to HMI or PLC MAC-addresses. Therefore, the attacker can block or modify the messages to the legitimate parties.

References

[1] G. Bernieri, M. Conti, and F. Turrin, "Kingfisher: an industrial security framework based on variational autoencoders," pp. 7–12, 11 2019.

[2] F. Keti and S. Askar, "Emulation of software defined networks using mininet in different simulation environments," in *2015 6th International Conference on Intelligent Systems, Modelling and Simulation*, pp. 205–210, 2015.

[3] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 105–110, July 2008.