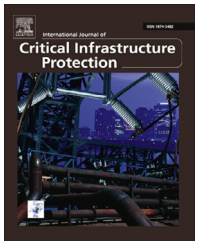
INTERNATIONAL JOURNAL OF CRITICAL INFRASTRUCTURE PROTECTION 22 (2018) 62–69

Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/IJCIP



On PLC network security

Asem Ghaleb[a], *Sami Zhioua*[a,*], *Ahmad Almulhem*[b]

[a] *Information and Computer Science Department, King Fahd University of Petroleum and Minerals, Saudi Arabia*
[b] *Computer Engineering Department, King Fahd University of Petroleum and Minerals, Saudi Arabia*

ARTICLE INFO

Article history:
Received 28 October 2015
Revised 19 October 2017
Accepted 22 May 2018
Available online 11 June 2018

Keywords:
Industrial Control Systems Security
SCADA Security

ABSTRACT

Programmable Logic Controller (PLC) is an important component in modern Industrial Control Systems (ICS) particular in Supervisory Control and Data Acquisition (SCADA) systems. Disturbing the normal operation of PLCs can lead to significant damages ranging from minor annoyance to large scale incidents threatening the life of people. While most of existing work in the SCADA security literature focuses on the communication between PLCs and field devices, this paper presents a network security analysis of the communication between PLCs and the engineering stations in charge of setting up and configuring them. Interestingly, this aspect of SCADA security was exploited by the most famous SCADA attack, namely, Stuxnet. Using a testbed with a common PLC device, we successfully carried out three network attacks leading to serious compromise of typical PLCs.

1. Introduction

Industrial control systems (ICS) refer to several classes of computer-based control systems that include supervisory control and data acquisition (SCADA) systems, distributed control systems (DCS), and Process Control Systems (PCS). These systems are widely used in the daily operation of many critical infrastructures and key resource (CIKR) sectors which are related directly to the health and well-being of citizens. They include sectors such as energy, water, transportation, chemical plants, communications, to name but few. A failure in any of these sectors may have negative cascading impact on our way of life. As such, their security is a major national concern worldwide [1].

SCADA is typically used to describe an ICS that monitors and controls transmission and distribution facilities. For example, electrical transmission and distribution systems deliver electricity, pipelines transport oil and gas, aqueducts and distribution systems provide drinking water. A commonly used device in SCADA systems is the Programmable Logic Controller (PLC). A PLC is a control device responsible for collecting and processing input and output (I/O) data from field devices (e.g. motors, pumps, sensors, etc.).

As its name indicates, a PLC is programmable, that is, by loading a new program, the PLC can be reconfigured to function in a different way. Typically, a new program is loaded by connecting the PLC with an engineering station (equipped with a configuration software) via direct wire or through a LAN. In addition to reprogramming the PLC, the engineering station can send control commands to the PLC such as *start, stop, check status*, etc. Therefore, an attacker who can interfere and/or compromise the communication between the engineering station and the PLC can cause a lot of damage to the whole SCADA system. Stuxnet attack [2,3], which targeted Iran's nuclear facilities, exploited this particular vulnerability and loaded a malicious program onto the PLCs. Apart from Beresford's talk in Black Hat, 2011 [4], this type of vulnerability has not been studied in the literature. Most of the existing work focuses on the HMI-PLC and on the PLC-field devices

* Corresponding author.
E-mail addresses: asem@kfupm.edu.sa (A. Ghaleb), zhioua@kfupm.edu.sa (S. Zhioua), ahmadsm@kfupm.edu.sa (A. Almulhem).

https://doi.org/10.1016/j.ijcip.2018.05.004
1874-5482/

communications [5–10] through protocols such as Modbus, DNP3, etc. The human machine interface (HMI) is a terminal display device which is usually used to display the status of the PLC and the control process in a graphical way. HMI enables operators to interact with the PLCs and give them some commands. Moreover, the process output, alarms, events, etc are displayed on the HMI. Some vendors deploy the HMI applications as part of the Engineering Station; however, the main functionality of the Engineering Station is to configure the PLC network topology and to design and write PLC logic programs.

This paper presents a security analysis of the network communication between PLCs and engineering stations. In the rest of the paper, engineering station refers to a host with PLC configuration software. In a testbed equipped with a common PLC, namely, Siemens S7-400 in addition to its corresponding configuration software, namely, Simatic PCS7 8.1, we successfully carried out three network security attacks which allowed to interfere with the PLC-PCS7 communication and send arbitrary commands to the PLC. The three security attacks, namely, replay, Man-In-The-Middle (MITM), and command modification, are common IT network security attacks, but they are not typically used to interfere with PLC-PCS7 communication.

The paper is organized as follows. In Section 2, we review related work from the literature. In Section 3, we provide some details about PLCs, their communication with the engineering stations, and related security considerations. The PLC lab setup is presented in Section 4. In Sections 5–7, we illustrate the replay, MITM, and stealth command modification attacks respectively. Finally, we provide conclusion remarks in Section 8.

2. Related work

A number of research papers detail network attacks on SCADA systems. Most of existing work focuses on studying attacks on Modbus protocol, considered as the most established network protocol for Industrial Control Systems. Huisting et al. proposed an exhaustive taxonomy of attacks targeting the Modbus protocol [11]. More than 200 attacks were reported and briefly discussed covering both variants of Modbus, namely, the Serial variant and the TCP variant. Although the taxonomy was exhaustive, no attack has been implemented nor tested. Hence, for most of the listed attacks, there are no guarantees that they are feasible in practice.

Gao et al. reported and tested different categories of network attacks on SCADA involving several protocols (Modbus, DNP3, Allen Bradley) [6]. The attack categories included response injection, command injection, man-in-the-middle, replay, and denial of service. The attacks have been tested using a SCADA security laboratory (Mississipi State University SCADA security lab) combining virtual hosts, simulated components, and physical devices (Water storage tank, etc.). According to the authors, most of the attacks were possible because all common ICS protocols are not using authentication nor digital signature. In addition to attack reporting, Gao et al. implemented an Intrusion Detection System (IDS) for SCADA based on Neural networks. The IDS works by distinguishing between normal and abnormal behaviors. The IDS detection accuracy was above 90% for response injection attacks, but around 12% for replay attacks.

Morris and Gao tested a more complete set of network attacks on SCADA with a total of 17 attacks ranging from reconnaissance, response injection, command injection and denial of service targeting mainly Modbus protocol [9]. The attacks were tested on the same SCADA security testbed used by Gao et al. [6]. Reconnaissance attacks ranged from address scan to device identification. Response injection attack consists in injecting false data values in response packets sent by polled devices. Command injection attack consists in injecting false commands by rogue devices to alter set points (e.g. Low and High values of water storage tank), to alter system control scheme (manual vs automatic), or to change register bits on mater devices. Denial of service attack is achieved by sending invalid Cyclic Redundancy Code (CRC), by transmitting out of turn packets, or by simple SYN flood.

The most recent work studying network attacks on SCADA systems is the one by Maynard et al. [10] which considers a less common protocol, namely, IEC60870-5-104 (104 for short, widely used in control communication for water, gas and electricity utilities and particularly common in European systems). The work details two network attacks, namely, man-in-the-middle and replay. Both attacks consist of three steps: detection (detects 104 hosts), capture (capturing packets using sniffers), and attack (either replaying or modifying packets). The replay attack consists of replaying packets as they are (without any modification). Consequentely, the attack does not work since the TCP/IP kernel drops packets with the same ack and seq numbers. Even Wireshark tags them in black since they are clearly replayed. The authors admit that the proof of concept Replay attack can be made more efficient using a python script to replace packet values which is exactly what we did in this paper. The MITM attack uses ARP poisoning between the SCADA server and the LAN gateway. Captured packets are first cleaned (non 104 packets are dropped), then the Cause of Transmission (CoT) flag is switched from ON to OFF which is used to choose the correct program for processing the packet. Two testing environments have been used for the MITM attack: a lab simulated environment and PRECYSE [12] which is a real electricity distribution testbed.

Almost all existing work in the literature focuses on studying network attacks while SCADA systems are running. All the attacks target the communication between MTUs (Master Terminal Units, e.g. PLCs) and RTUs (Remote Terminal Units, e.g. pumps, switches, etc.). This paper deviates from most of the existing work by focusing on the initial communication between a PLC and the engineering station (e.g. PCS7) whose aim is to control and reprogram the PLC.

The closest work to ours was presented by Beresford [4], in which several attacks on the communication between the engineering station and the PLC are reported. Beresford showed that several attacks are possible on Siemens PLC S7-1200 and S7-300 models such as replay, information retrieval, and even launching a remote shell on a PLC. However, by trying the reported attacks on our setting (Siemens PLC S7-400), they did not work. The explanation is that the PLC firmware in our setting is more recent and naturally more secure than the

firmware used in the Beresford's experiments. Also, our replay attack is more "interactive" than the Beresford's since some recorded packets are only replayed after a response is received from the PLC. Beresford's replay attack consists simply in sending all the recorded packets in sequence without considering any response from the PLC.

3. Programmable Logic Controller (PLC)

A PLC is a particular type of embedded devices that is programmed to manage and control physical components (motors, valves, sensors, etc.) based on system inputs and requirements. A PLC typically has three main components, namely, an embedded operating system, control system software, and analog and digital inputs/outputs. Hence, a PLC can be considered as a special digital computer executing specific instructions that collect data from input devices (e.g. sensors), sending commands to output devices (e.g. valves), and transmitting data to a central operations center.

PLCs are commonly found in supervisory control and data acquisition (SCADA) systems as field devices. Because they contain a programmable memory, PLCs allows a customizable control of physical components through a user-programmable interface. PLCs can be (re)configured using proprietary software installed on a standard computer (typically with Microsoft Windows OS). Reconfiguring the PLC consists in changing the control system software, known also as the programming layer of the PLC. This layer is in charge of providing the PLC device with the logic to manage the connected field devices. The programming layer can be reprogrammed using versions of typical languages such as C, Pascal, etc. To make the PLC programming more accessible, a graphical language called Ladder logic is used. Indeed, Ladder logic allows engineers, who may be unfamiliar with full-fledged programming languages, to reconfigure the PLC through an easy and intuitive graphical interface.

A very common example of the configuration software is the Siemens Simatic Step 7 [13] for Simatic controllers. The software allows engineers to perform three main tasks: (1) write the graphical Ladder logic code, (2) compile it to machine code for execution and (3) upload the compiled code to the device.

3.1. *PLC security issues*

PLCs are typically located at the edges of the SCADA network interfacing between the cyber and physical components. Hence, PLCs are critical to the operation of the industrial process. An adversary, seizing control of a PLC, may cause significant damage to the whole industrial process.

Besides, some characteristics of ICS field devices make them more vulnerable to cyber attacks. First, they are typically deployed for a long period of time (up to several decades). Second, ICS architectures often include several legacy devices lacking basic security features and not allowing the integration of new security features. Third, due to the high availability requirements, the replacement of field devices is not practically possible. Finally, common security solutions for IT systems are not applicable in a network involving ICS field devices because of performance and other requirements of control processes.

3.2. *Engineering station–PLC communication*

Major PLC manufacturers (Siemens, Allen-Bradley, Phoenix Contact, etc.) provide efficient software environments to program and configure their PLCs. The programs are written in a variety of languages including graphical languages such as Ladder logic. PLC programs need to be efficient, lightweight and guarantee secure communication with the other field devices once deployed.

Programming a PLC consists of uploading the written program to the PLC after it has been developed and tested at an engineering station. Typically, the engineering station is connected to the PLC with Ethernet. This communication can be point-to-point involving a simple Ethernet cable between the PLC and the engineering station or is a part of a network including other stations. Because the PLC program is in charge of controlling how the PLC works and commands field devices, the upload procedure should be performed in a secure way. An adversary who can interfere with this uploading procedure can launch a variety of attacks ranging from DOS to seizing full control of the PLC.

Simatic PCS7 is the programming environment for Siemens PLCs. It is a comprehensive software suite offering a variety of features to configure control systems, in particular PLCs. The software provides a graphical user interface for simple operation and clear display of configuration data. In order to upload a new configuration program, an engineering station with Simatic PCS7 software communicates with the PLC through Ethernet and using COTP (Connection Oriented Transport Protocol).

COTP protocol is an open systems transport layer protocol built on the top of TCP. COTP is not commonly used and is based on a very old specification (RFC 905 [14]). This enabled us to record COTP packets traveling between PLC and the Engineering Station and then targeting PLC with several attacks. We found that COTP related traffic is transferred in plain text with no authentication. PLCs using COTP for communication with the Engineering Stations depend on password protection or authentication which can be evaded easily due to the fact that packets are transferred in plain text and can be crafted. Very scarce documentation about COTP protocol is publicly available and few attempts were made to reverse engineer it [15]. Although the COTP protocol has been replaced by TCP in most applications, it is still being used by Simatic PCS7 software. This can be seen as a manifestation of security-by-obscurity which is a common protection measure in ICS.

As shown in Fig. 1, the three main commands that the engineering station with Simatic PCS7 software can send to the PLC are the following:

- Start command: turns the PLC on, assuming it is currently turned off. The start command is typically used when re-programming the PLC. In particular, the start command packets are sent along with the new PLC program packets.

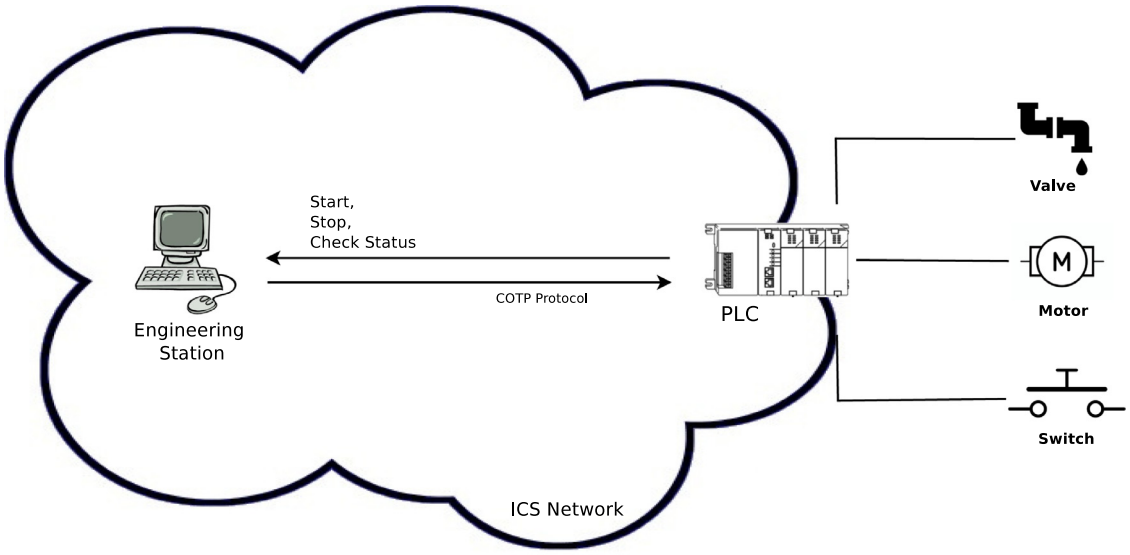


Fig. 1 – Simatic PCS7 - PLC communication channel.

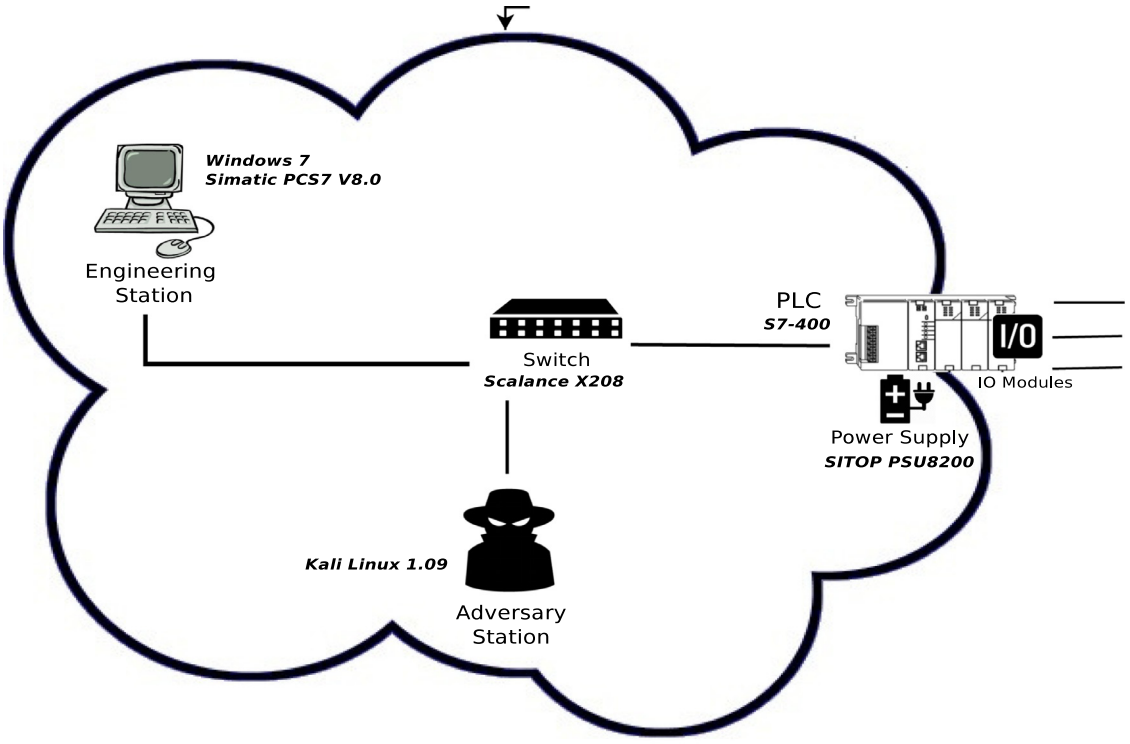


Fig. 2 – PLC lab setup.

- Stop command: turns off the PLC.
- Check status: enquires about the current status of the PLC.

4. PLC lab setup

Fig. 2 shows the PLC lab setup used to implement network attacks on PLC. The PLC used in the lab is Siemens S7-400 which is typically used in power plants (including nuclear), pipelines, oil and gas refineries, water and waste systems. Compared to the more popular S7-300, S7-400 PLC is made for bigger and complex machinery systems with a faster processor, larger work memory and more Input/Output ports. Power is supplied to the PLC with a SITOP PSU8200 box. The engineering station is a Windows 7 host equipped with Siemens Simatic PCS7 V8.0 software. Attacks are launched from an adversary station with Kali Linux 1.09. The PLC as well as the stations are connected through a network switch, namely, Siemens Scalance X208.

5. Replay attack

The first implemented PLC network attack is a typical replay attack. The attack consists of 3 steps: starting a PCS7 command (stop, start, etc.), capturing the packets, and replaying the captured packets at a later time. The captured packets corresponding to a given command are first processed by filtering out any packets that are not part of the command's traffic. Since PCS7-PLC communication uses the COTP protocol (port 102), any other packets are filtered out. In addition, only packets in the PCS7-PLC direction are kept (packets in the opposite direction are filtered out). The cleaned traffic for each command is then stored in a pcap file.

Initially, tcpreplay [16] suite is used to replay the recorded packets (cleaned pcap file). tcpreplay suite comes with different tools such as tcpprep (packets pre-processor that isolates packets in each direction), tcprewrite (pcap file editor which rewrites packet headers), tcpreplay (replays pcap files onto the network), etc. Using these tools, the pcap file is pre-processed before replaying by changing the source IP address and recomputing the checksum value in each packet. Once the preprocessed pcap file is replayed on the PLC, most of the packets are discarded by the PLC and the replay attack fails. After investigation, it turns out that the packets are discarded for two main reasons. First, the sequence (SEQ) and acknowledgment (ACK) numbers in the replayed packets are not changed. Consequently, the TCP/IP kernel at the PLC tags those packets as duplicates and discards them. Second, tcpreplay tool replays the packets in the pcap file one after the other without waiting for any response from the PLC. Hence, the PLC receives some packets out of the proper sequence and discards them. This problem has been recently observed by Maynard et al. [10].

To overcome these problems and to guarantee that the replayed packets are accepted by the TCP/IP kernel at the PLC, we resorted to write a customized python script using scapy [17]. Scapy is a powerful packet manipulation program written in python and hence can be easily used in python scripts. It features a variety of packet manipulation capabilities including: sniffing and replaying packets in the network, network scanning, tracerouting, etc. However, the most useful scapy features for our replay attack are the ability to rewrite the sequence and acknowledgment numbers and to match requests and replies.

Dealing with the duplicate sequence and acknowledgment numbers consists of recalculating these numbers and rewriting them with scapy. Manipulating packet headers using scapy is straightforward since any packet field is simply accessible by the dot operator (e.g. *ip.src, tcp.flags, rcv[TCP].seq*). Initially, random sequence and acknowledgment numbers are chosen. Then, at each packet sending, the numbers are incremented and added to the next packet.

Replaying packets in the appropriate sequence and time requires waiting for the response of some packets before releasing the next packet. Scapy provides several variants of the *Send* function which is in charge of sending a packet in the network. For packets not requiring a response (e.g. Acknowledgment packet), the simple *sendp* function is used. The sendp function takes as input the packet as well as the network interface. For packets requiring a response, several functions can be used:

- *sr*: Send and receive packets at layer 3
- *sr1*: Send packets at layer 3 and return only the first answer
- *srp*: Send and receive packets at layer 2
- *srp1*: Send and receive packets at layer 2 and return only the first answer
- *srloop*: Send a packet at layer 3 in loop and print the answer each time
- *srploop*: Send a packet at layer 2 in loop and print the answer each time

In our program, we used the srp1 function because there is always one single response packet sent by the PLC. Algorithm 1 shows the core of the python script using the scapy features. The *REPLAY* subroutine takes as input the pcap file, the network interface, the attacker's IP address and port number. In addition, arbitrary values are chosen to initialize the ACK and RSTACK numbers. The for loop inside the subroutine goes through the packets one by one. For each packet, IP and TCP checksums are removed (lines 7 and 8) so that the network interface card recalculates newer values, the source IP and port numbers are updated (lines 9 and 10), the sequence numbers are incremented (lines 12 and 20), the packet is replayed using either *sendp* function (for SYN and RST packets) or the *srp1* function (lines 14 and 19).

The two other subroutines, namely, *MITMCONF* and *MAIN* are added to make the attack script self-contained. *MITMCONF* is in charge of configuring both *Snort* and *syslog* while the *MAIN* is the driver function that will call either *MITMCONF* or *REPLAY* depending on the value of the last parameter.

The above python program has been tested using two attack scenarios. In the first scenario, the replay attack was launched from the same host (IP address) used for the capture, that is, the host with PCS7 software. In the second scenario, the replay attack was launched from a different host on the same network, that is, the attacker machine with Kali. In each scenario, two types of commands are tried, namely, start and stop. The replay attack was successful in both scenarios for both types of commands. Hence, an unknown attacker machine (without PCS7 software) on the same network can turn the PLC ON or OFF by simply replaying a start or stop command. This clearly might cause significant damage to a SCADA system.

6. Man In the middle attack

The communication between PCS7 host and the PLC uses COTP over Ethernet. Ethernet protocol uses Address Resolution Protocol (ARP). Hence, theoretically the communication is vulnerable to Man In The Middle (MITM) attacks through ARP Poisoning.

In a switched Ethernet network, a host A who tries to communicate with a host B (with a known IP address) needs its physical address (MAC). The MAC address can be obtained by broadcasting an ARP request to all hosts in the network. In a normal scenario, only host B will send a response with the correct IP-MAC pair. In an attack scenario, an attacker (host C) in the same network will send a fake response with a false IP-MAC claiming to be the owner of B's IP address. Typically, the attacker floods the network with its fake response forcing

Algorithm 1 Replay a sequence of captured packets using Scapy.

```
function REPLAY(pcapfile, eth_interface, srcIP, srcPort)
    recvSeqNum ← 0
    SYN ← True
    for packet in rdpcap(pcapfile) do
        ip ← packet[IP]
        tcp ← packet[TCP]
        del ip.chksum                    ▷ Clearing the checksums
        ip.src ← srcIP ▷ Specify the attacker machine IP and PORT
        ip.sport ← srcPort
        if tcp.flags == ACK or tcp.flags == RSTACK then
            tcp.ack ← recvSeqNum+1
            if SYN or tcp.flags == RSTACK then
                sendp(packet, iface=eth_interface)
                SYN ← False
                continue
            end if
        end if
        rcv ← srp1(packet, iface=eth_interface)
        recvSeqNum ← rcv[TCP].seq
    end for
end function
function MITMCONF(eth_interface)
    snort_conf_file ← '/etc/snort/snort.conf'
    syslog_Conf_file ← '/etc/syslog-ng/syslog-ng.conf'
    # Configuring Snort
    fsnort ← open(snort_Conf_file, 'a')
    fsnort.write("output alert_syslog: LOG_LOCAL6 LOG_ALERT")
    fsnort.close()
    # Configuring syslog
    fsyslog ← open(syslog_Conf_file, 'a')
    fsyslog.write("filter f_start_plc")
    fsyslog.write("facility(local6) and match("snort" value ("PLC START")
    fsyslog.write("filter f_stop_plc")
    fsyslog.write("facility(local6) and match("snort" value ("PLC STOP")
    fsyslog.write("Program ("python /root/replay.py stop.pcap "+ inface")
    fsyslog.close()
end function
function MAIN
    if len(sys.argv) != 6) then
        exit(1)
    end if
    if sys.argv[5] == 1 then
        MITMCONF(sys.argv[2])
    end if
    if sys.argv[5] == 2 then
        REPLAY(sys.argv[1], sys.argv[2], sys.argv[3], sys.argv[4])
    end if
end function
```

the victim host (A) to accept the false pairing and ignore the correct one sent by host B. ARP poisoning is typically launched between two hosts allowing the attacker to insert himself as a tunnel between the two victims and consequently sniff all packets between them.

In our scenario, an ARP poisoning MITM attack is implemented between the PCS7 host and the PLC using ettercap tool [18]. The attack is successful and all the packets exchanged between the PCS7 and PLC are tunneled through the attacker host (Kali). A MITM attack can be passive or active. A passive version consists in simply observing the traffic of the PLC and hence breaking the confidentiality of the commands sent to the PLC. An active version is more dangerous since it allows the attacker to tamper with the packets and commands and consequently interfere with the normal operation of the system.

7. Stealth command modification attack

The third attack is a combination of replay and MITM attacks which aims at sniffing the traffic between the PCS7 and PLC and then interfering with sent commands by replaying other commands in a stealth way. Through this attack, an adversary can completely change the behavior of the SCADA system since sending a command leads to the execution of another command.

The attack goes through three main steps: MITM attack, command detection, and replay of a false command. Fig. 3 illustrates the attack. Initially, the attacker (Kali) starts by launching a MITM attack to place himself between the PCS7 and the PLC exactly as described in the previous section. Then, it stays in an idle state observing the traffic passively and waiting for commands sent by the PCS7 host to the PLC (Step 1 in Fig. 3). For the sake of command detection in the network, Snort intrusion detection system (IDS) [19] is used. Snort is a signature-based network IDS which allows to detect patterns of traffic inside the network. Currently, Snort is configured to detect two types of commands, namely, start and stop. As soon as the attacker detects a command from the PCS7 host to the PLC (Step 2), a different command will be replayed to the PLC (Step 4). That is, if a start command is detected, the attacker replays a stop command to the PLC. If a stop command is detected, the attacker replays a start command (with a different PLC program[1]) to the PLC. However, it is easy to notice that if the attacker interferes with a start command to make it a stop command (or the opposite), the PCS7 will quickly notice that something is wrong. To make the attack as stealth as possible, the attacker continues the communication with the PCS7 host while impersonating the PLC (Step 3). So for the PCS7 host the communication appears to be perfectly normal. This technique has been used by Stuxnet in its famous attack on Iran's nuclear facility. Indeed, to make the attack stealth, Stuxnet recorded normal frequency values. Then, at attack time, it played those recorded frequencies to make the monitoring system believes that centrifuges are operating as normal [2,3].

[1] Recall that start command is sent along with the new PLC program packets (Section 3).

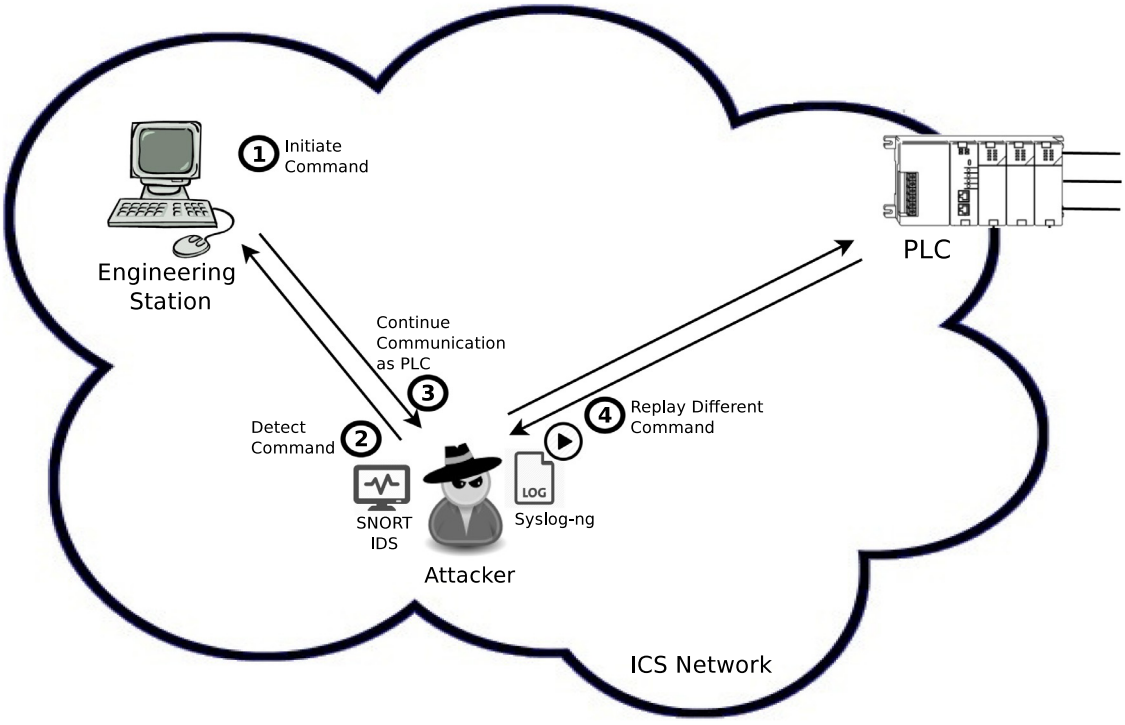


Fig. 3 – Stealth command modification attack.

Snort is an IDS which allows only to detect known patterns in the network traffic. However, the stealth command modification attack requires the launching of the replay attack (python program) as soon as a command is detected in the traffic. To fill this gap, Snort is configured to log alerts to Syslog-ng utility [20]. In turn, Syslog-ng is configured to trigger the replay attack upon the reception of appropriate Snort alerts.

8. Conclusion

Securing SCADA systems is a major concern worldwide, as such system are responsible for the daily operation of modern critical infrastructures and industries. In this work, we demonstrated that a SCADA system can be seriously compromised by mounting network attacks targeting PLCs. PLCs are very common components in SCADA systems. They sit between HMIs and field devices and are in charge of sending commands and receiving data to/from field devices. Since a PLC is programmable, it can be completely compromised by loading a malicious control program. Through the detailed description and implementation of three attacks (Replay, MITM, and command modification), we showed that the communication between the PLC and the engineering station can be compromised leading to serious SCADA system instability. We showed that, with open source tools and simple python scripts, one can mount successful attacks. In particular, programming and configuration traffic directed to PLCs may be replayed, sniffed, and/or modified. PLCs can be protected against the discussed attacks mainly by providing a level of encryption for the transferred packets. This can be a challenge specially with the light size of PLCs' memory which makes it difficult to do encryption and decryption inside the memory of PLCs. However, hardware cipher models can be used as extended modules attached to the PLCs. In addition, static entries of the ARP tables can be used to provide a level of defense against MITM attacks. The PLC response time during the normal communication is slightly less than while working under MITM attack, this factor can be used to detect and prevent MITM attacks.

Acknowledgment

The authors would like to acknowledge the support of the National Science, Technology and Innovation Plan (NSTIP) under project number 13-INF281-04.

REFERENCES

[1] A. Nicholson, S. Webber, S. Dyer, T. Patel, H. Janicke, Scada security in the light of cyber-warfare, Computers & Securityvol. 31(4) (2012) pp. 418–436.
[2] N. Falliere, L. Murchu, E. Chien, W32.Stuxnet Dossier, Symantec Security Response(2012).
[3] S. Zhioua, The middle east under malware attack: Dissecting cyber weapons, Proceedings of the IEEE ICDCS Workshop on Network Forensics, Security and Privacy (NFSP) (2013).
[4] D. Beresford, Exploiting Siemens Simatic S7 PLC, Black HatUSA (2011).
[5] P. Huitsing, R. Chandia, M. Papa, S. Sheno, Attack taxonomies for the Modbus protocol, International Journal of Critical Infrastructure Protection 1(0) (2008) 37–44.

[6] W. Gao, T. Morris, B. Reaves, D. Richey, SCADA Control System Command and Response Injection and Intrusion Detection, eCrime Researchers Summit (eCrime)(2010) 1–9.
[7] L. Pietre-Cambacedes, M. Tritschler, G. Ericsson, Cybersecurity myths on power control systems: 21 misconceptions and false beliefs, IEEE Transactions on Power Deliveryvol. 26(1) (2011) pp. 161–172.
[8] Y. Yang, K.M. Laughlin, T. Littler, S. Sezer, E.G. Im, Z. Yao, B. Prang-gono, H. Wang, Man-in-the-middle attack test-bed investigating cyber-security vulnerabilities in smart grid scada systems, Proceedings of the International Conference on Sustainable Power Generation and Supply (SUPERGEN) (2012) pp. 1–8.
[9] T. Morris, W. Gao, Industrial control system cyber attacks, Proceedings of the First International Symposium on ICS & SCADA Cyber Security Research, ICS-CSR(2013) pp. 22–29.
[10] P. Maynard, K.M. Laughlin, B. Haberler, Towards understanding man-in-the-middle attacks on IEC 60870-5-104 SCADA networks, in: Proceedings of the Second International Symposium on ICS & SCADA Cyber Security Research, ICS-CSR, 2014, pp. 30–42.
[11] M. Organization, Modbus protocol specification, http://www.modbus.org.
[12] K.M. Laughlin, S. Sezer, P. Smith, Z. Ma, F. Skopik, PRECYSE: Cyber-attack detection and response for industrial control systems, Proceedings of the Second International Symposium on ICS & SCADA Cyber Security Research, ICS-CSR(2014) pp. 67–71.
[13] A.G. Siemens, The simatic PCS7 process control system brochure, April 2013.
[14] Network Working Group, ISO transport protocol specification (RFC 905), April 1984.
[15] G. Devarajan, Unraveling SCADA protocols: Using sulley fuzzer, Proceedings of the DEFCON Fifteenth Hacking Conference(2007).
[16] tcpreplay, http://tcpreplay.synfin.net.
[17] P. Biondi, Scapy, http://www.secdev.org/projects/scapy.
[18] A.a. NaGA, Ettercap, http://ettercap.sourceforge.net.
[19] M. Roesch, Snort: Lightweight Intrusion Detection for Networks, LISA vol. 99(1) (1999) pp. 229–238.
[20] R. Gerhards, The syslog protocol (RFC 5425), March 2009.