

Mini Project #3: Reconnaissance and Attack on ICS Networks

CS 6263/ECE 8813: Cyber Physical System Security (Fall 2019)

Assigned: October 19, 2019 Due: November 2, 2019, 11:59pm EST

Introduction

In the previous project, we have learned how to read, write and modify a Ladder Logic program. This is a very important skill to have as an attacker when trying to compromise an ICS network through PLCs. But, before you can do that you need to have knowledge of the network such as the number of nodes communicating, the processes they are running, important set points, etc. Then, you can send malicious commands to the critical actuators with the same protocol and perform severe sabotage while sending false data to the HMI representing the normal operating condition to keep your attack stealthy. Since most of the ICS protocols (e.g., Modbus and DNP 3.0) are communicating with plain text, it is very easy to launch attacks on these systems. As the first step, an attacker would perform reconnaissance on the network by collecting and analyzing the network traffic. The next step will be sending the malicious packets in the ICS network to cause sabotage in the system.

1 Part 1 – Reconnaissance (20 Points)

Look at the provided *modbus_capture.pcapng* file. This file contains communication between several devices running the Modbus protocol associated with a specific process. Assume there is a reactor within a reactor chamber. The target process is to control the reactor temperature and maintain it around 100 C. There are two valves connected to the reactor chamber, one of which fills the reactor chamber and the other valve drains the reactor chamber. There is a master device in the network which polls the sensors data and sends the temperature set point and valve operations to the corresponding devices periodically. More specifically, the master device monitors two sets of sensors:

1. In the first set, there are three measurements: pressure sensor 1 (e.g., 144), temperature sensor 1 (e.g., 800), and fuel meter 1 (e.g., 560).
2. Similar to the previous set, in the second set, there are three measurements: pressure sensor 2 (e.g., 350), temperature sensor 2 (e.g., 530), and fuel meter 2 (e.g., 1070).

The master executes the control algorithm based on the collected sensor values and determines the temperature set point and the status of the valves. It toggles coils on a

PLC to simulate the opening or closing of one of the two valves in order to regulate the temperature. The master also writes the temperature set point to another PLC. This writing of *set point* is to indicate that the master is adjusting the reactor temperature to optimize the reactor functioning. Use Wireshark to analyze the given pcap file and answer the following questions in the csv file provided (**part1.csv**) based on the explained scenario:

1. How many different devices can you identify in the Wireshark capture? (Example Answer: 10)
2. What is the IPv4 Address of the master device? (Example Answer: 192.168.1.1)
3. What is the IPv4 Address of PLC controlling the temperature set point?
4. What is the IPv4 Address of PLC controlling the valves?
5. What is the IPv4 Address of sensor set 1?
6. What is the IPv4 Address of sensor set 2?
7. How many registers does the PLC controlling the temperature set point have?
8. What is the default port used to send Modbus messages? (Example Answer: 80)
9. Two of the devices in the network correspond to one physical device. What is the Mac address for that physical device? (Example Answer: 12:34:56:78:90:12)

NOTE: It is your responsibility that the formatting of the CSV is maintained. The first column is the question and the second should be the answer. Removing or adding columns (or other formatting problems) could result in point deductions. You must also save the provided CSV file as a CSV file (Suggestion: use Microsoft Excel).

2 Part 2 – Attack Implementation (80 Points)

Now that you understood and identified different devices and registers from the pcap file, it is time to create a Modbus simulation. There are 3 python files that are provided for you. Modify *tcp_master.py* to send commands that will set the temperature set point very high AND close both the valves, attempting to cause an explosion. Use Wireshark to capture the traffic between the master and the slaves and submit the pcap file.

You are welcome to use the provided VM (Appendix A) or use the provided Docker container for your implementation (Appendix B).

2.1 *run_simulation.py*

Using *run_simulation.py* create 3 devices, as described in the table below. There is a master device at 192.168.1.13, and the other two devices are slave devices. 192.168.1.14 is the slave device which controls the the valves, and 192.168.1.15 is the slave device which

controls the temperature set point. Skeleton code is provided, you will have to create additional host(s), link(s), slave(s). Additionally, you will have to add code to start the slave(s) and execute *tcp_master.py*.

Table 1: Device IPv4 Table

IPv4 Address	Device Description
192.168.1.13	Master
192.168.1.14	Controlling Valves (Slave)
192.168.1.15	Temperature Set Point Controlling Device (Slave)

2.2 *tcp_slave.py*

The majority of *tcp_slave.py* has already been completed. You are required to add 2 memory blocks and the initialization for those blocks. The first memory block is for 3 registers starting at address 100, e.g. Register 100, Register 101, Register 102. The second memory block is for 2 coils starting at address 0, e.g. Bit 0 and Bit 1. The first register in 192.168.1.15 represents the temperature value and the 2 coils in 192.168.1.14 represent the coils that control the valves. A 0 for the coil value represents an open-valve and a 1 represents a closed-valve.

2.3 *tcp_master.py*

There is skeleton code provided that illustrates how to format a few of the requests. You will be able to test the skeleton code once you have added the memory blocks from Part 2.2. Your implementation should consist of three parts:

1. Read the current temperature and coil values.
Send a READ_HOLDING_REGISTERS request from the Master to 192.168.1.15.
Send a READ_COILS request from the Master to 192.168.1.14.
2. Set the new temperature.
Send a WRITE_SINGLE_REGISTER request from Master to 192.168.1.15 to set the register at 100 (Register 100) to any temperature that is significantly greater than 100 C (e.g., 1000 C).
Send a WRITE_MULTIPLE_COILS request from Master to 192.168.1.14.
3. Read the new temperature and coil values
Send a READ_HOLDING_REGISTERS request from the Master to 192.168.1.15.
Send a READ_COILS request from the Master to 192.168.1.14

When you leave the Docker container it will automatically delete itself, so you will need to repeat steps 1-9 again to get the packet capture. The packets in the Docker container should have an *Ethernet II* layer.

Suggestions

When running a capture session, turn off WIFI/Internet as this reduces noise captured by Wireshark thus making it easier to identify Modbus packets. You can also explore Display Filters in Wireshark to quickly identify Modbus packets.

Evaluation

We will process your pcap file looking specifically for *12 Modbus packets*. The request packets (6 total) and response packets (6 total). Do not have additional Modbus packets in your pcap submission.

Deliverable and Submission Instructions

Create a zip file named `<First_Name>_<Last_Name>_mp3.zip`, that includes all of your files and submit it on Canvas. Your zip file should be generated in such way that when extracted all of the files are in the base directory (there should be no directories when un-zipped) ***Note: We will use an auto-grader, therefore any failure to follow the submission instructions will cause 20% point loss.***

```
<First_Name>_<Last_Name>_mp3
|-- part1.csv
|-- part2.pcap
|-- part3.pcap
|-- run_simulation.py
|-- tcp_master.py
|-- tcp_slave.py
```

Please check your submission multiple times as we will not accept any submissions after the deadline for any reasons (e.g. wrong file submission).

A *Setting Up Virtual Machine*

- We will be using Mininet for the project. Pre-built VM images including Mininet and other useful software is provided in the following link : [Mininet VM](#).
- Once the download is complete, import the VM onto VirtualBox (or any other VMM) by double-clicking the .ovf file. If you are using a Mac, an excellent option is [Parallels Desktop](#) (You can get a student discount for the full software)
- Bump memory up to at least 2GB
- Add an empty optical drive in the “Storage” part of settings by clicking the symbol that looks like a “CD with a plus”
- Turn on the VM. When prompted to login, use the following details:
 - Login: mininet
 - Password: mininet
- To get a GUI, on the command line type the following :
 - `sudo apt-get update`
 - `sudo apt-get install xinit ubuntu-desktop` (this takes a while)
 - `startx`
 - Ctrl+Alt+T to get a terminal. Type : `rm -rf ~/.config`
- [Install Guest Additions](#) – so that GUI resolution auto-adjusts
- Update Wireshark: `sudo apt-get install wireshark`
- Download the Modbus_tk library from [Modbus_tk](#). Go through the examples on the same Github page. They are very helpful.

```
sudo apt-get install pip
pip install modbus_tk
```

- Here are some additional documentation on Modbus protocol: [link 1](#), [link 2](#), [link 3](#).

NOTE: You should not have to update the version of Ubuntu, and the only software updates should be the ones described in Appendix A.

B *Docker Container (VM Alternative)*

You are welcome to use the Docker container for this assignment instead of the VM. Docker is a computer program that performs operating-system-level virtualization, known as "containerization". You can find more information [here](#). A docker image for this assignment has already been created. You can follow the instructions below to install and run your code in a Docker container. You will have to submit the resulting pcap file.

1. Create Docker Account and install [Docker](#) ([Docker Toolbox](#) for legacy systems)
2. Pull Docker Image

```
docker pull stripuramallu3/8813-mini-project-03-packages
```

3. Run Image to create container (this will put you in the `/root/` directory)

```
docker run -it --rm --privileged -e DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix \
-v /lib/modules:/lib/modules \
stripuramallu3/8813-mini-project-03-packages
```

This will put you in the `/root/` directory of the container

4. Update and install packages (Skip this step, packages already set-up. If there are issues with any package, you can set the packages up again)

```
apt-get update
apt-get install python-pip
apt-get install tshark
pip install modbus_tk
```

5. Use another shell to copy the files into the */root/* directory ([docker cp](#)). Example:

```
docker cp /path/to/run_simulation.py <container_id>:/root/  
docker cp /path/to/tcp_master.py <container_id>:/root/  
docker cp /path/to/tcp_slave.py <container_id>:/root/
```

6. Run *run_simulation.py* to start Mininet topology and devices
7. Start *tshark* inside mininet

```
mininet> h13 tshark -w h13_packet_capture.pcap &
```

8. Run *tcp-master.py*
9. Copy pcap file back to local and open with Wireshark

When you leave the Docker container it will automatically delete itself, so you will need to repeat steps 1-9 again to get the packet capture. The packets in the Docker container should have an *Ethernet II* layer.

C *Friendly Suggestions*

- You should NOT have to modify the line in `run_simulation.py` that invokes the slaves. You can change it to debug or add extra logging, but do so at YOUR OWN RISK. In previous semesters, many students changed this line, which resulted in problems that could not be fixed. If you choose to make modifications, you should change it back to the original/provided code. (Keep in mind you still have to invoke the second slave)
- For `tcp_slave.py` you are expected to create 2 blocks. The first block should create 3 contiguous HOLDING.REGISTERS starting at address 100, which will result in Registers 100, 101, and 102. The second block should create 2 coils starting at address 0. There are many ways to create blocks and still get the correct traffic output, but we are looking for the memory blocks specified in the assignment PDF.
- Once you have implemented `tcp_slave.py` you should be able to run the skeleton `tcp_master.py` and see the packets in the skeleton code execute. If your execute requests throw errors, then that means that the syntax for the execute function was not correct, or what you specified inside execute does not exist on the slave.
- If you continue to get a Connection Refused error, then it is almost certain that the issue falls under two categories:
 - You made a change to the slave invoking command in `run_simulation`
 - `tcp_slave.py` was not implemented correctly, specifically the creation of the memory blocks
- If your errors fall under some other category, then the issue might be with the environment itself. Please re-install the virtual machine and follow the set-up instructions.