

LAB 2: PERFORMING ATTACKS AND ADDING SECURITY MECHANISMS TO SCADA CONTROL SYSTEMS

Estimated Time: 1 hour and 40 minutes

Purpose: The purpose of this Lab exercise is to teach students how to create attacks on the control system and then how to create a defense mechanism by implementing iptables on the PLC Network.

Objective: Students will use Radzio! to perform a series of injection attacks on the PLC. These attacks should disrupt the normal behavior of the system. Additionally, students will write iptables rules to implement an access control list on the SCADA network traffic and therefore prevent the attacks from working.

Lab Setup and Requirements: To begin this lab, you will need to have VirtualBox and the Water Tank simulation running. Iptables will be configured and run on the Water Tank PLC.

EXERCISE #1 – INJECTION ATTACK ON THE LECTURER SIMULATION

On this exercise students will perform an injection attack by fabricating messages with different settings and sending them to the target PLC. Students will use the Radzio! software to fabricate the messages.

The target will be the heat exchanger PLC running on the lecturer's computer. After successfully attacking the lecturer simulation, a security mechanism will be put in place by the lecturer, and then the students will be asked to repeat the attack.

1. Download Radzio! from [here](#) and extract the contents of the zip file on a folder.



- Open RMMS.exe. On the main window, go to Connection->Settings. Select Modbus TCP under "Protocol", Register address starting from 0 under "Addressing convention", and type the target PLC address on "IP address: " field. Also, make sure that the TCP port is 502.

The "Connection settings" dialog box is shown with the following configurations:

- Protocol:** Modbus TCP (selected)
- Addressing convention:** Register address (starting from 0) (selected)
- Modbus RTU:** Port: COM5, Baudrate: 115200, Parity: NONE, Stop bits: 2
- Modbus TCP:** IP address: 192.168.137.124, TCP port: 502
- General:** Timeout (ms): 100, Delay between polls (ms): 0
- DTR:** Active (selected)
- RTS:** Active (selected)

Buttons: OK, Cancel

- Click on File->New and Connection->Connect. On the new spreadsheet that appears, select Holding registers to view the PLC memory data

The "Radzio! Modbus Master Simulator" main window is shown with the following configurations:

- Device Settings:** Device ID: 1, Address: 0, Length: 10
- Display options:** Decimal (selected), Integer, Hexadecimal, Binary, 32bit float, 32bit sw. float, 64bit float, 64bit sw. float
- ASCII display:** Enable (checked), 1 char/reg, 2 char/reg, 2 char/reg
- Table:**

	Alias	0	Alias	10
+0		0		0
+1		4000		0
+2		6000		0
+3		2		0
+4		0		0
+5		1		0
+6		4886		0
+7		0		0
+8		0		0
+9		0		0
- Status:** Polls: 1146, OK: 1146, Errors: 0
- Footer:** 192.168.137.124:502, Tx: 13.43 KB, 0.117 KB/s, Rx: 22.95 KB, 0.283 KB/s, Request

4. The number on the second line of the spreadsheet (+1) has the temperature setpoint multiplied by 100: $4000 = 40^{\circ}\text{C}$. Change this setting by double-clicking on the second line and inserting a new value.
5. After successfully attacking the lecturer's simulation, disconnect by going to Connection->disconnect. Wait until the instructor has put the defense mechanism in place and then try the attack again by clicking on Connection->connect
6. Were you able to connect to the target PLC?

EXERCISE #2 - START SCADA LAB ENVIRONMENT

1. To start the virtual machine in the VirtualBox Manager, select the scadalab VM, right-click and select Start>Normal Start. Login using the credentials provided in Lab 1 (username:ccre, password:ccre).
2. Open Terminal by going to Applications -> Terminal Emulator
3. Navigate to the scripts folder with the command:

```
cd /home/ccre/scadalab/scripts
```

4. Run the script to configure the network with the command:

```
./netstart.sh
```

If it asks for a password, type: ccre

5. Start the Water Tank simulation with the command:

```
./heatexchanger.sh
```

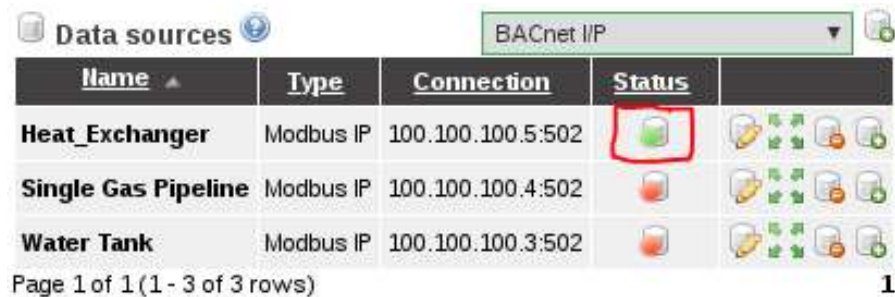
6. Launch the heat exchanger HMI by opening the internet browser (Applications > Web Browser) and navigate to:

100.100.100.2:8080/ScadaBR

Login to ScadaBR using username:admin, password:admin



- Click on Data Sources on the top menu and then enable heat exchanger data source to allow ScadaBR to pull data from OpenPLC.



- Click on Graphical Views, select the Heat Exchanger HMI from the drop down menu.



- Verify the system is working by changing the temperature setpoint and observing the results.

EXERCISE #3 – PREPARING RADZIO! FOR THE ATTACKS

On this exercise, students will prepare Radzio! Modbus to inject messages on the PLC running the Water Tank simulation

Section 1: Starting Radzio! Modbus

- Open Terminal by going to Applications -> Terminal Emulator
- Navigate to Radzio! folder with the command:

```
cd /home/ccre/scadalab/lab2/Radzio
```

- Start Radzio! with the command:

```
wine RMMS.exe
```



Section 2: Configuring Radzio! and connecting to the PLC

1. On the main window, go to Connection->Settings. Select Modbus TCP under "Protocol", Register address starting from 0 under "Addressing convention", and type 100.100.100.5 on the "IP address: " field, which is the target's IP address. Also, make sure that the TCP port is 502.
2. Click on File->New and Connection->Connect. On the new spreadsheet that appears, select Holding registers to view the PLC memory data, and make sure that the "Display options" is set to "Integer"
3. Click on Fine->New again and arrange the new window so that you can see both spreadsheets. On the new window, select Holding registers, and type 2048 on Address. Select "32bit sw. float" under "Display options".

The image shows two windows from the Radzio! software, both titled 'NONAME1' and 'NONAME2'. Each window has a 'Device Settings' section with 'Device ID' set to 1 and 'Addressing convention' set to 'Holding registers'. The 'Address' field is set to 0 for NONAME1 and 2048 for NONAME2. The 'Length' is set to 10 for both. The 'Display options' section shows 'Integer' selected for NONAME1 and '32bit sw. float' selected for NONAME2. The 'ASCII display' section has 'Enable' checked and '1 char/reg' selected for both. Below the settings are tables showing the data received from the PLC.

NONAME1 Data Table:

Alias	0	Alias	10
+0	7033		0
+1	5000		0
+2	0		0
+3	0		0
+4	0		0
+5	0		0
+6	0		0
+7	0		0
+8	0		0
+9	0		0

NONAME2 Data Table:

Alias	2048	Alias	2058
+0	0.4000		0.0000
+1			
+2	0.1000		0.0000
+3			
+4	0.0000		0.0000
+5			
+6	7033.3400		0.0000
+7			
+8	0.0000		0.0000
+9			

At the bottom of each window, there are status bars showing 'Polls', 'OK', and 'Errors' counts.

By following these steps, you now can see the memory data from the PLC. However, it can be a little hard to change settings from these windows because we currently don't know what each number means. Section 3 will guide you to open the PLC program on the PLCopen Editor to then identify where each variable is located in the PLC memory.

Section 3: Finding variables' location on the PLC memory

1. Open Terminal by going to Applications -> Terminal Emulator
2. Navigate to PLCopen Editor folder with the command:

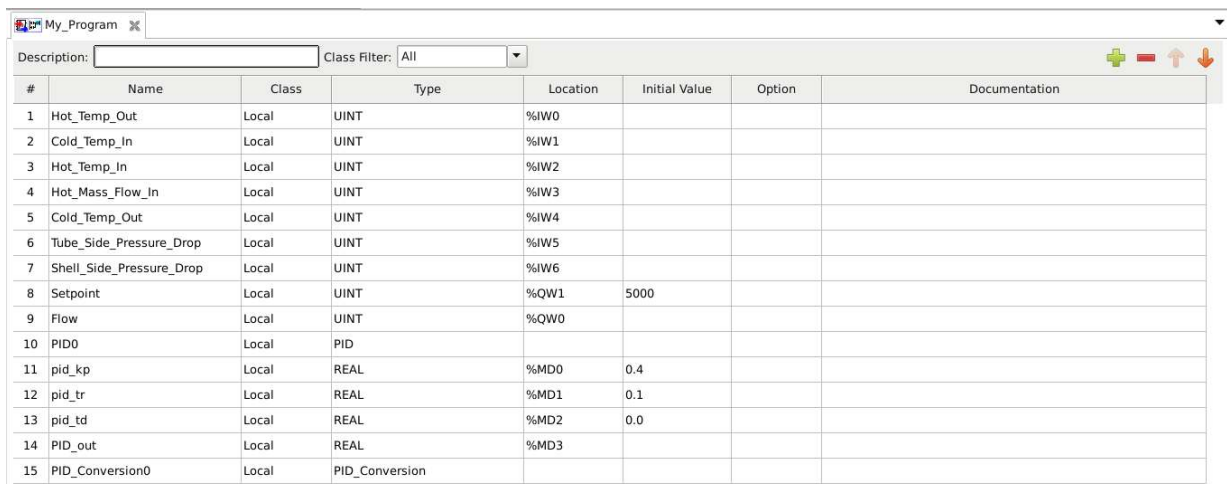
```
cd /home/ccre/scadalab/lab1/editor
```

3. Start the editor with the command:

```
python PLCOpenEditor.py
```

4. In the PLCopen Editor, select File>Open. Navigate to ccre/scadalab/lab1 folder (you can find the ccre folder on the left sidebar). Double click the heat_exchanger.xml file in the Lab 1 Directory.

5. Double-click on "My Program" and look at the variables table on the top of the screen



The screenshot shows the PLCopen Editor window titled "My Program". At the top, there is a "Description:" field and a "Class Filter:" dropdown set to "All". Below this is a table with 8 columns: #, Name, Class, Type, Location, Initial Value, Option, and Documentation. The table lists 15 variables, mostly of type UINT or REAL, with their locations in PLC memory (e.g., %IW0, %QW1, %MD0).

#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	Hot_Temp_Out	Local	UINT	%IW0			
2	Cold_Temp_In	Local	UINT	%IW1			
3	Hot_Temp_In	Local	UINT	%IW2			
4	Hot_Mass_Flow_In	Local	UINT	%IW3			
5	Cold_Temp_Out	Local	UINT	%IW4			
6	Tube_Side_Pressure_Drop	Local	UINT	%IW5			
7	Shell_Side_Pressure_Drop	Local	UINT	%IW6			
8	Setpoint	Local	UINT	%QW1	5000		
9	Flow	Local	UINT	%QW0			
10	PID0	Local	PID				
11	pid_kp	Local	REAL	%MD0	0.4		
12	pid_tr	Local	REAL	%MD1	0.1		
13	pid_td	Local	REAL	%MD2	0.0		
14	PID_out	Local	REAL	%MD3			
15	PID_Conversion0	Local	PID_Conversion				

6. Observe the 5th column called "Location". All variables stored in the PLC data memory are located at regions %Iwn, %QWn, or %MDn where *n* is a number from 0 to 1024. Write down the name and the location (the *n* position) of each %QWn and %MDn variable.

Section 3: Identifying variables on the PLC memory

1. Go back to Radzio! Modbus software.
2. Fill out both Holding Registers tables with the information you collected from the previous section. The row +0 should contain the variable on the PLC program located at %QW0, the row +1 should contain the variable located at %QW1, and so on...
3. You should end up with tables like these:

The image shows two screenshots of the Radzio! Modbus software interface, specifically the 'Holding registers' table for two different PLC addresses.

NONAME1

Device Settings: Device ID: 1, Address: 0, Length: 10. Display options: Decimal (selected), Integer, Hexadecimal, Binary, 32bit float, 32bit sw. float, 64bit float, 64bit sw. float. ASCII display: Enable (unchecked), 1 char/reg, 2 char/reg, 2 char/reg sw.

	Alias	0	Alias	10
+0	Flow	7033		0
+1	Setpoint	5000		0
+2		0		0
+3		0		0
+4		0		0
+5		0		0
+6		0		0
+7		0		0
+8		0		0
+9		0		0

Polls : 7167 OK : 7167 Errors : 0

NONAME2

Device Settings: Device ID: 1, Address: 2048, Length: 10. Display options: Decimal, Integer, Hexadecimal, Binary, 32bit float, 32bit sw. float (selected), 64bit float, 64bit sw. float. ASCII display: Enable (unchecked), 1 char/reg, 2 char/reg, 2 char/reg sw.

	Alias	2048	Alias	2058
+0	pid_kp	0.4000		0.0000
+1				
+2	pid_tr	0.1000		0.0000
+3				
+4	pid_td	0.0000		0.0000
+5				
+6	PID_out	7033.3400		0.0000
+7				
+8		0.0000		0.0000
+9				

Polls : 6655 OK : 6548 Errors : 106

Now the information you see makes more sense. You should now be ready to start the attacks.

EXERCISE #4 – INJECTION ATTACKS

On this exercise, students will use Radzio! Modbus to inject messages on the PLC running the Heat Exchanger simulation

Attack 1: Heating up the system

1. Set the Setpoint to 12000
2. On this setting, the cold mass input flow to keep the temperature at 120°C is too low. The PID algorithm cannot keep a steady state and oscillates.

Attack 2: Unreachable temperature

1. Set the Setpoint to 2000
2. It is physically impossible to reach any temperature colder than the cold input stream. The PID algorithm tries to increase the flow infinitely and only accumulates error.

Attack 3: Damaging the flow control valve

1. Set pid_kp to 1.8
2. Set Setpoint to 5000
3. Set valve_register to 0
4. On this setting, the proportional gain is too high which causes the system to oscillate badly!

Attack 4: Be creative

1. Mess up with the system in a bad way.
2. Check the ScadaBR HMI to see better the consequences of your attack.



EXERCISE #5 – SETTING UP DEFENSES

Iptables is a rule-based firewall software. The user can define which machine is allowed to communicate on the network by using a set of rules. On this exercise, students will write iptables rules that will prevent unauthorized users to connect to the PLC.

Section 1: Writing iptables rules

1. On Radzio! go to Connection->Disconnect.
2. Open Terminal by going to Applications -> Terminal Emulator
3. Log in the PLC shell by typing the command:

```
sudo docker exec -it plc0 bash
```

4. Create an iptables rules file with the command:

```
nano /etc/iptables.test.rules
```

5. This command will open an editor window showing a blank file. Type the following inside the editor:

```
*filter

#Drop everything but our output to internet
-P FORWARD DROP
-P INPUT DROP
-P OUTPUT ACCEPT

#Allow established connections (the responses to our outgoing traffic)
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

#Allow local programs that use loopback
-A INPUT -s 127.0.0.0/8 -d 127.0.0.0/8 -i lo -j ACCEPT

#Allow only the HMI (ScadaBR) to talk to this PLC
-A INPUT -s 100.100.100.2 -p tcp --dport 502 -m state --state NEW -j ACCEPT

COMMIT
```

Lines starting with the # character are comments and can be omitted. This file set rules to block all incoming connections (-P INPUT DROP), except the connections coming from the HMI (-A INPUT -s 100.100.100.2 ...)



```
GNU nano 2.7.4      File: /etc/iptables.test.rules      Modified

*filter

#Drop everything but our output to internet
-P FORWARD DROP
-P INPUT DROP
-P OUTPUT ACCEPT

#Allow established connections (the responses to our outgoing traffic)
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

#Allow local programs that use loopback
-A INPUT -s 127.0.0.0/8 -d 127.0.0.0/8 -i lo -j ACCEPT

#Allow only the HMI (ScadaBR) to talk to this PLC
-A INPUT -s 100.100.100.2 -p tcp --dport 502 -m state --state NEW -j ACCEPT

COMMIT

^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
^X Exit          ^R Read File     ^N Replace        ^U Uncut Text    ^T To Spell
```

6. Save the file and exit by typing Ctrl+X and then Y

Section 2: Applying the new rules and testing the configuration

1. Back to the terminal, type the following command to apply the recently created rules:

```
iptables-restore < /etc/iptables.test.rules
```

2. Verify if the new rules were applied with the command:

```
iptables -L
```

```
root@bf32a4dfac16:/OpenPLC_v2# iptables -L
Chain INPUT (policy DROP)
target    prot opt source                destination           state RELATED,ESTABLISHED
ACCEPT    all  --  anywhere              anywhere
ACCEPT    all  --  loopback/8           loopback/8
ACCEPT    tcp  --  HMI.plcnet           anywhere              tcp dpt:502 state NEW

Chain FORWARD (policy DROP)
target    prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target    prot opt source                destination
```

With this command, we can verify that the default policy for incoming connections (Chain INPUT) is DROP, which means that it will reject all incoming connections. The exceptions to this rule are the three listed under this section. The first exception states that it should accept all incoming connections that are responses from requests (state RELATED). The second exception states that it should accept all connections coming from loopback. These are internal connections and therefore are harmless. The third exception states that any connection from the HMI should be accepted.

3. Verify if the HMI can still communicate with the PLC by opening the Web Browser. Change a few settings (temperature setpoints) to make sure that the system is working properly.
4. Repeat Exercise #3 (Sections 1 and 2) and verify if you can still have access to the PLC



ACKNOWLEDGEMENTS

This lab was developed at the University of Alabama in Huntsville by Stefanie Smith, Ben McGee, Thiago Alves, Joseph Lee, and Tommy Morris.

OpenPLC is a completely open programmable logic controller with development environment, human machine interface, programmable logic controller source code, and reference hardware available at <http://www.openplcproject.com/>. The OpenPLC project was founded by Thiago Alves of the University of Alabama in Huntsville.

The Simulink models, human machine interface implementation, and ladder logic programming used for the gas pipeline and water storage tank test beds used for this laboratory exercise are the copyrighted property of the University of Alabama in Huntsville.

