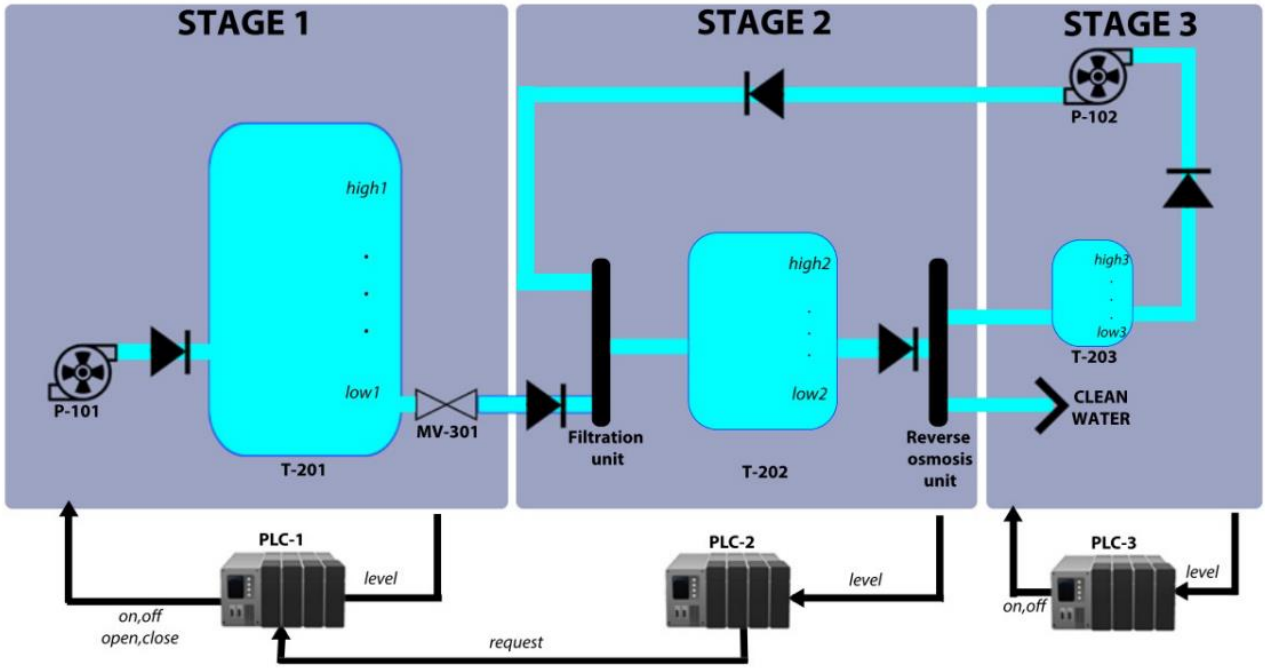
Network Security Project

Man In The Middle Part A: proof of correctness

Casarotti Giulio, Gallenti Gabriele, Simonetti Andrea, Tonelli Alessio

The aim of this brief report is to document the correct operation of the system developed to successfully execute a “Man In The Middle” attack between two entities laying inside the network at our disposal.

The above mentioned network is structured as follows:



In addition to the three PLCs there is a fourth device, called HMI, that is responsible for displaying to the user the system status.

For sake of simplicity, we will focus on a single couple of interacting devices only: PLC1 and HMI.

Entities involved.

Entity name	Entity address (IPv4)	Entity MAC address
PLC1	172.17.0.5	02:42:ac:11:00:05
HMI	172.17.0.2	02:42:ac:11:00:02
MITM device	172.17.0.6	02:42:ac:11:00:06

Let’s start with the canonical behavior. The system is running and the HMI periodically sends requests to the PLCs in order to keep the interface updated on the system status.

Here below you can see a piece of the conversation between the two.

	Source	Destination				
41 12:40:57,5648…	-0.000… 172.17.0.2	172.17.0.5	Modbus/TCP	78	Query: Trans:	0; Unit: 1, Func: 4: Read Input Registers
43 12:40:57,5651…	0.0001… 172.17.0.5	172.17.0.2	Modbus/TCP	77	Response: Trans:	0; Unit: 1, Func: 4: Read Input Registers

Detail of the request:

```
> Frame 41: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface docker0, id 0
v Ethernet II, Src: 02:42:ac:11:00:02 (02:42:ac:11:00:02), Dst: 02:42:ac:11:00:05 (02:42:ac:11:00:05)
    > Destination: 02:42:ac:11:00:05 (02:42:ac:11:00:05)
    > Source: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
      Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.17.0.2, Dst: 172.17.0.5
> Transmission Control Protocol, Src Port: 51796, Dst Port: 502, Seq: 1, Ack: 1, Len: 12
> Modbus/TCP
v Modbus
      .000 0100 = Function Code: Read Input Registers (4)
      Reference Number: 0
      Word Count: 1
```

Detail of the response:

```
> Frame 43: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface docker0, id 0
v Ethernet II, Src: 02:42:ac:11:00:05 (02:42:ac:11:00:05), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
    > Destination: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
    > Source: 02:42:ac:11:00:05 (02:42:ac:11:00:05)
      Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.17.0.5, Dst: 172.17.0.2
> Transmission Control Protocol, Src Port: 502, Dst Port: 51796, Seq: 1, Ack: 13, Len: 11
> Modbus/TCP
v Modbus
      .000 0100 = Function Code: Read Input Registers (4)
      [Request Frame: 41]
      [Time from request: 0.000208209 seconds]
      Byte Count: 2
    > Register 0 (UINT16): 70
```

As one might expect, PLC1 appears as the destination and HMI as the source for the request packet. And you can see exactly the reverse for the response packet.

Now we start the attack. The first phase consists of doing “Arp Spoofing”. We flood the network with ARP packets, in order to force both the HMI and the PLC1 to update their ARP tables. We do so in order to capture all the packets coming from the two entities and display to the HMI a configuration that isn’t the real one, but one we synthesized.

This is a normal ARP Request/Response exchange:

```
    22 12:40:57,2763…  6.9410… 02:42:ac:11:00:02 Broadcast         ARP          42    Who has 172.17.0.5? Tell 172.17.0.2
    23 12:40:57,2764…  0.0001… 02:42:ac:11:00:05 02:42:ac:11:00:02 ARP          42    172.17.0.5 is at 02:42:ac:11:00:05
```

Here below you can see the flooding process, that lasts until the attack termination:

```
22041 12:43:06,6950…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
22356 12:43:08,6962…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
22696 12:43:10,6968…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
22725 12:43:10,8794…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.6 is at 02:42:ac:11:00:06
23000 12:43:12,6978…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
23330 12:43:14,6982…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
23665 12:43:16,7015…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
24003 12:43:18,7024…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
24362 12:43:20,7071…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
24689 12:43:22,7076…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
25035 12:43:24,7078…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
25369 12:43:26,7084…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
25719 12:43:28,7089…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
26043 12:43:30,7096…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
26394 12:43:32,7108…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
26732 12:43:34,7115…  0.0… 02:42:ac:11:00:06  02:42:ac:11:00:02  ARP  42  172.17.0.5 is at 02:42:ac:11:00:06
```

Here below we show the Arp tables before, during and after the attack (correctly updated):

```
root@f12f3193cc24:/home/selenium# arp -a
? (172.17.0.1) at 02:42:e0:93:11:24 [ether] on eth0
? (172.17.0.3) at 02:42:ac:11:00:03 [ether] on eth0
? (172.17.0.4) at 02:42:ac:11:00:04 [ether] on eth0
? (172.17.0.5) at 02:42:ac:11:00:05 [ether] on eth0
```

```
root@f12f3193cc24:/home/selenium# arp -a
? (172.17.0.1) at 02:42:e0:93:11:24 [ether] on eth0
? (172.17.0.3) at 02:42:ac:11:00:03 [ether] on eth0
? (172.17.0.4) at 02:42:ac:11:00:04 [ether] on eth0
? (172.17.0.5) at 02:42:ac:11:00:06 [ether] on eth0
? (172.17.0.6) at 02:42:ac:11:00:06 [ether] on eth0
```

```
root@f12f3193cc24:/home/selenium# arp -a
? (172.17.0.1) at 02:42:e0:93:11:24 [ether] on eth0
? (172.17.0.3) at 02:42:ac:11:00:03 [ether] on eth0
? (172.17.0.4) at 02:42:ac:11:00:04 [ether] on eth0
? (172.17.0.5) at 02:42:ac:11:00:05 [ether] on eth0
? (172.17.0.6) at 02:42:ac:11:00:06 [ether] on eth0
```

After the first phase of “Arp Spoofing” we proceed by exposing a Modbus Server on the port 502. This Server will serve requests from HMI, getting the information from the “.json” capture files, and so, masquerading the PLC1.

Here below you can see a piece of the conversation between the HMI and the MITM device.

```
22114 12:43:07,2550…  0.0006… 172.17.0.2  172.17.0.5  Modbus/TCP  78  Query: Trans:  2422; Unit:  1, Func:  4: Read Input Registers
22116 12:43:07,2552…  0.0001… 172.17.0.5  172.17.0.2  Modbus/TCP  77  Response: Trans:  2422; Unit:  1, Func:  4: Read Input Registers
```

Detail of the request:

```
> Frame 22114: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface docker0, id 0
v Ethernet II, Src: 02:42:ac:11:00:02 (02:42:ac:11:00:02), Dst: 02:42:ac:11:00:06 (02:42:ac:11:00:06)
  > Destination: 02:42:ac:11:00:06 (02:42:ac:11:00:06)
  > Source: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.17.0.2, Dst: 172.17.0.5
> Transmission Control Protocol, Src Port: 53146, Dst Port: 502, Seq: 1, Ack: 1, Len: 12
> Modbus/TCP
v Modbus
    .000 0100 = Function Code: Read Input Registers (4)
    Reference Number: 0
    Word Count: 1
```

Detail of the response:

```
> Frame 22116: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface docker0, id 0
v Ethernet II, Src: 02:42:ac:11:00:06 (02:42:ac:11:00:06), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
  > Destination: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
  > Source: 02:42:ac:11:00:06 (02:42:ac:11:00:06)
    Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.17.0.5, Dst: 172.17.0.2
> Transmission Control Protocol, Src Port: 502, Dst Port: 53146, Seq: 1, Ack: 13, Len: 11
> Modbus/TCP
v Modbus
    .000 0100 = Function Code: Read Input Registers (4)
    [Request Frame: 22114]
    [Time from request: 0.000184068 seconds]
    Byte Count: 2
  > Register 0 (UINT16): 60
```

Once the attack has terminated, the canonical behavior is restored.

```
52050 12:46:05,7542…  -0.000… 172.17.0.2   172.17.0.5   Modbus/TCP   78   Query: Trans:  5678; Unit:   1, Func:   4: Read Input Registers
52053 12:46:05,7545…  0.0000… 172.17.0.5   172.17.0.2   Modbus/TCP   77   Response: Trans:  5678; Unit:   1, Func:   4: Read Input Registers
```

Detail of the request:

```
> Frame 52050: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface docker0, id 0
v Ethernet II, Src: 02:42:ac:11:00:02 (02:42:ac:11:00:02), Dst: 02:42:ac:11:00:05 (02:42:ac:11:00:05)
    > Destination: 02:42:ac:11:00:05 (02:42:ac:11:00:05)
    > Source: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
      Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.17.0.2, Dst: 172.17.0.5
> Transmission Control Protocol, Src Port: 41442, Dst Port: 502, Seq: 193, Ack: 169, Len: 12
> Modbus/TCP
v Modbus
      .000 0100 = Function Code: Read Input Registers (4)
      Reference Number: 0
      Word Count: 1
```

Detail of the response:

```
> Frame 52053: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface docker0, id 0
v Ethernet II, Src: 02:42:ac:11:00:05 (02:42:ac:11:00:05), Dst: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
    > Destination: 02:42:ac:11:00:02 (02:42:ac:11:00:02)
    > Source: 02:42:ac:11:00:05 (02:42:ac:11:00:05)
      Type: IPv4 (0x0800)
> Internet Protocol Version 4, Src: 172.17.0.5, Dst: 172.17.0.2
> Transmission Control Protocol, Src Port: 502, Dst Port: 41442, Seq: 169, Ack: 205, Len: 11
> Modbus/TCP
v Modbus
      .000 0100 = Function Code: Read Input Registers (4)
      [Request Frame: 52050]
      [Time from request: 0.000242262 seconds]
      Byte Count: 2
    > Register 0 (UINT16): 70
```

In conclusion, we can assert that, in the described environment, we managed to obfuscate a device and so, thanks to all the evidence collected, we can indeed prove that the attack ran effectively.