# Deep dive into an ICS Firewall

Looking for the Fire-Hole

Julien Lenoir & Benoît Camredon

Black Hat USA 2018

August 8, 2018

# Who are we?

**Julien Lenoir**

- Security evaluator
- Both embedded systems and IT evaluation
- Embedded system security:
  - reverse engineering
  - vulnerability research
- julien.lenoir (at) airbus.com

**Benoît Camredon**

- Security evaluator
- Mainly embedded systems audit/pentest
- Mainly interested in
  - Linux-related stuff
  - Network-related stuff
- benoit.camredon (at) airbus.com

# Introduction

## This presentation is about

- Evaluation of a firewall dedicated to industrial environments
- Called Tofino Xenon manufactured by Belden

## Important notes

- All vulnerabilities were responsibly disclosed to the vendor
- Fixed in firmware 03.2.00 released in November 2017
- Some vulnerabilities were assigned CVE numbers[1]
- Amount of work: 50 man-days split between two evaluators

[1] CVE-2017-11400, CVE-2017-11401, CVE-2017-11402

# Introduction

## Presentation outline

- Threats and constraints of Industrial Control Systems
- Tofino Xenon presentation
- Evaluation objectives
- Preliminary work
- Evaluation results
- Conclusion

# What do we mean by ICS systems?

### Definition

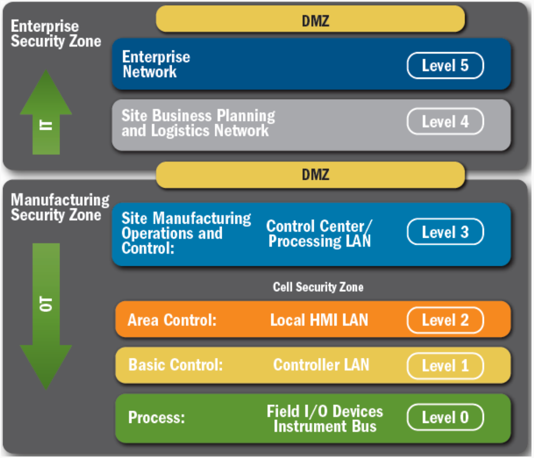
- Control systems and associated instrumentation used for industrial process control

### Many types of components

- SCADA
- Network devices
- PLCs

### Tofino Xenon firewall deployed

- Between level 2 and level 1 of Purdue model



Enterprise Security Zone
DMZ
IT
Enterprise Network
Level 5
Site Business Planning and Logistics Network
Level 4
DMZ
Manufacturing Security Zone
OT
Site Manufacturing Operations and Control:
Control Center/ Processing LAN
Level 3
Cell Security Zone
Area Control:
Local HMI LAN
Level 2
Basic Control:
Controller LAN
Level 1
Process:
Field I/O Devices Instrument Bus
Level 0

**Figure 1:** Overview of Purdue model[2]

[2] Source: Recommended Practice: defense-in-depth, 2016, NCCIC ICS-CERT

# ICS system vulnerabilities in 2017[3]

**A few figures:**

- 322 vulnerabilities identified in 2017
- 194 with CVSS score higher than 7
- Exploits published for 17 of them

**Top 4 sectors affected**

- Energy
- Critical manufacturing
- Water and wastewater systems
- Transportation systems

Impacted components: SCADA, network devices, PLCs, etc.

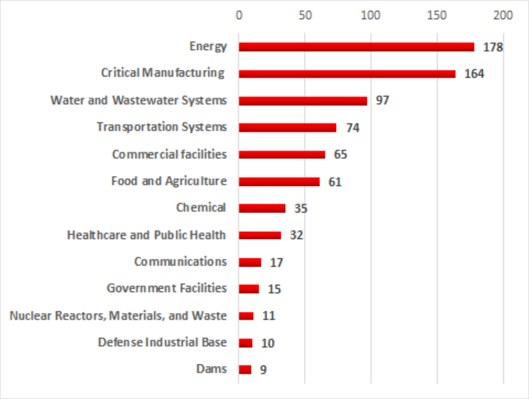


**Figure 2:** Number of vulnerable products used in different industries [3]

[3] Source: Kaspersky Lab ICS CERT

# Introduction

### Downtime or malfunction impacts

- Potential safety impacts if unplanned
- High downtime costs

### Dilemma

- Upgrades are hard
  - Some are old and not supported anymore
  - Most systems can be stopped only on very rare occasions
- Threats: real world attacks **do** exist[4]

### Dedicated firewalls

- In front of ICS
- Deep packet inspection of industrial protocols
- Only *valid* packets reach the ICS

[4] TRITON: How it Disrupted Safety Systems and Changed the Threat Landscape of Industrial Control Systems, Forever

# Goal and approach

### Objective: evaluate the firewall security level

- How effective is it to protect the assets?
- Is the firewall going to introduce new vulnerabilities in the network?

### Our constraints

- No physical tampering with the equipment ⇒ Software-only attacks
- Inputs: firmware updates, user guides, the appliance itself

### Our approach

- List all features and interfaces of the firewall
- Consider ANSSI[5] protection profiles (Security objectives, Threats, Attacker model)
- Offensive security evaluation of relevant features (reverse engineering, vulnerability research)

[5] French Network and Information Security Agency

# Tofino Xenon presentation

# Tofino Xenon

### Firewall function

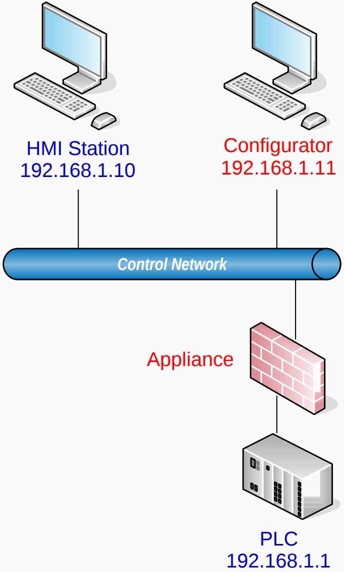
- Placed between protected assets and control network
- Logically transparent (no IP address)
- Filter classic protocols: TCP, UDP...
- Filter industrial protocols: ModBus, EtherNet/IP, OPC

### Two operating modes

- Test: logs only
- Operational: packets dropped

### Two components

- Physical: appliance
- Software: configurator



# Hardware part: Tofino Security Appliance (SA)

### From the outside

- Industrial form factor (DIN mount)
- Hardened case (heat, dust)
- Two Ethernet ports: open world and secure world
- One USB port: upgrade, configuration, logs export

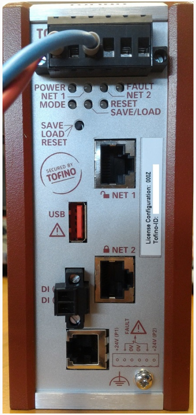
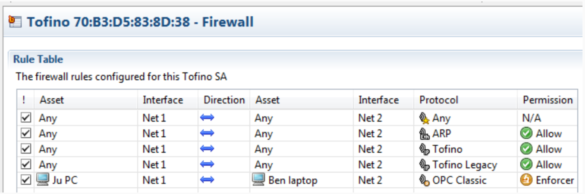


**Figure 3:** Appliance

# Software part: Configurator

**Only official way of configuring the appliance**

- Interface to manage one or more appliances
- Configure the firewall
  - Set all firewall rules
  - Customize DPI level
- Retrieve logs
- Two ways of applying configuration
  - USB: generate an encrypted configuration
  - Network: custom encrypted communication



**Tofino 70:B3:D5:83:8D:38 - Firewall**

**Rule Table**

The firewall rules configured for this Tofino SA

| ! | Asset | Interface | Direction | Asset | Interface | Protocol | Permission |
|---|---|---|---|---|---|---|---|
| ☑ | Any | Net 1 | ⇔ | Any | Net 2 | Any | N/A |
| ☑ | Any | Net 1 | ⇔ | Any | Net 2 | ARP | ✓ Allow |
| ☑ | Any | Net 1 | ⇔ | Any | Net 2 | Tofino | ✓ Allow |
| ☑ | Any | Net 1 | ⇔ | Any | Net 2 | Tofino Legacy | ✓ Allow |
| ☑ | Ju PC | Net 1 | ⇔ | Ben laptop | Net 2 | OPC Classic | Enforcer |

**Figure 4:** Configurator window

# Evaluation objectives

# ANSSI ICS firewall protection profiles [6]

### Assumptions

- Premises: equipment is not necessarily deployed in secure location
- Dimensioning: equipment is properly dimensioned for its tasks
- Administrators: trained and trustworthy
- Attacker: can purchase the device to look for vulnerabilities

### Attackers

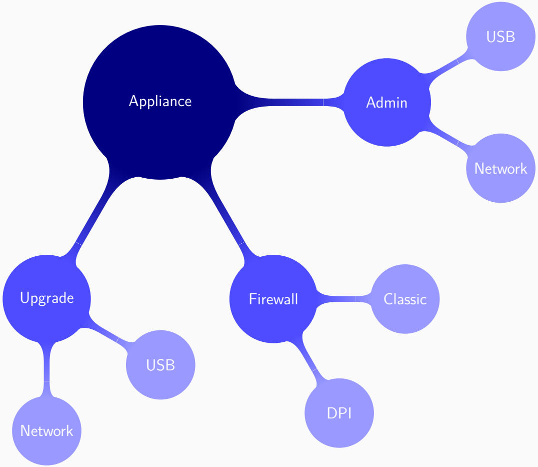
- Can plug a device to any port of the equipment (*e.g.* USB stick)
- Can be connected to the administration network
- Cannot physically take the device apart

[6] `https://www.ssi.gouv.fr/entreprise/guide/profils-de-protection-pour-les-systemes-industriels/`

# ANSSI firewall protection profiles

## Security objectives

1. Firewall
   - Policy enforcement
   - Protocol conformity analysis
2. Admin
   - Authentication
   - Access Control
3. Upgrade
   - Firmware signature



**Threats are defined as violation of these security objectives**

# Preliminary work

# Appliance: black box approach

### First try

- Analyze the firmware
- Available but fully encrypted

### Configuration

- Over USB: fully encrypted
- Over the network: custom encrypted protocol

### Reversing protocol steps

- Wireshark to discover the logic and packet formats
- Figure out how cryptography is used
- Reverse engineer the configurator to re-implement the protocol

# Tofino custom protocol

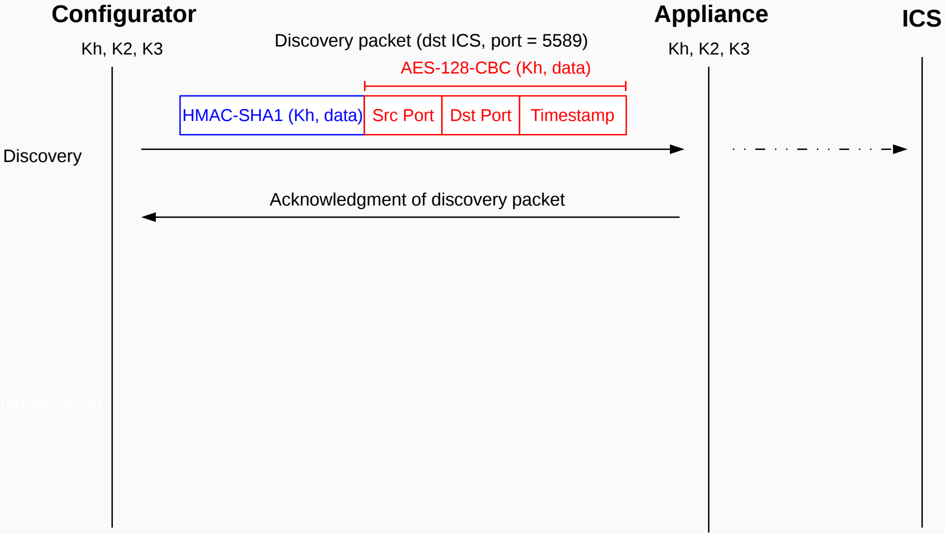
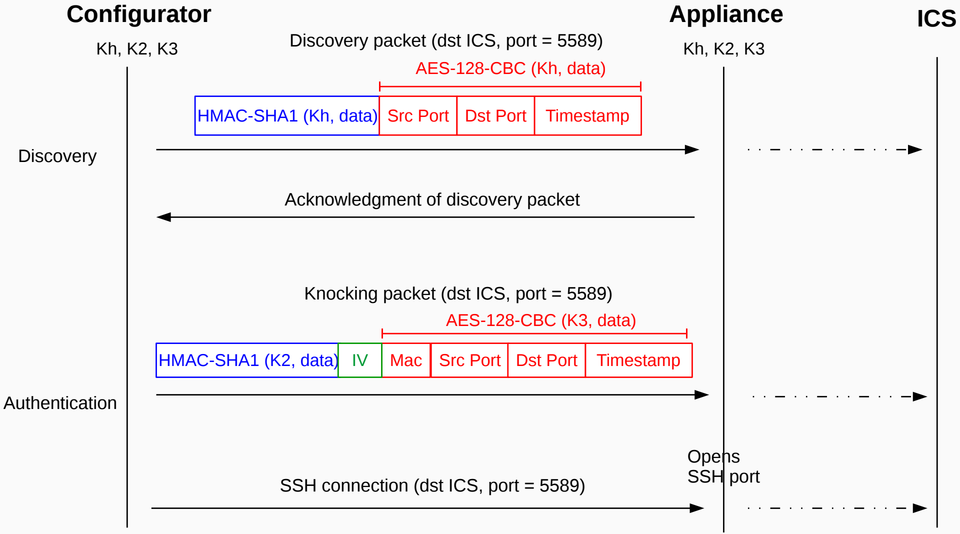


**Figure 5:** *Discovery and authentication protocols*

# Tofino custom protocol



Configurator

Kh, K2, K3

Appliance

Kh, K2, K3

ICS

Discovery

Discovery packet (dst ICS, port = 5589)

AES-128-CBC (Kh, data)

HMAC-SHA1 (Kh, data) | Src Port | Dst Port | Timestamp

Acknowledgment of discovery packet

Authentication

Knocking packet (dst ICS, port = 5589)

AES-128-CBC (K3, data)

HMAC-SHA1 (K2, data) | IV | Mac | Src Port | Dst Port | Timestamp

Opens
SSH port

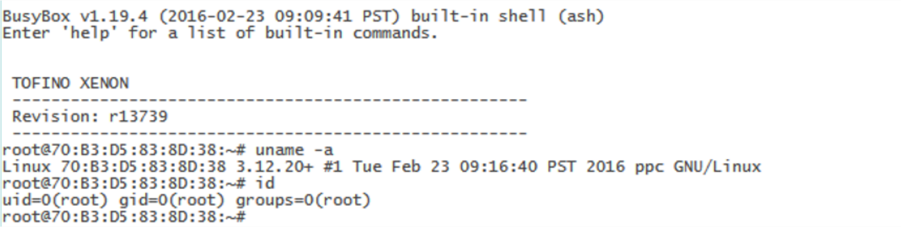
SSH connection (dst ICS, port = 5589)

**Figure 5:** *Discovery and authentication protocols*

# Shell access on appliance

## What we did

- Extract default keys (`K2`, `K3` and `ssh-rsa`) from the client
- Implement our own client for authentication



```
BusyBox v1.19.4 (2016-02-23 09:09:41 PST) built-in shell (ash)
Enter 'help' for a list of built-in commands.


 TOFINO XENON
 -----------------------------------------------------
 Revision: r13739
 -----------------------------------------------------
root@70:B3:D5:83:8D:38:~# uname -a
Linux 70:B3:D5:83:8D:38 3.12.20+ #1 Tue Feb 23 09:16:40 PST 2016 ppc GNU/Linux
root@70:B3:D5:83:8D:38:~# id
uid=0(root) gid=0(root) groups=0(root)
root@70:B3:D5:83:8D:38:~#
```

**Figure 6:** *No shell no game!*

## Results

- *root* shell with SSH on the appliance! Everything is running as root...

# Shell access on appliance

### Not a black box anymore

- Linux operating system
- PowerPC architecture

### Access to

- The whole file system content
- Internal configuration
- `iptables` rules

### We are able to

- Reverse engineer appliance binaries
- Do live debugging (gdb)

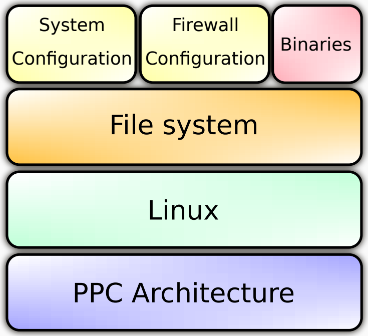


**Figure 7:** Tofino internals

# Evaluation

# Association with Tofino Appliance

### Association phase

- Configurator: generates new K2/K3
- Appliance: receives the new K2/K3

### Once association is done

- K2/K3 knowledge necessary for configuration
- Kh sufficient for discovery

### Security objective

- Network authentication objective is met

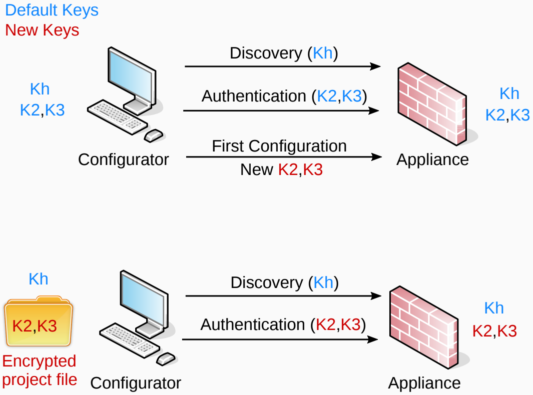


**Figure 8:** Association phase

# Firmware signature

### Two upgrade paths

- Network, using the configurator
- USB stick plugged into the equipment

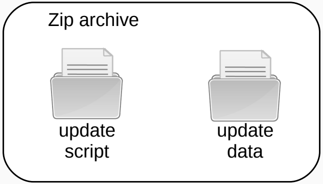
### USB update contains

- Update script
- Update data: kernel, file system

### Both encrypted

- AES-128-CBC, hard-coded keys

**Only the update script is signed (RSA 3072)!**



# Firmware signature

### Two upgrade paths

- Network, using the configurator
- USB stick plugged into the equipment

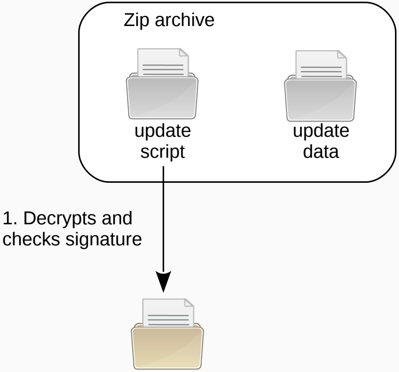
### USB update contains

- Update script
- Update data: kernel, file system

### Both encrypted

- AES-128-CBC, hard-coded keys

**Only the update script is signed (RSA 3072)!**



# Firmware signature

### Two upgrade paths

- Network, using the configurator
- USB stick plugged into the equipment

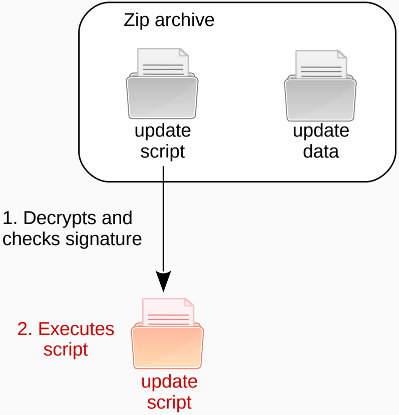
### USB update contains

- Update script
- Update data: kernel, file system

### Both encrypted

- AES-128-CBC, hard-coded keys

**Only the update script is signed (RSA 3072)!**



# Firmware signature

### Two upgrade paths

- Network, using the configurator
- USB stick plugged into the equipment

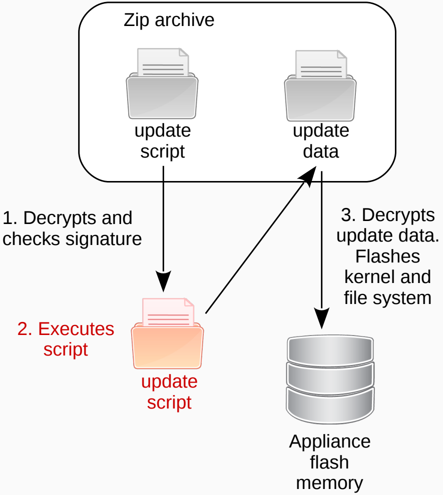
### USB update contains

- Update script
- Update data: kernel, file system

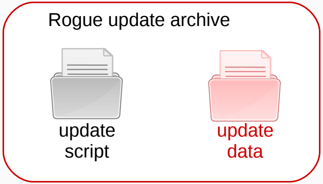
### Both encrypted

- AES-128-CBC, hard-coded keys

**Only the update script is signed (RSA 3072)!**



# USB firmware signature bypass



**An attacker can**

- Obtain an update file
- Get access to update crypto keys
- Generate a rogue update archive

**Rogue update archive**

- Genuine update script
- Modified update data: backdoored kernel

# USB firmware signature bypass

## Vulnerability

- Attack vector: physical access to USB Port
- Impact on appliance: full compromise

## Security objectives

- Firmware signature objective is **not met**

## Status

- Assigned CVE-2017-11400

# Firewall internals

### Firewall

- Classic filtering (TCP/UDP/IP) done by `netfilter`
- Three protocols analyzed at layer 7
  - EtherNet/IP (port 44818)
  - ModBus (port 502)
  - OPC Classic (port 135)
- Packets are sent to userland modules
- Modules then
  - Proceed to deep packet inspection
  - Decide whether packets are authorized or not

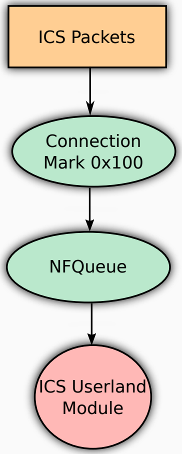
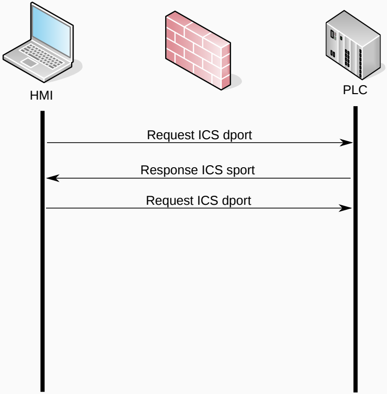


**Figure 9:** Tofino firewall internals

# Enforcing DPI rules

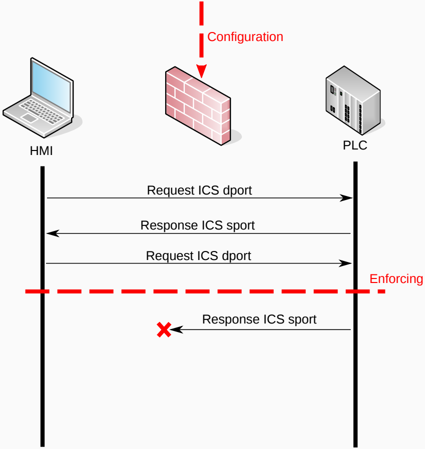
## What happens to already established communications?



# Enforcing DPI rules

## What happens to already established communications?

- Response sent would be dropped because connection is not tracked yet
  => **Enforcing DPI rule could break established connections**



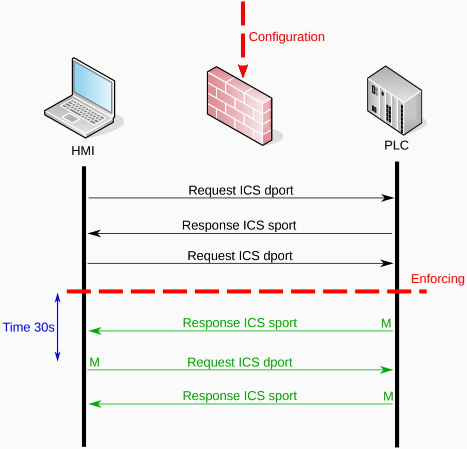
# Enforcing DPI rules

### What happens to already established communications?

- Response sent would be dropped because connection is not tracked yet
  => **Enforcing DPI rule could break established connections**

### Tofino Workaround

- Allow responses for 30s (matching supported ICS source port)
  - Custom kernel module to manage timeout



# Enforcing DPI rules

### What happens to already established communications?

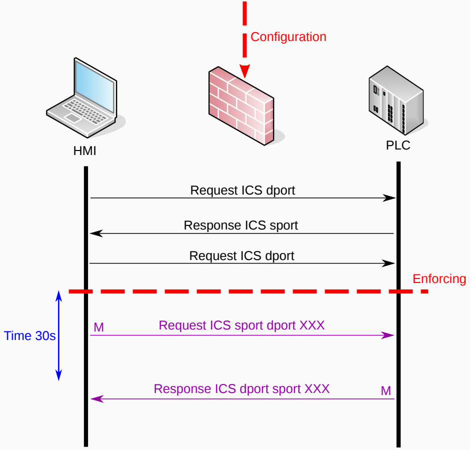
- Response sent would be dropped because connection is not tracked yet
  => **Enforcing DPI rule could break established connections**

### Tofino Workaround

- Allow responses for 30s (matching supported ICS source port)
  - Custom kernel module to manage timeout

### Timed rule vulnerability

- Window of opportunity for bypassing firewall
- High impact but low potentiality
- DPI checks still enforced



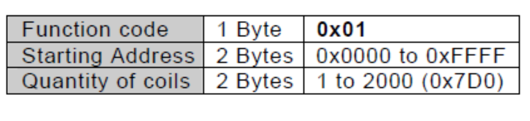
# DPI modules: purpose

**Two levels of analysis**

1. Packet format
   - Discard malformed packets
   - Avoid parsing error on ICS systems
2. Content filtering
   - ModBus, EtherNet/IP: master/slave model
   - Restrict to a set of allowed commands

# DPI modules: packet sanitization example

## `Read Coils` (function code 1) example



| | | |
|---|---|---|
| Function code | 1 Byte | **0x01** |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of coils | 2 Bytes | 1 to 2000 (0x7D0) |

**Figure 10:** Modbus 1.1b specification

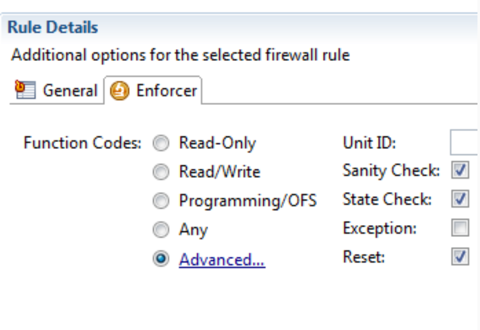
## Actual DPI module implementation

```
if (start_addr + qty - 1 > 0xFFFF)
{
  return ERROR_OVERFLOW_QUANTITY;
}
if (qty > 0x7D0)
{
  return ERROR_ABOVE_SPEC_MAXIMUM;
}
```

# DPI modules: content filtering

**Restrict allowed commands**

- Profile based
- Ability to limit to:
  - Read
  - Read/Write
  - ...



Rule Details

Additional options for the selected firewall rule

Function Codes: ○ Read-Only ○ Read/Write ○ Programming/OFS ○ Any ● Advanced...

Unit ID:

Sanity Check: ☑

State Check: ☑

Exception: □

Reset: ☑

**Figure 11:** Filter customization panel for MODBUS

# EtherNet/IP[7] filter

## EtherNet/IP protocol

- Over TCP and UDP
- Encapsulates Common Industrial Protocol (CIP) messages

## Bug found

- Parsing error in `ListInterfaces` (code 0x64) **reply**
- Out-of-bound write
- Unable to exploit it for code execution
- Exploited to bypass sanity checks of replies

## Status

- Reported and patched
- No CVE assigned

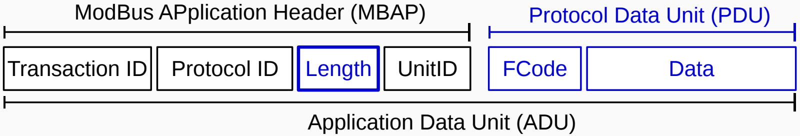
[7] Ethernet/Industrial Protocol

# ModBus filter: packet format

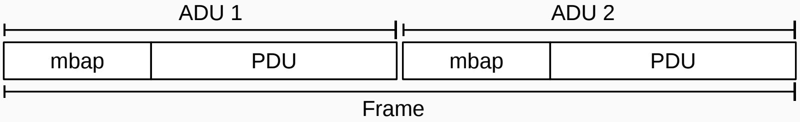
## ModBus protocol

- Over Ethernet (UDP/TCP) but also serial link
- One Application Data Unit (ADU) per command (function code)

## ADU format



## Multiple ADU supported by Tofino, even though not defined in the standard



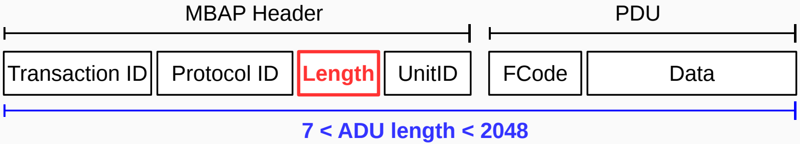
# ModBus filter: ADU check

## Checks on each ADU

1. Function code is authorized
2. Sanity checks
    - ADU size within standard boundaries
    - Sanity checks on PDU data

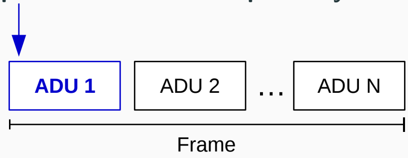
## Problem

- Length field of MBAP header not compared to maximum allowed by the standard



# ModBus filter: multiple ADU

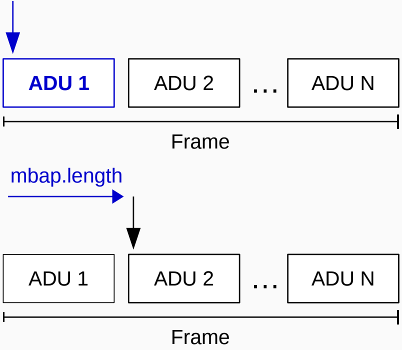
Multiple ADU’s checked sequentially



```
 while (1)
 {
   if( !ADU_check() )
     return FRAME_INVALID;

   if ((frame_size - mbap.length) < 7)}
     break;

   move_to_next_ADU(mbap.length);
 }

if ((frame_size - mbap.length) <= 0)
  return FRAME_VALID;
```

# ModBus filter: multiple ADU

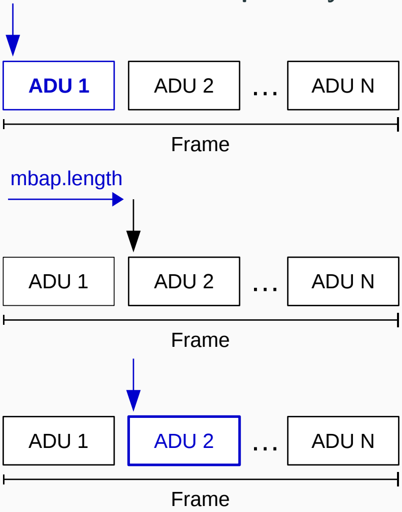
## Multiple ADU's checked sequentially



```
while (1)
{
  if( !ADU_check() )
    return FRAME_INVALID;

  if ((frame_size - mbap.length) < 7)}
    break;

  move_to_next_ADU(mbap.length);
}

if ((frame_size - mbap.length) <= 0)
  return FRAME_VALID;
```

# ModBus filter: multiple ADU

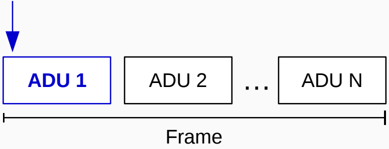
## Multiple ADU's checked sequentially



```
while (1)
{
  if( !ADU_check() )
    return FRAME_INVALID;

  if ((frame_size - mbap.length) < 7)}
    break;

  move_to_next_ADU(mbap.length);
}

if ((frame_size - mbap.length) <= 0)
  return FRAME_VALID;
```

# ModBus filter: multiple ADU

## In case of invalid length

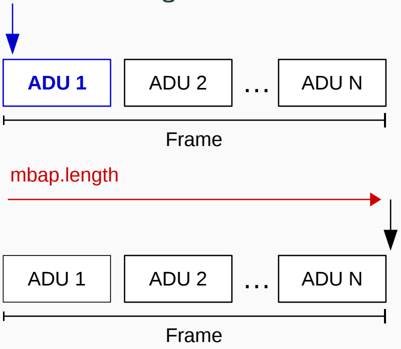


```
 while (1)
 {
   if( !ADU_check() )
    return FRAME_INVALID;

   if ((frame_size - mbap.length) < 7)}
    break;

   move_to_next_ADU(mbap.length);
 }

if ((frame_size - mbap.length) <= 0)
  return FRAME_VALID;
```

# ModBus filter: multiple ADU

AIRBUS

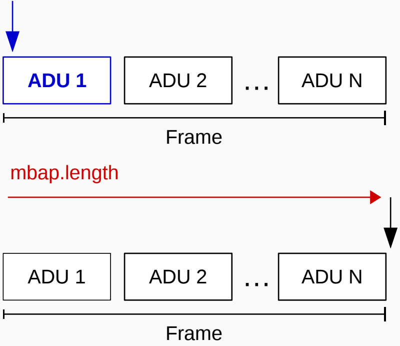
## In case of invalid length



```
while (1)
{
  if( !ADU_check() )
    return FRAME_INVALID;

  if ((frame_size - mbap.length) < 7)
    break;

  move_to_next_ADU(mbap.length);
}

if ((frame_size - mbap.length) <= 0)
  return FRAME_VALID;
```

# ModBus filter: multiple ADU

**In case of invalid length**



```
while (1)
{
  if( !ADU_check() )
    return FRAME_INVALID;

  if ((frame_size - mbap.length) < 7)
    break;

  move_to_next_ADU(mbap.length);
}

if ((frame_size - mbap.length) <= 0)
 return FRAME_VALID;
```

**Exit loop**

- After first ADU check
- Frame is considered valid

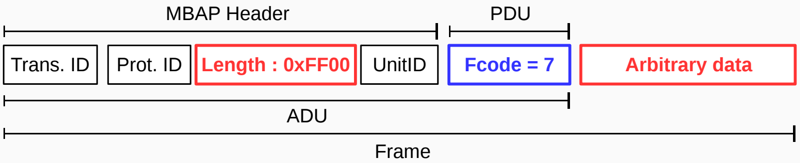
# ModBus filter

## Consequences

- With invalid `mbap.length`:
  - First ADU is inspected then loop exited
  - Arbitrary data in frame tail unchecked!

## Choosing the first ADU

- Function code 7: described as serial line only in the standard
- One-byte PDU: no data, only the function code
- In all pre-defined Fcode white lists (read, read/write, programming)



# ModBus filter

### Filter bypass

- Works in almost every configuration, unless FC 7 (**serial line only**) is blacklisted
- Potential impacts:
  - Crash with parsing errors
  - Unchecked ADU's in frame tail, un-allowed function codes

### Security objectives

- ModBus protocol conformity analysis objective is **not met**

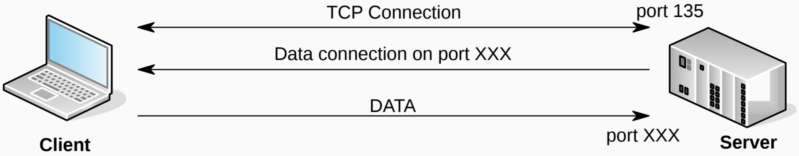
### Status

- Assigned CVE-2017-11401

# OPC Classic filter

## Protocol

- Use TCP Port 135
- Based on Microsoft COM/DCOM technology
- Dynamic ports
- Bi-directional protocol: connections can be initiated in both directions



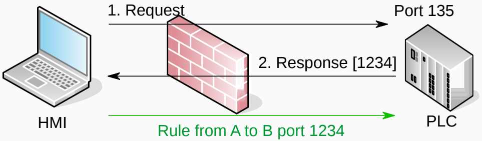
## Hard to filter with a classic firewall

- A dynamic TCP port is negotiated by the application layer
- Module needed to track connection (like FTP)

# OPC module flaws: packet direction

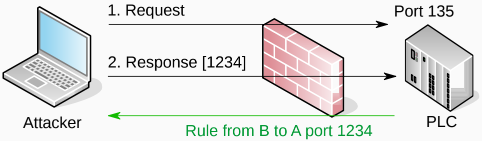
### Behavior

- A netfilter rule is dynamically added by the OPC module when a new communication port is specified instead of using connection tracker



### Problem

- No verification of the state machine...
- Ability to create a netfilter rule from the *PLC* to the *Attacker* on a chosen port
- Low impact



# OPC module flaws: small packets

### OPC Message Structure

- OPC packet is composed of a 19-byte header

### Logical flaw

- Small packets are not filtered
- TCP fragmentation attack
  - Fragments will go through the filter
  - Reassembled by the target
- OPC module can be bypassed

```
if(TCP) {
  rvalue = 1;
  if(size(TCP_DATA) > 19) {
    rvalue = parse_opc(TCP_DATA);
  }
} else {
  rvalue = -1;
}
return rvalue;
```

### Security objectives

- OPC protocol conformity is **not met**

# From findings to complete firewall bypass...

# Firewall complete bypass

### Combined findings summary

1. Timed rule vulnerability: firewall bypass for 30s
2. Arbitrary firewall rule injection (if OPC filter is enabled)
3. Bypass sanity checks and payload filtering using fragmentation (if OPC filter is enabled)

# Firewall complete bypass

## Combined findings summary

1. Timed rule vulnerability: firewall bypass for 30s
2. Arbitrary firewall rule injection (if OPC filter is enabled)
3. Bypass sanity checks and payload filtering using fragmentation (if OPC filter is enabled)

## Attack scenario

- Attacker triggers a rule creation (finding 2)
  - `netfilter` behavior: When a rule is inserted all rules are reapplied...
  - Resets the *timedrule* timer

# Firewall complete bypass

## Combined findings summary

1. Timed rule vulnerability: firewall bypass for 30s
2. Arbitrary firewall rule injection (if OPC filter is enabled)
3. Bypass sanity checks and payload filtering using fragmentation (if OPC filter is enabled)

## Attack scenario

- Attacker triggers a rule creation (finding 2)
  - `netfilter` behavior: When a rule is inserted all rules are reapplied...
  - Resets the *timedrule* timer
- For 30s, attackers can reach any target port (finding 1)

# Firewall complete bypass

## Combined findings summary

1. Timed rule vulnerability: firewall bypass for 30s
2. Arbitrary firewall rule injection (if OPC filter is enabled)
3. Bypass sanity checks and payload filtering using fragmentation (if OPC filter is enabled)

## Attack scenario

- Attacker triggers a rule creation (finding 2)
  - `netfilter` behavior: When a rule is inserted all rules are reapplied...
  - Resets the *timedrule* timer
- For 30s, attackers can reach any target port (finding 1)
- By fragmenting packets, sanity checks are bypassed (finding 3)

# Firewall complete bypass

### Prerequisite

- OPC filter enabled between *Attacker* and *Target*

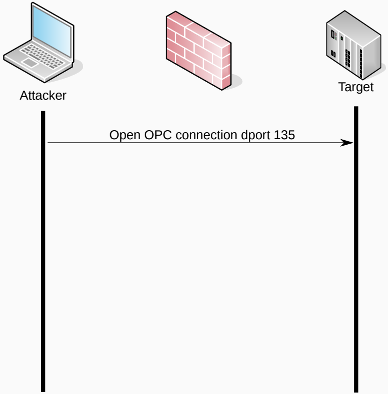
# Firewall complete bypass

## Prerequisite

- OPC filter enabled between *Attacker* and *Target*

## Steps

1. Open OPC connection from *Attacker* to *Target*



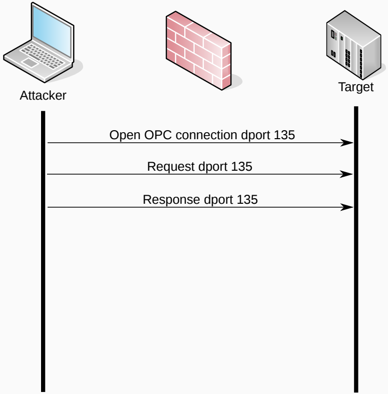
# Firewall complete bypass

## Prerequisite

- OPC filter enabled between *Attacker* and *Target*

## Steps

1. Open OPC connection from *Attacker* to *Target*
2. Send request and response



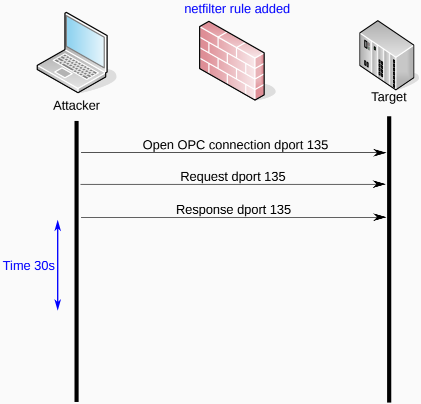
# Firewall complete bypass

## Prerequisite

- OPC filter enabled between *Attacker* and *Target*

## Steps

1. Open OPC connection from *Attacker* to *Target*
2. Send request and response
   - Rule dynamically added by the Tofino
   - Reset of timers



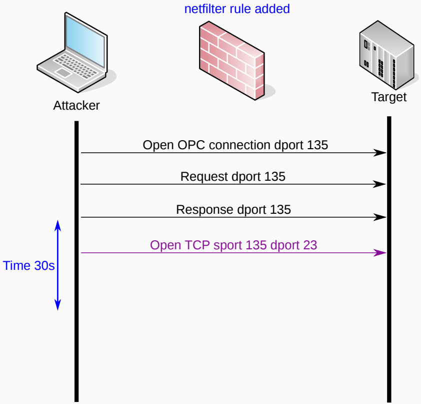
# Firewall complete bypass

## Prerequisite

- OPC filter enabled between *Attacker* and *Target*

## Steps

1. Open OPC connection from *Attacker* to *Target*
2. Send request and response
   - Rule dynamically added by the Tofino
   - Reset of timers
3. Open TCP connection to vulnerable port (with sport 135)



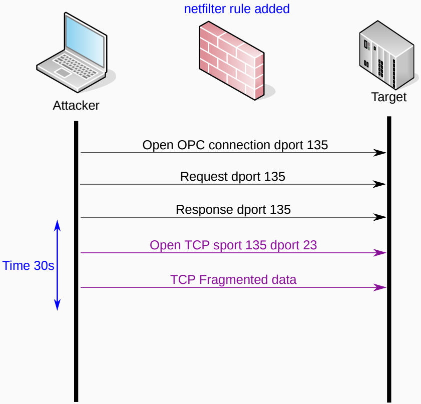
# Firewall complete bypass

## Prerequisite

- OPC filter enabled between *Attacker* and *Target*

## Steps

1. Open OPC connection from *Attacker* to *Target*
2. Send request and response
   - Rule dynamically added by the Tofino
   - Reset of timers
3. Open TCP connection to vulnerable port (with sport 135)
4. Send fragmented data (reassembled by the target)



# Firewall complete bypass

### Security objectives

- If OPC filtering is enabled, the **whole** filtering objective is **not met** (limited to devices allowed to use OPC)

### Status

- Assigned CVE-2017-11402

# Conclusion

# Conclusion

## Before trusting a security product

- Perform deep evaluation
- Follow a known and recognized referential (ANSSI's is a good example)

## In the Tofino case

- Good points:
  - Use of open-source reliable components, modular
  - Vulnerabilities are mostly implementation errors
  - Good reaction to responsible disclosure
- Room for improvement
  - Lack of hardening

## Once evaluation is done

- The production limitations are known
- Recommendation can be sent: **apply the patches**

# Takeaways

**Architecture, design, risk analysis are crucial but do not overlook implementation!**

**Deep evaluation is vital to check implementation**

**Interesting attacks are usually the result of multiple low impact bugs chained together**

# Thanks

**We want to thank**

- Our colleagues for their support
- The vendor: Belden
- **You for listening!**

Any questions?