CLAROTY

TEAM82

Team82

From Exploits to Forensic Evidence: Unraveling the Unitronics Attack

Noam Moshe

Claroty Research, Claroty Team82

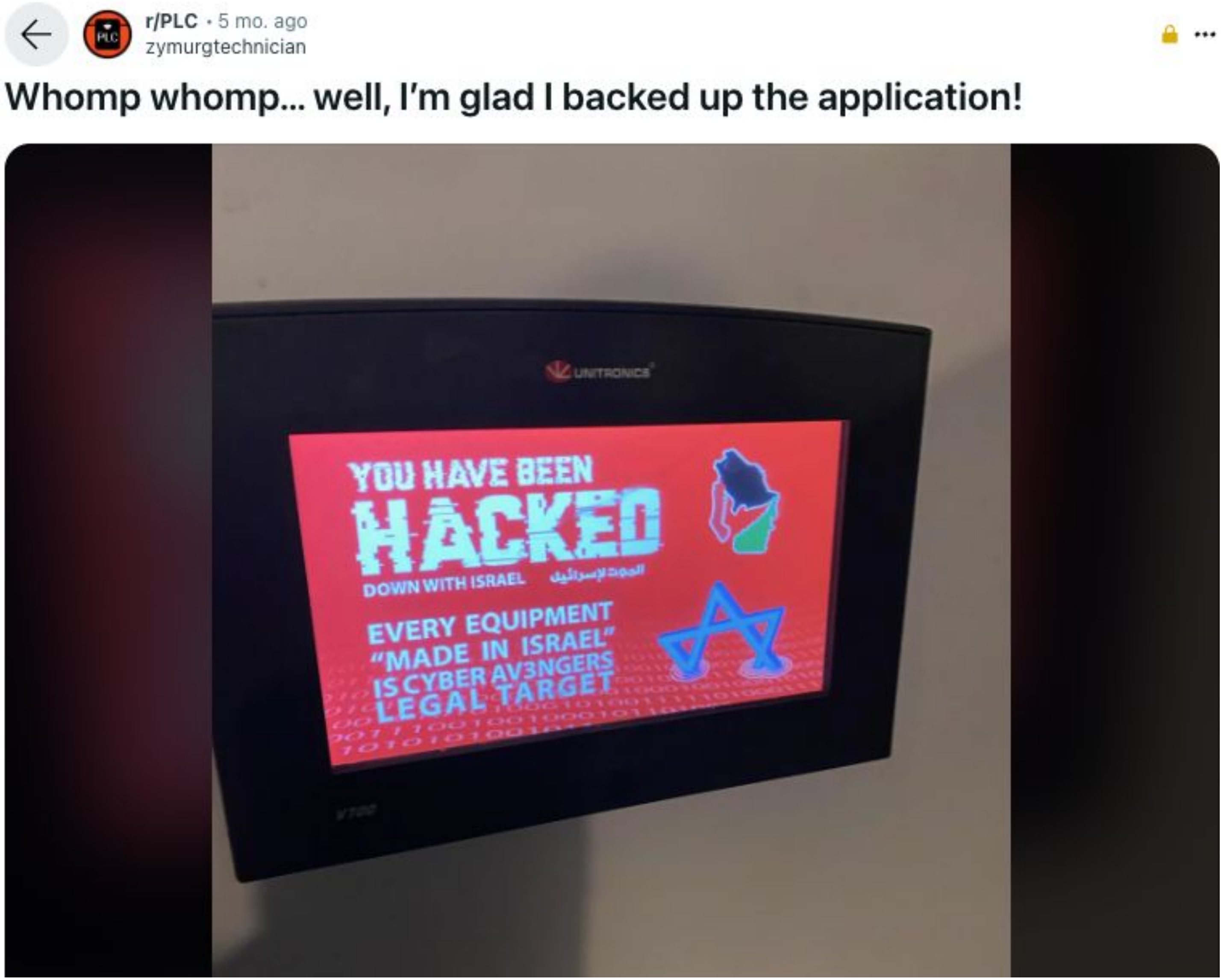
$whoami

Noam Moshe

Vulnerability researcher - mostly breaking IoT clouds. Master of Pwn @ Pwn2Own ICS 2023.

* Special thanks to Claroty Team82 researchers: Sharon Brizinov, Vera Mens, Tomer Goldschmidt

So what’s the sitch?

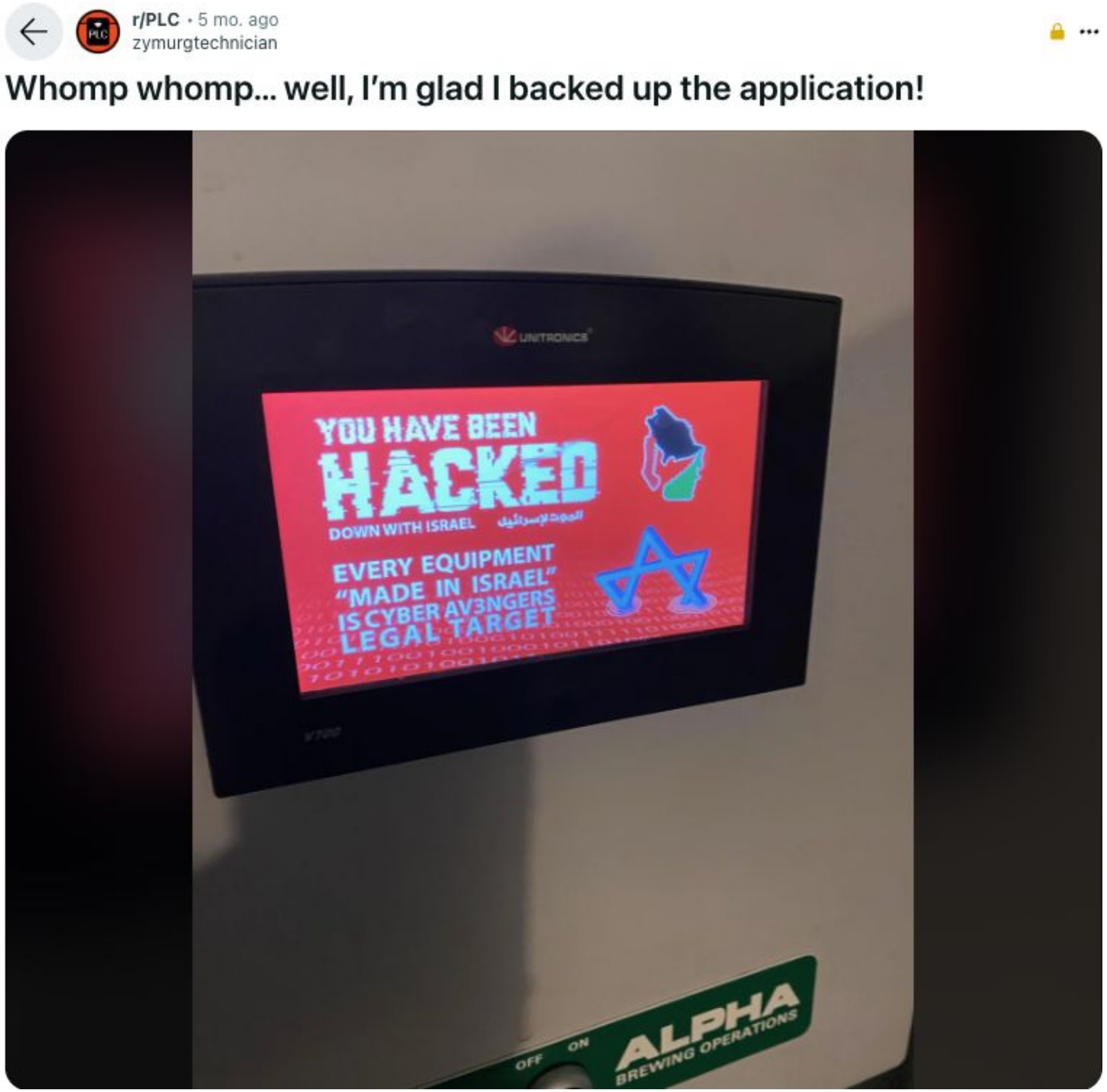


r/PLC · 5 mo. ago
zymurgtechnician

Whomp whomp... well, I’m glad I backed up the application!

So what's the sitch?

- **Nov '23**: APT targets Unitronics PLCs
  - CyberAv3ngers
- Used in water facilities worldwide

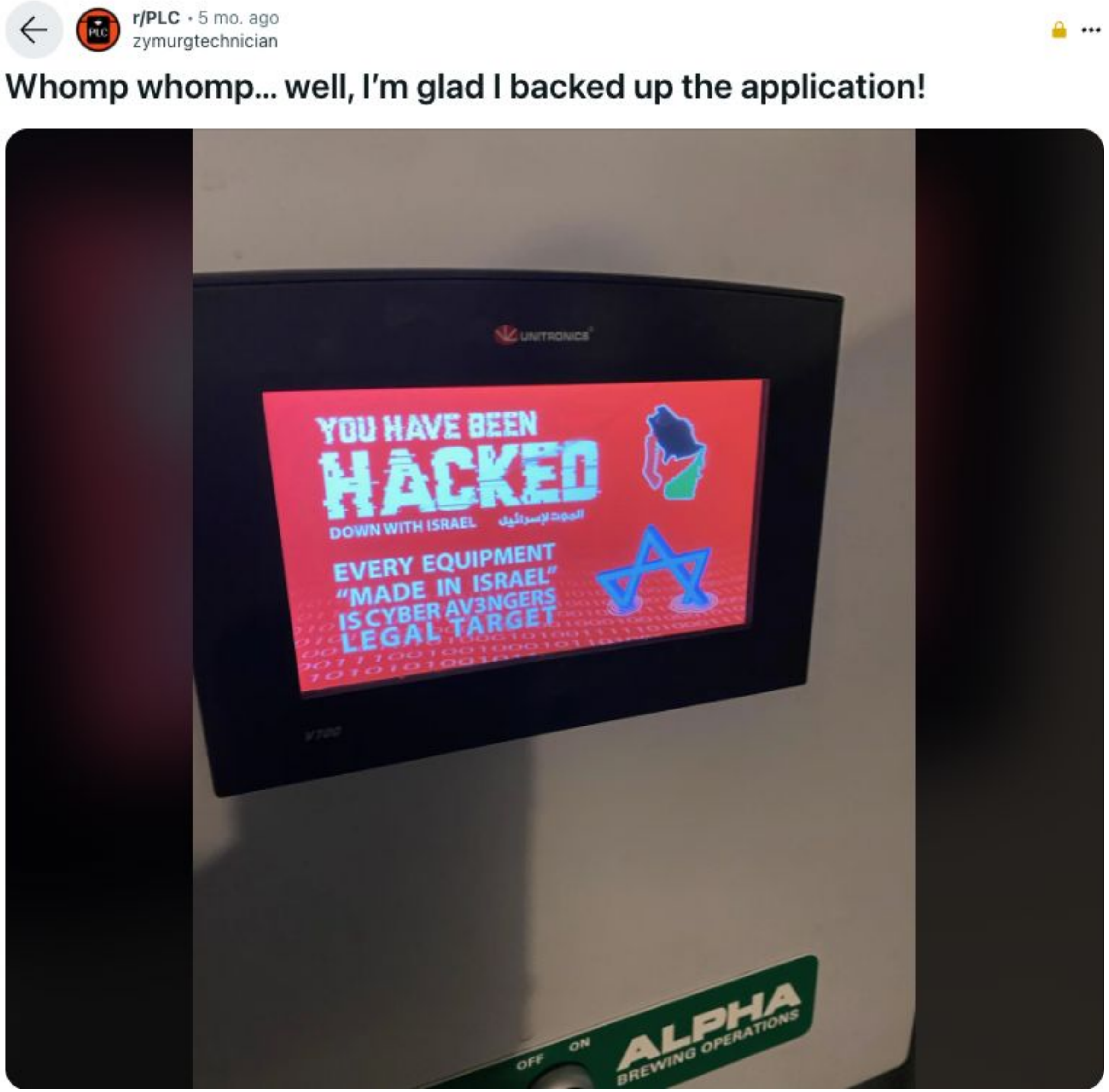

r/PLC • 5 mo. ago
zymurgtechnician

Whomp whomp... well, I'm glad I backed up the application!

So what's the sitch?

- **Nov '23**: APT targets Unitronics PLCs
  - CyberAv3ngers
- Used in water facilities worldwide

- **Why??**



r/PLC • 5 mo. ago
zymurgtechnician

Whomp whomp... well, I'm glad I backed up the application!

Fear and Panic

US sanctions Iranian officials over cyber-attacks on water plants

2 February 2024

Share

By Azadeh Moshiri, BBC News

The US has imposed sanctions on six officials
Revolutionary Guard Corps (IRGC) which it sa
attacks on American water plants late last yea

Iranian-Linked Hacks Expose Failure to Safeguard US Water System

- The EPA, lawmakers, water associations can't agree on rules
- Nation's water systems are poorly protected from cyber threats

SECURITY / TECH / POLICY

Cyberattacks are targeting US water systems, warns EPA and White House

The Municipal Water Authority of Aliquippa, PA (pictured) was targeted by a cyber attack last year. Image: AP Photo / Gene J Puskar

/ States are being asked to assess vulnerabilities at water utilities following attacks linked the Chinese and Iranian governments.

By Jess Weatherbed, a news writer focused on creative industries, compu internet culture. Jess started her career at TechRadar, covering news and hardware reviews.

Mar 20, 2024, 5:12 PM GMT+2

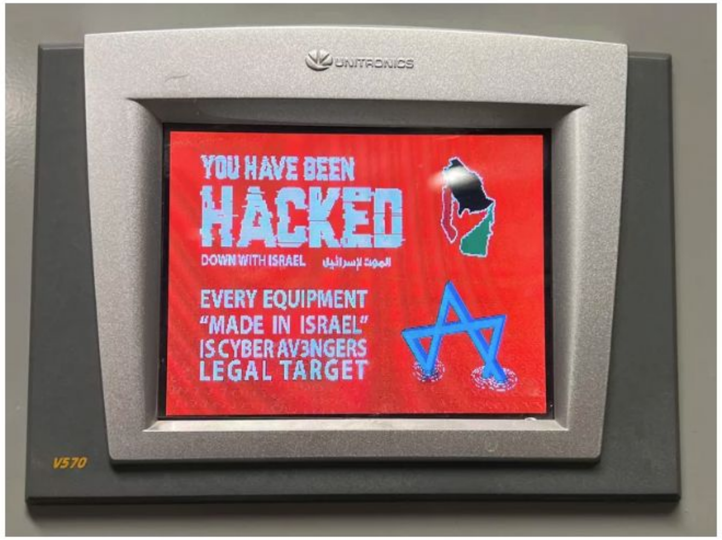
3 Comments (3 New)

TECHNOLOGY

Iran-linked cyberattacks threaten equipment used in U.S. water systems and factories

UPDATED DECEMBER 2, 2023 · 1:51 PM ET

Juliana Kim

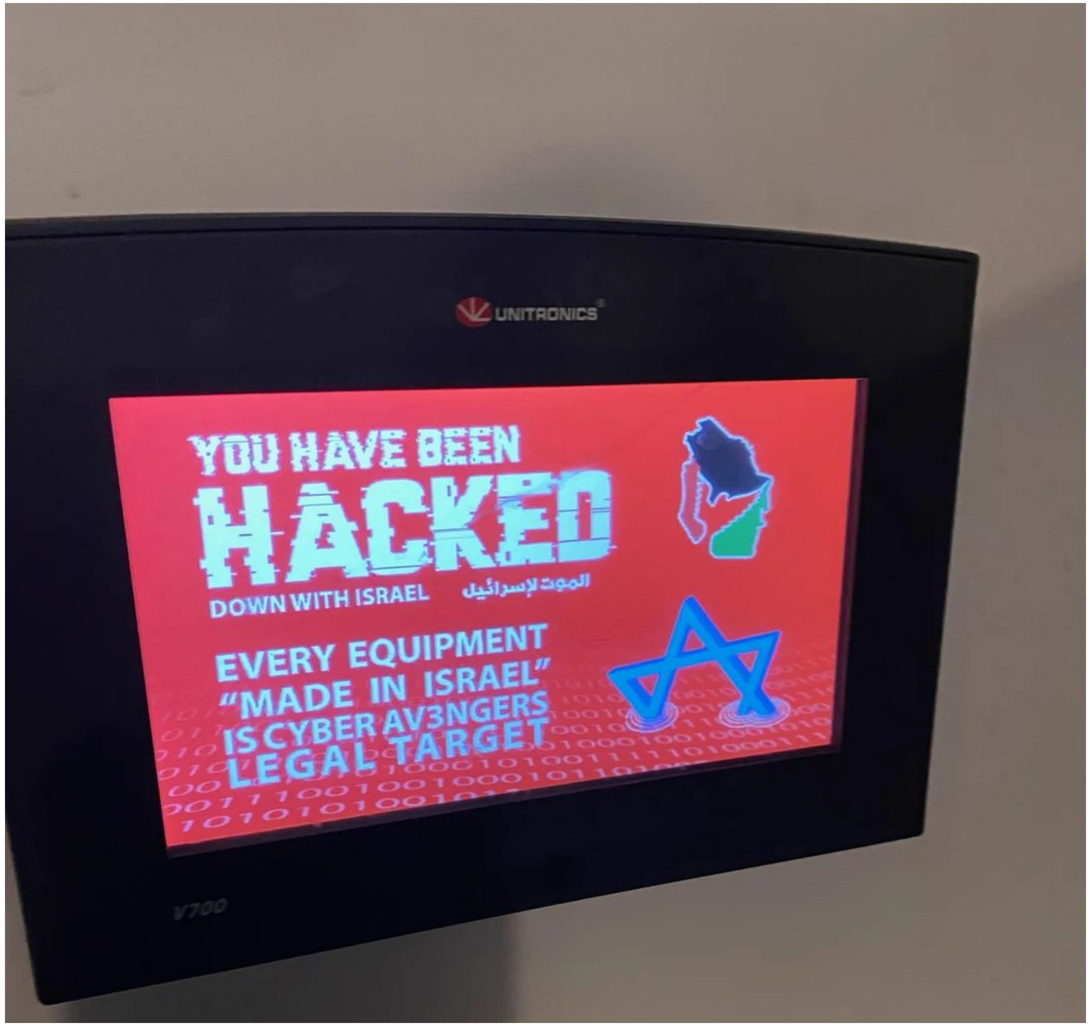


This photo provided by the Municipal Water Authority of Aliquippa shows the screen of a Unitronics device that was hacked in Aliquippa, Pa., on Nov. 25.

Municipal Water Authority of Aliquippa via AP

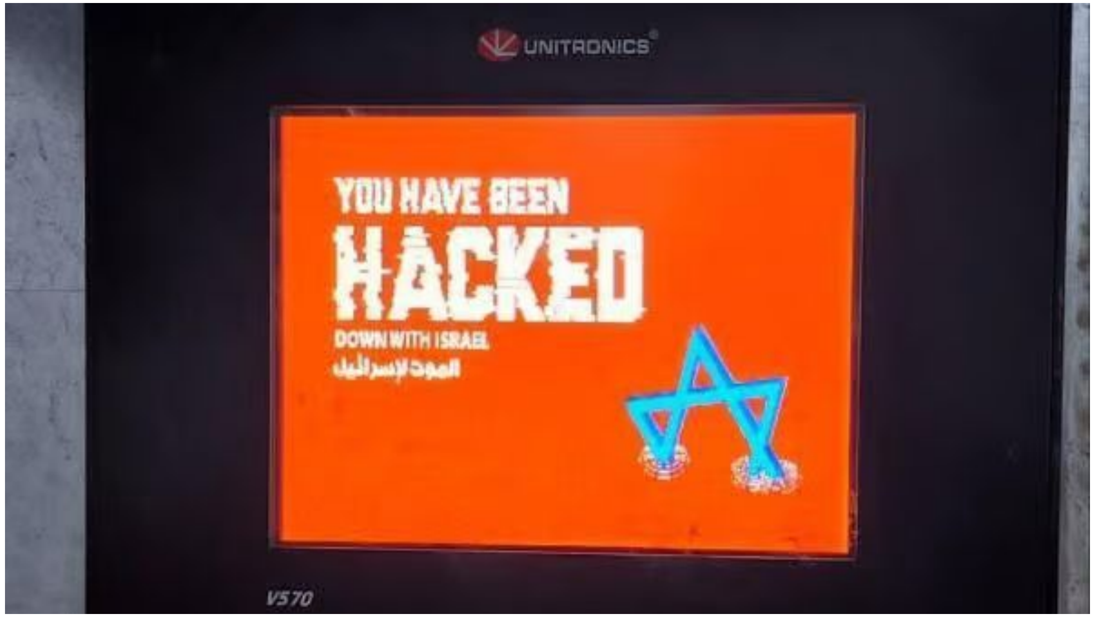
Modern Defacing ICS Style

- Defacing HMI screens
- How?
  - Downloading new project
  - Override current logic
- Was the defacement the only thing the attackers did?



Not The First Time

- **Feb ‘22** - Same attack on Israeli devices:
    - 1.5~ years prior
- Same PLC lineup
- Attackers were not identified
    - Probably same APT: shared assets



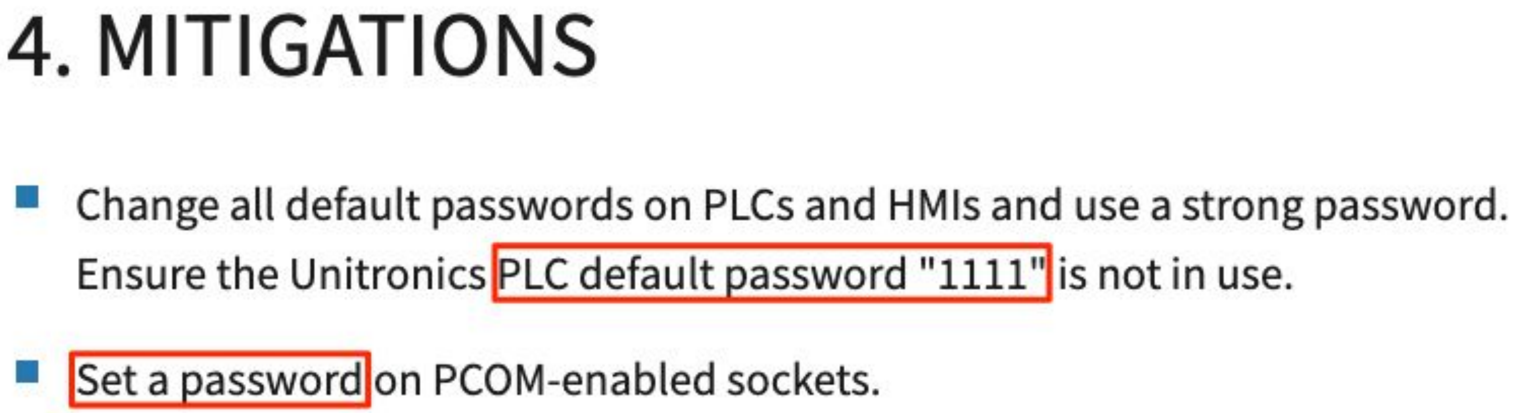
2022 Attack on Israeli Parcel Services

Unitronics Vision 101

- PLC + HMI
- Vendor is an Israeli PLC makers
- Old PLCS - Samba and Vision Series
- PCOM protocol (serial or TCP/20256)
- Almost no security mechanisms
  - No encryption
  - “Weak” authentication

“Weak” Authentication?

- From CISA advisory, they recommend:
  - Change default password
  - Add PCOM password



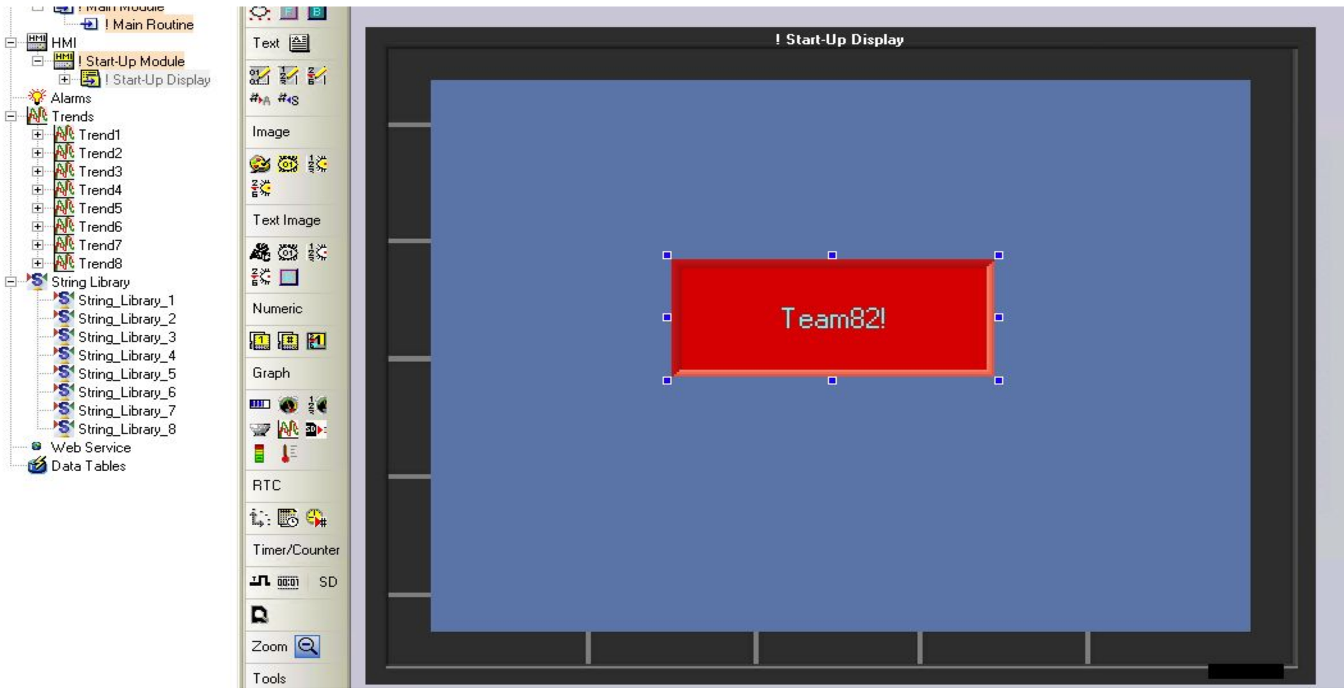
4. MITIGATIONS

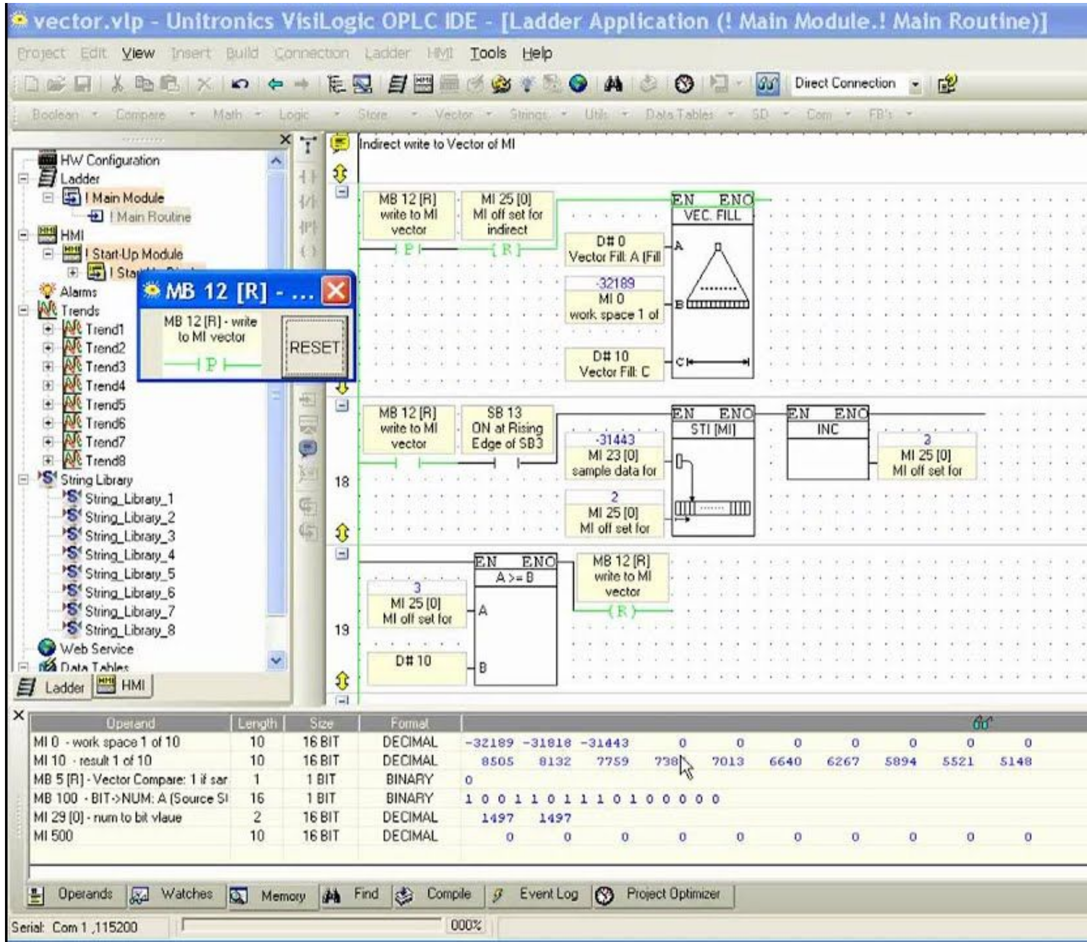
- Change all default passwords on PLCs and HMIs and use a strong password. Ensure the Unitronics PLC default password "1111" is not in use.
- Set a password on PCOM-enabled sockets.

However...

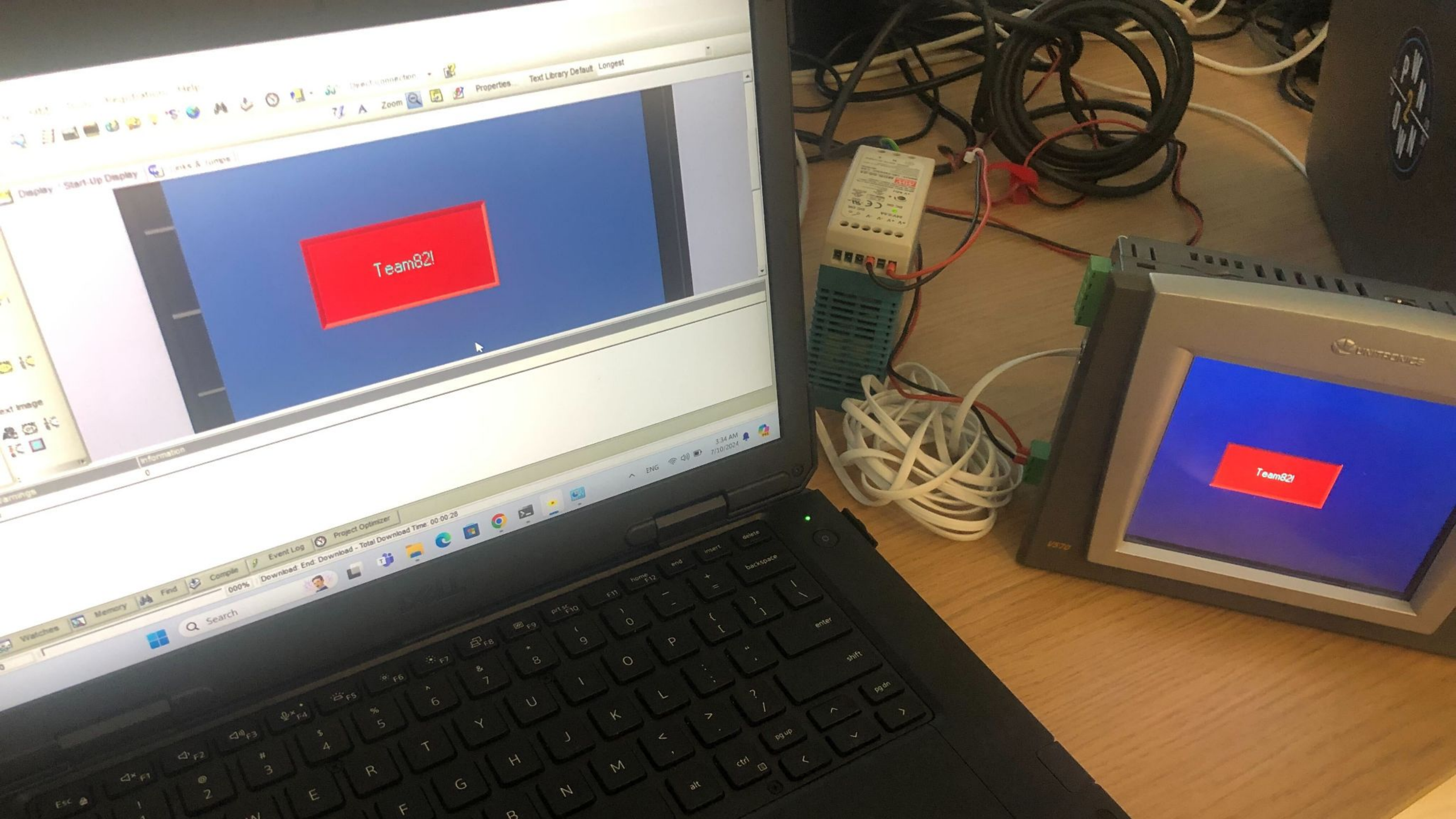
More Like No Authentication!

- Prior to v9.9.00 - **no PCOM authentication**
- To attack you need:
  - EWS: Visilogic
  - IP




vector.vlp - Unitronics VisiLogic OPLC IDE - [Ladder Application (! Main Module.! Main Routine)]

Operand	Length	Size	Format										
MI 0 - work space 1 of 10	10	16 BIT	DECIMAL	-32189	-31818	-31443	0	0	0	0	0	0	0
MI 10 - result 1 of 10	10	16 BIT	DECIMAL	8505	8132	7759	738	7013	6640	6267	5894	5521	5148
MB 5 [R] - Vector Compare: 1 if sar	1	1 BIT	BINARY	0									
MB 100 - BIT->NUM: A (Source Si	16	1 BIT	BINARY	1 0 0 1 1 0 1 1 1 0 1 0 0 0 0 0									
MI 29 [0] - num to bit vlaue	2	16 BIT	DECIMAL	1497	1497								
MI 500	10	16 BIT	DECIMAL	0	0	0	0	0	0	0	0	0	0

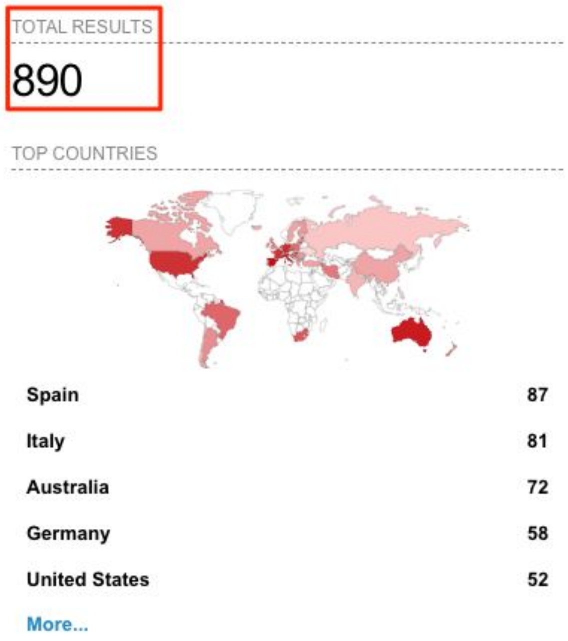


Download: End: Download - Total Download Time: 00:00:28

There are no internet-facing PLCs right?
Right???

Hundreds of Exposed Devices

- Using shodan.io:
  - 900 devices
  - PCOM exported
- Unpatched devices have no authentication!



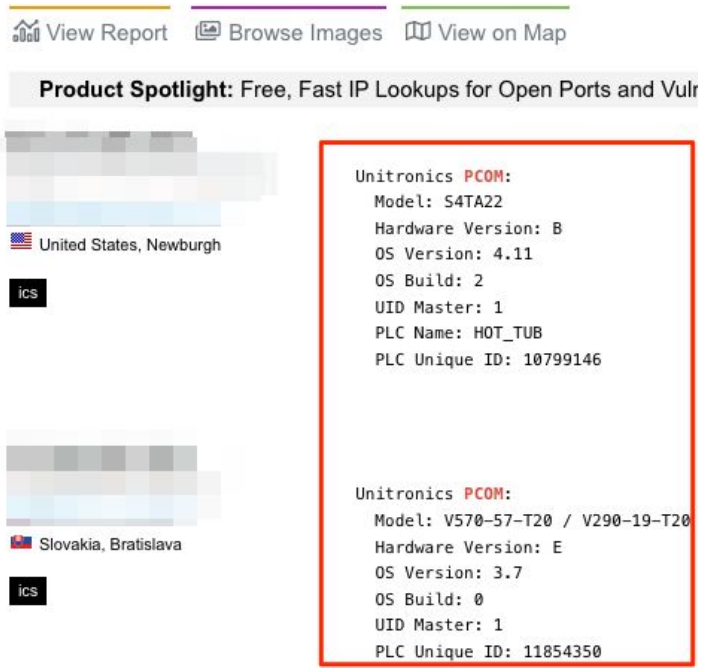
TOTAL RESULTS

890

TOP COUNTRIES

Spain	87
Italy	81
Australia	72
Germany	58
United States	52

More...



Product Spotlight: Free, Fast IP Lookups for Open Ports and Vulr

[redacted]

United States, Newburgh

ics

```
Unitronics PCOM:
  Model: S4TA22
  Hardware Version: B
  OS Version: 4.11
  OS Build: 2
  UID Master: 1
  PLC Name: HOT_TUB
  PLC Unique ID: 10799146
```

[redacted]

Slovakia, Bratislava

ics

```
Unitronics PCOM:
  Model: V570-57-T20 / V290-19-T20
  Hardware Version: E
  OS Version: 3.7
  OS Build: 0
  UID Master: 1
  PLC Unique ID: 11854350
```

Real Video of the APT Attack!

We Were Noted of This Attack

- We began investigating
  - There is no forensic tools for such device!

- Develop new forensic tools
  - Extract evidence from affected PLCs

We Were Noted of This Attack

- We began investigating
  - There is no forensic tools for such device!

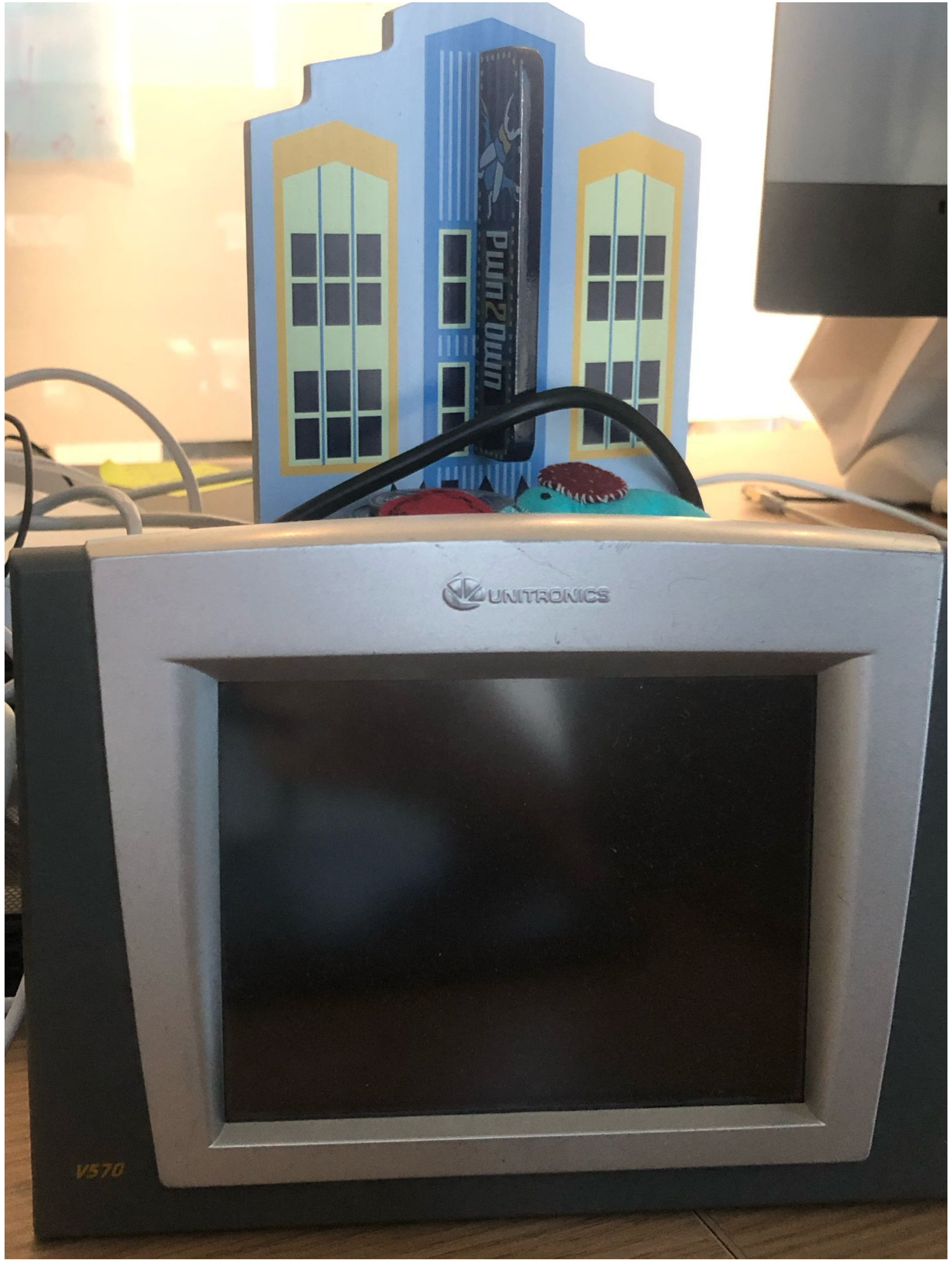
- Develop new forensic tools
  - Extract evidence from attacked PLCs

- Wait, evidence **from the PLC???**
  - This is an embedded system!
  - This was a new-ish approach

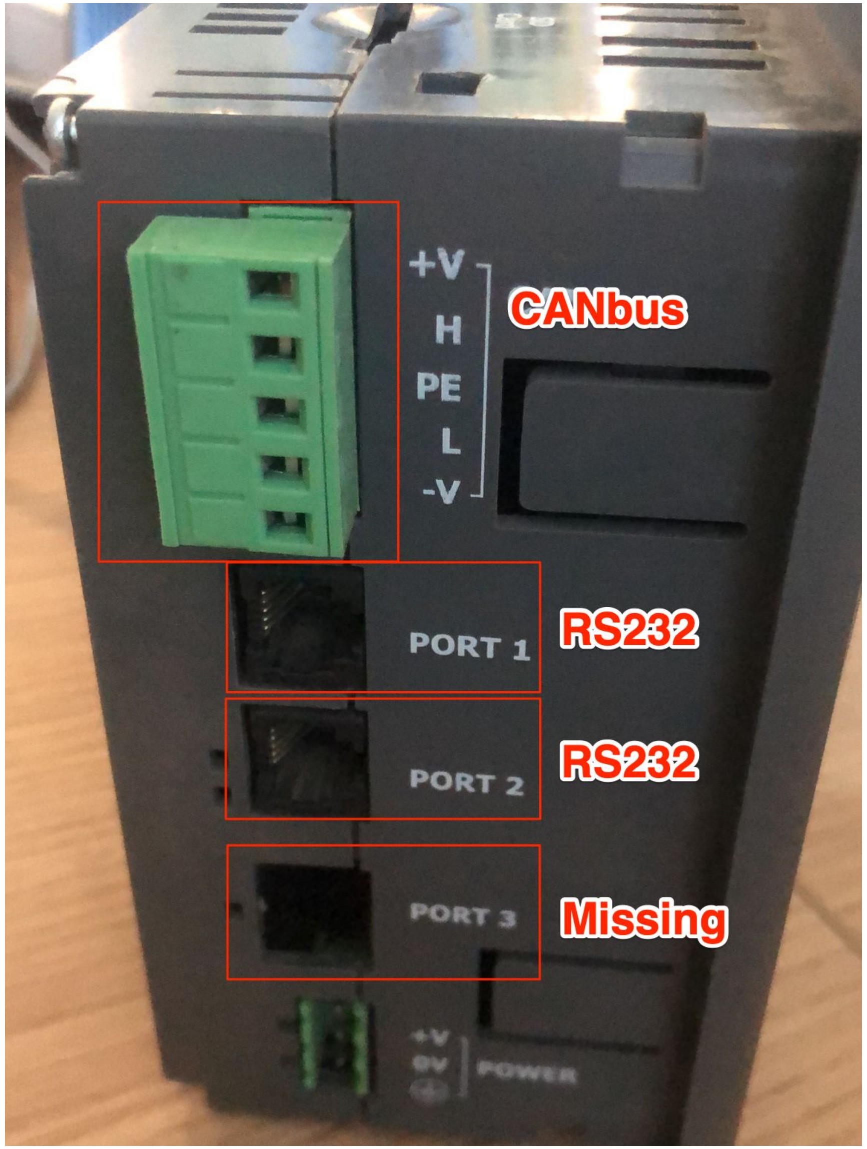
The Old Approach of Forensic in ICS

- In most cases - evidence is collected from Windows machines
  - Triton, Stuxnet, …
- In this case - attack did not involve Windows machines
  - Can we extract forensic data from the PLC?
- No evidence was collected from PLCs
  - No evidence stored on PLC?
  - Not easy to collect it
  - Microsoft released a ICS evidence collection tool - ICSpector

So We Bought a Device…



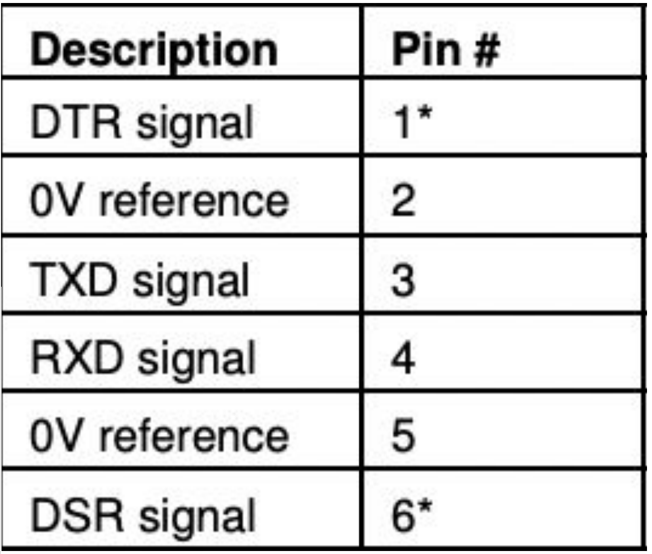
Uh-oh, our device is missing an Ethernet card



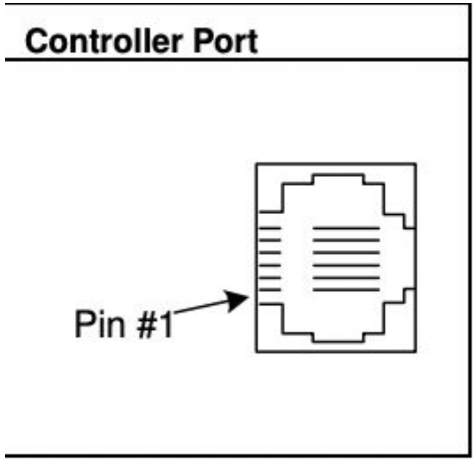
Let’s Build One!

Pin Layout

- **Vision pin layout (RJ11):**

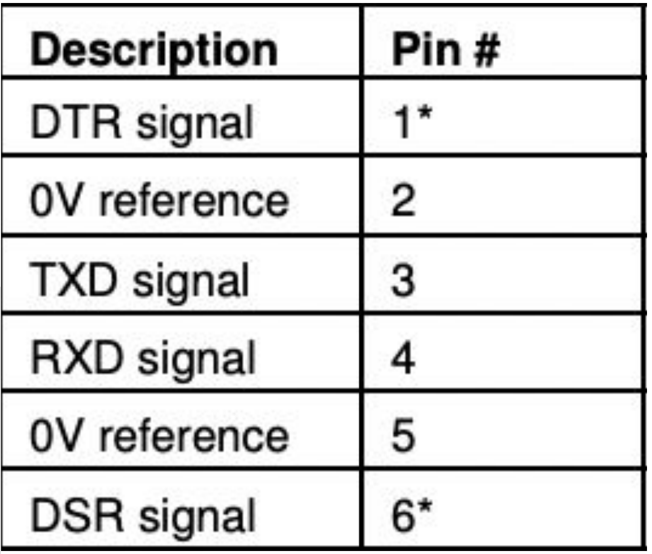


Description	Pin #
DTR signal	1*
0V reference	2
TXD signal	3
RXD signal	4
0V reference	5
DSR signal	6*

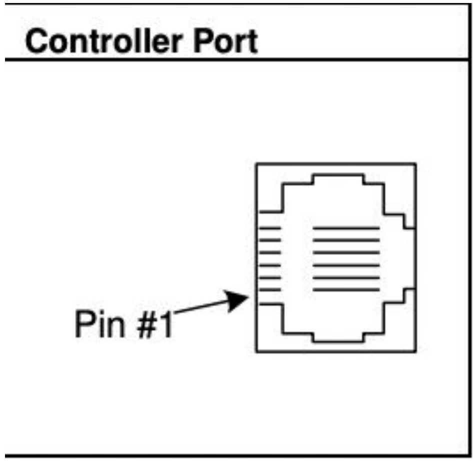


Pin Layout

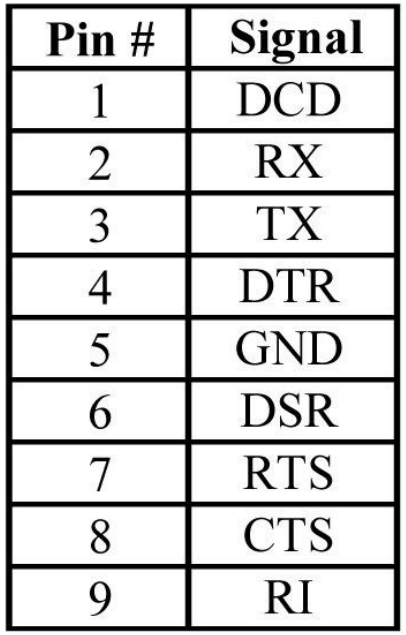
- **Vision pin layout (RJ11):**



Description	Pin #
DTR signal	1*
0V reference	2
TXD signal	3
RXD signal	4
0V reference	5
DSR signal	6*

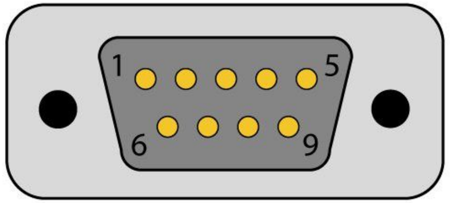


- **DB9 pin layout:**



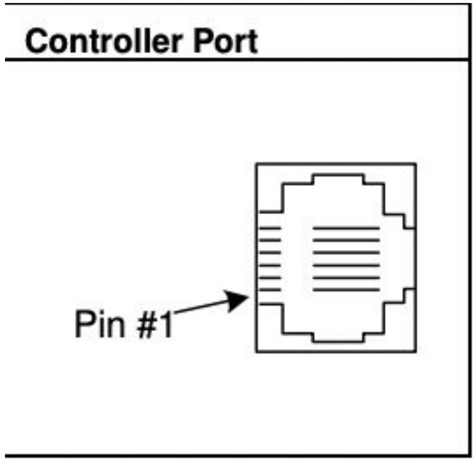
Pin #	Signal
1	DCD
2	RX
3	TX
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

DB9M Connector



Pin Layout

- Vision pin layout (RJ11):

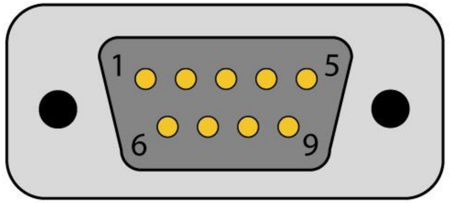


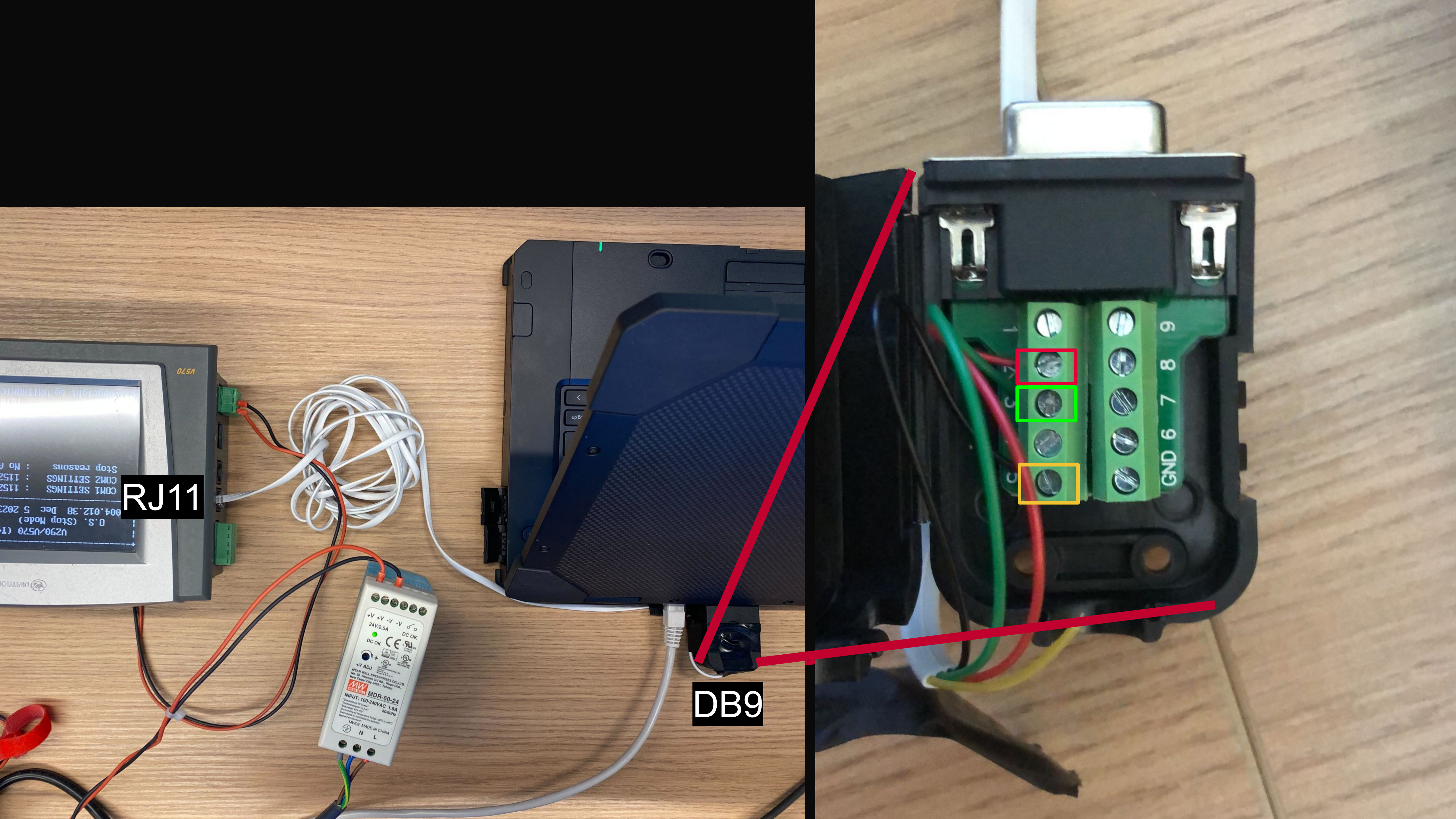
Description	Pin #
DTR signal	1*
0V reference	2
TXD signal	3
RXD signal	4
0V reference	5
DSR signal	6*

- DB9 pin layout:

Pin #	Signal
1	DCD
2	RX
3	TX
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

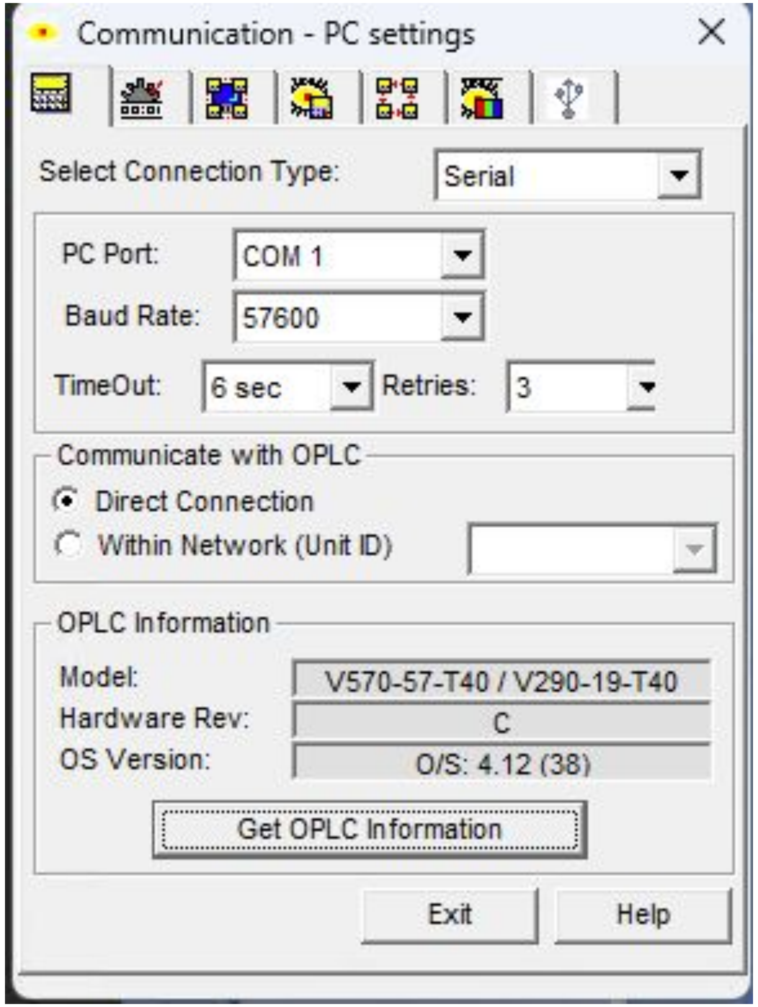
DB9M Connector




RJ11
DB9
1
2
3
5
9
8
7
6
GND

Connecting EWS to PLC Using Serial

- We can connect to the PLC
- Can debug/RE the binaries
  - Start understanding the protocol



Communication - PC settings

Select Connection Type: Serial

PC Port: COM 1

Baud Rate: 57600

TimeOut: 6 sec Retries: 3

Communicate with OPLC

● Direct Connection

○ Within Network (Unit ID)

OPLC Information

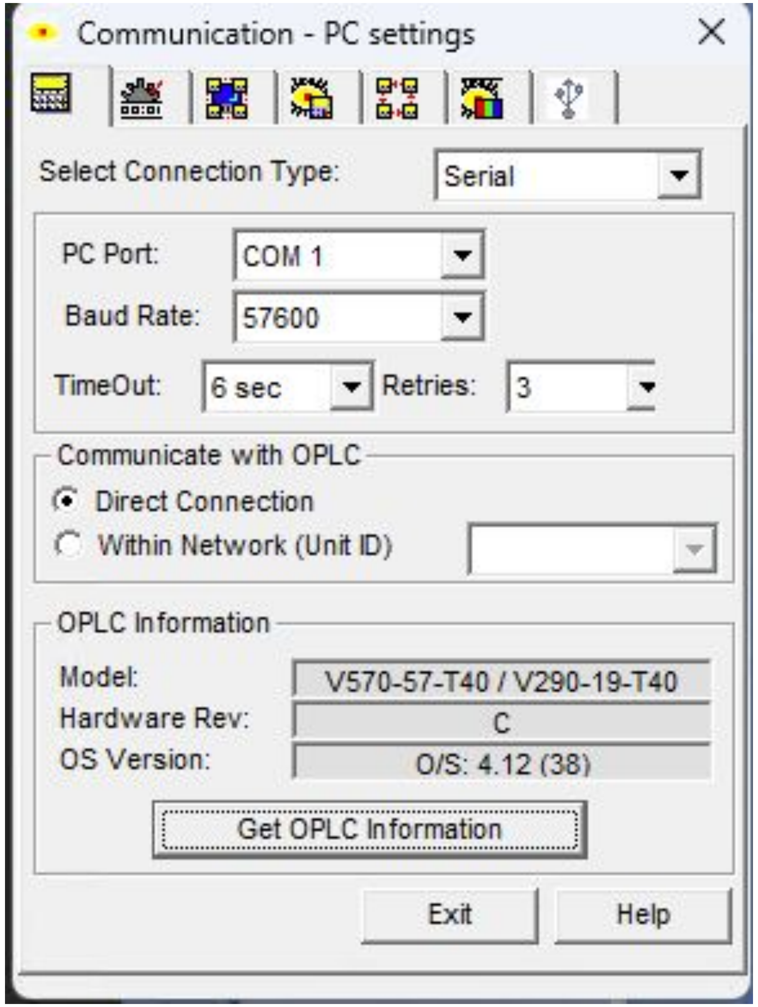
Model: V570-57-T40 / V290-19-T40

Hardware Rev: C

OS Version: O/S: 4.12 (38)

Connecting EWS to PLC Using Serial

- But…
- We cannot MiTM/sniff the packets
  - Engineering Work Station opens serial port in exclusive mode
  - Cannot capture data



Communication - PC settings

Select Connection Type: Serial

PC Port: COM 1

Baud Rate: 57600

TimeOut: 6 sec Retries: 3

Communicate with OPLC

● Direct Connection

○ Within Network (Unit ID)

OPLC Information

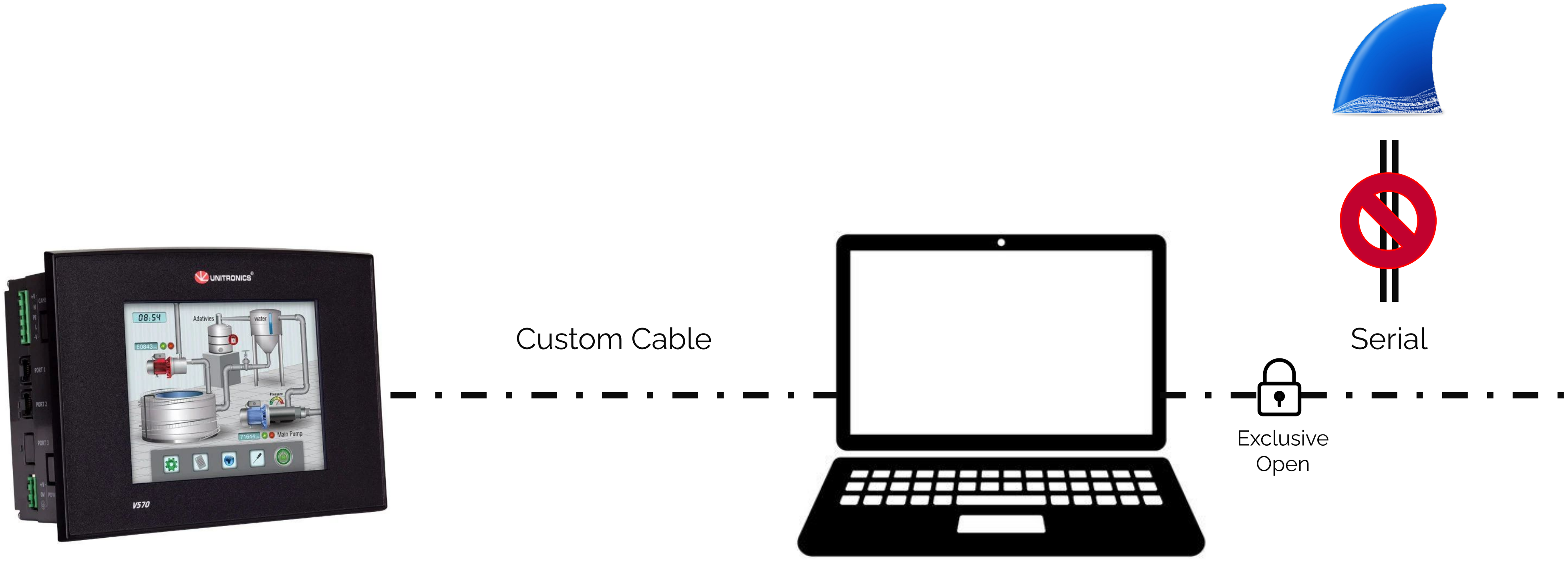
Model: V570-57-T40 / V290-19-T40

Hardware Rev: C

OS Version: O/S: 4.12 (38)

Let’s figure out a plan

Current Situation



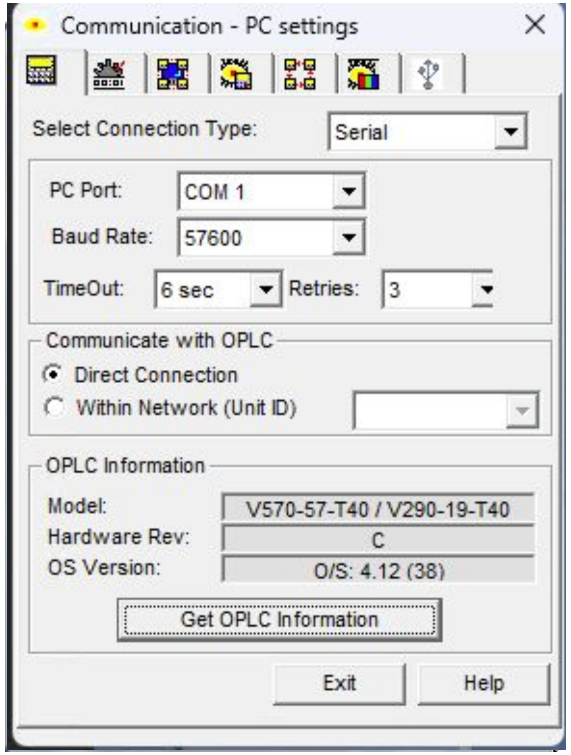
Custom Cable

Serial

Exclusive Open

Vision Device

Computer



Communication - PC settings

Select Connection Type: Serial

PC Port: COM 1

Baud Rate: 57600

TimeOut: 6 sec Retries: 3

Communicate with OPLC

● Direct Connection

○ Within Network (Unit ID)

OPLC Information

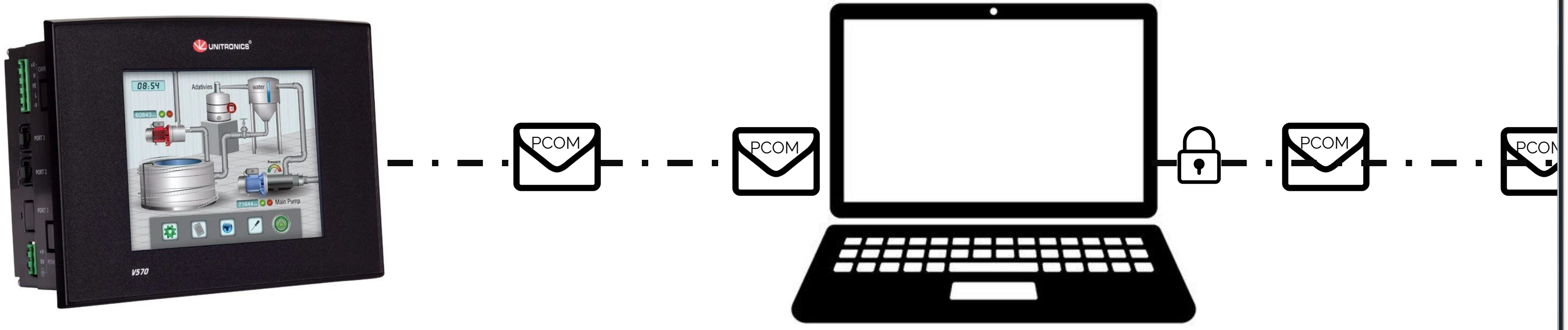
Model: V570-57-T40 / V290-19-T40

Hardware Rev: C

OS Version: O/S: 4.12 (38)

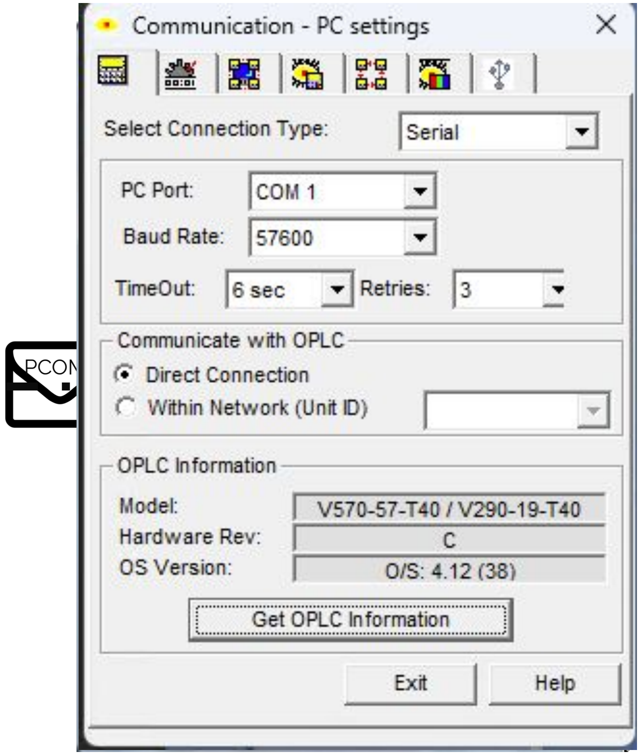
EWS

Current Situation



Vision Device

Computer


Communication - PC settings

Select Connection Type: Serial

PC Port: COM 1

Baud Rate: 57600

TimeOut: 6 sec Retries: 3

Communicate with OPLC

● Direct Connection

○ Within Network (Unit ID)

OPLC Information

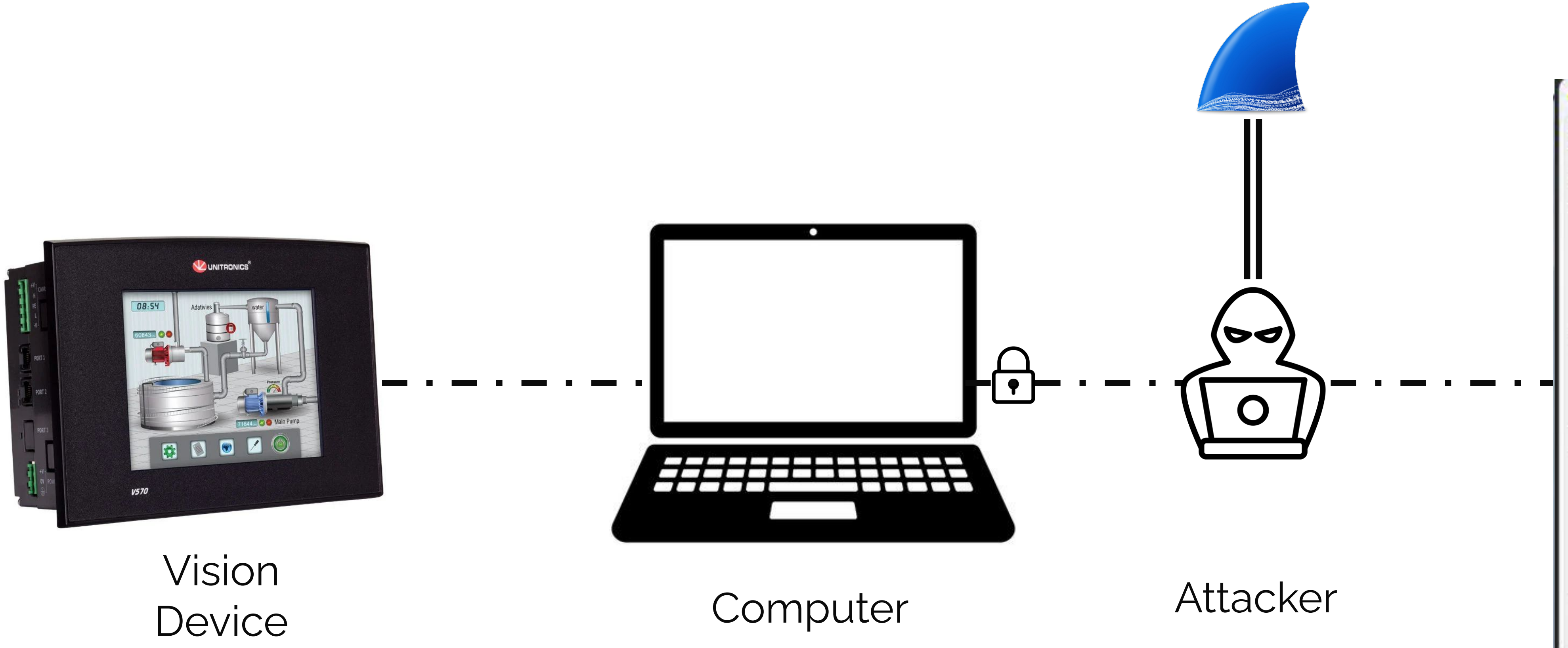
Model: V570-57-T40 / V290-19-T40

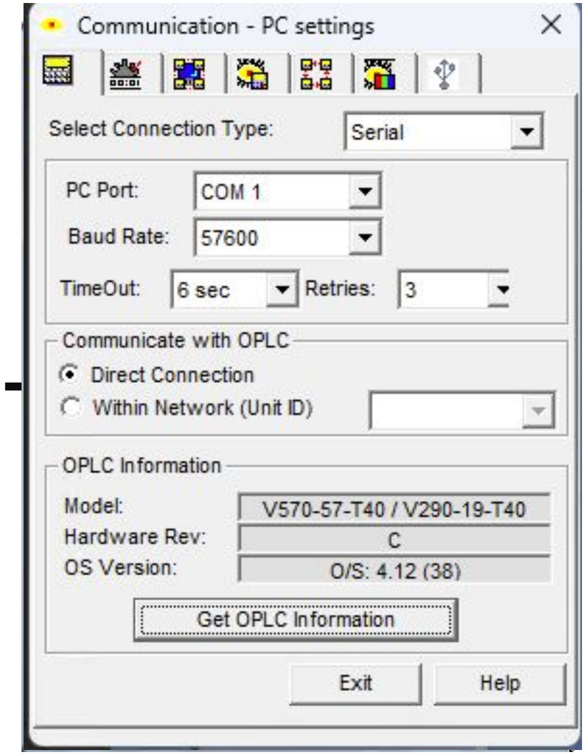
Hardware Rev: C

OS Version: O/S: 4.12 (38)

EWS

What We Want - MiTM & Sniffing





Communication - PC settings

Select Connection Type: Serial

PC Port: COM 1

Baud Rate: 57600

TimeOut: 6 sec Retries: 3

Communicate with OPLC

● Direct Connection

○ Within Network (Unit ID)

OPLC Information

Model: V570-57-T40 / V290-19-T40

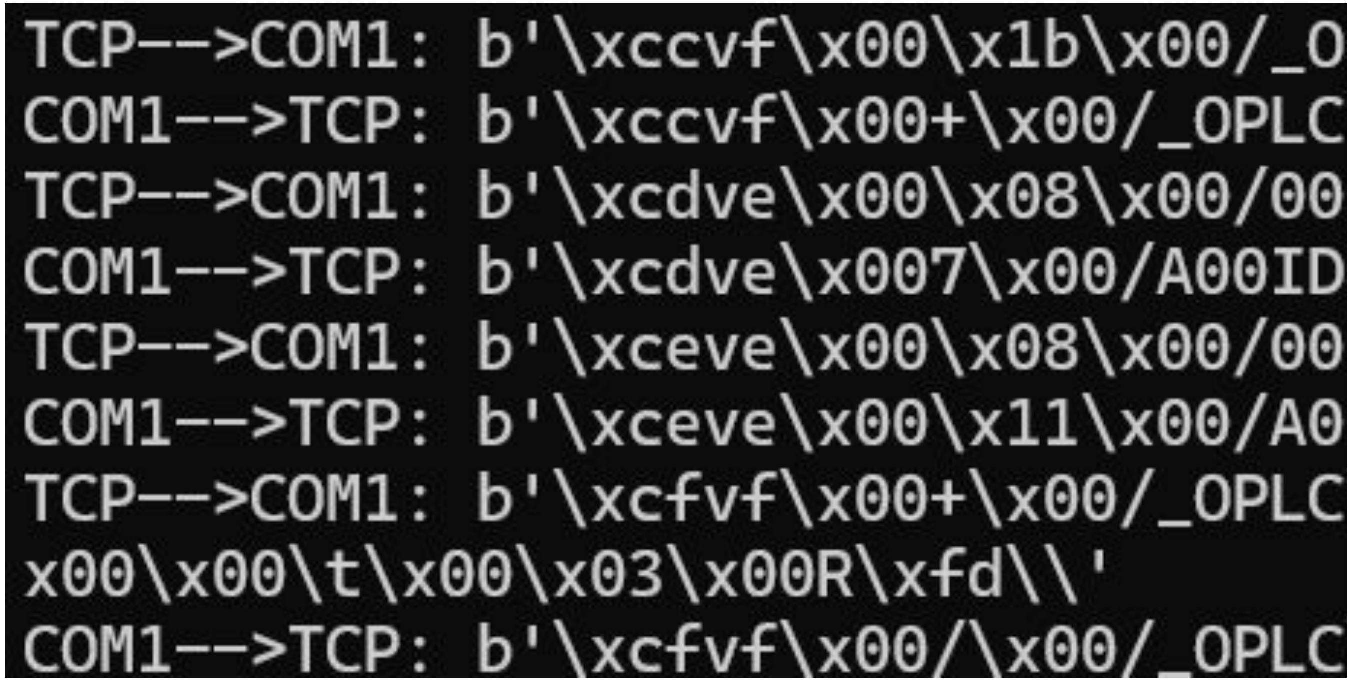
Hardware Rev: C

OS Version: O/S: 4.12 (38)

Tool #1 - PCOM2TCP

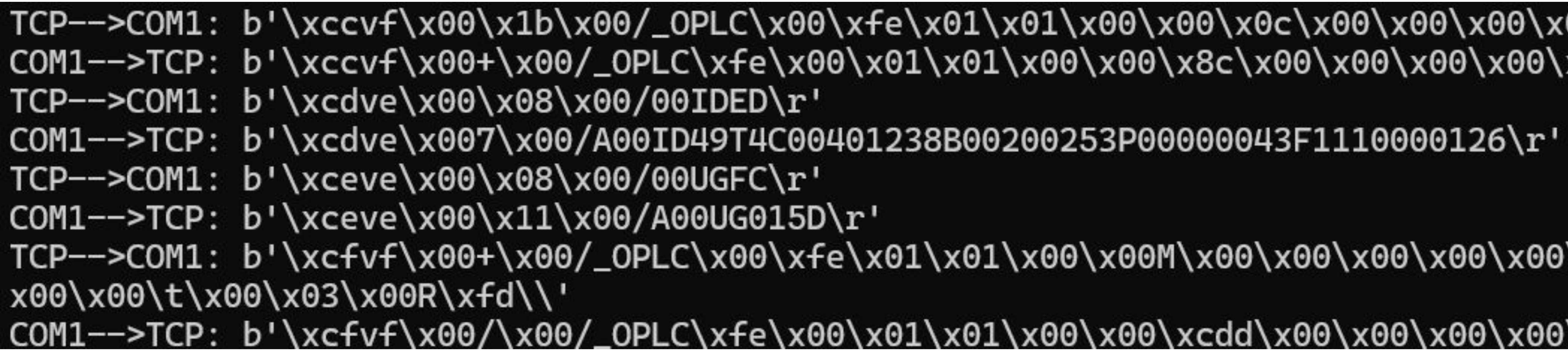
- Encapsulates serial COM in PCOM\TCP layer

- We now can:
  - Use wireshark
  - MiTM



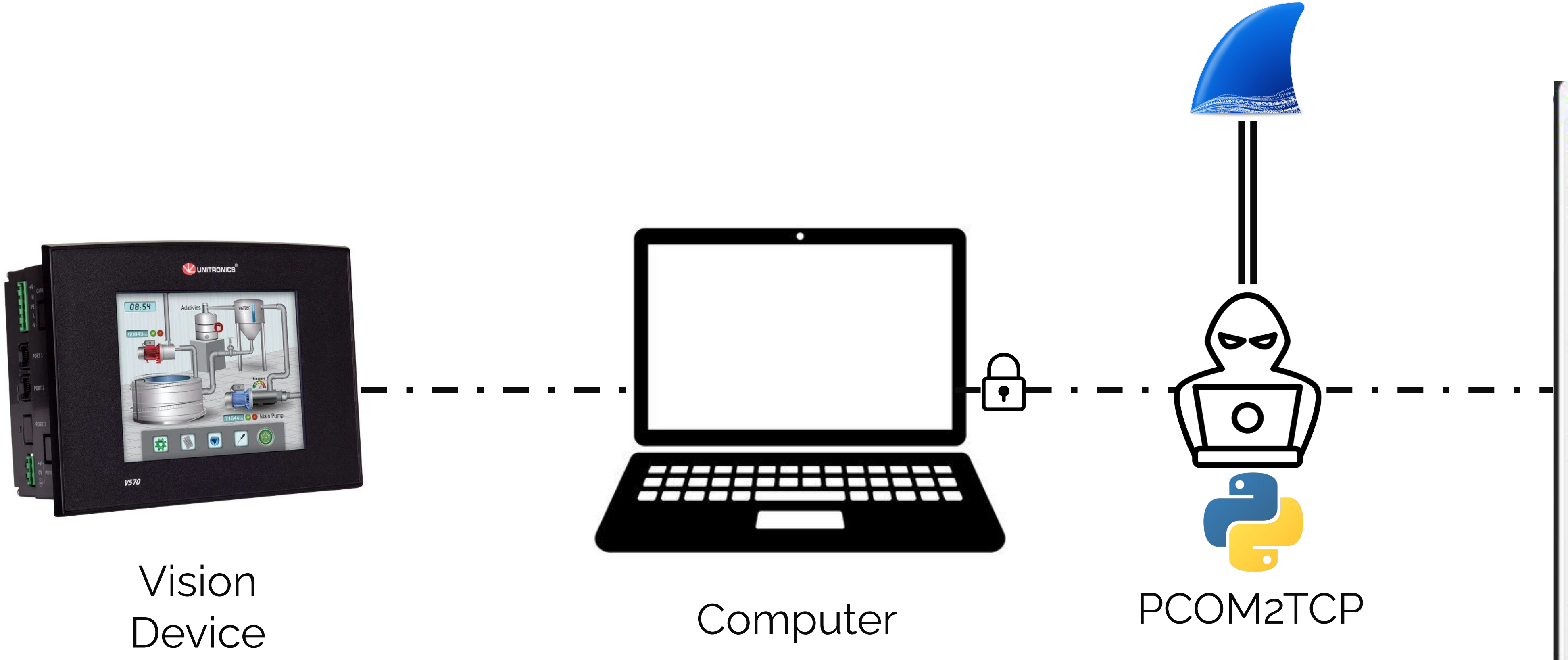
```
TCP-->COM1: b'\xccvf\x00\x1b\x00/_O
COM1-->TCP: b'\xccvf\x00+\x00/_OPLC
TCP-->COM1: b'\xcdve\x00\x08\x00/00
COM1-->TCP: b'\xcdve\x007\x00/A00ID
TCP-->COM1: b'\xceve\x00\x08\x00/00
COM1-->TCP: b'\xceve\x00\x11\x00/A0
TCP-->COM1: b'\xcfvf\x00+\x00/_OPLC
x00\x00\t\x00\x03\x00R\xfd\\'
COM1-->TCP: b'\xcfvf\x00/\x00/_OPLC
```

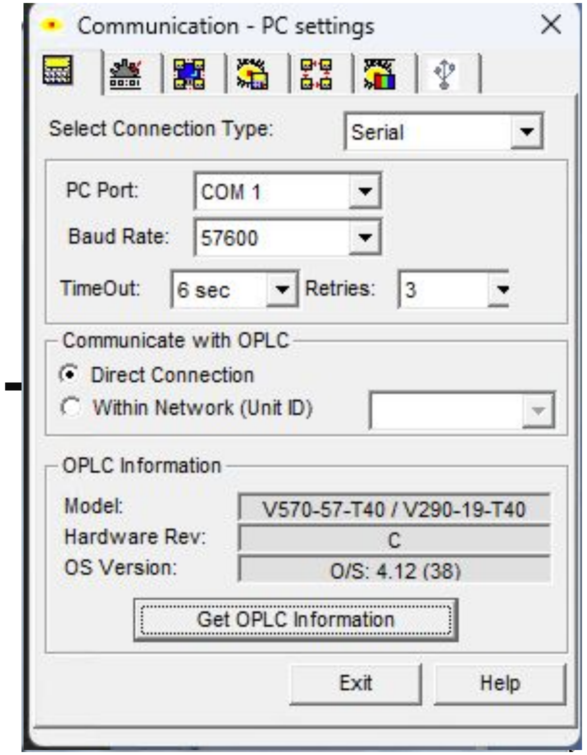
PCOM2TCP



```
TCP-->COM1: b'\xccvf\x00\x1b\x00/_OPLC\x00\xfe\x01\x01\x00\x00\x0c\x00\x00\x00\x
COM1-->TCP: b'\xccvf\x00+\x00/_OPLC\xfe\x00\x01\x01\x00\x00\x8c\x00\x00\x00\x00\
TCP-->COM1: b'\xcdve\x00\x08\x00/00IDED\r'
COM1-->TCP: b'\xcdve\x007\x00/A00ID49T4C00401238B00200253P00000043F1110000126\r'
TCP-->COM1: b'\xceve\x00\x08\x00/00UGFC\r'
COM1-->TCP: b'\xceve\x00\x11\x00/A00UG015D\r'
TCP-->COM1: b'\xcfvf\x00+\x00/_OPLC\x00\xfe\x01\x01\x00\x00M\x00\x00\x00\x00\x00
x00\x00\t\x00\x03\x00R\xfd\\'
COM1-->TCP: b'\xcfvf\x00/\x00/_OPLC\xfe\x00\x01\x01\x00\x00\xcdd\x00\x00\x00\x00
```

PCOM2TCP - MiTM & Sniffing




Communication - PC settings

Select Connection Type: Serial

PC Port: COM 1

Baud Rate: 57600

TimeOut: 6 sec Retries: 3

Communicate with OPLC

● Direct Connection

○ Within Network (Unit ID)

OPLC Information

Model: V570-57-T40 / V290-19-T40

Hardware Rev: C

OS Version: O/S: 4.12 (38)

EWS

PCOM Protocol 101

- Communication layer: Serial vs. TCP
  - TCP/20256
- Two mods: Binary vs. ASCII
  - Binary: `0x01` (read), `0x02` (auth)
  - ASCII: `ID` (get id), `UG` (get unit-id)
- Unencrypted
- Basic Wireshark dissector + documentation
  - prior research: A Comprehensive Security Analysis of a SCADA Protocol: from OSINT to Mitigation, Luis Rosa et al., 2019. Thanks! :)

PCOM Binary Format

MAGIC (/_OPLC)	ID (0x00)	Reserved (0xFE01010000)	Opcode (0x0C)	Reserved (0x00)	Command Details (0x000000006A0 0)	Length (0x7E00)	Header CRC (0x4DFC)	Data ...	Footer CRC (0x4DFC)	MAGIC (\ )
6 Bytes	1 Byte	5 Bytes	1 Byte	1 Byte	6 Bytes	2 Bytes	2 Bytes		2 Bytes	

Command examples:

Desc	Request	Response
Read Operand	0x4D	0xCD
Get PLC Name	0x0C	0x8C

opcode is a request or response?
Check MSB:

`0b00000000` => request

`0b10000000` => response

PCOM ASCII Format

MAGIC (/)	UID (0x3030)	Command Code (ID)	Data (…)	CRC (0x4346)	Suffix (\r)
1\2 Bytes req \ resp	2 Bytes	Changes		2 Bytes	1 Byte

Command examples:

Code	Description
ID	Send Identification Command
UG	Get Unit ID Command

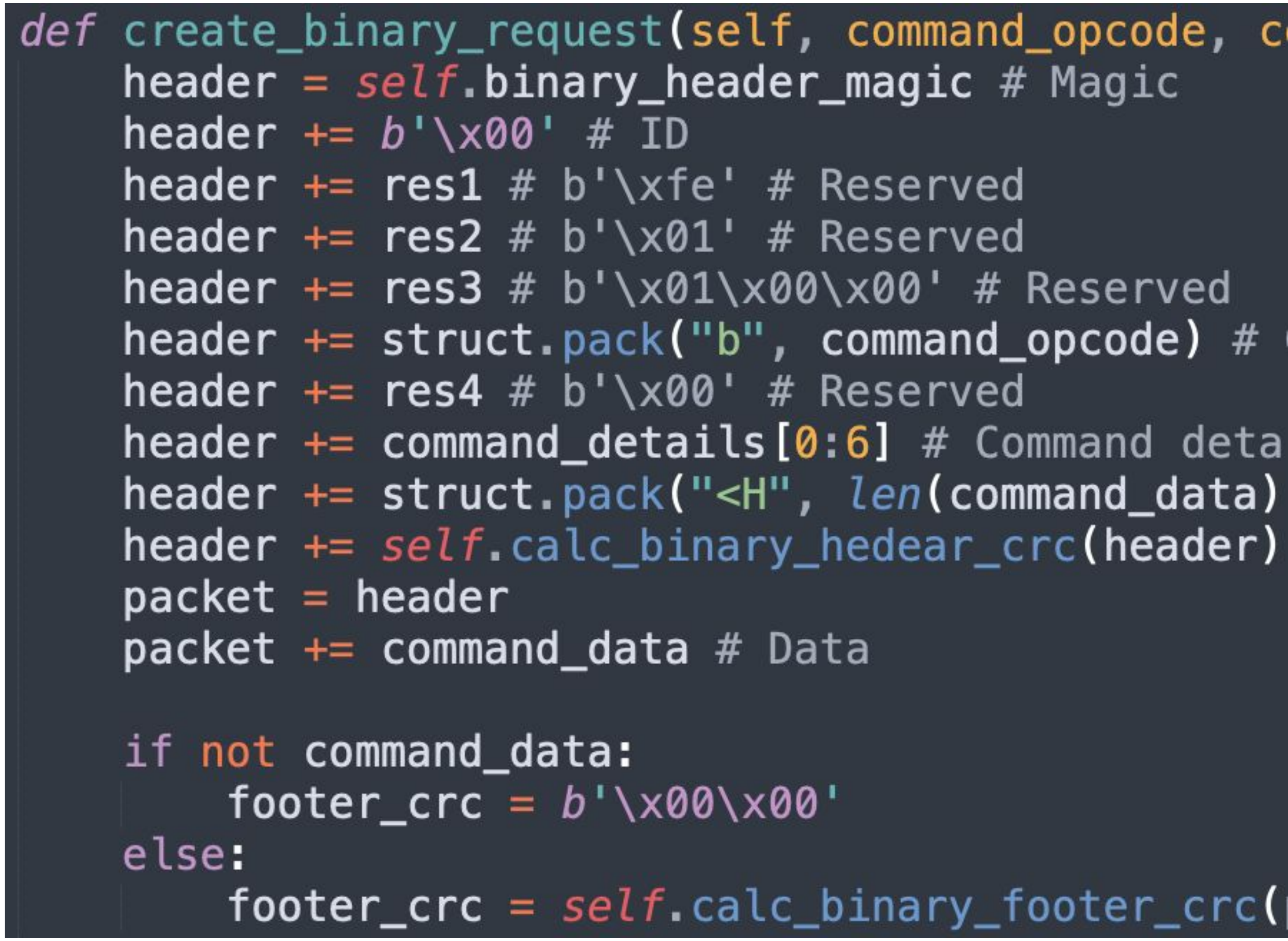
Different magic for requests and responses:

/ => request

/a => response

Tool #2 - PCOMClient

- Supports:
  - PCOM\TCP and serial
  - PCOM Binary and PCOM ASCII
- Interface for adding opcodes
- Many built-in opcodes and operations

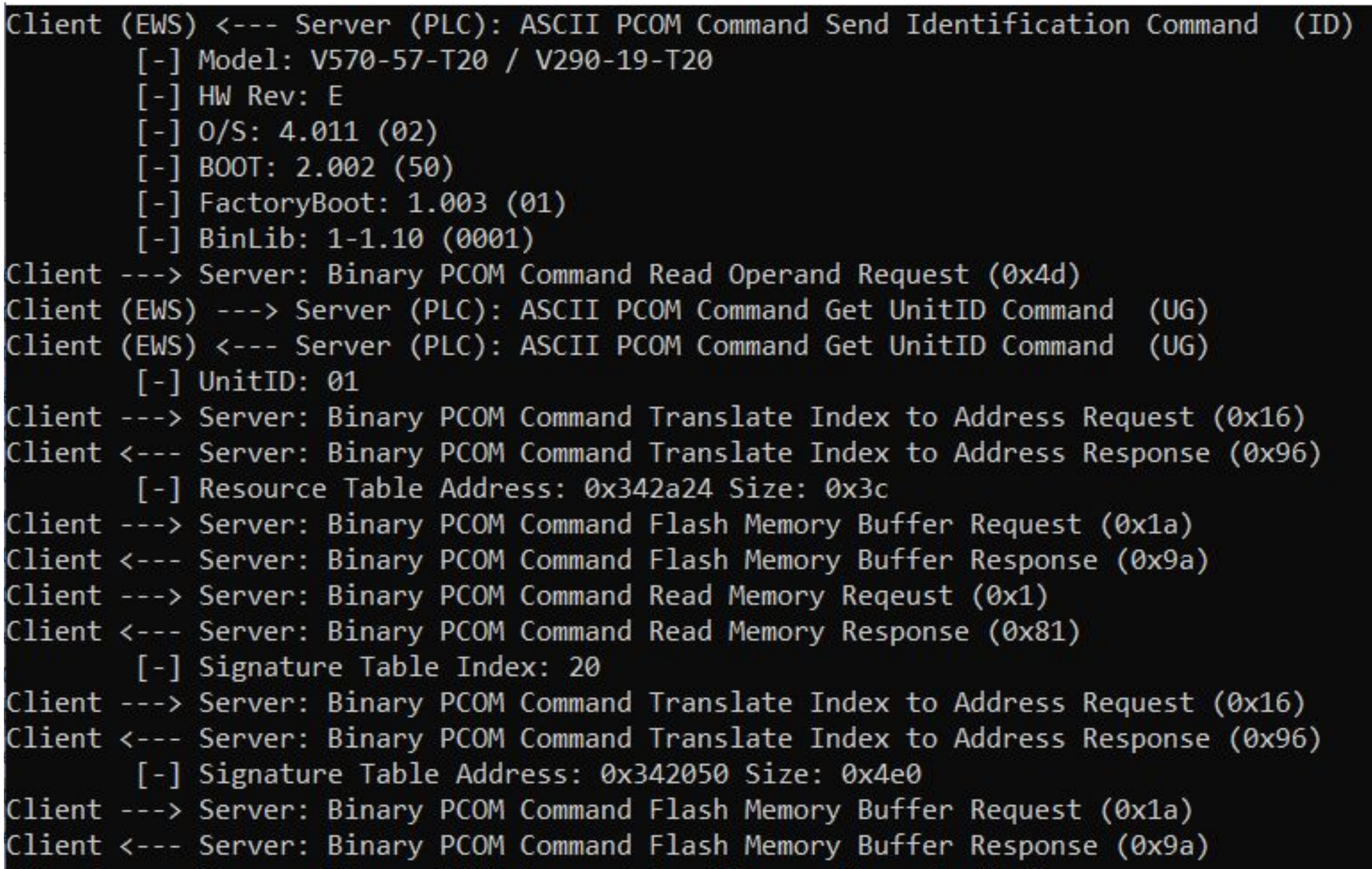


```
def create_binary_request(self, command_opcode, c
    header = self.binary_header_magic # Magic
    header += b'\x00' # ID
    header += res1 # b'\xfe' # Reserved
    header += res2 # b'\x01' # Reserved
    header += res3 # b'\x01\x00\x00' # Reserved
    header += struct.pack("b", command_opcode) #
    header += res4 # b'\x00' # Reserved
    header += command_details[0:6] # Command deta
    header += struct.pack("<H", len(command_data)
    header += self.calc_binary_hedear_crc(header)
    packet = header
    packet += command_data # Data

    if not command_data:
        footer_crc = b'\x00\x00'
    else:
        footer_crc = self.calc_binary_footer_crc(
```

Tool #2 - PCOMClient

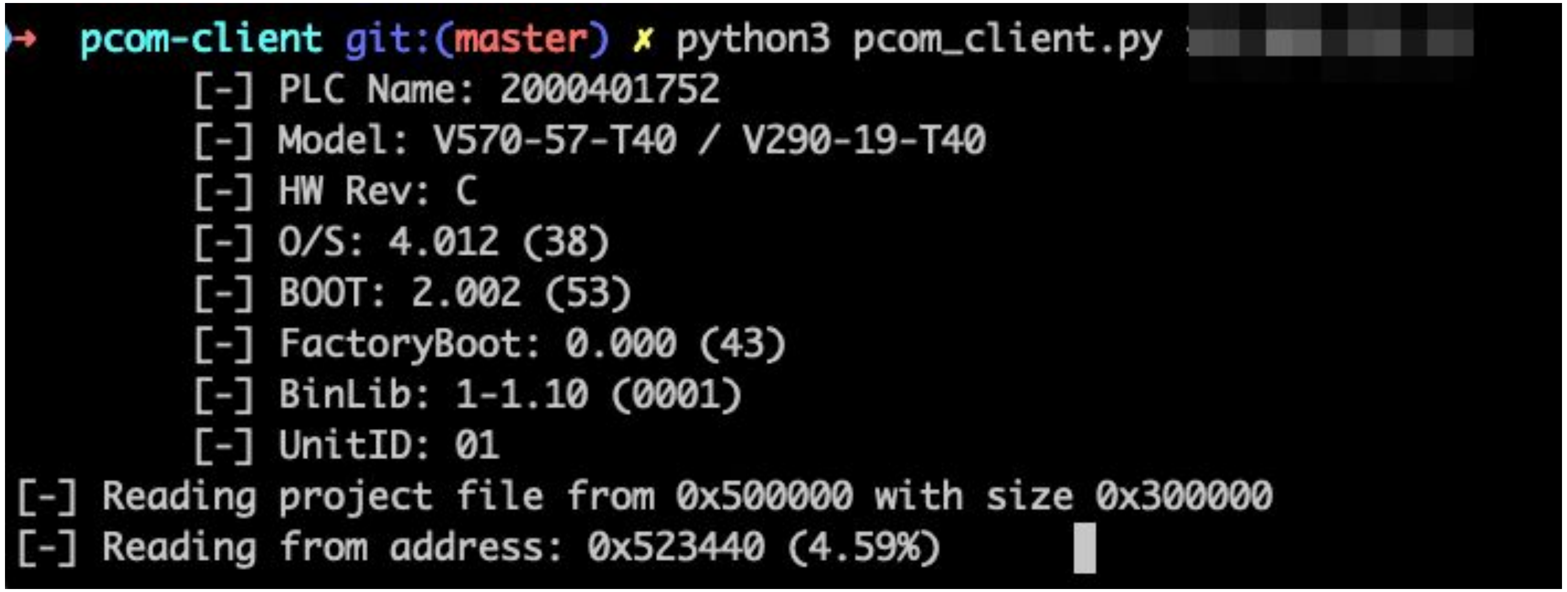
Releasing today as open-source tool!



```
Client (EWS) <--- Server (PLC): ASCII PCOM Command Send Identification Command  (ID)
        [-] Model: V570-57-T20 / V290-19-T20
        [-] HW Rev: E
        [-] O/S: 4.011 (02)
        [-] BOOT: 2.002 (50)
        [-] FactoryBoot: 1.003 (01)
        [-] BinLib: 1-1.10 (0001)
Client ---> Server: Binary PCOM Command Read Operand Request (0x4d)
Client (EWS) ---> Server (PLC): ASCII PCOM Command Get UnitID Command  (UG)
Client (EWS) <--- Server (PLC): ASCII PCOM Command Get UnitID Command  (UG)
        [-] UnitID: 01
Client ---> Server: Binary PCOM Command Translate Index to Address Request (0x16)
Client <--- Server: Binary PCOM Command Translate Index to Address Response (0x96)
        [-] Resource Table Address: 0x342a24 Size: 0x3c
Client ---> Server: Binary PCOM Command Flash Memory Buffer Request (0x1a)
Client <--- Server: Binary PCOM Command Flash Memory Buffer Response (0x9a)
Client ---> Server: Binary PCOM Command Read Memory Reqeust (0x1)
Client <--- Server: Binary PCOM Command Read Memory Response (0x81)
        [-] Signature Table Index: 20
Client ---> Server: Binary PCOM Command Translate Index to Address Request (0x16)
Client <--- Server: Binary PCOM Command Translate Index to Address Response (0x96)
        [-] Signature Table Address: 0x342050 Size: 0x4e0
Client ---> Server: Binary PCOM Command Flash Memory Buffer Request (0x1a)
Client <--- Server: Binary PCOM Command Flash Memory Buffer Response (0x9a)
```

Arbitrary Memory Read/Write

- Discovering function codes:
  - 0x01 - memory **READ**
  - 0x41 - memory **WRITE**
- Let's analyze!

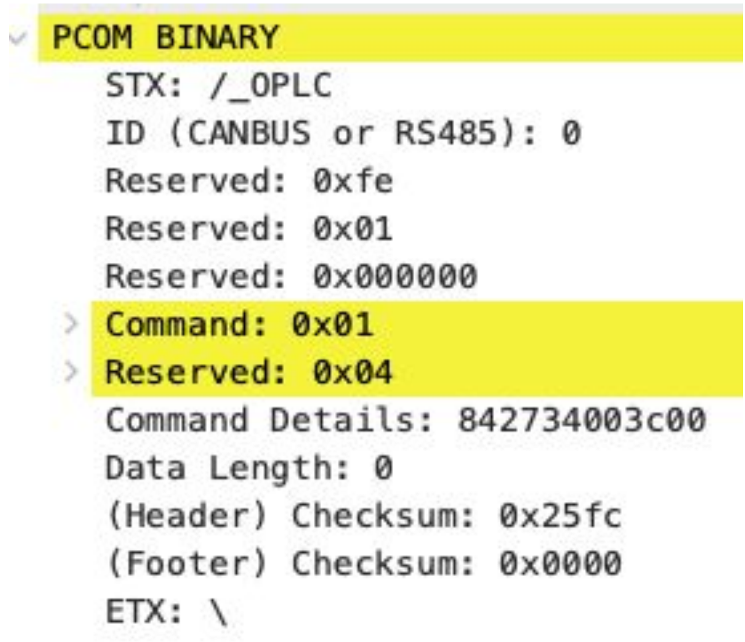


```
➜  pcom-client git:(master) ✗ python3 pcom_client.py [redacted]
        [-] PLC Name: 2000401752
        [-] Model: V570-57-T40 / V290-19-T40
        [-] HW Rev: C
        [-] O/S: 4.012 (38)
        [-] BOOT: 2.002 (53)
        [-] FactoryBoot: 0.000 (43)
        [-] BinLib: 1-1.10 (0001)
        [-] UnitID: 01
[-] Reading project file from 0x500000 with size 0x300000
[-] Reading from address: 0x523440 (4.59%)
```

Read/Write Memory Structure

MAGIC (/_OPLC)	ID (0x00)	Reserved (0xFE01000000)	Opcode (0x01)	Reserved (0xXX)	Command Details (0xXXXXXXXXXXXX)	Length (0x0000)	Header CRC (0x4DFC)	Footer CRC (0x4DFC)	MAGIC (\ )
6 Bytes	1 Byte	5 Bytes	1 Byte	1 Byte	6 Bytes	2 Bytes	2 Bytes	2 Bytes	

Opcode: 0x01



```
PCOM BINARY
    STX: /_OPLC
    ID (CANBUS or RS485): 0
    Reserved: 0xfe
    Reserved: 0x01
    Reserved: 0x000000
  > Command: 0x01
  > Reserved: 0x04
    Command Details: 842734003c00
    Data Length: 0
    (Header) Checksum: 0x25fc
    (Footer) Checksum: 0x0000
    ETX: \
```

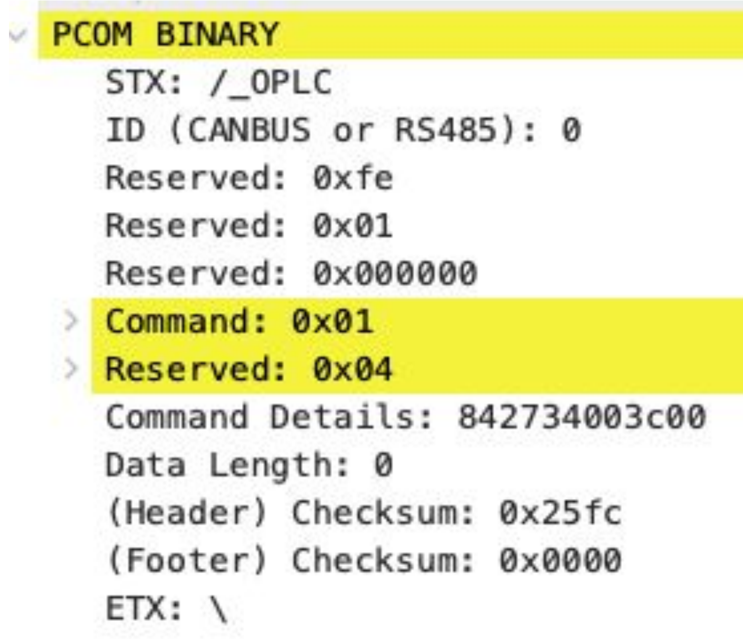
Read/Write Memory Structure

MAGIC (/_OPLC)	ID (0x00)	Reserved (0xFE01000000)	Opcode (0x01)	Reserved (0xXX)	Command Details (0xXXXXXXXXXXXX)	Length (0x0000)	Header CRC (0x4DFC)	Footer CRC (0x4DFC)	MAGIC (\ )
6 Bytes	1 Byte	5 Bytes	1 Byte	1 Byte	6 Bytes	2 Bytes	2 Bytes	2 Bytes	

Opcode: 0x01

Reserved: 0xFE01000000

Reserved 2: 0x01 \ 0x04 (changes memory region/chip)



```
PCOM BINARY
    STX: /_OPLC
    ID (CANBUS or RS485): 0
    Reserved: 0xfe
    Reserved: 0x01
    Reserved: 0x000000
  > Command: 0x01
  > Reserved: 0x04
    Command Details: 842734003c00
    Data Length: 0
    (Header) Checksum: 0x25fc
    (Footer) Checksum: 0x0000
    ETX: \
```

Read/Write Memory Structure

MAGIC (/_OPLC)	ID (0x00)	Reserved (0xFE01000000)	Opcode (0x01)	Reserved (0xXX)	Command Details (0xXXXXXXXXXXXX)	Length (0x0000)	Header CRC (0x4DFC)	Footer CRC (0x4DFC)	MAGIC (\ )
6 Bytes	1 Byte	5 Bytes	1 Byte	1 Byte	6 Bytes	2 Bytes	2 Bytes	2 Bytes	

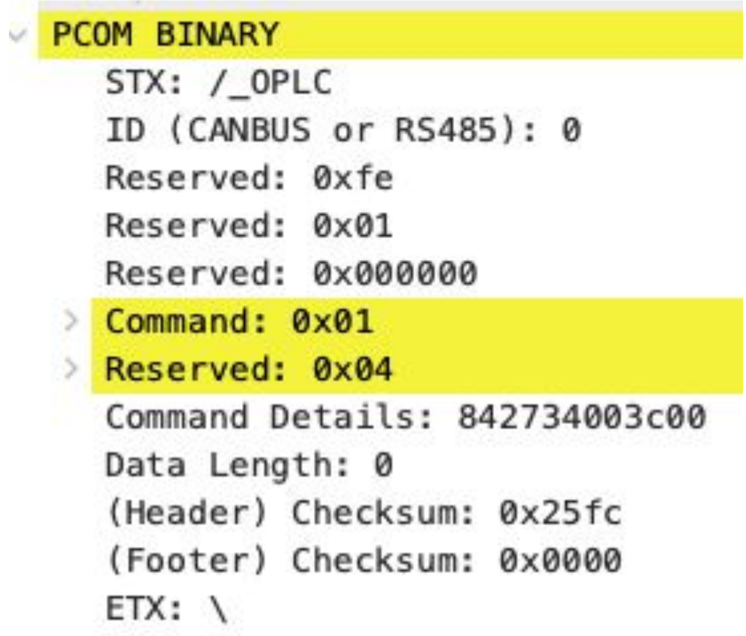
Opcode: 0x01

Reserved: 0xFE01000000

Reserved 2: 0x01 \ 0x04

Command Details:

- High 4 Bytes: Address (LE)
- Low 2 Bytes: Length (LE)



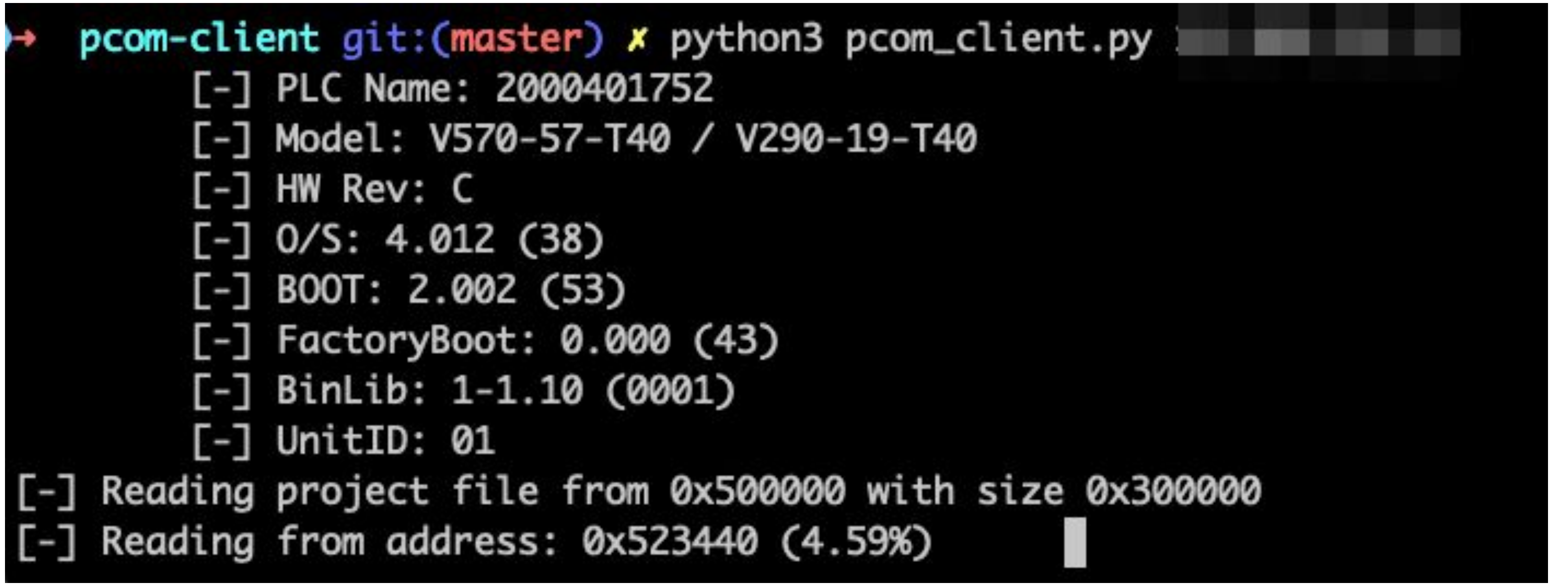
```
PCOM BINARY
  STX: /_OPLC
  ID (CANBUS or RS485): 0
  Reserved: 0xfe
  Reserved: 0x01
  Reserved: 0x000000
> Command: 0x01
> Reserved: 0x04
  Command Details: 842734003c00
  Data Length: 0
  (Header) Checksum: 0x25fc
  (Footer) Checksum: 0x0000
  ETX: \
```

PCOMClient - Capabilities

- ✓ Transport layer
    - Serial + TCP
- ✓ PCOM Flavors
    - Binary + ASCII
- ✓ Memory Read/Write

We Have Arbitrary Read: Now what?

- Dump entire memory region (RAM)
  - `0x00000000 - 0x00FFFFFF`
- Look for interesting sections
  - Strings
  - Opcodes
  - Structures
  - Resources



```
➜  pcom-client git:(master) ✗ python3 pcom_client.py [redacted]
        [-] PLC Name: 2000401752
        [-] Model: V570-57-T40 / V290-19-T40
        [-] HW Rev: C
        [-] O/S: 4.012 (38)
        [-] BOOT: 2.002 (53)
        [-] FactoryBoot: 0.000 (43)
        [-] BinLib: 1-1.10 (0001)
        [-] UnitID: 01
[-] Reading project file from 0x500000 with size 0x300000
[-] Reading from address: 0x523440 (4.59%)
```

Bad News: Some regions are protected

- Can't **WRITE** to some regions
    - write-protected (unwriteable memory)

- Can't **READ** from some regions
    - Return zeroed out memory + error

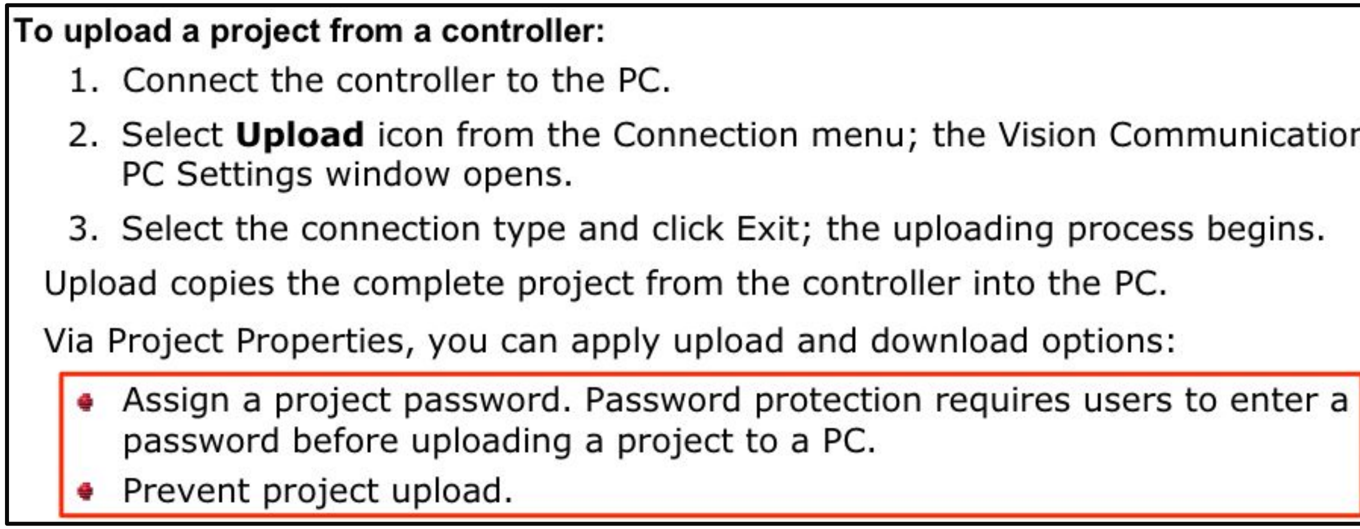
- What's in these memory regions???

Password Mechanism: Upload Password

- Program-related memory regions are protected

- Requires *Upload Password* to read them

- EWS *authenticates* using specific opcode



To upload a project from a controller:

1. Connect the controller to the PC.
2. Select **Upload** icon from the Connection menu; the Vision Communicatior PC Settings window opens.
3. Select the connection type and click Exit; the uploading process begins.

Upload copies the complete project from the controller into the PC.

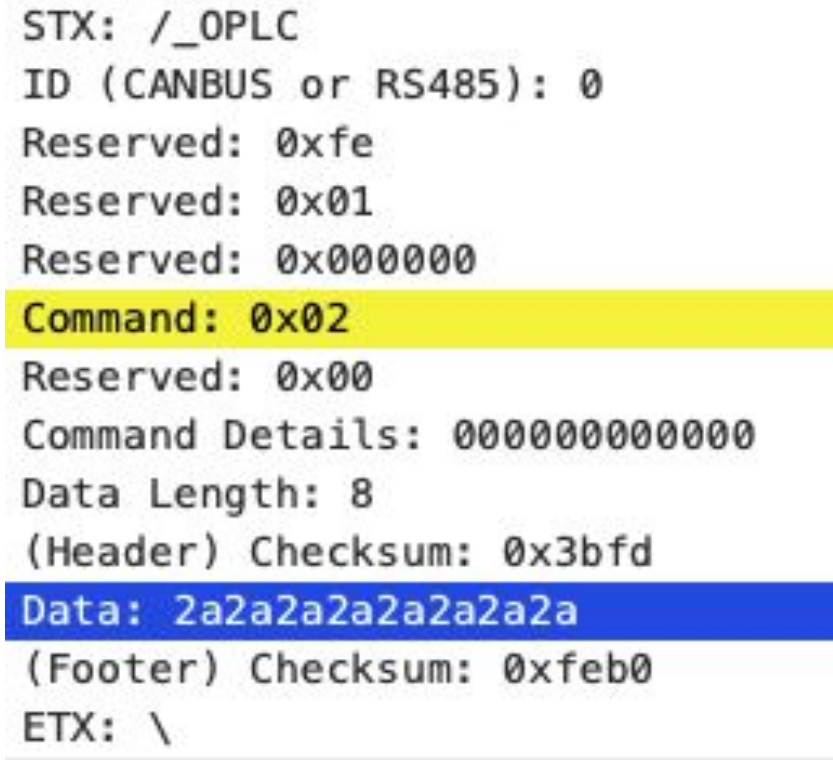
Via Project Properties, you can apply upload and download options:

- Assign a project password. Password protection requires users to enter a password before uploading a project to a PC.
- Prevent project upload.

Authenticate Memory Structure

MAGIC (/_OPLC)	ID (0x00)	Reserved (0xFE01000000)	Opcode (0x02)	Reserved (0x00)	Command Details (0x000000000000)	Length (0x0800)	Header CRC (0x4DFC)	Password (0xA2A2..A2)	Footer CRC (0x4DFC)	MAGIC (\ )
6 Bytes	1 Byte	5 Bytes	1 Byte	1 Byte	6 Bytes	2 Bytes	2 Bytes	8 Bytes	2 Bytes	

Opcode: 0x02



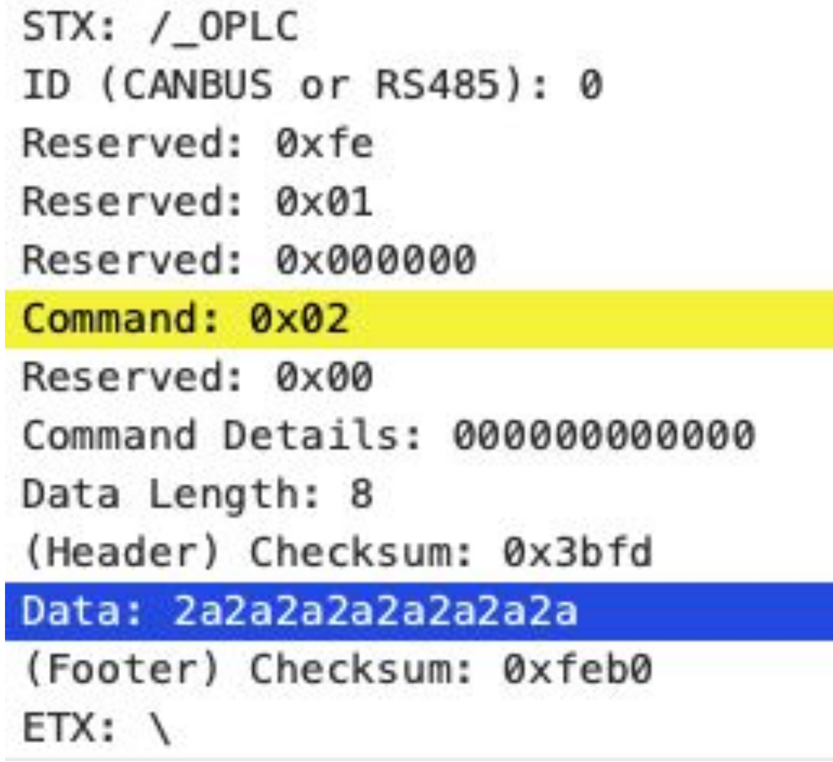
```
STX: /_OPLC
ID (CANBUS or RS485): 0
Reserved: 0xfe
Reserved: 0x01
Reserved: 0x000000
Command: 0x02
Reserved: 0x00
Command Details: 000000000000
Data Length: 8
(Header) Checksum: 0x3bfd
Data: 2a2a2a2a2a2a2a2a
(Footer) Checksum: 0xfeb0
ETX: \
```

Authenticate Memory Structure

MAGIC (/_OPLC)	ID (0x00)	Reserved (0xFE01000000)	Opcode (0x02)	Reserved (0x00)	Command Details (0x000000000000)	Length (0x0800)	Header CRC (0x4DFC)	Password (0xA2A2..A2)	Footer CRC (0x4DFC)	MAGIC (\ )
6 Bytes	1 Byte	5 Bytes	1 Byte	1 Byte	6 Bytes	2 Bytes	2 Bytes	8 Bytes	2 Bytes	

Opcode: 0x02

Data Length: 0x08 (Const - password length)



```
STX: /_OPLC
ID (CANBUS or RS485): 0
Reserved: 0xfe
Reserved: 0x01
Reserved: 0x000000
Command: 0x02
Reserved: 0x00
Command Details: 000000000000
Data Length: 8
(Header) Checksum: 0x3bfd
Data: 2a2a2a2a2a2a2a2a
(Footer) Checksum: 0xfeb0
ETX: \
```

Authenticate Memory Structure

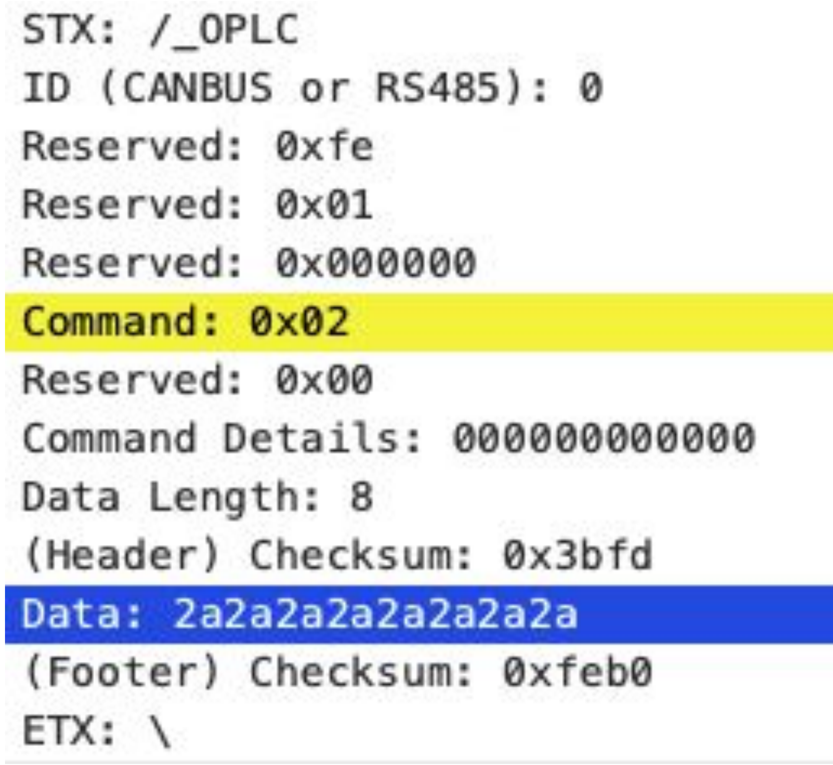
MAGIC (/_OPLC)	ID (0x00)	Reserved (0xFE01000000)	Opcode (0x02)	Reserved (0x00)	Command Details (0x000000000000)	Length (0x0800)	Header CRC (0x4DFC)	Password (0xA2A2..A2)	Footer CRC (0x4DFC)	MAGIC (\ )
6 Bytes	1 Byte	5 Bytes	1 Byte	1 Byte	6 Bytes	2 Bytes	2 Bytes	8 Bytes	2 Bytes	

Opcode: 0x02

Data Length: 0x08

Data: password

- charest: digits, asterisk
- length: 8 bytes (fixed)
- Default password: ******** (8 asterisks)



```
STX: /_OPLC
ID (CANBUS or RS485): 0
Reserved: 0xfe
Reserved: 0x01
Reserved: 0x000000
Command: 0x02
Reserved: 0x00
Command Details: 000000000000
Data Length: 8
(Header) Checksum: 0x3bfd
Data: 2a2a2a2a2a2a2a2a
(Footer) Checksum: 0xfeb0
ETX: \
```

PCOMClient - Capabilities

✓ Transport layer
  - Serial + TCP

✓ PCOM Flavors
  - Binary + ASCII

✓ Memory Read/Write

✓ Authentication

PCOM Function Codes - All supported in our tool!

Func Code Req / Resp	Desc
0x01 / 0x81	Read Memory
0x02 / 0x82	Check Password
0x0C / 0x8C	Get PLC Name
0x10 / 0x90	Find Resource
0x16 / 0x96	Translate Resource Index to Address
0x1A / 0x9A	Flush Memory Buf
0x41 / 0xC1	Write Memory

Func Code Req / Resp	Desc
0x42 / 0xC2	Reset Upload Password
0x4D / 0xCD	Read Operand
0xFF	Error
ID (ASCII)	Get PLC Version
UG (ASCII)	Get UnitID
GF (ASCII)	Read Integer
CSS (ASCII)	Stop PLC

Project Upload

- Some of the attacked PLCs were password protected
  - By attackers? before attack?
  - Who knows…

- Can we get the old project back?

- **Can we get the attacker's project???**
  - => Extract **TONS** of forensic evidence from project

Let’s Break the Upload Password!

Analyzing Upload Password

- There is a password reset process
  - rewrite the project + change password

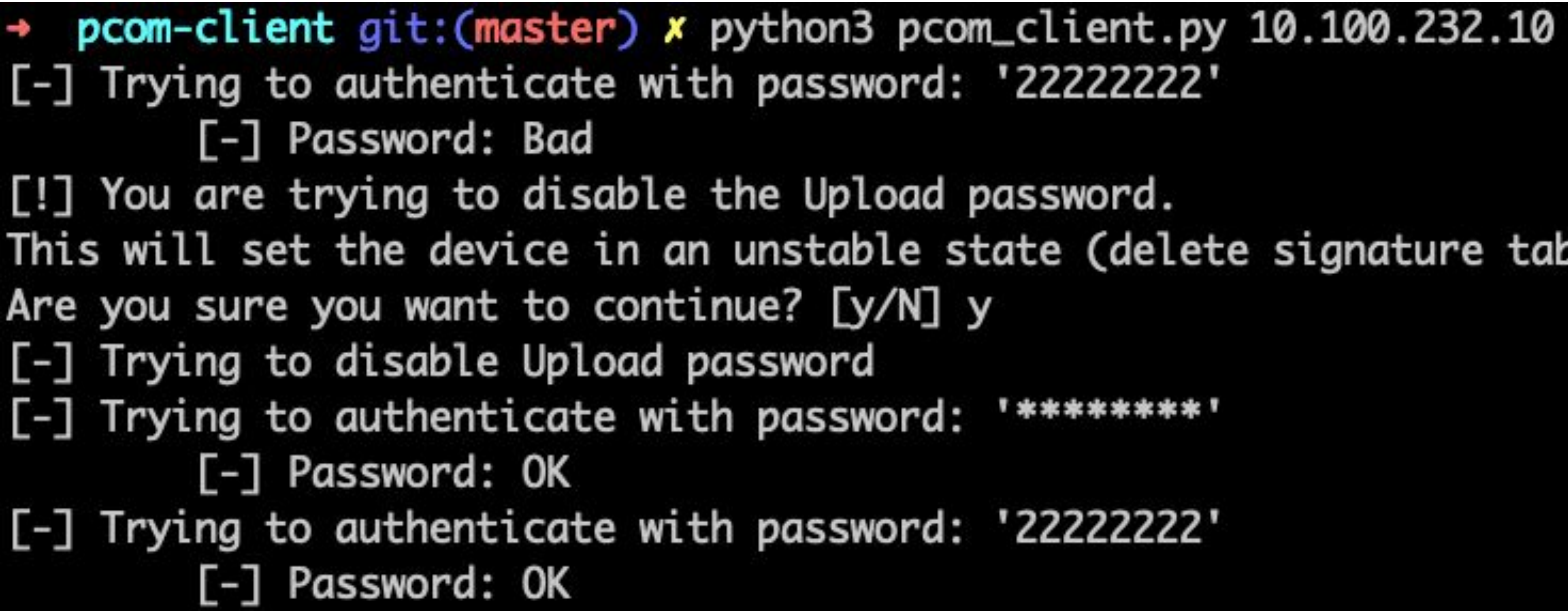
- We **don't** want to do that
  - Don't have old project
  - Don't want to overwrite evidence

- We found another technique!

Opcode: 0x42

After: ANY password will be accepted!

Password Reset Command (CVE-2024-38434)



```
➜  pcom-client git:(master) ✗ python3 pcom_client.py 10.100.232.10
[-] Trying to authenticate with password: '22222222'
        [-] Password: Bad
[!] You are trying to disable the Upload password.
This will set the device in an unstable state (delete signature tab
Are you sure you want to continue? [y/N] y
[-] Trying to disable Upload password
[-] Trying to authenticate with password: '*********'
        [-] Password: OK
[-] Trying to authenticate with password: '22222222'
        [-] Password: OK
```

PCOMClient: Capabilities

✓ Transport layer

  - Serial + TCP

✓ PCOM Flavors

  - Binary + ASCII

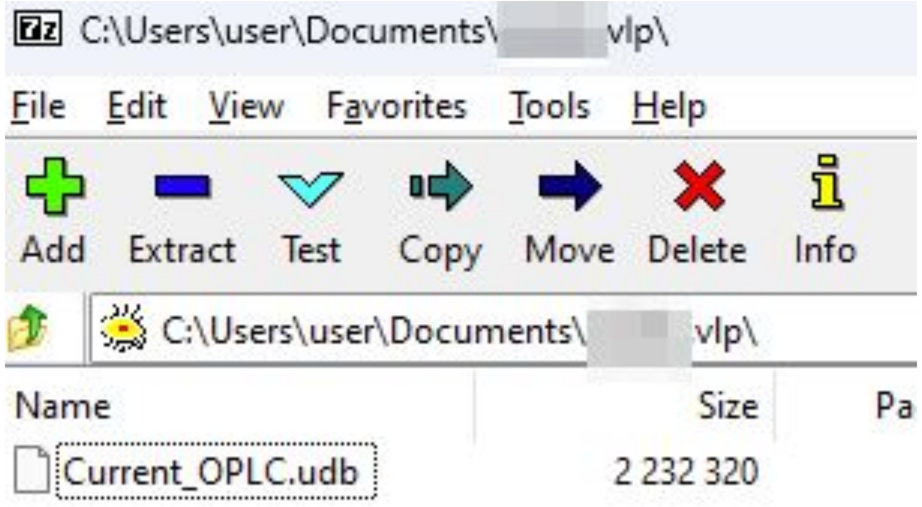
✓ Memory Read/Write

✓ Authentication

✓ Upload Password Bypass (CVE-2024-38434)

Unitronics Project File

- `access.db` database
- Containing all of the information related to the project
    - Functions
    - Assets
    - Metadata
- On PLC, saved as an encrypted ZIP



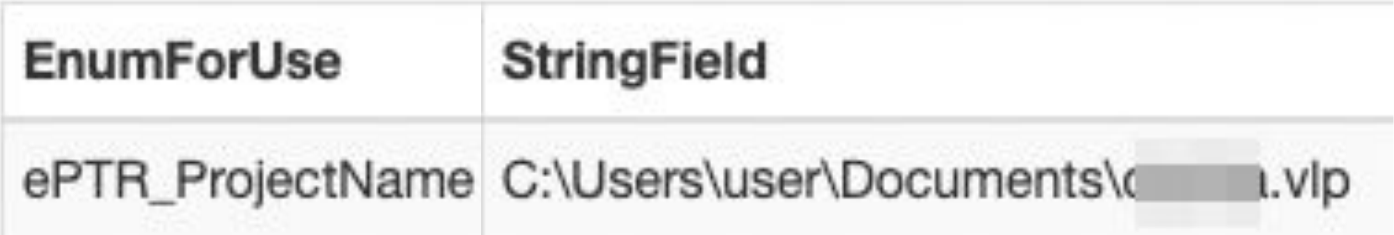
C:\Users\user\Documents\[redacted]vlp\

C:\Users\user\Documents\[redacted]vlp\

Name	Size	Pa
Current_OPLC.udb	2 232 320	

Project File: Forensic Evidence

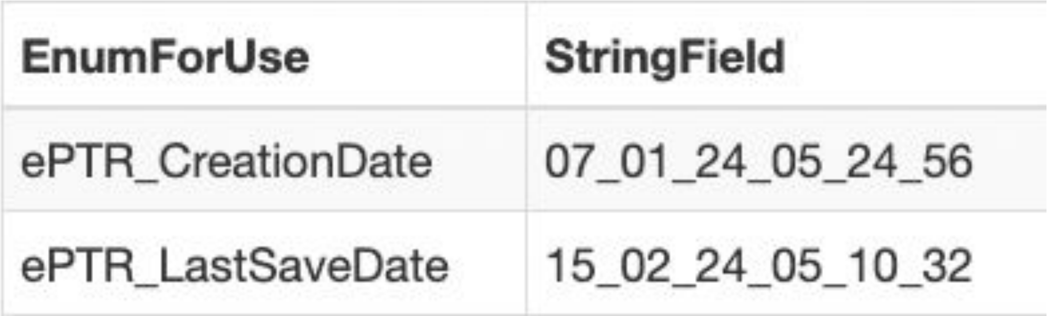
- Full project path
  - **Table**: `ProjectTable`
  - many times contains the username



EnumForUse	StringField
ePTR_ProjectName	C:\Users\user\Documents\[redacted].vlp

Project File: Forensic Evidence

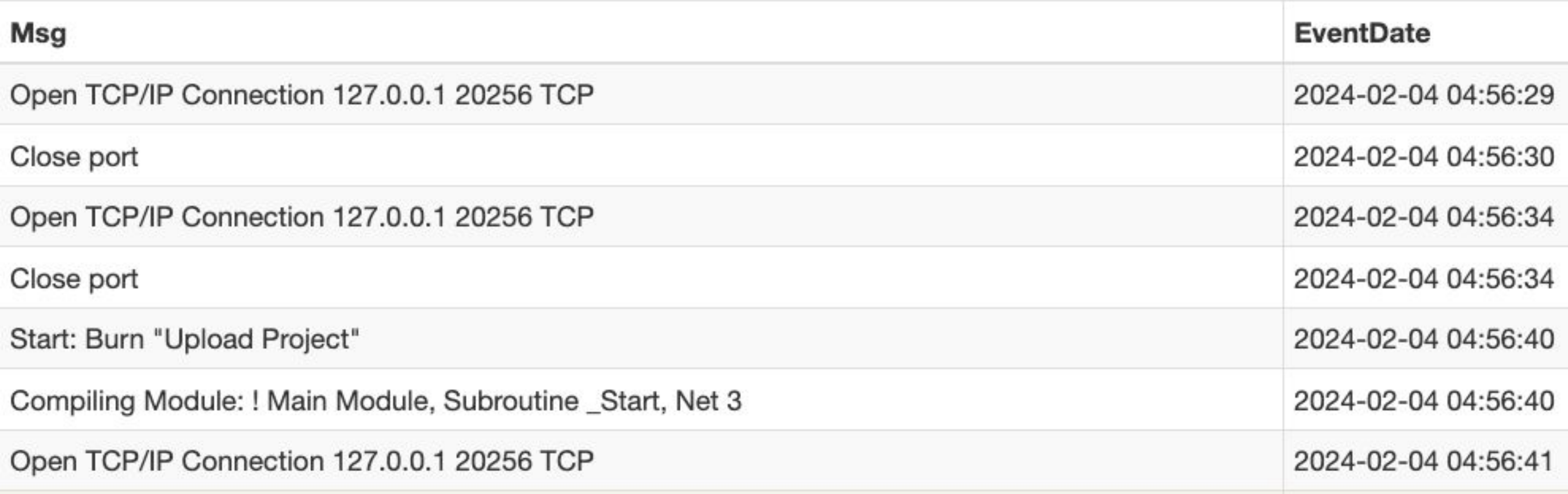
- Project Dates
  - **Table**: `ProjectTable`
  - project creation/modification dates



EnumForUse	StringField
ePTR_CreationDate	07_01_24_05_24_56
ePTR_LastSaveDate	15_02_24_05_10_32

Project File: Forensic Evidence

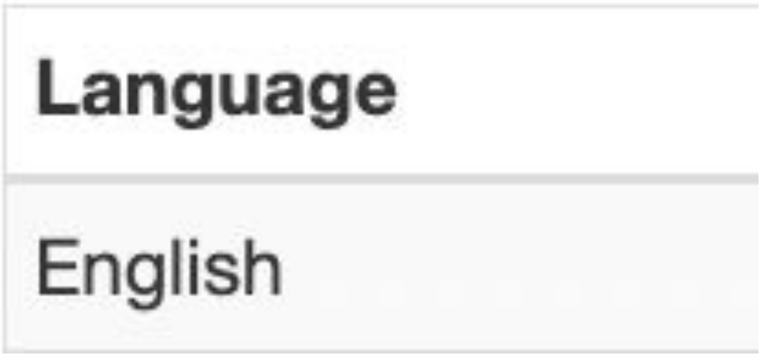
- Project Events
  - **Table**: `Events`
  - Events related to project (+ dates)



Msg	EventDate
Open TCP/IP Connection 127.0.0.1 20256 TCP	2024-02-04 04:56:29
Close port	2024-02-04 04:56:30
Open TCP/IP Connection 127.0.0.1 20256 TCP	2024-02-04 04:56:34
Close port	2024-02-04 04:56:34
Start: Burn "Upload Project"	2024-02-04 04:56:40
Compiling Module: ! Main Module, Subroutine _Start, Net 3	2024-02-04 04:56:40
Open TCP/IP Connection 127.0.0.1 20256 TCP	2024-02-04 04:56:41

Project File: Forensic Evidence

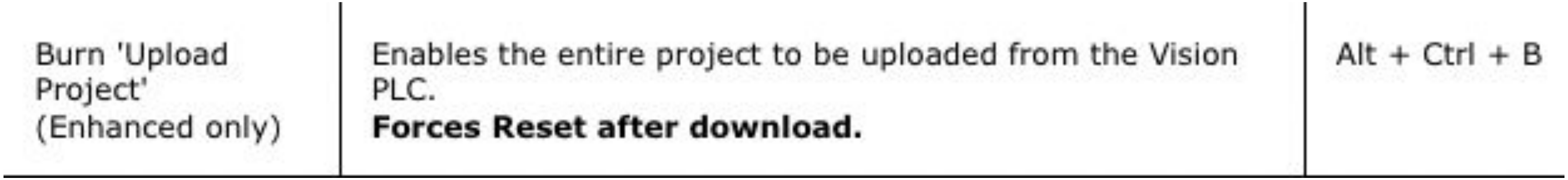
- Computer languages
  - **Table**: `tblKeyboards`
  - Languages installed on computer



Language
English

No Upload Project

- Attacker’s did not “burn” project
  - Download without enabling upload
- Can’t extract evidence



Burn 'Upload Project' (Enhanced only)	Enables the entire project to be uploaded from the Vision PLC. **Forces Reset after download.**	Alt + Ctrl + B

Signature Log: The answer to our prayers

- We discovered the signature log - unexpected forensic source
    - From strings, RE, documentations
- Everything that happened
    - Download/upload
    - Turn on/off
    - etc.
- Exactly what we need!

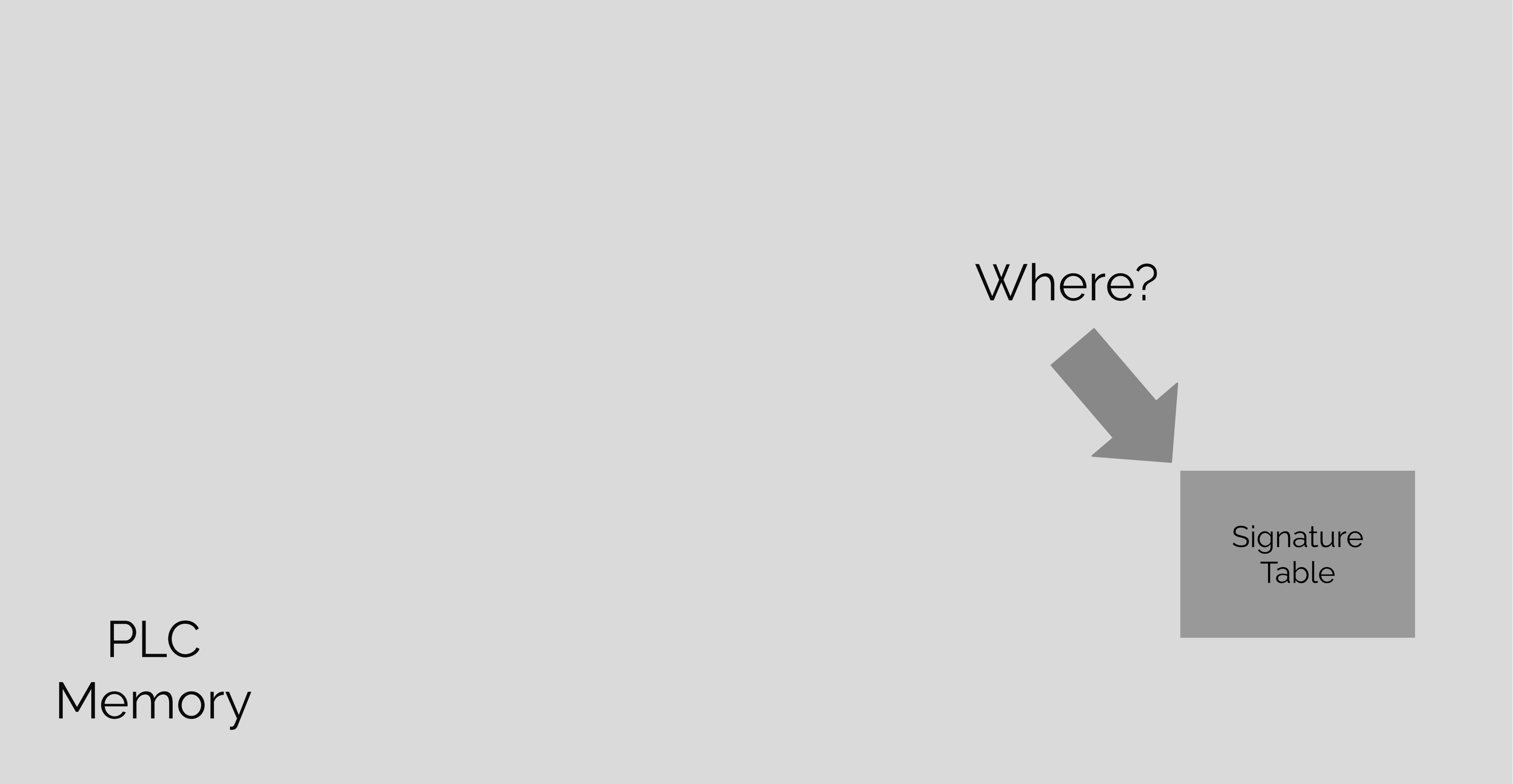
Our Goal: Read Signature Log

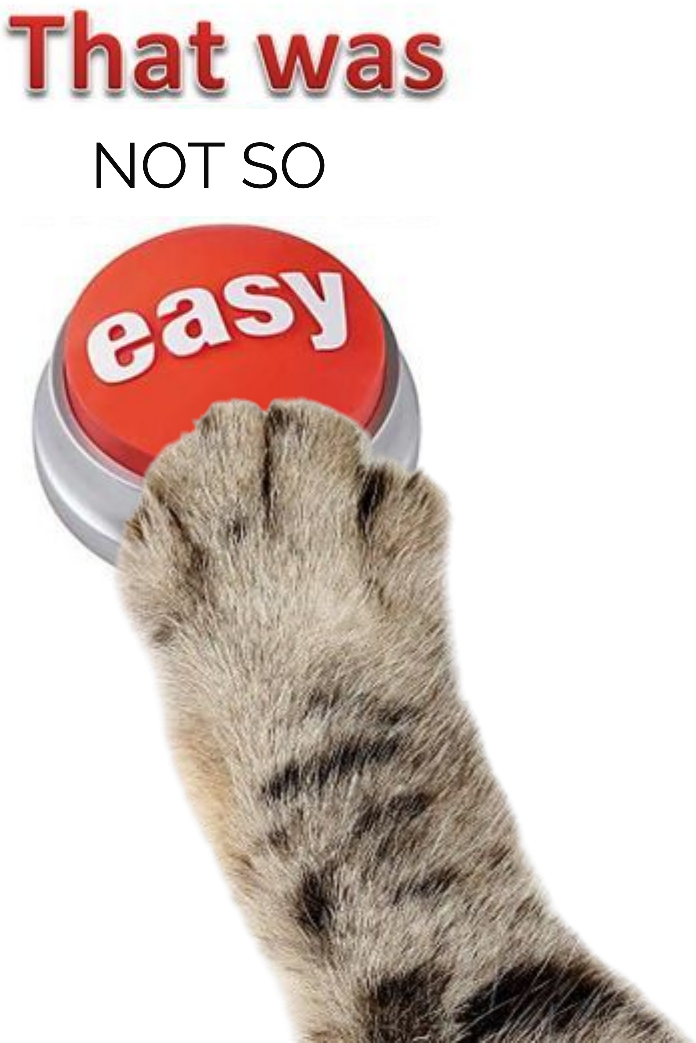
Our Goal: Read Signature Log

Signature Table

PLC Memory

Our Goal: Read Signature Log



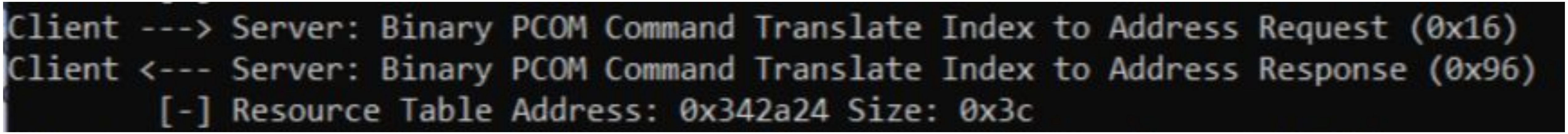


NOT SO

Step 1: Get Resource Table Address

Find resource table address

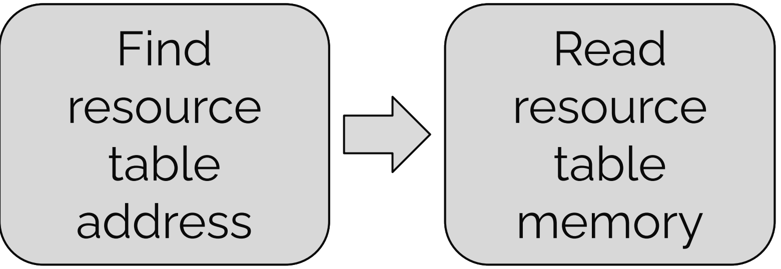
Opcode: 0x16



```
Client ---> Server: Binary PCOM Command Translate Index to Address Request (0x16)
Client <--- Server: Binary PCOM Command Translate Index to Address Response (0x96)
          [-] Resource Table Address: 0x342a24 Size: 0x3c
```

* Everything is my interpretation

Step 2 - Read Resource Table Address

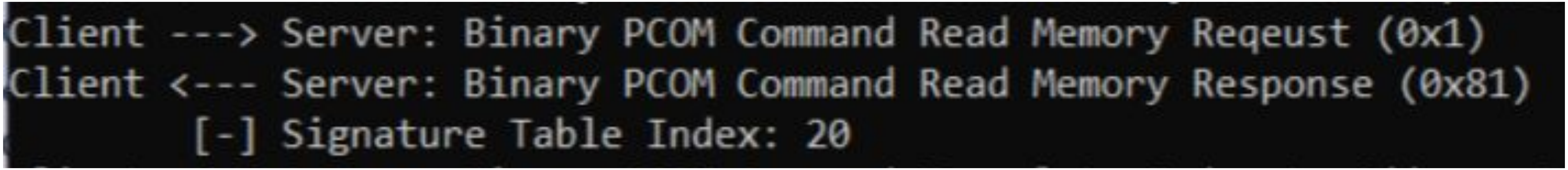


Find resource table address

Read resource table memory

Opcode: 0x16

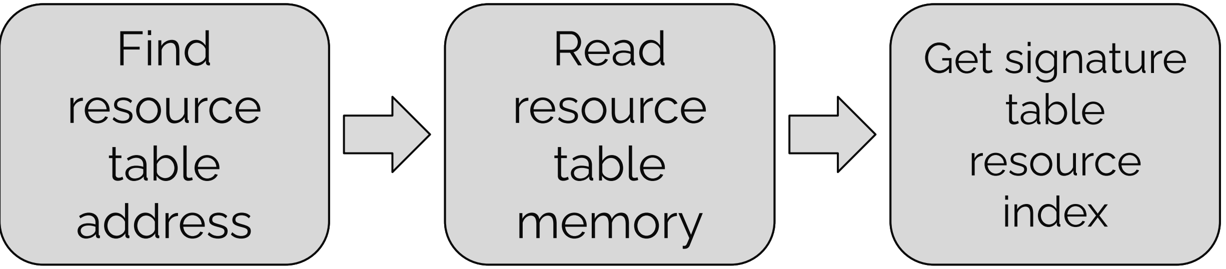
Opcode: 0x01



```
Client ---> Server: Binary PCOM Command Read Memory Reqeust (0x1)
Client <--- Server: Binary PCOM Command Read Memory Response (0x81)
        [-] Signature Table Index: 20
```

* Everything is my interpretation

Step 3: Get Signature Table Index From Resource Table



Find resource table address

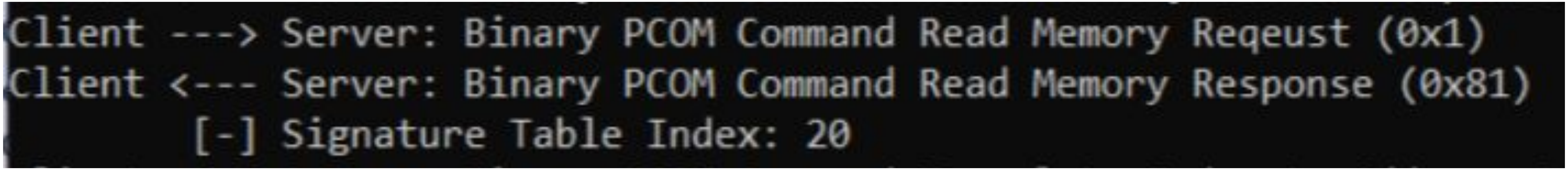
Read resource table memory

Get signature table resource index

Opcode: 0x16

Opcode: 0x01

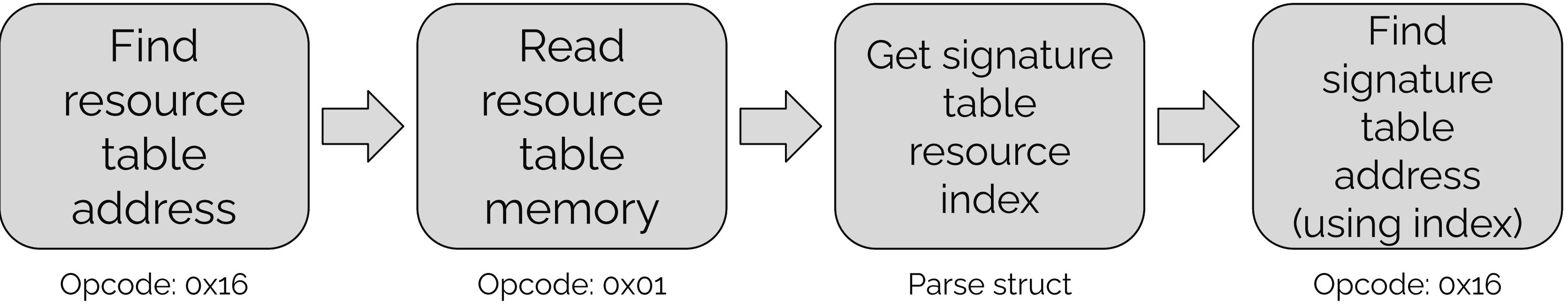
Parse struct



```
Client ---> Server: Binary PCOM Command Read Memory Reqeust (0x1)
Client <--- Server: Binary PCOM Command Read Memory Response (0x81)
        [-] Signature Table Index: 20
```

* Everything is my interpretation

Step 4: Get Signature Table Address



Find resource table address

Opcode: 0x16

Read resource table memory

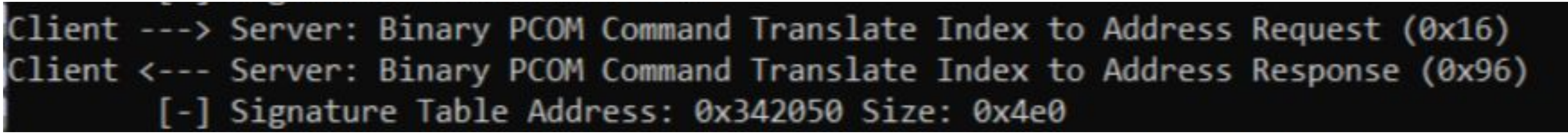
Opcode: 0x01

Get signature table resource index

Parse struct

Find signature table address (using index)

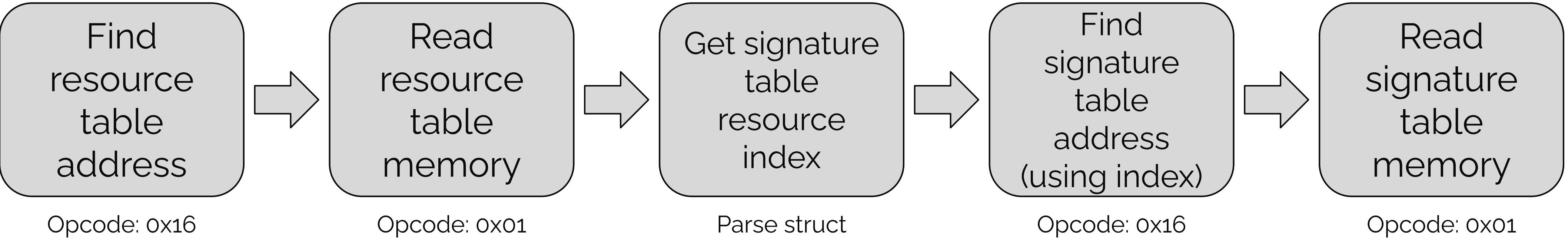
Opcode: 0x16



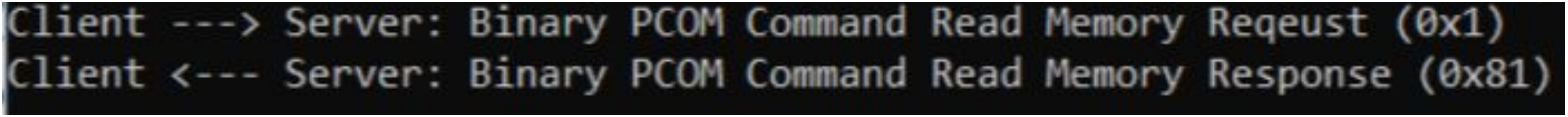
```
Client ---> Server: Binary PCOM Command Translate Index to Address Request (0x16)
Client <--- Server: Binary PCOM Command Translate Index to Address Response (0x96)
        [-] Signature Table Address: 0x342050 Size: 0x4e0
```

* Everything is my interpretation

Step 5: Read signature table address



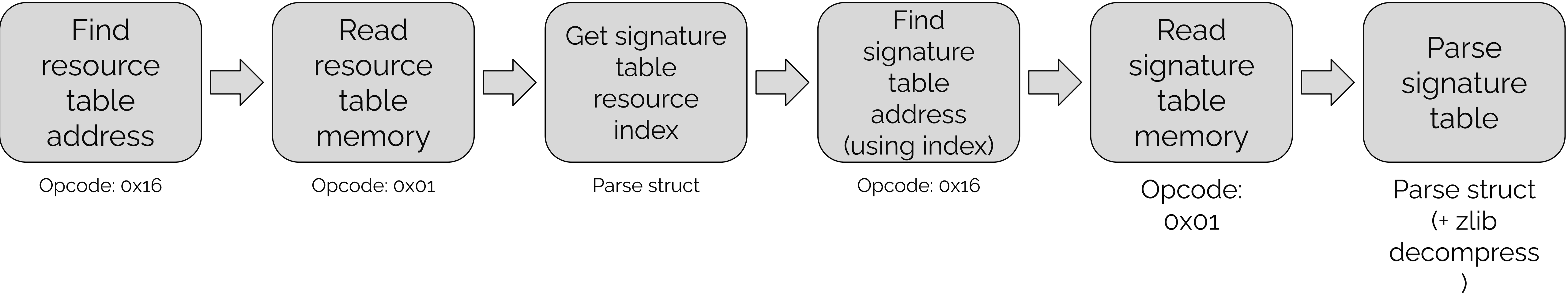
Find resource table address
Opcode: 0x16
Read resource table memory
Opcode: 0x01
Get signature table resource index
Parse struct
Find signature table address (using index)
Opcode: 0x16
Read signature table memory
Opcode: 0x01



```
Client ---> Server: Binary PCOM Command Read Memory Reqeust (0x1)
Client <--- Server: Binary PCOM Command Read Memory Response (0x81)
```

* Everything is my interpretation

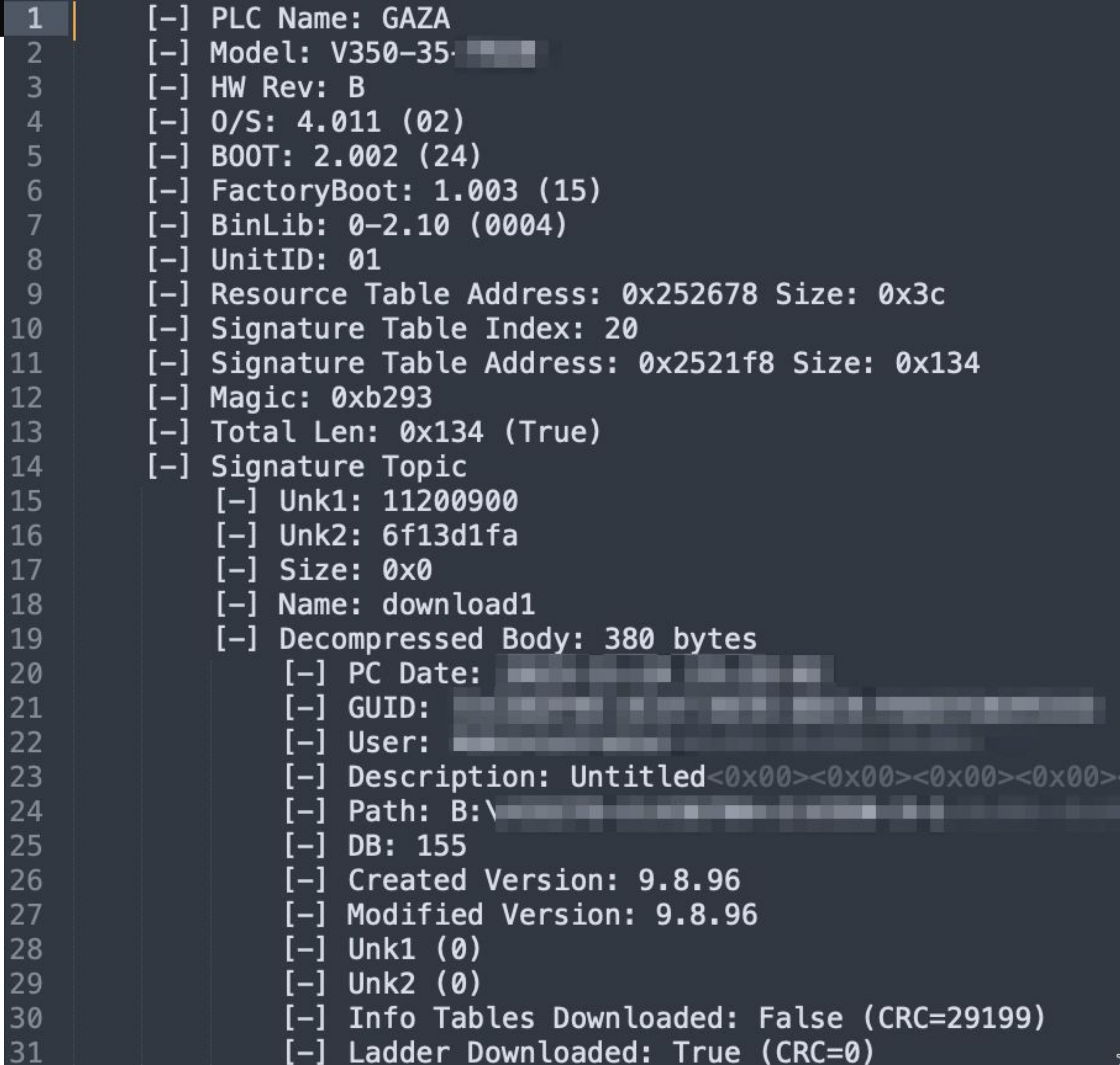
Step 6: Parse signature table



1. Find resource table address — Opcode: 0x16
2. Read resource table memory — Opcode: 0x01
3. Get signature table resource index — Parse struct
4. Find signature table address (using index) — Opcode: 0x16
5. Read signature table memory — Opcode: 0x01
6. Parse signature table — Parse struct (+ zlib decompress)

* Everything is my interpretation

Signature Log



```
[-] PLC Name: GAZA
[-] Model: V350-35-[redacted]
[-] HW Rev: B
[-] O/S: 4.011 (02)
[-] BOOT: 2.002 (24)
[-] FactoryBoot: 1.003 (15)
[-] BinLib: 0-2.10 (0004)
[-] UnitID: 01
[-] Resource Table Address: 0x252678 Size: 0x3c
[-] Signature Table Index: 20
[-] Signature Table Address: 0x2521f8 Size: 0x134
[-] Magic: 0xb293
[-] Total Len: 0x134 (True)
[-] Signature Topic
    [-] Unk1: 11200900
    [-] Unk2: 6f13d1fa
    [-] Size: 0x0
    [-] Name: download1
    [-] Decompressed Body: 380 bytes
        [-] PC Date: [redacted]
        [-] GUID: [redacted]
        [-] User: [redacted]
        [-] Description: Untitled<0x00><0x00><0x00><0x00>
        [-] Path: B:\[redacted]
        [-] DB: 155
        [-] Created Version: 9.8.96
        [-] Modified Version: 9.8.96
        [-] Unk1 (0)
        [-] Unk2 (0)
        [-] Info Tables Downloaded: False (CRC=29199)
        [-] Ladder Downloaded: True (CRC=0)
```

PCOMClient - Capabilities

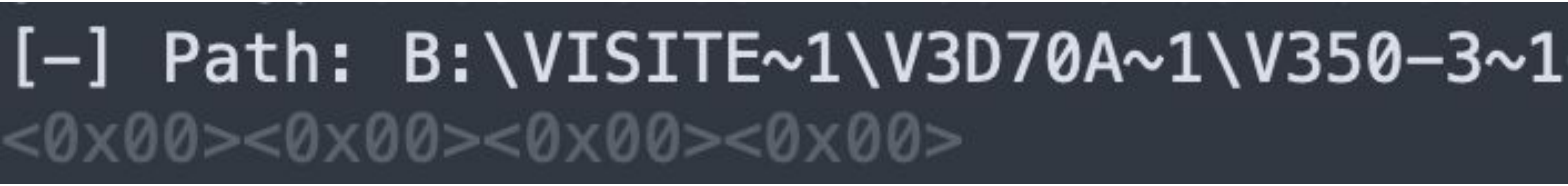
- ✓ Transport layer
  - Serial + TCP
- ✓ PCOM Flavors
  - Binary + ASCII
- ✓ Memory Read/Write

- ✓ Authentication

- ✓ Password Bypass (CVE-2024-38434)
- ✓ Signature Log fetcher + parser

Signature Log: Forensic Evidence

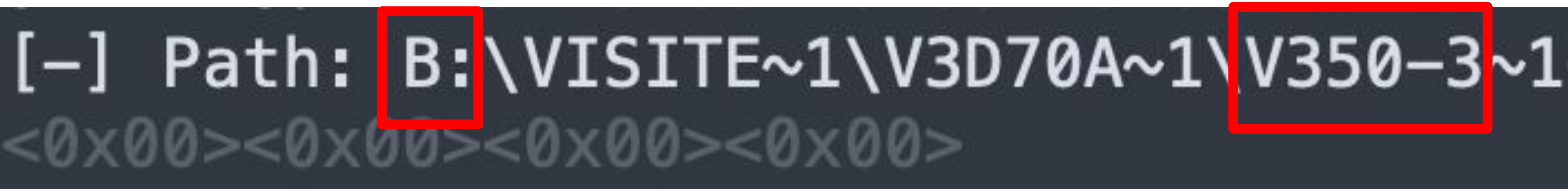
- **Project path**
  - Limited to 40 characters
  - Uses Windows short names
  - Usually contains username/path



```
[-] Path: B:\VISITE~1\V3D70A~1\V350-3~1
<0x00><0x00><0x00><0x00>
```

Signature Log: Forensic Evidence

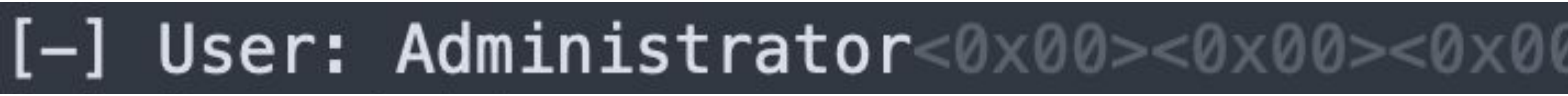
- **Project path**
  - Attackers used weird drive letter (B:/)
  - They created different projects for each device type



```
[-] Path: B:\VISITE~1\V3D70A~1\V350-3~1
<0x00><0x00><0x00><0x00>
```

Signature Log: Forensic Evidence

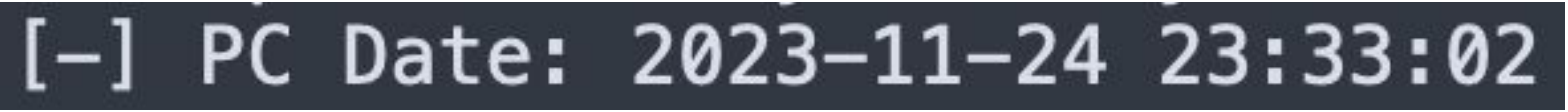
- **Username**
  - Limited to 16 characters



```
[-] User: Administrator<0x00><0x00><0x00
```

Signature Log: Forensic Evidence

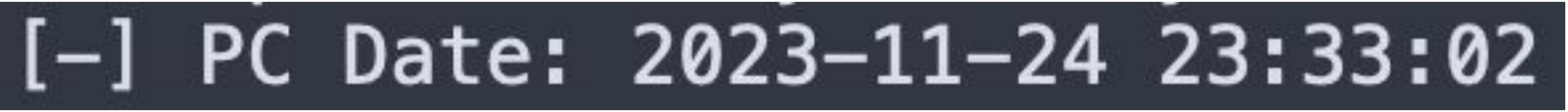
- **Connection Date**
    - From attacker's computer
    - Down to the second



```
[-] PC Date: 2023-11-24 23:33:02
```

Signature Log: Forensic Evidence

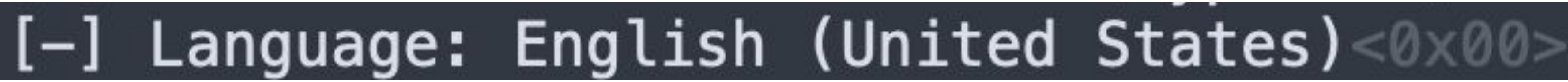
- **Connection Date**
  - Shows attacker's time zone
  - Can be used to correlate evidence from other sources (logs)



```
[-] PC Date: 2023-11-24 23:33:02
```

Signature Log: Forensic Evidence

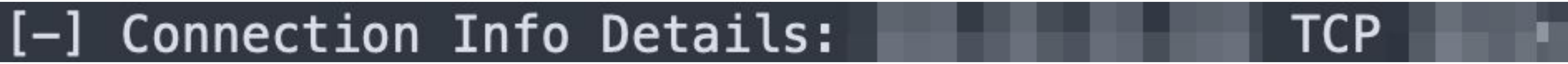
- **Keyboard Layout**
  - Taken from attacker's computer



```
[-] Language: English (United States)<0x00>
```

Signature Log: Forensic Evidence

- **Connection string**
  - IP/PORT used by attacker
  - Shows the target IP (tunneling/internet exposed device)

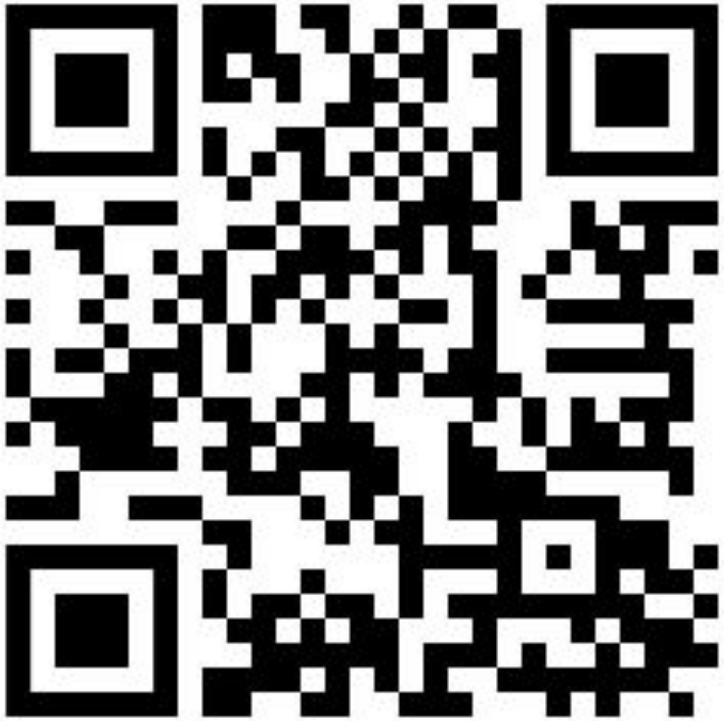


```
[-] Connection Info Details: [redacted] TCP [redacted]
```

Forensic Evidence

Forensic Evidence	Is Inside Signature Table	Is Inside Project File
Project Path	Yes	Yes
PC Username	Yes	No (could be in path)
Project File Creation Date	No	Yes
PLC Connection Dates	Yes	Yes
Computer Keyboards	Yes	Yes
PLC Connection String	Yes	Yes
Images used in Project File	No	Yes
Project Functions	No	Yes

Link To Project



* Help us by adding code to this project

Summary & Takeaways

- Sometimes - there are no IT logs

- Can't rely on vendors
  - Don't have the knowledge/motivation

- Community must require more logs from actual PLCs

- When all else fails - go to the community!
  - Develop community forensic tools

Thank you

CLAROTY

TEAM82