Blog

INDUSTROYER.V2: Old Malware Learns New Tricks

Daniel Kapellmann Zafra, Raymond Leong, Chris Sistrunk, Ken Proska, Corey Hildebrandt, Keith Lunden, Nathan Brubaker Apr 25, 2022 14 mins read Threat Intelligence ICS Operational Technology Malware Analysis

On April 12, 2022, CERT-UA and ESET reported that a cyber physical attack impacted operational technology (OT) supporting power grid operations in Ukraine. The attack leveraged different pieces of malware including a variant of INDUSTROYER, a well-known piece of attack-oriented ICS malware originally deployed in December 2016 to cause power outages in Ukraine.

The attack is significant not only because OT-targeted attacks are rare, but also because this is the first instance in which code from broadly known attack-oriented OT malware was redeployed against a new victim. Despite five years of substantial analysis into INDUSTROYER from a variety of researchers, the actor still attempted to repurpose the tool and customized it to reach new targets. INDUSTROYER.V2 (Mandiant's name for the new variant) reinforces the notion that OT malware can be tailored for use against multiple victims, which has serious implications for other publicly known OT malware families like INCONTROLLER.

While much of the story surrounding INDUSTROYER.V2's deployment is already publicly available, Mandiant further analyzed the malware to share additional insights for defenders and the OT community. In this blog post we document additional technical details of INDUSTROYER.V2 based on our analysis of two different samples. We also provide detection rules to identify related activity.

If you need support responding to related activity, please contact Mandiant Consulting. Further analysis of related threats——including additional malware that was deployed alongside INDUSTROYER.V2——is available as part of Mandiant Advantage Threat Intelligence.

INDUSTROYER.V2 In a Nutshell

INDUSTROYER.V2 is similar to its predecessor, however this variant contains more targeted functionality. Unlike the original INDUSTROYER, which was a framework that leveraged external modules to implement four different OT protocols, this variant is self-contained and only implements the IEC 60870-5-104 (IEC-104) communications protocol. IEC-104 is used for power system monitoring and control over TCP and is mainly implemented in Europe and the Middle East.

Most importantly, the new malware variant enables the actor to embed customized configurations that modify the malware's behavior to specific intelligent electronic devices (IEDs) (e.g., protection relays, merging units, etc.) within the target environment. The design change to embed custom configurations in INDUSTROYER.V2 reduces the effort required by the actor to reproduce the attack against different victim environments and enables the actor to contain the impact to specific targeted IEDs.

Two Custom INDUSTROYER.V2 Samples Show the Breadth of the Attack

To fully understand the implications of the new customization capabilities in INDUSTROYER.V2, we analyzed and compared two different samples. The second sample, which is likely a recompiled version, is publicly available and in online malware scanning platforms (MD5: 7c05da2e4612fca213430b6c93e76b06).

- We believe both samples are related to the same operation. The compilation timestamps were within minutes of each other and about two weeks before the intended attack. It is possible the actor was modifying the malware's configuration to customize the payload for different targets.
- Each sample contains different configurations hardcoded within the binary. One sample contains eight unique hardcoded target IP addresses, whereas the other only contains three.
- In both samples the malware terminated a specific process. However, the defined filepath used to concatenate with the process differed between the two samples. This shows a nuanced understanding of the victim environment.
- Figure 1 shows an example of INDUSTROYER.V2 configuration for the publicly available sample.

Figure 1: Example INDUSTROYER.V2 configuration for publicly available sample

Based on the slight differences between both samples, we can infer additional details about the scale of the attack, the likely level of access that the actor had within the victim networks, and the reconnaissance likely performed by the attacker prior to the deployment of the malware.

- As shown by the details embedded in the malware configurations, the actor conducted at least some internal network reconnaissance to identify specific IEDs in the victim environments and understand how to access them.
- The malware configurations target devices across specific subnets, highlighting that the actor succeeded in identifying and penetrating surrounding networks.
- The actor's successful implementation of IEC-104 to interact with the targeted devices indicates a robust understanding of the protocol and knowledge of the victim environment. For example, in the samples we analyzed the actor manipulated a selected list of Information Object Addresses (IOAs), which are used to interact with power line switches or circuit breakers in a remote terminal unit (RTU) or relay configuration.
- Conversely, the malware code itself shows some degree of carelessness or potential time constraints. For example, the INDUSTROYER.V2 samples contain limited obfuscation and defense evasion methods. The lack of obfuscation in the binaries provides defenders with quick hints on its functionality and ability to target OT assets.

Outlook

Extensible frameworks, such as the original INDUSTROYER and INCONTROLLER, are often preferred by threat actors due to the flexibility of their modular design, allowing deployment of specific payloads to target different victim assets or communication protocols. However, in the case of INDUSTROYER.V2, the actor reimplemented only one of the original components from the earlier framework and created a new self-contained executable.

It is unclear why the threat actor made the particular modifications to INDUSTROYER.V2. Perhaps the actor wanted to develop a more streamlined version to target a very specific environment, or they did not want to expose more valuable or capable tools, or they simply believed this approach would be efficient since it would not require additional resources to impact the target.

Regardless of the motivations, the reuse of code from known OT malware highlights the value of hunting and detections based on known indicators. For instance, some detections we built for the original INDUSTROYER successfully identified INDUSTROYER.V2 in the wild. While it is often believed that OT malware is not likely to be utilized in more than one environment, tools that take advantage of insecure by design OT features——such as INDUSTROYER.V2 does——can be employed multiple times to target multiple victims. The OT security community should recognize these tools as frameworks or capabilities, and not merely features of isolated cyber security incidents and one time use tools.

Technical Analysis of INDUSTROYER.V2

INDUSTROYER.V2 is written in C++ and implements the IEC-104 protocol to modify the state of remote terminal units (RTUs) over TCP. IEC-104 protocol TCP clients are called control stations and the TCP servers are called remote stations. The malware crafts configurable IEC-104 Application Service Data Unit (ASDU) messages, also known as telegrams, to change the state of a remote station's Information Object Addresses (IOAs) to ON or OFF. IOAs identify a specific data element on a device and may correspond to power line switches or circuit breakers in an RTU or relay configuration. The malware is a self-contained executable where the operator can set up configuration parameters to target specific remote stations, define execution options, and craft ASDU messages. It also accepts the optional command line arguments (-o) to print debug messages to an output file or (-t) to create a time delay before execution.

Configuration Capabilities

After the command line interface arguments are parsed, INDUSTROYER.V2 iterates through embedded configuration entries. The execution of the program is highly configurable. Based on our analysis of two INDUSTROYER.V2 samples, the malware contains configuration entries structured as strings in the order shown in Table 1.

Table 1: Configuration Structure

Position	Configuration Entry Description
1	Station IP Address
2	Station Port

3 Entry Index Value

4 Enable Hard-Coded Telegrams with specified Range

5 Enable Configuration Options 6 - 14

5b If Entry 4 is Enabled, Telegram Start Range

6 Enable Process Termination

6b If entry 4 is Enabled, Telegram End Range

7 Process Name to Terminate

8 Enable File Rename

9 Directory Path for File Rename

10 Sleep Prior to IEC-104 functionality

11 Sleep Duration Seed Value

12 Execution Control

13 Sleep Duration Seed Value

14 Unused

15 Command State — ON/OFF

16 Change Option — Send Inverted ON/OFF commands

17 Number of ASDU Data Entries

18 First ASDU Data Entry

An example extracted configuration entry from sample 7c05da2e4612fca213430b6c93e76b06 is presented in Figure 2.

Figure 2: Example configuration entry

```
192.168.XXX.XXX 2404 2 0 1 1 Example StoppedProcess.exe 1 "Example PATH" 0 1 0 0 1 0 0 8 1104 0 0 0 1 1
1105 0 0 0 1 2 1106 0 0 0 1 3 1107 0 0 0 1 4 1108 0 0 0 1 5 1101 0 0 0 1 6 1102 0 0 0 1 7 1103 0 0 0 1 8
```

If configuration entry #4 is enabled, the malware crafts ASDU telegrams to deliver Select and Execute commands and modify the state of a remote station's IOA to OFF. The IOA ranges which these telegrams are sent to are provided by configuration entries #5 and #6. However, this option was not enabled in recovered samples. A configuration mapping against the example in Figure 2 is included in the Appendix.

All the configurations we examined had the following options enabled: process termination, file rename, and use of ASDU data entries. The ASDU data entries are used to craft specific ASDU messages to the remote station and the data entry is structured in the format shown in Table 2.

Table 2: ASDU Entry Structure

Position	ASDU Data Entry Description
1	Station Information Object Address (IOA)
2	Set Message Type — Single or Double Command
3	Set Command Type — Select or Execute
4	Invert Default ON/OFF State
5	Execution Control
6	ASDU Entry Index

After parsing each configuration entry, INDUSTROYER.V2 enumerates running processes to identify if a specific hard-coded process is running and terminates it. Once this process is stopped, the malware enumerates running processes again and terminates whichever process is specified by the operator in the configuration.

If the file rename option is enabled, the malware creates a file path using its configuration data and adds a .MZ extension to this file. This may be a technique to prevent the specified process terminated earlier from relaunching.

For each configuration entry, a thread is created that implements IEC-104 communication with the controlled system. IEC-104 uses the Application Protocol Data Unit (APDU) specification.

An APDU frame can be composed of either just an Application Protocol Control Information (APCI) frame; or an APCI header and a subsequent Application Service Data Unit (ASDU) frame.

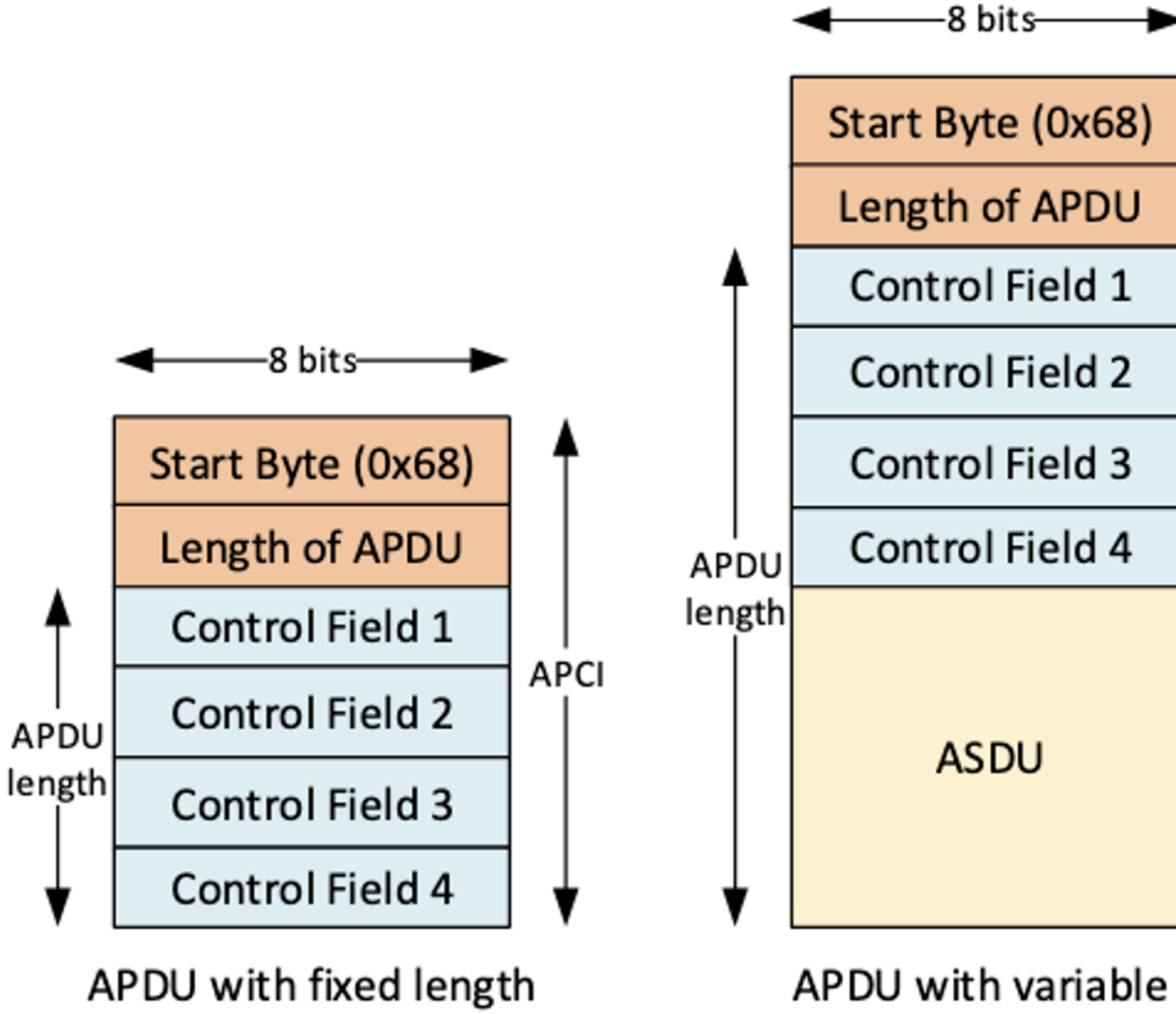

Figure 3: APDU frame format by Brno University of Technology

Execution

INDUSTROYER.V2 first sends control function messages, which are contained within an APCI frame. The first control message is a Test Frame (TESTFR). The malware sends a TESTFR ACT to the remote station which verifies an established connection. If one exists, a remote station responds with a corresponding TESTFR CON.

Next, the malware opens a data transfer channel with the remote station using a subsequent control message type of Start Data Transfer (STARTDT). By default, data transfer is not enabled on an active connection between a control station and remote station. Therefore, the malware sends a STARTDT ACT to activate a data transfer channel and a remote station sends a corresponding STARTDT CON, to confirm a successful activation.

With data transfer enabled, the malware utilizes an ASDU frame to send subsequent commands to the remote station. ASDU messages, also known as telegrams, are a set of application functions defined by IEC-104 to monitor and control remote stations. Further information describing the ASDU frame is available for reference here.

The malware sends a General Interrogation command, which allows it to obtain the current status of monitored digital and analog signals of the remote station. The malware then uses embedded ASDU data entries to craft a specific command to modify the target's IOA to either ON or OFF.

These commands are crafted using options defined within its configuration and the individual ASDU data entry. For example, in the configuration we extracted from sample 7c05da2e4612fca213430b6c93e76b06 (presented in Figure 2) the first ASDU data entry is:

- 1104 0 0 0 1 1

Based on configuration entry 15 (OFF state), entry 16 (Disable Change option), and its ASDU entry values (described in Table 2), the malware crafts an ASDU packet with the following characteristics:

- Information Object Address: 1104
- ASDU message type: C-DC_NA_1 (Double Command)
- ASDU command type: Execute
- Set state value: OFF

The ASDU message is shown in decoded network traffic in Figure 4.

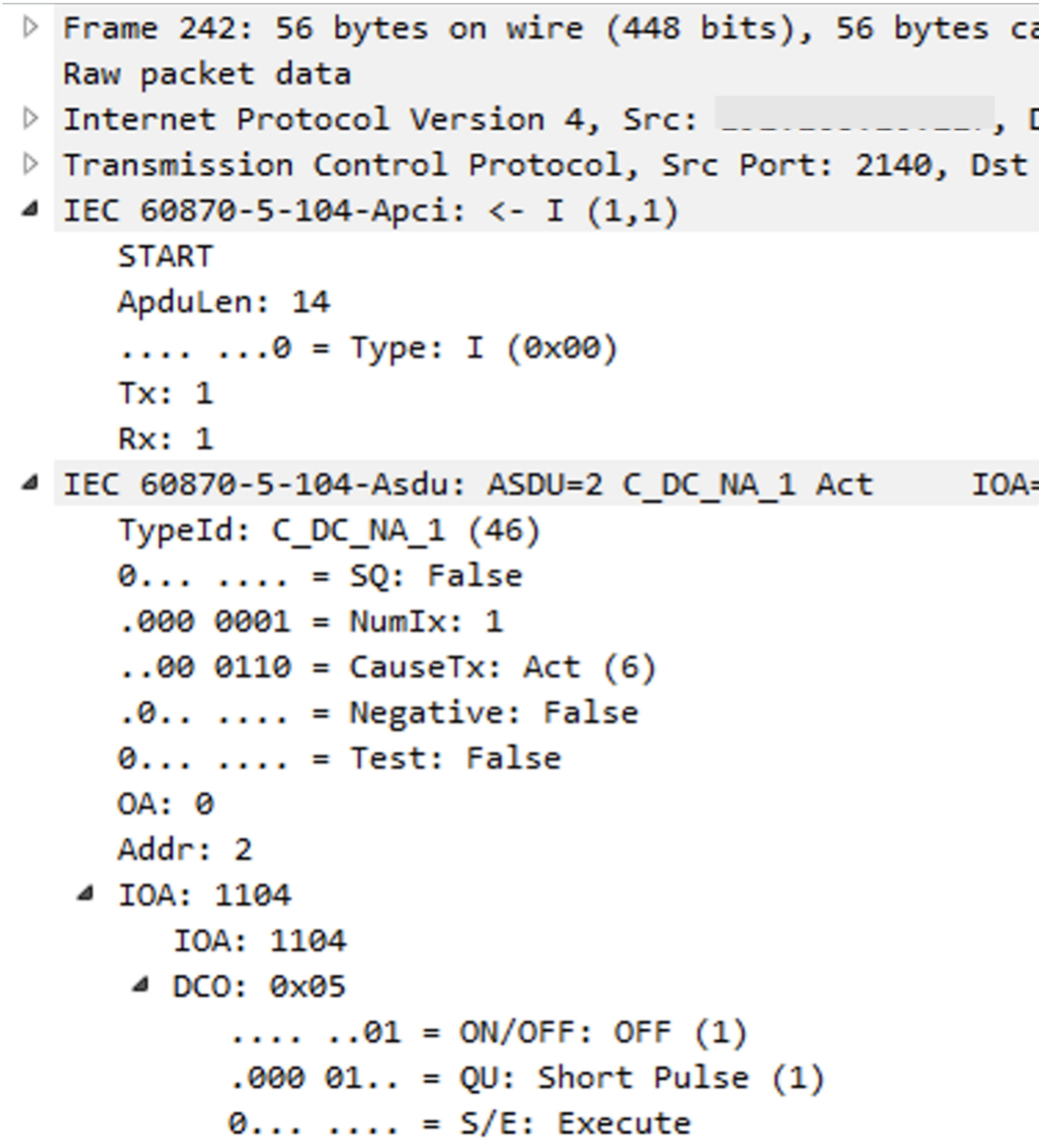

Figure 4: Crafted ASDU message

For each targeted remote station in a configuration entry, the malware iterates through corresponding ASDU data entries, crafting specified telegrams, and sends it to the remote station. The malware's configuration settings may direct it to craft an additional ASDU message, which inverts the ON/OFF state in a command and sends this additional message to the remote station's IOA.

A high-level description of the communication sequence is the following:

1. Send Test Frame messages to verify an established connection
2. Send Start Data Transfer messages to open a data transfer channel
3. Send a General Interrogation command retrieving the status of the remote station
4. Send crafted Single or Double command types to modify the state of the remote station's IOA

Figure 5 illustrates the described message sequence, captured between INDUSTROYER.V2 and a lab emulated IEC-104 remote station. It displays the initial TESTFR, STARTDT, and Interrogation commands, followed by the crafted ASDU commands delivered to specific remote station IOAs.

```
--> 68 04 43 00 00 00                  --> 16:16:22 TESTFR ACT Request  Length 4 [TESTFR_ACT]
<-- 68 04 83 00 00 00                  <-- 16:16:22 TESTFR CON Response  Length 4 [TESTFR_CON]
--> 68 04 07 00 00 00                  --> 16:16:23 STARTDT ACT Request  Length 4 [STARTDT_ACT]
<-- 68 04 0B 00 00 00                  <-- 16:16:23 STARTDT CON Response  Length 4 [STARTDT_CON]
--> 68 0E 00 00 00 00                  --> 16:16:24 Interrogation Request  Length 14 I NS:0 NR:0
    64 01 06 00 03 00 00 00 00 14          ASDU: Type ID 100 <Interrogation command> Count 1 Cause of transmission
<-- 68 0E 00 00 02 00                  <-- 16:16:24 Interrogation Response  Length 14 I NS:0 NR:1
    64 01 07 00 03 00 00 00 00 14          ASDU: Type ID 100 <Interrogation command> Count 1 Cause of transmission
<-- 68 0E 02 00 02 00                  <-- 16:16:24 Interrogation Response  Length 14 I NS:1 NR:1
    64 01 0A 00 03 00 00 00 00 14          ASDU: Type ID 100 <Interrogation command> Count 1 Cause of transmission
--> 68 04 01 00 04 00                  --> 16:16:27 Supervisory Request  Length 4 S NR:2
--> 68 0E 02 00 04 00                  --> 16:16:28 Single Cmd Execute Request  Length 14 I NS:1 NR:2
    2D 01 06 00 03 00 9A FC 01 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  6 <Acti
<-- 68 0E 04 00 04 00                  <-- 16:16:28 Single Cmd Execute Response  Length 14 I NS:2 NR:2
    2D 01 07 00 03 00 9A FC 01 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  7 <Acti
<-- 68 0E 06 00 04 00                  <-- 16:16:28 Single Cmd Execute Response  Length 14 I NS:3 NR:2
    2D 01 0A 00 03 00 9A FC 01 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  10 <Act
--> 68 04 01 00 08 00                  --> 16:16:30 Supervisory Request  Length 4 S NR:4
--> 68 0E 04 00 08 00                  --> 16:16:31 Single Cmd Execute Request  Length 14 I NS:2 NR:4
    2D 01 06 00 03 00 99 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  6 <Acti
<-- 68 0E 08 00 06 00                  <-- 16:16:31 Single Cmd Execute Response  Length 14 I NS:4 NR:3
    2D 01 07 00 03 00 99 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  7 <Acti
<-- 68 0E 0A 00 06 00                  <-- 16:16:31 Single Cmd Execute Response  Length 14 I NS:5 NR:3
    2D 01 0A 00 03 00 99 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  10 <Act
--> 68 04 01 00 0C 00                  --> 16:16:33 Supervisory Request  Length 4 S NR:6
--> 68 0E 06 00 0C 00                  --> 16:16:35 Single Cmd Execute Request  Length 14 I NS:3 NR:6
    2D 01 06 00 03 00 9B 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  6 <Acti
<-- 68 0E 0C 00 08 00                  <-- 16:16:35 Single Cmd Execute Response  Length 14 I NS:6 NR:4
    2D 01 07 00 03 00 9B 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  7 <Acti
<-- 68 0E 0E 00 08 00                  <-- 16:16:35 Single Cmd Execute Response  Length 14 I NS:7 NR:4
    2D 01 0A 00 03 00 9B 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  10 <Act
--> 68 04 01 00 10 00                  --> 16:16:37 Supervisory Request  Length 4 S NR:8
--> 68 0E 08 00 10 00                  --> 16:16:38 Single Cmd Execute Request  Length 14 I NS:4 NR:8
    2D 01 06 00 03 00 9C 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  6 <Acti
<-- 68 0E 10 00 0A 00                  <-- 16:16:38 Single Cmd Execute Response  Length 14 I NS:8 NR:5
    2D 01 07 00 03 00 9C 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  7 <Acti
<-- 68 0E 12 00 0A 00                  <-- 16:16:38 Single Cmd Execute Response  Length 14 I NS:9 NR:5
    2D 01 0A 00 03 00 9C 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  10 <Act
--> 68 04 01 00 14 00                  --> 16:16:40 Supervisory Request  Length 4 S NR:10
--> 68 0E 0A 00 14 00                  --> 16:16:42 Single Cmd Execute Request  Length 14 I NS:5 NR:10
    2D 01 06 00 03 00 9D 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  6 <Acti
<-- 68 0E 14 00 0C 00                  <-- 16:16:42 Single Cmd Execute Response  Length 14 I NS:10 NR:6
    2D 01 07 00 03 00 9D 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  7 <Acti
<-- 68 0E 16 00 0C 00                  <-- 16:16:42 Single Cmd Execute Response  Length 14 I NS:11 NR:6
    2D 01 0A 00 03 00 9D 74 02 05          ASDU: Type ID 45 <Single command> Count 1 Cause of transmission  10 <Act
```

Figure 5: INDUSTROYER.V2 message sequence with emulated IEC-104 Remote Station

The detailed nature of how a specific remote station is targeted, down to the unique IOAs and the state each IOA must be modified to create an intended effect, demonstrates a comprehensive understanding of, or visibility into the victim environment.

We note that although the IOAs targeted by the malware can provide important context to the actor's precise intent, IOA mappings often differ between manufacturers, devices, and even users. For this reason, accurate interpretation of the actions intended by the actor requires additional knowledge about the targeted assets.

While at this moment we do not have such information, we explored possible IOA matches based on publicly available documentation about specific products. For example, knowing that ABB RTU's and relays are heavily deployed in the targeted region we performed open source analysis. In our earlier example we observed an IOA equivalent to 1104, which we then mapped to public product documentation from an ABB Distribution Recloser Relay corresponding to "50BFT:InStr status".

In this status, 50 is the ANSI number for circuit breaker, which is how relay elements are numbered when setting a protection relay. Then 50BFT stands for circuit breaker failure protection. We provide an appendix illustrating additional mapping of IOAs extracted from the configuration of the public INDUSTROYER.V2 sample against the same distribution recloser relay.

Overlaps With Previous INDUSTROYER Variant

The two versions of INDUSTROYER contain overlaps in code and similarities in execution flow and functionality. We identified the following shared features in execution and functionality:

- Both versions contain code that first terminates a specific process on the target remote station, prior to establishing IEC-104 communication.
- Both versions craft specific ASDU messages according to provided configuration settings.
- Both versions contain an ability to deliver pre-defined ASDU messages to a specified IOA range.
- Both versions contain an option to direct the malware to craft an additional ASDU message which inverts the previous ON/OFF command and sends it to the target remote station.

One difference we identified between both variants is that, unlike its predecessor, INDUSTROYER.V2 contains altered debugging messages which obfuscate the meaning of the outputs. However, we note these debugging messages are formatted and printed at similar execution points of key functions. Further, the obfuscation was not implemented in key portions of IEC-104 code which are reused in both versions, which enables us to visualize the overlaps.

For example, both INDUSTROYER versions use very similar code to parse APDU traffic and print specific parsed fields. Figure 6 is a screenshot of INDUSTROYER.V2 on the left and on the right a screenshot of the original INDUSTROYER.

```
  msg_oa_ioa = x_new_wrap();
  wsprintf((int)msg_oa_ioa, "\n\t\tASDU:%u | OA:%u | IOA:%u | ", asdu_address, originator_address, IOA);
  asdu_cause_from_code = x_get_asdu_cause_from_code(*(unsigned __int8 *)(*(_DWORD *)&apci->ControlFields.
                                                                         + 2));
  strcpy(cause, asdu_cause_from_code);
  telegram_code = **(unsigned __int8 **)&apci->ControlFields.control_field_3;
  telegram_type_string = x_get_asdu_type_from_code(telegram_code);
  code = *(unsigned __int8 *)(*(_DWORD *)&apci->ControlFields.control_field_3 + 2);
  msg_cause_type = x_new_wrap();
  wsprintf(
    (int)msg_cause_type,
    "\n\t\tCause: %s (x%X) | Telegram type: %s (x%X)",
    cause,
    code,
    telegram_type_string,
    telegram_code);
}
if ( *(_BYTE *)(*(_DWORD *)&apci->start_byte_104 + 6) == 3 )// Handle APCI U-Format Type
{
  if ( (apci_in->ControlFields.control_field_1 & STARTDT_ACT) != 0 )
  {
    msg_startd_con = x_new_wrap();
    wsprintf((int)msg_startd_con, "STARTDT con");// start data transfer confirmation
  }
  if ( (apci_in->ControlFields.control_field_1 & STOPDT_CON) != 0 )
  {
    msg_startd_act = x_new_wrap();
    wsprintf((int)msg_startd_act, "STARTDT act");// start data transfer activation
  }
  if ( (apci_in->ControlFields.control_field_1 & STOPDT_ACT) != 0 )
  {
    msg_stopdt_con = x_new_wrap();
    wsprintf((int)msg_stopdt_con, "STOPDT con");// stop data transfer confirmation
  }
  if ( (apci_in->ControlFields.control_field_1 & STOPDT_CON0) != 0 )
  {
    msg_stopdt_con_ = x_new_wrap();
    wsprintf((int)msg_stopdt_con_, "STOPDT con");// stop data transfer confirmation
  }
  if ( (apci_in->ControlFields.control_field_1 & TESTFR_ACT) != 0 )
  {
    msg_testfr_con = x_new_wrap();
    wsprintf((int)msg_testfr_con, "TESTFR con");// test frame confirmation
  }
  if ( (apci_in->ControlFields.control_field_1 & (unsigned __int8)TESTFR_CON) != 0 )
  {
    msg_testfr_act = x_new_wrap();
    wsprintf((int)msg_testfr_act, "TESTFR act");// test frame activation
  }
}
msg_n = x_new_wrap();
```

Figure 6: APDU traffic handling in INDUSTROYER.V2 (Left) and INDUSTROYER.104 (Right)

Additional notable code overlaps between the two versions exist in implementation of ASDU frame creation, sending of APDU messages, change option execution, and thread setup for IEC-104 functionality.

Appendix: YARA Rules

```
rule MTI_Hunting_INDUSTROYERv2_Bytes {

meta:

author = "Mandiant"

date = "04-09-2022"

description = "Searching for executables containing bytecode associated with the INDUSTROYER.V2 malware
family."


strings:
```

```
$bytes = {8B [2] 89 [2] 8B 0D [4] 89 [2] 8B 15 [4] 89 [2] A1 [4] 89 [2] 8B 0D [4] 89 [2] 8A 15 [4] 88 [2] 8D [2] 5? 8B [2] E8}

condition:

filesize < 3MB and

uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and

$bytes

}

rule MTI_Hunting_INDUSTROYERv2_Strings {

meta:

author = "Mandiant"

date = "04-09-2022"

description = "Searching for executables containing strings associated with the INDUSTROYER.V2 malware family."

strings:

$a1 = "M%X - %02d:%02d:%02d" nocase ascii wide

$a2 = "%02hu:%02hu:%02hu:%04hu" nocase ascii wide

$a3 = "%s M%X " nocase ascii wide

$a4 = "%s: %d: %d" nocase ascii wide

$a5 = "%s M%X %d (%s)" nocase ascii wide

$a6 = "%s M%X SGCNT %d" nocase ascii wide

$a7 = "%s ST%X %d" nocase ascii wide

$a8 = "Current operation : %s" nocase ascii wide

$a9 = "Sent=x%X | Received=x%X" nocase ascii wide

$a10 = "ASDU:%u | OA:%u | IOA:%u | " nocase ascii wide

$a11 = "Cause: %s (x%X) | Telegram type: %s (x%X" nocase ascii wide

$b1 = "Length:%u bytes | " nocase ascii wide

$b2 = "Unknown APDU format !!!" nocase ascii wide

$b3 = "MSTR ->> SLV" nocase ascii wide

$b4 = "MSTR <<- SLV" nocase ascii wide

condition:

filesize < 3MB and

uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550 and
```

```
(1 of ($a*) and 1 of ($b*))

}
```

Appendix: Mapped Configuration Example

Example Config	192.168.	2404	2	0	1	1		1		0	1
Config Position	1	2	3	4	5	6	7	8	9	10	11
Config Description	Station IP Address	Station Port	Entry Index Value	Enable Hard-Coded Telegrams with specified Range	Enable Configuration Options 6 - 14	Enable Process Termination	Process Name to Terminate	Enable File Rename	Directory Path for File Rename	Sleep Prior to IEC-104 functionality	Sleep Duration Seed Value
ASDU Entry Position											
ASDU Entry Description											

Table 3: Mapped configuration example

Appendix: Example IOAs From Publicly Available Sample Mapping to An ABB Distribution Recloser Relay

IOA	Description
1101	50BFT:InPosCIsA Status
1102	50BFT:InPosCIsB Status
1103	50BFT:InPosCIsC Status
1104	50BFT:InStr status

1105 50BFT:InStrA status

1106 50BFT:InStrB status

1107 50BFT:InStrC status

1108 50BFT:general

1201

1202

1203

1204

1250

1251

1252

1253

1254

1255

1256

1257

1258

1259

1260

1261

1262

1263

1264

1265

1301

1302

1304

1401

1402

1403

1404

130202

160921

160923

160924

160925

160927

160928

190202

260202

260901

260902

260903

260904

260905

260906

260907

260908

260909

260910

260911

260912

260914

260915

260916

260918

260920

290202

338501

Acknowledgements

This research was made possible thanks to the hard work of many people not listed on the byline. A huge thanks to CERT UA and ESET. Special thanks to Josh Triplett, Conor Quigley, and Wesley Mok.