VEDERE LABS

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Industroyer2 and INCONTROLLER

In-depth Technical Analysis of the Most Recent ICS-specific Malware

Contents

Acknowledgment

We would like to thank Dr. Emmanuele Zambon at the Eindhoven University of Technology for his analysis of Industroyer2, included in Section 2.1 of this report, and his technical review of this document.

1. Executive Summary

Industroyer2 and INCONTROLLER, also known as PIPEDREAM, are the newest examples of ICS-specific malware and were disclosed to the public almost simultaneously on April 12 and 13, 2022, respectively.

Industroyer2 leverages OS-specific wipers and a dedicated module to communicate over the IEC-104 industrial protocol. INCONTROLLER is a full toolkit containing modules to send instructions to or retrieve data from ICS devices using industrial network protocols, such as OPC UA, Modbus, CODESYS, Machine Expert Discovery and Omron FINS. Additionally, Industroyer2 has a highly targeted configuration, while INCONTROLLER is much more reusable across different targets.

ICS-specific malware is still very rare when compared to commodity malware, such as ransomware or banking trojans. Industroyer2 and INCONTROLLER follow previous-known examples, such as Stuxnet, Havex, BlackEnergy2, Industroyer and TRITON, shown in the timeline figure below.

Both Industroyer2 and INCONTROLLER were caught before causing physical disruption. Industroyer2 is believed to have been developed and deployed by the Sandworm APT, linked to the Russian GRU, which was behind the original attacks on the Ukrainian power grid in 2015 and 2016. The Industroyer2 incident follows recent activity against the APT in 2022, such as the disruption of the Cyclops Blink botnet. There is still no conclusive evidence about the actors behind INCONTROLLER, their motives or objectives.

Both new malwares show that abusing often insecure-by-design native capabilities of OT equipment continues to be the preferred modus operandi of real-world attackers. Vedere Labs recently disclosed a set of 56 insecure-by-design vulnerabilities in OT equipment called OT:ICEFALL, which included Omron controllers that were targeted by INCONTROLLER. The emergence of new vulnerabilities and new malware exploiting the insecure-by-design nature of OT supports the need for robust OT-aware network monitoring and deep packet inspection capabilities.

This briefing presents the most detailed (to date) public technical analysis of Industroyer2 and INCONTROLLER (Section 2), a list of IoCs extracted from those samples and other shared intelligence (Section 3) and recommended mitigations (Section 4).

Although there have been previous reports about both malware families analyzed in this research, we present the following new contributions:

- A functionality in Industroyer2 to discover the target's Common Address of ASDU. Despite not being used given the hardcoded configuration of our sample, it might have been a tool used in previous reconnaissance stages to gather information about the target (Section 2.1.2)
- An analysis of the similarity of the IEC-104 implementation in Industroyer that reveals it is very probably a modified version of a publicly available implementation (Section 2.1.3)
- The most detailed public description so far of Lazycargo, a part of INCONTROLLER, which became publicly available (Section 2.2.1)

2. Technical Analysis

2.1. Industroyer2

ESET researchers responded to a cyber incident affecting an energy provider in Ukraine. This response resulted in the discovery of a new variant of the Industroyer malware, which ESET together with CERT-UA named Industroyer2. Industroyer is an infamous piece of malware that was used in 2016 by the Sandworm APT group to cut the power in Ukraine.

Several researchers pointed out that the new sample bears a lot of similarities with the original Industroyer. However, while the original version supported several industrial network protocols, the version used in the new incident supports only the IEC-104 protocol. The sample tests connectivity to a list of hardcoded control stations and sends sets of hardcoded commands over the IEC-104 protocol, setting specific Information Object Addresses (IOA) for specific Application Service Data Unit (ASDU) addresses to either the "ON" or "OFF" state. As ESET researchers pointed out, this may lead to power cuts within the targeted ICS systems.

We have analyzed the IEC-104 sample with SHA-1 fdeb96bc3d4ab32ef826e7e53f4fe1c72e580379 and presumed filename 40_115.exe. Our static analysis revealed details of the hardcoded configuration and logic workflow of the sample.

2.1.1. Configuration

The configuration is built as an array of strings. Every array item specifies the configuration for a single IEC-104 target server and is specified as a space-separated list of tokens. Tokens can be logically grouped in a header, followed by an optional list of Information Object (IO)-specific parameters. The format of the header is reported in the table below.

Name	Optional	Description
Target IP	No	IP address of the target IEC-104 server.
Target Port	No	TCP port of the target IEC-104 server.
Common Address	No	Common Address of ASDU associated with the target IEC-104 server.
Operational Mode	No	If set to 0, the sample will derive which IOs to interact with from the optional list of IO parameters that follows the header. If set to 1, the sample will derive which IOs to interact with from the optional IOA range information that follows this token.
IOA Range Start	Yes	Information Object Address range start. This token is only specified if Operational Mode is 1.
IOA Range End	Yes	Information Object Address range end. This token is only specified if Operational Mode is 1.
Extended Config	No	If set to 1, the configuration header is extended with 9 extra tokens.

Boolean Flag	Yes	Unused. This token is only specified if Extended Config is 1.
Target Executable	Yes	Executable name of the process to kill before attempting connection with the target IEC-104 server. This token is only specified if Extended Config is 1.
Rename Executable	Yes	If set to 1, the executable previously specified will also be renamed to prevent watchdog restarts. This token is only specified if Extended Config is 1.
Target Executable Folder	Yes	Path to the folder where the target executable is stored. This token is only specified if Extended Config is 1.
Interaction Delay	Yes	Delay (in minutes) before a connection is attempted to the target IEC-104 server after killing the target executable. This token is only specified if Extended Config is 1.
Default Sleep Time	Yes	Delay (in seconds) applied after sending commands with a certain priority level. This token is only specified if Extended Config is 1.
Special Priority	Yes	Priority level for configuring a different sleep time. This token is only specified if Extended Config is 1.
Special Sleep Time	Yes	Delay (in seconds) applied after sending commands with priority level specified above. This token is only specified if Extended Config is 1.
Boolean Flag	Yes	Unused. This token is only specified if Extended Config is 1.
Default IO State	No	If set to 1, the state of single and double IOs will be set to On, otherwise the state will be set to Off.
Additional Inverted IO State	No	If set to 1, the sample will send additional commands for each configured IO inverting the default state.
IO Count	No	Number of IO-specific parameter groups following the header.

The format of each IO-specific parameter group is reported in the table below.

Name	Optional	Description
IOA	No	Address of the Information Object.
Type ID	No	Type of IEC-104 command used for setting the IO value. Possible values are 0 for double command IOs (C_DC_NA_1) and 1 for single command IOs (C_SC_NA_1).
SBO	No	If set to 1, the sample will use the Select Before Operate paradigm to set the IO value.
Invert Default State	No	If set to 0, the state of the IO will be set to the default value specified in the header. If set to 1, the state of the IO will be set to the inverse of the default value.
Priority	No	Priority of commands for this IO. The sample will send commands to the target IEC-104 server processing IOs with lower to higher priority.
Index	No	Defines the order by which commands for this IO will be processed as compared to the ones with the same priority.

Using this knowledge, it is possible to examine the configuration hardcoded in this sample. The configuration header is displayed in the table below.

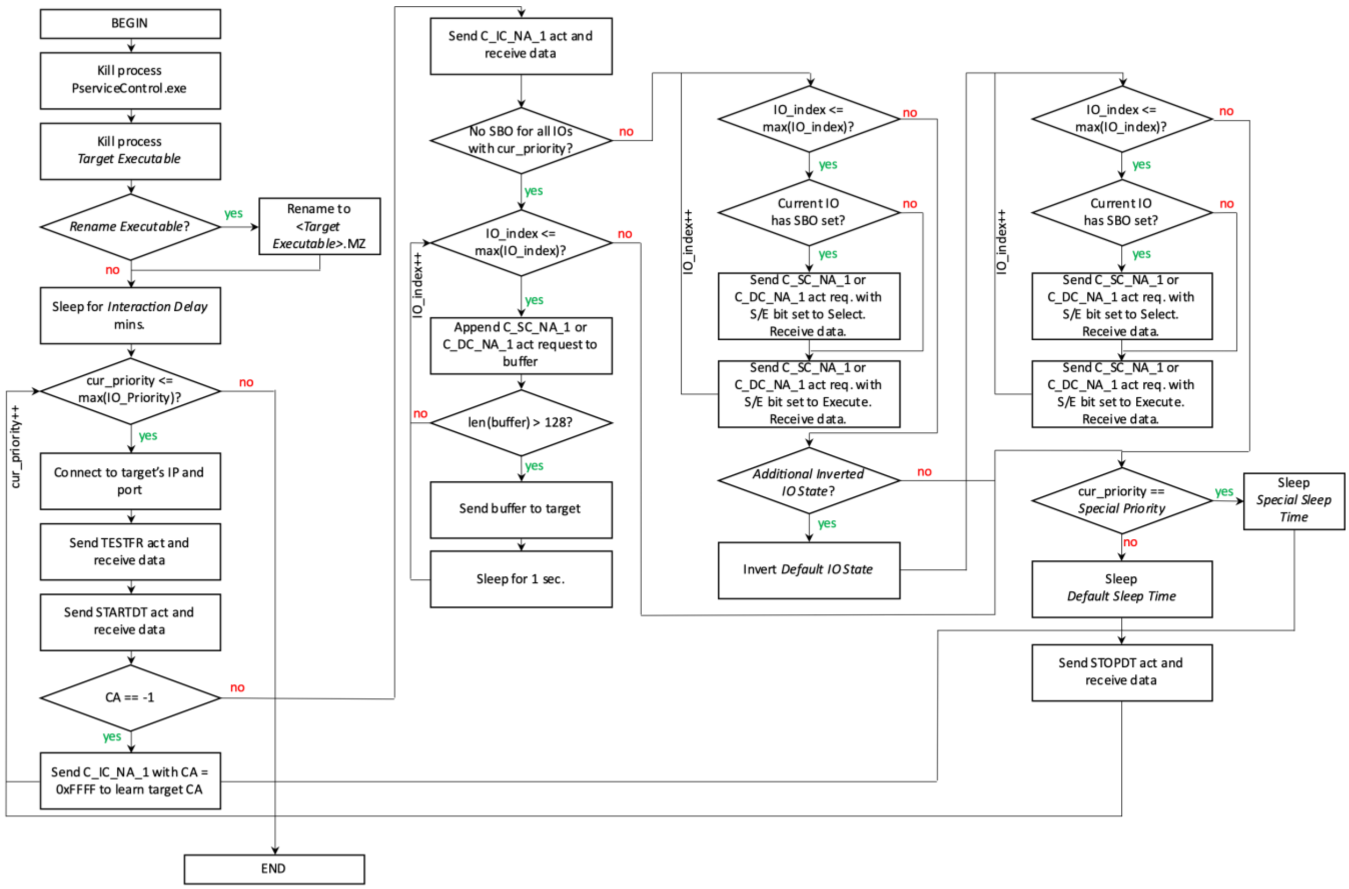
Field	Target 1	Target 2	Target 3
Target IP	10.82.40.105	192.168.122.2	192.168.121.2
Target Port	2404	2404	2404
Common Address	3	2	1
Operational Mode	0	0	0
IOA Range Start	N/A	N/A	N/A
IOA Range End	N/A	N/A	N/A
Extended Config	1	1	1
Boolean Flag	1	1	1
Target Executable	PService_PPD.exe	PService_PPD.exe	PService_PPD.exe
Rename Executable	1	1	1
Target Executable Folder	D:\OIK\DevCounter	D:\OIK\DevCounter	D:\OIK\DevCounter
Interaction Delay	0	0	0
Default Sleep Time	1	1	1
Special Priority	0	0	0
Special Sleep Time	0	0	0
Boolean Flag	1	1	1
Default IO State	0	0	0
Invert IO Value	0	0	0
IO Count	44	8	16

The first IO-specific group of parameters for each configuration item is reported in the table below as an example.

Field	Target 1	Target 2	Target 3
IOA	130202	1104	1258
Type ID	1 (C_SC_NA_1)	0 (C_DC_NA_1)	0 (C_DC_NA_1)
SBO	0 (Direct Operate)	0 (Direct Operate)	0 (Direct Operate)
Invert Default State	1	0	0
Priority	1	1	1
Index	1	1	1

2.1.2. Logic of Operation

The Industroyer2 sample is meant to be executed in the machine acting as IEC-104 controlling station for its targets. The workflow below displays a high-level representation of the sample's logic.



For each configuration item, the sample parses the configuration string and creates a data structure that holds configuration parameters, as well as runtime parameters.

Killing running services and renaming executable

It then kills the process with executable name "PServiceControl.exe", as well as the process with executable name "PService_PDD.exe", which is also renamed as "PService_PDD.exe.MZ". Killing the "PService_PDD.exe" service causes the interruption of any existing communication with target IEC-104 servers, which usually supports at most one active connection at a time. Having interrupted existing connections, Industroyer2 is free to connect to the targets. Renaming the service is a possible measure to prevent automatic service restarts. This behavior suggests some ties to the BlackEnergy malware, which also killed a service called "PService_PDD.exe" before execution.

After this initial phase, the sample spawns a thread responsible for interaction with the target. At first, the thread is set to sleep for a time specified by the Interaction Delay parameter. This delay could be needed to ensure the target realizes the existing connection with the master is interrupted and becomes ready to accept new connections.

The thread then loops over the priority levels configured for all IOs, from lower to higher priority levels.

Target connection

The sample connects to the target using the IP and port specified in the configuration. Upon success, it first sends a TESTFR act IEC-104 message, followed by a STARTDT act message, which starts the data transfer between the controlling station and the controlled station.

Once the target is connected and data transfer is enabled, the sample verifies if the Common Address of ASDU (CA) for the target is known in the configuration.

Discovery of the target's Common Address of ASDU

If the target's CA unknown (i.e., set to -1 in the configuration), the sample sends a general interrogation command activation message (C_IC_NA_1 act) with CA set to 65535, which is a special address defined in the standard as "global" for broadcast purposes. The target IEC-104 server will respond with a general interrogation command activation confirmation message containing its true CA. In this way, the sample can learn the CA of the target server. After learning the CA of the target, the sample sends a STOPDT act message to stop IEC-104 data transfer and disconnects from the target.

To the best of our knowledge, this discovery functionality was not documented in previous technical reports and, despite not being used given the hardcoded configuration of our sample, it might have been a tool used in previous reconnaissance stages to gather information about the target(s).

Changing the position of configured IOs

If the target's CA is known, the sample sends a general interrogation command activation message (C_IC_NA_1 act).

In case the configuration for all IOs with a certain priority level excludes the use of the Select Before Operate (SBO) paradigm, the sample first generates for all IOs with that priority either single command (C_SC_NA_1 act) or double command (C_DC_NA_1 act) activation messages (depending on the configuration) with the Select/Execute bit set to Execute, and then sends messages in batches of data of 128 bytes max. We notice that the thread executing these operations is put to sleep for a fixed amount of time (one second) after generating the command corresponding to a certain IO, regardless of whether commands are being sent to the target or just buffered locally. We could not find a meaningful explanation for this behavior.

In case at least one of the IOs with the current priority level is configured to use the SBO paradigm, the sample does not buffer commands. Instead, it iterates over all configured IOs and directly sends either single command (C_SC_NA_1 act) or double command (C_DC_NA_1 act) activation messages (depending on the configuration). In case the configuration specifies to use SBO, the sample first sends a single or double command with the Select/Execute bit set to Select. In both cases, the sample always sends the single or double command with the Select/Execute bit set to Execute.

The parameters of single or double commands that the sample sends to the target are set as follows:

- Cause of Transmission: hardcoded to 6 (activation)
- Originator Address: hardcoded to 0
- Common Address of ASDU: as specified in the "Common Address" configuration parameter
- Information Object Address: as specified in the "IOA" configuration parameter
- Qualifier: hardcoded to 2 (short pulse)
- Select/Execute bit: according to the logic described above
- Single/Double Command: initially set according to the "Default IO State" configuration parameter and possibly inverted according to IO's "Invert Default State" configuration parameter

In case the IO's configuration parameter "Invert Default State" is set to true, the sample sends the single/double commands once more by temporarily inverting the value of the "Default IO State" configuration parameter. This causes flipping the position of the targeted single or double Information Objects (from On to Off or vice versa).

Before repeating all these operations for IOs with the next priority level, the sample sets threat to sleep for an amount of time specified in either the "Default Sleep Time" or the "Special Sleep Time" configuration parameters (depending on whether the current priority level is the special priority level configured in the "Special Priority" parameter), and then sends a STOPDT act message to stop IEC-104 data transfer and disconnects from the target.

2.1.3. IEC-104 Protocol Implementation

Our analysis revealed that the code used in the sample to craft IEC-104 messages shows extensive similarities with code in a public github repository. The repository contains a lightweight "C++ realization of IEC-60870-5-104 for LPC1768+FreeRTOS+lwIP" and is maintained by Oleksandr Popovych, a Ukrainian developer who describes himself as "AI Dealer", "Machine Learning Evangelist" and "Deep Learning Practitioner".

By static analysis of the sample, we were able to identify 18 of the 23 functions defined in the repository for the three C++ classes corresponding to IEC-104 layers (APCI, ASDU and APDU). Of these functions, 15 have the exact same function signature as defined in the repository, three have function signatures with only marginal differences (e.g., addition of a function argument) and 15 also have the exact same function body. The major difference we identified is in the implementation of the APCI class, which in the sample was simplified by only supporting management of one single Information Object per APCI PDU. Based on these observations, it is reasonable to conclude the creators of Industroyer2 adapted the code shared by Popovych to fit their needs.

The table below reports a list of the functions defined in Popovych's code, annotated with our findings on the sample binary in terms of function presence, similarity of the function signature and similarity of the function body.

Class	Function	Found in Sample	Signature Similarity	Body Similarity
APCI	APCI()	Yes	Complete	Complete
APCI	~APCI()	Yes	Complete	Complete
APCI	clear()	Yes	Complete	Complete
APCI	get()	Yes	Complete	Complete
APCI	set()	Yes	Minor (one unused argument added)	Complete
APCI	valid()	Yes	Complete	Complete
ASDU	ASDU()	Yes	Complete	Complete
ASDU	~ASDU()	Yes	Complete	Complete

ASDU	clear()	Yes	Complete	Major (member variables are different)
ASDU	get()	Yes	Complete	Major (member variables are different)
ASDU	set()	Yes	Minor (argument data_length added)	Major (member variables are different)
ASDU	addIO()	No	N/A	N/A
ASDU	valid()	Yes	Complete	Complete
APDU	APDU()	Yes	Complete	Complete
APDU	~APDU()	Yes	Complete	Complete
APDU	clear()	Yes	Complete	Complete
APDU	get()	Yes	Complete	Complete
APDU	set()	Yes	Minor (one unused argument added)	Complete
APDU	valid()	Yes	Complete	Complete
APDU	addIO(int)	No	N/A	N/A
APDU	addIO(InformationObject)	No	N/A	N/A
APDU	setDUI()	No	N/A	N/A
APDU	setAPCI()	No	N/A	N/A

The two snippets of code below show an example of the same function as defined in Popovych's code (left) and as decompiled from the sample (right). It is clear the code is identical once one factors out the artefacts introduced by the C++ compiler.

```
void APCI::clear()
{
  start = STARTHEAD;
  length = 4;
  format = U_FORMAT;
  func = 0;
  ssn = 0;
  rsn = 0;
}
```

```
void APCI::clear(void *this)
{
  *((_BYTE *)this + 4) = 0x68;                    // start = STARTHEAD
  *((_BYTE *)this + 5) = 4;                       // length = 4
  *((_BYTE *)this + 6) = 3;                       // format = U_FORMAT
  *((_BYTE *)this + 7) = 0;                       // func = 0
  *((_DWORD *)this + 2) = 0;                      // ssn = 0
  *((_DWORD *)this + 3) = 0;                      // rsn = 0
}
```

Besides the code for serializing/deserializing IEC-104 messages, the sample includes functions for sending and receiving the necessary IEC-104 messages. We could identify code supporting the following functionalities:

- Send a TESTFR_act message (test connection activation) and process incoming messages
- Send a TESTFR_con message (test connection confirmation)
- Send a STARTDT_act message (start data transfer activation) and process incoming messages

- Send a C_IC_NA_1_act message (interrogation command activation) and process incoming messages
- Send a C_IC_NA_1_act message (interrogation command activation) with CA set to the global address, receive incoming messages and learn the CA reported in the received C_IC_NA_1_con message (interrogation command confirmation)
- Send a C_SC_NA_1 act message (single command activation) or C_DC_NA_1_act message (double command activation) and process incoming messages
- Send an S_FRAME to acknowledge the receipt of incoming I_FRAMEs
- Process incoming messages and:
    - respond to TESTFR_act messages with TESTFR_con messages
    - update the Receiver Sequence Number in case an I_FRAME is received
    - acknowledge received I_FRAMEs by sending an S_FRAME with the updated receiver sequence number

As can be inferred from the list above, the implemented subset of the IEC-104 protocol client-side functionality is extremely limited and is directed at covering only the subset that is strictly necessary for the attack. However, this choice led to an implementation that does not conform to the state machine and timeout mechanisms defined in the IEC-104 standard. While this may not necessarily be a problem for interoperability with permissive IEC-104 server implementations, such as those implemented by most of IEC-104 server simulators freely downloadable from the internet, for servers with a stricter implementation this might result in the malware failing to deliver the intended commands to the target.

This same implementation issue was previously observed in the original Industroyer/CrashOverride malware.

2.1.4. Dynamic Behavior

We confirm our findings about the operation logic of the sample by running the sample against an IEC-104 server simulator and capturing the traffic generated by the sample. The figure below shows the commands sent by the sample to the target with IP address 192.168.122.2. After the general station interrogation command, we can observe the eight double commands sent by the sample with position OFF, cause of transmission 6 (activation), S/E bit set to Execute and qualifier set to 1 (short pulse), corresponding the eight Information Objects defined in the configuration for this target.

Parameter name	Parameter value	Source host	Destination host	Details
IOA 0	Station interrogation (global)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 100, Cause TX 6 (act)
IOA 1104	OFF (Qualifier: 1 (short pulse), Execute)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 46, Cause TX 6 (act)
IOA 1105	OFF (Qualifier: 1 (short pulse), Execute)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 46, Cause TX 6 (act)
IOA 1106	OFF (Qualifier: 1 (short pulse), Execute)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 46, Cause TX 6 (act)
IOA 1107	OFF (Qualifier: 1 (short pulse), Execute)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 46, Cause TX 6 (act)
IOA 1108	OFF (Qualifier: 1 (short pulse), Execute)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 46, Cause TX 6 (act)
IOA 1101	OFF (Qualifier: 1 (short pulse), Execute)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 46, Cause TX 6 (act)
IOA 1102	OFF (Qualifier: 1 (short pulse), Execute)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 46, Cause TX 6 (act)
IOA 1103	OFF (Qualifier: 1 (short pulse), Execute)	192.168.122.100...	192.168.122.2 ...	IEC 60870-5-104 ASDU Type ID 46, Cause TX 6 (act)

2.1.5. Other Considerations

During the incident, additional malware samples were deployed: CaddyWiper, OrcShred, SoloShred, and AwfulShred. These are wiper malwares designed for Windows, Linux and Solaris operating systems and used to cause damage to the infected machines by wiping all the data, and to clean up the host-based indicators of compromise.

It is still unknown how the attackers obtained initial access to the IT assets of the victim. According to CERT-UA, CaddyWiper was distributed over the victim's network using the Windows group policy mechanism (GPO) set through the POWERGAP powershell script. This script has also been used to schedule the execution of CaddyWiper, which relied on ArguePatch[1] loader to decrypt itself. (TailJump shellcode was used as well.) The lateral movement between network segments of the victim was performed via SSH tunnels.

Multiple researchers agree that the attackers were deeply familiar with the victim's network and the attack was tailor-made rather than opportunistic. For example, Industroyer2 relies on a built-in hard-coded configuration that lists the IP addresses of controlled stations, their TCP ports, ASDU addresses and specific commands to be sent over the IEC-104 protocol. The fact that the IP addresses of these stations are located within entirely different subnets (as found in several public Industroyer2 samples) suggests that the victim environment could have improper network segmentation controls in place.

The Industroyer2 sample lacks any detection evasion mechanisms, such as control flow obfuscation or config encryption, or privilege escalation capabilities. This serves as additional evidence of the "bespoke" nature of the attack: The attackers could have had total control of the target environment and be aware of the exact malware protection mechanisms deployed (or lack thereof). According to the timeline of the incident published by the ESET researchers, CaddyWiper was scheduled to launch on the same compromised machine after the Industroyer2 executable has had finished its task. Had the attack been successful, the researchers might not have obtained the sample in the first place. All this evidence explains (at least in part) the lack of analysis protection mechanisms within the Industroyer2 binary.

2.2. CISA AA22-103A: APT Cyber Tools Targeting ICS/SCADA Devices (aka INCONTROLLER, aka PIPEDREAM)

On April 13, the Department of Energy, CISA, NSA and the FBI released a cybersecurity advisory about new capabilities developed by APTs targeting industrial control systems. The toolkit described in the advisory includes three tools that enable attackers to send instructions to or retrieve data from ICS devices using industrial network protocols, such as OPC UA, Modbus (and its proprietary Schneider Modicon extension), Codesys and Omron FINS.

The tools within the toolkit are named differently by different researchers but have the following functionality:

- ***Lazycargo***: One of the tools exploits CVE-2020-15368, a vulnerability in the AsrDrv103.sys driver of the RGB controller for AsRock PC motherboards. This tool installs and exploits the vulnerable driver on a target system to achieve persistence and perform lateral movement after the initial compromise of Windows-based engineering workstation and/or human-machine Interface (HMI) machines.
- ***Icecore/Dusttunnel***: A tool that provides reconnaissance and command and control functionality.
- ***Codecall/Evilscholar***: This tool is a framework that communicates over the Modbus protocol; it also leverages Codesys automation software. The framework contains modules to scan, interact with and attack at least three Schneider Electric programmable logic controllers (PLCs): M251, M258 and M221 Nano. The capabilities targeting these PLCs could possibly be extended against other Codesys-based PLCs manufactured by other vendors.
- ***Omshell/Badomen***: A framework that has capabilities for scanning and interacting with Omron Sysmac NEX PLCs via HTTP, Telnet and Omron Fins protocols. It has capabilities for interacting with OMRON servo drives used for precision motion control operations.

[1] A legitimate component of IDA Pro used for remote debugging.

- ***Tagrun/Mousehole***: This tool is used for identifying Open Platform Communication Unified Architecture (OPC UA) servers, as well as enumerating, reading and writing OPC structures and tags. It can be also used for brute-forcing credentials.

Currently, only a sample of ***Lazycargo*** is available for public analysis. We found the sample ***69296ca3575d9bc04ce0250d734d1a83c1348f5b6da756944933af0578bd41d2*** on vx-underground and analyzed it in depth.

2.2.1. Lazycargo Analysis

The sample is a binary executable that requires administrative privileges to run and expects one argument, as shown in the figure below:

At first glance, the binary contains a lot of interesting information: We clearly see that there are some debug symbols leftovers that suggest the binary may be an “exploit for the AsRock Driver”, that the file is likely to have some embedded executable code in its *.data* section and that it uses a number of potentially malicious Win32 API calls, as shown in the figure below.

property	value
md5	56934C754D40EE92EDFFC56082AF6A65
sha1	AC104858FCC7FBF0D73F0BD89DB8C42C8D62390C
sha256	C09381355FC40B302AC57749F65F018E5AAC0F52F23D06B4B60EAB1E79ED328F
age	1
size	103 bytes
format	RSDS
debugger-stamp	0x5CEFD9A4 (Thu May 30 13:24:52 2019 | UTC)
path	C:\Users\User1\Desktop\dev projects\SignSploit1\x64\Release\AsrDrv_exploit.pdb
Guid	E0658672-20C9-49CA-95A7-8944F11D642

property	va...	v...	value
name	.t...	.r...	.data
md5	A...	C...	D9A1F1A4D48906DA1D9F33EAE0F0EAEF
entropy	5....	4....	6.122
file-ratio (99.77%)	65...	2...	9.08 %
raw-address	0x...	0...	0x0005EC00
raw-size (444416 bytes)	0x...	0...	0x00009E00 (40448 bytes)
virtual-address	0x...	0...	0x0000000040061000
virtual-size (449671 bytes)	0x...	0...	0x0000B760 (46944 bytes)
entry-point	0...	-	-
characteristics	0x...	0...	0xC0000040
writable	-	-	x
executable	x	-	-
shareable	-	-	-
discardable	-	-	-
initialized-data	-	x	x
uninitialized-data	-	-	-
unreadable	-	-	-
self-modifying	-	-	-
virtualized	-	-	-
file	-	-	executable, offset: 0x00060010, size: 24948

functions (100)	blacklist (22)	anonymous (0)	library (3)
WriteFile	x	-	kernel32.dll
ExitProcess		-	kernel32.dll
GetCommandLineA		-	kernel32.dll
GetCommandLineW		-	kernel32.dll
HeapAlloc		-	kernel32.dll
HeapSize		-	kernel32.dll
HeapValidate		-	kernel32.dll
GetSystemInfo		-	kernel32.dll
CompareStringW		-	kernel32.dll
LCMapStringW		-	kernel32.dll
GetFileType		-	kernel32.dll
WaitForSingleObject		-	kernel32.dll
GetExitCodeProcess	x	-	kernel32.dll
CreateProcessW	x	-	kernel32.dll
GetFileAttributesExW		-	kernel32.dll
OutputDebugStringW		-	kernel32.dll
WriteConsoleW		-	kernel32.dll
GetConsoleCP		-	kernel32.dll
GetConsoleMode		-	kernel32.dll
GetFileSizeEx		-	kernel32.dll
SetFilePointerEx		-	kernel32.dll
FindClose		-	kernel32.dll
FindFirstFileExW	x	-	kernel32.dll
FindNextFileW	x	-	kernel32.dll
IsValidCodePage		-	kernel32.dll
GetACP		-	kernel32.dll
GetOEMCP		-	kernel32.dll
GetCPInfo		-	kernel32.dll
MultiByteToWideChar		-	kernel32.dll
WideCharToMultiByte		-	kernel32.dll
GetEnvironmentStringsW	x	-	kernel32.dll
FreeEnvironmentStringsW		-	kernel32.dll
SetEnvironmentVariableW	x	-	kernel32.dll
SetStdHandle		-	kernel32.dll
RegQueryValueExW		-	advapi32.dll
RegOpenKeyExW		-	advapi32.dll
StartServiceW		-	advapi32.dll
CloseServiceHandle		-	advapi32.dll
OpenServiceA		-	advapi32.dll
CreateServiceA	x	-	advapi32.dll
OpenSCManagerW		-	advapi32.dll
RegCloseKey		-	advapi32.dll
BCryptEncrypt	x	-	bcrypt.dll
BCryptGenerateSymmetricKey	x	-	bcrypt.dll
BCryptGetProperty	x	-	bcrypt.dll
BCryptOpenAlgorithmProvid...	x	-	bcrypt.dll
BCryptDestroyKey	x	-	bcrypt.dll
BCryptCloseAlgorithmProvid...	x	-	bcrypt.dll

```
; __int64 __fastcall main_routine(int, __int64)
main_routine proc near

lpOverlapped= qword ptr -5E0h
dwStartType= dword ptr -5D8h
dwErrorControl= dword ptr -5D0h
lpBinaryPathName= qword ptr -5C8h
lpLoadOrderGroup= qword ptr -5C0h
lpdwTagId= qword ptr -5B8h
lpDependencies= qword ptr -5B0h
lpServiceStartName= qword ptr -5A8h
lpPassword= qword ptr -5A0h
var_590= qword ptr -590h
var_588= qword ptr -588h
lpBuffer= qword ptr -580h
var_570= qword ptr -570h
ptr_second_stage_shellcode= xmmword ptr -568h
ptr_total_size= xmmword ptr -558h
NumberOfBytesRead= dword ptr -548h
rwargs= rweverything_args ptr -540h
ReOpenBuff= _OFSTRUCT ptr -520h
var_20= qword ptr -20h
var_10= byte ptr -10h

mov     rax, rsp
push    rbp
push    r14
push    r15
lea     rbp, [rax-508h]
sub     rsp, 5F0h
mov     [rsp+600h+var_590], 0FFFFFFFFFFFFFFFEh
mov     [rax+8], rbx
mov     [rax+18h], rsi
mov     [rax+20h], rdi
mov     rax, cs:__security_cookie
xor     rax, rsp
mov     [rbp+500h+var_20], rax
mov     rbx, rdx        ; command line string
cmp     ecx, 2          ; the number of command line elements must be at least 2
jge     short loc_1400010B5
```

From the command line message above, it is obvious that the binary expects a path to an unsigned device driver (a *.sys* file). The following disassembly fragment shows the beginning of the main routine of the sample and confirms this.

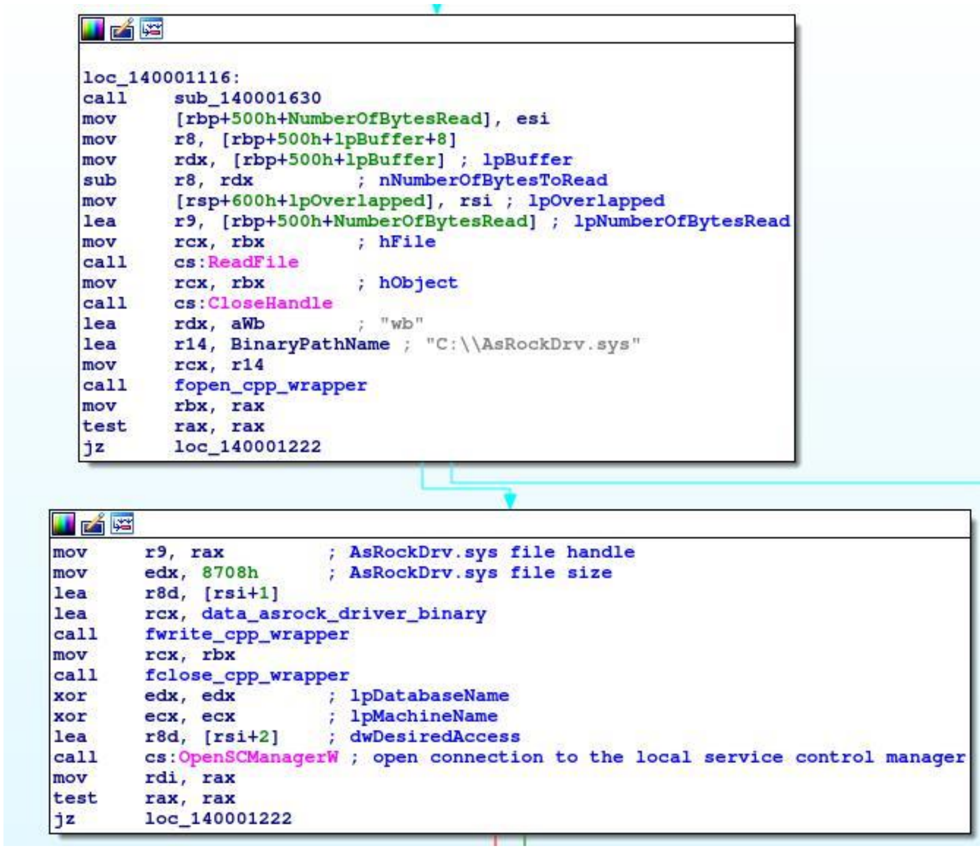
Therefore, to examine the behavior of the binary further, we must provide a command line argument as follows. In fact, this should be an unsigned driver, but we can get by with this argument.

```
Administrator: C:\Windows\system32\cmd.exe

FLARE Fri 05/13/2022  2:52:28.37
C:\Users\IEUser\Downloads+>69296ca3575d9bc04ce0250d734d1a83c1348f5b6da756944933af0578bd41d2.exe C:\Windows\System32\driv
ers\beep.sys
```

When the path to a *.sys* file is provided, the sample will get the file handle using the OpenFile() function, read its size of disk using GetFileSize() and read its contents into the memory using ReadFile().

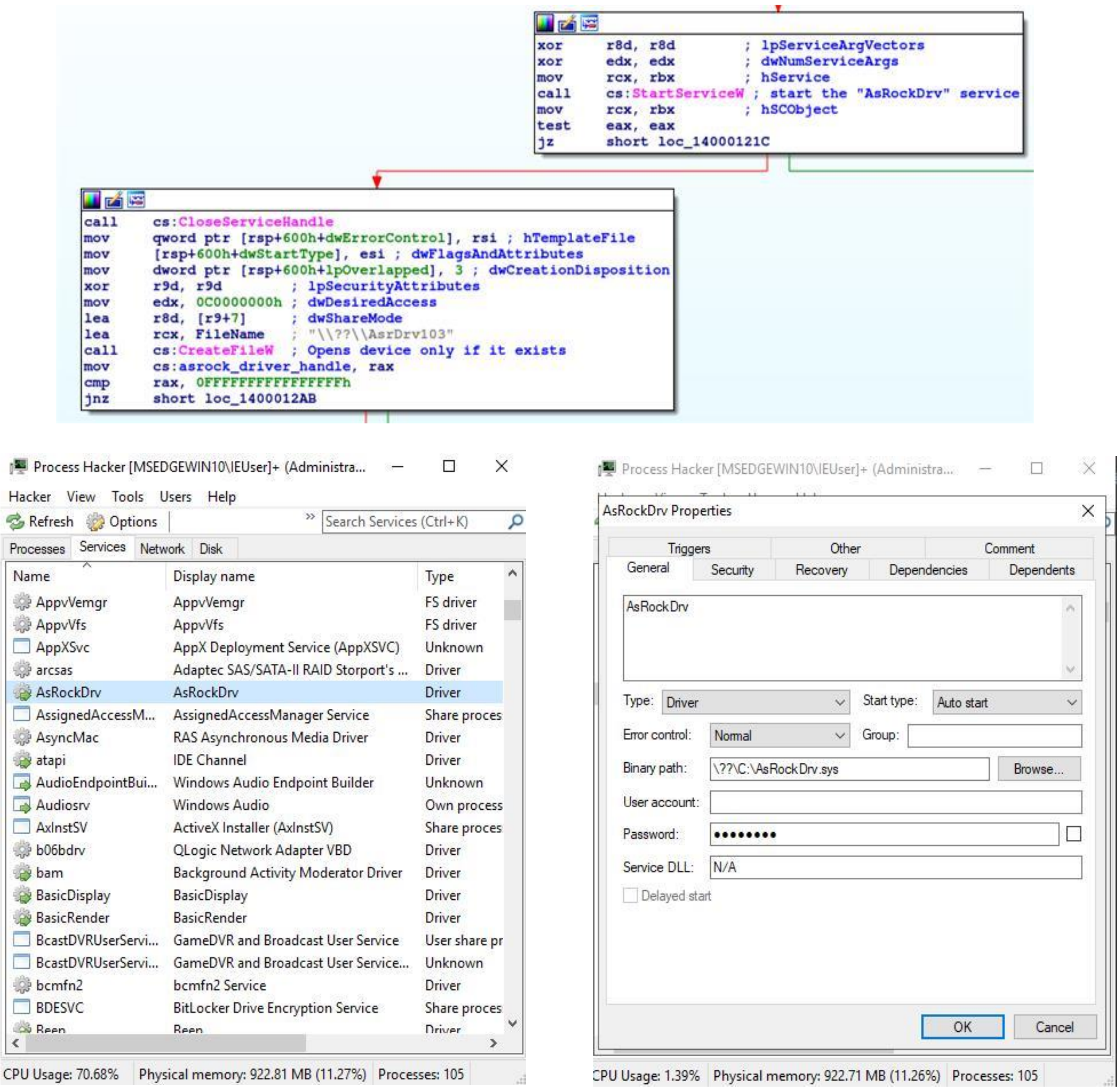
Next, the sample creates an empty file “C:\AsRockDrv.sys” and writes into it some binary content located in its *.data* section:

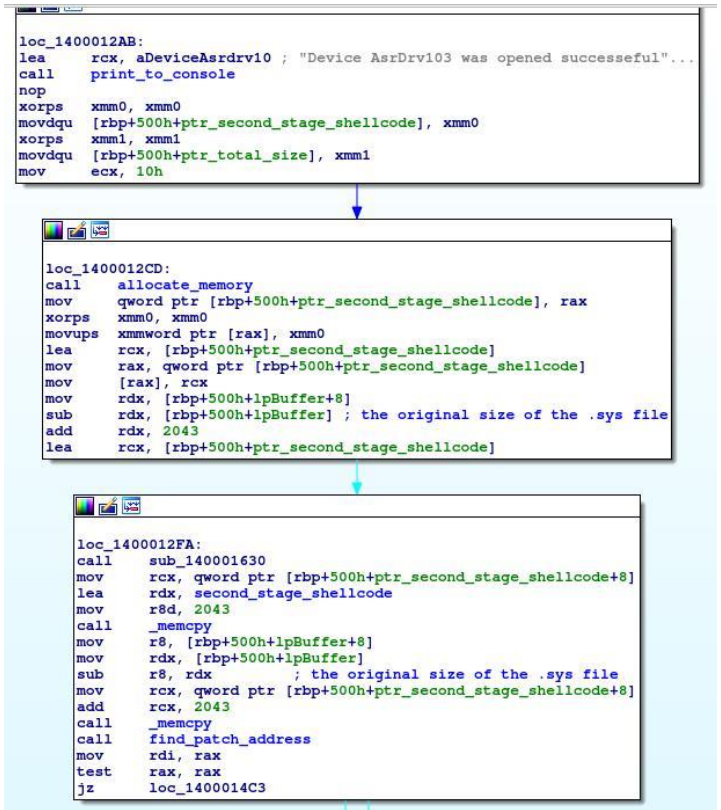


This binary content is a vulnerable **AsRock** driver, for which a publicly available exploit has been available for quite some time (**CVE-2020-15368**). We encourage the reader to look at the original write-up to have a better understanding of the various moving parts of the binary in question. However, this driver exploitation technique is not new. Notice that the *.data* section also contains three other shellcode fragments. (More on that later.)

After the contents of the **AsRock** driver are written to the disk, the binary loads it as a service, initiates the driver’s device and opens a file handle to it.

Next, the binary copies a shellcode fragment located in its *.data* section into memory – we call it "***second_stage_shellcode***" – and copies the contents of the *.sys* file provided as an argument into an adjacent memory location.





Then, the sample calls the "***find_patch_address()***" function that performs many things under the hood. In particular, it exploits **CVE-2020-15368** to read physical memory and to find an address of a function located within the loaded **AsRock** driver: This function has a specific ioctl handler tied to it, and it can be invoked from user-mode programs with DeviceIoControl() or NtDeviceIoControlFile() functions.

The two code snippets below provide an intuition on **CVE-2020-15368** and how it has been leveraged in the binary in question. In particular, the second snippet shows the approximate logic within the vulnerable **AsRock** driver: It provides unrestricted physical memory read and write capabilities (including kernel space) to any user-mode program. The **AsRock** driver developers have restricted access to these operations by accepting only encrypted ioctl data. However, the AES key used for encryption/decryption is hardcoded, therefore malware writers can easily circumvent that.

```
/*
* This snippet was taken from
    https://github.com/stong/CVE-2020-15368
*/

HANDLE hDevice = CreateFileA("\\\\.\\GlobalRoot\\Device\\AsrDrv104", GENERIC_READ | GENERIC_WRITE | SYNCHRONIZE,
    FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

// ... set up the encrypted ioctl data

BOOL result = DeviceIoControl(hDevice, 0x22EC00, ioctl_data, in_buf_size, out_buf, sizeof(out_buf), &bytes_returned, NULL);
```

```
/*
 * Most of this snippet was taken from
    https://github.com/stong/CVE-2020-15368
   Please keep in mind that this is not the exact logic, but rather
   an intuition
*/

if ( IoControlCode == 0x22EC00 or IoControlCode == 0x22E808)
{

  char enc_key[32];
  memset(enc_key, 0, sizeof(enc_key));
  memmove(enc_key, "C110DD4FE9434147B92A5A1E3FDBF29A", 32ui64);
  memcpy(enc_key + 13, ioctl_args->key, 16);

  size_t cb_decrypted = 0;
  my_decrypted_cmd* decryptedCmd = NULL;
  DWORD iv_size = ioctl_args->iv_size;
  DWORD input_size = *(DWORD*)(ioctl_args + bufferLen - 6);

  // really just calls BCrypt API to get an AES implementation
  if ( (unsigned int)decrypt_ioctl_params(enc_key, 32, ioctl_args->iv, iv_size, ioctl_args + bufferLen - input_size - 6,
       input_size, &decryptedCmd, &cb_decrypted) )
  {
    // Decryption failed
    if ( decryptedCmd )
      ExFreePoolWithTag(decryptedCmd, 0);
    irp->IoStatus.Status = 0xC000000D;
    goto Fail_Out;
  }
  IoControlCode = decryptedCmd->opcode;
  Rweverything_Args = &decryptedCmd->args;

  if (IoControlCode == 0x22EC00)
  {
    // write physical memory
  }
  else
  {
    // read physical memory
  }
}
else // not 0x22EC00 or 0x22E808
{
  if ( IoControlCode != 0x22E858 &&
       IoControlCode != 0x22E860 &&
       IoControlCode != 0x22E800 &&
       IoControlCode != 0x22E804 )// whitelisted control codes
    // do something else
}
```

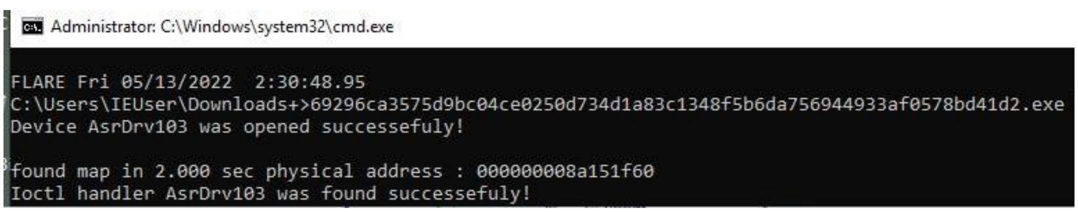
The "***find_patch_address()***" function obtains information about the physical memory by reading the "***HKLM\Hardware\ResourceMap\System Resources\Physical Memory***" system registry key. Next, it exploits the **AsRock** driver to read the physical memory pages and search for 160 bytes of assembly code located in that memory ("***original_asrock_function_fragment***"). This assembly code fragment is the beginning of one of the functions located within the **AsRock** driver itself – it is one of the unencrypted ioctl handlers that can be reached with the I/O control code ***0x22E858*** (here, we call this function "***ioctl_22E858_handler()***"):

```
  else
  {
    if ( (_DWORD)LowPart == 0x22E858 )
      goto LABEL_78;
    if ( (_DWORD)LowPart != 0x22E860 && (_DWORD)LowPart != 0x22E800 && (_DWORD)LowPart != 0x22E804 )
      LowPart = v111;
  }
```

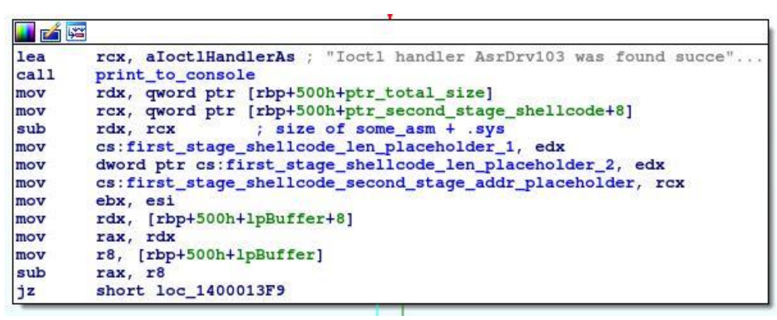
```
LABEL_78:
      v81 = ioctl_22E858_handler((unsigned __int16 *)MasterIrp);
      a2->IoStatus.Status = v81;
      if ( v81 >= 0 )
        a2->IoStatus.Information = 516i64;
      goto LABEL_147;
```

Below, we show the assembly snippet that illustrates the logic that searches for the physical memory address:

After the physical memory address of the **AsRock** ioctl handle of interest has been found, the binary outputs the following message and passes this address further down its logic:



Another shellcode fragment located in the *.data* section of the binary (we call it "***first_stage_shellcode***") is used to overwrite the contents of the ioctl handler within the **AsRock** driver (the one discussed above). We will explain the details of this fragment later. However, before the ioctl handler within the **AsRock** driver is patched, the "***first_stage_payload_shellcode***" gets some adjustments:



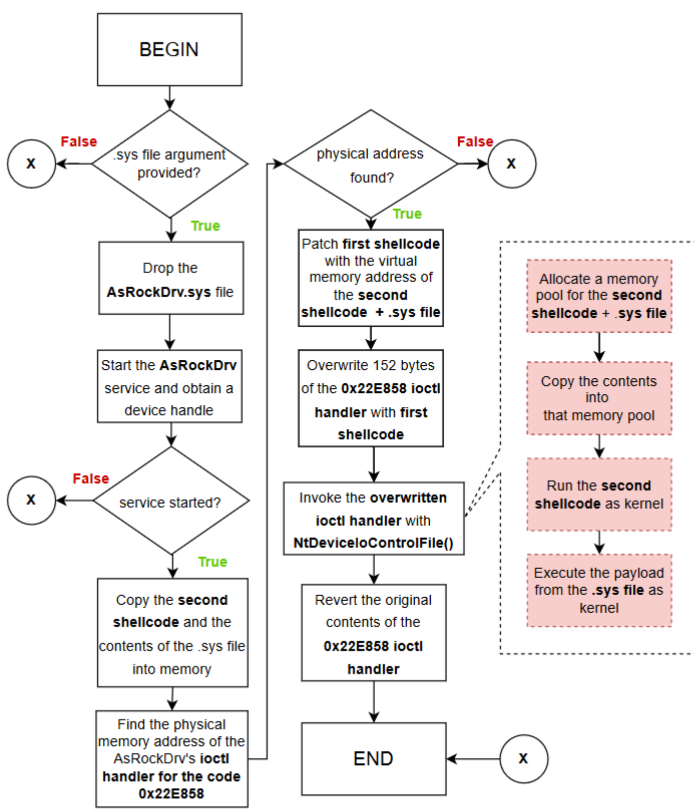
Specifically, the total length of the *.sys* file (the argument to the binary) and the second stage shellcode gets inserted into two places, and the virtual memory address of the second shellcode fragment gets inserted as well. Once the first stage shellcode is adjusted, the binary exploits the **AsRock** driver again to write the shellcode into the physical memory at the location where the "***ioctl_22E858_handler()***" function of the **AsRock** driver is loaded. Then it invokes the modified handler, executing the first stage shellcode within the privileged process of the **AsRock** driver (calling the handler via the NtDeviceIoControlFile() function). Immediately after, the "***ioctl_22E858_handler()***" contents are reverted back to the original code ("***original_asrock_function_fragment***") to ensure the stability of the system in case the ioctl handler is called by other drivers/services.

```
loc_1400013F9:
mov     [rbp+500h+rwargs.phys_addr], rdi
mov     [rbp+500h+rwargs.length], 152
mov     [rbp+500h+rwargs.granularity], 2
lea     rax, first_stage_shellcode
mov     [rbp+500h+rwargs.buff], rax
lea     r9, [rbp+500h+rwargs]
lea     rdx, [rbp+500h+rwargs]
mov     ecx, 22E80Ch     ; OPCODE_WRITE_PHYSMEM
call    driver_call
lea     rcx, aIoctlHandlerPa ; "ioctl handler patched!\n"
call    print_to_console
lea     rcx, aTryStartShell ; "try start shell  \n"
call    print_to_console
xor     eax, eax
mov     [rbp+500h+rwargs.phys_addr], rax
mov     qword ptr [rbp+500h+rwargs.length], rax
mov     dword ptr [rsp+600h+lpdwTagId], 500h
lea     rax, [rbp+500h+ReOpenBuff]
mov     [rsp+600h+lpLoadOrderGroup], rax
mov     dword ptr [rsp+600h+lpBinaryPathName], 500h
lea     rax, [rbp+500h+ReOpenBuff]
mov     qword ptr [rsp+600h+dwErrorControl], rax
mov     [rsp+600h+dwStartType], 22E858h
lea     rax, [rbp+500h+rwargs]
mov     [rsp+600h+lpOverlapped], rax
xor     r9d, r9d
xor     r8d, r8d
xor     edx, edx
mov     rcx, cs:asrock_driver_handle
call    cs:NtDeviceIoControlFile
mov     [rbp+500h+rwargs.phys_addr], rdi
mov     [rbp+500h+rwargs.length], 160
mov     [rbp+500h+rwargs.granularity], 2
lea     rax, original_asrock_function_fragment
mov     [rbp+500h+rwargs.buff], rax
lea     r9, [rbp+500h+rwargs]
lea     rdx, [rbp+500h+rwargs]
mov     ecx, 22E80Ch     ; OPCODE_WRITE_PHYSMEM
call    driver_call
jmp     short loc_1400014D0
```

The adjusted first stage shellcode is shown on the snippet below. When the patched "***ioctl_22E858_handler()***" function is triggered, it allocates a memory pool of the size **"*sizeof(second_stage_shellcode) + sizeof(argument .sys file)"*** using the function *ExAllocatePoolWithTag();* then, it copies the contents of the buffer that holds the second stage shellcode and the *.sys* file from the process of the malware sample into that memory pool. Finally, it executes the second stage shellcode with kernel privileges. It looks like this assembly fragment was written by hand.

```
            sub    rsp,0x48
            lea    rcx,[rip+0x2245]
            mov    rcx,QWORD PTR [rcx]
            mov    QWORD PTR [rsp+ExAllocatePoolWithTag],rcx
            lea    rcx,[rip+0x2226]
            mov    rcx,QWORD PTR [rcx]
            mov    QWORD PTR [rsp+ExFreePoolWithTag],rcx
            mov    r8d,0x31313131                      # ULONG Tag
            mov    edx,0x1abfb                         # SIZE_T NumberOfBytes <--- sizeof(.sys file) + sizeof(shellcode_2)
            xor    ecx,ecx                             # POOL_TYPE PoolType
            call   QWORD PTR [rsp+ExAllocatePoolWithTag] # ExAllocatePoolWithTag(POOL_TYPE PoolType, SIZE_T NumberOfBytes, ULONG Tag)
            mov    QWORD PTR [rsp+allocated_mem],rax   # PVOID P
            cmp    QWORD PTR [rsp+allocated_mem],0x0
----------- je     LOC_EXIT
|           mov    DWORD PTR [rsp+counter],0x0
| --------- jmp    LOC_START_COPY
| | ->LOC_COPY_LOOP:
| | |       mov    eax,DWORD PTR [rsp+counter]
| | |       inc    eax
| | |       mov    DWORD PTR [rsp+counter],eax
| --->LOC_START_COPY:
|   |       cmp    DWORD PTR [rsp+counter],0x1abfb     # sizeof(.sys file) + sizeof(shellcode_2)
| --------- jge    LOC_EXIT_COPY_LOOP:
| | |       movsxd rax,DWORD PTR [rsp+counter]
| | |       movsxd rcx,DWORD PTR [rsp+counter]
| | |       movabs rdx,0x27cb3f1eb80                   # addr of the second stage shellcode in memory
| | |       mov    r8,QWORD PTR [rsp+allocated_mem]
| | |       movzx  eax,BYTE PTR [rax+rdx*1]
| | |       mov    BYTE PTR [r8+rcx*1],al
| | ------- jmp    LOC_COPY_LOOP
| --->LOC_EXIT_COPY_LOOP:
|           call   QWORD PTR [rsp+allocated_mem]       # jump to the first address in the allocated pool (copy of shellcode_2)
|           mov    edx,0x31313131                      # ULONG Tag
|           mov    rcx,QWORD PTR [rsp+allocated_mem]   # PVOID P
----->LOC_EXIT:
            call   QWORD PTR [rsp+ExFreePoolWithTag]   # ExFreePoolWithTag(PVOID P, ULONG Tag)
            add    rsp,0x48
            ret
```

The second stage shellcode consists of a common kernel shellcode pattern for resolving NT kernel API addresses by hash (see here, for example) and functionality to load and invoke the argument-supplied unsigned driver. While this unsigned driver is missing, it seems highly likely this is a kernel-level rootkit component and possibly works in conjunction with the implant referred to as ICECORE by Mandiant and DUSTTUNNEL by Dragos. The diagram below illustrates the simplified execution flow of the sample:



It is peculiar to see that while the malicious actors behind this tool were clearly inspired by the original proof-of-concept exploit for CVE-2020-15368, there are some crucial differences between the original and the present implementation. That the malicious actors managed to easily weaponize someone's work is worrisome and serves as another argument in favor of formal vulnerability disclosure and response practices.

2.2.2. Codecall/Evilscholar

According to the available reports, the PLCs possibly targeted by the Codecall toolset mostly fall within the Schneider Electric Machine Expert product family, formerly called SoMachine. Machine Expert PLCs are relatively low-cost PLCs used in machine automation for motion control, mechatronics, and motor and drive management purposes.

The table below lists the reportedly targeted controllers and protocols in addition to vulnerabilities that have been identified.

Controller	Targeted Protocols	Identified Vulnerabilities
M221	Machine Expert Discovery (27126/UDP, 27127/UDP) Modbus TCP (502/TCP)	https://www.ndss-symposium.org/wp-content/uploads/bar2019_74_Kalle_paper.pdf https://www.osti.gov/servlets/purl/1808195 https://www.sciencedirect.com/science/article/pii/S2666281722000051 http://www.people.vcu.edu/~iahmed3/publications/ifip_sec_2019_attack.pdf https://www.cisa.gov/uscert/ics/advisories/ICSA-17-089-02 https://www.se.com/ww/en/download/document/SEVD-2018-233-01/ https://www.se.com/ww/en/download/document/SEVD-2018-235-01/ https://www.se.com/ww/en/download/document/SEVD-2018-270-01/ https://www.se.com/ww/en/download/document/SEVD-2019-045-01/ https://www.se.com/ww/en/download/document/SEVD-2020-315-05/
M241 M251	Machine Expert Discovery (27126/UDP, 27127/UDP) Machine Expert CODESYS (1740-1743/UDP, 1105/TCP) Modbus TCP (502/TCP)	https://www.cisa.gov/uscert/ics/advisories/ICSA-17-089-02 https://download.schneider-electric.com/files?p_Doc_Ref=SEVD-2021-130-05 https://www.se.com/ww/en/download/document/SEVD-2020-105-02/ https://www.se.com/ww/en/download/document/SEVD-2019-134-02/
M238	Modbus TCP (via TwidoPort gateway	-

	module) (502/TCP)	
M258	Machine Expert Discovery (27126/UDP, 27127/UDP) Machine Expert CODESYS (1740-1743/UDP, 1105/TCP) Modbus TCP (502/TCP)	https://www.se.com/ww/en/download/document/SEVD-2020-105-02/ https://www.se.com/ww/en/download/document/SEVD-2019-134-02/
LMC058 LMC078	Machine Expert Discovery (27126/UDP, 27127/UDP) Machine Expert CODESYS (1740-1743/UDP, 1105/TCP) Modbus TCP (502/TCP)	https://www.se.com/ww/en/download/document/SEVD-2019-134-02/

As shown in the table above, most likely because of its comparative affordability, the Machine Expert product family has seen quite a bit of public security research, in particular the M221. This has resulted in a sizeable body of information on the internals, proprietary protocols and uncovered vulnerabilities in these products that an attacker could weaponize. According to prior reports, Codecall possesses at least the following capabilities (and possibly more):

- Discover and identify Machine Expert PLCs over the network using the Discovery protocol
- Brute-force PLC passwords using CODESYS
- Using CODESYS functionality to enumerate, download, upload and delete files
- Sever legitimate connections to the PLC, possibly to facilitate credential capture
- Manipulating IP routing information
- Trigger a DoS on the PLC requiring a power cycle and configuration recovery
- Send Modbus commands to read/write registers, request IDs, etc.

Machine Expert Discovery

The Machine Expert Discovery protocol is a proprietary Schneider Electric protocol for discovery, identification and network configuration of Machine Expert PLCs. While the protocol is ostensibly encrypted, this is done with a hardcoded key and a weak algorithm (as covered by CVE-2019-6820) allowing rogue clients to abuse this protocol for discovery and configuration manipulation purposes.

CODESYS

CODESYS is one of the most popular IEC 61131-3 logic runtime environments and is used by dozens of vendors across the world. Both its V2 and V3 incarnations and the myriad security issues have been well documented by public security research. As such, attackers with capabilities for the CODESYS environment and protocol could potentially target products by multiple vendors, meaning defenders will need to be aware of any CODESYS-based assets in their inventories rather than simply focus on Machine Expert products.

Modbus

The Modbus protocol is one of the most ubiquitous and famously insecure-by-design OT protocols in existence. Off-the-shelf capabilities to interact with Modbus can be found all over the internet and, as such, are nothing special in and of themselves. The harder part of carrying out OT-oriented attacks leveraging Modbus lies in understanding a given PLC's internal Modbus map, which maps Modbus addresses to internal variables and I/Os. Without this understanding, an attacker is forced to either guess, brute-force or infer this mapping from long-term network traffic and operations surveillance. However, retrieving the PLC's configuration through CODESYS, as described above, will provide the attacker with these mappings.

Another item of interest is that the Machine Expert Basic series (which includes the M221), contrary to the wider Machine Expert family, does not use the CODESYS protocol but instead uses a Machine Expert Basic dialect of the proprietary Schneider Electric UMAS Modbus extension (function code 0x5A). While UMAS has been the subject of quite some public security research and the Machine Expert Basic extension has not, there still is some common functionality. As such, it seems interesting that no capabilities for this protocol appear to have been integrated into Codecall. This could either be a result of the target set's demands (focusing on Machine Expert with the basic series being of lesser interest) or could point to capability modules that have not yet been recovered.

2.2.3. Omshell/Badomen

According to the available reports, the devices possibly targeted by the Omshell toolset are related to machine automation, including machine controllers from the NJ and NX series, servo drives, fieldbus couplers and power supplies.

The table below lists the reportedly targeted devices and protocols, in addition to vulnerabilities that have been identified.

Controller	Targeted Protocols	Identified Vulnerabilities
NJ501-1300	Omron FINS (9600/TCP, 9600/UDP) HTTP (80/TCP) Telnet	https://www.cisa.gov/uscert/ics/advisories/icsa-19-346-03

NX1P2	Omron FINS (9600/TCP, 9600/UDP) HTTP (80/TCP) Telnet	-
NX-SL3300	-	-
NX-ECC203	-	-
R88D-1SN10F-ECT	-	-
S8VK	-	-

As shown in the table above, compared to the Schneider Electric Machine Expert or older Omron Cx family of PLC, the NJ and NX series have not seen much public security research, indicating the attacker likely had to invest significant efforts into developing capabilities for these platforms. According to prior reports, Omshell possesses at least the following capabilities and possibly more:

- Scan for Omron PLCs using the FINS protocol
- Interact with Omron PLC web services using HTTP
- Enumerate and communicate with devices (e.g., servo drives or power supplies) nested behind PLCs
- Backup and restore Omron PLC configurations
- Wipe and reset Omron PLCs
- Activate telnet service on Omron PLCs and use it to upload and execute binaries
- Deploy an additional Omron PLC-native implant for additional fine-grained capabilities

Omron FINS

The Omron Factory Interface Network Service (FINS) is a proprietary but publicly well-documented protocol for PLC communication and engineering operations among the popular Omron Cx and NJ/NX series. While this protocol has some security features, these are typically not enabled and have historically suffered from bypass flaws. The FINS protocol can be used for a wide array of potentially dangerous operations ranging from PLC enumeration and discovery to starting and stopping the PLC, reading and writing logic and memory, manipulating and deleting files, and wiping and resetting the PLC.

Device Nesting

The reported ability of Omshell to enumerate and interact with devices nested behind PLCs is of particularly novel interest. Typically, PLCs control instruments or clusters of secondary PLCs via serial or industrial Ethernet-based fieldbus networks nested behind them. These devices are typically not directly addressable by attackers residing in IP-based OT networks if no pass-through protocol features are available. At most, they can be controlled in a limited fashion through whatever variables are mapped and exposed by the master PLCs.

An attacker seeking to achieve more complicated effects, including disabling safety systems, could possibly need the ability to control these nested devices more directly, which would require them to take over the master PLC acting as a bridge. The Omshell ability to achieve code execution *on* the PLC and deploy an implant could hint at the desire to develop such fine-grained capabilities. Such implants would have to be tailored to the particular PLC platform (in case of many NJ and NX series PLCs this seems to be a combination of x86, QNX and/or Windows) and could persist for an indefinite amount of time due to the complete lack of endpoint security measures, introspection or forensics capabilities on PLCs.

2.2.4. Tagrun/Mousehole

The third OT-oriented component of INCONTROLLER is an OPC UA toolkit referred to as Tagrun. This toolkit is capable of identifying OPC UA servers, connecting to them using either default or attacker-supplied credentials and enumerating OPC UA structures which include configurations, tags and control points. This serves a potential dual purpose of discovery, reconnaissance and process comprehension on the one hand and the ability to manipulate tag values to affect operations on the other.

3. IoCs

IoC	Type	Description
7062403bccacc7c0b84d27987b204777f6078319c3f4caa361581825c1a94e87	File hash	SHA256 hash of the Industroyer2 sample from the original incident (CERT-UA)
ea16cb89129ab062843c84f6c6661750f18592b051549b265aaf834e100cd6fc	File hash	SHA256 hash of one of the Industroyer2 samples (public sources)
fc0e6f2effbfa287217b8930ab55b7a77bb86dbd923c0e8150551627138c9caa	File hash	SHA256 hash of the CaddyWiper sample from the original incident (CERT-UA)
43d07f28b7b699f43abd4f695596c15a90d772bfbd6029c8ee7bc5859c2b0861	File hash	SHA256 hash of the OrcShred sample from the original incident (CERT-UA)
bcdf0bd8142a4828c61e775686c9892d89893ed0f5093bdc70bde3e48d04ab99	File hash	SHA256 hash of the AwfulShred sample from the original incident (CERT-UA)
1724a0a3c9c73f4d8891f988b5035effce8d897ed42336a92e2c9bc7d9ee7f5a	File hash	SHA256 hash of the TailJump sample from the original incident (CERT-UA)

cda9310715b7a12f47b7c134260d5ff9200c147fc1d05f030e507e57e3582327	File hash	SHA256 hash of the ArguePatch sample from the original incident (CERT-UA)
69296ca3575d9bc04ce0250d734d1a83c1348f5b6da756944933af0578bd41d2	File hash	SHA256 hash of a Lazycargo sample from vx-underground
C:\Users\User1\Desktop\dev_projects\SignSploit1\x64\Release\AsrDrv_exploit.pdb	String	Path to the debug symbols found in a Lazycargo sample
HKLM\Hardware\ResourceMap\System Resources\Physical Memory	Windows registry key	A Windows registry key accessed by a Lazycargo sample
“PService_PPD.exe”	String	Name of the service/executable to be stopped and renamed in the infected machine
“D:\OIK\DevCounter”	String	Path where the service/executable to be stopped and renamed is located
91.245.255[.]243	IP address	Potentially, an IP address related to the initial access (according to CERT-UA)
195.230.23[.]19	IP address	Potentially, an IP address related to the initial access (according to CERT-UA)
C:\Users\peremoga.exe JRIBDFIMCQAKVBBP C:\Users\pa1.pay reg save HKLM\SYSTEM C:\Users\Public\sys.reg /y reg save HKLM\SECURITY C:\Users\Public\sec.reg /y reg save HKLM\SAM C:\Users\Public\sam.reg /y \\%DOMAIN%\sysvol\%DOMAIN%\Policies\%GPO ID%\Machine\zrada.exe \\%DOMAIN%\sysvol\%DOMAIN%\Policies\%GPO ID%\Machine\pa.pay C:\Windows\System32\rundll32.exe C:\windows\System32\comsvcs.dll MiniDump %PID% C:\Users\Public\mem.dmp full	Host-based indicators of compromise	Host-based indicators of compromise from the original incident (CERT-UA)

C:\Windows\Temp\link.ps1 C:\Users\peremoga.exe C:\Users\pa1.pay C:\Dell\vatt.exe C:\Dell\pa.pay C:\Dell\108_100.exe C:\tmp\cdel.exe		

4. Mitigation Recommendations

CISA recommends to:

- Isolate ICS/SCADA systems and networks from corporate networks and the internet. Limit network connections to only specifically allowed management and engineering workstations.
- Enforce multifactor authentication for remote access to ICS networks and devices whenever possible. Change passwords to ICS/SCADA devices on a consistent schedule. Only use admin accounts when required for specific tasks.
- Leverage an OT monitoring solution to alert on malicious indicators and behaviors, watching internal systems and communications for known hostile actions.
- Investigate symptoms of denial of service or delays in communications processing as signs of potential malicious activity.
- Monitor systems for loading of unusual drivers, particularly for ASRock driver if no ASRock driver is normally used on the system.
- Maintain backups for faster recovery after disruptive attacks.

More detailed recommendations are available on CISA alerts AA22-110A, AA22-103A and AA22-083A. Windows driver developers should also follow standard security guidelines to prevent exploitation.

5. References

- https://www.welivesecurity.com/2022/04/12/industroyer2-industroyer-reloaded/
- https://pylos.co/2022/04/23/industroyer2-in-perspective/
- https://www.securonix.com/blog/industroyer2-caddywiper-targeting-ukrainian-power-grid/
- https://www.emanueledelucia.net/industroyer2-the-ics-capable-malware-re-emerges-in-order-to-cause-critical-services-disruption/
- https://www.mandiant.com/resources/industroyer-v2-old-malware-new-tricks
- https://www.netresec.com/?page=Blog&month=2022-04&post=Industroyer2-IEC-104-Analysis
- https://www.nozominetworks.com/blog/industroyer2-nozomi-networks-labs-analyzes-the-iec-104-payload/
- https://www.cisa.gov/uscert/ncas/alerts/aa22-103a
- https://www.mandiant.com/resources/incontroller-state-sponsored-ics-tool
- https://www.dragos.com/blog/industry-news/chernovite-pipedream-malware-targeting-industrial-control-systems/
- https://download.schneider-electric.com/files?p_Doc_Ref=SESB-2022-01
- https://customers.codesys.com/index.php?eID=dumpFile&t=f&f=17113&token=0c173ece4a2f48bd30d6a67fa2f495119d5caefc&download