WHITE PAPER

Industroyer vs. Industroyer2: Evolution of the IEC 104 Component

AUTHORS
Giannis Tsaraias
Ivan Speziale
Nozomi Networks Labs is dedicated to reducing cyber risk for the world’s industrial and critical infrastructure organizations. Through its cybersecurity research and collaboration with industry and institutions, it helps defend the operational systems that support everyday life.

The Labs team conducts investigations into industrial device vulnerabilities and, through a responsible disclosure process, contributes to the publication of advisories by recognized authorities.

To help the security community with current threats, they publish timely blogs, research papers and free tools.

The Threat Intelligence and Asset Intelligence services of Nozomi Networks are supplied by ongoing data generated and curated by the Labs team.

To find out more, and subscribe to updates, visit nozominetworks/labs
# Table of Contents

1. Introduction to Industroyer & Industroyer2  
2. Industroyer & Industroyer2: The Evolving Source Code  
   2.1 Breaking Down the Samples  
   2.2 v2 Station Configuration  
   2.3 v2 IOA Configuration  
   2.4 v2 Command-line Parameters  
   2.5 v2 IEC 104 Interaction  
   2.6 Main Thread Spawning  
   2.7 TESTFR Frame Inserted in v2  
   2.8 Start/Stop Data Transfer Activation  
   2.9 Prepare/Send Station Command  
   2.10 Use of Streaming SIMD Extensions (SSE) Instructions  
   2.11 Parse_packet_and_log Function  
3. Summary  
4. Addendum: YARA Rule for Industroyer2  
6. References and Related Reading
1. Introduction to Industroyer & Industroyer2

Industroyer2 is the latest evolution of the notorious malware that was first deployed by threat actor Sandworm in Ukraine in 2016. As documented by ESET, this new artifact was used in the context of a broader operation against Ukrainian organizations in 2022.\(^1\)

The Industroyer artifacts retrieved in 2016 consisted of components targeting multiple industrial control system (ICS) protocols, specifically:

- IEC 60870-5-101,
- IEC 60870-5-104,
- IEC 61850,
- OPC DA.

Industroyer2, however, focuses only on IEC 60870-5-104 (IEC 104), which is just an update to the Industroyer component targeting the same protocol. This observation leads us to believe that, depending on the operational requirements, the threat actors’ implementation of these ICS protocols is part of a broader framework of capabilities that is selectively packaged into a specific deliverable.

In this paper, Nozomi Networks Labs analyzes the Operational Technology (OT) capabilities of Industroyer2, discusses the major changes between Industroyer and Industroyer2, and analyzes how the codebase has evolved over time.

A noteworthy characteristic of Industroyer deployments is the lack of any stealthy measures in the binaries. One plausible hypothesis is that the threat actor, having already compromised the target environment and performed advanced reconnaissance, is not concerned about potential security controls.

A second hypothesis is that due to time constraints, the operators would not have time to simultaneously obfuscate their activity and improve their posture in the environment by the time of malware delivery. Given the resources and expertise of the threat actor, we believe this scenario to be less likely. Nevertheless, it is clear that Sandworm is not concerned about different Industroyer versions being attributed to the same actor through comparison of the target artifacts.

The takeaway for security teams is that advanced threat actors are continuously refining their OT capabilities to adapt to different operational scenarios. In the current threat landscape it’s paramount to detect and respond to sophisticated attackers before they reach OT system—their ability to analyze the targeted environment and modify its status was demonstrated once more with Industroyer2.
2. Industroyer & Industroyer2: The Evolving Source Code

2.1 Breaking Down the Samples

In this section, we present a series of evidence that collectively and strongly supports the thesis that the two binaries, Industroyer and Industroyer2, were compiled from the same evolving source code.

Throughout our analysis, we will refer to the first version of Industroyer as “v1”, which corresponds to sample 7907dd95c1d36c7f8424a1bf8d809aef68f4b3d382c23a3c974a383cad (104.dll). We will refer to Industroyer2 as “v2”, which corresponds to sample d69665f56def7adae71971f066432e9f1510a7194386e5f8e8926aea7b88e08.

The screenshot below (Figure 1) compares similar functionalities in the binaries. The decompiled code of v1 is presented on the left while the matching part of v2 is shown on the right.

![Figure 1 - Example comparison between Industroyer v1 (left) and v2 (right).](image)

The syntax of the configuration is the most obvious visual difference between the two versions of the malware. However, this refactor is largely irrelevant for the internal structure of the executables. In both cases the configuration is normalized into a matching data structure, called `main_config` in our analysis, that is then used throughout the code.

As described by ESET, Industroyer v1 uses a classic INI configuration file that is passed as an argument to the Crash export of 104.dll. Meanwhile, the Industroyer v2 sample that we analyzed hardcodes its configuration inside the binary in the form of a string.

Below, we present the possible properties for the hardcoded station and Information Object Address (IOA) configurations embedded in the analyzed binary.
2. Industroyer & Industroyer2: The Evolving Source Code

2.2 v2 Station Configuration

The following screenshot (Figure 2) shows the first hardcoded station configuration embedded in the analyzed binary of v2. The sample embeds configurations for three different IP addresses in total.

Below (Figure 3), we present the possible properties for the hardcoded station and IOA configurations embedded in the analyzed binary of v2.

<table>
<thead>
<tr>
<th>Property</th>
<th>Acceptable Values</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target IP</td>
<td>IP address</td>
<td>IP of the station to connect to</td>
</tr>
<tr>
<td>Target port</td>
<td>Port number</td>
<td>Port of the station to connect to</td>
</tr>
<tr>
<td>ASDU</td>
<td>Integer</td>
<td>Application Service Data Unit address</td>
</tr>
<tr>
<td>Operation mode</td>
<td>Boolean</td>
<td>0 (Interaction with hardcoded IOA), 1 (Range mode)</td>
</tr>
<tr>
<td>Switch for process manipulation</td>
<td>Boolean</td>
<td>0 (Disable), 1 (Enable)</td>
</tr>
<tr>
<td>Reserved parameter</td>
<td>Boolean</td>
<td>-</td>
</tr>
<tr>
<td>Process name</td>
<td>String</td>
<td>Name of the process to be killed</td>
</tr>
<tr>
<td>Rename</td>
<td>Boolean</td>
<td>0 (Don’t rename), 1 (Rename)</td>
</tr>
<tr>
<td>Folder name</td>
<td>String</td>
<td>Folder name where the process targeted for killing and renaming is stored</td>
</tr>
<tr>
<td>Sleep time in minutes</td>
<td>Integer</td>
<td>Initial sleep time, used to add a delay before interacting with a station</td>
</tr>
<tr>
<td>Sleep time in seconds #1</td>
<td>Integer</td>
<td>Sleep time to use when invert SCO/DCO On/Off is set</td>
</tr>
<tr>
<td>Station index</td>
<td>Integer</td>
<td>Configuration station index to delay</td>
</tr>
<tr>
<td>Sleep time in seconds #2</td>
<td>Integer</td>
<td>Sleep time before STOPDT for the previously used station index</td>
</tr>
<tr>
<td>Initial SCO/DCO On/Off State</td>
<td>Boolean</td>
<td>0 (Initial state On), 1 (Initial state Off)</td>
</tr>
<tr>
<td>Invert SCO/DCO On/Off</td>
<td>Boolean</td>
<td>If set, it will interact with each IOA again, with SCO/DCO On/Off inverted</td>
</tr>
<tr>
<td>IOA count</td>
<td>Integer</td>
<td>Number of IOA following header</td>
</tr>
</tbody>
</table>
2.3 v2 IOA Configuration

An IOA is used to address one specific piece of data within a station. IOA configurations typically differ from station to station. In the screenshot below (Figure 4) you can see the IEC-104 testbed traffic using the first station configuration.

![Image](image_url)

**Figure 4 - IEC 104 testbed traffic using first station configuration.**

The table below (Figure 5) shows the configurable IOA properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Acceptable Values</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOA</td>
<td>Integer</td>
<td>Information Object Address</td>
</tr>
<tr>
<td>Single/Double command</td>
<td>Boolean</td>
<td>0 (Double command), 1 (Single command)</td>
</tr>
<tr>
<td>SCO/DCO Select/Execute</td>
<td>Boolean</td>
<td>0 (Execute), 1 (Select)</td>
</tr>
<tr>
<td>SCO/DCO On/Off</td>
<td>Boolean</td>
<td>0 (Off), 1 (On)</td>
</tr>
<tr>
<td>Priority</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>IOA entry index in the configuration list</td>
</tr>
</tbody>
</table>

**Figure 5 - IOA Configuration.**
2. Industroyer & Industroyer2: The Evolving Source Code

2.4 v2 Command-line Parameters

While v1 included a separate component to load and launch payloads contained in different Dynamic-link Libraries (DLLs), the v2 sample provides the user with the ability to set certain command-line options.

As shown in Figure 6, two command-line flags are supported by the v2 executable; namely, -o and -t. The -o flag can be used to store the execution output log into a file instead of printing it to standard output. The -t flag can be used to perform a delayed execution. For example, running the program with -t 10 as an argument at 1:08 PM will cause a time delay of approximately two minutes before the executable spawns its main thread at 1:10 PM.

```c
if (phNumArgs)
{
  if (lphOptChar)
  {
    v2 = log_to_shared_file(v2, lphOptChar);
    wait_on[0][v2];
    close_handles[0];
  }
  if (string_sanity_checks(hWnd, hWnd + 4 * phNumArgs, L"-o"))
  {
    if (v11)
    {
      lock);
      set_filename_shared_0(v11);
    }
    v2 = lock();
    outer_write_to_shared_file(v2, "hello", 22);
  }  
  lpCmdLine = 4;
  vh = outer_process_heap_alloc(248);
  if (v30)
  {
    v9 = initialize_critical_section(v30);
    vh = lock();
  lock_thread_spawning_after_pServiceControl_renamed = v9;
  for (i = 1; i < 3; i++)
  {
    lpCmdLine = config[i];
    par_e_config_kill_rename_process_spawn_thread(lpCmdLine);
    outer_killing_threads_closing_handle(lock_thread_spawning_after_pServiceControl_renamed, 0, 0);
  }
  v6 = lock_thread_spawning_after_pServiceControl_renamed;
  v9 = lock_thread_spawning_after_pServiceControl_renamed;
  if (lock_thread_spawning_after_pServiceControl_renamed)
  {
  lock_thread_spawning_after_pServiceControl_renamed();
  if (lock_thread_spawning_after_pServiceControl_renamed)
  v2 = delete_critical_section_and_free(v7, i);
  }
  v5 = 0;
  LockFree(hWnd);
  ExitProcess(0);
```

Figure 6 - Command-line argument handling.
2. Industroyer & Industroyer2: The Evolving Source Code

2.5 v2 IEC 104 Interaction

After terminating `PServiceControl.exe`, and `PService_PPD.exe` (based on the configuration) being renamed with `.MZ` appended to its name, the v2 sample begins IEC 104 interaction.

```c
for ( dwProcessId = find_proc_by_name("PServiceControl.exe");
         dwProcessId; )
    dwProcessId = find_proc_by_name(main_config->second_switch_process_name) ;

    hProcess = OpenProcess(1u, 0, dwProcessId );
    TerminateProcess(hProcess, 0);
    //
    //
    while ( 1 )
    {
        dwProcess = find_proc_by_name(main_config->second_switch_process_name);
        if ( !dwProcess )
            break;
        handle = OpenProcess(1u, 0, dwProcess);
        TerminateProcess(handle, 0);
    }

    if ( main_config->second_switch_p3_set_if_set_copy_to_foldername && main_config != 0xFFFEEEE0 )
    {
        lpExistingFileName = process_heap_alloc(0x100u);
        lpNewFileName = process_heap_alloc(0x100u);
        lpMem = process_heap_alloc(0x100u);

        memset(lpExistingFileName, 0xff, 256);
        memset(lpNewFileName, 0xff, 256);
        memset(lpMem, 0xff, 256);

        copy_string(lpExistingFileName, main_config->second_switch_foldername);
        copy_string(lpNewFileName, main_config->second_switch_foldername);

        strcat(lpExistingFileName, "]\"");
        strcat(lpNewFileName, "]\"");

        lpExistingFileName = strcat(lpExistingFileName, main_config->second_switch_process_name);
        lpNewFileName = strcat(lpNewFileName, main_config->second_switch_process_name);
        lpExistingFileName = strcat(lpExistingFileName, ".MZ");

        lpNewFileName = strcat(lpNewFileName, ".MZ");

        MoveFileA(lpExistingFileName, lpNewFileName);

        LastError = GetLastError();
        lock = ::lock();
        write_to_shared_file_0(lock, " MZ %Ax \n", LastError);
        process_heap_free(lpExistingFileName);
        process_heap_free(lpNewFileName);
    }
return 1;
```

Figure 7 - Process termination and file renaming.
The default operation mode (0) set in the station configurations present in our sample produces the following series of commands:

- **TESTFR**
- **STARTDT**
- **C_IC_NA_1 (100)**
- **STOPDT**

For each IOA configuration:
- **C_SC_NA_1 (45)** or **C_DC_NA_1 (46)** command, depending on the Single/Double command field in the configuration

If the operation mode is set to 1 instead, the sample expects to find a starting index and an ending index following the station configuration, which is then used as a range of IOAs to iterate through. In this case, the following series of commands are generated in our testbed:

- **TESTFR**
- **STARTDT**
- **C_IC_NA_1 (100)**
- **STOPDT**

For each IOA in the range `start_index → end_index`:
- **C_DC_NA_1 (46)** with DCO Off and Select
- **C_DC_NA_1 (46)** with DCO Off and Execute

In Figures 8a and 8b, we show both Single and Double commands for range modes starting with 2 and ending with 9:

![Command Example](image)

**Figure 8a** - Range mode with start index 2 and end index 9, Single commands.
### Figure 8b - Range mode with start index 2 and end index 9, Single commands.

<table>
<thead>
<tr>
<th>IEC</th>
<th>60870-5-104</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>60870-5-104</td>
<td>50 &lt;- U</td>
<td>(TESTFR act)</td>
</tr>
<tr>
<td>42</td>
<td>60870-5-104</td>
<td>50 &lt;- U</td>
<td>(STARTDT act)</td>
</tr>
<tr>
<td>44</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(0,0) ASDU=3 C_DC_NA_1 Act I0A=0</td>
</tr>
<tr>
<td>49</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(3)</td>
</tr>
<tr>
<td>50</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(1,3) ASDU=3 C_DC_NA_1 Act I0A=2</td>
</tr>
<tr>
<td>53</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(4)</td>
</tr>
<tr>
<td>55</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(2,4) ASDU=3 C_DC_NA_1 Act I0A=2</td>
</tr>
<tr>
<td>52</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(5)</td>
</tr>
<tr>
<td>56</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(3,5) ASDU=3 C_DC_NA_1 Act I0A=3</td>
</tr>
<tr>
<td>58</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(6)</td>
</tr>
<tr>
<td>59</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(4,6) ASDU=3 C_DC_NA_1 Act I0A=3</td>
</tr>
<tr>
<td>61</td>
<td>60870-5 ASDU</td>
<td>50 &lt; S</td>
<td>(7)</td>
</tr>
<tr>
<td>62</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(5,7) ASDU=3 C_DC_NA_1 Act I0A=4</td>
</tr>
<tr>
<td>64</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(8)</td>
</tr>
<tr>
<td>65</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(6,8) ASDU=3 C_DC_NA_1 Act I0A=4</td>
</tr>
<tr>
<td>68</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(9)</td>
</tr>
<tr>
<td>69</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(7,9) ASDU=3 C_DC_NA_1 Act I0A=5</td>
</tr>
<tr>
<td>71</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(10)</td>
</tr>
<tr>
<td>72</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(8,10) ASDU=3 C_DC_NA_1 Act I0A=5</td>
</tr>
<tr>
<td>74</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(11)</td>
</tr>
<tr>
<td>75</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(9,11) ASDU=3 C_DC_NA_1 Act I0A=6</td>
</tr>
<tr>
<td>77</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(12)</td>
</tr>
<tr>
<td>78</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(10,12) ASDU=3 C_DC_NA_1 Act I0A=6</td>
</tr>
<tr>
<td>81</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(13)</td>
</tr>
<tr>
<td>82</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(11,13) ASDU=3 C_DC_NA_1 Act I0A=7</td>
</tr>
<tr>
<td>84</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(14)</td>
</tr>
<tr>
<td>85</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(12,14) ASDU=3 C_DC_NA_1 Act I0A=7</td>
</tr>
<tr>
<td>87</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(15)</td>
</tr>
<tr>
<td>88</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(13,15) ASDU=3 C_DC_NA_1 Act I0A=8</td>
</tr>
<tr>
<td>90</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(16)</td>
</tr>
<tr>
<td>91</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(14,16) ASDU=3 C_DC_NA_1 Act I0A=8</td>
</tr>
<tr>
<td>94</td>
<td>60870-5 ASDU</td>
<td>50 &lt; S</td>
<td>(17)</td>
</tr>
<tr>
<td>95</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(15,17) ASDU=3 C_DC_NA_1 Act I0A=9</td>
</tr>
<tr>
<td>97</td>
<td>60870-5 ASDU</td>
<td>50 &lt;- S</td>
<td>(18)</td>
</tr>
<tr>
<td>98</td>
<td>60870-5 ASDU</td>
<td>60 &lt;- I</td>
<td>(16,18) ASDU=3 C_DC_NA_1 Act I0A=9</td>
</tr>
<tr>
<td>100</td>
<td>60870-5-104</td>
<td>50 &lt;- S</td>
<td>(19)</td>
</tr>
<tr>
<td>101</td>
<td>60870-5-104</td>
<td>50 &lt;- U</td>
<td>(STOPDT act)</td>
</tr>
</tbody>
</table>
2.6 Main Thread Spawning

The main thread of both samples contains the code responsible for issuing the malicious IEC 104 packets. In v1, the main thread is spawned from the Crash export, while in v2 the execution starts from the regular PE entry point. In both cases the configuration is parsed before reaching this stage (Figure 9).

Beginning with this function, the usage of a structure dubbed `main_config` in our decompilation (Figure 10), becomes pervasive throughout the codebase. In both the samples this structure operates as the main glue between the configuration and the rest of the code, independently from the configuration format used.

The way in which `main_config` is used is a strong indicator of how the two executables were compiled from the same source code and updated over time.
2. Industroyer & Industroyer2: The Evolving Source Code

2.7 TESTFR Frame Inserted in v2

TESTFR frames in IEC 104 are used between the controlling station and the controlled station to periodically check the status of a connection and eventually detect communication problems. After having established a TCP connection, Industroyer v1 begins emitting STARTDT frames. This marks the beginning of a data transfer from the controlling station to the controlled station.

Industroyer v2, instead, takes the extra step of sending a TESTFR frame as we can also observe in the traffic dump (Figure 12).

![Main thread comparison.](image1.png)

![TESTFR frame in Industroyer v2.](image2.png)
2.8 Start/Stop Data Transfer Activation

The functions responsible for creating and sending \texttt{STARTDT} and \texttt{STOPDT} frames are essentially the same across the two executables. We can spot minor differences in the way dynamic memory is allocated, but the only functional difference is a sleep timeout. In v1, it is customizable through the configuration, and in v2 is hardcoded to one second for both the functions.

![STARTDT frame creation.](image)

We can also observe how the invocation of function \texttt{parse_received_packet} varies slightly between v1 and v2 of the malware. From a functional perspective, the most important update is the ability to reply to \texttt{TESTFR} activation commands with \texttt{TESTFR} confirmation frames.

![STOPDT frame creation.](image)
2.9 Prepare/Send Station Command

The function named in our decompilation as `iec104_prepare_and_send_station_command` (Figure 15) is found in both versions of the malware with similar semantics. Nevertheless, we can appreciate how in v2 the function can receive more IEC 104 parameters to properly customize the packet payload.

![Figure 15 - Function iec104_prepare_and_send_station_command invocation.](image)

A plausible reason for this v2 function is that at first the developers designed an abstraction that satisfied the initial requirements, which over time changed to incorporate more flexibility. This is also the general feeling that an analyst gets when assessing the evolution of this codebase. A first rough version is developed to achieve a specific goal and it eventually morphed into a full-fledged framework to surgically manipulate IEC 104 payloads.
2.10 Use of Streaming SIMD Extensions (SSE) Instructions

Some of the IEC 104 commands are assembled from a bytes template that is hardcoded in the binaries. The curiosity is that in v1 these bytes are handled with x86 SSE instructions, while in v2 regular non-SSE instructions are used instead. This is typically due to the threat actor choosing different optimization settings upon compilation (Figure 16).

The function dubbed `Parse_packet_and_log` used in the malware provides some basic dissection of the packets received from the controlled station in response to the issued IEC 104 commands. We discovered an interesting typo introduced in Industroyer v2 (line 164) where the `STOPDT con` string is logged rather than the correct `STOPDT act`, as found in Industroyer v1 (Figure 17).

Although this typo does not have functional consequences, it is an interesting artifact that can seldom be found in a refactored codebase.

Figure 16 - Different compiler optimizations between the v1 and v2.

Figure 17 - Function Parse_packet_and_log.
There are a couple of functions used in `Parse_packet_and_log` which map a code (cause and typeid) to a verbose string description. For unknown reasons, the body of these functions has been removed from the v2 executable. It is extremely unlikely that this is due to the fear of being detected, as there are no such precautions throughout the malware. We speculate that this might be due to some pre-processor directives.

![Figure 18 - Function Parse_packet_and_log.](image-url)
3. Summary

We conducted a comparative analysis of the artifact known as Industroyer2 against the first deployment of the same toolkit. The evidence presented strongly suggests that the threat group is updating the codebase over time to meet operational requirements as they evolve.

Additionally, we provided a thorough breakdown of the configuration format used by Industroyer2, illustrating the different options available to customize the behavior of the IEC 104 payload.

Finally, we want to highlight a major difference between advanced threat actors and more ordinary adversaries. Sophisticated operators can not only compromise targets in-depth to reach the OT network, but they also have the technical capabilities to analyze the targeted environment and craft custom tools to manipulate OT systems.

4. Addendum: YARA Rule for Industroyer2

Below is a YARA rule for Industroyer2:

```yara
// Created by Nozomi Networks Labs

rule industriy2_nn {
  meta:
    author = "Nozomi Networks Labs"
    name = "Industroyer2"
    description = "Industroyer2 malware targeting power grid components."
    actor = "Sandworm"
    hash = "D69665F56DEDF7AD4E71971F06432E59F1510A7194386E5F0E8926AE7B88E00"

  strings:
    $s1 = "%02d:%lS" wide ascii
    $s2 = "PService_PPD.exe" wide ascii
    $s3 = "D:\OIK\DevCounter" wide ascii
    $s4 = "MSTR ->> SLV" fullword wide ascii
    $s5 = "MSTR <-> SLV" fullword wide ascii
    $s6 = "Current operation : %s"
    $s7 = "Switch value: %s"
    $s8 = "Unknown APDU format !!!"
    $s9 = "Length:%u bytes |"
    $s10 = "Sent=x%x | Received=x%x"
    $s11 = "ASDU:%u | OA:%u | IOA:%u |"
    $s12 = "Cause: %s (x%X) | Telegram type: %s (x%X)"

  condition:
    5 of them
}
```
6. References and Related Reading


Related Reading

- “Cyberattack by Sandworm Group (UAC-0082) on Ukrainian energy facilities using malicious programs INDUSTROYER2 and CADDYWIPER (CERT-UA # 4435),” CERT-UA, April 12, 2022.
Nozomi Networks
The Leading Solution for OT and IoT Security and Visibility

Nozomi Networks accelerates digital transformation by protecting the world’s critical infrastructure, industrial and government organizations from cyber threats. Our solution delivers exceptional network and asset visibility, threat detection, and insights for OT and IoT environments. Customers rely on us to minimize risk and complexity while maximizing operational resilience.