

Operation 'Harvest': A Deep Dive into a Long-term Campaign

 mcafee.com/blogs/enterprise/mcafee-enterprise-atr/operation-harvest-a-deep-dive-into-a-long-term-campaign/

ARCHIVED STORY

By **Christiaan Beek** · September 14, 2021

A special thanks to our Professional Services' IR team, [ShadowServer](#), for historical context on C2 domains, and Thomas Roccia/Leandro Velasco for malware analysis support.

Executive Summary

Following a recent Incident Response, McAfee Enterprise's Advanced Threat Research (ATR) team worked with its Professional Services IR team to support a case that initially started as a malware incident but ultimately turned out to be a long-term cyber-attack.

From a cyber-intelligence perspective, one of the biggest challenges is having information on the tactics, techniques, and procedures (TTPs) an adversary is using and then keeping them up to date. Within ATR we typically monitor many adversaries for years and collect and store data, ranging from indicators of compromise (IOCs) to the TTPs.

In this report, ATR provides a deep insight into this long-term campaign where we will map out our findings against the Enterprise MITRE ATT&CK model. There will be parts that are censored since we respect the confidentiality of the victim. We will also zoom in and look at how the translation to the MITRE Techniques, historical context, and evidence artifacts like PlugX and Winnti malware led to a link with another campaign, which we highly trust to be executed by the same adversary.

IOCs that could be shared are at the end of this document.

McAfee customers are protected from the malware/tools described in this blog. MVISION Insights customers will have the full details, IOCs and TTPs shared via their dashboard. MVISION Endpoint, EDR and UCE platforms provide signature and behavior-based prevention and detection capability for many of the techniques used in this attack. A more detailed blog with specific recommendations on using the McAfee portfolio and integrated partner solutions to defend against this attack can be found [here](#).

Technical Analysis

Initial Infection Vectors [TA0001]

Forensic investigations identified that the actor established initial access by compromising the victim's web server [T1190]. On the webserver, software was installed to maintain the presence and storage of tools [T1105] that would be used to gather information about the victim's network [T1083] and lateral movement/execution of files [T1570] [T1569.002]. Examples of the tools discovered are PSEXEC, Procdump, and Mimikatz.

Privilege Escalation and Persistence [TA0004, TA0003]

The adversary has been observed using multiple privilege escalation and persistence techniques during the period of investigation and presence in the network. We will highlight a few in each category.

Besides the use of Mimikatz to dump credentials, the adversaries used two tools for privilege escalations [T1068]. One of the tools was "RottenPotato". This is an open-source tool that is used to get a handle to a privileged token, for example, "NT AUTHORITY\SYSTEM", to be able to execute tasks with System rights.

Example of RottenPotato on elevating these rights:

 Figure 1. RottenPotato

Figure 1. RottenPotato

The second tool discovered, "BadPotato", is another open-source tool that can be used to elevate user rights towards System rights.


 Figure 2. BadPotato

Figure 2. BadPotato

The BadPotato code can be found on GitHub where it is offered as a Visual Studio project. We inspected the adversary's compiled version using DotPeek and hunted for artifacts in the code. Inspecting the File (COFF) header, we observed the file's compilation timestamp:

| *TimeDateStamp: 05/12/2020 08:23:47 – Date and time the image was created*

PlugX

Another major and characteristic privilege escalation technique the adversary used in this long-term campaign was the malware PlugX as a backdoor. PlugX makes use of the technique "DLL Sideloaded" [T1574.002]. PlugX was observed as usual where a single (RAR) executable contained the three parts:

- Valid executable.
- Associated DLL with the hook towards the payload.
- Payload file with the config to communicate with Command & Control Server (C2).

The adversary used either the standalone version or distributed three files on different assets in the network to gain remote control of those assets. The samples discovered and analyzed were communicating towards two domains. Both domains were registered during the time of the campaign.

One of the PlugX samples consisted of the following three parts:

Filename	Hashes
HPCustPartic.exe	SHA256: 8857232077b4b0f0e4a2c3bb5717fd65079209784f41694f8e1b469e34754cf6
HPCustPartUI.dll	SHA256: 0ee5b19ea38bb52d8ba4c7f05fa1ddf95a4f9c2c93b05aa887c5854653248560
HPCustPartic.bin	SHA256: 008f7b98c2453507c45dacd4a7a7c1b372b5fafc9945db214c622c8d21d29775

The .exe file is a valid and signed executable and, in this case, an executable from HP (HP Customer participation). We also observed other valid executables being used, ranging from AV vendors to video software. When the executable is run, the DLL next to it is loaded. The DLL is valid but contains a small hook towards the payload which, in our case, is the .bin file. The DLL loads the PlugX config and injects it into a process.

We executed the samples in a test setup and dumped the memory of the machine to conduct memory analysis with volatility. After the basic forensically sound steps, we ran the malfind plugin to detect possible injected code in a process. From the redacted output of the plugin, we observed the following values for the process with possible injected code:

```
Process: svchost.exe Pid: 860 Address: 0xb50000
Process: explorer.exe Pid: 2752 Address: 0x56a000
Process: svchost.exe Pid: 1176 Address: 0x80000
Process: svchost.exe Pid: 1176 Address: 0x190000
Process: rundll32.exe Pid: 3784 Address: 0xd0000
Process: rundll32.exe Pid: 3784 Address: 0x220000
```

One observation is the mention of the SVCHOST process with a ProcessID value of 1176 that is mentioned twice but with different addresses. This is similar to the RUNDLL32.exe that is mentioned twice with PID 3785 and different addresses. One way to identify what malware may have been used is to dump these processes with the relevant PID using the procdump module, upload them to an online analysis service and wait for the results. Since this is a very sensitive case, we took a different approach. Using the best of both worlds (volatility and Yara) we used a

ruleset that consists of malware patterns observed in memory over time. Running this ruleset over the data in the memory dump revealed the following (redacted for the sake of readability) output:

 Figure 3. Output Yarascan memory dump

Figure 3. Output Yarascan memory dump

The output of the Yara rule scan (and there was way more output) confirmed the presence of PlugX module code in PID 1176 of the SVCHOST service. Also, the rule was triggered on PID 3784, which belonged to RUNDLL32.exe.

Investigating the dumps after dynamic analysis, we observed two domain names used for C2 traffic:

- sery.brushupdata.com
- dnssery.brushupdata.com

In particular, we saw the following hardcoded value that might be another payload being downloaded:

```
| sery.brushupdata.com/CE1BC21B4340FEC2B8663B69
```

The PlugX families we observed used DNS [T1071.001] [T1071.004] as the transport channel for C2 traffic, in particular TXT queries. Investigating the traffic from our samples, we observed the check-in-signature (“20 2A 2F 2A 0D”) that is typical for PlugX network traffic:

```
| 00000000:    47 45 54 20 2F 42 34 42 42 44 43 43 30 32 39 45
| 00000010:    31 31 39 37 31 39 46 30 36 35 36 32 32 20 48 54
| 00000020:    54 50 2F 31 2E 31 0D 0A 41 63 63 65 70 74 3A 20
| 00000030:    2A 2F 2A 0D 0A 43 6F 6F 6B 69 65 3A 20 44 36 43
| 00000040:    57 50 2B 56 5A 47 6D 59 6B 6D 64 6D 64 64 58 55
| 00000050:    71 58 4D 31 71 31 6A 41 3D 0D 0A 55 73 65 72 2D
```

During our analysis of the different PlugX samples discovered, the domain names as mentioned above stayed the same, though the payload values were different. For example:

- hxxp://sery.brushupdata.com/B4BBDCC029E119719F065622
- hxxp://sery.brushupdata.com/07FDB1B97D22EE6AF2482B1B
- hxxp://sery.brushupdata.com/273CDC0B9C6218BC1187556D

Other PlugX samples we observed injected themselves into Windows Media Player and started a connection with the following two domains:

- center.asmlbigip.com

- sec.asmlbigip.com

Hello Winnti

Another mechanism observed was to start a program as a service [T1543.003] on the Operating System with the acquired System rights by using the *Potato tools. The file the adversary was using seemed to be a backdoor that was using the DLL file format (2458562ca2f6fabddae8385cb817c172).

The DLL is used to create a malicious service and its name is “*service.dll*”. The name of the created service, “SysmainUpdate”, is usurping the name of the legitimate service “SysMain” which is related to the legitimate DLL *sysmain.dll* and also to the Superfetch service. The dll is run using the command “`rundll32.exe SuperFrtch.dll, #1`”. The export function has the name “WwanSvcMain”.

The model uses the persistence technique utilizing *svchost.exe* with *service.dll* to install a rogue service. It appears that the dll employs several mechanisms to fingerprint the targeted system and avoid analysis in the sandbox, making analysis more difficult. The DLL embeds several obfuscated strings decoded when running. Once the fingerprinting has been done, the malware will install the malicious service using the API *RegisterServiceHandlerA* then *SetServiceStatus*, and finally *CreateEventA*. A description of the technique can be found [here](#).

The malware also decrypts and injects the payload in memory. The following screenshot shows the decryption routine.

 Figure 4. Decryption routine

Figure 4. Decryption routine

When we analyzed this unique routine, we discovered similarities and the mention of it in a publication that can be read [here](#). The malware described in the article is attributed to the Winnti malware family. The operating method and the code used in the DLL described in the article are very similar to our analysis and observations.

The process dump also revealed further indicators. Firstly, it revealed artifacts related to the DLL analyzed, “C:\ProgramData\Microsoft\Windows\SuperfRtch\SuperfRtch.dat”. We believe that this dat file might be the loaded payload.

Secondly, while investigating the process dump, we observed activities from the backdoor that are part of the data exfiltration attempts which we will describe in more detail in this analysis report.

A redacted snippet of the code would look like this:

```
Creating archive ***.rar
Adding [data from location]
0%
OK
```

Another indicator of discovering Winnti malware was the following execution path we discovered in the command line dump of the memory:

```
cmd /c klcsngtgui.exe 1560413F7E <abbreviation-victim>.dat
```

What we observed here was the use of a valid executable, the AES 256 decryption key of the payload (.dat file). In this case, the payload file was named using an abbreviation of the victim company's name. Unfortunately, the adversary had removed the payload file from the system. File carving did not work since the disk/unallocated space was overwritten. However, reconstructing traces from memory revealed that we were dealing with the Winnti 4.0 malware. The malware was injected into a SVCHOST process where a driver location pointed to the config file. We observed in the process dump the exfiltration of data on the system, such as OS, Processor (architecture), Domain, Username, etc.

Another clue that helped us was the use of DNS tunneling by Winnti which we discovered traces of in memory. The hardcoded 208.67.222.222 resolves to a legitimate OpenDNS DNS server. The IP is pushed into the list generated by the malware at runtime. At the start of the malware, it populates the list with the system's DNS, and the OpenDNS server is only used as a backup to ensure that the C2 domain is resolved.

Another indicator in the process dump was the setup of the C2 connection including the User-Agent that has been observed being used by Winnti 4.0 malware:

```
Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/57.0.2987.133 Safari/537.36
```

Other Persistence Activities

WMI activity [T1546.003] was also observed to execute commands on the systems.

From a persistence point of view, scheduled tasks [T1053.005] and the use of valid accounts [T1078] acquired through the use of Mimikatz, or creating LSASS dumps, were observed being employed during the length of the campaign.

Lateral Movement

From a lateral movement perspective, the adversary used the obtained credentials to hop from asset to asset. In one particular case, we observed a familiar filename: "PsExec.exe". This SysInternals tool is often observed being used in lateral movement by adversaries, however, it

can also be used by the sysadmins of the network. In our case, the PsExec executable had a file size of 9.6 MB where the original PsExec (depending on 32- or 64-bit version) had a maximum file size of 1.3 MB. An initial static inspection of the file resulted in a blob of code that was present in the executable which had a very high entropy score (7.99). When running the file from the command line, the following output was observed:

Figure 5. PsExec output

Figure 5. PsExec output

The error notification and the 'Impacket' keyword tipped us off and, after digging around, we found more. The fake PsExec is an open-source Python script that is a PsExec alternative with shell/backdoor capability. It uses a script from this location: [hxxps://github.com/SecureAuthCorp/impacket/blob/master/examples/psexec.pyi](https://github.com/SecureAuthCorp/impacket/blob/master/examples/psexec.pyi). The file is large since it incorporates a low-level protocol interaction from Impacket. The Python library combined with the script code is compiled with py2exe. The file was compiled during the time of the latest attack activities and signed with an expired certificate.

Data Exfiltration

From what we observed, the adversary had a long-term intention to stay present in the victim's network. With high confidence, we believe that the adversary was interested in stealing proprietary intelligence that could be used for military or intellectual property/manufacturing purposes.

The adversary used several techniques to exfiltrate the data. In some cases, batch (.bat) scripts were created to gather information from certain network shares/folders and use the 'rar' tool to compress them to a certain size [T1020] [T1030]. Example of content in a batch script:

```
| C:\Windows\web\rar.exe a -[redacted] -r -v50000 [Target-directory]
```

On other occasions, manual variants of the above command were discovered after using the custom backdoor as described earlier.

When the data was gathered on a local system using the backdoor, the files were exfiltrated over the backdoor and the rar files were deleted [T1070.004]. Where external facing assets were used, like a web server, the data was stored in a location in the Internet Information Services (IIS) web server and exfiltrated over HTTP using GET requests towards the exact file paths [T1041] [T1567] [T1071].

An example of the [redacted] web traffic in the IIS logfiles:

Date /Time	Request	TCP Src port	Source IP	User-Agent
Redacted	GET /****/[redacted].rar	80	180.50.*.*	MINIXL
redacted	GET /****/[redacted].rar	80	209.58.*.*	MINIXL

The source IP addresses discovered belonged to two different ISP/VPN providers based in Hong-Kong.

The User-Agent value is an interesting one, "MINIXL". When we researched that value, we discovered a blog from Dell SecureWorks from 2015 that mentions the same User-Agent, but also a lot of the artifacts mentioned from the blog overlapped with the observations and TTPs of Operation Harvest [\[link\]](#).

What we could retrieve from open-source databases is that the use of this particular User-Agent is very limited and seems to originate from the APAC region.

Who did it?

That seems to be the one-million-dollar question to be asked. Within McAfee, attribution is not our main focus, protecting our customers is our priority. What we do care about is that if we learn about these techniques during an investigation, can we map them out and support our IR team on the ground, or a customer's IR team, with the knowledge that can help determine which phase of the attack the evidence is pointing to and based on historical data and intelligence, assist in blocking the next phase and discover more evidence?

We started by mapping out all MITRE ATT&CK Enterprise techniques and sub-techniques, added the tools used, and did a comparison against historical technique data from the industry. We ended up with four groups that shared techniques and sub-techniques. The Winnti group was added by us since we discovered the unique encryption function in the custom backdoor and indicators of the use of the Winnti malware.

 Figure 6. ATT&CK technique comparison

Figure 6. ATT&CK technique comparison

The diagram reflecting our outcome insinuated that APT27 and APT41 are the most likely candidates that overlap with the (sub-)techniques we observed.

Since all these groups are in a certain time zone, we extracted all timestamps from the forensic investigation with regards to:

- Registration of domain
- Compile timestamps of malware (considering deception)
- Timestamps of command-line activity
- Timestamps of data exfiltration
- Timestamps of malware interaction such as creation, deletion, etc.

When we converted all these timestamps from UTC to the aforementioned groups' time zones, we ended up with the below scheme on activity:

 Figure 7. Adversary's time of operation

Figure 7. Adversary's time of operation

In this campaign, we observed how the adversary mostly seems to work from Monday to Thursday and typically during office hours, albeit with the occasional exception.

Correlating ATT&CK (sub-)techniques, timestamps, and tools like PlugX and Mimikatz are not the only evidence indicators that can help to identify a possible adversary. Command-line syntax, specific code similarity, actor capability over time versus other groups, and unique identifiers are at the top of the 'pyramid of pain' in threat intelligence. The bottom part of the pyramid is about hashes, URLs, and domains, areas that are very volatile and easy to change by an adversary.

 Figure 8. Pyramid of Pain

Figure 8. Pyramid of Pain

Beyond investigating those artifacts, we also took possible geopolitical interests and potential deception into consideration when building our hypothesis. When we mapped out all of these, we believed that one of the two previously mentioned groups were responsible for the campaign we investigated.

Our focus was not about attribution though, but more around where the flow of the attack is, matches against previous attack flows from groups, and what techniques/tools they are using to block next steps, or where to locate them. The more details we can gather at the top of 'the pyramid of pain', the better we can determine the likely adversary and its TTP's.

That's all Folks!

Well, not really. While correlating the observed (sub-)techniques, the malware families and code, we discovered another targeted attack against a similar target in the same nation with the major motivation of gathering intelligence. In the following diagram we conducted a high-level comparison of the tools being used by the adversary:

 Figure 9. Tools comparison

Figure 9. Tools comparison

Although some of the tools are unique to each campaign, if taken into consideration over time with when they were used, it makes sense. It demonstrates the development of the actor and use of newer tools to conduct lateral movement and to obtain the required level of user rights on systems.

Overall, we observed the same modus operandi. Once an initial foothold was established, the adversary would deploy PlugX initially to create a few backdoors in the victim's network in case they were discovered early on. After that, using Mimikatz and dumping Isass, they were looking to get valid accounts. Once valid accounts were acquired, several tools including some of their own tools were used to gain information about the victim's network. From there, several shares/servers were accessed, and information gathered. That information was exfiltrated as rar files and placed on an internet-facing server to hide in the 'normal' traffic. We represent that in the following graphic:


 Figure 10. Attack flow

Figure 10. Attack flow

In the 2019/2020 case we also observed the use of a malware sample that we would classify as part of the Winnti malware family. We discovered a couple of files that were executed by the following command:

```
| Start Ins64.exe E370AA8DA0 Jumper64.dat
```

The Winnti loader 'Ins64.exe' uses the value 'E370AA8DA0' to decrypt the payload from the .dat file using the AES-256-CTR decryption algorithm and starts to execute.

After executing this command and analyzing the memory, we observed a process injection in one of the svchost processes whereby one particular file was loaded from the following path:

```
| C:\programdata\microsoft\windows\caches\ieupdate.dll
```

 Figure 11. Memory capture

Figure 11. Memory capture

The malware started to open up both UDP and TCP ports to connect with a C2 server.

UDP Port 20502

TCP Port 20501

 Figure 12. Network connections to C2

Figure 12. Network connections to C2

Capturing the traffic from the malware we observed the following as an example:


 Figure 13. Winnti HTTP traffic to C2

Figure 13. Winnti HTTP traffic to C2

The packet data was customized and sent through a POST request with several headers towards the C2. In the above screenshot the numbers after "POST /" were randomly generated.

The User-Agent is a good network indicator to identify the Winnti malware since it is used in multiple variants:

```
| Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)  
| Chrome/50.0.2661.94 Safari/537.36
```

Indeed, the same User Agent value was discovered in the Winnti sample in Operation Harvest and seems to be typical for this malware family.

The cookie value consists of four Dword hex values that contain information about the customized packet size using a XOR value.

We learned more about the packet structure of Winnti from this [link](#).

Applying what we learned about the handshake, we observed the following in our traffic sample:

Dword value 0 = 52 54 00 36

Dword value 1 = 3e ff 06 b2

Dword value 2 = 99 6d 78 fe

Dword value 3 = 08 00 45 00

Dword value 4 = 00 34 00 47

Initial handshake order:

 Figure 14.

Based on our cross-correlation with samples and other OSINT resources, we believe with a high confidence that this was a Winnti 4.0 sample that connects with a confirmed Winnti C2 server.

The identified C2 server was 185.161.211.97 TCP/80.

Timeline of Events

When analyzing the timestamps from this investigation, like we did for operation Harvest, we came to the below overview:

 Figure 15. Beijing working hours case 2019/2020

Figure 15. Beijing working hours case 2019/2020

Again, we observed that the adversary was operating Monday to Friday during office hours in the Beijing time-zone.

Conclusion

Operation Harvest has been a long-term operation whereby an adversary maintained access for multiple years to exfiltrate data. The exfiltrated data would have either been part of an intellectual property theft for economic purposes and/or would have provided insights that would be beneficial in case of military interventions. The adversaries made use of techniques very often observed in this kind of attack but also used distinctive new backdoors or variants of existing malware families. Combining all forensic artifacts and cross-correlation with historical and geopolitical data, we have high confidence that this operation was executed by an experienced APT actor.

After mapping out all data, TTP's etc., we discovered a very strong overlap with a campaign observed in 2019/2020. A lot of the (in-depth) technical indicators and techniques match. Also putting it into perspective, and over time, it demonstrates the adversary is adapting skills and

evolving the tools and techniques being used.

On a separate note, we observed the use of the Winnti malware. We deliberately mention the term 'malware' instead of group. The Winnti malware is known to be used by several actors. Within every nation-state cyber-offensive activity, there will be a department/unit responsible for the creation of the tools/malware, etc. We strongly believe that is exactly what we observe here as well. PlugX, Winnti and some other custom tools all point to a group that had access to the same tools. Whether we put name 'X' or 'Y' on the adversary, we strongly believe that we are dealing with a Chinese actor whose long-term objectives are persistence in their victims' networks and the acquisition of the intelligence needed to make political/strategic or manufacturing decisions.

MITRE ATT&CK Techniques

Technique ID	Technique Title	Context Campaign
T1190	Exploit Public-facing application	Adversary exploited a web-facing server with application
T1105	Ingress Tool transfer	Tools were transferred to a compromised web-facing server
T1083	File & Directory Discovery	Adversary browsed several locations to search for the data they were after.
T1570	Lateral Tool Transfer	Adversary transferred tools/backdoors to maintain persistence
T1569.002	System Services: Service Execution	Adversary installed custom backdoor as a service
T1068	The exploitation of Privilege Escalation	Adversary used Rotten/Bad Potato to elevate user rights by abusing API calls in the Operating System.
T1574.002	Hijack Execution Flow: DLL Side-Loading	Adversary used PlugX malware that is famous for DLL-Side-Loading using a valid executable, a DLL with the hook towards a payload file.
T1543.003	Create or Modify System Process: Windows Service	Adversary launched backdoor and some tools as a Windows Service including adding of registry keys
T1546.003	Event-Triggered Execution: WMI Event Subscription	WMI was used for running commands on remote systems

T1053.005	Scheduled task	Adversary ran scheduled tasks for persistence of certain malware samples
T1078	Valid accounts	Using Mimikatz and dumping of lsass, the adversary gained credentials in the network
T1020	Automated exfiltration	The PlugX malware exfiltrated data towards a C2 and received commands to gather more information about the victim's compromised host.
T1030	Data transfer size limits	Adversary limited the size of rar files for exfiltration
T1070.004	Indicator removal on host	Where in the beginning of the campaign the adversary was sloppy, during the last months of activity they became more careful and started to remove evidence
T1041	Exfiltration over C2 channel	Adversary used several C2 domains to interact with compromised hosts.
T1567	Exfiltration over Web Service	Gathered information was stored as 'rar' files on the internet-facing server, whereafter they were downloaded by a specific ip range.
T1071.004	Application layer protocol: DNS	Using DNS tunneling for the C2 traffic of the PlugX malware

Indicators of Compromise (IOCs)

Note: the indicators shared are to be used in a historical and timeline-based context, ranging from 2016 to March 2021.

Operation Harvest:

PlugX C2:

sery(.)brushupdata(.)com

Dnssery(.)brushupdata(.)com

Center(.)asmlbigip(.)com

Tools:

Mimikatz

PsExec

RottenPotato

BadPotato

Operation 2019/2020

PlugX malware:

f50de0fae860a5fd780d953a8af07450661458646293bfd0fed81a1ff9eb4498
26e448fe1105b5dadae9b7607e3cca366c6ba8eccf5b6efe67b87c312651db01
e9033a5db456af922a82e1d44afc3e8e4a5732efde3e9461c1d8f7629aa55caf
3124fcb79da0bdf9d0d1995e37b06f7929d83c1c4b60e38c104743be71170efe

Winnti:

800238bc27ca94279c7562f1f70241ef3a37937c15d051894472e97852ebe9f4
c3c8f6befa32edd09de3018a7be7f0b7144702cb7c626f9d8d8d9a77e201d104
df951bf75770b0f597f0296a644d96fbe9a3a8c556f4d2a2479a7bad39e7ad5f

Winnti C2: 185.161.211.97

Tools:

PSW64 6e983477f72c8575f8f3ff5731b74e20877b3971fa2d47683aff11cfd71b48c6
NTDSDumpEx 6db8336794a351888636cb26ebefb52aeaa4b7f90dbb3e6440c2a28e4f13ef96
NBTSCAN c9d5dc956841e000bfd8762e2f0b48b66c79b79500e894b4efa7fb9ba17e4e9e
NetSess ddeeedc8ab9ab3b90c2e36340d4674fda3b458c0afd7514735b2857f26b14c6d
Smbexec e781ce2d795c5dd6b0a5b849a414f5bd05bb99785f2ebf36edb70399205817ee
Wmiexec
14f0c4ce32821a7d25ea5e016ea26067d6615e3336c3baa854ea37a290a462a8

Mimikatz

RAR command-line

TCPdump