# AD Protocols

mercredi 27 novembre 2024 12:18 PM

While Windows operating systems use a variety of protocols to communicate, Active Directory specifically requires Lightweight Directory Access Protocol (LDAP), Microsoft's version of Kerberos, DNS for authentication and communication, and MSRPC which is the Microsoft implementation of Remote Procedure Call (RPC), an interprocess communication technique used for client-server model-based applications.

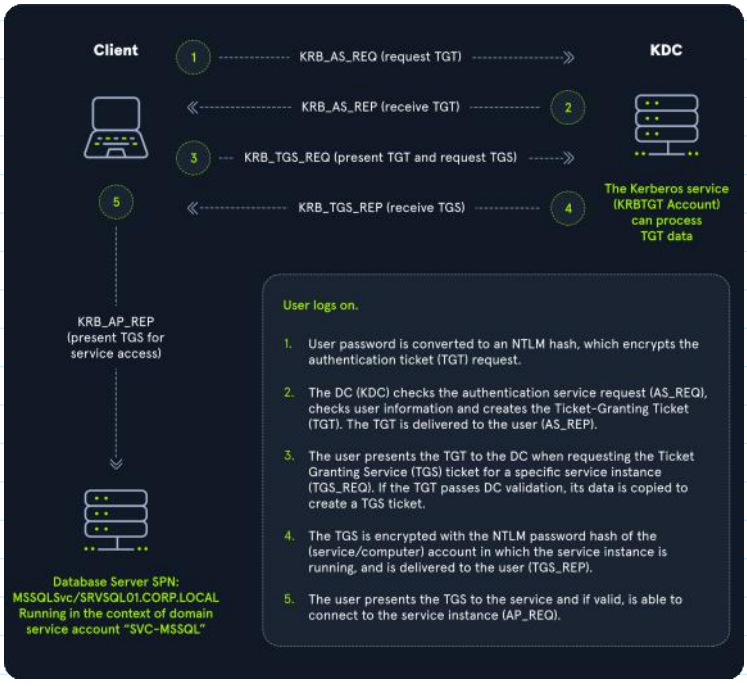
## Kerberos

Kerberos has been the default authentication protocol for domain accounts since Windows 2000. Kerberos is an open standard and allows for interoperability with other systems using the same standard.

The Kerberos protocol **uses port 88** (both TCP and UDP). When enumerating an Active Directory environment, we can often **locate Domain Controllers by performing port scans looking for open port 88** using a tool such as Nmap.

Kerberos authentication effectively decouples users' credentials from their requests to consumable resources, ensuring that their password isn't transmitted over the network (i.e., accessing an internal SharePoint intranet site). The Kerberos Key Distribution Centre (KDC) does not record previous transactions. Instead, the Kerberos Ticket Granting Service ticket (TGS) relies on a valid Ticket Granting Ticket (TGT). It assumes that if the user has a valid TGT, they must have proven their identity. The following diagram walks through this process at a high level.

| | |
|---|---|
| **1** | The **user logs on**, and **their password is converted to an NTLM hash**, which is **used to encrypt the TGT ticket**. This decouples the user's credentials from requests to resources. |
| **2** | The **KDC service on the DC checks** the authentication service request (**AS-REQ**), **verifies** the user information, and **creates** a Ticket Granting Ticket (**TGT**), which is delivered to the user. |
| **3** | The **user presents** the **TGT to the DC**, **requesting** a Ticket Granting Service (**TGS**) ticket **for a specific service**. This is the TGS-REQ. **If** the **TGT** is successfully **validated**, its data is copied to create a **TGS ticket** . |
| **4** | The **TGS is encrypted with the NTLM password hash** of the service or computer account in whose context the service instance is running and is **delivered** to the **user** in the TGS_REP. |
| **5** | The **user presents** the **TGS to the service**, and **if** it is **valid**, the **user** is **permitted** to connect to the resource (AP_REQ). |




## DNS

Active Directory Domain Services (AD DS) uses DNS to allow clients (workstations, servers, and other systems that communicate with the domain) to

locate Domain Controllers and for Domain Controllers that host the directory service to communicate amongst themselves. DNS is used to resolve hostnames to IP addresses and is broadly used across internal networks and the internet. Private internal networks use Active Directory DNS namespaces to facilitate communications between servers, clients, and peers. **AD maintains a database of services running** on the network **in the form of service records (SRV)**. These service records **allow clients in an AD** environment **to locate services that they need**, such as a file server, printer, or Domain Controller.

When a client joins the network, it first needs to find a Domain Controller for authentication and other AD services.

- **DNS Query for SRV Record:**
  - The client queries the **DNS service** for a special type of record called an **SRV record**. This record provides information about available DCs.

- **DNS Response (SRV Record):**
  - The DNS server responds with a list of SRV records. Each record contains:
    - The **hostname** of a Domain Controller (e.g., dc01.domain.com).
    - The priority and weight of the DC.
    - The port number (e.g., 389 for LDAP).

- **Transmitting the DC's Hostname to the Client:**
  - The client retrieves the hostname of the selected Domain Controller from the DNS response.
  - Using this hostname (e.g., dc01.domain.com), the client initiates communication with the Domain Controller for authentication or other AD services.

The client then uses this hostname to obtain the IP address of the Domain Controller. DNS uses TCP and UDP port 53. UDP port 53 is the default, but it falls back to TCP when no longer able to communicate and DNS messages are larger than 512 bytes.

- **Why the Client Resolves the IP Address of the DC**
  - After retrieving the **hostname** of the Domain Controller (e.g., dc01.domain.com) from the SRV record, the client needs the **IP address** to establish a connection to the DC. This is because network communication is based on IP addresses, not hostnames.

- **DNS Server Responds:**
  - The DNS server responds with the corresponding IP address of the hostname.

# LDAP

## What is LDAP in the Context of AD?

Lightweight Directory Access Protocol (LDAP) is a standard protocol that enables systems to interact with directory services.
Active Directory supports **Lightweight Directory Access Protocol (LDAP) for directory lookups** :

1. **Purpose of LDAP:**
   - It provides a way for systems, applications, and services to perform:
     - **Authentication:** Verifying user credentials.
     - **Authorization:** Determining access rights.
     - ***Directory Lookups:*** Querying objects (users, groups, computers, etc.) stored in AD.

   ***LDAP is* the language that applications use to communicate with other servers** that provide directory services. In other words, LDAP is **how systems in the network environment can "speak" to AD.**

2. **Ports Used:**
   - Port **389**: For standard LDAP communication.
   - Port **636**: For secure communication (LDAP over SSL or LDAPS).

The relationship between AD and LDAP can be compared to Apache and HTTP. The same way Apache is a web server that uses the HTTP protocol, Active Directory is a directory server that uses the LDAP protocol.

While uncommon, you may come across organization while performing an assessment that do not have AD but are using LDAP, meaning that they most likely use another type of LDAP server such as OpenLDAP.

## How LDAP Works in AD

1. **Connection Initialization:**
   - The client (e.g., an application, service, or device) initiates a connection to the ***Domain Controller***, which serves as the **LDAP server** (also called a **Directory System Agent**).

2. **LDAP Requests:**
   - Once connected, the client can perform a variety of LDAP operations:
     - **Bind**: Authenticate the client with the server:
     - **Search**: Query AD for specific objects or attributes.
     - **Modify**: Update attributes of objects in the directory.
     - **Add/Delete**: Create or remove objects in the directory.

3. **Authentication:**
   - A common use of LDAP is to verify user credentials during logon or when accessing a resource. This involves the **bind** operation:

     - **Simple Bind (No Kerberos):**
       - The client sends a **username** and **password** directly to the LDAP server.
       - This can occur over plaintext (insecure) or over LDAPS (secure, encrypted).
       - No Kerberos tickets are involved in this process.
     - **SASL Bind (Kerberos is Optional):**
       - **SASL (Simple Authentication and Security Layer)** supports multiple authentication mechanisms, including:
         - **Kerberos** via GSSAPI (Generic Security Services Application Program Interface).
         - Other methods like NTLM or DIGEST-MD5.
       - If you choose **Kerberos (GSSAPI)** as the authentication mechanism, then Kerberos **must** be set up and a valid **TGT (Ticket Granting Ticket)** is required to obtain a **TGS (Ticket Granting Service)** ticket for the LDAP service.
     - **Anonymous Bind (No Kerberos):**
       - The client binds to the LDAP server without providing any credentials.
       - Typically used for public directory lookups where no authentication is needed.

4. **Response:**
   - The LDAP server (AD Domain Controller) responds to client requests with the requested data or the outcome of an operation (success, failure, etc.).

## Why Is LDAP Important in AD Environments?

### For Defenders:

i. **Understanding Authentication Flows:**
   - Knowing how LDAP works helps defenders secure authentication flows and recognize abnormal patterns that might indicate attacks.

ii. **Misconfigurations:**
   - Unsecured LDAP traffic (port 389) transmits data, including credentials, in plaintext. This **can be intercepted by attackers**. Enforcing LDAPS (port 636) mitigates this.

iii. **Audit and Monitoring:**
   - Monitoring LDAP traffic can help detect malicious activities, such as unauthorized queries or modifications to AD.

### For Attackers:

i. **Reconnaissance:**
   - Attackers can **query AD via LDAP to enumerate:**
     - Users and groups.
     - Computers and their configurations.
     - Domain trust relationships.

ii. **Privilege Escalation:**
   - Understanding LDAP operations allows attackers to identify misconfigurations, such as **over-permissive access controls.**

iii. **Persistence:**
   - Attackers might use LDAP **to modify AD objects or attributes to maintain access.**

## Example LDAP Session in AD

1. **Bind Operation:**
   - The client sends credentials (e.g., username@domain.com and password) to the LDAP server for authentication.
2. **Search Operation:**
   - The client queries the LDAP server:

```
Search Base: DC=mydomain,DC=com
Filter: (&(objectClass=user)(sAMAccountName=jdoe))
Attributes: displayName, email
```

3. **LDAP Server Response:**
   - The server returns attributes of the user object (displayName and email) if the client has sufficient permissions.

# NOTICE :

## When Kerberos is Involved

If the client is configured to use **Kerberos authentication** with LDAP (e.g., via SASL/GSSAPI), the following steps occur:

1. **Obtain a TGT:**
   - The client authenticates with the **Key Distribution Center (KDC)** to obtain a **TGT**.
2. **Obtain a Service Ticket for LDAP:**
   - Using the TGT, the client requests a service ticket (TGS) for the LDAP service (e.g., `ldap/dc01.mydomain.com`).
3. **Authenticate to the LDAP Server:**
   - The client presents the service ticket to the LDAP server, which validates it and grants access.

This process ensures secure, ticket-based authentication without directly transmitting passwords.

# Recap :

**LDAP (Lightweight Directory Access Protocol)** is not an authentication mechanism itself; rather, it is a **protocol** used for querying and managing directory services, such as Active Directory (AD), OpenLDAP, or other directory-based systems.

However, LDAP **facilitates authentication** by acting as the protocol through which authentication credentials (like usernames and passwords) are sent and verified against the directory service.

## LDAP's Role in Authentication

LDAP serves as a **communication protocol** that enables applications and systems to:

1. **Search the directory:**
   - Look up user accounts, groups, or other objects in the directory.
2. **Authenticate users:**
   - Facilitate the process of verifying credentials (e.g., during a "bind" operation).
3. **Authorize users:**
   - Retrieve user attributes, such as group memberships, to determine access permissions.

# MSRPC

As mentioned above, MSRPC is Microsoft's implementation of Remote Procedure Call (RPC), an interprocess communication technique used for client-server model-based applications. **Windows systems use MSRPC to access systems in Active Directory** using four key RPC interfaces.

| Interface Name | Functionality | Usage |
|---|---|---|
| **LSARPC** | • Provides access to the **Local Security Authority (LSA)** system, responsible for managing security policies.<br>• Allows operations such as managing **audit policies** and **domain security policies**. | • IT administrators manage security policies.<br>• Attackers could misuse LSARPC to query sensitive security configurations. |
| **Netlogon** | A **background service** that facilitates **authentication** of users, computers, and services in a domain. | • Integral to the Kerberos authentication process and domain logins.<br>• Attackers could abuse it for **authentication-related attacks**, such as exploiting insecure channel bindings or forging service tickets. |

| | | |
|---|---|---|
| **SAMR (Security Account Manager Remote Protocol)** | • Provides access to the **domain account database**, including users, groups, and computers.<br>• Enables **CRUD** (Create, Read, Update, Delete) operations on **domain security principals.**<br><br>**( Security principals are objects in AD that can authenticate** ; *users, computer accounts, or even threads/processes that run in the context of a user or computer account* ) | • Admins use it for account and group management.<br>• **Reconnaissance by Attackers:**<br>  • Tools like **BloodHound** use SAMR to gather information about users, groups, and trust relationships.<br>  • Attackers build "attack paths" to map routes to administrative privileges.<br>• **Defensive Measure:**<br>  • Restrict SAMR queries to administrators by changing the **Windows registry key** RestrictRemoteSAM to block non-admin users. |
| **DRSUAPI** | • Implements the **Directory Replication Service Remote Protocol**, responsible for **replication** between Domain Controllers (DCs).<br><br>• Allows synchronization of AD data (e.g., user accounts, group policies) across DCs. | • Necessary for maintaining consistency across multiple DCs in a domain.<br>• **Exploitation by Attackers:**<br>  • Attackers use tools (e.g., **DCSync**) to extract the **NTDS.dit** database, containing password hashes for all domain accounts.<br>  • Extracted hashes can be used in **Pass-the-Hash attacks** or cracked offline to retrieve cleartext passwords.<br>• **Defensive Measure:**<br>  • Monitor and restrict permissions for replication operations.<br>  • Use **monitoring tools** to detect anomalous replication requests. |

## Attacker Techniques and Defenses

1. **Using SAMR and BloodHound for Reconnaissance:**
   - Attackers enumerate:
     - User accounts.
     - Group memberships.
     - Trust relationships.
   - **Defense:**
     - Restrict SAMR queries to privileged users.

2. **Extracting NTDS.dit with DRSUAPI:**
   - Attackers retrieve:
     - Password hashes for all domain users.
   - **Defense:**
     - Monitor and control replication permissions.
     - Use logging and auditing to detect unauthorized replication attempts.

3. **Netlogon Vulnerabilities:**
   - Exploits like **Zerologon** (CVE-2020-1472) leverage Netlogon flaws to gain domain admin privileges.
   - **Defense:**
     - Patch systems regularly.
     - Monitor Netlogon channel communications.

4. **LSARPC Misuse:**
   - Attackers query sensitive domain security policies.
   - **Defense:**
     - Limit LSA exposure through firewalls and access controls.

# NTLM Authentication

Aside from Kerberos and LDAP, Active Directory uses several other authentication methods which can be used (and abused) by applications and services in AD. These include **LM**, **NTLM**, **NTLMv1**, and **NTLMv2**.

**LM** and **NTLM** here are the **hash names**, and **NTLMv1** and **NTLMv2** are **authentication protocols** that utilize the LM or NT hash. Below is a quick comparison between these hashes and protocols, which shows us that, while not perfect by any means, Kerberos is often the authentication protocol of choice wherever possible. It is essential to understand the difference between the hash types and the protocols that use them.

**Hash Protocol Comparison**

| Hash/Protocol | Cryptographic technique | Mutual Authentication | Message Type | Trusted Third Party |
|---|---|---|---|---|
| **NTLM** | Symmetric key cryptography | No | Random number | Domain Controller |
| **NTLMv1** | Symmetric key cryptography | No | **MD4** hash, random number | Domain Controller |
| **NTLMv2** | Symmetric key cryptography | No | **MD4** hash, random number | Domain Controller |
| **Kerberos** | Symmetric key cryptography & asymmetric cryptography | **Yes** | Encrypted ticket **using DES**, **MD5** | Domain Controller/Key Distribution Center (KDC) |

# HASHES

## LM

- **Old Hashing Mechanism:**
  - Introduced in 1987, **LM hash** was used on early versions of Windows (starting with Windows NT) and OS/2.
  - It was the default password hashing mechanism in Windows versions prior to **Windows Vista/Server 2008**.
- **Password Constraints:**
  - **Length Limitation:** LM hashes only support passwords up to **14 characters**.
  - **Case Insensitivity:** Passwords are **converted to uppercase** before being hashed, meaning they are **not case sensitive**.
  - This severely limits the **keyspace** for brute-forcing, reducing the complexity of potential passwords and making them easier to crack.
- **Hashing Process:**
  - **Splitting:** A 14-character password is divided into **two 7-character chunks**.
  - **Padding:** If the password is shorter than 14 characters, it is padded with **NULL characters** to reach 14.
  - **Encryption:** Each 7-character chunk is used to create a **DES key**, and both chunks are encrypted using the string **KGS!@#$%** (a hardcoded constant). This results in two 8-byte ciphertexts.
  - The two ciphertexts are then concatenated to form the final **LM hash**.
- **Weaknesses:**
  - **Limited Password Space:** The maximum length of passwords is 14 characters, which is very short by modern standards.
  - **Case Insensitivity:** The algorithm converts all characters to uppercase, which significantly reduces the complexity of passwords (only 69 possible characters are used).
  - **Broken DES Encryption:** DES is an outdated and **insecure** encryption algorithm. The LM hash effectively uses a **weak DES-based** approach, making it vulnerable to **brute force attacks**.
  - **Parallel Cracking:** Since the password is split into two 7-character chunks, an attacker only needs to brute-force each chunk separately. This makes cracking LM hashes fast, especially using modern tools like **Hashcat** with GPU acceleration.
- **Default Behavior (Prior to Windows Vista/Server 2008):**
  - Older Windows systems (NT4, Windows 2000, XP, etc.) stored **both the LM hash and the NTLM hash** of the user's password.
  - The LM hash could be easily cracked if attackers gained access to the SAM or NTDS.dit database, which stores password hashes.

### Disabling LM Hashing:

- **Windows Vista and Server 2008 and later** disabled the storage of LM hashes by default due to their weak security.
- However, in **environments with older systems** or misconfigured systems, LM hashes can still be encountered.
- **Group Policy** can be configured to prevent LM hashes from being stored, further improving security in modern environments.

## NTLM (NT LAN Manager) Hashing

NTLM is the authentication protocol **used in modern Windows systems**, with its primary method of storing passwords in hashed form. It's a **challenge-**

**response protocol** **used to authenticate a user**, and **the hashed passwords are stored in the SAM database** (on local machines) or **NTDS.DIT** (on Domain Controllers).

Here are the key aspects of NTLM hashes:

## NTLM Hash Structure:

1. **NT Hash:**
   - The NT hash is the **MD4 hash** of the **UTF-16** encoded password (**in little-endian format**).
   - It is a **stronger** password hash than the older **LM hash**, which was limited to 14 characters and had various weaknesses.
   - Example NT hash: b4b9b02e6f09a9bd760f388b67351e2b.

2. **NTLM Hash Format:**
   - A full NTLM hash entry is usually displayed in a format like this:

     **Rachel**:**500**:**aad3c435b514a4eeaad3b935b51304fe**:**e46b9e548fa0d122de7f59fb6d48eaa2**:::

     **Breaking it down:**
     - **Rachel** is the username.
     - **500** is the **Relative Identifier (RID)** for the user (500 is typically used for the **Administrator** account).
     - **aad3c435b514a4eeaad3b935b51304fe** is the **LM hash** (if LM hashing is enabled).
     - **e46b9e548fa0d122de7f59fb6d48eaa2** is the **NT hash** (this is the one used in modern Windows systems).
     - NT / LM are always represented on 16 bytes ( 128 bits ) since this is the output of MD4.

3. **NTLM Hash vs. LM Hash:**
   - **NTLM hash** is the **stronger** of the two and can store passwords up to **Unicode characters** (65,536 character set), unlike **LM hashes**, which are limited to 14 characters and are case-insensitive.
   - Even though NTLM is stronger than LM, it still has weaknesses that attackers can exploit, such as being susceptible to **pass-the-hash attacks**.

## Security Considerations:

1. **Offline Brute-Force Attacks:**
   - While **NTLM** is more secure than **LM**, it is still **vulnerable** to offline brute-force attacks.
   - **Hashcat** and other tools can be used to crack NTLM hashes offline.
   - With modern GPUs, cracking NTLM hashes (particularly for shorter passwords, such as 8 characters or less) can be done relatively quickly. In fact, **8-character NTLM hashes** can be cracked in **under 3 hours** in some cases.

2. **Pass-the-Hash Attack:**
   - **NTLM hashes** can be used directly in a **pass-the-hash** attack, where the attacker doesn't need the plaintext password.
   - In this attack, the NTLM hash (such as **e46b9e548fa0d122de7f59fb6d48eaa2**) can be used to authenticate to other systems, such as **SMB** services, without needing to know the cleartext password.

```
$ crackmapexec smb 10.129.41.19 -u rachel -H e46b9e548fa0d122de7f59fb6d48eaa2
SMB 10.129.43.9 445 DC01 [*] Windows 10.0 Build 17763 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True) (SMBv1:False)
SMB 10.129.43.9 445 DC01 [+] INLANEFREIGHT.LOCAL\rachel:e46b9e548fa0d122de7f59fb6d48eaa2 (Pwn3d!)
```

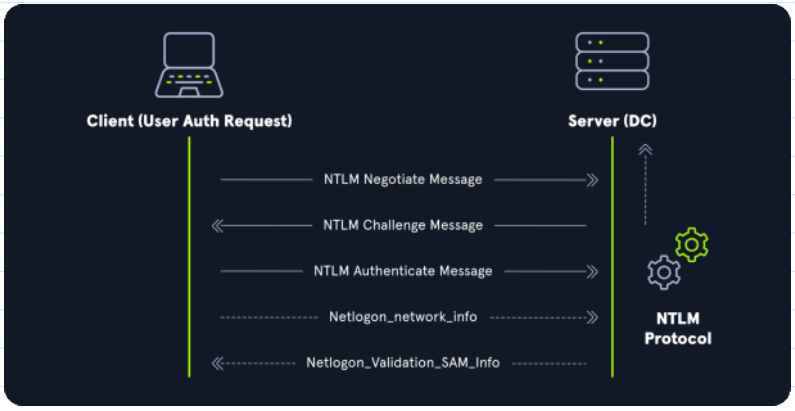
3. **Lack of Salt:**
   - **Neither LM nor NTLM hashes** use a **salt** (a random value added to the password before hashing).
   - This means that if two users have the same password, their hashes will be identical. This vulnerability can be exploited to reduce the complexity of cracking the hashes.

# AUTHENTICATION

# NTLMv1 (Net-NTLMv1) Authentication:

NTLMv1 is an older version of the **NT LAN Manager** authentication protocol, used primarily in Windows environments for network authentication. Although it laid the foundation for modern NTLM authentication, it has significant vulnerabilities that make it less secure today.

## Challenge/Response Mechanism:




NTLMv1 employs a **challenge/response** system where:

1. Client Negotiation Message
2. **The server sends a challenge** (an 8-byte random number) to the client.
3. **The client responds** with a 24-byte value calculated using both the **LM hash** and the **NT hash** (if available).

The response is generated by the following algorithm:

- The challenge is encrypted with the LM and NT hashes, combined with a series of padding and specific constants.
- The **response** is the concatenation of the encrypted challenges (using the DES encryption algorithm):

```
response = DES(K1, C) | DES(K2, C) | DES(K3, C)
```

Where:

- **K1, K2, K3** are derived from the LM and NT hashes. ( K1 | K2 | K3 are portions of the LM/NT hash (16 bytes) , and 5-bytes of 0 are appended to the end of each key to achieve 7 bytes length key requirement for DES. )
- **C** is the 8-byte server challenge (random number).
- The result is a **24-byte response**.

## Structure of a NTLMv1 Hash:

An NTLMv1 hash is created through this challenge/response process. Here's an **example of an NTLMv1 hash:**

```
u4-netntlm::kNS:338d08f8e26de93300000000000000000000000000000000:9526fb8c23a90751cdd619b6cea564742e1e4bf33006ba41:cb8086049ec4736c
```

This hash can be broken down into:

- **u4-netntlm**: Identifier for the hash type (Net-NTLM).
- **kNS**: Some network-specific identifier or flags.
- **338d08f8e26de93300000000000000000000000000000000**: Represents the **LM hash** (if available).
- **9526fb8c23a90751cdd619b6cea564742e1e4bf33006ba41**: Represents the **NT hash**.
- **cb8086049ec4736c**: The **24-byte response** generated from the challenge and the NT/LM hashes.

## Weaknesses and Vulnerabilities:

1. **Cracking**:
   - While **Net-NTLMv1** provides basic challenge/response authentication, it is still vulnerable to offline **brute-force cracking** and **rainbow table attacks** if the hash is captured.
   - The reliance on **DES encryption** and relatively weak response construction makes it easier to crack, especially in systems with weak passwords.
2. **No Pass-the-Hash**:

- Unlike **NTLM** hashes, **Net-NTLMv1** hashes **cannot** be used directly in **pass-the-hash** attacks because the authentication relies on the challenge/response process and not on the static hash alone.
- However, this does not make it completely secure—**NTLM relay attacks** and **capture-and-crack** techniques can still expose vulnerabilities.

3. **Relay Attacks**:
   - **NTLMv1** is vulnerable to **relay attacks**, where the attacker intercepts the response from the client and replays it to authenticate to another system, often without the knowledge of the victim.
4. **Deprecation**:
   - **NTLMv1** has been deprecated in modern versions of Windows, and it is recommended to **disable** NTLMv1 and use stronger alternatives like **NTLMv2** or **Kerberos** for authentication.

# NTLMv2 (Net-NTLMv2) Authentication:

NTLMv2 is a significantly stronger and more secure evolution of the **NTLM** authentication protocol, introduced with **Windows NT 4.0 Service Pack 4** and becoming the default authentication method with **Windows 2000 Server** and later versions. NTLMv2 was designed to address the weaknesses and vulnerabilities in **NTLMv1**, particularly its susceptibility to **spoofing** and **brute-force attacks**. The protocol is widely used in modern Windows environments for **network authentication**.

## Improved Challenge/Response Mechanism:

NTLMv2 addresses the vulnerabilities in **NTLMv1** by using a more complex **challenge/response** mechanism. It relies on the following components:

1. **Server Challenge (SC)**:
   - An **8-byte random value** sent from the server to the client to initiate the challenge.
2. **Client Challenge (CC)**:
   - An **8-byte random value** generated by the client to provide additional randomness to the authentication process.
3. **Modified Client Challenge (CC*)**:
   - This includes the original **client challenge (CC)**, a **timestamp** for temporal binding, a **second client challenge (CC2)**, and the **domain name** to prevent replay attacks and provide more contextual data to enhance security.
4. **NT-Hash**:
   - This is the **MD4 hash of the user's password** encoded in **UTF-16** (as used in the **NTLM** and **NTLMv1** protocols).
5. **HMAC-MD5:**
   - The authentication process uses **HMAC-MD5** (Hashed Message Authentication Code) to combine the **NT-Hash** with the user's **name** and **domain name** to generate a **v2-Hash**.

## NTLMv2 Challenge/Response Flow:

SC = 8-byte server challenge, random
CC = 8-byte client challenge, random
**CC*** = (**CC**, **time**, **CC2**, **domain name**)
**v2-Hash** = HMAC-MD5(**NT-Hash**, **user name**, **domain name**)
LMv2 = HMAC-MD5(**v2-Hash**, SC, CC)
NTv2 = HMAC-MD5(**v2-Hash**, SC, **CC***)
**response** = LMv2 **|** CC **|** NTv2 **|** CC*

## Structure of a NTLMv2 Hash:

Here is an example of a full **NTLMv2 hash**:

```
admin::N46iSNekpT:08ca45b7d7ea58ee:88dcbe4446168966a153a0064958dac6:5c7830315c7830310000000000000b45c67103d0
7d7b95acd12ffa11230e0000000052920b85f78d013c31cdb3b92f5d765c783030
```

This hash can be broken down into:

- **admin**: The **username**.
- **N46iSNekpT**: A **salt** or unique value associated with the password.
- **08ca45b7d7ea58ee**: The **LMv2 response** generated from the challenge and user credentials.
- **88dcbe4446168966a153a0064958dac6**: The **NTv2 response** generated from the modified client challenge and credentials.
- **Additional Data**: Further fields that contain information like domain name and time-based values, ensuring the challenge/response is fresh and unique.

## Security Enhancements:

- **Stronger Hashing**: The use of **HMAC-MD5** significantly enhances the security of the hash over the older **LM** and **NTLM** hashes.
- **Prevention of Replay Attacks**: By including time-based values and domain-specific information in the challenge, NTLMv2 is resistant to **replay attacks** (which are common vulnerabilities in older protocols).
- **Temporal Binding**: The inclusion of **time** in the modified client challenge ensures that the authentication request is tied to a specific period, preventing attackers from reusing old authentication requests.
- **Robust Against Spoofing**: **NTLMv2** is less vulnerable to **man-in-the-middle attacks** and **spoofing** compared to **NTLMv1** due to its more complex and context-sensitive challenge/response system.

## NTLMv2 Hash Cracking:

Despite being significantly more secure than its predecessors, NTLMv2 hashes are still vulnerable to **brute force attacks** and **rainbow table attacks** if the password is weak. Tools like **Hashcat** and **John the Ripper** can still be used to attempt to crack these hashes, but the process is considerably more computationally expensive and time-consuming compared to older NTLM hashes.

**NTLMv2 vs. NTLMv1:**

- **NTLMv2** is a major improvement over **NTLMv1**:
  - **NTLMv2** uses **HMAC-MD5** for stronger hashing and protects against many of the vulnerabilities present in **NTLMv1**.
  - **NTLMv1** is susceptible to **spoofing** and **offline brute-force cracking**.
  - **NTLMv2** binds authentication to both the client and server, making it harder to attack.

| Protocol | Introduced | Encryption/Hashing | Main Features | Vulnerabilities | Best Use Case | Recommendation |
|---|---|---|---|---|---|---|
| **NTLMv1** | Windows NT | **DES** (LM + NT hash) | Simple challenge-response (with 3-way handshake) | Vulnerable to **replay** and **man-in-the-middle** attacks, **brute-force** attacks | Older, non-secure environments | Avoid if possible, use NTLMv2 |
| **NTLMv2** | Windows NT 4.0 SP4 | **HMAC-MD5** (NT hash) | Enhanced challenge-response (**server/client + time-based challenges**) | **Still vulnerable to brute-force attacks** (but more secure than NTLMv1) | Secure networks, environments without Kerberos | Preferred over NTLM, stronger than NTLMv1 |
| **Kerberos** | Windows 2000 | **AES** (Kerberos ticket encryption) | Ticket-based (**TGT + Service Ticket**) | Requires careful configuration, **susceptible to pass-the-ticket attacks** | Default in AD environments, secure authentication | Preferred authentication method for AD environments |
| **LDAP** | Early Windows 2000 (Active Directory) | **None** (plain text, optional TLS) | While Kerberos is the primary authentication protocol in AD, LDAP can be used as a backup or secondary method, especially in legacy environments or for read-only operations. It allows applications to **authenticate users by checking the credentials against the AD database**. | Plain text (without TLS/SSL), susceptible to MITM attacks if not encrypted | Directory queries and read operations | Use over SSL (LDAPS) for secure communication |
| **SAML** | Not AD-specific (used for federated identity) | **XML signature**, encryption (varies) | Federated authentication (single sign-on) | Requires proper configuration and trust relationships | Cross-domain authentication, cloud services | Use for web-based or federated authentication |
| **Smart Card** | Windows 2000+ (with certificates) | **PKI** (Public Key Infrastructure) | Two-factor authentication using smart cards | Dependent on physical card security | High-security environments requiring two-factor auth | Recommended for environments needing strong identity verification |

# Domain Cached Credentials (MSCache2)

In Active Directory (AD) environments, **Domain Cached Credentials (DCC)**—specifically MSCache version 1 and version 2 (MSCache2)—**are used as a solution for situations where a domain-joined computer *cannot communicate with a* domain controller (DC),** such as during network outages or other technical issues. This system allows users to authenticate locally to the computer using previously cached credentials instead of needing to reach the DC for authentication.

## How Domain Cached Credentials Work:

1. **Caching of Credentials**:
   - When a domain user logs into a machine, the **last ten hashes** of domain users that have successfully authenticated are saved locally on the host in the **HKEY_LOCAL_MACHINE\SECURITY\Cache** registry key.
   - This caching allows users to log in even when the computer cannot reach the domain controller (e.g., in offline scenarios).
2. **MSCache v1 vs MSCache v2**:
   - MSCache v2 (also known as DCC2) is a more modern and more secure version compared to MSCache v1.
   - It uses a **stronger hashing algorithm** (MD4) to store the cached passwords, improving the security of these credentials over MSCache v1.
3. **Format of MSCache2 Hashes**:
   - Cached credentials are stored in a specific format, which is different from other NTLM hashes.
   - A typical MSCache2 hash looks like:

     ```
     $DCC2$10240#bjones#e4e938d12fe5974dc42a90120bd9c90f
     ```

     - **10240**: The version of the MSCache algorithm (this identifies it as MSCache2).
     - **bjones**: The username.
     - **e4e938d12fe5974dc42a90120bd9c90f**: The actual hashed value ( MD4 ) .

4. **Hashing Algorithm**:
   - The hashed password is a **slow-to-crack hash**.
   - Even with powerful GPU rigs, attempting to crack the MSCache2 hashes is generally **ineffective** unless there is a **weak password** in use.
5. **Cracking MSCache2 Hashes**:
   - Cracking these hashes using tools like **Hashcat** is difficult due to their complexity.
   - MSCache2 hashes can be slow to crack, making brute-forcing attacks time-consuming and inefficient.
   - Attackers typically need a **targeted approach** to crack these hashes or rely on weak passwords.
6. **Access :**

   **Local Admin Access:** To obtain MSCache2 hashes, an attacker would **need local administrative access to the machine**, which is typically gained through exploiting other vulnerabilities or misconfigurations.