Windows secrets extraction: a summary

synacktiv.com/publications/windows-secrets-extraction-a-summary

Rédigé par Julien Egloff - 20/04/2023 - dans Pentest - Téléchargement
Post-exploitation in Windows environments often implies secrets collection. The collected secrets can be reused for lateral or vertical movement, making them high value assets. Most people already know the LSASS process, but other secrets such as LSA secrets and DPAPI ones could also allow privilege escalation or access to sensitive resources. This article will describe the different types of secrets that can be found within a Windows machine, and public tools that can be used to retrieve them.

After compromising a Windows host and having obtained local administrator privileges, secrets extraction is usually the first step performed to elevate privileges in the context of an Active Directory domain or to perform lateral movements inside an internal network.

As such, the following type of secrets can be retrieved:

- Secrets in LSASS process.
- Secrets in registry such as LSA secrets.
- DPAPI secrets.

This article will describe each of them, the information they contain, available tools to recover them and the existing detection risks. While no new technique will be introduced here, this summary should still help in future internal network or red team assessments. Finally, even if this listing tries to be exhaustive, some techniques or tools might be missing.

LSASS

Recovering LSASS memory is probably the most known technique to retrieve sensitive secrets. Indeed, this process is responsible for handling authentication on Windows and can contain the following elements:

- User / Machine hashes.
- Cleartext credentials (if *wdigest* is enabled).
- Kerberos tickets (TGT and ST).
- DPAPI cached keys.

As such, successfully recovering these elements is often the best way to elevate privileges from a first compromise to `Domain Administrator` in an Active Directory environment. As this process is highly sensitive, attackers and defenders are battling, antivirus are getting better and new techniques are regularly discovered to bypass them.

Moreover, Microsoft has introduced different protections over the years to protect against these attacks such as *RunAsPPL* and *Credential Guards*.

As this article is focused on the offensive side, let's talk about the different techniques one can use to retrieve LSASS secrets.

Userland techniques

While many tools have been published over the years, two of them, that are actively maintained, are enough to cover most credential recovery methods: *lsassy* and *nanodump*.

Lsassy

Lsassy has been released in 2019 and is still maintained by its creator *hackndo*. This Python tool mainly focuses on performing dumping and parsing operations remotely.

Dumping is implemented by interfacing with various external tools:

- *comsvcs*
- *comsvcs_stealth*
- *dllinject*
- *procdump*
- *procdump_embedded*
- *dumpert*
- *dumpertdll*
- *ppldump*
- *ppldump_embedded*
- *mirrordump*
- *mirrordump_embedded*
- *wer*
- *EDRSandBlast*
- *nanodump*
- *rdrleakdiag*
- *silentprocessexit*
- *sqldumper*

Memory parsing relies on the *pypykatz* project.

However nowadays, using *lsassy* is not advised if discretion is needed. Indeed, code execution is performed using *impacket* implementations such as scheduled tasks, `wmi` or `smb`, which are often detected by antiviruses.

Nanodump

Nanodump was released in 2021 and is developed in C by *S4ntiagoP*. The compiled binary can be executed using *lsassy* or the traditional `RDP`, `WinRM`, etc. It supports the following dumping methods:

- Duplicate a high privileged existing LSASS handle.

- Duplicate a low privileged existing LSASS handle and then elevate it.
- Leak an LSASS handle into nanodump via seclogon.
- Leak an LSASS handle into another process via seclogon and duplicate it.
- Make seclogon open a handle to LSASS and duplicate it.
- Open a handle to LSASS using a fake calling stack.
- Force WerFault.exe to dump LSASS via SilentProcessExit.
- Force WerFault.exe to dump LSASS via Shtinkering.

The tool supports creating an invalid memory dump to prevent antiviruses from deleting it. However, it seems the implemented techniques are nowadays blocked by most detection software. Indeed, the functions of the Windows API used to perform userland memory dumps are heavily monitored by AV products, using hooks or *Event Tracing for Windows* (ETW).

Using kernel drivers

As the previously described tools are frequently blocked or detected by antivirus products, other tools have been developed to leverage known vulnerable drivers signed by Microsoft in order to gain read/write primitives in the Windows kernel. A website lists known vulnerable drivers that can be used for such tools: https://www.loldrivers.io/.

It is however important to note that loading drivers on *Windows* is usually done using a service, which might raise flags or alerts when you are in the context of a red team.

Physmem2profit

Physmem2profit is a tool developed in C# and Python released in 2020 by WithSecureLabs. It has a client / server architecture where the client is Python-based (but Python 3.7 and later is not supported), and the server is run on the compromised host and is built in C#. The client connects to the server and mounts the `WinpMem` driver, offering direct physical memory access. The server will then try to find the LSASS process using *rekall* and reconstruct a dump.

Because the dump is constructed on the attacker's machine, it does not touch the compromised host, avoiding potential antivirus detection.

Even if this tool works properly, there are some drawbacks:

- The *rekall* framework is deprecated, making the discovery of the LSASS process not reliable on recent Windows versions and builds.
- Dumping an LSASS process is usually pretty long (~10 minutes) because of the architecture of the tool.
- Reconstructed dumps are also not reliable, and it may be necessary to restart the client in order to retrieve credentials of a connected user.

EDRSandblast

EDRSandblast is a tool developed in C and released in 2021 by Wavestone. It compiles as an executable that has to be run on the victim's machine using `RDP`, `WinRM` or other methods. It currently supports two drivers: `RTCore64` and `DBUtils_2_3` but can be extended with other vulnerable drivers.

This tool uses the driver's read and write primitives to temporarily disable all antivirus hooks and/or detection mechanisms before creating a memory dump of the LSASS process. It is very reliable even if some dumps are not usable and antivirus might still catch the dump being written to the disk. Writing the dump on a network share might help preventing such detection. The tool's GitHub page details how these actions are performed if you are interested in the technical implementation.

The tool requires offsets in `ntoskrnl.exe` to properly work, either by providing a CSV file or using the `--internet` switch. This option is not advised when stealth is required. Therefore, it will be necessary to update the offsets before the tool can work.

Moreover, *EDRSandblast* offers the possibility to enable the `wdigest` feature on a live system without requiring a reboot to retrieve credentials of new interactive connections. This feature also requires offsets for `wdigest.dll`.

DumpIt

DumpIt is a closed source tool developed by Comae and mainly used for forensic analysis, that uses a driver to dump all the physical memory into a file. Because the dumps size depends on the amount of memory available, retrieving it from a remote system is usually slow. To ease this process, MemProcFS allows mounting live memory dumps. The `minidump` folder of the LSASS process then contains a dump file that is compatible with *mimikatz* and *pypykatz*.

Moreover, such dumps can be used by the *volatility* framework to retrieve various information such as registry secrets. However, at the time of writing, there exists no plugin on the latest version of the framework to reconstruct a *minidump* or retrieve information from the LSASS process.

Registry secrets

Windows registry hives also contain different secrets and retrieving them can often yield privileged accounts within an Active Directory environment. The following secrets are stored in this base:

- SAM secrets.
- LSA secrets (`Local Security Authority`).
- Software secrets.

SAM

The SAM base (`Security Account Manager`) holds secrets related to local accounts. Querying information on this hive requires `SYSTEM` authority or to have the `SeBackupPrivilege` to make a backup of the hive.

Moreover, such sensitive information is stored encrypted with the *bootkey*. The latter is unique to each computer and is computed from information inside the `HKLM\SYSTEM\CurrentControlSet\Control\Lsa` registry key.

LSA secrets

LSA secrets are stored in the `SECURITY\Policy\Secrets` key and are encrypted using the `LsaKey` which itself is derived from the *bootkey*. Here as well, `SYSTEM` privileges are required to query the necessary keys. The following secrets can be found, among others:

- `DPAPI_SYSTEM` which contains the DPAPI machine and user key for local DPAPI.
- `NL$KM` which contains an encryption key for the `MsCache` also stored in the SECURITY hive (`SECURITY\Cache`). `MsCache` are the latest hashed credentials for domain users.
- `$MACHINE.ACC` contains the computer account credentials when joined to an Active Directory domain.
- `DEFAULTPASSWORD` is the default password when the auto-logon feature is configured.
- Keys starting with `_SC_` corresponds to non-interactive service account credentials that could be local or for a domain user.

Tooling

To recover the hives of an online system, two native utilities can be used: `reg` and `vssadmin` (shadow copy):

```
> reg save HKLM\SAM sam
> vssadmin create shadow /for=C:
> copy \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\System32\config\SAM
SAM
```

Their use is however often detected and blocked by commonly used security products.

Secretsdump.py can then be used to extract the secrets stored within the hives:

```
$ secretsdump.py -sam sam -security security -system system local
```

This tool can also be used remotely to perform all operations automatically on a live system, by performing the following actions:

- Starting the `SvcRegistry` service, if not already started.
- Using the previous service to perform a `reg save`.
- Retrieving the extracted hives over SMB.
- Parsing the files locally.

As for most Impacket tools, its use is often detected and blocked. For this reason, we developed *SharpSecretsdump*, a C# version to perform these actions on a live system. It has the following advantages:

- Runs natively on Windows.
- Does not have any dependence.
- Can be used easily from PowerShell with reflection.

DPAPI

DPAPI is an acronym for `Data Protection API` and has been integrated to Windows 2000. This mechanism, allowing data protection by encryption, is widely used by Microsoft and other software such as Chrome, Edge or SCCM. When a software offers a "remember me" or "save credentials" feature, DPAPI is probably used to encrypt credentials before storing them.

The API is very simple and makes its usage straightforward and not error-prone:

```
DPAPI_IMP BOOL CryptProtectData(
  [in]           DATA_BLOB                 *pDataIn,
  [in, optional] LPCWSTR                   szDataDescr,
  [in, optional] DATA_BLOB                 *pOptionalEntropy,
  [in]           PVOID                     pvReserved,
  [in, optional] CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
  [in]           DWORD                     dwFlags,
  [out]          DATA_BLOB                 *pDataOut
);
```

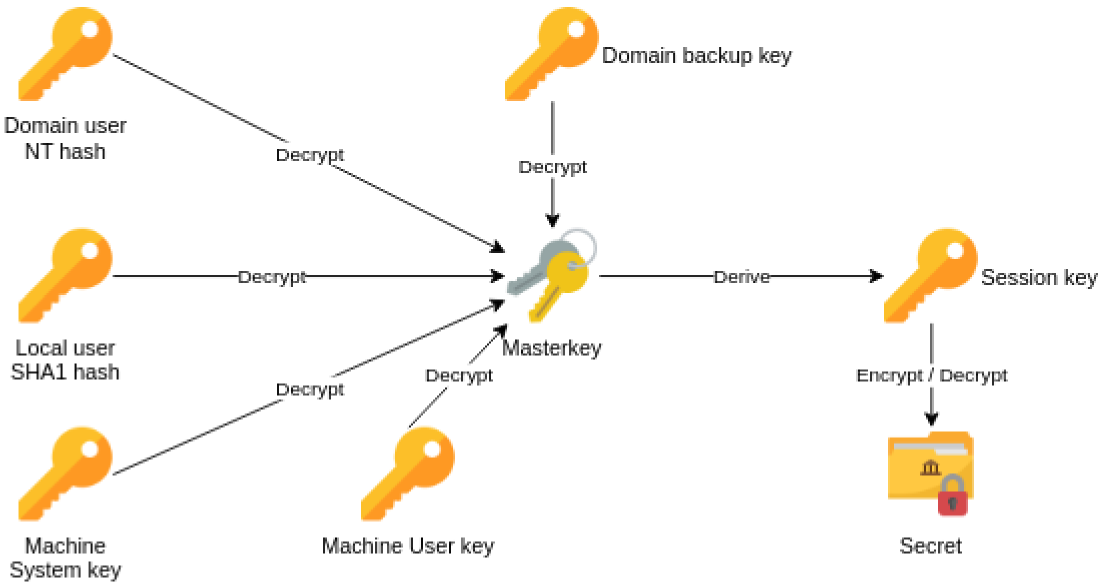
Each parameter is self-explanatory and the `pOtionalEntropy` is almost always `null`. The interesting parameter is `dwFlags`:

> `[in] dwFlags`
>
> This parameter can be one of the following flags.
>
> | Value | Meaning |
> | --- | --- |
> | **CRYPTPROTECT_LOCAL_MACHINE** | When this flag is set, it associates the data encrypted with the current computer instead of with an individual user. Any user on the computer on which **CryptProtectData** is called can use CryptUnprotectData to decrypt the data. |
> | **CRYPTPROTECT_UI_FORBIDDEN** | This flag is used for remote situations where presenting a user interface (UI) is not an option. When this flag is set and a UI is specified for either the protect or unprotect operation, the operation fails and GetLastError returns the ERROR_PASSWORD_RESTRICTION code. |
> | **CRYPTPROTECT_AUDIT** | This flag generates an audit on protect and unprotect operations. Audit log entries are recorded only if szDataDescr is not **NULL** and not empty. |

Except `CRYPTPROTECT_LOCAL_MACHINE`, the other flags will make the data encrypted using user credentials rather than the system DPAPI keys. A .NET version of this API is also available.

Decryption is then performed using the `CryptUnprotectData` function and is very similar.

Let's talk briefly about the implementation of DPAPI. Its encryption mechanism relies on *masterkeys* (MK), used to encrypt the data. Masterkeys are encrypted with a derivative from the user's password or the DPAPI system key. The following diagram is a high level view of DPAPI's inner working:



User masterkeys are located in the `C:\Users\<user>\AppData\Roaming\Microsoft\Protect\<SID>` folder, and machine masterkeys in `C:\Windows\System32\Microsoft\Protect\S-1-5-18\User` and `C:\Windows\System32\Microsoft\Protect\S-1-5-18`. The first is machine's user masterkeys used for `System`, `LocalService` or `NetworkService` accounts when encrypting in the context of a user while the latter is the machine's system masterkeys used when using `CryptProtectData` with the `CRYPTPROTECT_LOCAL_MACHINE` flag. Important facts about masterkeys:

- A masterkey's filename is a lowercase GUID, e.g. `4f31fb52-7d53-45be-b531-9f82a0682a43.`
- A masterkey file can contain two different masterkeys, when joined to a domain, one is encrypted with derivated user password while the other is encrypted with the DPAPI domain backup key. The domain backup key can be recovered with Domain Admin privileges, this key is never changed.
- They are rotated every 3 months or when the user's password is changed.
- They are cached decrypted in LSASS.
- They are considered "System files" and not displayed by default with commands or inside an explorer. For them to be shown in the explorer, the "Hide protected operating system files" option must be unchecked. In PowerShell, the `-Hidden` option can be specified when using `Get-ChildItem`.

Other files related to masterkeys exist. The `Preferred` file inside the same folder contains the GUID of the currently used masterkey whereas the `CREDHIST` file, one folder above, contains information to decrypt older masterkeys.

The masterkey decryption will have different prerequisites depending on the context, as shown in the previous diagram:

- Domain user: its password or NT hash, or the domain backup key.
- Local user: its password or SHA1 hash.
- Local DPAPI: both `system`and `security` hives to compute the key.

Different types of secrets are encrypted using DPAPI:

- Credentials
- Vault
- DPAPI blob
- RSA / NGC

Credentials

Credentials is a type of secrets that uses DPAPI and is handled by Windows. An interface is available in the `Control Panel` under `Credential Manager` where users can add credentials manually. Windows will also store user-specific credentials there, such as `One Drive` and `Skype` tokens, or RDP credentials. A credential named `didlogical` is also present on most systems and stores information about *Windows Live* applications.

Machine credentials also exist. For example, when creating a scheduled task with non-interactive logon, the user credentials are encrypted in such format. They are usually interesting in the context of a penetration test because privileged account credentials can be discovered.

Each credential is saved as a file, in `C:/Users/User/AppData/Local/Microsoft/Credentials/` for users and in `C:/Windows/System32/config/systemprofile/AppData/Local/Microsoft/Credentials/` for the system. As for masterkeys, they are considered "system files". They all have a 32 hexadecimal characters filename.

Vault

The vault is also accessible through the `Credential Manager`, but its use is rare. Indeed, it is mainly used when saving credentials in Internet Explorer and for Edge versions before December 2020 (Edge has switched to a Chromium base now).

Each credential is stored in a file that matches the following path: `AppData/*/Microsoft/Vault/[0-9a-fA-F-]+/[0-9a-fA-F]{40}/*.vcrd`.

Other files are related to the vault such as `vpol` and `vsch`. The `vpol` file associated to a vault is required to recover credentials as it contains the vault schema.

As for masterkeys, they are considered "system files".

DPAPI blob

A DPAPI blob is the output of the `CryptProtectData` function. Its structure is not officially documented but has been reversed:

```
typedef struct _KULL_M_DPAPI_BLOB {
	DWORD	dwVersion;
	GUID	guidProvider;
	DWORD	dwMasterKeyVersion;
	GUID	guidMasterKey;
	DWORD	dwFlags;

	DWORD	dwDescriptionLen;
	PWSTR	szDescription;

	ALG_ID	algCrypt;
	DWORD	dwAlgCryptLen;

	DWORD	dwSaltLen;
	PBYTE	pbSalt;

	DWORD	dwHmacKeyLen;
	PBYTE	pbHmackKey;

	ALG_ID	algHash;
	DWORD	dwAlgHashLen;

	DWORD	dwHmac2KeyLen;
	PBYTE	pbHmack2Key;

	DWORD	dwDataLen;
	PBYTE	pbData;

	DWORD	dwSignLen;
	PBYTE	pbSign;
} KULL_M_DPAPI_BLOB, *PKULL_M_DPAPI_BLOB;
```

As of today, `dwVersion` is always 1 and, `guidProvider` is `DF9D8CD0-1501-11D1-8C7A-00C04FC297EB` and `dwMasterKeyVersion` is also 1. DPAPI blobs are stored differently for each software implementation but are usually stored as files or base64-encoded strings in a database. Therefore, knowing the previous information, it is easy to fingerprint a blob when encountering binary data:

```
$ hd blob1
00000000  01 00 00 00 d0 8c 9d df  01 15 d1 11 8c 7a 00 c0  |.............z..|
00000010  4f c2 97 eb 01 00 00 00  b0 4f 23 6c 66 19 43 4b  |O........O#lf.CK|
00000020  a3 17 23 4c 19 10 73 61  00 00 00 00 02 00 00 00  |..#L..sa........|
[...]
```

For example, Wi-Fi credentials are stored as DPAPI blobs inside the following files: `C:\ProgramData\Microsoft\Wlansvc\Profiles\Interfaces\*.xml`. They are encrypted with machine's user masterkeys.

Chrome also uses this feature. The JSON file `C:\Users\<user>\AppData\Local\Google\Chrome\User Data\Local State` contains a key named `encrypted_key` with a DPAPI blob prefixed with the string *DPAPI*. This secret is used to encrypt and decrypt all saved passwords and cookie values.

Other examples include:

- `rdg` file generated from RDP connection files with credentials.
- Azure CLI credentials.
- SCCM NAA.

RSA / NGC

NGC is an acronym for Next Generation Cryptography which is basically DPAPI with different crypto providers such as "Microsoft Software Key Storage Provider" which uses RSA. Its usage is also rare, and we never needed to decrypt any information that was encrypted this way.

If you are interested in DPAPI, a detailed article goes into more details regarding the cryptographic implementation.

Tooling

There are not many native tools to perform DPAPI operations. The `cmdkey /list` command can be used to list credentials when executed in the context of a user. Moreover, Wi-Fi passwords can also be recovered with `netsh`:

```
> netsh wlan show profile

> netsh wlan show profile SSID key=clear
```

To decrypt DPAPI blobs, PowerShell can be used directly on the targeted system. The following example uses the `LocalMachine` context:

```
PS C:\Users\user> $a = [System.Convert]::FromBase64String("AQAAANCMnd[...]")
PS C:\Users\user> $b = [System.Security.Cryptography.ProtectedData]::Unprotect($a,
$null, [System.Security.Cryptography.DataProtectionScope]::LocalMachine)
PS C:\Users\user> [System.Text.Encoding]::ASCII.GetString($b)
Hello World
```

On the other hand, many custom tools exist with some of them usable on a live system:

Others can be used offline after recovering the necessary files:

Moreover, recent tools have been released to automate everything:

- DonPAPI
- dploot

Because these tools do not necessarily cover every case, it is important to know how to perform decryption manually. The `dpapi.py` script from Impacket is perfect for this.

When decrypting a DPAPI secret, there are always 3 steps to be performed:

1. Identify the required masterkey.

2. Decrypt it.
3. Decrypt the secret.

Dpapi.py allows decrypting the majority of DPAPI secrets such as: `credential`s, `vaults` and DPAPI blobs (`unprotect`). The following example shows the offline decryption of a DPAPI blob retrieved with a call to `CryptProtectData`, in the context of a user:

```
$ dpapi.py unprotect -file blob
[BLOB]
Version          :        1 (1)
Guid Credential  : DF9D8CD0-1501-11D1-8C7A-00C04FC297EB
MasterKeyVersion :        1 (1)
Guid MasterKey   : 1F2A5BF7-4F38-4F61-B1AE-7B5731032A90
Flags            :        0 ()
Description      :
CryptAlgo        : 00006610 (26128) (CALG_AES_256)
Salt             :
b'fdefe12ca5255e4ee432d38211ae02e7bd0618313fc2f27680ea64579fbcb93e'
HMacKey          : b''
HashAlgo         : 0000800e (32782) (CALG_SHA_512)
HMac             :
b'101e04a3c5215d75f6c39c4f2df80f65d6ec4bdfe8b1f57470c5aa4f5fafb678'
Data             : b'61ce6366feed014a0465a965e2efc78a'
Sign             :
b'44a66903b3c08b748dd17f7105147b9144343e88f1d04ef63ae1ec3e4b83482fd50313dcec533fa5
75f93ffc16af9fda31a1bf72484602fee6667450625f2f0a'

Cannot decrypt (specify -key or -sid whenever applicable)
```

The `1F2A5BF7-4F38-4F61-B1AE-7B5731032A90` masterkey is required, let's decrypt it:

```
$ dpapi.py masterkey -file 1f2a5bf7-4f38-4f61-b1ae-7b5731032a90 -password password
-sid S-1-5-21-3258530061-3600341354-3570646120-1000
[MASTERKEYFILE]
Version     :        2 (2)
Guid        : 1f2a5bf7-4f38-4f61-b1ae-7b5731032a90
Flags       :        5 (5)
Policy      :       15 (21)
MasterKeyLen: 000000b0 (176)
BackupKeyLen: 00000090 (144)
CredHistLen : 00000014 (20)
DomainKeyLen: 00000000 (0)

Decrypted key with User Key (SHA1)
Decrypted key:
0x6de4c0044f613034cf73a764d89311e33ca29955c9f32bac77992d0417edeb5e19da8d6fea7a969b
e92521afc0ba1c77701fd7573498e0b4a26d8242c9e2ba36
```

The SID of the user can be recovered in the path containing masterkeys: `C:\users\user\appdata\roaming\microsoft\protect\`**`S-1-5-21-3258530061-3600341354-3570646120-1000`**`\`. The masterkey can also be decrypted differently depending on the context: SHA1 for local users or NT hash for domain users. The `-pvk` option can be used with the domain backup key, while the `-key` argument should be used for system DPAPI in conjunction with the retrieved keys (from `secretsdump.py` for example) or by specifying the `-system` and `-security` switches.

And now the final step:

```
$ dpapi.py unprotect -file blob -key
0x6de4c0044f613034cf73a764d89311e33ca29955c9f32bac77992d0417edeb5e19da8d6fea7a969b
e92521afc0ba1c77701fd7573498e0b4a26d8242c9e2ba36
Successfully decrypted data
0000  48 65 6C 6C 6F 20 57 6F  72 6C 64  Hello World
```

For automated offline approaches and more complex use cases not covered by `dpapi.py`, *diana* and *dpapilab-ng* also contain many helpful scripts.

Side notes

The update of masterkeys is performed when a user is changing its password through the Windows GUI, therefore, some actions changing the password of a user might desynchronize masterkeys and make them undecryptable with the latest password (e.g. RPC calls, `net user`, etc.).

Moreover, masterkeys can also be used to perform offline brute force of a user's password using *John The Ripper*:

```
$ DPAPImk2john.py -S S-1-5-21-3258530061-3600341354-3570646120-1000 -c local -mk
1f2a5bf7-4f38-4f61-b1ae-7b5731032a90
$DPAPImk$2*1*S-1-5-21-3258530061-3600341354-3570646120-
1000*aes256*sha512*8000*46717b1fd50eb3d12559ed19016bd2dc*288*a897faef12ebdc3bf0721
a938d3de72f348f51b6e0236782f664f0d3a21b1c0bf49da1f3671014f636aad84a5d5aa9946a5a240
028e3d5e35cbcca03902f035e2d21ccbbbcbdb65a8316595d3da821cc0858a48e5f34275b742be3085
9c57b0742984f7692f3eced795c39a58d2fc6a9e7b88031aad8020f11695981edc6c06e69863c775ee
325523d74ebcbb9a3e052
```

Conclusion

Windows secrets extraction is an essential step, especially in red team assessments where trophies may be outside an Active Directory environment. As such, it is important to understand where to find sensitive information and how to decrypt it. This article only talked about specific Windows secrets, but other software (Firefox, mRemoteNG, etc.) may also store their secrets differently, and should not be omitted.