Find Windows Secrets (Pseudo-Code)

Shadow Copy

```
- Some files/directories backuped
- ACLs are duplicated from original files/directories

- Access shadow copies (PS)
   - $ShadowCopies = Get-WMIObject -Class Win32_ShadowCopy -Computer "localhost"
   - If ($ShadowCopies)
      - ForEach ($ShadowCopy in $ShadowCopies)
         - $ShadowCopyPath = $ShadowCopy.GetPropertyValue("DeviceObject")
         - [System.IO.File]::Copy("$ShadowCopyPath\<SubPath>", <SavePath>)

- Create shadow copy (PS)
   - $NewShadowCopy = (Get-WMIObject -List Win32_ShadowCopy -ComputerName "localhost").
Create("<Path>", "ClientAccessible")
   - NewShadowCopyID = $NewShadowCopy.GetPropertyValue("ShadowID")
   - ShadowCopies = Get-WMIObject -Class Win32_ShadowCopy -Computer "localhost"
   - ForEach ($ShadowCopy in $ShadowCopies)
      - If ($ShadowCopy.GetPropertyValue("ID") -eq $NewShadowCopyID)
      - $ShadowCopyPath = $ShadowCopy.GetPropertyValue("DeviceObject")
      - [System.IO.File]::Copy("$ShadowCopyPath\<SubPath>", <SavePath>)
      - $ShadowCopy.Delete()
      - Break
```

Session Tokens

```
 - Integrity levels
   - Low: Sandbox processes like web browsers
   - Medium: Applications executing in the context of a regular user
   - High: Administrators can execute applications at high integrity
   - System: Only used for SYSTEM services
   A process of a certain integrity level cannot modify a process of higher integrity level but opposite is
possible

- Two types of access tokens describe into TOKEN_STATISTICS.TokenType
   - TokenPrimary : Describe security context of a process
   - TokenImpersonation : Allow a thread to use a different security context with four levels
      - Anonymous : Unused
      - Identify : Allow to identify as the user, but cannot perform action as the user
      - Impersonate : Allow to perform local action in the context of the user
      - Delegate : Allow to perform local and remote action on domain in the context of the user
      - Impersonation level ImpersonationLevel depend on LogonType
         - https://learn.microsoft.com/en-us/windows-server/identity/securing-privileged-access/
reference-tools-logon-types
         - Network Logon (LogonType 3) -> No credentials into lsass.exe -> Cannot act on the domain

- Administrator receive two access tokens when authenticating
   - First medium integrity used by default to run processes (Primary)
   - Second high integrity is used when "Run as administrator" is choose for an application (
Impersonation) -> Trigger UAC

- Privileges are included into tokens (SeXXXXPrivilege)
   - Contain two bitmasks
      - First sets the privileges that are presents and cannot be modified
      - Second registers if the present privileges are enabled or disabled and can be modified with
AdjustTokenPrivileges()
   - whoami /priv display all present privileges (enabled or not)
   - We cannot modify present privileges but we can add some (will take effect after user account logout/
login) with LsaAddAccountRights()

- Every access token is linked to a sole logon session (into lsass.exe)
   - which is identifiable via the TOKEN_STATISTICS.AuthenticationID.LowId = LogonID parameter = 32 bit
   - TOKEN_STATISTICS.AuthenticationID = 64 bit LUID
   - TokenSessionId = Identify the attached user session ID

- Session tokens are checked to allow/disallow process/thread action
- Logon sessions are lost after reboot thus session tokens too

- Recover session tokens with LA rights
   - Enable SeDebugPrivilege to manipulate processes
   - Enumerate all processes with EnumProcesses() to get all processes IDs
   - For each processes query process tokens TokenPrimary: OpenProcess()/OpenProcessToken()
   - May also query thread tokens TokenImpersonation: OpneThread()/OpenThreadToken()
   - Get tokens information: GetTokenInformation() with different TOKEN_INFORMATION_CLASS
      - LogonID, TokenSessionId, SID, Account Name, Domain Name, TokenType, LogonType,
ImpersonationLevel (for TokenImpersonation)
   - If this is the token to impersonate: DuplicateTokenEx() from a TokenPrimary or TokenImpersonation to
a TokenPrimary or TokenImpersonation
      - CreateProcessWithLogon()
         - Require no privileges
         - Take Domain / Username / Password as input
         - Allow to spawn a graphical process while impersonating the token identity
      - ImpersonateLoggedOnUser()
         - Require the SeImpersonatePrivilege
         - Take TokenPrimary or TokenImpersonation as input
         - Allow the calling thread to impersonate the token
            - If the thread start new process/threads It will use calling process token identity
      - CreateProcessWithToken()
         - Require High integrity level and SeImpersonatePrivilege
         - Take TokenPrimary as input
         - Spawn a new graphical process while impersonating the token identity
         - If impersonated token have different session ID than calling process graphical session will be
unusable
            - Except for NT\SYSTEM user impersonated because It have full permission on Windows Station
and Desktop objects
            - Give Everyone group full permissions on the calling process Windows Station and Desktop
objects
               - Require SeSecurityPrivilege to edit ACLs
            - OR set the token session ID to match the calling process with SetTokenInformation
               - Require System integrity level and SeTcbPrivilege
      CreateProcessAsUser()
         - Require System integrity level and SeAssignPrimaryTokenPrivilege
            - Optional SeIncreaseQuotaPrivilege will be transperently enabled during the API call
         - Take TokenPrimary as input
         - Spawn a new terminal process while impersonating the token identity
```

LSASS

```
- Logon session itself is stored into lsass.exe process and can contain
   - MSV Secrets
      - LM/NT Hash
   - Credman Secrets
      - ClearText pwds
   - Wdigest Secrets
      - ClearText pwds
   - Kerberos Secrets
      - TGTs/TGS
   - DPAPI Secrets
      - DPAPI MasterKeys decrypted
   - Cloudap Secrets
   - LiveSSP Secrets
   - SSP Secrets
   - TSPKG Secrets
   - LSA Keys for decrypting previous secrets
      - IV
      - DES Key
      - AES Key

- Process
   - Enable SeDebugPrivilege to dump process memory
      - AdjustTokenPrivileges()/OpenProcessToken()/LookupPrivilegeValue()
   - Get a handle to lsass.exe
      - ProcOpen method
         - Get a direct handle to lsass.exe with OpenProcess() WinAPI
      - DupHandle method
         - List system handles with NtQuerySystemInformation() WinAPI
         - Iterate over each system handles and duplicate them with DuplicateHandle() WinAPI
         - If the handle point to a process called lsass.exe we can store It (check with GetProcessImageFileName() WinAPI)
      - Other methods https://github.com/Hackndo/lsassy#dumping-methods
   - Get memory pages from handle
      - Get loaded modules from lsass.exe
         - Store timestamp of MSV1 module: Will be used to differentiate MSV template version depending on OS
         - EnumProcessModules()/GetModuleFileNameEx()/GetModuleInformation()
      - Get memory pages info from handle on lsass.exe
         - VirtualQueryEx()
      - For all pages, for all modules, found which page is into region addr of module and add the page to module pages
   - Then, with templates (depending on OS version and architecture) for each secrets type, find secret signature in memory and iterate over the
linked list to get secrets (starting with LSA Keys for decrypting other secrets)
      - ReadProcessMemory() WinAPI
   - Each secret found in memory are decrypted with LSA Keys found in memory too
      - If (Size % 8)
         - ClearText = AESDecrypt (Key = LSADecryptor["AES_Key"], Data = EncPwd, IV = LSADecryptor["IV"], Mode = "CFB")
      - Else
         - ClearText = TripleDESDecrypt (Key = LSADecryptor["DES_Key"], Data = EncPwd, IV = LSADecryptor["IV"][0..7], Mode = "CBC")

- Logon types and credentials in lsass.exe memory : https://docs.microsoft.com/en-us/windows-server/identity/securing-privileged-access/
reference-tools-logon-types
- Secrets in lsass.exe are lost after reboot
```

DPAPI

```
- DPAPI Secrets (or DPAPI Blob) are encrypted/decrypted with MasterKeys and CryptProtectData()/
CryptUnprotectData() from Windows API
- MasterKeys are encrypted with PreKeys
- MasterKeys are stored encrypted into MasterKey Files
   - Users MasterKey' Files
      C:\Users\<USER>\AppData\Roaming\Microsoft\Protect\<UserSID>\<MKGUID>
   - System MasterKey' Files
      C:\Windows\System32\Microsoft\Protect\User\<MKGUID> (System User Master Key File)
      C:\Windows\System32\Microsoft\Protect\<MKGUID> (System Machine Master Key File)
   - Each DPAPI Blob store <MKGUID> to know which Master Key file use for DPAPI decryption
- Two types of PreKeys
   - Users' PreKeys
   - System PreKeys (System User PreKey and System Machine PreKey) from DPAPI_SYSTEM of LSA Storage (
encrypted with LSA Secret Key)
- Users' PreKeys can be computed from their (password + SID) or (NT hash + SID)
   - Key1 = HMAC-SHA1 (SHA1 (Pwd), SID + "\x00") (For local users)
   - Key2 = HMAC-SHA1 (SHA1 (NTHash), SID + "\x00") (For domain users)
   - Key3 = HMAC-SHA1 (PKBKDF2-HMAC-SHA256 (PKBKDF2-HMAC-SHA256 (NTHash, SID), SID), SID + "\x00") (
For users of "Protected users" group)

- Each MasterKey File contain 5 entries
   - Headers and system information
   - MasterKey encrypted with Users' PreKeys or System PreKeys (depending on DPAPI encryption/decryption
context)
   - DomainBackupMasterKey encrypted with RSA public key of DC (RSA keys pair generated and send to DC
when generating Master Key)
   - LocalBackupEncryptionKey encrypted with System Machine PreKey from DPAPI_SYSTEM of LSA Storage
   - CREDHIST GUID
      - In Windows 2000, It stored the LocalBackupMasterKey encrypted, which could be decrypted by any
administrator with the LocalBackupEncryptionKey and allowed to retrieve every Users' MasterKeys
      - After Windows 2000, It point to a CREDHIST File which contain Old User's PreKeys chain encrypted with
user current's password
- For each MasterKey File : Master Key/Domain Backup Master Key/Local Backup Master Key (Windows 2000)
point to the same Master Key value once decrypted
- DPAPI encryption/decryption context
   - "CRYPTPROTECT_UI_FORBIDDEN" = 0x1 = Used when user interface is not available. For example, when using
remote access.
   - "CRYPTPROTECT_LOCAL_MACHINE" = 0x4 = Data is protected using local computer account. Any
administrator user of the system may be able to decrypt it.
   - "CRYPTPROTECT_CRED_SYNC" = 0x8 = Forces synchronizing user's credentals. Normally runs automatically
upon user password change.
   - "CRYPTPROTECT_AUDIT" = 0x10 = Enables audit during encryption/dectyption
   - "CRYPTPROTECT_VERIFY_PROTECTION" = 0x40 = The flag checks security level of DPAPI blob. If the default
security level is higher than current security level of the blob, the function returns error CRYPT_I_NEW_
PROTECTION_REQUIRED as advice to reset securiry for the source data.
   - "CRYPTPROTECT_CRED_REGENERATE" = 0x80 = Regenerate local computer passwords.
   - "CRYPTPROTECT_SYSTEM" = 0x20000000 = Indicates that only system processes can encrypt/decrypt data.

- DPAPI Secrets can be:
   - Cookies/Pwds from IE, Chrome (Encrypted with User MasterKeys)
   - Wi-Fi passwords (Encrypted with System MasterKeys)
   - E-mail account passwords in Outlook, Windows Mail, etc.
   - Passwords from Remote Desktop Connection Manager
   - Internal FTP manager account passwords
   - Encryption key in Windows CardSpace and Windows Credential Vault Manager
   - Any data encrypted with CryptProtectData()

- Get-DPAPISecrets:
   - Compute PreKeys (Users' PreKeys with gathered Pwds/NTHashes/System PreKeys with DPAPI_SYSTEM from
LSA Storage/PreKeys from gathered Pwds/NTHashes into LSASS)
   - Retrieve all MasterKey Files and try to decrypt each part (Master Key/Domain Backup Master Key/Local
Backup Master Key (Windows 2000)) with PreKeys to obtain the decrypted MasterKey value
      - For System MasterKey' Files we know that we have to use System PreKeys (and we always have them)
      - For Users MasterKey' we don't know which User PreKeys to use (and we may have not them), BUT we can
validate the decryption success
   - Find DPAPI Secrets
      - Chrome cookies/pwds have known locations
      - Wi-Fi passwords have known locations

- MasterKeys can be stored and retrieved from LSASS

- Get-ChromePwds: Chrome Pwds/Cookies are encrypted with Users' MasterKeys, thus we cannot always
decrypt them (We need their Pwd/NT Hash -> User PreKeys -> User MasterKeys)
   - Found encrypted datas, many potential paths
      - C:\Users\<User>\[Local/Roaming/LocalLow]\[""/Google]\Chome\User Data\[""/Default]\[Local State/Login
Data/Cookies]
      - Local State is a file that contain a key encrypted with DPAPI
      - Login Data is a SQLite file that contain passwords (= password_value) encrypted (action_url, username_
value, password_value FROM logins)
      - Cookies is a file that contain cookies (= encrypted_value) encrypted (host_key, name, path, encrypted_
value FROM cookies)
   - Depending on Chrome versions
      - Chrome version < v80
         - Try to decrypt DPAPI Blob with Decrypt-DPAPIBlob and MasterKeys and get the secret Pwd/Cookie
      - Chrome version >= v80
         - Try to decrypt Local State Key with Decrypt-DPAPIBlob and MasterKeys
         - From password_value/encrypted_value extract Nonce, CipherText, Tag : "v10"|Nonce|CipherText|Tag
         - Secret Pwd/Cookie = AES256-GCMDecrypt (SecretKey = LocalStateKey decrypted, IV = Nonce,
AuthTag = Tag, AuthData = "", CipherText = CipherText)

- Get-WiFiPwds: With System MasterKeys we can always decrypt Wi-Fi pwds
   - Encrypted password for each Wireless interface and each SSID is located at C:\ProgramData\Microsoft\
Wlansvc\Profiles\Interfaces\<IDForWirelessInterface>\<IDForSSID>.xml
```

Credential Vault Manager

```
- Store logins credentials for applications, Web, etc.
- Process
   - Find all VPOL files
      - C:\Users\<User>\AppData\[Local,Roaming\LocalLow]\Microsoft\Vault\<SID>\Policy.vpol
      - C:\ProgramData\Microsoft\Vault\<SID>\Policy.vpol
      - C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\Vault\<SID>\Policy.vpol
   - Try to decrypt them with MasterKeys (Decrypt-DPAPIBlob)
   - Each VPOL file decrypted provide two VPOL keys
   - Find all VCRD files
      - C:\Users\<User>\AppData\[Local,Roaming\LocalLow]\Microsoft\Credentials\[A-Za-z0-9]{32}
      - C:\ProgramData\Microsoft\Credentials\[A-Za-z0-9]{32}
      - C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\Credentials\[A-Za-z0-9]{32}
   - ForEach VCRD files
      - ForEach VPOL keys
         - ClearText = AESDecrypt (Key = VPOL key, Data = $Attribute["Data"], IV = $Attribute["IV"], Mode = "CBC")
```

Hashed BootKey

```
- AQWERTY = "!@#$%^&()qwertyUIOPAzxcvbnmQQQQQQQQQQQQ)(@&%\x00"
- ANUM = "0123456789012345678901234567890123456789\x00"
- Structure DOMAIN_ACCOUNT_F = "HKLM\SAM\SAM\Domains\Account\F"
- If DOMAIN_ACCOUNT_F[Key0] == 0x01
      - Structure SAM_KEY_DATA = DOMAIN_ACCOUNT_F[Key0]
      - RC4Key = MD5 (SAM_KEY_DATA[Salt] + AQWERTY + BootKey + ANUM)
      - Hashed BootKey = RC4Encrypt (Key = RC4Key, Data = SAM_KEY_DATA[Key] + SAM_KEY_DATA[Checksum])
- ElseIf DOMAIN_ACCOUNT_F[Key0] = 0x02
      - This is Windows 2016 TP5 on in theory (it is reported that some W10 and 2012R2 might behave this way also,
according to secretsdump.py)
      - Structure SAM_KEY_DATA_AES = DOMAIN_ACCOUNT_F[Key0]
      - Hashed BootKey = AESDecrypt (Key = BootKey, Data = SAM_KEY_DATA_AES[Data][:SAM_KEY_DATA_AES[
Data]+SAM_KEY_DATA_AES[DataLen]], IV = SAM_KEY_DATA_AES[Salt], Mode = "CBC")
```

SAM

```
- Contain LM/NT hashes obfuscated of local users for PTH or cracking

1. Get LM/NT hashes encrypted for every user's RID
   - Structure USER_ACCOUNT_V = HKLM\SAM\SAM\Domains\Account\Users\<RID>\V
   - If USER_ACCOUNT_V[Data][NTHashOffset:NTHashOffset+NTHashLength] == 0x01 : Old style hashes (< Windows
10 v1607)
      - If USER_ACCOUNT_V[LMHashLength] == 20
         - We have encrypted LMHash
         - Structure SAM_HASH_LM = USER_ACCOUNT_V[Data][LMHashOffset:LMHashOffset+LMHashLength]
         - Enc_LMHash = SAM_HASH_LM[4:]
      - If USER_ACCOUNT_V[NTHashLength] == 20
         - We have encrypted NTHash
         - Structure SAM_HASH_NT = USER_ACCOUNT_V[Data][NTHashOffset:NTHashOffset+NTHashLength]
         - Enc_NTHash = SAM_HASH_NT[4:]
   - Else : New style hashes (>= Windows 10 v1607)
      - If USER_ACCOUNT_V[LMHashLength] > 24
         - We have encrypted LMHash
         - Structure SAM_HASH_AES_LM = USER_ACCOUNT_V[Data][LMHashOffset:LMHashOffset+LMHashLength]
         - Enc_LMHash = SAM_HASH_AES_LM[24:]
      - If USER_ACCOUNT_V[NTHashLength] > 24
         - We have encrypted LMHash
         - Structure SAM_HASH_AES_NT = USER_ACCOUNT_V[Data][NTHashOffset:NTHashOffset+NTHashLength]
0x20]
         - Enc_NTHash = SAM_HASH_AES_NT[24:]

- Compute DESKeys = RIDToDESKeys (HKLM\SAM\SAM\Domains\Account\Users\<RID>)
- ANTPASSWORD = "NTPASSWORD\x00"
- ALMPASSWORD = "LMPASSWORD\x00"

2. If Enc_LMHash
   - If not New style
      - RC4Key_LM = MD5 (HashedBootKey[0:0x10] + RID + ALMPASSWORD)
      - Obf_LMHash = RC4Encrypt (Key = RC4Key_LM, Data = Enc_LMHash)
   - Else
      - Obf_LMHash = AESDecrypt (Key = HashedBootKey[0:0x10], Data = Enc_LMHash, IV = SAM_HASH_AES_LM[Salt],
Mode = "CBC")[0:0x10]
   - LMHash = DESDecrypt (Key = DESKeys[0], Data = Obf_LMHash[0:8]) + DESDecrypt (DESKeys[1], IV = Obf_LMHash[
8:16], Mode = "ECB")

3. If Enc_NTHash
   - If not New style
      - RC4Key_NT = MD5 (HashedBootKey[0:0x10] + RID + ANTPASSWORD)
      - Obf_NTHash = RC4Encrypt (Key = RC4Key_NT, Data = Enc_NTHash)
   - Else
      - Obf_NTHash = AESDecrypt (Key = HashedBootKey[0:0x10], Data = Enc_NTHash, IV = SAM_HASH_AES_NT[Salt],
Mode = "CBC")[0:0x10]
   - NTHash = DESDecrypt (Key = DESKeys[0], Data = Obf_NTHash[0:8]) + DESDecrypt (DESKeys[1], IV = Obf_NTHash[
8:16], Mode = "ECB")
```

LM Hash

```
- Algo
   - The user's password (as an OEM string) is converted to uppercase.
   - This password is null-padded or truncated to 14 bytes.
   - This "fixed" password is split into two 7-byte halves.
   - These values are used to create two DES keys (one from each 7-byte half).
   - Each of these keys is used to DES-encrypt the constant ASCII string "KGS!@#$%" (resulting in two 8-byte
ciphertext values).
   - These two ciphertext values are concatenated to form a 16-byte value - the LM hash.
- Null LM hash = aad3b435b51404eeaad3b435b51404ee
- Turned off by default starting in Windows Vista/Server 2008
   - Null LM hash is set if turned off
   - Can be enabled for backward compatibility
   - Set key NoLMHash to 0 into HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa
- PTH or crack
```

NT Hash

```
- Algo
   - The user's password is converted to UTF-16LE
   - NT Hash = MD4 of previous result
- Used by default on Windows
- PTH or crack
```

BootKey (SysKey)

```
1. Permutation (Concatenation of
classes HKLM\SYSTEM\
CurrentControlSet\Control\Lsa{JD,
Skew1,GBG,Data})
   - With class information
retrieved with RegOpenKeyEx()/
RegQueryInfoKey() WinAPI
   - With permutation = [ 0x8, 0x5,
0x4, 0x2, 0xb, 0x9, 0xd, 0x3, 0x0,
0x6, 0x1, 0xc, 0xe, 0xa, 0xf, 0x7 ]
```

LSA Storage

```
- Protected storage for credentials data used by the Local Security Authority (LSA) in Windows

1. Find LSA Secret Key
   - Useful to decrypt LSA secrets
   - Enc_LSASecretKey = HKLM\Security\Policy\PolEKList (Vista style) or HKLM\Security\Policy\
PolSecretEncryptionKey
   - If >= Windows Vista
      - Structure LSA_SECRET = Enc_LSASecretKey
      - Update = BootKey
      - For i in range (1000) : Update += LSA_SECRET[EncryptedData][:32]
      - Key = SHA256 (Update)
      - Data = LSA_SECRET[EncryptedData][32:]
      - For i in range (0, len(Data), 16)
         - Block = Data[i:i+16]
         - If (len(Block) < 16) : Block += "\x00" * (16 - len(Block))
         - Plaintext += AESDecrypt (Key = Key, Data = Block, IV = "\x00" * 16, Mode = "CBC")
      - Structure LSA_SECRET_BLOB = PlainText
      - LSASecretKey = LSA_SECRET_BLOB["Secret"][52:][:32]
   - Else
      - Update = BootKey
      - for i in range (1000) : Update += Enc_LSASecretKey[60:76]
      - Key = MD5 (Update)
      - PlainText = RC4Decrypt (Key = Key, Data = Enc_LSASecretKey[12:60])
      - LSASecretKey = PlainText[0x10:0x20]

2. Each secrets CurrVal/OldVal (as Data) can be decrypted with DecryptLSASecret (LSASecretKey, Data)
   - If >= Windows Vista
      - Structure LSA_SECRET = Data
      - Update = LSASecretKey
      - For i in range (1000) : Update += LSA_SECRET[EncryptedData][:32]
      - Key = SHA256 (Update)
      - Data = LSA_SECRET[EncryptedData][32:]
      - For i in range (0, len(Data), 16)
         - Block = Data[i:i+16]
         - If (len(Block) < 16) : Block += "\x00" * (16 - len(Block))
         - Plaintext += AESDecrypt (Key = Key, Data = Block, IV = "\x00" * 16, Mode = "CBC")
      - Structure LSA_SECRET_BLOB = PlainText
      - return LSA_SECRET_BLOB["Secret"]
   - Else
      - Structure LSA_SECRET = Data
      - EncryptedSecretSize = Data[:4]
      - Value = Data[len(Data)-EncryptedSecretSize:]
      - Key0 = LSASecretKey
      - For i in range (0, len(Value), 8):
         - CipherText = Value[:8]
         - StrKey = Key0[:7]
         - Key = STRToKey(StrKey)
         - PlainText += DESDecrypt (Key = Key, Data = CipherText, IV = Key, Mode = "ECB")
         - Key0 = Key0[7:]
         - Value = Value[8:]
         - If len(Key0) < 7
            - Key0 = LSASecretKey[len(Key0):]
      - Structure LSA_SECRET_XP = PlainText
      - return LSA_SECRET_XP["Secret"]

3. Find LSA Secrets into registry HKLM\Security\Policy\Secrets that can contain
   - $MACHINE.ACC
      - https://adsecurity.org/?p=280
      - Computer machine account password for computer joined to an AD domain
      - Updated into AD by client every 30 days (after windows 2000)
      - The password is 120 characters (UTF16, or 240 bytes)
      - Use to list domain users, admins, browsing shares, etc. on DC
      - $MACHINE.ACC_Plain = Decrypt-LSASecret (LSASecretKey, HKLM:\SECURITY\Policy\Secrets\$
MACHINE.ACC\<CurrVal>/<OldVal>)
      - $MACHINE.ACC_NT = MD4 ($MACHINE.ACC_Plain)
   - DefaultPassword
      - Password used to logon to Windows if auto-logon is enabled
      - DefaultPWD = UTF-16LE (Decrypt-LSASecret (LSASecretKey, HKLM:\SECURITY\Policy\Secrets\
DefaultPassword\<CurrVal>/<OldVal>))
      - DefaultLogin = HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\
DefaultUserName
      - DefaultDomain (for domain account) = HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\
Winlogon\DefaultDomainName
   - NL$KM
      - Secret key used to deobfuscate cached domain credentials
      - NL$KM = Decrypt-LSASecret (LSASecretKey, HKLM:\SECURITY\Policy\Secrets\NL$KM\<
CurrVal>/<OldVal>)
   DPAPI_SYSTEM
      - Keys to decrypt DPAPI Storage
      - DPAPI_SYSTEM = Decrypt-LSASecret (LSASecretKey, HKLM:\SECURITY\Policy\Secrets\DPAPI_
SYSTEM\<CurrVal>/<OldVal>)
      - Version = DPAPI_SYSTEM[0:4]
      - MachineKey = DPAPI_SYSTEM[4:24]
      - UserKey = DPAPI_SYSTEM[24:44]
   - _SC_<ServiceName>
      - Service account passwords
      - Secret = UTF-16LE (Decrypt-LSASecret (LSASecretKey, HKLM:\SECURITY\Policy\Secrets\_SC_<
ServiceName>\<CurrVal>/<OldVal>))
      - Find account name that launch service with Get-WmiObject Win32_Service -Property Name,
StartName
   - ASPNET_WP_PASSWORD
      - ASP.Net WordPress password
      - Password = UTF-16LE (Decrypt-LSASecret (LSASecretKey, HKLM:\SECURITY\Policy\Secrets\
ASPNET_WP_PASSWORD\<CurrVal>/<OldVal>))
   - L$_SQSA_S-<SID>
      - Answers to security questions stored by Windows
      - JSON = json.loads ((UTF-16LE (Decrypt-LSASecret (LSASecretKey, HKLM:\SECURITY\Policy\
Secrets\L$_SQSA_S-<SID>\<CurrVal>/<OldVal>))).replace("\xA0", " "))
      - If JSON["version"] == 1
         - For Item in JSON["questions"]
            - Question = Item["question"]
            - Answer = Item["answer"]
      - Else : No parser
   - There are others secrets that can be stored
```

Cached Domain Credentials

```
- Cached on local system after successful login so that domain members can logon to the machine even if the DC
cant be reached

- Stored
   - Into HKLM\SECURITY\Cache\NL$<X>
   - Hashed with MsCash (Windows XP/2003) or MsCash2 (Windows Vista/7/2008 and up) algorithm
   - Hashes are stored obfuscated with NL$KM key from LSA Storage
   - Deobfuscate
      - IterationCount = 10240
      - If HKLM\SECURITY\Cache\NL$IterationCount exist
         - record = HKLM\SECURITY\Cache\NL$IterationCount
         - If record > 10240
            - IterationCount = record & 0xfffffc00
         - Else
            - IterationCount = record * 1024
      - For each NL$<X>
         - Structure NL_RECORD = HKLM\SECURITY\Cache\NL$<X>
         - If NL_RECORD[IV] != 16 * "\x00"
            - If ((NL_RECORD[Flags] & 1) == 1)
               - If >= Windows Vista
                  - PlainText = AESDecrypt (Key = NL$KM[16:32], Data = NL_RECORD[EncryptedData], IV = NL_RECORD[
IV] Mode = "CBC")
               - Else
                  - Key = HMAC_MD5 (Key = NL$KM, Data = NL_RECORD[IV])
                  - PlainText = RC4Encrypt (Key = Key, Data = NL_RECORD[EncryptedData])
            - Else
               - Unknown case
            - MSCashHash = PlainText[:0x10]
            - PlainText = PlainText[0x48:]
            - UserName = PlainText[:NL_RECORD[UserLength]].decode ("UTF-16LE")
            - PlainText = PlainText[pad(NL_RECORD[UserLength]) + pad(NL_RECORD[DomainNameLength]):]
            - DomainName = Plaintext[:pad(NL_RECORD[DnsDomainNameLength])].decode ("UTF-16LE")
         - Else
            - Unknown case

- Cannot be used for PTH since it's not LM/NT hash, only cracked
- Used by default and stored at max 10 credentials (default but can be changed)
- Disable : Set CachedLogonsCount to 0 into HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
```

NTDS.DIT

```
- Contain LM/NT hashes of all domain users for PTH or cracking
- Stored on DC at C:\Windows\NTDS\ntds.dit but file is locked
- Access file with
   - Shadow copy
   - DRSUAPI : Uses drsuapi RPC interface create a handle, trigger replication, and combined with additional drsuapi
calls to convert the resultant linked-lists into readable format
      - Also called "DCSync" attack
      - Require following three permissions on AD
         - Replicating Directory Changes
         - Replicating Directory Changes All
         - Replicating Directory Changes In Filtered Set

- Parse NTDS.DIT shadow copy as Microsoft Extensive Storage Engine (ESE) format
   - Extract headers from NTDS at page 1
   - Parse DB starting at page 4
   - Open page table "datatable" and position a cursor at the leaf levels for fast reading
   - Search PEKList into page table "datatable" (we may found user account record while searching, store them for
later processing)
   - Decrypt the PEKList if founded with BootKey and store PEK Keys
      - KeyMaterial = EncPEKListData[8..23]
      - EncPEKList = EncPEKListData[24..(EncPEKListData.Length-1)]
      - If (EncPEKListData[0..3] == [2,0,0,0]) # Up to Windows 2012 R2
         - Update = BootKey
         - For i in range (1000) { Update += KeyMaterial }
         - Key = MD5 (Update)
         - PEKs into RC4 (Key, EncPEKList)
      - ElseIf (EncPEKListData[0..3] == [3,0,0,0]) # Windows 2016 TP4 and up
         - PEKs into AESDecrypt (Key = BootKey, Data = EncPEKList, IV = KeyMaterial, Mode = "CBC")
   - Now we have PEK Keys, Let decrypt each user record
      - Starting from users already cached when searching Encrypted PEKList
      - Then search other users into NTDS after Encrypted PEKList and decrypt LM/NT hashes
```

Legend
- Intermediate secret
- End secret