give me the password and I'll rule the world

dpapi, what else?
who am I?

A question I often ask to myself… let socials do the job.

Francesco Picasso

Reality Net System Solutions

@dfirfpi

blog.digital-forensics.it
do you need … ?

(given an unencrypted Windows OS…)

• to get WiFi password?
• to get Internet Explorer saved credentials?
• to get Chrome saved credentials?
• to get stored Windows Credentials?
• to get Windows Mail credentials?
• to get Dropbox decrypted databases?
• to get iCloud authentication tokens?
• to decrypt EFS files?
• and much more…
… then you need to understand *data protection API* … DPAPI

- first introduced with Windows 2000
- “simply” and smart
- it receives plaintext and returns ciphertext
  - and *vice versa*, hopefully…
- does not provide any storage facility
- not officially documented (*did you expect it?*)
- `CryptProctedData` and `CryptUnprotecData`
crypt «protect»

- from an application perspective
  - «please encrypt my secret»

- Then the application must store somewhere the result
  - the encrypted_blob

- blob? Yes, something truly opaque

```c
typedef struct _CRYPTOAPI_BLOB {
    DWORD cbData;    // data length
    BYTE *pbData;    // data...
} many_aliases_here;
```

```c
BOOL WINAPI CryptProtectData(
    _In_    DATA_BLOB *pDataIn,
    _In_    LPCWSTR szDataDescr,
    _In_    DATA_BLOB *pOptionalEntropy,
    _In_    PVOID pvReserved,
    _In_opt_ CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
    _In_    DWORD dwFlags,
    _Out_   DATA_BLOB *pDataOut
);
```

CryptProtectData (  
    cleartext_blob,  
    NULL,  
    NULL,  
    NULL,  
    0,  
    encrypted_blob  
);  // for purists: pseudo-code...

@dfirfpi on 5
DFIR Summit  
Prague 2014
crypt unprotect

- extremely easy to use
- application needs to know almost nothing and should take care of storage only.

- `pOptionalEntropy` (aka `entropy`)
  - Remember it for later discussion..

```c
BOOL WINAPI CryptUnprotectData(
    _In_    DATA_BLOB *pDataIn,
    _Out_opt_    LPWSTR *pszDataDescr,
    _In_opt_    DATA_BLOB *pOptionalEntropy,
    _Reserved_    PVOID pvReserved,
    _In_opt_    CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
    _In_    DWORD dwFlags,
    _Out_    DATA_BLOB *pDataOut
);

CryptUnprotectData(
    encrypted_blob,
    NULL,
    NULL,
    NULL,
    NULL,
    0,
    cleartext_blob
); // it's pseudo-code again...
```
Digging inside DPAPI
DPAPIck

All the next information comes from an awesome reverse engineering project presented at BlackHat@2010

Reversing DPAPI and Stealing Windows Secrets Offline by Jean-Michel Picod and Elie Bursztein

with the solid python code dpapick https://bitbucket.org/jmichel/dpapick

Windows Xp/Vista only (up to slide 29, at least…)

The presentation covers the main topics, for details refer to the paper.
de-opaquing blobs

dpapi blobs contains the encrypted data and all the info and reference to decrypt them

this is the blob returned by CryptProtectData

DPAPI BLOB (partial)

| version | = 1 |
| provider | = df9d8cd0-1501-11d1-8c7a-0c04fc297eb |
| mkey | = ba76b33f-4b97-4e97-9603-865fb972cec |
| descr | = «cool dpapi blob» |
| cipherAlgo | = AES-256 [0x6610] |
| hashAlgo | = sha512 [0x800e] |
| data | = da971a5559ead33ad490b22df5073216... |
| salt | = 24c980da28f49f6f3aa4f3165ad... |
| cipher | = d45f5bfe0c72a4737092f47354c45e... |
| crc | = 710d3fadb2ac02e0631d812e93ef6f... |

- hash algorithm used to derive the decryption key, crypto algorithm used to decrypt the key
- IVs and material used for derivation and decryption
- it's not self-standing
- to decrypt the data the Master Key is needed
Master Key (file) contains the encrypted «master key» data used to decrypt blobs.

- or, better, the necessary data used to derive the blobs decryption keys.
- it’s a file totally managed by the OS
- it’s «self-standing»
- the user password protects the encrypted «master key» data

---
	his is the blob returned by CryptProtectData

**MASTER KEY** *(partial)*

MasterKeyFile ba76b33f-4b97-4e97-9603-7865fb972cec

version = 2
Flags = 0x5
MasterKey = 176
BackupKey = 144
CredHist = 20
DomainKey = 0
+ Master Key: Masterkey block
  cipher algo = AES-256 [0x6610]
  hash algo = sha512 [0x800e]
  rounds = 8000
  IV = 3a2c56da9f122e4eea2ff56fba6fef78
  ciphertext = 174556341a1e8f40c6f6d77bd692...
user pwd decrypts the masterkey

masterkey is used to derive the decryption key for the specific blob (using its info too)

the blob decryption key decrypts (finally) our data!
DPAPI locations

- `<user profile dir>\AppData\Roaming\Microsoft\Protect`
  - per user based
  - and **system**?
  - **multiple** master keys?
  - what is **CREDHIST** file?

**tight bound** to the current installation!
SYSTEM DPAPI

- **SYSTEM** has its master keys too
  - `%WINDIR%\System32\Microsoft\Protect`

- CRYPTPROTECT_LOCAL_MACHINE flag used in CryptProtectData

- For example, **WiFi** password are SYSTEM DPAPI BLOBs
  - grab them from XML files in `\ProgramData\Microsoft\WwanSvc\Profiles`

- User password? No, system password
  - **LSA Secrets** -> **SECURITY** hive -> **DPAPI_SYSTEM**

- Which **security**? No security at all… not only for WiFi
master keys and credhist

• OS renews the current master key every 3 months
• Previous **MKs** must be kept to allow “old” BLOBs decryption

• A **password change** results in a re-encryption of the **MK**
  (something to be said here)

• Old MKs? **CREDHIST** file keeps track of all user’s **passwords**
  (sha1(utf16le(password)))
DPAPI applied
what we need
Almost three… each application could do anything before and/or after CryptUnprotectData calls.
OS level

get the lever
the traditional way

• **crack** the NTLM password hash

• Can use **rainbow tables**

• **do you think that** 13,759,005,997,841,642 (~ 2^53,6) elements are enough?
  (ntlm_mixalpha-numeric#1-9)

• No guarantee, a **long** even simple password will easily survive…
Do you remember?

- **Memory** is the **target**, since it **has** our **keys** in different **shapes**

- Why the password **could be** in RAM?
  - **SSO** (Single Sign on) is the first and main reason
  - Windows is the second 😊

- Which data, Where and How to use it?
Benjamin Delpy aka gentil_kiwi started to dig inside the *lsass* process in 2007, and he first discovered the presence of users’ **cleartext** password. Well-known in the pentest world for **pass the hash** attack, his research and tool contains a wealth of info. Simply awesome.

**Site**>  http://blog.gentilkiwi.com/mimikatz

**Code**>  https://github.com/gentilkiwi/mimikatz

The presentation covers only few details.
authentication packages

SSP Security Service Providers

Target data is inside the **Lsass** process
reversible encryption for **passwords**

### mimikatz cleartext table

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
</table>
| **msv1_0** | • NTLM  
  • No password, HASH |
| **tspkg** | • Terminal Server *(password)* *(no password, password)* if Credential Delegation policy i is On, default Off |
| **wdigest** | • HTTP/SASL authentication *(password)* *(password)*, unless – as in Win8.1 – the following value is 0. HKLM\SYSTEM\CurrentControlSet\Control\SecurityProviders\WDigest\UseLogonCredential” |
| **livessp** | • Windows live *(password)* *(password)* |
| **kerberos** | • Kerberos *what else?* *(password)* *(no password, or after tgt, password)* |
mimikatz can handle lsass process dump

The most complete and up to date solution is **WinDbg** plus the **mimilib.dll** extension

the easiest and cross platform. Using the mimikatz plugin I wrote. Only **wdigest** is supported, on **Vista** and **Windows 7** (x86 and x64)

RAM dumps

hibernation files

VmWare vmem files

Processes dumps

... whatever has **lsass** process inside it...
Ouch! **Paged out!**

RAM dumps are «physical»

**Stuffs** could be **paged out**

Until tools will manage both RAM + **swap** file… we have an issue…

Searching inside the pagefile?

**On scene…**

So, if passwords and/or DAPI is a **target**

After the RAM acquisition go for **lsass** process dump

Using **procdump**, for example
windbg mimikatz walkthrough

1. get a memory dump of lsass.exe
2. git clone mimikatz
3. build mimilib with Visual Studio for your architecture using SDK
4. run windbg
5. open the crash dump configure symbols!!
6. load mimilib
7. just mimikatz it!
win8.1, live and local accounts

---

**Authentication Id**
0:144929 (00000000:000000235bc)

**Session**
Interactive from 1

**User Name**
user

**Domain**
WIN-FAOBG580E65

**SID**
S-1-5-21-1228076315-4144300480-3078395761-1001

| nwsp | [00000003] Private
|---|---
| * Username | nr@gmail.com
| * Domain | MicrosoftAccount
| * NTLM | 0752771ece46f5c668a6402f9e4e8ad66
| * SHA1 | 64350217111f9036f0d290538
| tkspkg | K0

**wdigest**

| * Username | nr@gmail.com
| * Domain | MicrosoftAccount
| * Password | (null)

**live esposa**

| * Username | nr@gmail.com
| * Domain | MicrosoftAccount
| * Password | Netflix123

**kerberos**

| * Username | nr@gmail.com
| * Domain | MicrosoftAccount
| * Password | (null)

**sssp**

| * Masterkey | 0000000000000000
| * QUID | ((62650b-93df-4d98-9473-90f36959df4d)
| * Time | 09/10/2014 08:30:27
| * Key | 094a2d54f1a1de366e1c05532fb5d2960b16d1058694
| * Credhash | 

---

**Authentication Id**
0:144929 (00000000:000000235bc)

**Session**
Interactive from 1

**User Name**
user

**Domain**
WIN-FAOBG580E65

**SID**
S-1-5-21-1228076315-4144300480-3078395761-1001

| nwsp | [00000003] Private
|---|---
| * Username | user
| * Domain | WIN-FAOBG580E65
| * NTLM | 3b13f1ba60e65eb38e83eb7026d40
| * SHA1 | 74b87b1e1234f71fe4737956c2428d145bf4
| [00010000] Credential Keys
| * NTLM | 3b13f1ba6065eb38e83eb7026d40
| * SHA1 | 74b87b1e1234f71fe4737956c2428d145bf4

**tasks**

| * Username | user
| * Domain | WIN-FAOBG580E65
| * Password | (null)

**live esposa**

| * Username | user
| * Domain | WIN-FAOBG580E65
| * Password | (null)

**kerberos**

| * Username | user
| * Domain | WIN-FAOBG580E65
| * Password | (null)

**sssp**

| * Masterkey | 
| * Credhash | 

---

@dfirfpi on 26
unlocking DPAPI… forever?

- **Prekey**… \( \text{HMAC\textunderscore SHA1}(\text{sha1(utf16le(password))), utf16le(SID+’\0’)) \)
  - See previous slide…

- The **msv** SSP has the NTLM and **SHA1** password hashes…

- Not the password… **but DPAPI can be unlocked!**

- The msv SSP cannot be removed …

- Anyway, OS **must cache something**…
DPAPI level

get the the stand point
dpapick *new life!*

• *dpapick* is **the** tool to decrypt blobs
  • “*ok, but for Windows 7, 8 and 8.1?*”

• After 4 years and right for the SANS DFIR meeting (showtime!)

• I continued the reverse engineering of DAPI

• I provided the code patch to handle blob decryption up to Win 8.1

• **warning:** only **local** account on Windows 8.1
  • *Live* accounts uses the new DPAPI-NG
Stay tuned! Few days for the beta

- **Jean-Michel Picod** (kudos!!) is working on the new dpapick release
  - He is integrating the patch into the code
  - He is adding some cool features…
  - Testing =)
  - We are working on adding more plugins to handle specific apps blob decoding

- Missing> a bunch of stuffs, as *lsa secrets* decoding.
finally the apps

scenarios
scenarios

- Just two for time reasons

- *Get cleartext data*
- *Get credentials to get data from the cloud*

- **Bonus**: it’s possible to search for blobs, since the crypto provider GUID spotted is always the same
  - “df9d8cd0-1501-11d1-8c7a-0c04fc297eb”
dropbox scenario

get cleartext data
dropbox

- heavily obfuscated clients code

- configuration (config.dbx) and file sync status (filecache.dbx)
  - but they were SQLITE database (.db) before that (2012)

- even with virtualization no static cleartext form…

- How they are encrypted? Could be DPAPI?
how dropbox encrypt dbx?

didn’t know at that time

spotted two keys in the registry (user)

... the only two that are there? 😊 ...

that’s a blob!

DID YOU JUST CALL ME...

Blob?!?!
dropbox dpapi

additional entropy D114A55212655F74BD772E37E64AEE9B

where did it come from?

it’s fixed … ?!?!?

Nicolas Ruff
Florian Ledoux

“A critical analysis of Dropbox software security” (!! 2011 - 2012 !!)
dropbox dbx decryption

they almost did everything to decrypt dbx!! check the code here

https://github.com/newsoft
sqlite encryption extension

add-on to sqlite that allows read/write ops on encrypted database
RC4 - AES128 (OFB, CCM) – AES256 (OFB)

quite complicated, made by hwaci (http://www.hwaci.com/sw/sqlite/see.html)

licensed software (source code, perpetual license, $2000)

but Ledoux and Ruff found it online…

“that part of the source code is available from SQLite Web site”

check here
https://github.com/newsoft/sqlite3-dbx
iCloud scenario
access inaccessible data
devices

- **iPhone**, no access code or even broken, no way to get the data
- **Laptop** Windows with **iCloud** installed

iCloud keeps your mail, documents, contacts and calendars up-to-date between your iPhone, iPad, iPod Touch, Mac and Windows PC.
• Once installed, you’ll be prompted once for credentials
  • So, where does it keep the secret?
  • File com.apple.AOSKit.plist
  • path: <user profile>\AppData\Roaming\ Apple Computer\Preferences
  • Binary plist, decode64 the data section
  • A huge dpapi blob.
iCloud windows DPAPI

- Crypt(Un)protectData is called with the **entropy** parameter set.
- Which is a **fixed** 128 byte long hexadecimal string.

```
\x1D\xAC\xA9\xF9\xD9\xB9\x46\x3E\x48\x7D\x3E\x0A\x62\x07\xDD\x26'
\x06\x67\x81\x03\x97\x82\x13\xA5\x80\x79\x4F\x0F\x43\x15\xDD
\x7B\x14\x6C\xE5\x4B\x46\x0D\xC1\xSE\xEF\xD6\xE7\x75\x06\x8B
\x49\x30\x4C\x0F\x30\xA0\x9E\xFD\x93\x85\xF1\xC8\x2A\x75\x1C\x38
\x05\x79\x01\xE2\x97\xD8\xAF\x80\x38\x60\x0B\x71\x0E\x68\x53\x77
\x2F\x0F\xE6\x4D\x1A\xE5\x8E\x5C\xC2\x52\x3D\x21\x74\x40\x4B\x55\x06
\xAE\x3B\x7D\xBD\x5B\x9D\x7E\x32\xED\xF6\x06\x24\xD9\x29\xA4\xA5
\x8E\x26\x23\xFD\xEE\xF1\x4C\x0F\x74\x5E\x58\xF8\x91\x74\xEF\x91
```

- **Unlock** DPAPI with user password or SHA1(password).
- Decrypt the previous BLOB with **dpapick**.
- You’ll get another binary plist.
  - With the first 32bits providing its real length.
iCloud windows token

- The iCloud token is the pair (dsPrsId, mmeAuthToken).

```xml
<string>email@icloud.com</string>
<string>appleIdAlias</string>
<string>appleIdAliases</string>
<array/>
<false/>
<key>dsPrsId</key>
<string>123456789</string>
<key>firstName</key>
<string>Jailbreaking</string>

<key>cloudKitToken</key>
<string>
<key>brMigrated</key>
<false/>
<key>mailToken</key>
<string>
<key>mmeAuthToken</key>
<string>AQABCDEF0123456789ABCCCDDEFFBgt0=</string>
```
iCloud windows token

- Using **Elcomsoft EPBB** (*Elcomsoft Phone Password Breaker*)
  - No marketing here, they are the only providing such a cool feature
  - Go online with the decrypted token and get the all the data
  - Indeed they truly keep your data ;)

![Elcomsoft EPBB interface](image)
exit strategy

levels completed!
conclusions

• **Offensive Digital Investigations**
  • *to cope with security measures (anti-forensics?)*

• Given a ram dump, you can unlock DPAPI (swapped data issue)

• A bit manual…

  • **Dpapick** code next to be released
    • with automatic scripts (aka **probes**) *Push the button* (“who said hurrah??! “).

• We have not seen many stuffs…
  • Just one: **pin / picture password** in Win8 allow instant recover or user password…
I have a dream

• What about **dpapick** + **rekall** + **plaso**?

• You could, just with a command, grab
  • All passwords/SHA1 from **hiberfil** files
    • Even using VSS on previous hiberfils!
  • Then search/parse BLOBs and automatically decrypt them

• Cool, isn’t it?
thank you

!!!