Decrypting DPAPI data

Jean-Michel Picod, Elie Bursztein
EADS, Stanford University
Data Protection API

• Introduced in Windows 2000
• Aim to be an easy way for application to store safely data on disk
• Tie encryption key to user password
Developer point of view

Application

DPAPI
Developer point of view

Application

DPAPI
Developer point of view

Application

DPAPI
Developer point of view

Application

DPAPI
DPAPI is a simple API*
Why digging deeper?

- Offline forensic
- EFS on Linux
- Security?
Previous work

- Multiples attempts to analyze DPAPI
  - Some incomplete (Wine)
  - Some close source (Nir Sofer - NirSoft)
Take away

• Decrypt offline sensitive data
• Recovers user previous password
• Do a key escrow attack
Outline

- DPAPI overview
- Decryption process
- Security design implications
- DPAPIck demo
How the system interacts with DPAPI

DPAPI

cryptoAPI
crypt32.dll
How the system interacts with DPAPI

DPAPI cryptoAPI crypt32.dll

Local Security Authority cryptoAPI crypt32.dll
How the system interacts with DPAPI

DPAPI

Local Security Authority

cryptoAPI

crypt32.dll

EFS

Encrypted file
How the system interacts with DPAPI

DPAPI

cryptoAPI
crypt32.dll

Local Security Authority

cryptoAPI
crypt32.dll

EFS

EFS Encrypted file
How the system interacts with DPAPI

- **DPAPI**
  - cryptoAPI
  - crypt32.dll

- **Local Security Authority**
  - cryptoAPI
  - crypt32.dll

- **EFS**
  - Encrypted file

- **EFS user private key**
How the system interacts with DPAPI

DPAPI
- cryptoAPI
- crypt32.dll

Local Security Authority
- cryptoAPI
- crypt32.dll

EFS

Encrypted file

EFS

user private key
BOOL WINAPI CryptUnprotectData (
*pDataIn,
*ppszDataDescr,
*pOptionalEntropy,
pvReserved,
*pPromptStruct,
dwFlags,
*pDataOut
DPAPI CryptUnprotectData Function

BOOL WINAPI CryptUnprotectData (
*pDataIn,
*ppszDataDescr,
*pOptionalEntropy,
pvReserved,
*pPromptStruct,
dwFlags,
*pDataOut

Encrypted data aka data blob
BOOL WINAPI CryptUnprotectData ( 

*pDataIn,

*ppszDataDescr,

*pOptionalEntropy,

pvReserved,

*pPromptStruct,

dwFlags,

*pDataOut

Optional description
BOOL WINAPI CryptUnprotectData (  
    *pDataIn, 
    *ppszDataDescr, 
    *pOptionalEntropy,          Optional entropy (salt)  
    pvReserved, 
    *pPromptStruct, 
    dwFlags, 
    *pDataOut)
BOOL WINAPI CryptUnprotectData (  
*pDataIn,  
*ppszDataDescr,  
*pOptionalEntropy,  
pvReserved,  
*pPromptStruct,  
dwFlags,  
*pDataOut  
)
DPAPI CryptUnprotectData Function

BOOL WINAPI CryptUnprotectData (  
    *pDataIn, 
    *ppszDataDescr, 
    *pOptionalEntropy, 
    pvReserved,  
    *pPromptStruct, 
    dwFlags,  
    *pDataOut

    Decrypted data
Derivation scheme

User
Derivation scheme

User → SHA1(password) → Pre key
Derivation scheme

User → SHA1(password) → Pre key → Master Key
Derivation scheme

User \[\rightarrow\] \text{SHA1(password)} \[\rightarrow\] Pre key \[\rightarrow\] Master Key

Master Key \[\rightarrow\] Blob key \[\rightarrow\] Blob key \[\rightarrow\] Blob key
Derivation scheme

User → SHA1(password) → Pre key → Master Key → Blob key → Blob key → Blob key
Blob structure

- Returned to the application (opaque structure)
- Store user encrypted data
- Contains decryption parameters
key subtleties

- SHA1 passwords are in UTF-16LE
- SID for HMAC are also in UTF-16LE (don’t forget the \0 !)
- Windows 2000 do not use SHA1/3DES. We think it uses SHA1/RC4 (Anyone want to try ?).
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbIV;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
<tr>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbIV;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>

**Nb of crypto providers**
data blob structure key fields

DWORD cbProviders;
GUID *arrProviders; ← Crypto providers GUID
DWORD cbKeys;
GUID *arrKeys;
WCHAR *ppszDataDescr;
DWORD idCipherAlgo;
BYTE *pbIV;
DWORD idHashAlgo;
BYTE *pbSalt;
BYTE *pbCipher;
BYTE *pbHMAC;
## data blob structure key fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbIV;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>

---

**Nb of masters keys**

---

Jean-Michel Picod, Elie Bursztein

http://www.dpapick.com

Tuesday, February 2, 2010
### Data Blob Structure Key Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td><code>cbProviders</code></td>
</tr>
<tr>
<td>GUID</td>
<td><code>*arrProviders</code></td>
</tr>
<tr>
<td>DWORD</td>
<td><code>cbKeys</code></td>
</tr>
<tr>
<td>GUID</td>
<td><code>*arrKeys</code></td>
</tr>
<tr>
<td>WCHAR</td>
<td><code>*ppszDataDescr</code></td>
</tr>
<tr>
<td>DWORD</td>
<td><code>idCipherAlgo</code></td>
</tr>
<tr>
<td>BYTE</td>
<td><code>*pbIV</code></td>
</tr>
<tr>
<td>DWORD</td>
<td><code>idHashAlgo</code></td>
</tr>
<tr>
<td>BYTE</td>
<td><code>*pbSalt</code></td>
</tr>
<tr>
<td>BYTE</td>
<td><code>*pbCipher</code></td>
</tr>
<tr>
<td>BYTE</td>
<td><code>*pbHMAC</code></td>
</tr>
</tbody>
</table>

- **Masters keys GUID**
data blob structure key fields

DWORD      cbProviders;
GUID       *arrProviders;
DWORD      cbKeys;
GUID       *arrKeys;
WCHAR      *ppszDataDescr;  ← Optional description
DWORD      idCipherAlgo;
BYTE       *pbIV;
DWORD      idHashAlgo;
BYTE       *pbSalt;
BYTE       *pbCipher;
BYTE       *pbHMAC;
### data blob structure key fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbIV;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>

Encryption algorithm ID
data blob structure key fields

- DWORD cbProviders;
- GUID *arrProviders;
- DWORD cbKeys;
- GUID *arrKeys;
- WCHAR *ppszDataDescr;
- DWORD idCipherAlgo;
- BYTE *pbIV;
- BYTE *pbSalt;
- BYTE *pbCipher;
- BYTE *pbHMAC;

Initialization vector
data blob structure key fields

- DWORD cbProviders;
- GUID *arrProviders;
- DWORD cbKeys;
- GUID *arrKeys;
- WCHAR *ppszDataDescr;
- DWORD idCipherAlgo;
- BYTE *pbIV;
- DWORD idHashAlgo;
- BYTE *pbSalt;
- BYTE *pbCipher;
- BYTE *pbHMAC;

Hash algorithm ID
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbIV;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>

Salt generated by DPAPI
<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbIV;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>

Encrypted data
## Data Blob Structure Key Fields

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbIV;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>

Blob HMAC
Master key structure

- Store the key used to decrypt blob
- Encrypted with the user password
- Renewed every 3 months
The master key file

Header
The master key file

<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys infos</td>
</tr>
</tbody>
</table>
## The master key file

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
</tr>
<tr>
<td>Keys infos</td>
</tr>
<tr>
<td>Master key</td>
</tr>
</tbody>
</table>
The master key file

<table>
<thead>
<tr>
<th>Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys infos</td>
</tr>
<tr>
<td>Master key</td>
</tr>
<tr>
<td>Key ?</td>
</tr>
<tr>
<td>Header</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>Keys infos</td>
</tr>
<tr>
<td>Master key</td>
</tr>
<tr>
<td>Key ?</td>
</tr>
<tr>
<td>Footer</td>
</tr>
</tbody>
</table>

The master key file

Tuesday, February 2, 2010
Header structure

dwVersion;
nullPad1;

Master key

szKeyGUID[36];

nullPad2;
### Header structure

<table>
<thead>
<tr>
<th>Header</th>
<th>dwVersion;</th>
<th>File version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys infos</td>
<td>nullPad1;</td>
<td></td>
</tr>
<tr>
<td>Master key</td>
<td>szKeyGUID[36];</td>
<td></td>
</tr>
<tr>
<td>Key ?</td>
<td>nullPad2;</td>
<td></td>
</tr>
<tr>
<td>Footer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Header structure

<table>
<thead>
<tr>
<th>Header</th>
<th>dwVersion;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys infos</td>
<td>nullPad1;</td>
</tr>
<tr>
<td>Master key</td>
<td>szKeyGUID[36];</td>
</tr>
<tr>
<td>Key ?</td>
<td>nullPad2;</td>
</tr>
<tr>
<td>Footer</td>
<td>Master key GUID</td>
</tr>
</tbody>
</table>

Tuesday, February 2, 2010
Key infos structure

Header
Keys infos
Master key
Key ?
Footer

dwUnknown;
cbMasterKey;
cbMysteryKey;
dwHMACLen;
nullPad3;
Key infos structure

- Header
- Keys infos
  - Master key
  - Key ?
- Footer

- dwUnknown;
- cbMasterKey;
- cbMysteryKey;
- dwHMACLen;
- nullPad3;

Master Key struct length
Key infos structure

- Header
- Keys infos
- Master key
- Key ?
- Footer

```

dwUnknown;

cbMasterKey;

cbMysteryKey;

dwHMACLen;
nullPad3;
```

Key ? struct length

Tuesday, February 2, 2010
Key infos structure

- Header
- Keys infos
  - Master key
  - Key ?
- Footer

- dwUnknown;
- cbMasterKey;
- cbMysteryKey;
- dwHMACLen;  \(\rightarrow\) HMAC length
- nullPad3;
Master key structure

dwMagic;
pbSalt[16];
cbIteration;
idMACAlgo;
idCipherAlgo;
pbCipheredKey[];
Master key structure

dwMagic;
pbSalt[16];
cbIteration;
idMACAlgo;
idCipherAlgo;
pbCipheredKey[];

Key salt
Master key structure

Header
Keys infos
Master key
Key ?
Footer

dwMagic;
pbSalt[16];
cbIteration;
idMACAlgo;
idCipherAlgo;
pbCipheredKey[];

PBKDF2 nb rounds
Master key structure

dwMagic;
pbSalt[16];
cbIteration;
idMACAlgo;
idCipherAlgo;
pbCipheredKey[];

HMAC algorithm ID
Master key structure

dwMagic;
pbSalt[16];
cbIteration;
idMACAlgo;
idCipherAlgo;
pbCipheredKey[];

Encryption Algo id
Master key structure

dwMagic;
pbSalt[16];
cbIteration;
idMACAlgo;
idCipherAlgo;
pbCipheredKey[];

Encrypted key
Decrypting the Master key

\[
\text{DPAPIDecryptKey}(\text{sha1}, \text{encKey}) \{ \\
\text{tmp-key} = \text{HMAC}(\text{sha1}, \text{SID}) \\
\text{pre-key} = \text{PBKDF2}(\text{decryptKey}, \text{Salt}, \text{ID\_ALGO}, \text{nbIteration}) \\
\text{3desKey} = \text{pre-key}[0 - 23] \\
\text{3desIV} = [24 - 31] \\
(hmac[0-35], \text{DWORD}[36-39], \text{master-key}[40-104]) = \text{3des-cbc}(\text{3desKey}, \text{iv}, \text{encKey}) \\
\}
\]
key structure

- Seems to have the same structure than the master key
- One round of derivation (XP not Seven)
- 256 bits (half size of the real master-key)
Possible explanation

- The documentation states a compatibility mode for Windows 2000 exists.
- The registry key to trigger it is unknown.
- If we are correct and W2k uses RC4, then the mystery key is possibly a RC4 key (256 bits is the correct size).
- PBKDF2 used to compute the IV?
Possible explanation continued

• We know that RC4 have a weak key scheduling algorithm (remember WEP ?)

• Might be a potential weakness (or not)
Header structure

<table>
<thead>
<tr>
<th>Header</th>
<th>dwMagic;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys infos</td>
<td>credHist[16];</td>
</tr>
<tr>
<td>Master key</td>
<td></td>
</tr>
<tr>
<td>Key ?</td>
<td></td>
</tr>
<tr>
<td>Footer</td>
<td></td>
</tr>
</tbody>
</table>

Tuesday, February 2, 2010
Header structure

- Header
- Keys infos
- Master key
- Key ?

```
dwMagic;
credHist[16]; → Password GUID
```
<table>
<thead>
<tr>
<th></th>
<th>XP</th>
<th>Vista</th>
<th>Seven</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PBKDF2 rounds</strong></td>
<td>4000</td>
<td>24000</td>
<td>Variable (factor ?)</td>
</tr>
<tr>
<td><strong>Symmetric algorithm</strong></td>
<td>3DES</td>
<td>3DES</td>
<td>AES</td>
</tr>
<tr>
<td><strong>Hash algorithm</strong></td>
<td>SHA1</td>
<td>SHA1</td>
<td>SHA512</td>
</tr>
</tbody>
</table>
Decrypting a blob

Data blob
Decrypting a blob

Data blob → Master key GUID → Master key file
Decrypting a blob

Data blob → Master key GUID → Master key file

Master key file → Salt, Nb iterations → Pre key
Decrypting a blob

Data blob → Master key GUID → Master key file

Salt, Nb iterations → Pre key

SHA1(password) → User SID
Decrypting a blob

- Data blob
- Master key GUID
- Master key file
- Salt, Nb iterations
- Pre key
- SHA1(password)
- User SID
- Master key
Decrypting a blob

Data blob

Master key GUID

Master key file

Salt, Nb iterations

Cipher + Key

Pre key

SHA1(password)

User SID

Master key
Decrypting a blob

Data blob

Master key GUID

Master key file

Cipher + Key

Salt, Nb iterations

Pre key

SHA1(password)

User SID

Master key

Blob key
Decrypting a blob

Data blob

Master key GUID

Master key file

Salt, Nb iterations

Pre key

SHA1(password)

User SID

Master key

Cipher + Key

Salt + IV

Blob key

Master key file

Master key

Blob key
Decrypting a blob

Data blob ➔ Master key GUID ➔ Master key file

Salt + IV ➔ Cipher + Key ➔ Pre key

Salt, Nb iterations ➔ SHA1 (password) ➔ User SID

Additional password ➔ Master key ➔ Blob key
Decrypting a blob

Data blob → Master key GUID → Master key file

Pre key → Salt, Nb iterations → SHA1(password) → User SID

Salt + IV → Cipher + Key

Salt + IV → Master key 

Master key → Pre key

Master key → Blob key

Blob key → Additional password

Blob key → Additional entropy
Did I miss something?

• How the OS knows the current master key?
• How the OS decides to renew the master key?
• What happen when the user changes his password?
Key renewal process

- Renewed every 3 months automatically
  - Passive process: executed when CryptProtect called
- Hardcoded limit (location unknown)
  - Possibly in psbase.dll (MS crypto provider)
  - Can be reduced by using registry override
Master key selection

- All master keys are kept because Windows can’t tell if a key is still used
- Keys are stored in `%APPDATA%/Microsoft/Protect/[SID]`
- Current master key is specified in the Preferred file
The Preferred file

• Simply contains:
  “GUID master key” . “timestamp”

• The key is renewed when
  current time > timestamp
The Preferred file

- Simply contains:
  "GUID master key" . "timestamp"

- The key is renewed when
  current time > timestamp

➡ Key escrow attack: Plant a key and update the Preferred file every 3 months (e.g using the task scheduler)
User password renewal

- Master keys are re-encrypted when the password change
- Experimentally not all of them, just the last few ones
Decrypting a blob

Data blob → Master key GUID

Master key file

Cipher + Key

Salt + IV

Pre key

Salt, Nb iterations

Master key

SHA1(password)

Blob key

Additional password

Additional entropy
Decrypting a blob

Data blob → Master key GUID → Master key file → CREDHIST GUID → CREDHIST

- Cipher + Key
- Salt, Nb iterations
- Pre key → SHA1(password)
- Master key
- Blob key

- Salt + IV
- Additional password
- Additional entropy
SHA1 (password)
CREDHIST overview

Structure
pass n-1

Decrypt

SHA1 (password)
CREDHIST overview

Structure
pass n-1

Structure
pass n-2

Decrypt

SHA1(password)

Decrypt

Decrypt
CREDHIST overview

Structure pass n-3

Structure pass n-2

Structure pass n-1

Decrypt

SHA1(password)
CREDHIST overview

Structure pass 1

Structure pass 2

...

Structure pass n-3

Structure pass n-2

Structure pass n-1

SHA1(password)

Decrypt

Decrypt

Decrypt

Decrypt
CREDHIST entry structure main fields

idHashAlgo;
dwRounds;
dwCipherAlgo;
bSID[12];
dwComputerSID[3];
dwAccountID;
bData[28];
bPasswordID[16]
CREDHIST entry structure main fields

- idHashAlgo;
- dwRounds;
- dwCipherAlgo;
- bSID[12];
- dwComputerSID[3];
- dwAccountId;
- bData[28];
- bPasswordID[16]

Hash algo ID
CREDHIST entry structure main fields

idHashAlgo;
dwRounds; → Nb rounds
idHashAlgo;
dwRounds;
dwCipherAlgo;
bSID[12];
dwComputerSID[3];
dwAccountId;
bData[28];
bPasswordID[16]
CREDHIST entry structure main fields

- idHashAlgo;
- dwRounds;
- dwCipherAlgo;
- bSID[12];
- dwComputerSID[3];
- dwAccountID;
- bData[28];
- bPasswordID[16]
CREDHIST entry structure main fields

idHashAlgo;
dwRounds;
dwCipherAlgo;
bSID[12];
dwComputerSID[3];
dwAccountID;
bData[28];
bPasswordID[16]
CREDHIST entry structure main fields

- idHashAlgo;
- dwRounds;
- dwCipherAlgo;
- bSID[12];
- dwComputerSID[3];
- dwAccountID;
- bData[28];
- bPasswordID[16]
CREDHIST entry structure main fields

idHashAlgo;
dwRounds;
dwCipherAlgo;
bSID[12];
dwComputerSID[3];
dwAccountId;
bData[28];
bPasswordID[16]
CREDHIST entry structure main fields

- idHashAlgo;
- dwRounds;
- dwCipherAlgo;
- bSID[12];
- dwComputerSID[3];
- dwAccountID;
- bData[28];
- bPasswordID[16]

Encrypted password SHA1
CREDHIST entry structure main fields

idHashAlgo;
dwRounds;
dwCipherAlgo;
bSID[12];
dwComputerSID[3];
dwAccountId;
bData[28];
bPasswordId[16]  ←  Password GUID
Decryption algorithm overview

DecryptCredhist{

SID = (USID-ComputerID-AccountID)

\text{tmp-key} = \text{HMAC(sha1, SID)}

\text{pre-key} = \text{PBKDF2(decryptKey, Salt, ID_ALGO, nblIteration)}

\text{3desKey} = \text{pre-key}[0 - 23]

\text{3desIV} = [24 - 31]

(SHA1[0-19], HMAC[20-39]) = 3des-cbc

(3desKey, iv, encKey)
DPAPick demo

grab a copy from http://dpapick.com
LSA

- LSASS secret contains a DPAPI_SYSTEM value
- Length == 2 * SHA1
- Usage are unknown
- We think that 1 of them is used as a SYSTEM account “password”
- Need to be confirmed
• Certificate private key is encrypted with DPAPI

• Key are stored in

• To read EFS file offline, we just need to import the user certificate and its private keys in our key store.

• Work in progress in DPAPick
What is next

• Can we build a rogue crypto provider?
• What are the two SHA1 stored in the LSA?
• Where is stored the renewal hard lime?
• CryptDeriveKey needed to be reversed to have a fully portable implementation (Everything else is already portable)
Conclusion

- Open the door to offline forensic
- First step toward EFS on alternative systems
- CREDHIST allows to recover previous passwords
- DPAPick : http://dpapick.com
- Some things remain unknown
Questions ?

Thanks to the nightingale team