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 and the account SID
Developer point of view

Application

DPAPI
DPAPI is a simple API*

Why digging deeper?

- Offline forensic
- EFS on Linux
- Security / cool things?
Previous work

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

• Decrypt offline sensitive data
• Recover user previous passwords (Yes all of them)
• Do a key escrow attack
Outline

• DPAPI overview
Outline

• DPAPI overview
• Decryption process
Outline

• DPAPI overview

• Decryption process

• Security design implications
Outline

- DPAPI overview
- Decryption process
- Security design implications
- DPAPIck demo
**Crypto 911 HMAC**

- HMAC (Message authentication code)
  - Usually used to detect data tampering
  - Used here to derive encrypt key and IV

\[
\begin{align*}
\text{ipad} &= 0x36 \oplus \text{key} \\
\text{opad} &= 0x5c \oplus \text{key} \\
\text{HMAC} &= (\text{opad} \cdot \text{SHA1(ipad.data)})
\end{align*}
\]
• PBKDF2 = Password based key derivation function
• Basically it is a hash function (SHA1 for us) applied n times to slow down the computation.
• Used to defend against brute-force
• Salt is used against rainbow tables attacks.
• 3DES : Triple DES encryption
  • Encrypt, Decrypt, Encrypt
  • Exist in two flavor : 2 keys or 3 keys (64 bits each)
  • Windows use the strong version with 3 keys
How the system interacts with DPAPI
How the system interacts with DPAPI

DPAPI
\textit{cryptoAPI}
\textit{crypt32.dll}
How the system interacts with DPAPI

DPAPI
cryptoAPI
crypt32.dll

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

<table>
<thead>
<tr>
<th>DPAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryptoAPI</td>
</tr>
<tr>
<td>crypt32.dll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local Security Authority</th>
</tr>
</thead>
<tbody>
<tr>
<td>cryptoAPI</td>
</tr>
<tr>
<td>crypt32.dll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted file</td>
</tr>
</tbody>
</table>

Wednesday, February 3, 2010
How the system interacts with DPAPI

- DPAPI
  - cryptoAPI
  - crypt32.dll

- Local Security Authority
  - cryptoAPI
  - crypt32.dll

- EFS
  - Encrypted file

http://www.dpapick.com

Wednesday, February 3, 2010
How the system interacts with DPAPI

- **DPAPI**: cryptoAPI, crypt32.dll
- **Local Security Authority**: cryptoAPI, crypt32.dll
- **EFS**
  - Encrypted file
  - user private key
How the system interacts with DPAPI

- DPAPI
 cryptoAPI
crypt32.dll
- EFS
  Encrypted file
  Local Security Authority
  cryptoAPI
crypt32.dll
BOOL WINAPI CryptUnprotectData ( 
    *pDataIn,
    *ppszDataDescr,
    *pOptionalEntropy,
    pvReserved,
    *pPromptStruct,
    dwFlags,
    *pDataOut
)
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
BOOL WINAPI CryptUnprotectData ( 

*pDataIn, 

*ppszDataDescr, 

*pOptionalEntropy, 

pvReserved, 

*pPromptStruct, 

dwFlags, 

*pDataOut 

Optional entropy (salt)
DPAPI CryptUnprotectData Function

BOOL WINAPI CryptUnprotectData (  

    *pDataIn,  
    *ppszDataDescr,  
    *pOptionalEntropy,  
    pvReserved,  
    *pPromptStruct,  
    dwFlags,  
    *pDataOut  

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

Decrypted data
Derivation scheme

User → SHA1(password) → Pre key
Derivation scheme

User $\xrightarrow{\text{SHA1(password)}}$ Pre key $\xrightarrow{}$ Master Key
Derivation scheme

User

SHA1(password) → Pre key

Pre key → Master Key

Master Key → Blob key
Master Key → Blob key
Master Key → Blob key
Derivation scheme

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

http://www.dpapick.com
Blob structure

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

• SHA1 password 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 ?).
### data blob structure key fields

<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>*pbSalt;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbUnknown;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>
data blob structure key fields

<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>*pbSalt;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbUnknown;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>

Nb of crypto providers: 18
data blob structure key fields

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

Crypto providers GUID
data blob structure key fields

(DWORD) cbProviders;
(GUID) *arrProviders;
(DWORD) cbKeys;
(GUID) *arrKeys;
(WCHAR) *ppszDataDescr;
(DWORD) idCipherAlgo;
(BYTE) *pbSalt;
(DWORD) idHashAlgo;
(BYTE) *pbUnknown;
(BYTE) *pbCipher;
(BYTE) *pbHMAC;

Nb of masters keys
<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>*pbSalt;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbUnknown;</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>*pbSalt;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbUnknown;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
</tr>
</tbody>
</table>

Optional description
data blob structure key fields

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

Encryption algorithm ID
data blob structure key fields

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

Salt generated by DPAPI
data blob structure key fields

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

Hash algorithm ID
## Data Blob Structure Key Fields

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
<td></td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
<td></td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
<td></td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbUnknown;</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
<td><code>Unknown data</code></td>
</tr>
<tr>
<td>Field Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>cbProviders;</td>
<td></td>
</tr>
<tr>
<td>GUID</td>
<td>*arrProviders;</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>cbKeys;</td>
<td></td>
</tr>
<tr>
<td>GUID</td>
<td>*arrKeys;</td>
<td></td>
</tr>
<tr>
<td>WCHAR</td>
<td>*ppszDataDescr;</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>idCipherAlgo;</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbSalt;</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbUnknown;</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbCipher;</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbHMAC;</td>
<td></td>
</tr>
</tbody>
</table>

**Encrypted data**
### 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>*pbSalt;</td>
</tr>
<tr>
<td>DWORD</td>
<td>idHashAlgo;</td>
</tr>
<tr>
<td>BYTE</td>
<td>*pbUnknown;</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

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

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

- dwVersion;
- nullPad1;
- szKeyGUID[36];
- nullPad2;

File version
Header structure

- dwVersion;
- nullPad1;
- szKeyGUID[36];
- nullPad2;

Master key GUID
Key infos structure

Header

Keys infos

Master key

Key ?

Footer

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

<table>
<thead>
<tr>
<th>Header</th>
<th>dwUnknown;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys infos</td>
<td>cbMasterKey;</td>
</tr>
<tr>
<td>Master key</td>
<td>cbMysteryKey;</td>
</tr>
<tr>
<td>Key ?</td>
<td>dwHMACLlen;</td>
</tr>
<tr>
<td>Footer</td>
<td>nullPad3;</td>
</tr>
</tbody>
</table>

Master Key struct length: 22
Key info structure

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

dwUnknown;

cbMasterKey;

cbMysteryKey;

dwHMACLen;

nullPad3;

Key ? struct length

Wednesday, February 3, 2010
Key infos structure

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

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

HMAC length
Master key structure

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

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

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

PBKDF2 nb rounds

Wednesday, February 3, 2010
Master key structure

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

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

Encrypted key
Decrypting the Master key

DPAPIDecryptKey(sha1, encKey) {

tmp-key = HMAC(sha1, SID)

pre-key = PBKDF2(decryptKey, Salt, ID_ALGO, nbIteration)

3desKey = pre-key[0 - 23]

3desIV = [24 - 31]

(hmac[0-35], DWORD[36-39], master-key [40-104]) = 3des-cbc(3desKey, iv, encKey)
}

Wednesday, February 3, 2010
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 state a compatibility mode for windows 2000 exist.
- 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 (256bits is the correct size).
- PBKDF2 used to compute the IV ??
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>Keys infos</th>
<th>Master key</th>
<th>Key ?</th>
</tr>
</thead>
</table>

```
dwMagic;
credHist[16];
```
Header structure

<table>
<thead>
<tr>
<th>Header</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys infos</td>
<td></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>

```
dwMagic;
credHist[16];  ← Password GUID
```
## Differences between windows version

<table>
<thead>
<tr>
<th></th>
<th>XP</th>
<th>Vista</th>
<th>Seven</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBKDF2 rounds</td>
<td>4000</td>
<td>24000</td>
<td>Variable (factor ?)</td>
</tr>
<tr>
<td>Symmetric algorithm</td>
<td>3DES</td>
<td>3DES</td>
<td>AES</td>
</tr>
<tr>
<td>Hash algorithm</td>
<td>SHA1</td>
<td>SHA1</td>
<td>SHA512</td>
</tr>
</tbody>
</table>
Decryption 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
- Salt, Nb iterations
- Pre key
Decrypting a blob

- Data blob
- Master key file
- Pre key

- Master key GUID
- Salt, Nb iterations
- 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

Cipher + Key → Pre key

Salt, Nb iterations → 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

Cipher + Key

Salt + IV

Master key

Blob key

User SID

SHA1(password)
Decrypting a blob

Data blob → Master key GUID → Master key file → Salt, Nb iterations → Pre key → Master key → Blob key

Cipher + Key → Salt + IV → SHA1(password) → User SID → Additional password
Decrypting a blob

Data blob

Master key GUID

Master key file

Salt, Nb iterations

Pre key

SHA1(password)

User SID

Master key

Additional password

Blob key

Cipher + Key

Salt + IV

Additional entropy
DecryptBlob() {
kt = SHA1(masterkey)
opad = 0x5c xor kt
ipad = 0x36 xor kt
i = SHA1(opad.SHA1(ipad . salt).entropyCond)
kd = CryptDeriveKey(i) //not reversed (yet)
CryptDecrypt(data, kd)
}
Did I miss something?
Did I miss something?

• How the OS knows the current master key?
Did I miss something?

- How the OS knows the current master key?
- How the OS decides to renew the master key?
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

Salt + IV → Cipher + Key → Pre key

Salt, Nb iterations → SHA1(password) → Master key

Additional password → Additional entropy → Blob key
Decrypting a blob

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

- Cipher + Key
- Salt + IV
- Pre key
- Salt, Nb iterations
- Master key
- Blob key
- SHA1(password)
- Additional password
- Additional entropy
CREDHIST overview

SHA1(password)
CREDHIST overview

Structure
pass n-1

Decrypt

SHA1(password)
CREDHIST overview

SHA1(password) → Decrypt → Structure pass n-1 → Decrypt → Structure pass n-2 → Decrypt → Structure pass n-3
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; → Nb rounds

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]

User USID
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] → Password GUID
DecryptCredhist{

SID = (USID-ComputerID-AccountID)

tmp-key = HMAC(sha1, SID)

pre-key = PBKDF2(decryptKey, Salt, ID_ALGO, nbIteration)

3desKey = pre-key[0 - 23]

3desIV = [24 - 31]

(SHA1[0-19], HMAC[20-39]) = 3des-cbc
(3desKey, iv, encKey)
DPAPIck demo
Warning

• **DPAPIck** is in **ALPHA stage**. Use it at your own risk! You have been warned. It is just a POC

• Know bugs:
  • No HMAC checks -> No key check.
  • No Seven support, tested only on XP
  • No conditional entropy / strong password in UI
  • Don’t choose the correct master key by itself
  • Buffer overflows :)

Jean-Michel Picod, Elie Bursztein

http://www.dpapick.com
DPAPIck future

- We made the choice to release early so you know we are telling the truth and everyone can start playing.
- We will provide a more robust version and eventually open the source code so one day Linux will read EFS files :)
- It just too soon for this.
• 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