LSA secrets in Windows

Passcape Software

1 What are LSA secrets?

LSA secrets is a special protected storage for important data used by the **Local Security Authority** (LSA) in Windows. LSA is designed for managing a system's local security policy, auditing, authenticating, logging users on to the system, storing private data. Users' and system's sensitive data is stored in secrets. Access to all secret data is available to system only. However, as shown below, some programs, in particular Windows Password Recovery, allow to override this restriction.

2 What is stored in LSA secrets?

Originally, the secrets contained cached domain records. Later, Windows developers expanded the application area for the storage. At this moment, they can store PC users' text passwords, service account passwords (for example, those that must be run by a certain user to perform certain tasks), Internet Explorer passwords, RAS connection passwords, SQL and CISCO passwords, SYSTEM account passwords, private user data like EFS encryption keys, and a lot more.

For example, the **NL$KM** secret contains the cached domain password encryption key. **L$RTMTIMEBOMB** stores the amount of time left until the expiration of an inactivated copy of Windows. **L$HYDRAENCKEY** stores the public RSA2 key used in the Remote Desktop Protocol. Incidentally, even despite the fact that the automatic login is not set, in certain versions of Windows 7 secrets can contain the plaintext of the administrator account password, thus compromising the entire target system.

3 Where are LSA secrets stored?

LSA secrets are stored in an encrypted form in the Windows registry, in the **HKEY_LOCAL_MACHINE/Security/Policy/Secrets** key. The parent key, **HKEY_LOCAL_MACHINE/Security/Policy**, contains the additional data, necessary for accessing and decrypting the secrets. Here are the descriptions of some values of this key.

Key:	**HKEY_LOCAL_MACHINE/Security/Policy/SecDesc**
Value name:	
Data type:	**REG_BINARY**
Description:	security descriptor for accessing the registry tree with secrets.

Key:	**HKEY_LOCAL_MACHINE/Security/Policy/PolState**
Value name:	
Data type:	**REG_BINARY**
Description:	current state of the secrets subsystem.

Key:	**HKEY_LOCAL_MACHINE/Security/Policy/PolRevesion**
Value name:	
Data type:	**REG_BINARY**
Description:	contains the version of the subsystem.

Key:	**HKEY_LOCAL_MACHINE/Security/Policy/PolPrDmS**
Value name:	

Data type:	**REG_BINARY**
Description:	domain SID.

Key:	**HKEY_LOCAL_MACHINE/Security/PolicyPolPrDmN**
Value name:	
Data type:	**REG_BINARY**
Description:	domain name.

Key:	**HKEY_LOCAL_MACHINE/Security/PolicyPolEKList**
Value name:	
Data type:	**REG_BINARY**
Description:	contains the list of encryption keys for LSA secrets.

The value 1.1 in **PolRevesion** matches the NT operation system, 1.5 – Windows 2000, 1.7 – Windows XP and Win2K3, 1.9 – Windows Vista, 1.10 – Windows 7. Before Windows Vista, only one encryption key was stored in the registry, in the **PolSecretEncryptionKey** value. Beginning with Windows Vista, **PolEKList** can contain several encryption keys.

4 LSA Secrets in detail

On the physical level, secrets are stored in a binary registry file SECURITY with the secret name for the key. For example, **Security/Policy/Secrets/$MACHINE.ACC**. Each secret in the registry is represented by five values:

1. **CurrVal** - current encrypted value of the secret.
2. **CupdTime** - last update time, as an 8-byte FILETIME structure.
3. **OldVal** - previous value of the secret.
4. **OupdTime** - previous update time.
5. **SecDesc** - security descriptor, i.e. which users can access the secret, and which are banned from accessing it.

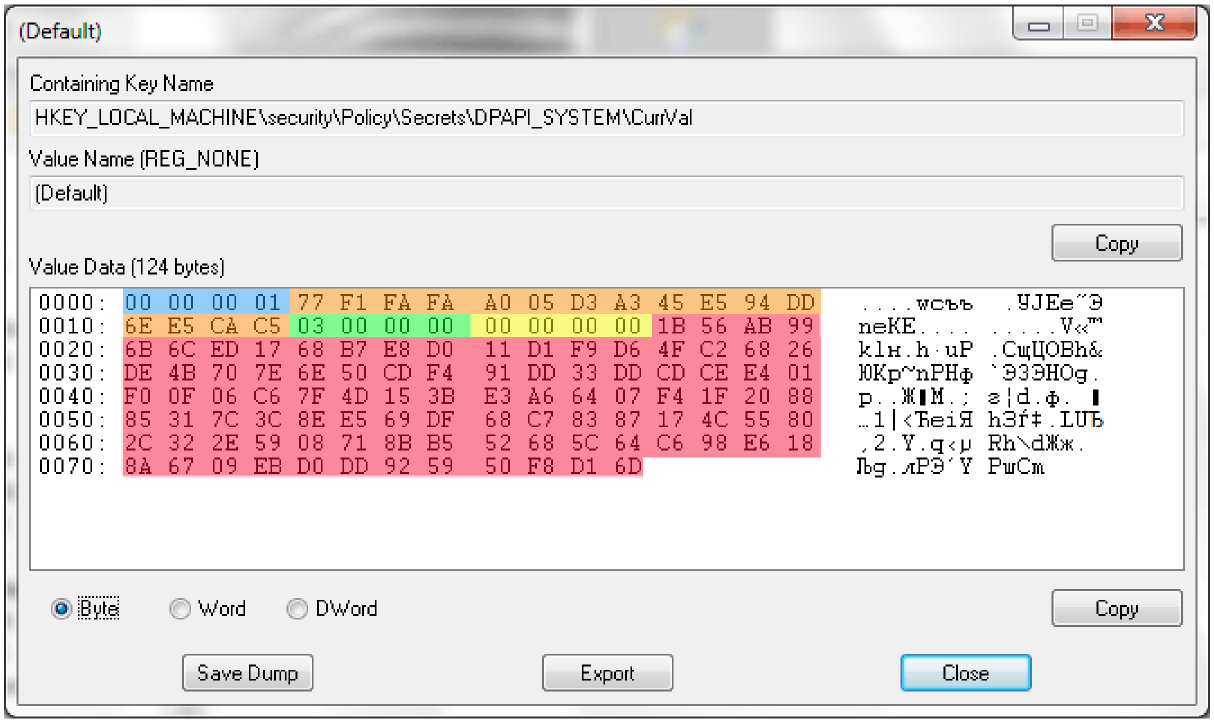
If the system is unable to read/decrypt one of the secrets, it writes the sixth value in it, **PolMod**, which indicates that the secret is damaged. For example, if a transaction to the LSA database was not completed due to a power outage or registry file damage.

5 CurrVal and OldVal data structure

Beginning with version 1.9, the structure of secrets has changed dramatically; therefore, we are not going to cover the old format. Instead of a single encryption key, now you can bind each secret to any value on the encryption key list (**PolEKList**).

There is also an option to select an encryption algorithm! So, the first 4 bytes in the data structure is the version of the data; then there follows a 16-byte encryption key identifier for locating the necessary key on the list. That is followed by a DWORD with an identifier for a list of encryption algorithms the secret is encrypted with.

For example, the value 3 matches a bundle of the SHA-256 hashing algorithm and the AES-256 block encryption algorithm. The algorithm identifier is followed by a 4-byte value with different flags used during the decryption. And, finally, there goes the encrypted data. See the figure.



Containing Key Name
HKEY_LOCAL_MACHINE\security\Policy\Secrets\DPAPI_SYSTEM\CurrVal
Value Name (REG_NONE)
(Default)
Value Data (124 bytes)

```
0000:  00 00 00 01 77 F1 FA FA  A0 05 D3 A3 45 E5 94 DD   ....wcъъ  .ЧJЕе˜Э
0010:  6E E5 CA C5 03 00 00 00  00 00 00 00 1B 56 AB 99   neKE....  ....V«™
0020:  6B 6C ED 17 68 B7 E8 D0  11 D1 F9 D6 4F C2 68 26   klн.h·uP  .СщЦОВh&
0030:  DE 4B 70 7E 6E 50 CD F4  91 DD 33 DD CD CE E4 01   ЮKp~nPHф  `Э3ЭHОg.
0040:  F0 0F 06 C6 7F 4D 15 3B  E3 A6 64 07 F4 1F 20 88   p..Ж▮M.;  г¦d.ф. ▮
0050:  85 31 7C 3C 8E E5 69 DF  68 C7 83 87 17 4C 55 80   …1|<Ћeiя  hЗѓ‡.LUЂ
0060:  2C 32 2E 59 08 71 8B B5  52 68 5C 64 C6 98 E6 18   ,2.Y.q‹µ  Rh\dЖж.
0070:  8A 67 09 EB D0 DD 92 59  50 F8 D1 6D               Љg.лPЭ'Y  PшCm
```

6 LSA secret encryption in Windows 2000, XP, 2003

Up to Windows Vista, the decryption of the secrets looked rather trivial. First, one needed to decrypt the secrets encryption key. Here is what it looked like:

```
BOOL CSecrets::DecryptPrimaryKey()
{
   BYTE rc4key[0x10];

   MD5Init();
   MD5Update(m_pSyskey,0x10);
   for ( int i=0; i<1000; i++)
      MD5Update(((LPBYTE)m_pCypherKey)+0x3C,0x10);
   MD5Final(rc4key);

   RC4SetKey(rc4key,0x10);
   RC4Decrypt(((LPBYTE)m_pCypherKey)+0xC,0x30);

   return ( memcmp(((LPBYTE)m_pCypherKey)+0xC,CYPHERKEY_AUTHENTIFICATOR,0x10)==0 );
}
```

Where **m_pSyskey -** 16-byte SYSKEY value;
m_pCypherKey - value from the registry key
HKEY_LOCAL_MACHINE/Security/Policy/PolSecretEncryptionKey

Once the secrets encryption key was obtained, one could proceed to the decryption of the secrets. The secrets were encrypted using DES algorithm.

7 Lsa secret encryption in Windows Vista and later OSes

In Windows Vista (and higher OSes), the encryption algorithm, as it was mentioned earlier, has become much more sophisticated. First, one still needs to decrypt the list of the encryption keys (yes, now multiple keys are allowed), stored in **HKEY_LOCAL_MACHINE/Security/Policy/PolEKList**. Then proceed to the actual secrets. Each secret now stores a key identifier, encryption algorithm identifier and the actual encrypted data. A working algorithm for decrypting the key looks like:

- Read the key value and find the encryption key identifier.
- On the list of encryption keys (PolEKList), find the necessary key using the identifier you obtained earlier.
- Decrypt the secret using the algorithm identifier and the found key.

Thus, secrets in the LSA database can be not only encrypted with different algorithms, but also have different original context. For example, use SYSKEY from other PC.

8 Reading and editing secrets

There is a set of API for handling secrets available to software developers. Thus, any Windows application can create and read its own secrets, but only within the boundaries of current user context. Refer to appendix 1 with the source code for reading secrets.

If you require viewing or editing LSA secrets, for instance, to delete your account's text password, you can take advantage of Windows Password Recovery tool, which has a convenient plugin for handling LSA secrets. By the way, this plugin works with both current operating system's secrets and with external registry files.

9 Appendix

Source code of the program for reading LSA Secrets. Note that not all secrets can be read under user context. Besides, the Administrator privileges are required. The executable of the program can be downloaded at the following link.

```
// LsaSecretReader.cpp : Defines the entry point for the console application.
```

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include <ntsecapi.h>


#pragma comment (lib, "Advapi32")

PLSA_UNICODE_STRING InitLsaString(LPWSTR wszString, PLSA_UNICODE_STRING lsastr)
{
        if ( !lsastr )
                return NULL;

        if ( wszString )
        {
                lsastr->Buffer=wszString;
                lsastr->Length=(USHORT)lstrlenW(wszString)*sizeof(WCHAR);
                lsastr->MaximumLength=lsastr->Length+2;
        }
        else
        {
                lsastr->Buffer=L"";
                lsastr->Length=0;
                lsastr->MaximumLength=2;
        }

        return lsastr;
}


int _tmain(int argc, _TCHAR* argv[])
{
        NTSTATUS status;
        LSA_OBJECT_ATTRIBUTES att;
        LSA_HANDLE pol;
        LSA_UNICODE_STRING secret, *data=NULL;

        if ( argc!=2 )
        {
                _tprintf(TEXT("Syntax: %s secretnamen"),argv[0]);
                return 1;
        }

        memset(&att,0,sizeof(att));

        status=LsaOpenPolicy(NULL,&att,0,&pol);
        if ( status!=ERROR_SUCCESS )
        {
                _tprintf(TEXT("LsaOpenPolicy error: %lXn"),status);
                return 2;
        }
        InitLsaString(argv[1],&secret);
        status=LsaRetrievePrivateData(pol,&secret,&data);
        if ( status!=ERROR_SUCCESS )
        {
                _tprintf(TEXT("LsaRetrievePrivateData error: %lXn"),status);
                return 3;
        }
        LsaClose(pol);

        if ( data && data->Buffer && data->Length )
        {
                for ( USHORT i=0; i<data->Length; i+=16 )
                {
                        _tprintf(TEXT("%04X: "),i);
                        LPBYTE ptr=(LPBYTE)data->Buffer;
                        ptr+=i;
                        for ( int j=0; j<min(16,data->Length-i); j++ )
                                _tprintf(TEXT("%02X "),ptr[j]);
                        _tprintf(TEXT("n"));
                }
        }
        else
        {
                _tprintf(TEXT("No data"));
        }

        return 0;
}
```

Example of the output

```
C:>LsaSecretReader.exe DPAPI_SYSTEM
0000: 01 00 00 00 73 4F 19 CF 6B B7 6C 8A BC 6D 35 EF
0010: 19 9C A6 3E 9A 80 A7 0C 9D D4 FD B1 20 C6 B1 A5
0020: 7A 87 5F 2B 51 3E 1D E0 45 9B 99 B2
```