

## About Network Password & Credential Store

Windows provides 'Credential Store' framework to store the network based passwords in a secure encrypted format. This provides convenient and reliable mechanism to store the passwords for network logins so that user don't have to enter the password every time while accessing the network resources.

Not only Windows uses it to store network authentication passwords, but also other applications such as Outlook, Windows Live Messenger, Remote Desktop, GMail Notifier etc uses the same mechanism for storing their login passwords.

Windows also allows applications to seamlessly manage this 'Credential Store' using Credential Management API functions such as CredEnumerate, CredRead, CredWrite, CredProtect, CredUnprotect, CredDelete etc.

## Location of Credential Store

Windows 'Credential Store' keeps the user credentials in the encrypted format at user specific locations. The storage mechanism is slightly different for XP and Vista/Win7 platforms.

### For Windows XP

On Windows XP, the encrypted user credentials are stored in the hidden file called 'Credentials' inside both APPDATA and LOCALAPPDATA locations mentioned below.

```
APPDATA Location - C:\Documents and Settings\\Application
Data\Microsoft\Credentials\\
```

```
LOCALAPPDATA Location - C:\Documents and Settings\\Local
Settings\Application Data\Microsoft\Credentials\\
```

### For Vista & Windows 7

Vista onwards, the user credentials are stored in the multiple files with random name (generated using GUID) inside both APPDATA and LOCALAPPDATA locations mentioned below. (There will be separate credential file for each of the network accounts)

```
APPDATA Location - C:\Users\\AppData\Roaming\Microsoft\Credentials\<
```

```
LOCALAPPDATA Location -
C:\Users\\AppData\Local\Microsoft\Credentials\<
```

Based on type of password and application, one of these locations are chosen to store the corresponding credential file. For example, Windows Live Messenger & Remote Desktop login passwords are stored at

LOCALAPPDATA location and all other type of passwords are stored at APPDATA location.

These credential files are hidden and not visible even if you have enabled 'show hidden files/folders' in the folder settings. To view these files, use the command prompt and navigate to the above locations and then issue the 'dir /a' command.

Each of these credential file begins with standard signature at the beginning of the file which can be used for its verification. Here are the first 16 bytes representing the identity of such credential file.

```
0x01, 0x00, 0x00, 0x00, 0xD0, 0x8C, 0x9D, 0xDF, 0x01, 0x15, 0xD1, 0x11, 0x8C, 0x7A,  
0x00, 0xC0
```

## Types of Network Passwords

Windows 'Credential Store' supports different type of network passwords. Each type uses different kind of encryption and storage mechanism. Also each type of password requires different level of access privileges for decryption.

Here are the primary types....

- Generic Password
- Domain Password
- Domain Visible Password / .NET Passport
- Certificates

'Domain Password' type provides highest level of security as these passwords can only be managed by a specific system process called LSASS.exe. No other process, even though it is running as administrator can decrypt such passwords. All network authentication, Remote Desktop and Outlook exchange server login passwords belongs to this type.

'Generic Password' type is used to store all user specific credentials and only that user can decrypt such passwords. Internet Explorer (basic & digest authentication) and Windows Live Messenger uses this method to store their login credentials.

'Domain Visible Password' is similar to 'Generic Password' mechanism. However in case of 'Generic Password' both username & password are encrypted and for 'Domain Visible Password' only password is encrypted. Also both types use different salt as entropy for decryption of the password. MSN messenger 7.0 uses this method to store the login passwords with the name as '.Net Passport'

## Recovering 'Generic Network' Passwords

Generic network passwords are user specific and can be decrypted only in the security context of corresponding user. As mentioned earlier Windows Credential Management functions can be used to manage these passwords.

Here is the sample code which demonstrates listing all the generic passwords and then decrypting them

using **CryptUnprotectData** function.

```
void EnumerateGenericNetworkPassword()
{
    DATA_BLOB DataIn;
    DATA_BLOB DataOut;
    DATA_BLOB OptionalEntropy;
    tmpSalt[37];
    char *strSalt={"abe2869f-9b47-4cd9-a358-c22904dba7f7"};

    char strURL[1024];
    char strCredentials[1024];
    char strUsername[1024];
    char strPassword[1024];

    //Create the entropy/salt required for decryption...
    for(int i=0; i< 37; i++)
        tmpSalt[i] = (short int)(strSalt[i] * 4);

    OptionalEntropy.pbData = (BYTE *)&tmpSalt;
    OptionalEntropy.cbData = 74;

    DWORD Count;
    PCREDENTIAL *Credential;

    //Now enumerate all http stored credentials...
    if(CredEnumerate(NULL,0,&Count,&Credential))
    {
        for(int i=0;i<Count;i++)
        {
            if( Credential[i]->Type == CRED_TYPE_GENERIC)
            {
                DataIn.pbData = (BYTE *)Credential[i]->CredentialBlob;
                DataIn.cbData = Credential[i]->CredentialBlobSize;

                if(CryptUnprotectData(&DataIn, NULL,
                                    &OptionalEntropy, NULL,
                                    NULL,0,&DataOut))
                {
                    //Extract username & password from credentails (username:password)
                    sprintf_s(strCredentials, 1024, "%S", DataOut.pbData);

                    char *ptr = strchr(strCredentials, ':');
                    *ptr = '\\0';
                    strcpy_s(strUsername, 1024, strCredentials);
                    ptr++;
                    strcpy_s(strPassword, 1024, ptr);
                }
            }
        }
    }
}
```

```

        printf("Generic Network Password account details,
              Username=%s, Password=%s", strUsername, strPassword);

    }

}

} // End of FOR loop

CredFree(Credential);
}

} //End of function

```

The above code uses the **CredEnumerate** function to go through all the stored network password accounts for current user. Next it checks if the account type is CRED\_TYPE\_GENERIC. If generic type of account is found then it decrypts the user credential data using the CryptUnprotectData function which is part of Windows Crypto API package. Upon successful decryption it contains both username and password in the clear text separated by semicolon.

### Recovering 'Domain Visible Network' (.Net Passport) Passwords

This method uses the similar mechanism as the generic type to encrypt the password. Only difference is that here different salt is used to encrypt/decrypt the credentials. Also in this case only password is encrypted rather than both username & password pair as in case of generic method.

Here is the modified working code for decrypting this type of network passwords

```

void EnumerateDotNetPassportPassword()
{
    DATA_BLOB DataIn;
    DATA_BLOB DataOut;
    DATA_BLOB OptionalEntropy;
    tmpSalt[37];
    char *strSalt={"82BD0E67-9FEA-4748-8672-D5EFE5B779B0"};

    char strCredentials[1024];
    char strUsername[1024];
    char strPassword[1024];

    //Create the entropy/salt required for decryption...
    for(int i=0; i< 37; i++)
        tmpSalt[i] = (short int)(strSalt[i] * 4);

    OptionalEntropy.pbData = (BYTE *)&tmpSalt;

```

```

OptionalEntropy.cbData = 74;

DWORD Count;
PCREDENTIAL *Credential;

//Now enumerate all http stored credentials....
if(CredEnumerate(NULL,0,&Count,&Credential))
{
    for(int i=0;i<Count;i++)
    {
        if( Credential[i]->Type == CRED_TYPE_DOMAIN_VISIBLE_PASSWORD)
        {
            DataIn.pbData = (BYTE *)Credential[i]->CredentialBlob;
            DataIn.cbData = Credential[i]->CredentialBlobSize;

            sprintf_s(strUsername, 1024, "%S", Credential[i]->UserName);

            if(CryptUnprotectData(&DataIn, NULL,
                                &OptionalEntropy, NULL,NULL,0,&DataOut))
            {
                //Decrypted data contains password in clear text
                sprintf_s(strPassword, 1024, "%S", DataOut.pbData);

                printf(".Net Passport Account details,
                       Username=%s, Password=%s", strUsername, strPassword);

            }

        }

    }

} // End of FOR loop

CredFree(Credential);
}

} //End of function

```

The above code uses the CredEnumerate function to go through all the stored network password accounts for current user. Next it checks if the account type is CRED\_TYPE\_DOMAIN\_VISIBLE\_PASSWORD. If such an account is found then it decrypts the password data using the CryptUnprotectData function. Upon successful decryption it contains the password in clear text.

This method is also termed as '.Net Passport' because it is mainly used by MSN Messenger which stored its login password with the name as '.Net Passport'.

## Recovering 'Domain Network' Passwords

Domain network password method uses more stricter technique for encrypting the credentials thus providing better security over other methods. Only system process, LSASS.EXE can encrypt or decrypt these kind of passwords. LSASS is a Windows core system process responsible for enforcing the security and executing various security oriented tasks.

So in order to decrypt domain passwords one has to perform decryption in the context of LSASS process. This can be achieved by injecting remote thread into LSASS process using CreateRemoteThread function. This is similar to the technique used by pwdump tool to dump the LM/NTLM hashes for the user accounts on the system.

Once the thread is injected, it can use the special undocumented function LsaICryptUnprotectData exported from Lsasrv.dll to decrypt the credentials buffer. Here is the prototype of the function LsaICryptUnprotectData.

```
typedef int (WINAPI *LPFUN_LSAICRYPTUNPROTECTDATA)
(
    LPBYTE encCredData,
    DWORD encCredDataSize,
    DWORD reserved1,
    DWORD reserved2,
    DWORD reserved3,
    DWORD reserved4,
    DWORD dwFlags,
    DWORD reserved5,
    LPBYTE *decCredData,
    LPDWORD decCredDataSize
);
```

This function takes the credential buffer & its size as input and returns the decrypted credential buffer. This decrypted buffer starts with the header whose structure is given below

```
struct DecryptedDataHeader
{
    DWORD dwHeaderId; //0x01 for XP & 0x30 for Vista/Win7
    DWORD dwBufferSize; //size of the entire decrypted data
};
```

Here first field contains the signature and second field indicates the size of entire decrypted buffer.

After the header follows the network password structure for each of the stored accounts. This structure can represent any type of network password accounts and not just the domain password. However username & password is decrypted only for domain accounts and for all other types password is kept in the encrypted format.

The contents of this structure is given below. The internal fields of the structure varies slightly for Vista/Win7 from XP. Hence care needs to be taken while decoding this decrypted data on XP and Vista/Win7 platforms.

[ Note : All the structure information presented below is derived based on the reverse engineering work. So things may slightly differ from real stuff ]

```
// For Windows XP
struct DecryptedNetAccount
{
    DWORD dwItemSize; //total size of this item for XP
    DWORD dwUnknown;
    DWORD dwType;
    DWORD dwFileTimeLowDate;
    DWORD dwFileTimeHighDate;
    DWORD dwZero;
    DWORD dwPersist; //3 => enterprise 2=> local machine
    char unknown[12];
    DWORD dwCredNameSize;
    char strCredName[dwCredNameSize];
    DWORD dwCommentSize;
    char strComment[dwCommentSize];
    DWORD dwAliasSize;
    char strAlias[dwAliasSize];
    DWORD dwUserNameSize;
    char strUserName[dwUserNameSize];
    DWORD dwPasswordSize;
    char password[dwPasswordSize];
    char padding[unknown]; //To make next entry aligned on 8th byte
};

// For Vista & Windows 7
struct DecryptedNetAccountVista
{
    DWORD dwZero;
    DWORD dwType;
    DWORD dwzero;
    DWORD dwFileTimeLowDate;
    DWORD dwFileTimeHighDate;
    DWORD dwSomeSize;
    DWORD dwPersist; //3 => enterprise 2=> local machine
    char unknown[12];
    DWORD dwCredNameSize;
    char strCredName[dwCredNameSize];
    DWORD dwCommentSize;
    char strComment[dwCommentSize];
    DWORD dwAliasSize;
```

```

char strAlias[dwAliasSize];
DWORD dwUnknownSize; // only for vista/win7
char strUnknown[dwUnknownSize]; //only for vista/win7
DWORD dwUserNameSize;
char strUserName[dwUserNameSize];
DWORD dwPasswordSize;
char password[dwPasswordSize];
};

```

The above structure is of variable length and depends upon the length of the various text fields. Each of these text fields are NULL terminated UNICODE strings except the password field. Also at the end of each structure, extra padding (On windows XP only) bytes are added so as to align next entry on 8 byte boundary.

On Vista/Windows 7 platform, only one network account is stored per credential file.

Here is the sample code to decode this decrypted credential data and display the username/password information for all the stored domain password accounts.

```

// Check for valid signature
dwHeaderId = decDataBuffer->dwHeaderId
if( dwHeaderId != 0x1 && dwHeaderId != 0x30 )
{
    printf("\n Decrypted data is not valid, mismatch in the header");
    return;
}

//Set the index to first entry
index = sizeof(DecryptedDataHeader);

while( index < dwBufferSize )
{
    DecryptedDataItem *decDataItem = (DecryptedDataItem*) &pDecData[index];

    if( decDataItem->index == CRED_TYPE_DOMAIN_PASSWORD )
    {
        printf("\n Network Name = %S", decDataItem->strCredName);

        //move index to username field and print it
        printf("\n Username = %S ", decDataItem->strUserName);

        //next move to password field and print it out...
        WideCharToMultiByte(CP_ACP, 0,
            decDataItem->strPassword,

```

```

        decDataItem->dwPasswordSize,
        strPassword, 1000, NULL, NULL );
printf("\n Password = %s ", strPassword);

}

//move to next password entry for XP platform
if( XP platform )
    index = index + decDataItem->dwItemSize
else
    break; //for vista/win7 only one account per credential file is stored
}

```

The above code first verifies if the decrypted data is valid by comparing the signature bytes in the header. Next it loops through each of the structure of domain password type and prints the network name, username & password in clear text.

Also note that the decrypted buffer from LsaICryptUnprotectData contains data for all type of network passwords, not just the domain type. But the password for other types is still in the encrypted format which can be further decrypted using the same procedure as explained earlier for respective types.