

## About Password Manager of Internet Explorer

Like most browsers, Internet Explorer also has the single sign-on feature which stores the username/password for already authenticated websites. Whenever user login to any website, IE prompts the user for consent to store the password for future use. If user acknowledges then username/password along with website link will be stored in IE secret store. So the next time onwards whenever user visits the same website, IE automatically populates the username/password field from its store thus preventing user from entering credentials every time.

## Internals of IE Password Manager

Internet Explorer stores two type of passwords, Autocomplete and HTTP basic authentication based passwords. Autocomplete passwords are normal website login passwords such as email, forum websites. HTTP basic authentication password is the one which is required to login to website itself. As soon as user tries to access the website, IE prompts with login dialog box asking for username/password. Generally proxy servers and router/modem configuration websites uses these kind of authentication mechanism. Here is the screenshot of login dialog box shown by IE while accessing the router webpage.

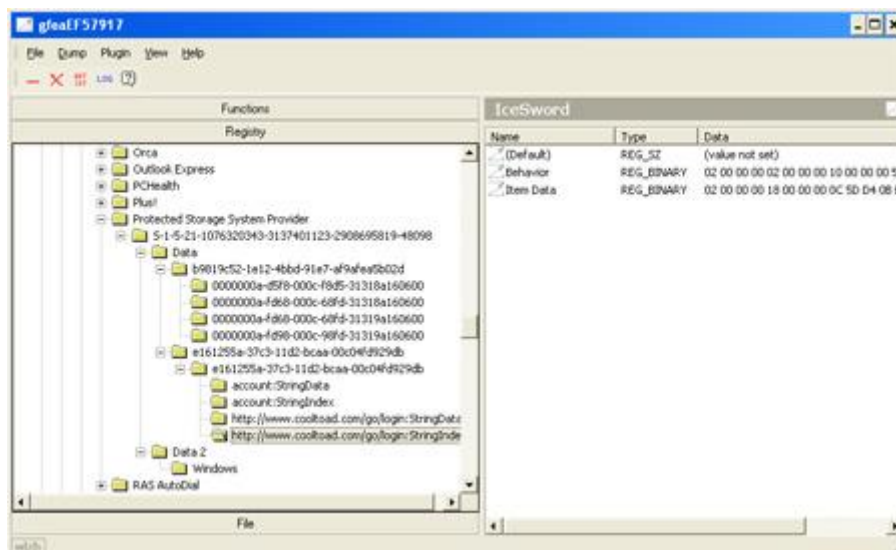


## Password Store Location for IE version 4 to 6.0

Internet Explorer below version 7 stores both Autocomplete and HTTP basic authentication passwords in the secure location known as 'Protected Storage'. Windows has introduced 'Protected Storage' to allow applications such as IE, Outlook to store the secrets securely in an encrypted format. Below is the registry location corresponding to the 'Protected Storage'.

```
HKEY_CURRENT_USER\Software\Microsoft\Protected Storage System Provider
```

The actual contents of this registry location are encrypted and not visible in Regedit tool. However you can use IceSword's registry editor to view the actual encrypted contents as shown in the below screenshot.

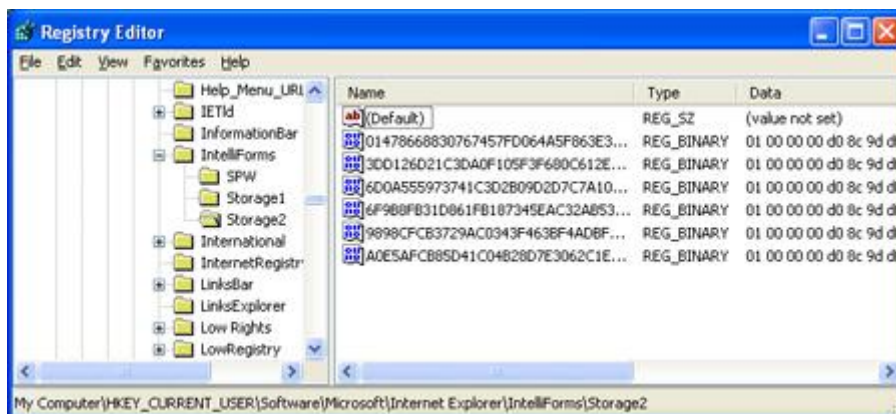


### Password Store Location for IE version 7 or more

With version 7 onwards IE has changed the location of password store to provide better security mechanism compared to existing 'Protected Storage'. Now IE stores all the Autocomplete passwords in below mentioned registry location in an encrypted format.

```
HKKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\IntelliForms\Storage2
```

Here is the screenshot of typical entries stored at this location



Here each entry corresponds to a hash of the website for which username/password has been stored. So one must know the website login link in order to recover the password.

The HTTP basic authentication passwords are stored in the 'Credentials store'. The 'Credentials Store' is newly introduced secret store mechanism by Windows and it is generally used to store the network login passwords. Its location is given below.

```
[Windows XP] C:\Documents and Settings\[username]\Application Data\Microsoft\Credentials
[Windows Vista+] C:\Users\[username]\AppData\Roaming\Microsoft\Credentials
```

## Recovering Passwords from IE Secret Store

Based on IE version, different method need to be used to retrieve the stored passwords. For IE version 4 to 6.0, its enough to decrypt the passwords from 'Protected Storage' which contains both Autocomplete and HTTP authentication passwords. For IE version 7 and above we have to decrypt the http authentication passwords from 'Credentials Store' and Autocomplete passwords from IE registry storage location.

The remaining section of this article explains all these methods in detail along with sample code.

## Decrypting Passwords from Protected Storage

Protected Storage is also used by other applications such as Outlook, MSN messenger to store the passwords. So we need to separate out these passwords from the Autocomplete and HTTP basic authentication passwords stored by IE.

Windows provides API functions to retrieve the passwords stored in the 'Protected Storage'. All these API functions are exported from pstorec.dll which is installed as part of Windows system. One can use PStoreCreateInstance function to get the pointer to protected storage and then use the enumeration functions to list all the secrets in plain text. The code sample below illustrate this method using the 'Protected Storage' API functions.

```
#import "pstorec.dll" no_namespace

void ListIEProtectedStorageSecrets()
{
    IPStorePtr PStore;
    IEnumPStoreTypesPtr EnumPStoreTypes;
    GUID TypeGUID;
    char strSiteUrl[1024];
    char strSiteCredentials[1024];
    char szItemGUID[1024];
    char strUsername[1024];
    char strPassword[1024];

    HRESULT hRes = PStoreCreateInstance(&PStore, 0, 0, 0);

    hRes = PStore->EnumTypes(0, 0, &EnumPStoreTypes);

    while( EnumPStoreTypes->raw_Next(1, &TypeGUID, 0) == S_OK )
    {
        sprintf_s(szItemGUID, 1024, "%x", TypeGUID);

        IEnumPStoreTypesPtr EnumSubTypes;
```

```

hRes = PStore->EnumSubtypes(0, &TypeGUID, 0, &EnumSubTypes);

GUID subTypeGUID;
while(EnumSubTypes->raw_Next(1,&subTypeGUID,0) == S_OK)
{
    IEnumPStoreItemsPtr spEnumItems;
    HRESULT hRes = PStore->EnumItems(0, &TypeGUID, &subTypeGUID, 0, &spEnumItems);

    //Now enumerate through each of the stored entries.....

    LPWSTR strWebsite;
    while( spEnumItems->raw_Next(1, &strWebsite, 0) == S_OK)
    {
        sprintf_s(strSiteUrl, 1024, "%ws", siteName);

        unsigned long psDataLen = 0;
        unsigned char *psData = NULL;
        char *sptr;
        _PST_PROMPTINFO *pstiinfo = NULL;

        //read the credentials for this website entry
        hRes = PStore->ReadItem(0, &TypeGUID, &subTypeGUID, siteName, &psDataLen,
&psData, pstiinfo, 0);

        if( lstrlen((char *)psData)<(psDataLen-1) )
        {
            int i=0;
            for(int m=0; m<psDataLen; m+=2)
            {
                if(psData[m]==0)
                    strSiteCredentials[i]=' ';
                else
                    strSiteCredentials[i]=psData[m];
                i++;
            }

            strSiteCredentials[i-1]=0;
        }
        else
        {
            sprintf_s(strSiteCredentials, 1024, "%s", psData);
        }

        //Now decode the username & password from strSiteCredentials for different types

        //5e7e8100 - IE:HTTP basic authentication based passwords
        //username and passwords are seperated by ':'
        if(lstrcmp(szItemGUID, "5e7e8100") ==0 )

```

```

    {
        strPassword[0]=0;
        sptr = strstr(strSiteCredentials, ":");

        if( sptr != NULL )
        {
            strcpy_s(strPassword, 1024, sptr+1);
            *sptr = 0;
            strcpy_s(strUsername, 1024, strSiteCredentials);
        }

        printf("\n website = %S, username = %s, password = %s", strSiteUrl,
strUsername, strPassword);
    }

    //e161255a - IE autocomplete passwords
    if(lstrcmp(szItemGUID,"e161255a")==0)
    {

        if(strstr(strSiteUrl, "StringIndex" ) == 0 )
        {
            if(strstr(strSiteUrl,":String")!=0)
                *strstr(strSiteUrl,":String")=0;

            lstrcpy_n(strPassword,strSiteUrl,8);

            if( !( ( strstr(strPassword,"http://")==0)&&( strstr(strPassword,"https://")==0)
) )
            {
                //username & passwords are seperated by ','
                strPassword[0]=0;
                sptr = strstr(strSiteCredentials, ",");
                if( sptr != NULL )
                {
                    strcpy_s(strPassword, 1024, sptr+1);
                    *sptr = 0;
                    strcpy_s(strUsername, 1024, strSiteCredentials);
                }

                printf("\n website = %s, username = %s, password = %s", strSiteUrl,
strUsername, strPassword);

            }
        }

    } //end of autocomplete if

```

```

    } //inner while loop

    } //middle while loop

    } //top while loop

} //end of function

```

## Decrypting HTTP Basic Authentication Passwords from Credentials Store

IE since version 7 onwards uses 'Credentials Store' to store the HTTP basic authentication passwords. The same store is also used to store the network login passwords as well. However its easy to distinguish between the two as the passwords stored by IE begins with the identifier text 'Microsoft\_WinInet' and they are of type 1.

The passwords are encrypted using the Windows Cryptography functions after salting them with the text generated from the GUID 'abe2869f-9b47-4cd9-a358-c22904dba7f7'. The decrypted text contains the username & password pair separated by semicolon.

Windows provides the Credential Management functions to add/remove the secrets from 'Credentials Store'. We can use the function called **CredEnumerate** to enumerate through these secrets and then decrypt them using CryptUnprotectData function as shown in the below code example.

Credits : Thanks to SapporoWorks for original work [Reference 2]

```

void DecryptIEHttpAuthPasswords()
{
    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 == 1) &&
            _strnicmp(Credential[i]->TargetName, "Microsoft_WinInet_",
strlen("Microsoft_WinInet_")) == 0 )
        {
            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("\n\n Website=%s\n Username=%s\n Password=%s", &Credential[i]-
>TargetName, strUsername, strPassword);

            }
        }
    } // End of FOR loop

    CredFree(Credential);
}

} //End of function

```

The sample output of this program is shown below....

```

Website=Microsoft_WinInet_www.google.com:443/Please log in to your Google Account
Username=gmailuser
Password=gmailpass

Website=Microsoft_WinInet_192.168.1.100:80/WebAdmin

```

```
Username=admin
Password=admin
```

## Decrypting Autocomplete Passwords for IE

IE 7 onwards uses tricky method to store the Autocomplete passwords as explained earlier. Instead of storing the website directly, the hash of the website link is stored in the registry with encrypted username/password information. This way user can always get the username/password automatically whenever he/she visits the corresponding website in IE. But the password stealing is circumvented to certain extent as the cracker now need to know the website login link to get the password.

However one can use the website links stored in the IE history to try for the match similar to the brute force approach used in traditional password recovery method. This is very effective solution as all the visited websites will be stored in IE history automatically unless the user has explicitly deleted it.

The code sample below shows how to calculate the hash of the website link and then decrypting the secrets using Windows cryptography functions. This is the modified and extended version of original article published by SapporoWorks[2].

```
Credits : Thanks to SapporoWorks for original work [Reference 2]
```

```
//
// Calculate the hash for the Website URL
//
void GetURLHashString(wchar_t *wstrURL, char *strHash, int dwSize)
{
    HCRYPTPROV hProv = NULL;
    HCRYPTHASH hHash = NULL;

    CryptAcquireContext(&hProv, 0, 0, PROV_RSA_FULL, CRYPT_VERIFYCONTEXT);

    CryptCreateHash(hProv, CALG_SHA1, 0, 0, &hHash);

    if( CryptHashData(hHash, (unsigned char *)wstrURL, (wcslen(wstrURL)+1)*2, 0) )
    {
        // retrieve 20 bytes of hash value
        DWORD dwHashLen=20;
        BYTE Buffer[20];

        //Get the hash value now...
        if( CryptGetHashParam(hHash, HP_HASHVAL, Buffer, &dwHashLen, 0) )
        {
            //Convert the 20 byte hash value to hexadecimal string format...
            char TmpBuf[1024];
            unsigned char tail=0; // used to calculate value for the last 2 bytes
```

```

    for(int i=0;i<20;i++)
    {
        unsigned char c = Buffer[i];
        tail+=c;
        sprintf_s(TmpBuf, 1024, "%s%2.2X", strHash, c);
        strcpy_s(strHash, dwSize, TmpBuf);
    }

    // add the last 2 bytes
    sprintf_s(TmpBuf, 1024, "%s%2.2X",strHash,tail);
    strcpy_s(strHash, dwSize, TmpBuf);

}

CryptDestroyHash(hHash);
}

CryptReleaseContext(hProv, 0);
}

//
// IE Autocomplete Secret Data structures decoded by Nagareshwar
//

//Main Decrypted Autocomplete Header data
struct IEAutoComplteSecretHeader
{
    DWORD dwSize;           //This header size
    DWORD dwSecretInfoSize; // = sizeof(IESecretInfoHeader) + numSecrets *
sizeof(SecretEntry);
    DWORD dwSecretSize;     //Size of the actual secret strings such as username &
password
    IESecretInfoHeader IESecretHeader; //info about secrets such as count, size etc
    //SecretEntry secEntries[numSecrets]; //Header for each Secret String
    //WCHAR secrets[numSecrets];         //Actual Secret String in Unicode
};

//One Secret Info header specifying number of secret strings
struct IESecretInfoHeader
{
    DWORD dwIdHeader;       // value - 57 49 43 4B
    DWORD dwSize;          // size of this header....24 bytes
    DWORD dwTotalSecrets; // divide this by 2 to get actual website entries
    DWORD unknown;

```

```

    DWORD id4;          // value - 01 00 00 00
    DWORD unknownZero;
};

// Header describing each of the secrets such as username/password.
// Two secret entries having same SecretId are paired
struct SecretEntry
{
    DWORD dwOffset;    //Offset of this secret entry from the start of secret entry
strings
    BYTE  SecretId[8]; //UNIQUE id associated with the secret
    DWORD dwLength;    //length of this secret
};

//
// For each website URL from IE history, this function checks for match with stored hash
// and then decrypts the secrets...
//
void DecryptIEAutocompletePassword(wchar_t *wstrURL)
{
    char strIEStorageKey[] = "Software\\Microsoft\\Internet
Explorer\\IntelliForms\\Storage2";
    char strUrlHash[1024];
    LONG status;
    BOOL result;
    HKEY hKey;
    DWORD DWORD dwSize = 1024;
    DWORD BufferLength=5000;
    DWORD dwType;
    BYTE Buffer[5000];

    //Get the hash for the passed URL
    GetURLHashString(wstrURL, strUrlHash, 1024);

    //Check if this hash matches with stored hash in registry
    if( DoesURLMatchWithHash(strUrlHash) == FALSE )
        return;

    //Now retrieve the encrypted credentials for this registry hash entry...
    if(ERROR_SUCCESS != RegOpenKeyEx(HKEY_CURRENT_USER, strIEStorageKey, 0,
KEY_QUERY_VALUE, &hKey))
        return;

    //Retrieve encrypted data for this website hash...

    //Now get the value...

```

```

status = RegQueryValueEx(hKey, strUrlHash, 0, &dwType, Buffer, &BufferLength);
RegCloseKey(hKey);

if( status != ERROR_SUCCESS || strlen((char*)Buffer) < 1 )
    return;

//Now decrypt the Autocomplete credentials....
DATA_BLOB DataIn;
DATA_BLOB DataOut;
DATA_BLOB OptionalEntropy;
DataIn.pbData = Buffer;
DataIn.cbData = BufferLength;
OptionalEntropy.pbData = (unsigned char*)wstrURL;
OptionalEntropy.cbData = (wcslen(wstrURL)+1)*2;

if( CryptUnprotectData(&DataIn, 0, &OptionalEntropy, NULL, NULL, 0, &DataOut) )
{
    IEAutoComplteSecretHeader *IEAutoHeader = (IEAutoComplteSecretHeader*)
DataOut.pbData;

    //check if the data contains enough length....
    if( DataOut.cbData >= (IEAutoHeader->dwSize + IEAutoHeader->dwSecretInfoSize +
IEAutoHeader->dwSecretSize) )
    {

        //Get the total number of secret entries (username & password) for the site...
        int dwTotalSecrets = IEAutoHeader->IESecretHeader.dwTotalSecrets / 2;

        SecretEntry *secEntry = (SecretEntry*) ( DataOut.pbData +
sizeof(IEAutoComplteSecretHeader) );
        BYTE *secOffset = (BYTE *) ( DataOut.pbData + IEAutoHeader->dwSize + IEAutoHeader-
>dwSecretInfoSize );
        BYTE *curSecOffset;

        // Each time process 2 secret entries for username & password
        for(int i=0; i< dwTotalSecrets ; i++)
        {
            //Get the current secret's offset, username
            curSecOffset = secOffset + secEntry->dwOffset;
            wstrUserName = (WCHAR*) curSecOffset;

            //Get the next secret's offset i.e password
            secEntry++;
            curSecOffset = secOffset + secEntry->dwOffset;
            wstrPassword = (WCHAR*) curSecOffset;

            printf("%S : username = %S & password = %S", wstrURL, wstrUserName,
wstrPassword);

```

```

        //move to next entry
        secEntry++;

    }

}

LocalFree(DataOut.pbData);
}

} //End of function

```

Here each website can have more than one username/password pairs. If user has entered different secrets for same website or if same website link is associated with multiple accounts like in case of Gmail, Orkut etc then multiple secrets will be stored per website.

The final decrypted Autocomplete data will have IEAutoCompleteSecretHeader structure followed by SecretHeader's and clear text Secret strings as shown below...

```

IEAutoCompleteSecretHeader
SecretHeader_0 //Each SecretHeader is represented by SecretEntry structure
SecretHeader_1
...
SecretHeader_n
Secret_0 //Each Secret is null terminated unicode string
Secret_1
...
Secret_n

```

Before we proceed, we need to find out number of secrets within this decrypted data. One simple way is to use dwTotalSecrets variable of IESecretInfoHeader structure which is part of IEAutoCompleteSecretHeader. Here dwTotalSecrets stands for total number of secrets stored for this website. Here secret can represent either username or password. So dividing this number by two will give us actual username/password pairs for the website.

Now, we go through each of the secrets, reading 2 secrets (username & password) at a time. Each time SecretEntry header is read to get the offset of associated unicode Secret string and then move on to next SecretEntry header until we have done with reading all the secrets.