Outlook passwords

Passcape Software

1 Introduction

"Use strong passwords that combine upper- and lowercase letters, numbers, and symbols. Weak passwords don't mix these elements. Strong password: Y6dh!et5. Weak password: House27."

Excerpt from MS Office Outlook user manual

This article was originally meant to tell you about a funny passwords collisions in Outlook's **PST** files. Later on, it was expanded to demonstrate that despite the drawbacks, Outlook's advantages far exceed its closest competitors, as well as to explain the techniques used for storing personal data. Besides, it is very convenient to follow the development of the cryptography using Outlook as an example. It can be generally projected to the development of the entire line of Windows operating system as a whole.

2 PST protection password

So, let's begin with the point that Microsoft Office Outlook's .PST file is file-type data storage on a local computer, which stores contacts, notes, e-mail messages, and other items arranged by a certain order. A .PST file can be used as a default location for delivering e-mail messages. It can be also used for ordering and backing up data.

To protect the content of a .PST file and restrict the unauthorized access to it by third parties, one can set a password of up to15 characters long (Figure 1). In such a case, it is commonly thought that the .PST file cannot be opened unless one knows the original password. Let's see how true that is.

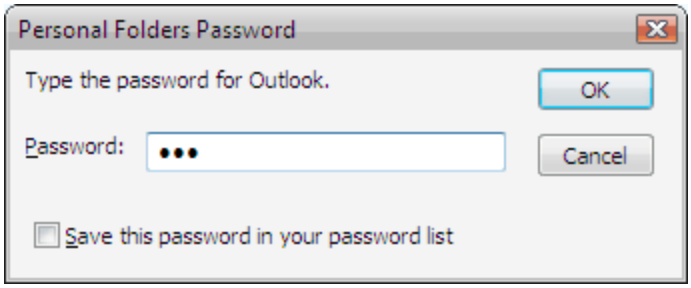

Figure 1. Outlook PST password dialog.

A .PST file access password is neither remembered nor stored in an explicit form. Instead, the computer calculates the password's hash value and stores it in the .PST file or, if the option 'Save this password in your password list' (Figure 1) is selected, in **Windows Registry**, encrypted additionally.

The most interesting thing is that the password hash calculation algorithm is not the actual hashing algorithm - it is rather a simple **CRC32** checksum calculation routine. CRC32 is a redundancy check algorithm, and it surely isn't a hashing routine. For one or another reason, Microsoft had decided to use that algorithm instead of stronger ones like **SHA-1**. Keeping it as

inheritance from older versions of Outlook, MS has not been changing the algorithm for long time, being led, as it seems, by backward compatibility thoughts. On the other hand, it is still unclear, why that wasn't changed when Outlook 2003 was released (Outlook 2003's .PST format has 64-bit internal addressing and is not compatible with the previous versions).

Thus, a password hash appears to be the 32 bits of its checksum. Let's look at CRC32 performance closer:

```
DWORD CPstReader::Crc32( LPBYTE pPassword )
{
   assert( pPassword );

   //set initial crc to zero
   DWORD crc=0;

   //till the end of string
   while ( *pPassword )
      crc = (crc>>8) ^pCRCTable[(BYTE)crc ^(*pPassword++)];

   return crc;
}
```

As one may see from the fragment of the source code, the password (pPassword) is fed at the input, its checksum appears at the output. CRC32's most vulnerable thing is that the **32-bit length** of the password hash is obviously not long enough, and therefore two different passwords' checksums may match! For example, the checksum is the same for passwords **1** and **orxgnm** or for **mozart** and **2920347097**. If you think such collisions are rare, you are much mistaken.

What amazing things these collisions are! The reverse dependence occurs frequently enough also: the longer the password is, the simpler is the collision matching for that password. Let's take a look at the example from Outlook's user manual, the excerpt from which was picked as preface for this article, which was chosen deliberately. There is a simpler combination that may easily replace our 'reliable' password **Y6dh!et5** - it's a 5-character string **JlSfw**.

Yet another interesting observation: if a checksum stored in a .PST file is equal to zero, the program 'thinks' that no password is set. However, since we know that passwords with the same checksums do occur, we can suppose there are passwords, which checksums are also equal to zero. Such passwords indeed exist. Here is a short portion of the large list: **1Rj78C**, **5J8j84**, **ArTniW**.

Crc32("1Rj78C")=0, Crc32("5J8j84")=0, Crc32("ArTniW")=0. If we set one of such passwords to protect our .PST file, we will actually have that file unprotected, and whenever someone tries to access it next time, it will not even prompt for a password. Don't believe it? - Try it yourself.

An experiment has shown that on average it takes about a minute to recover an Outlook hash password using the brute force attack. However, the crypto analysis of CRC32 has revealed that the algorithm is completely reversible for short passwords (up to 4 characters) and

partially reversible for all others. That means, one can recover the original password or its CRC32 equivalent password, that will be indistinguishable for Outlook, almost instantly. It has been proven that it requires not more than 7 characters to pick a collision (password with the same checksum as the original password).

3 PST encryption

If we look at PST file options (Figure 2), we may notice that Outlook, beside everything else, allows encrypting the content. In such case, the password checksum is not kept in the open form in a .PST file; instead, it is additionally encrypted with an encryption algorithm. Let's review the algorithms used for the encryption.

During creation of a new .PST file, Outlook prompts us to choose among the 3 file types:
1. Not encrypted
2. Compressible encryption
3. Strong encryption

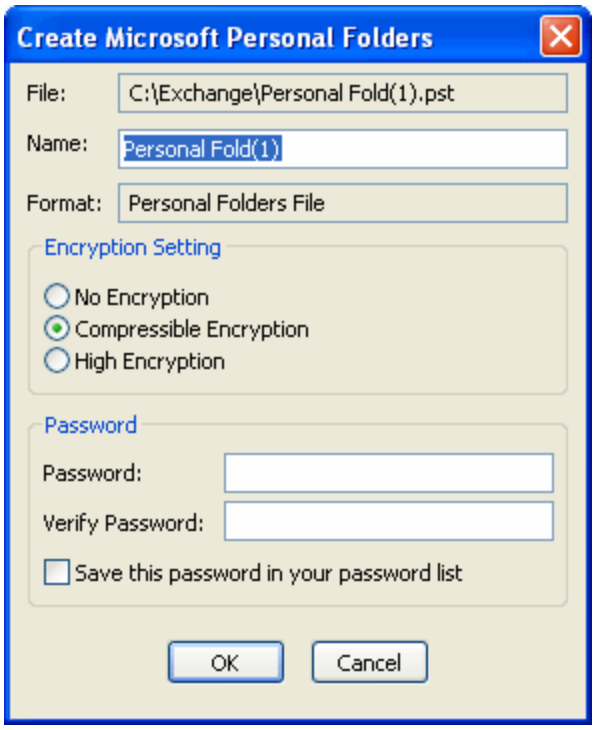

Figure 2. Creating new PST file.

If **no encryption** is selected for a PST file, all of the user's data: contacts, messages, passwords, etc. will be stored in the open form, available to other users. That data can be viewed with, for example, a text editing program.

The **compressible** encryption algorithm is made the way that each character being encrypted is substituted with a different character taken from a special table. Here is the algorithm:

```
BOOL CPstReader::Decrypt1( LPBYTE buf, int iSize )
{
   assert( buf );

   BYTE y=0;
   int x=0;

   //Check input buffer
   if ( buf==NULL )
      return FALSE;

   //Check encryption type
   if ( m_pst.encryption!=PST_ENCRYPT_COMPESSIBLE )
      return FALSE;

   //actual decryption
   while ( iSize-- )
   {
      y=buf[x];
      buf[x++]=m_pTable[y];
   }

   return TRUE;
}
```

The substitution table (**m_pTable**) is meaningfully created the way that allows the optimum further compression of the text encrypted with this algorithm. However, the algorithm itself does not compress the content; it only creates favorable conditions for that.

The **strong encryption** is also a sort of the first substitution algorithm. However, unlike the previous one, it provides much stronger encryption. Another difference in this algorithm is the PST file encrypted with this algorithm cannot be made as compact as it can be made with the first method.

```
BOOL CPstReader::Decrypt2( LPBYTE buf, int iSize, DWORD id )
{
   assert( buf );

   int x=0;
   BYTE y=0;
   WORD wSalt;

   //Check input buffer
   if ( buf==NULL )
      return FALSE;

   //Check encryption type
   if ( m_pst.encryption!=PST_ENCRYPT_STRONG )
      return FALSE;
```

```
    //prepare encryption key from block ID
    wSalt=HIWORD(id) ^LOWORD(id);

    //actual decryption
    while ( iSize-- )
    {
        y=buf[x];

        y+=LOBYTE(wSalt);
        y=m_pTable2[y];

        y+=HIBYTE(wSalt);
        y=m_pTable2[y+0x100];

        y-=HIBYTE(wSalt);
        y=m_pTable2[y+0x200];

        buf[x++]= y - LOBYTE(wSalt++);
    }

    return TRUE;
}
```

To recover a block of data, one must know that block identifier. Without that identifier the recovery process will seem to be somewhat difficult. Again, if we take a closer look at the algorithm's source code, we may easily notice that only a 16-bit block identifier is used, and that allows cutting the number of iterations for picking the encryption key with the brute force attack by much.

4 Techniques for storing e-mail account passwords

As you are reading this article, you may get a somewhat false idea that MS Outlook uses only weak, unreliable password encryption algorithms. That is not quite so. The truth, as the saying goes, will be found in comparison. What is more, we have only analyzed PST. Let's now review and compare Outlook's password encryption mechanisms used for e-mail account with other popular programs.

The majority of popular e-mail clients simply don't care about the safety of users' passwords stored in the programs. For example, Eudora, TheBat! or the older versions of Netscape use the archaic **BASE64** algorithm or its derivations for encrypting user's data. IncrediMail simply applies the **XOR** gamma to the original password (a tongue is not willing to call this thing encryption), the older versions of Opera browser did not encrypt e-mail passwords at all; they stored them as plain text !

The issue is a bit better with the newer versions of the popular e-mail clients Thunderbird, Opera M2, and Outlook Express. All of them use reliable, time-proven algorithms with

mastering keys. That's normally a bundle of **MD5 + RC4** or **SHA + 3DES** or their derivates. Thunderbird, Opera M2, and Outlook Express store their master keys and encryption keys along with encrypted passwords. That allows reverting those passwords seamlessly (or almost seamlessly.)

What concerns Outlook (by the way, don't mix it up with Outlook Express), here we see the most intriguing picture. The entire chronology of the development password storage techniques for Outlook's e-mail accounts can be divided into four periods:

- Prehistoric Age
- Stone Age
- Middle Ages
- Technological Progress Age

More details now.

4.1 Prehistoric age

Prehistoric Age - this is the period of the first steps. They say the first versions of the program were capable of encrypting passwords stored in the registry with the **BASE64** algorithm. That already was an achievement by that time's standards. To recover such passwords, one needed a calculator and a couple of convolutions in the brain.

4.2 Stone age

Stone Age - Outlook 9x - the time of first experiments. A new encryption algorithm that uses the encryption key and unique record identifier appears at this time. It's the age's know-how. Data encrypted with this algorithm could not be recovered without the key and record identifier. Many of today's e-mail clients cannot brag about having such algorithm in their arsenal even today. However, the general idea of that type of encryption was darkened by one crucial weakness - the encryption key and record identifier were stored in the registry, along with the encrypted data.

4.3 Middle ages

Middle Ages - Outlook 2000 - the first standards. E-mail account passwords were now kept in Windows' Protected Storage (one of the upcoming articles will cover the Protected Storage in greater detail). Here is the password decryption algorithm in Protected Storage:

1. Key1 is created for decrypting master key, using SHA(Salt) + SHA(SID) + SHA(Salt). Salt is a global constant. SID is user identifier.
2. Key2 is created for decrypting master key, using SHA(MKSalt) + SHA(key1). MKSalt is binary data unique for each master key stored with it. Key1 is the 20 bytes of data received on the previous step.
3. Master key is decrypted with the DES algorithm and Key2.
4. The recovered master key participates in the decryption of the data encryption key. The data encryption key is stored with data itself and is different for each data record. It consists of 16 bytes, the first half of which is used for the decryption, and the second half is used for the validity check.
5. Now, using this decrypted data key one can decrypt the data record itself (passwords, credentials and other sensitive information).

What a great and stylish idea! This algorithm's major features:

- Single master key for all records. Therefore, it's enough to decrypt it once (steps 1-3). This speeds up the decryption process without bringing down the security level in whole.
- User SID participates in the decryption of the master key; therefore, each user will have a unique master key. Thus, it becomes clear that it is impossible to decrypt the data unless one knows the SID (which is unique for each user). This, however, is somewhat darkened by SID being stored in the registry along with the master key.
- This algorithm, although it was created over 10 years ago, is firm against the so popular these days attacks using rainbow tables.

The suggested encryption routine, which, by the way, was being used already when Internet Explorer 4 was released, was a great breakthrough in data encryption. All modern browsers (Opera, Mozilla, Firefox, and Internet Explorer, up through version 6) use similar password encryption schemes. Take a note that the best Outlook's strongest competitors' creative imagination has stopped here, in the Middle Ages of encryption technology.

4.4 Technological progress age

Technological Progress Age - Outlook 2003 is marching ahead of the world. The new version of the popular e-mail client uses a new encryption algorithm, which continues and logically develops the older one. This algorithm is based upon an important item - it is bound to user's password. This article was not meant to describe the details of the algorithm's function, for this would take quite a few pages of text. Instead, we will just mention that one needs to know at least three things to recover passwords encrypted with this algorithm (Figure 3):

1. User's master key
2. User's SID
3. User's password

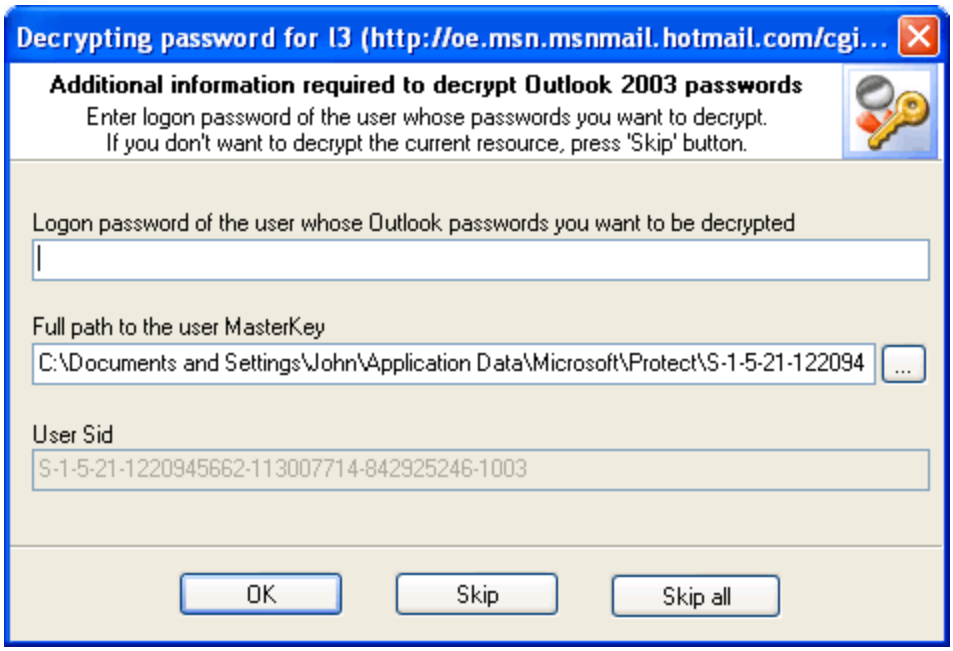

Figure 3. Outlook 2003 e-mail account passwords recovery.

Briefly, the new DPAPI algorithm's advantages are:

- Master key is now stored in a separate folder on the local computer. The access to that folder is partially restricted.
- Master key length is 512 bits, which eliminates the possibility of picking a match in the nearest future.
- New encryption algorithms used, particularly SHA-HMAC, which use a variable number of iterations in loop (4000 by default).
- Encryption algorithms (for both master key and actual data) are fully customizable. One may set any properties supported by the operating system.
- Data protection can be realized using the operating system's permissions level.
- The algorithm is bound to user's logon password.

The encryption of passwords for a logged in user is absolutely transparent. The password is asked for one time only, when user logs on to the operating system. The operating system takes care of the rest. Even if a potential hacker gains physical access to the encrypted data, he will not be able to decrypt that data unless he knows the user's password.

5 Conclusion

We see that with the advent of each new version of this popular e-mail client Microsoft introduces something new, not known until the moment, thus proving that they indeed care about end-user's security. The new Outlook 2003's encryption mechanism is especially great.

Aren't you curious, what the release of the next version, Outlook 2007, will uncover? Let's take the liberty to suppose that PST passwords that are now stored in Windows' registry and

additionally encrypted the way they used to in Outlook 9x (we covered this method when we were talking about the Stone Age) will be encrypted as they are now encrypted in Outlook 2003. At least, that would be logical. It is difficult to predict something for e-mail accounts passwords. We are most likely to see the further development of the new Outlook 2003-based DPAPI encryption algorithm or a new bundle of protected storage + user's password. One way or the other, we will look forward to the release of the new version of this popular application, and we are definitely going to tell about the encryption mechanisms used in the program in one of the upcoming articles.