



Wannacry reverse engineering

Finding the killswitch and unpacking the malware

Itay Yogev
08.11.2023

Introduction

WannaCry היא תוכנת כופר שפורסמה לראשונה ב-12 במאי 2017. התוכנה הצפינה קבצים על מחשבים נגועים ודרשה תשלום כופר של 600 דולר בביטקוין כדי לפענחם.

התקפת WannaCry הייתה מתקפת הסייבר הגדולה ביותר בהיסטוריה, והיא פגעה במאות אלפי מחשבים ביותר מ-150 מדינות ברחבי העולם. בין הקורבנות היו בתי חולים, חברות תעופה, מפעלים וארגונים ממשלתיים.

ההתקפה גרמה לנזק משמעותי, כאשר בתי חולים נאלצו לסגור את מערכות המחשוב שלהם, חברות תעופה נאלצו לבטל טיסות, ומפעלים נאלצו להפסיק את פעילותם.

כיצד פועלת WannaCry?

WannaCry פועלת באמצעות פירצה במערכת ההפעלה Windows של חברת מיקרוסופט. הפירצה ידועה בשם EternalBlue, והיא נחשפה לראשונה על ידי קבוצת הסייבר Shadow Brokers במרץ 2017.

WannaCry משתמשת בפירצה זו כדי להדביק מחשבים Windows שאינם מעודכנים עם הגרסה האחרונה של מערכת ההפעלה. לאחר שהיא נכנסת למחשב, התוכנה מתחילה להצפין את כל הקבצים על המחשב.

השלכות ההתקפה

התקפת WannaCry גרמה לנזק משמעותי ברחבי העולם. בין השלכות של ההתקפה ניתן למנות:

- נזק כלכלי: ההערכות הן שההתקפה גרמה לנזק כלכלי של מאות מיליוני דולרים.
- נזק ציבורי: ההתקפה פגעה בשירותים חיוניים, כגון בתי חולים וחברות תעופה.
- נזק תדמיתי: ההתקפה פגעה בתדמית של חברות ומוסדות רבים.

Analysis

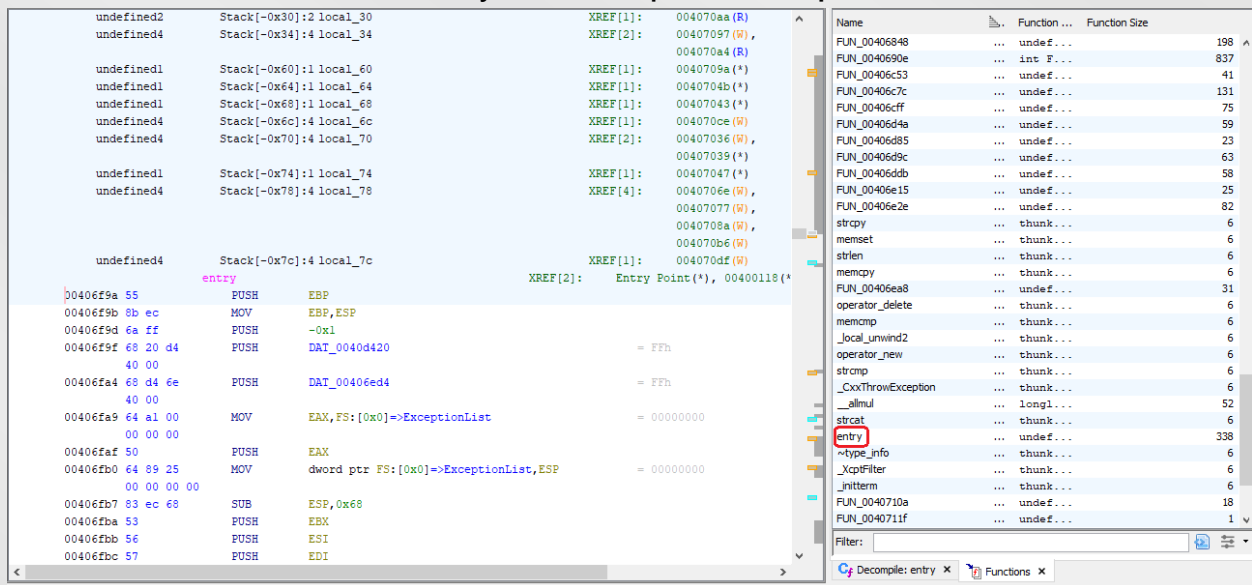
לפני שאנחנו מתחילים עם האנליזה, חשוב לציין שיש מספר גרסאות שונות של Wannacry. ה-malware שאנחנו ננתח מכילה את ה-256SHA הבא:

1d51d46e07d1aaaf34b8b43371bb71aa87812b226341944ab661c286e66de4cd

ניתן למצוא את ה-sample של ה-malware במגוון אתרים, [הנה ה-sample](#) מתוך האתר של MalwareBazaar.

בוא נתחיל עם האנליזה

כאשר אנחנו רואים את כל הפונקציות בתוך הקובץ, אנחנו לא רואים פונקציות תחת השם "main" או "winmain", אבל כן מצאנו פונקציה בשם "entry"



הפונקציה entry, היא קוד ברירת המחדל שיופיע בכל התחלה של קובץ exe במערכת ההפעלה של windows. בוא נראה מה entry עושה

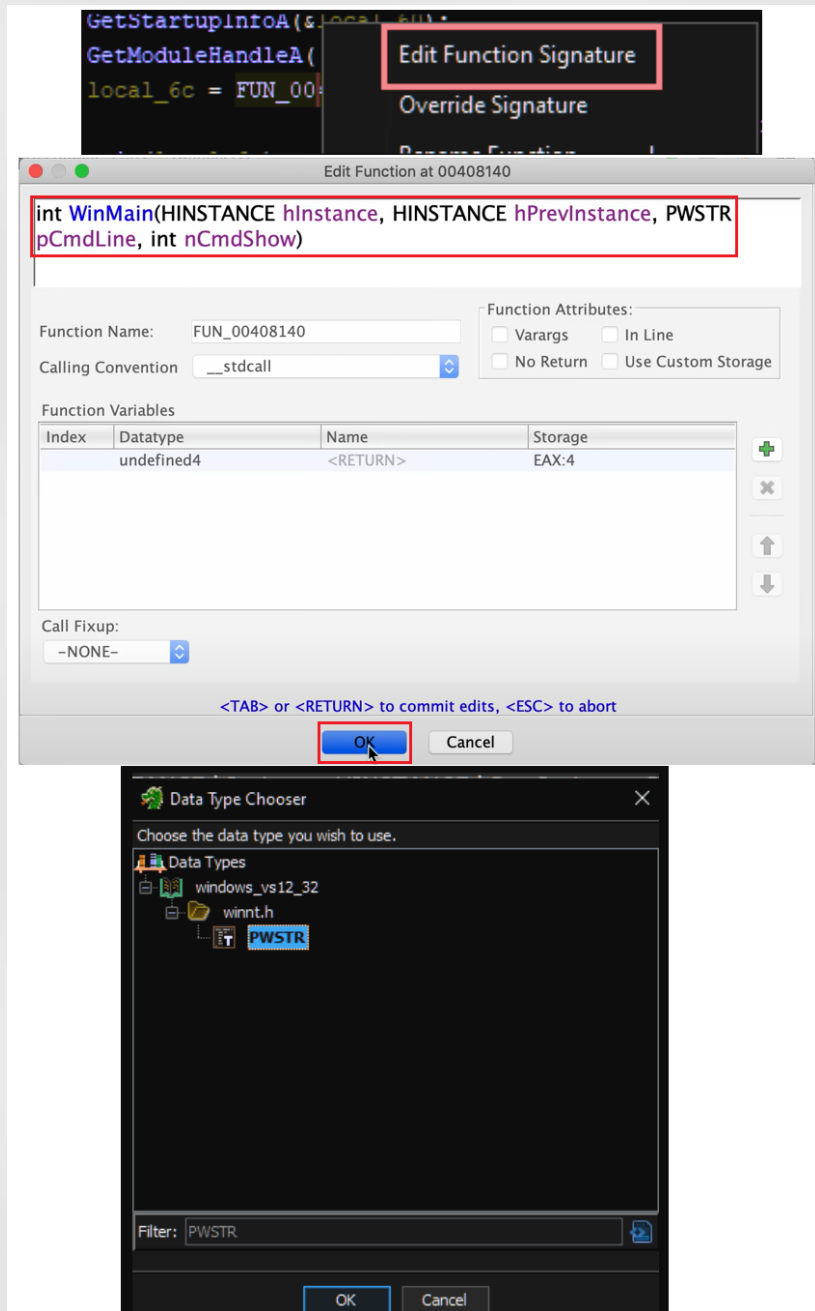
Entry

אם נרד לתחתית הקוד, נוכל לראות קריאה לפונקציה:

```
local_6c = FUN_0040183b();
```

אני מניח שהפונקציה הזאת היא פונקציית winMain.

בוא נגדיר את ה-function signature ל-WinMain כדי שיהיה לנו יותר נוח:



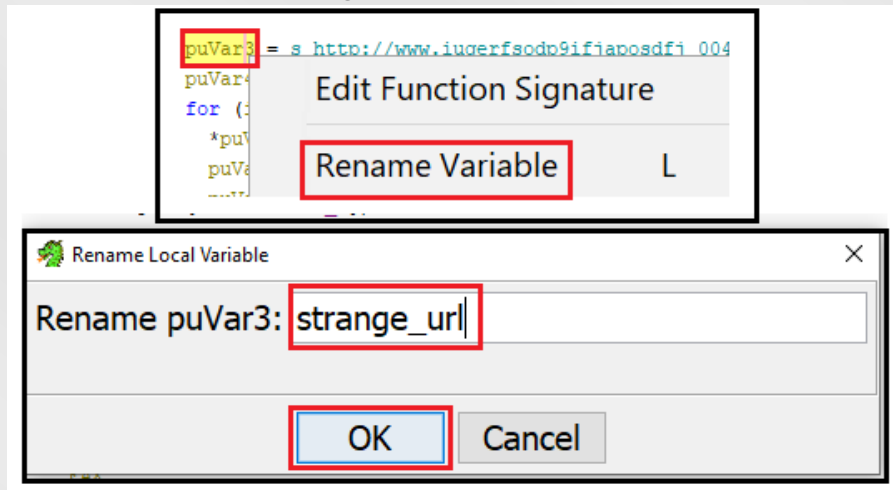
כעת, בוא נחקור את הפונקציה WinMain

WinMain

עכשיו כשעברנו לפונקציה, אפשר לראות שיש לנו משתנה שהערך שלו היא כתובת URL:

```
puVar3 = (undefined4 *)s_http://www.iuqerfsodp9ifjaposd;  
puVar4 = local_50;  
for (iVar2 = 0xe; iVar2 != 0; iVar2 = iVar2 + -1) {  
    *puVar4 = *puVar3;  
}
```

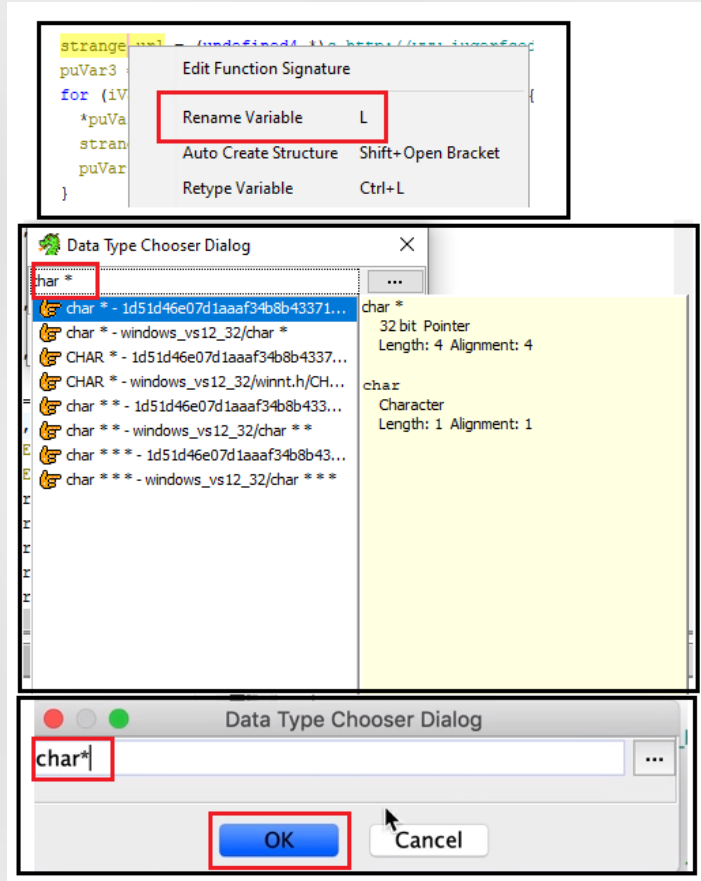
בוא נשנה לו את השם ונקרא לו "strange_url" בצורה הבאה:



1. קליק ימני על המשתנה
2. Rename Variable
3. כתיבת השם החדש (strange_url) ולחיצה על ok

לצורך הנוחות שלי, אני לא אפרט שוב איך משנים שם של משתנה, בכללי, יהיו פה מספר המרות ושיטות לעבור עם Ghidra שאראה פעם אחת, ולאחר מכן לא אחזור עליהם. אז בכל פעם שאבצע פעולות דומות אלו, חשוב להבין מה נעשה.

מכיוון שהבנו שמדובר ב-string, נגדיר אותו תחת הערך char* (ככה מגדירים string ב-C ו-C++). מבצעים את זה בצורה הבאה:



אפשר לראות 2 שורות מתחת לאיפה שעצרנו, שיש קוד של לולאת for. הערך שהלולאה מתחילה איתו הוא 0x0. אפשר לשנות אותו למספר דצימלי רגיל, לכן כדי לעשות זאת, נלחץ:



לאחר הסתכלות על כל הקוד (גם בצד ימין וגם בצד שמאל), ניתן להבין מה קורה כאן:

<pre> LEA EDI=>local_50,[ESP + 0x8] XOR EAX,EAX MOVSD. REP ES:EDI,strange_url = "http://www.iuqerfsodp9ifjapo MOVSB ES:EDI,strange_url=>s_http://www.iuqerfsodp9if... = "http://www.iuqerfsodp9ifjapo MOV dword ptr [ESP + local_17],EAX </pre>	<pre> 18 strange_url = s_http://www.iuqerfsodp9ifjaposdfj_00 19 puVar3 = local_50; 20 for (iVar2 = 14; iVar2 != 0; iVar2 = iVar2 + -1) { 21 *puVar3 = *(undefined4 *)strange_url; 22 strange_url = strange_url + 4; 23 puVar3 = puVar3 + 1; </pre>
--	--

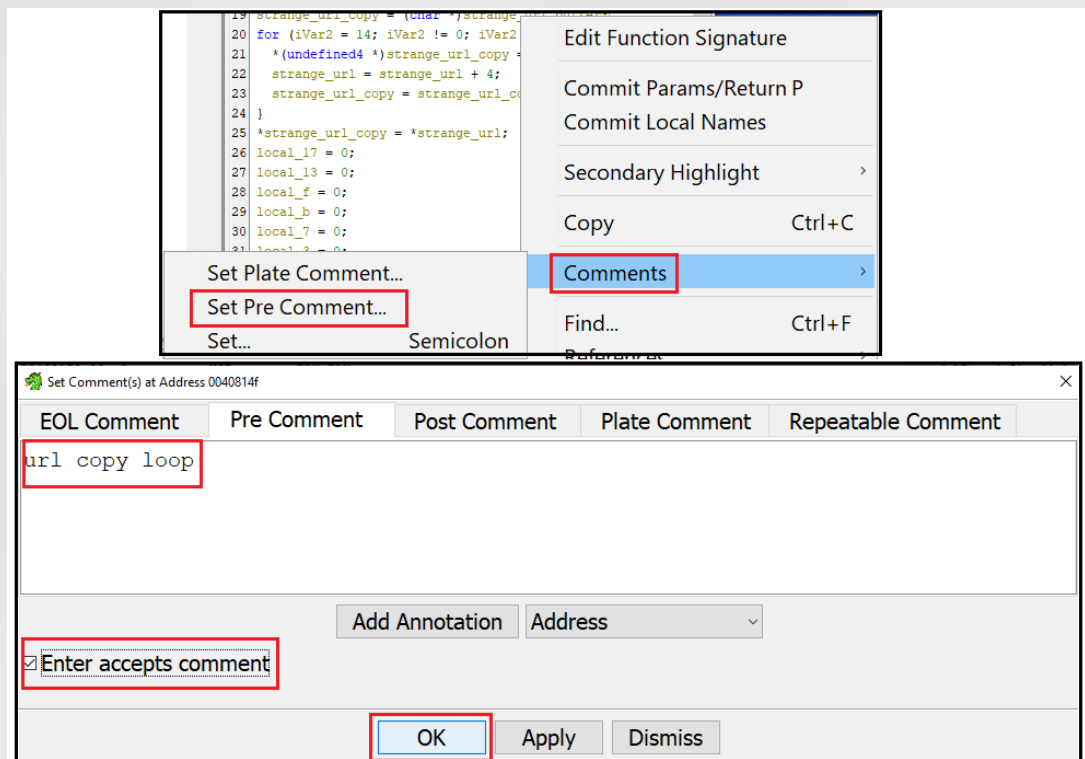
האוגר EDI מקבל ערך של כתובת ב-stack, ויש לזלזל שמעתיקה את כל ה-URL למשתנה puVar3. בכל איטרציה היא:

1. מעבירה 4 בתים מה-URL למשתנה puVar3
2. מקדמת את המשתנים puVar3 ו-Strange_URL.

לצורך הנוחות, נבצע כמה שינויים:

1. אתן למשתנה puVar3 את השם stange_url_copy
2. אגדיר ל-stange_url_copy טיפוס נתונים מסוג char*
3. אתן למשתנה local_50 את השם stange_url_buffer

כדי שלא נשכח מה נעשה כאן, אוסיף תגובה שמתארת את מה שדיברנו עליו בצורה הבאה:



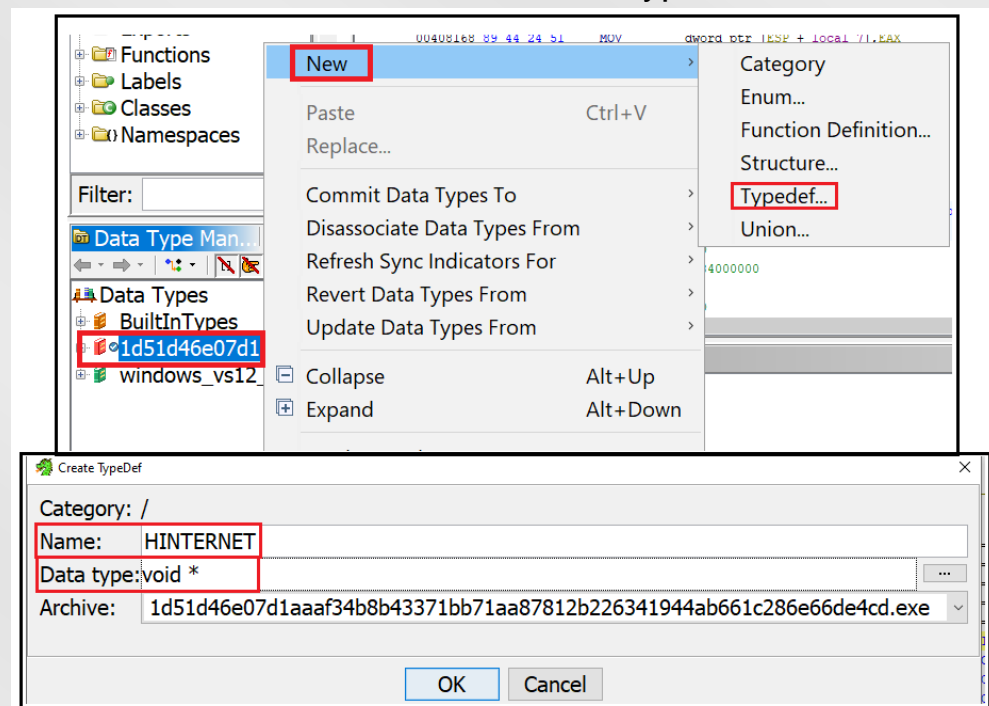
במידה ונרד קצת למטה, אפשר לראות את השימוש בפונקציות InternetOpenA ו-InternetOpenUrlA, לכן, אני אגדיר אותה בעזרת edit function signature:

```
34 uVar1 = InternetOpenA(0,1,0,0,0);
35 InternetOpenUrlA(uVar1,strange_url_buffer,0,0,0x84000000,
36 InternetCloseHandle(uVar1);
37 InternetCloseHandle(0);
38 FUN_00408090();
39 return 0;
```

אבל מסתבר שיש שגיאה:



Ghidra לא מכירה את טיפוס הנתונים HINTERNET שהפונקציה אמורה להחזיר אז מה עושים? ניצור custom type



1. קליק ימני על הספר האדום
2. New > Typedef
3. רישום של השם של סוג המשתנה (HINTERNET) וקביעה של סוג נתונים מסוג void*. למה דווקא סוג זה? כי הוא יכול להצביע לכל סוג נתונים.
4. נלחץ OK

שימו לב: יש צורך למחוק מהקוד באתר של מיקרוסופט את כל הסוגריים ואת הנקודה פסיק בסוף הקוד. לדוגמה, הקוד של InternetOpenA המקורי:

```
HINTERNET InternetOpenA(  
    [in] LPCSTR lpszAgent,  
    [in] DWORD dwAccessType,  
    [in] LPCSTR lpszProxy,  
    [in] LPCSTR lpszProxyBypass,  
    [in] DWORD dwFlags  
);
```

לאחר מחיקה יראה ככה:

```
HINTERNET InternetOpenA(  
    LPCSTR lpszAgent,  
    DWORD dwAccessType,  
    LPCSTR lpszProxy,  
    LPCSTR lpszProxyBypass,  
    DWORD dwFlags  
)
```

לאחר שתיקנו את השגיאה,
נשנה את 2 הפונקציות: InternetOpenA ו-InternetOpenUrlA

עכשיו, בוא נסתכל על הקוד. המסקנות הם:

```

uVar1 = InternetOpenA(0,1);
iVar2 = InternetOpenUrlA(uVar1,&uStack100,0,0,0x84000000,0);
43  if (iVar2 == 0) {
44      InternetCloseHandle(uVar1);
45      InternetCloseHandle(0);
46      FUN_00408090();
47      return 0;
48  }
49  InternetCloseHandle(uVar1);
50  InternetCloseHandle(iVar2);
51  return 0;
52 }

```

- ה-malware בודק אם ניתן לקבל תגובה מכתובת ה-URL המוזרה. אם לא ניתן לקבל תגובה וה-handle מחזיר NULL (ההשוואה לאפס), אנחנו עוברים לפונקציה 00408090_FUN
- במידה ופתיחת ה-URL של ה-C&C הצליחה, בודק אם ה-URL שמצביע עליו פעיל, התוכנה wannacry נסגרת.

השיטה הזאת נקראת kill switch וזאת הייתה הדרך להפסיק את wannacry בכל המחשבים. ה-malware בודק אם הוא מצליח להתחבר לדומיין עם כתובת מג'ונרט. במידה וזה עובד, הוא מפסיק את עצמו. חוקר סייבר בשם Marcus Hutchins קנה את אותו דומיין כמה שעות לאחר תחילת ההתקפה וחיבר את הדומיין לאינטרנט. מה שהוא לא ידע, זה שכך הוא עצר את ההתפשטות של ה-malware.

ממחקר שנעשה בנושא, יש 2 השערות למה שיטה זאת בוצעה:

1. המפתחים השתמשו בשיטה זאת בכוונה, כדי שיהיה להם איזשהו מנגנון עצירת חירום.
 2. שיטה נגד sandbox - ב-sandbox יש שיטות לגרום לתוכנות זדוניות לחשוב שהן מתחברות לשרתים חיצוניים. ואז מה שקורה הוא שכל כתובת שתוכנה הרשומה מנסה להגיע אליה מקבלת תגובה - גם אם הדומיין לא רשום. ככה ה-malware מבין שהוא ב-sandbox ומפסיק את עצמו
- בוא נמשיך לחקור את ה-malware. אשנה את השם של הפונקציה 00408090_FUN ל-wannacry_real_entry.

FUN_00408090

כאשר נכנסים לפונקציה, ניתן לראות בשורה 12 את השימוש בפונקציית `GetModuleFileNameA`. מקריאה באתר של מייקרוסופט והסתכלות על הקוד (לא תמיד רואים את זה בקוד), אפשר להבין שהיא מחזירה את הנתיב ל-malware.

```
12 GetModuleFileNameA((HMODULE) 0x0, s_c:\users\admin\download
13 piVar1 = (int *) __p__argc();
14 if (*piVar1 < 2) {
15     FUN_00407f20();
16     return;
17 }
```

ניתן לראות שיש פונקציה שנקראת:

`__p__argc()`

פונקציה זאת מחזירה את מספר הארגומנטים שנקראו עם התוכנית למשתנה `1piVar`. אפשר לראות שתנאי ה-`if` בודק אם הפונקציה מחזירה פחות מ-2 ארגומנטים. במידה והיא מחזירה פחות, השיטה `20FUN_00407f` מתבצעת. במידה ויש יותר ארגומנטים, מתבצע קוד אחר שכרגע לא ניתן לראות בתמונה. בוא נבצע כמה שינויים בשמות:

1. ניתן לשיטה `20FUN_00407f` את השם `no_argument_handled`
2. ניתן למשתנה `1piVar` את השם `argc` (מתוך השם `arguments counter`).

בוא נראה מה יש בתוך שיטת `no_arguments_handled`

no_arguments_handled

```
undefined4 no_argument_handled(void)

{
    FUN_00407c40();
    FUN_00407ce0();
    return 0;
}
```

ניתן לראות שיש בשורות הראשונות 2 פונקציות.
בוא נראה מה יש בתוך הפונקציה הראשונה FUN_00407c40.

FUN_00407c40

```
9  sprintf(exec_with_args,s_%s_-m_security_00431330,malware_p
10  hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003
11  if (hSCManager != (SC_HANDLE)0x0) {
12      hService = CreateServiceA(hSCManager,s_mssecsvc2.0_00431
13                                  s_Microsoft_Security_Center_(2
14                                  exec_with_args,(LPCSTR)0x0,(LP
15                                  (LPCSTR)0x0);
16      if (hService != (SC_HANDLE)0x0) {
17          StartServiceA(hService,0,(LPCSTR *)0x0);
18          CloseServiceHandle(hService);
19      }
20      CloseServiceHandle(hSCManager);
21      return 0;
22  }
23  return 0;
```

בוא נתחיל לנתח שורה אחרי שורה:

שורה 9

```
sprintf(local_104, s_%s_-m_security_00431330,
s_c:\users\admin\downloads\ef9dfe1_0070f760);
```

אפשר לראות שיש שימוש בפונקציה sprintf. פונקציה זאת היא פונקציה מוכרת ב-C. בוא נראה נפרק את הארגומנטים בפונקציה:
הסבר:

- 104_Local - במשתנה זה ישמר כל הפלט שיורכב
- 00431330_s_%s_-m_security - הטקסט שיוכנס למשתנה 104_Local.
- איפה שרשום %s יוכנס הארגומנט הבא
- ef9dfe1_0070f760S_c:\users\admin\downloads - הנתביב של ה-malware

סה"כ שהפלט שיצא הוא:

```
c:\users\admin\downloads\ef9dfe1_0070f760 -m security
```

אחרי שהבנו מה ישמר ב-104_Local, ניתן לו שם יותר ברור כמו: exec_with_args

שורה 10

```
hSCManager = OpenSCManagerA((LPCSTR)0x0,(LPCSTR)0x0,0xf003f);
```

הסבר:

- hSCManager - משתנה, בהמשך נבין מה הוא מקבל
- OpenSCManagerA - פונקציה שפותחת את ה-Service Control Manager. ה-Service Control Manager הוא חלק ממערכת ההפעלה של Windows, ואחראי על ניהול השירותים שרצים ברקע, וכולל את מאגר כל השירותים במחשב. הפונקציה מחזירה מצביע (handle) לבסיס הנתונים של ה-Service Control Manager. הערכים בהמשך זה ע"פ איך שהפונקציה בנויה:
- (LPCSTR)0x0 - משמש לציון שם המחשב שאליו מתחברים. במקרה זה הוא 'NULL' (אפשר לראות שהוא אפס - x00), מה שאומר שהפונקציה תתחבר למחשב המקומי.
- (LPCSTR)0x0 - משמש לציון שם בסיס הנתונים של מנהל השירותים. גם כאן, ערך זה הוא 'NULL' (אפשר לראות שהוא אפס - x00), מה שאומר שהוא פותח את בסיס הנתונים ברירת המחדל 'SERVICES_ACTIVE_DATABASE'.
- xf003f0 - הקוד שאחראי על ציון ההרשאות הרצויות למנהל השירותים. במקרה זה, ע"פ האתר של מייקרוסופט, הקוד xf003f0 מציין את ההרשאות הבסיסיות.

שורה 11

```
if (hSCManager != (SC_HANDLE)0x0)
```

אם המצביע שונה מערך של אפס (x00), כלומר - אם החיבור הצליח, תיכנס לבלוק בשורה 12, אם החיבור לא הצליח - תעבור לבלוק בשורה 23

שורה 12-15

```
hService =  
CreateServiceA(hSCManager,s_mssecsvc2.0_004312fc,s_Microsoft_Secu  
rity_Center_(2.0)_S_00431308,0xf01ff,0x10,2,1,exec_with_args,(LPCSTR)  
0x0,(LPDWORD)0x0,(LPCSTR)0x0,(LPCSTR)0x0,(LPCSTR)0x0);
```

CreateServiceA היא פונקציה שיוצרת service חדש. הפונקציה מכילה 12 ארגומנטים לכן לא נתחיל להסביר את כולם, אבל באופן כללי, הקוד יוצר שירות חדש בשם Service (2.0) Microsoft Security Center מוג xf01ff0 (שירות מערכת) עם סוג

הפעלה 100x (הפעלה ידנית). שירות זה מפעיל את `exec_with_args` (אפשר לראות שהוא מופיע בארגומנטים של הפונקציה)

אם הפונקציה הצליחה, היא מחזירה מצביע (handle) ל-service שנוצר. במקרה זה, המצביע הוא `hService`.

שורה 16

```
if (hService != (SC_HANDLE)0x0) {
```

אם ה-service נוצר בצורה מוצלחת (ערכו לא אפס - `0x0`), הבלוק בשורה 17 מתבצע.
אם ה-service לא נוצר בהצלחה - הקוד בשורה 20 מתבצע.

שורה 17

```
StartServiceA(hService,0,(LPCSTR *)0x0);
```

הפונקציה `StartServiceA` מדליקה את השירות שנוצר

שורה 18

```
CloseServiceHandle(hService);
```

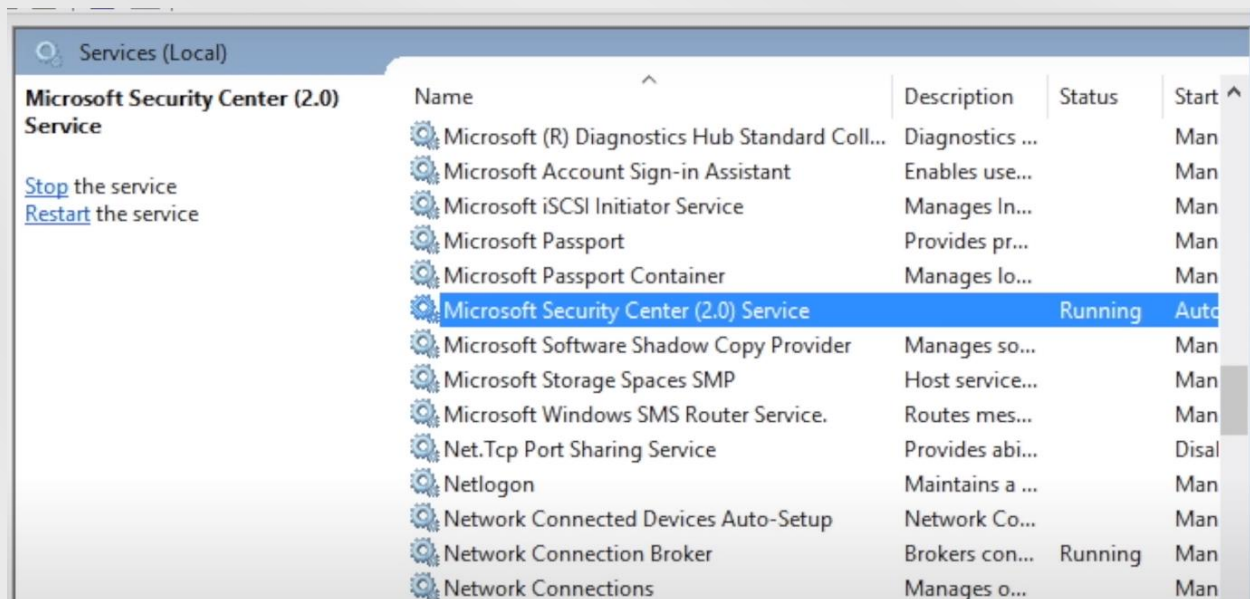
הפונקציה `CloseServiceHandle` סוגרת את המצביע ל-service החדש שנוצר (`hService`).

שורה 19

```
CloseServiceHandle(hSCManager);
```

הפונקציה `CloseServiceHandle` סוגרת את המצביע החדש של בסיס הנתונים של windows service control manager - שהוא `hSCManager`.

הפונקציה `CloseServiceHandle` יכולה לסגור גם מצביע ל-service וגם מצביע לבסיס נתונים של windows service control manager.
כעת שהבנו מה הפונקציה `FUN_00407c40` עושה, ניתן לה את השם "create_wannacry_service". הנה צילום מסך של ה-service החדש שנוצר:



טוב, בוא נחזור אחורה לשיטת no_argument_handled

```
undefined4 no_argument_handled(void)
{
    create_wannacry_service();
    FUN_00407ce0();
    return 0;
}
```

בוא נראה את התוכן של הפונקציה השנייה FUN_00407ce0

FUN_00407ce0

```
32  hModule = GetModuleHandleW((LPCWSTR)&lpModuleName_004313b4);
33  if (hModule != (HMODULE)0x0) {
34      DAT_00431478 = GetProcAddress(hModule,s_CreateProcessA_004313a4);
35      DAT_00431458 = GetProcAddress(hModule,s_CreateFileA_00431398);
36      DAT_00431460 = GetProcAddress(hModule,s_WriteFile_0043138c);
37      DAT_0043144c = GetProcAddress(hModule,s_CloseHandle_00431380);
38      if (((DAT_00431478 != (FARPROC)0x0) && (DAT_00431458 != (FARPROC)
39          (DAT_00431460 != (FARPROC)0x0)) && (DAT_0043144c != (FARPROC)
40          hResInfo = FindResourceA((HMODULE)0x0,(LPCSTR)0x727,&DAT_004313
41      if (hResInfo != (HRSRC)0x0) {
42          hResData = LoadResource((HMODULE)0x0,hResInfo);
43          if (hResData != (HGLOBAL)0x0) {
44              pvVar2 = LockResource(hResData);
45              if (pvVar2 != (LPVOID)0x0) {
46                  DVar3 = SizeofResource((HMODULE)0x0,hResInfo);
47                  if (DVar3 != 0) {
48                      cStack_208 = '\0';
49                      puVar7 = &uStack_207;
50                      for (iVar4 = 0x40; iVar4 != 0; iVar4 = iVar4 + -1) {
51                          *puVar7 = 0;
52                          puVar7 = puVar7 + 1;
53                      }
54                      *(undefined2 *)puVar7 = 0;
55                      *(undefined *)((int)puVar7 + 2) = 0;
56                      cStack_104 = '\0';
57                      puVar7 = &uStack_103;
```

עד שורה 31 מדובר בהגדרות של משתנים, לכן לא הוספתי את זה למסמך. בוא נראה מה קורה בהמשך הקוד:

שורה 32

`hModule = GetModuleHandleW((LPCWSTR)&lpModuleName_004313b4);`

אפשר להבין שיש שימוש בפונקציה `GetModuleHandleW`, כלומר שהפונקצייה מחזירה למשתנה `hModule` את ה-handle של המודול בסוגריים. אך לא ברור מה המודול בסוגריים, בוא נסתכל על האסמבלי של הפונקציה:

`CALL dword ptr [->KERNEL32.DLL: GetModuleHandleW`

עכשיו זה קצת יותר ברור שהמודול הוא kernel32.dll, שהוא dll שמאפשר תקשורת עם kernel-ה.

שורה 33

```
if (hModule != (HMODULE)0x0)
```

אם הערך של ה-handle של hModule לא שווה לאפס (כלומר הפונקציה עבדה), תיכנס לבלוק שנמצא שורה לאחר מכן.

שורה 34

```
DAT_00431478 = GetProcAddress(hModule,s_CreateProcessA_004313a4);
```

יש שימוש בפונקציה GetProcAddress כדי לקבל את הכתובת של הפונקציה CreateProcessA (שיוצרת process חדש). פונקציה זאת נמצא בתוך ה-dll של kernel32.dll. הפונקציה מחזירה מצביע לכתובת של הפונקציה שביקשה.

שורה 35

```
DAT_00431458 = GetProcAddress(hModule,s_CreateFileA_00431398);
```

יש שימוש בפונקציה GetProcAddress כדי לקבל את הכתובת של הפונקציה CreateFileA (שפותחת או יוצרת קובץ חדש). פונקציה זאת נמצא בתוך ה-dll של kernel32.dll. הפונקציה מחזירה מצביע לכתובת של הפונקציה שביקשה.

שורה 36

```
DAT_00431460 = GetProcAddress(hModule,s_WriteFile_0043138c);
```

יש שימוש בפונקציה GetProcAddress כדי לקבל את הכתובת של הפונקציה WriteFile (שרושמת על קובץ). פונקציה זאת נמצא בתוך ה-dll של kernel32.dll. הפונקציה מחזירה מצביע לכתובת של הפונקציה שביקשה.

שורה 37

```
DAT_0043144c = GetProcAddress(hModule,s_CloseHandle_00431380);
```

יש שימוש בפונקציה GetProcAddress כדי לקבל את הכתובת של הפונקציה CloseHandle (שסוגרת handle). פונקציה זאת נמצא בתוך ה-dll של kernel32.dll. הפונקציה מחזירה מצביע לכתובת של הפונקציה שביקשה.

לצורך הנוחות, אשנה את כל השמות של המצביעים (pointers) בשורות 34-37 לשמות של הפונקציות שהם מצביעות עליהם:

```
34 CreateProcessA = GetProcAddress(hModule,s_CreateProcessA_0043137c);
35 CreateFileA = GetProcAddress(hModule,s_CreateFileA_00431380);
36 WriteFile = GetProcAddress(hModule,s_WriteFile_0043138c);
37 CloseHandle = GetProcAddress(hModule,s_CloseHandle_00431394);
```

בוא נמשיך עם שאר הקוד:

```
38 if (((CreateProcessA != (FARPROC)0x0) && (CreateFileA != (FARPROC)0x0) && (WriteFile != (FARPROC)0x0) && (CloseHandle != (FARPROC)0x0))) {
39     hResInfo = FindResourceA((HMODULE)0x0,(LPCSTR)0x727,&DAT_0043137c);
40     if (hResInfo != (HRSRC)0x0) {
41         hResData = LoadResource((HMODULE)0x0,hResInfo);
42         if (hResData != (HGLOBAL)0x0) {
43             pvVar2 = LockResource(hResData);
44             if (pvVar2 != (LPVOID)0x0) {
45                 DVar3 = SizeofResource((HMODULE)0x0,hResInfo);
46                 if (DVar3 != 0) {
47                     cStack_208 = '\0';
48                     puVar7 = &uStack_207;
49                     for (iVar4 = 0x40; iVar4 != 0; iVar4 = iVar4 + 1) {
50                         *puVar7 = 0;
51                         puVar7 = puVar7 + 1;
52                     }
53                 }
54             }
55         }
56     }
57 }
```

שורה 38-39

```
if (((((CreateProcessA != (FARPROC)0x0) && (CreateFileA != (FARPROC)0x0) && (WriteFile != (FARPROC)0x0) && (CloseHandle != (FARPROC)0x0))) {
```

אם כל ה-pointers החדשים שנוצרו לא שווים לאפס (כלומר המצביעים תקינים), תמשיך לקוד בבלוק שנמצא שורה אחת מתחת.

שורה 40

```
hResInfo = FindResourceA((HMODULE)0x0,(LPCSTR)0x727,&DAT_0043137c);
```

ה-malware משתמש בפונקציה FindResourceA שמחפשת resource שכתובתו היא 727x0 (בדצימלי זה 1831). Resource הוא מידע שהכרחי להפעלת התוכנה, לדוגמה: תמונות, קבצי טקסט, קבצי אודיו, קבצי וידאו ועוד.

הפונקציה מחזירה handle לבלוק המידע של המשאב (בלוק המידע מכיל מידע על המשאב, כגון את שמו, את סוג הנתונים שלו ואת גודל הנתונים שלו). לצורך הנוחות אשנה את השם של hResInfo ל-res1831_info

שורה 41

```
if (resource1831_info != (HRSRC)0x0)
```

אם ה-handle למידע של המשאב לא אפס (כלומר הפונקציה בשורה 40 הצליחה), תמשיך לקוד שנמצא בשורה הבאה.

שורה 42

```
hResData = LoadResource((HMODULE)0x0,res1831_info);
```

שימוש בפונקציה LoadResource שטוענת את המשאב לזיכרון הארגומנט הראשון. ארגומנטים:

1. ה-handle למודול הרלוונטי, אם הערך הוא NULL או אפס - נעשה שימוש ביישום הנוכחי

2. Handle לבלוק המידע של המשאב (res1831_info)

הפונקציה מחזירה handle למשאב בזיכרון, לכן נקרא ל-hResData בשם "res1831_handle"

שורה 43

```
if (res1831_handle != (HGLOBAL)0x0)
```

אם ה-handle ל-resource לא שווה אפס (כלומר הטעינה הצליחה), תעבור לשורה הבאה

שורה 44

```
pvVar2 = LockResource(res1831_handle);
```

נועלת את המשאב בזיכרון. הפונקציה מחזירה מצביע (pointer) לבית (byte) הראשון של ה-resource בזיכרון. לצורך הנוחות נתן למשתנה 2pvVar את השם res1831_locked.

שורה 45

if (res1831_locked != (LPVOID)0x0)

אם הערך של res1831_locked לא שווה לאפס (כלומר הפעולה בשורה הקודמת הצליחה), תיכנס לבלוק בשורה הבאה

שורה 46

DVar2 = SizeofResource((HMODULE)0x0,res1831_info);

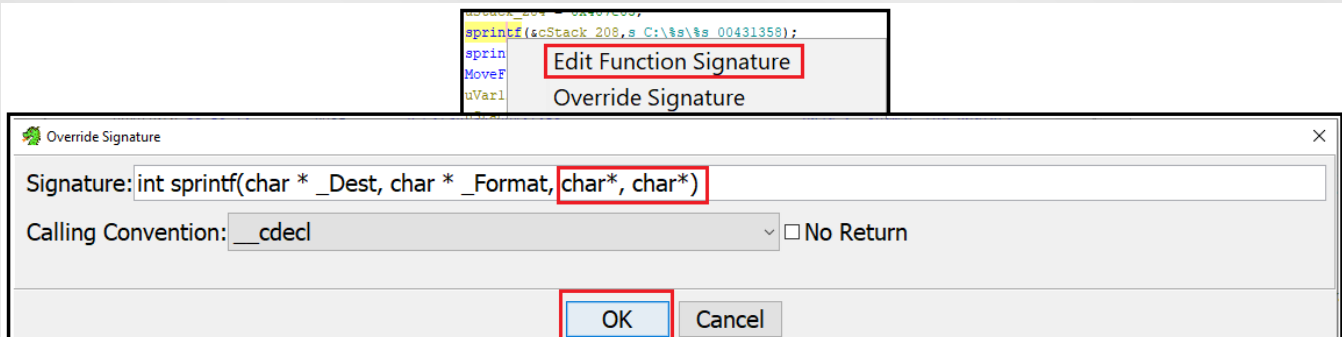
נעשה שימוש בפונקציה SizeofResource שמחזירה את גודל המשאב בבתים - bytes.
לצורך הנוחות נתן למשתנה DVar2 את השם res1831_size.

```
48     cStack_208 = '\\0';
49     puVar5 = &uStack_207;
50     for (iVar2 = 0x40; iVar2 != 0; iVar2 = iVar2 + -1)
51         *puVar5 = 0;
52         puVar5 = puVar5 + 1;
53     }
54     *(undefined2 *)puVar5 = 0;
55     *(undefined *)((int)puVar5 + 2) = 0;
56     cStack_104 = '\\0';
57     puVar5 = &uStack_103;
58     for (iVar2 = 0x40; iVar2 != 0; iVar2 = iVar2 + -1)
59         *puVar5 = 0;
60         puVar5 = puVar5 + 1;
61     }
```

ב-2 הפסקאות הבאות יש 2 לולאות שמאפסות את הערכים של 2 משתנים שמצביעים למקומות ב-stack (שורה 49 ושורה 57)

```
62     *(undefined2 *)puVar5 = 0;
63     *(undefined *)((int)puVar5 + 2) = 0;
64     uStack_284 = 0x407e03;
65     sprintf(&cStack_208,s_C:\\%s\\%s_00431358);
66     sprintf(&cStack_104,s_C:\\%s\\qeriuwjhrf_00431344);
67     MoveFileExA(&cStack_208,&cStack_104,1);
68     uVar12 = 2;
69     uStack_284 = 0;
70     pcStack_28c = &cStack_208;
71     uStack_288 = 0x40000000;
72     uStack_290 = 0x407e49;
73     iVar2 = (*CreateFileA)();
```

ניתן לראות שבשורות 65 ו-66 יש קריאות לפונקציה sprintf, אך הפונקציה אמורה לקבל 2 משתנים וזה נראה כאילו הארגומנטים האלו חסרים. למי שלא מכיר את הפונקציה, המשתנים אמורים להיכנס איפה שיש את הסימן %s. כדי לפתור את זה נצטרך לערוך את הפונקציה באמצעות שימוש ב-override signature:



לאחר העריכה, ניתן לראות שהפונקציה השתנתה:

```

63     sprintf(&tasksche_path,s_C:\%s\%s_00431358,s_WINDOWS_00431364,s_tasksche.exe_0043136c)
64     ;
65     sprintf(&qeriuwjhrf_path,s_C:\%s\qeriuwjhrf_00431344,s_WINDOWS_00431364);
66     MoveFileExA(&tasksche_path,&qeriuwjhrf_path,1);
67     CreateFileHandle =
68         (*CreateFileA) (&tasksche_path,0x40000000,0,(LPSECURITY_ATTRIBUTES)0x0,2,4,
69             (HANDLE)0x0);
70     if (CreateFileHandle != (HANDLE)0xffffffff) {
71         (*WriteFile) (CreateFileHandle,res183Locked.hProcess,res183l_size,
72             (LPDWORD)&res183Locked,(LPOVERLAPPED)0x0);
73         (*CloseHandle) (CreateFileHandle);
74         res183Locked.hThread = (HANDLE)0x0;
75         res183Locked.dwProcessId = 0;
76         res183Locked.dwThreadId = 0;
77         ppCVar7 = &_Stack_250.lpReserved;
78         for (iVar3 = 0x10; iVar3 != 0; iVar3 = iVar3 + -1) {
79             *ppCVar7 = (LPSTR)0x0;
80             ppCVar7 = ppCVar7 + 1;
81         }
82         uVar4 = 0xffffffff;
83         ppuVar8 = &PTR_DAT_00431340;
84         do {
85             ppuVar9 = ppuVar8;
86             if (uVar4 == 0) break;
87             uVar4 = uVar4 - 1;
88             ppuVar9 = (undefined **) ((int)ppuVar8 + 1);
89             cVar1 = *(char *)ppuVar8;
            ppuVar8 = ppuVar9;
        } while (cVar1 != '\0');

```

```
sprintf(&cStack_208,s_C:\%s\%s_00431358,s_WINDOWS_00431364,s_tasksche  
e.exe_0043136c);  
sprintf(&cStack_104,s_C:\%s\qeriuwjhrf_00431344,s_WINDOWS_00431364);
```

ב-2 שורות אלו מתבצעת אותה פעולה:

1. המשתנה cStack_208 מאחסן בתוכו את המחרוזת הבאה:

C:\WINDOWS\tasksche.exe

2. המשתנה cStack_104 מאחסן בתוכו את המחרוזת הבאה:

C:\WINDOWS\qeriuwjhrf

לצורך הנוחות נשנה להם את השמות:

1. למשתנה cStack_208 נקרא "tasksche_path".

2. למשתנה cStack_104 נקרא "qeriuwjhrf_path".

הפונקציה sprintf שומרת את הערך של המשתנים בתוך ה-buffer במחשב

שורה 68

```
MoveFileExA(&tasksche_path,&qeriuwjhrf_path,1);
```

יש שימוש בפונקציה MoveFileExA שמעבירה את המחרוזת ב-tasksche_path למשתנה qeriuwjhrf_path ודורסת את התוכן שהיה בתוכו

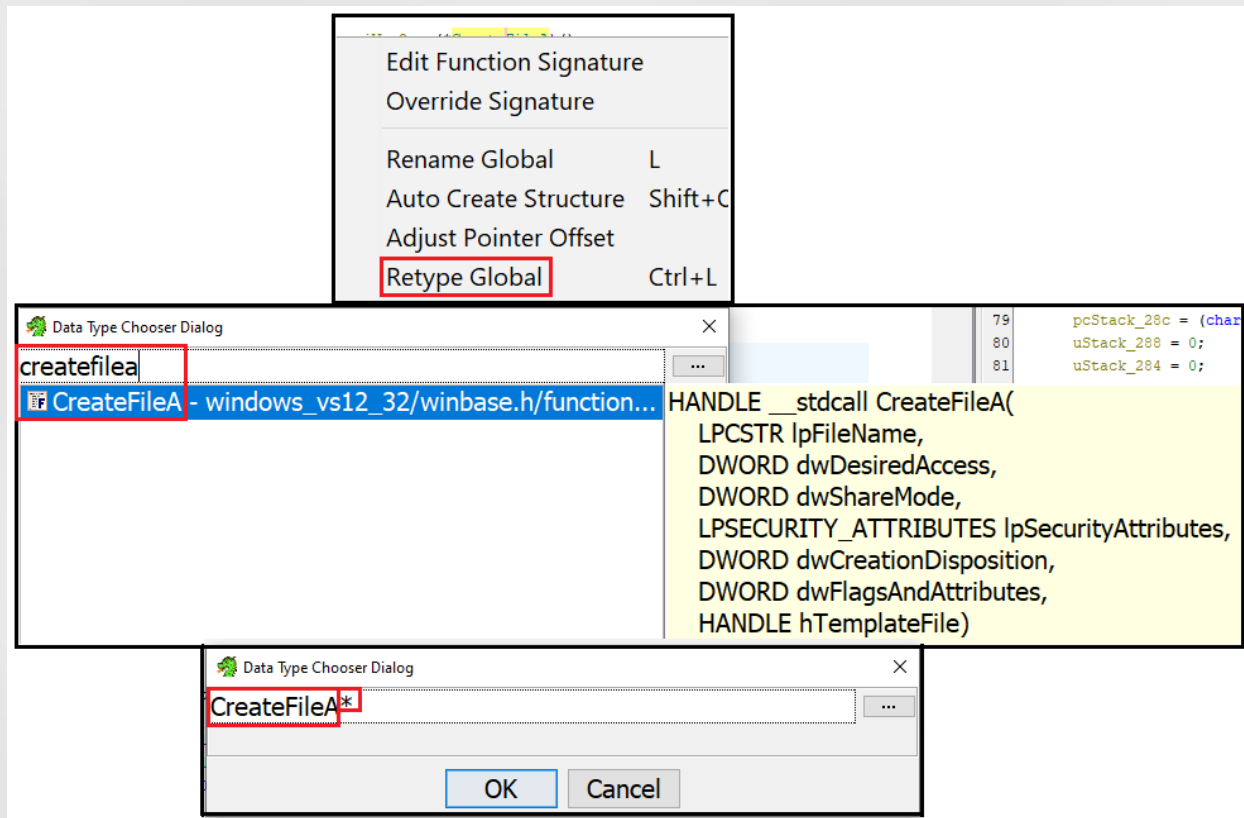
שורה 67-69

השורה היא:

```
iVar2 = (*CreateFileA());
```

ואפשר לראות שהפונקציה לא מוגדרת כמו שצריך, CreateFileA אמורה לקבל 7 פרמטרים.

בוא נגדיר אותה כמו שצריך בעזרת Global Retype:



הנה התוצאה של השינוי:

```

66     sprintf(&qeriuwjhrf_path,s_C:\\%s\\qeriuwjhrf_00431344,s_WI
67     MoveFileExA(&tasksche_path,&qeriuwjhrf_path,1);
68     pvVar2 = (*CreateFileA)(&tasksche_path,0x40000000,0,(LPSE

```

אפשר לראות שהקוד מקבל 6 פרמטרים (כמו שהוא צריך לקבל):

```

pvVar2 =
(*CreateFileA)(&tasksche_path,0x40000000,0,(LPSECURITY_ATTRIBUTE
S)0x0,2,4,(HANDLE)0x0);

```

לא נתחיל להתעכב עכשיו על כל 7 הפרמטרים, אבל בקצרה - המשתנה `tasksche_path` נפתח עם גישה לקריאה וכתיבה, והפונקציה מחזירה `handle` לקובץ.

לצורך הנוחות, בוא ניתן למשתנה `2pvVar` את השם "CreateFileHandle"

בוא נראה את המשך הקוד:


```

71 if (CreateFileHandle != (HANDLE)0xffffffff) {
72     (*WriteFile) (CreateFileHandle,apvStack_260[0],res1831_size
73     (*CloseHandle) (CreateFileHandle);
74     ppvVar6 = apvStack_260;
75     for (iVar2 = 0x10; iVar2 != 0; iVar2 = iVar2 + -1) {
76         *ppvVar6 = (LPVOID)0x0;
77         ppvVar6 = ppvVar6 + 1;
78     }
79     uVar3 = 0xffffffff;
80     ppuVar7 = &PTR_DAT_00431340;

```

שורה 71

אם CreateFileHandle תקין, תמשיך לשורה הבאה

שורה 72

גם כאן השורה הייתה:

```
(*WriteFile)(CreateFileHandle,aRes1831Locked[0],res1831_size,apvStack_260);
```

ואחרי שימוש ב-global retype היא:

```
(*WriteFile)(CreateFileHandle,Res1831Locked,res1831_size,(LPDWORD)
&pvStack_260,(LPOVERLAPPED)0x0);
```

מתבצע רישום של התוכן של resource 1831 לתוך הקובץ tasksche.exe.
ובשורה 73 ה-handle לקובץ tasksche.exe בשם CreateFileHandle נסגר.

ניתן להשתמש בכלי wrestool שעובד על linux כדי לחקור עוד את ה-resource 1831.
לצורך כך, אלך למכונת ה-kali linux שלי ואוריד את הכלי ע"י שימוש בפקודה הבאה:
sudo apt-get install icoutils

```

(kali@kali)-[~/Downloads]
$ wrestool 1d51d46e07d1aaaf34b8b43371bb71aa87812b226341944ab661c286e66de4cd.exe
--type='R' --name=1831 --language=1033 [offset=0x3100a4 size=3514368] [permission denied]
--type=16 --name=1 --language=1033 [type=version offset=0x66a0a4 size=944] are you root?

(kali@kali)-[~/Downloads]
$ wrestool --name=1831 -R -x 1d51d46e07d1aaaf34b8b43371bb71aa87812b226341944ab661c286e66de4cd.exe > 1831.bin
Frontend lock (/var/lib/dpkg/lock) frontend lock (/var/lib/dpkg/lock), are you root?

(kali@kali)-[~/Downloads]
$ file 1831.bin
1831.bin: PE32 executable (GUI) Intel 80386, for MS Windows, 4 sections

```

בפקודה הראשונה

ניתן לראות שהכלי זיהה בשורה של ה-resources (השורה הראשונה) 2 resources וביניהם ה-resource שאנחנו מעוניינים לחקור.

בפקודה השנייה

ביצעתי extract ל-resource

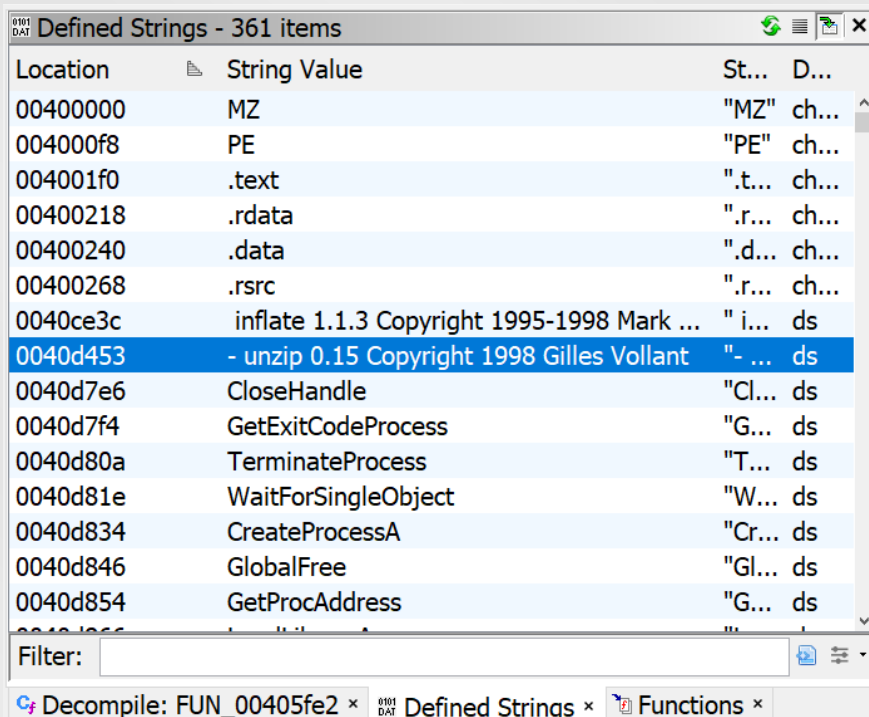
בפקודה השלישית

העלתי מידע על אותו קובץ, כנראה מדובר ב-PE EXE, לכן נעתיק אותו בחזרה למכונת windows שלנו ונחקור אותו עם ghidra.

בוא נעבור לקובץ bin החדש שחילצנו:

1831.bin

בוא נסתכל על ה-strings ע"י לחיצה על הקטגוריה "Window" ואז Defined Strings.



Location	String Value	St...	D...
00400000	MZ	"MZ"	ch...
004000f8	PE	"PE"	ch...
004001f0	.text	".t..."	ch...
00400218	.rdata	".r..."	ch...
00400240	.data	".d..."	ch...
00400268	.rsrc	".r..."	ch...
0040ce3c	inflate 1.1.3 Copyright 1995-1998 Mark ...	"i..."	ds
0040d453	- unzip 0.15 Copyright 1998 Gilles Vollant	"- ..."	ds
0040d7e6	CloseHandle	"Cl..."	ds
0040d7f4	GetExitCodeProcess	"G..."	ds
0040d80a	TerminateProcess	"T..."	ds
0040d81e	WaitForSingleObject	"W..."	ds
0040d834	CreateProcessA	"Cr..."	ds
0040d846	GlobalFree	"Gl..."	ds
0040d854	GetProcAddress	"G..."	ds

אפשר לראות שמצאנו string לא ברור שמכיל את המילים "unzip". נלך לצד האסמבלי שלו וניתן לראות שיש פונקציה ליד ה-string:

```
s__unzip_0.15_Copyright_1998_Gille_0040d454 XREF[0,1]: FUN_00405fe2:00405ff9(R)
0040d453 2d 20 75 ds "- unzip 0.15 Copyright 1998 Gilles Vollant "
```

נחלץ עלייה דאבל קליק ונשנה לה את השם ל-"unzip_something" בנוסף לזה, אפשר לראות ב-strings את הסיומות של הקבצים שעוברים הצפנה:

Location	String Value
0040e5f4	.ps1
0040e600	.vbs
0040e61c	.dip
0040e628	.dch
0040e634	.sch
0040e640	.brd
0040e64c	.jsp
0040e658	.php
0040e664	.asp
0040e678	.java
0040e684	.jar
0040e690	.class
0040e6a8	.mp3
0040e6b4	.wav
0040e6c0	.swf

ו-3 כתובות bitcoin:

0040f440	115p7UMMngoj1pMvkpHijcRdfJNXj6LrLn
0040f464	12t9YDPgwueZ9NyMgw519p7AA8isjr6SMw
0040f488	13AM4VW2dhxYgXeQepoHkHSQuy6NgaE...
0040f4b4	Global\MsWinZonesCacheCounterMutexA

בוא נחזור לתווית הפונקציות. בתווית זאת ניתן לראות פונקציה שנקראת entry. וכמו בתחילת הניתוח שעשינו מקודם, נכנס אל הפונקציה, נרד לשורות התחתונות ביותר ונערוך את ה-function signature ל-WinMain.

כעת, בוא נכנס ל-WinMain ונראה מה יש שם:

WinMain

```
26  *(undefined2 *)puVar8 = 0;  
27  *(undefined *) ((int)puVar8 + 2) = 0;  
28  GetModuleFileNameA((HMODULE) 0x0, &local_210, 0x208);  
29  FUN_00401225(0x40f8ac);  
30  piVar2 = (int *) __p__argc();  
31  if (*piVar2 == 2) {
```

תחילת הקוד מכיל הגדרת משתנים ולולאות, לא משהו מיוחד.
אפשר לראות שיש באחת השורות העליונות קריאה לפונקציה לא ברורה

שורה 28

יש שימוש בפונקציה GetModuleFileNameA שמחזירה לפרמטר השני (local_210) את הנתביב המלא של ה-process הנוכחי שרץ (של ה-malware).
נקרא למשתנה local_210 בשם "filename"

שורה 29

יש פונקציה לא ברורה בשם FUN_00401225, בוא נכנס לפונקציה לחקור אותה.

FUN_00401225

```
26  GetComputerNameW(&computername,&computername_size);
27  local_8 = 0;
28  _Seed = 1;
29  computername_length = wcslen(&computername);
30  if (computername_length != 0) {
31      computername_ptr = &computername;
32      do {
33          _Seed = _Seed * (ushort)*computername_ptr;
34          local_8 = local_8 + 1;
35          computername_ptr = computername_ptr + 1;
36          computername_length = wcslen(&computername);
37      } while (local_8 < computername_length);
38  }
39  srand(_Seed);
40  random_num1 = rand();
41  iVar3 = 0;
42  iVar1 = random_num1 % 8 + 8;
43  if (0 < iVar1) {
44      do {
45          random_num2 = rand();
46          *(char *) (iVar3 + param_1) = (char) (random_num2 % 0x1a) + 'a';
```

עד שורה 26 מדובר רק בהגדרה של משתנים, לכן העתקתי את הקוד החל משורה 26.

שורה 26 - יש שימוש בפונקציה GetComputerNameW. הפרמטר הראשון (local_19c) מקבל את שם המחשב, והפרמטר השני (local_c) מקבל את מספר התווים שמרכיבים את שם המחשב. שיניתי את הפרמטר הראשון והשני לשמות הבאים:

computername

computername_size

שורה 29 - ניתן לראות שימוש במשתנה computername עם הפונקציה wcslen, שהיא פונקציה שמחשבת אורך של מחרוזת, ונשמר תחת המשתנה 1sVar. שיניתי את 1sVar לשם הבא: computername_length

שורה 30 - 31 - יש בדיקה שאורך של שם המחשב לא אפס.
אם לא, הקוד מייצר pointer לשם של המחשב.

שורה 32-36 - יש בבלוק ה-do הזה מספר פעולות שמתבצעות כל עוד התנאי של ה-while בשורה 37 קורה (כאשר 8_local קטן מאורך המחשב). סדרת הפעולות הם:

שורה 33 - הכפלה של ה-hash של Seed_ במשתנה computernam_ptr ועדכון של seed איתו
שורה 34 - העלאה ב-1 של 8_local
שורה 35 - עדכון המצביע computernam_ptr כך שהוא יצביע על התו הבא של שם המחשב
שורה 36 - ניתן לראות שימוש במשתנה computernam עם הפונקציה wcslen, שהיא פונקציה שמחשבת אורך של מחרוזת, ונשמר תחת המשתנה computernam_length

בסוף התהליך, נוצר בתוך seed סוג של hash שעורבל עם כל תו של שם המחשב

שורה 39 - מבססת את פונקציית rand בהתבסס על הערך seed שנמצא בפנים
שורה 40 - יוצרים איזשהו מספר אקראי שנשמר במשתנה 1random_num

```
45     random_num2 = rand();
46     param_1[iVar3] = (char)(random_num2 % 26) + 'a';
47     iVar3 = iVar3 + 1;
48 } while (iVar3 < iVar1);
49 }
50 for (; iVar3 < random_num1 % 8 + 11; iVar3 = iVar3 + 1) {
51     iVar1 = rand();
52     param_1[iVar3] = (char)(iVar1 % 10) + '0';
53 }
54 param_1[iVar3] = '\\0';
55 return;
```

שורה 44-48 - מתבצעות מספר פעולות כל עוד התנאי ב-while אמת והוא ש-3iVar קטן מ-1iVar. הפעולות הם:

- שורה 45 - יצירת מספר רנדומלי בשם 2random_num
כל איטרציה זה מספר רנדומלי אחר
- שורה 46 - עדכון המשתנה unknown בתו ה-3iVar.
כרגע 3iVar הוא אפס, אבל כל פעם זה יגדל בתו אחד כי בשורה הבאה ערכו של 3iVar גדל ב-1.
Unknown יהיה תו של אות בעזרת הפעולה החישובית הבאה:
`unknown[iVar3] = (char)(random_num2 % 26) + 'a';`
- הגדלה של 3iVar ב-1

זהו, ככה זה רץ עד שעוברים על חלק מהתווים של המשתנה unknown

שורה 50-53 רצות בלולאת for על המשך התווים של המשתנה unknown, רק שהפעם, המשך הערך של המשתנה unknown יהיו מספרים ולא תווים כמו בפעם הקודמת.
כל איטרציה יש rand ל-1iVar ואז מתבצעת ההשמה באופן הלוגי הבא:
`unknown[iVar3] = (char)(iVar1 % 10) + '0';`

שורה 54 - שמה למשתנה unknown את הערך שלש ואפס 0 בתו האחרון.

לסיכום

הפונקציה 00401225_FUN מייצרת מחרוזת אקראית שמורכבת מתווים ומספרים, לדוגמה:
1234567890abcdefghijklmnopqrstuvwxyz

לצורך נוחות:

1. אתן ל-param_1 את השם randomstring_output
2. אקרא לפונקציה 00401225_FUN בשם randomstring_generator

בוא נחזור לפונקציית WinMian

WinMain

```
28 randomstring_generator((char *)&randomstring);
29 piVar2 = (int *)__p__argc();
30 if (*piVar2 == 2) {
31     _Str2 = &_Str2_0040f538;
32     piVar2 = (int *)__p__argv();
33     iVar7 = strcmp(*(char **) (*piVar2 + 4), (char *)_Str2);
34     if ((iVar7 == 0) && (bVar1 = FUN_00401b5f((wchar_t *)0x0), CONCAT31(extraout_var,bVar1)
35         CopyFileA(filename,s_tasksche.exe_0040f4d8,0);
36         DVar3 = GetFileAttributesA(s_tasksche.exe_0040f4d8);
37         if ((DVar3 != 0xffffffff) && (iVar7 = FUN_00401f5d(), iVar7 != 0)) {
38             return 0;
39         }
40     }
41 }
42 pcVar4 = strrchr(filename,0x5c);
43 if (pcVar4 != (char *)0x0) {
44     pcVar4 = strrchr(filename,0x5c);
45     *pcVar4 = '\\0';
46 }
47 SetCurrentDirectoryA(filename);
48 FUN_004010fd(1);
49 FUN_00401dab((HMODULE)0x0);
50 FUN_00401e9e();
51 FUN_00401064(s_attrib+_h_._0040f520,0,(LPDWORD)0x0);
52 FUN_00401064(s_icaccls._/grant_Everyone:F/T/C_0040f4fc,0,(LPDWORD)0x0);
53 iVar7 = FUN_0040170a();
54 if (iVar7 != 0) {
55     FUN_004012fd();
```

לאחר שהבנו מה עושה הפונקציה, שיניתי בשורה 28 את שם הפרמטר x40f8ac0 ל-
randomstring.

בשורה 28, מספר הארגומנטים שנקראו עם הפעלת התוכנית נשמרים תחת המשתנה
2piVar (לצורך הנוחות שיניתי את השם שלו ל-argc)
במידה והוא שווה ל-2, יש את הערך _Str2_0040f538 שנכנסת למשתנה 2Str_. כדי
להבין מה זה הערך _Str2_0040f538, נלחץ דאבל קליק עליו ונסתכל על האסמבלי.
ניתן לראות שהערך הוא string:

```
                                _Str2_0040f538
0040f538 2f 69 00 00      char *      DAT_0000692f
```

לאחר שימוש ב-clear code bytes ניתן לראות שמדובר בתו סלש ובתו ם:

_Str2_0040f538		
??	2Fh	/
??	69h	i
??	00h	
??	00h	

אבצע שימוש ב-terminatedCString כדי להגדיר אותו מחדש, וקיבלנו את השורה הבאה:

```
pcVar7 = s_/i_0040f538;
```

לאחר שהבנו שבמשתנה pcVar7 יש את הערך סלש ואז האות i, אשנה לו את השם ל-i_/s

שורה 32 - יש השמה של המערך argv למשתנה argc, לכן לצורך נוחות אקרא לו argc_or_argv - עכשיו ערכו הוא argv.

שורה 33 - מתבצעת השוואה עם הפונקציה strcmp (קיצור של string compare, פונקציה שמשווה בין 2 ערכים) בין המשתנה argc_or_argv לבין הערך i/. אתן למשתנה שמקבל את ההשוואה (5iVar) את השם arg4_cmp. במידה והם באמת שווים, הערך שלו יהיה אפס.
שורה 34 - כאן יש 2 תנאים:

1. במידה והערך של arg4_cmp הוא אפס, כלומר ההשוואה באמת דומה
2. במידה והמשתנה 1bVar לא אפס, כלומר, במידה והפונקציה FUN_00401b5f הצליחה.

בוא נראה מה הפונקציה FUN_00401b5f עושה

FUN_00401b5f

```
37 MultiByteToWideChar(0,0,(LPCSTR)&randomstring,-1,&local_cc,99);
38 GetWindowsDirectoryW(&local_4dc,0x104);
39 local_4da._2_2_ = 0;
40 swprintf(&local_2d4,0x40f40c,&local_4dc);
41 DVar1 = GetFileAttributesW(&local_2d4);
42 if ((DVar1 == 0xffffffff) || (iVar3 = FUN_00401af6(&local_2d4,&local_cc,param_1), iVar3 ==
43     swprintf(&local_2d4,0x40f3f8,&local_4dc);
44     iVar3 = FUN_00401af6(&local_2d4,&local_cc,param_1);
45     if ((iVar3 == 0) && (iVar3 = FUN_00401af6(&local_4dc,&local_cc,param_1), iVar3 == 0)) {
46         GetTempPathW(0x104,&local_2d4);
47         pwVar2 = wcsrchr(&local_2d4,L'\\');
48         if (pwVar2 != (wchar_t *)0x0) {
49             pwVar2 = wcsrchr(&local_2d4,L'\\');
50             *pwVar2 = L'\0';
51         }
52         iVar3 = FUN_00401af6(&local_2d4,&local_cc,param_1);
53         return iVar3 != 0;
54     }
55 }
56 return true;
57 }
```

העתיקתי את הקוד החל משורה 37, מכיוון שלפני זה מדובר בהגדרת משתנים וזוהי פחות רלוונטי לניתוח שלנו.

שורה 37 - מתבצע שימוש בפונקציה MultiByteToWideChar. פונקציה זאת ממירה מחרוזת תווים שמקודדת ב-MultiByte למחרוזת תווים שמקודדת בשני סיביות WideChar. לצורך הבנה, MultiByte ו-WideChar הם שני סוגים של קידוד תווים. בשורה הזאת הפונקציה ממירה את הערך randomstring. מי שמקבל את התוצאה הוא local_cc, לכן לצורך נוחות אשנה לו את השם ל-rnandomstring_w.

שורה 38 - מתבצע שימוש בפונקציה GetWindowsDirectoryW, שהיא פונקצייה שמחזירה את הנתבי של תיקיית windows או את הנתבי לתיקיית C (תלוי איפה הותקנה מערכת ההפעלה) לפרמטר הראשון שבקוד נקרא local_4dc. התוצאה שנקבל תהיה:

C:\

או

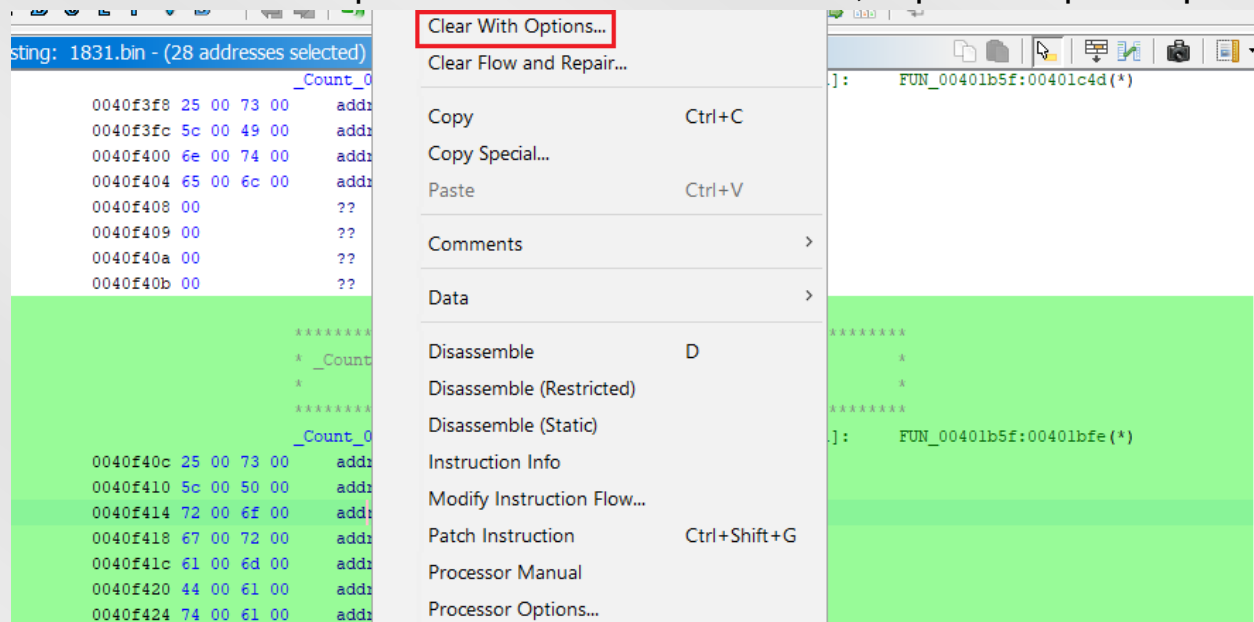
C:\Windows

לכן אשנה למשתנה local_4dc את השם ל-C_or_windows_directory

שורה 40 - ישנו שימוש בפונקציה swprintf, שמדפיסה מחרוזת תווים מקודדת ב-Unicode לבאפר. הפרמטר השני אמור להיות מספרי וצריך להכיל את מספר התווים המרבי שניתן להדפיס לבאפר, אך ניתן לראות שבמקום מופיע לנו x40f40c0. נלחץ על x40f40c0 דאבל קליק, ונגיע לחלק הבא:

_Count_0040f40c			
0040f40c	25 00 73 00	addr	Rsrc_XIA_80a_409[3276597]
0040f410	5c 00 50 00	addr	Rsrc_XIA_80a_409[982892]
0040f414	72 00 6f 00	addr	Rsrc_XIA_80a_409[3014530]
0040f418	67 00 72 00	addr	Rsrc_XIA_80a_409[3211127]
0040f41c	61 00 6d 00	addr	Rsrc_XIA_80a_409[2883441]
0040f420	44 00 61 00	addr	Rsrc_XIA_80a_409[2096980]
0040f424	74 00 61 00	addr	Rsrc_XIA_80a_409[2097028]
0040f428	00	??	00h
0040f429	00	??	00h
0040f42a	00	??	00h
0040f42b	00	??	00h

מכיוון שהחלק הזה לא קריא, סימנתי אותו והשתמשתי ב-clear with options:



ניתן לראות שקיבלתי את התוצאה הבאה:

	DAT_0040f40c			XREF[1]:	FUN_00401b5f:00401bfe(*)
0040f40c 25	??	25h	%		? -> 00730025
0040f40d 00	??	00h			
0040f40e 73	??	73h	s		? -> 005c0073
0040f40f 00	??	00h			
0040f410 5c	??	5Ch	\		? -> 0050005c
0040f411 00	??	00h			
0040f412 50	??	50h	P		? -> 00720050
0040f413 00	??	00h			
0040f414 72	??	72h	r		? -> 006f0072
0040f415 00	??	00h			
0040f416 6f	??	6Fh	o		? -> 0067006f
0040f417 00	??	00h			
0040f418 67	??	67h	g		? -> 00720067
0040f419 00	??	00h			
0040f41a 72	??	72h	r		? -> 00610072
0040f41b 00	??	00h			
0040f41c 61	??	61h	a		? -> 006d0061
0040f41d 00	??	00h			
0040f41e 6d	??	6Dh	m		? -> 0044006d
0040f41f 00	??	00h			
0040f420 44	??	44h	D		? -> 00610044
0040f421 00	??	00h			
0040f422 61	??	61h	a		? -> 00740061
0040f423 00	??	00h			
0040f424 74	??	74h	t		? -> 00610074
0040f425 00	??	00h			
0040f426 61	??	61h	a		

הערך הזה הוא string של unicode, לכן אגדיר אותו ע"י קליק ימני על בלוק הנתונים (DAT_0040f40c), לחיצה על Data ואז על TerminatedUnicode.
 אבל רגע, הפרמטר השני ב-sprintf אמור להיות מספר של גודל המשתנה שמקבל את הערך של ההדפסה, איך זה הגיוני?
 ב-malware של wannacry, הפונקציה הזאת בנויה בעיצוב מיוחד שלא מקבל גודל בתור פרמטר. לכן, אמחק את הפרמטר של size בפונקציה sprintf בעזרת edit function signature.

```
sprintf(&local_2d4,u_%"s\ProgramData_0040f40c,&C_or_windows_directory);
```

אפשר להבין לבד מה קורה כאן, הערך של C_or_windows_directory מתחבר עם programdata\, והכל נשמר בפרמטר הראשון שהוא &local_2d4.
 נוצר בערך משהו כזה:

C:\ProgramData

או

C:\Windows\ProgramData

אשנה ל-&local_2d4 את השם ל-pd_intel_temp.

בהמשך נבין למה בחרתי את השם הזה.

שורה 41 - מתבצע שימוש בפונקציה `GetFileAttributesW` ששמחזירה את תכונות הנתביב של `program data`. התכונות הללו יכולות להגדיר את סוג התיקיה, את זמינותו, את השימושים האפשריים בו ועוד. התוצאה נשמרת במשתנה `1DVar`, לכן אשנה את שמו ל-`pd_attr`.

שורה 42 - 43 - אם:

1. הנתביב לא נמצא והתכונות לא עברו
2. פונקציית `6FUN_00401af` לא הצליחה (החזירה אפס)

בוא ניכנס אל הפונקציה `6FUN_00401af` בשביל להבין מה קורה

FUN_00401af6

```
2 undefined4 __cdecl FUN_00401af6(LPCWSTR param_1,LPCWSTR param_2, wchar_t *param_3)
3
4 {
5     BOOL BVar1;
6     DWORD DVar2;
7
8     CreateDirectoryW(param_1, (LPSECURITY_ATTRIBUTES) 0x0);
9     BVar1 = SetCurrentDirectoryW(param_1);
10    if (BVar1 != 0) {
11        CreateDirectoryW(param_2, (LPSECURITY_ATTRIBUTES) 0x0);
12        BVar1 = SetCurrentDirectoryW(param_2);
13        if (BVar1 != 0) {
14            DVar2 = GetFileAttributesW(param_2);
15            SetFileAttributesW(param_2, DVar2 | 6);
16            if (param_3 != (wchar_t *) 0x0) {
17                swprintf(param_3, u_0040eb88, param_1, param_2);
18            }
19            return 1;
20        }
21    }
22    return 0;
23 }
```

הפונקציה FUN_00401af6() לוקחת שלושה פרמטרים:

פרמטרים:

1. pd_intel_temp - כרגע הוא מאחסן את הנתוב ל-ProgramData
 2. randomstring_w - שהוא ערכו של randomstring
 3. param_1 - מצביע ל-1 של הפונקציה create_random_hidden_directory שהוא המספר אפס
- עצרתי בשורה 11, לא ברור לי מה זה הפרמטר השני

שורה 8 - ישנו שימוש בפונקציה CreateDirectoryW שיוצר תיקייה חדשה בנתיב של ProgramData (ספק אם זה עובד)

שורה 9 - יש שימוש בפונקציה SetCurrentDirectoryW שמעביר את ה-process שרץ (שהוא ה-malware) לנתיב שנמצאת בתוך הסוגריים - שהוא הנתיב ל-ProgramData.
אני מזכיר, הנתיב הוא:

C:\ProgramData

או

C:\Windows\ProgramData

אם SetCurrentDirectoryW מצליחה, ערך ההחזר הוא שונה מאפס. אם הפונקציה נכשלת, ערך ההחזר הוא אפס.

שורה 10-11 - אם הפונקציה הקודמת ההצליחה, תשתמש בפונקציה CreateDirectoryW ותיצור תיקייה חדשה בתוך הנתבי של ProgramData עם השם של randomstring. זה יהיה בערך:

C:\Windows\ProgramData

או:

C:\ProgramData

שורה 12 - ה-process נכנס אל התיקייה החדשה שהוא יצר
שורה 13 - בדיקה אם זה הצליח, תמשיך לקוד בשורות הבאות
שורה 14 - קבלת התכונות של התיקייה החדשה שיצרנו לתוך המשתנה 2DVar
שורה 15 - שימוש בפונקציה SetFileAttributesW שמשנה את התכונות של התיקייה.
הערך של 6 מייצג את התכונה FILE_ATTRIBUTE_HIDDEN, כלומר לתיקייה hidden
שורה 16 - אם 3_param הוא שונה מאפס.
שורה 17 - שימוש בפונקציה swprintf כדי לאחסן במשתנה 3_param את הנתבי המלא של התיקייה החדשה שיצרנו. זה יהיה משהו בערך כמו:

C:/windows/ProgramData/askhjfdhsdk343242340

או:

C:/ProgramData/askhjfdhsdk343242340

הפונקציה מחזירה 1 אם הכל עבד, ו-0 אם נכשלה

סיכום:

הפונקציה יוצרת תיקייה חדשה בנתבי ל-ProgramData, משנה את התיקייה ל-hidden, ומחזירה 1 אם הפעולות הצליחו ו-0 אם נכשלו.
לצורך הנוחות, אקרא לפונקציה create_new_dir

בחזרה לפונקציה FUN_00401b5f

FUN_00401b5f

```

44 if ((pd_attr == 0xffffffff) ||
45     (iVar2 = create_new_dir(&pd_intel_temp,&randomstring_w,L'\0'), iVar2 == 0)) {
46     swprintf(&pd_intel_temp,u_%s\Intel_0040f3f8,&c_or_windows_directory);
47     /* C:\Windows\ProgramData\Intel or C:\ProgramData\Intel */
48     iVar2 = create_new_dir(&pd_intel_temp,&randomstring_w,L'\0');
49     if ((iVar2 == 0) &&
50         (iVar2 = create_new_dir(&c_or_windows_directory,&randomstring_w,L'\0'), iVar2 == 0)) {
51         GetTempPathW(260,&pd_intel_temp);
52         /* temp path */
53         pwVar1 = wcsrchr(&pd_intel_temp,L'\\');
54         if (pwVar1 != (wchar_t *)0x0) {
55             pwVar1 = wcsrchr(&pd_intel_temp,L'\\');
56             *pwVar1 = L'\0';
57         }
58         iVar2 = create_new_dir(&pd_intel_temp,&randomstring_w,L'\0');
59         return iVar2 != 0;
60     }
61 }
62 return true;

```

שורה 44-45 - אם:

3. הנתיב לא נמצא והתכונות לא עברו

4. פונקציית FUN_00401af6 שהיא create_new_dir (שיוצרת תיקייה חדשה ב-hidden) לא הצליחה (החזירה אפס)

תעשה את כל הפעולות הבאות:

שורה 46 - ישנו שימוש בפונקציה swprintf שמחבר את הערך של c_or_windows_directory עם הפרמטר השני שנקרא Count_0040f3f8 ומכניס את הכל לתוך התוצאה של הפרמטר הראשון. לאחר שימוש ב-clear with options ובהגדרה שלו כ-unicode (כמו שעשינו פעם קודמת), מדובר במחרוזת עם הערך של Intel:

```

                                u_%s\Intel_0040f3f8
0040f3f8 25 00 73          unicode      u"%s\\Intel"
                                00 5c 00
                                49 00 6e ...

```

אז תאכלס, הפרמטר הראשון בפונקציה swprintf, שהוא pd_intel_temp, כבר לא עם הנתיב של program data כמו שהיה מקודם, אלא משהו כזה:
C:\Windows\ProgramData\Intel

או:

C:\ProgramData\Intel

שורה 48 - כניסה לפונקציה הקודמת שחקרנו create_new_dir עם הנתיב החדש.
התוצאה תהיה:

C:\Windows\ProgramData\Intel\randomstring

או

C:\ProgramData\Intel\randomstring

שורה 49-50 - כאן יש 2 תנאים:

1. אם הפונקציה create_new_dir שבוצעה בשורה הקודמת (עם תיקיית intel) לא הצליחה עם הנתיבים:

:Windows\ProgramData\Intel

או:

C:\ProgramData\Intel

2. אם הפונקציה create_new_dir לא הצליחה ביחד עם הנתיב של המשתנה
C_or_windows_directory שהוא:

C:\ProgramData

או

C:\Windows\ProgramData

תעשה את הפעולות הבאות:

שורה 51 - יש שימוש בפונקציה GetTempPathW שמחזיר לפרמטר pd_intel_temp את הנתיב של תיקיית temp. זה למה קוראים לו pd_intel_temp.

שורה 64 - יש שימוש נוסף בפונקציה create_new_dir. הפעם עם הנתיב של temp

סיכום:

הפונקציה מנסה ליצור תיקייה חדשה ב-hidden (בעזרת הפונקציה `create_new_dir`)
בנתיבים שונים

1. בהתחלה זה `C:\Windows\ProgramData` או `C:\ProgramData`
2. לאחר מכן ב-`C:\Windows\ProgramData\Intel` או ב-`C:\ProgramData\Intel` וגם
ב-`C:\Windows` או `C:`
3. ואז בתיקיית `temp`

לאחר שהבנו איך הפונקציה עובדת, בוא נקרא לפונקציה `FUN_00401b5f` בשם
`create_random_hidden_directory`

בוא נחזור לפונקציית `WinMain`

WinMain

בוא נחזור לאיפה שהפסקנו:

```
33     if ((arg4_cmp == 0) &&
34         (bVar1 = create_random_hidden_directory((wchar_t *)0x0), CONCAT31(extraout_var,bVar1) != 0))
35     {
36         CopyFileA(filename,s_tasksche.exe_0040f4d8,0);
37         DVar2 = GetFileAttributesA(s_tasksche.exe_0040f4d8);
38         if ((DVar2 != 0xffffffff) && (arg4_cmp = FUN_00401f5d(), arg4_cmp != 0)) {
39             return 0;
40         }
41     }
42 }
43 s_/i = strrchr(filename,0x5c);
44 if (s_/i != (char *)0x0) {
45     s_/i = strrchr(filename,0x5c);
46     *s_/i = '\\0';
47 }
48 SetCurrentDirectoryA(filename);
49 FUN_004010fd(1);
50 FUN_00401dab((HMODULE)0x0);
51 FUN_00401e9e();
52 FUN_00401064(s_attrib+h_.0040f520,0,(LPDWORD)0x0);
53 FUN_00401064(s_icaccls._/grant_Everyone:F_/T_/C_0040f4fc,0,(LPDWORD)0x0);
54 arg4_cmp = FUN_0040170a();
55 if (arg4_cmp != 0) {
56     FUN_004012fd();
57     arg4_cmp = FUN_00401437(local_6e8,(LPCSTR)0x0,0,0);
58     if (arg4_cmp != 0) {
59         local_8 = 0;
60         psVar3 = (short *)FUN_004014a6(local_6e8,s_t.wnry_0040f4f4,&local_8);
```

שורה 33 - 34 - כאן יש 2 תנאים:

3. במידה והערך של arg4_cmp הוא אפס, כלומר ההשוואה באמת דומה

4. במידה והמשתנה 1bVar לא אפס, כלומר, במידה והפונקציה FUN_00401b5f הצליחה.

במידה ו-2 תנאים אלה מתקיימים, תמשיך לשורה הבאה

שורה 36 - יש שימוש בפונקציה CopyFileA, פונקציה זאת מעתיקה את המשתנה filename (מזכיר שדיברנו עליו כבר, שהוא מכיל את הנתבי המלא של ה-process הנוכחי, שזה ה-malware) ונותנת לו את השם tasksche.exe

שורה 37 - יש שימוש בפונקציה GetFileAttributesA כדי לקבל את התכונות של הקובץ של ה-malware בעל השם tasksche.exe. התכונות עוברות למשתנה 2DVar.

שורה 38 - כאן יש 2 תנאים:

1. אם התכונות לא אפס (xfffffff0), כלומר הצליחו לעבור
2. אם הפונקציה FUN_00401f5d הצליחה, כלומר הערך שיוצא לא אפס

אז תחזיר את הערך אפס.

בוא נראה מה הפונקציה FUN_00401f5d עושה

FUN_00401f5d

```
2 undefined4 FUN_00401f5d(void)
3
4 {
5     int iVar1;
6     undefined4 *puVar2;
7     CHAR local_20c;
8     undefined4 local_20b;
9
10    local_20c = DAT_0040f910;
11    puVar2 = &local_20b;
12    for (iVar1 = 0x81; iVar1 != 0; iVar1 = iVar1 + -1) {
13        *puVar2 = 0;
14        puVar2 = puVar2 + 1;
15    }
16    *(undefined2 *)puVar2 = 0;
17    *(undefined *)((int)puVar2 + 2) = 0;
18    GetFullPathNameA(s_tasksche.exe_0040f4d8,0x208,&local_20c,(LPSTR *)0x0);
19    iVar1 = FUN_00401ce8(&local_20c);
20    if ((iVar1 != 0) && (iVar1 = FUN_00401eff(0x3c), iVar1 != 0)) {
21        return 1;
22    }
23    iVar1 = FUN_00401064(&local_20c,0,(LPDWORD)0x0);
24    if ((iVar1 != 0) && (iVar1 = FUN_00401eff(0x3c), iVar1 != 0)) {
25        return 1;
26    }
27    return 0;
```

נתחיל את האנליזה משורה 18 מכיוון שלפני זה יש רק הגדרת משתנים

שורה 18 - יש שימוש בפונקציה GetFullPathNameA על הקובץ של tasksche.exe והמשתנה local_20c מקבל את הנת"ב.
מעכשיו נקראו למשתנה local_20c בשם path_to_tasksche.exe

שורה 19 - הפונקציה FUN_00401ce8 משתמשת בנת"ב של tasksche.exe שנמצא במשתנה local_20c.

בוא נראה מה הפונקציה FUN_00401ce8 עושה

FUN_00401ce8

```
12 local_c = 0;
13 local_8 = OpenSCManagerA((LPCSTR)0x0, (LPCSTR)0x0, 0xf003f);
14 if (local_8 == (SC_HANDLE)0x0) {
15     uVar1 = 0;
16 }
17 else {
18     local_10 = OpenServiceA(local_8, (LPCSTR)&randomstring, 0xf01ff);
19     if (local_10 == (SC_HANDLE)0x0) {
20         sprintf(local_410, s_cmd.exe/_c_"%s"_0040f42c, path_to_taskche.exe);
21         hService = CreateServiceA(local_8, (LPCSTR)&randomstring, (LPCSTR)&randomstring, 0xf01ff, 0x10, 2, 1,
22             local_410, (LPCSTR)0x0, (LPDWORD)0x0, (LPCSTR)0x0, (LPCSTR)0x0,
23             (LPCSTR)0x0);
24         uVar1 = local_c;
25         if (hService != (SC_HANDLE)0x0) {
26             StartServiceA(hService, 0, (LPCSTR *)0x0);
27             CloseServiceHandle(hService);
28             local_c = 1;
29             uVar1 = local_c;
30         }
31     }
32     else {
33         StartServiceA(local_10, 0, (LPCSTR *)0x0);
34         CloseServiceHandle(local_10);
35         uVar1 = 1;
36     }
37     CloseServiceHandle(local_8);
38 }
39 return uVar1;
```

לצורך הנוחות, שיניתי את השם של param_1 לאותו שם של הפרמטר שהפונקציה FUN_00401ce8 השתמשה בו שהוא path_to_taskche.exe (כי זה אותו משתנה)

שורה 13 - יש שימוש בפונקציה OpenSCManagerA, שמחזירה handle לבסיס הנתונים של ה-Service Control Manager. לצורך הנוחות נקרא למשתנה local_8 בשם csmanager.

שורה 14-15 - אם csmanager שווה לאפס (כלומר לא הצליח), הערך של uVar1 יהיה אפס. במידה והוא לא אפס (כלומר הפונקציה OpenSCManagerA הצליחה), תמשיך לכל שאר הקוד.

שורה 18 - יש שימוש בפונקציה OpenServiceA שמנסה לפתוח service קיים עם השם randomstring (ניתן לראות בפרמטר השני). אני רק מזכיר שאף פעם לא היה לנו service כזה, לכן הערך של המשתנה שמקבל את התוצאה local_10 יהיה אפס

שורה 19 - במידה ו-local_10 שווה לאפס (כלומר, במידה ואתה לא מצליח לפותח service בשם של randomstring - מה שנכון), תמשיך לפעולות הבאות בתוך הבלוק.

שורה 20 - יש שימוש בפונקציה sprintf, שזאת פונקציה ששומרת את הפלט שנוצר בתוך המשתנה local_410. ה-string יהיה:

```
cmd.exe /c "%s"
```

הערך ב-s%" הוא path_to_taskche.exe, לכן הפלט הוא:

```
cmd.exe /c path to malware
```

כלומר, פקודה לפתוח את cmd ולהפעיל את ה-malware. ניתן ל-local_10 את השם cmd_with_malware

שורה 21 - 23 - מתבצע שימוש בפונקציה CreateServiceA, שיוצרת service חדש שמכיל את הקוד שמפעיל את ה-malware (המשתנה cmd_with_malware). הפונקציה מחזירה מזהה (handle) ל-service שנוצר שהוא hService. נקרא לו h_malware_service

שורה 25 - אם h_malware_service לא שווה לאפס (כלומר הצליח להיווצר) שורה 26 - תשתמש בפונקציית StartServiceA שתפעיל את ה-service של ה-malware.

שורה 27 - שימוש בפקודת closeServiceHandle כדי לסגור את ה-handle ל-service בשם h_malware_service

שורה 29 - תחזיר את הערך 1 (local_c כרגע שווה ל-1) והמשתנה uVar1 מקבל את הערך שלו.

שורה 32 - else לתנאי שמנסה לפתוח service קיים עם השם randomstring. כלומר, אם הצלחת למצוא service עם השם של המשתנה randomstring ולפתוח אותו.

שורה 33 - תפעיל את ה-service (שימוש בפונקציית StartServiceA)

שורה 34 - תסגור את ה-handle לאותו service (שימוש בפקודת closeServiceHandle)

שורה 35 - מעדכן את המשתנה uVar1 ל-1 (בסוף הוא יחזיר את הערך הזה)

שורה 37 - סגירת ה-handle של ה-Service Control Manager ע"י הפונקציה
CloseServiceHandle

שורה אחרונה - תחזיר את ערכו של uVar1

הפקודות האחרות סוגרות את ה-handles ומחזיר 1 אם אחת מ-2 השיטות עבדו
(הפעלת ה-service החדש או הפעלת ה-service של randomstring)

סיכום:

הפונקציה מנסה לפתוח את מנהל השירותים במערכת ה-Windows.
אם הפתיחה הצליחה, היא מנסה לפתוח שירות קיים בשם "randomstring". אם
השירות לא נמצא, היא מכינה מחרוזת שמכילה פקודה להריץ את "taskche.exe" ויוצרת
שירות חדש שמפעיל את המחרוזת הזו.
אם השירות החדש נוצר בהצלחה, היא מפעילה אותו
אם השירות "randomstring" קיים, היא מפעילה אותו.
לסיום, הפונקציה מחזירה 1 אם היא הצליחה ליצור או להפעיל שירות בהצלחה, במידה
ולא היא מחזירה 0.

לאחר שהבנו מה קורה בפונקציה FUN_00401ce8, ניתן לה את השם
create_tasksche_service

בוא נחזור לפונקציה FUN_00401f5d

FUN_00401f5d

```
1
2 undefined4 FUN_00401f5d(void)
3
4 {
5     int iVar1;
6     undefined4 *puVar2;
7     CHAR path_to_taskche.exe;
8     undefined4 local_20b;
9
10    path_to_taskche.exe = DAT_0040f910;
11    puVar2 = &local_20b;
12    for (iVar1 = 0x81; iVar1 != 0; iVar1 = iVar1 + -1) {
13        *puVar2 = 0;
14        puVar2 = puVar2 + 1;
15    }
16    *(undefined2 *)puVar2 = 0;
17    *(undefined *)((int)puVar2 + 2) = 0;
18    GetFullPathNameA(s_tasksche.exe_0040f4d8,0x208,&path_to_taskche.exe,(LPSTR *)0x0);
19    iVar1 = create_tasksche_service(&path_to_taskche.exe);
20    if ((iVar1 != 0) && (iVar1 = FUN_00401eff(0x3c), iVar1 != 0)) {
21        return 1;
22    }
23    iVar1 = FUN_00401064(&path_to_taskche.exe,0,(LPDWORD)0x0);
24    if ((iVar1 != 0) && (iVar1 = FUN_00401eff(0x3c), iVar1 != 0)) {
25        return 1;
26    }
27    return 0;
28 }
```

שורה 20 - כאן יש 2 תנאים:

1. אם iVar1 לא שווה לאפס, כלומר הפונקציה create_tasksche_service שמנסה

להפעיל service זדוני הצליחה

2. אם והפונקציה FUN_00401eff גם הצליחה כי ערכו של iVar1 לא שווה לאפס

תחזיר 1.

בוא נראה מה התוכן של הפונקציה FUN_00401eff

FUN_00401eff

```
2 undefined4 __cdecl FUN_00401eff(int param_1)
3
4 {
5     HANDLE hObject;
6     int iVar1;
7     char local_68 [100];
8
9     sprintf(local_68,(char *)&_Format_0040f4ac,s_Global\MsWinZonesCacheCounterMut_0040f4b4,0);
10    iVar1 = 0;
11    if (0 < param_1) {
12        do {
13            hObject = OpenMutexA(0x100000,1,local_68);
14            if (hObject != (HANDLE)0x0) {
15                CloseHandle(hObject);
16                return 1;
17            }
18            Sleep(1000);
19            iVar1 = iVar1 + 1;
20        } while (iVar1 < param_1);
21    }
22    return 0;
23 }
```

שורה 9 - שימוש בפקודה sprintf. כפי שאנחנו כבר מכירים, אמור להיות שימוש ב-%s או ב-d% בפרמטר השני. אנסה לסדר אותו כמו שעשינו בפעמים הקודמות:

1. clear code bytes

2. Data > TerminatedCString

ניתן לראות שהוא הסתדר:

_Format_0040f4ac			
0040f4ac	25 73 25	ds	"%s%d"
	64 00		
0040f4b1	00	ds	""
0040f4b2	00	ds	""
0040f4b3	00	ds	""

מה שנכנס במשתנה local_68 שנמצא ב-array buffer הוא:
Global\MsWinZonesCacheCounterMutexA0

כלומר, נוצר mutex. למי שלא מכיר, mutex הוא מנגנון שמאפשר רק ל-thread אחד לגשת למשאב משותף וחוסם thread אחרים. כאשר thread רוצה לגשת למשאב, הוא חייב לרכוש תחילה את ה-mutex. הוא כמו מפתח לדלת. הוא מאפשר לך לגשת למשאב משותף, כמו קבצי טקסט או זיכרון, מבלי שהאחרים יוכלו לגשת אליו בו זמנית. דבר זה נועד למנוע התנגשויות.

שורה 11 - אם אפס קטן מ-param_1 תיכנס לבלוק הבא ותעשה את הפעולות הבאות. הערך של param_1 הוא 60 (ע"פ התוכן בפונקציה FUN_00401f5d). לכן נשנה את השם ל-param_1 לשם sixty.

שורה 13 - יש שימוש בפונקציה OpenMutexA. הפונקציה OpenMutexW מאפשרת לך לפתוח mutex קיים. בשורה זאת היא מנסה לפתוח את ה-mutex שראינו לפני כן: Global\\MsWinZonesCacheCounterMutexA0 המשתנה hObject אמור לקבל את ה-handle ל-mutex.

שורה 14 - אם hObject שונה מאפס (כלומר הפונקציה OpenMutexA הצליחה) שורה 15 - 16 - סגור את ה-handle ל-mutex בעזרת הפונקציה CloseHandle, ותחזיר את הערך 1

שורה 18 - 22 - אם זה לא הצליח, תעשה sleep ותנסה שוב (ינסה 60 פעמים). במידה וגם זה לא הצליח, תחזיר 0.

סיכום:

הפונקציה "acquire_tasksche_mutex" יוצרת Mutex ומנסה לפתוח אותו. Mutex הוא מנגנון שמאפשר רק לתוכנית אחת לגשת למשאב משותף ומונע גישה של תוכניות אחרות למשאב זה באותו זמן. במקרה הזה, WannaCry משתמשת ב-Mutex כדי למנוע מתוכנות אחרות לפתוח את הקבצים שהיא נעלה. אם הפונקציה הצליחה לפתוח את ה-Mutex, היא מחזירה 1 כסימון להצלחה. אם היא לא הצליחה לפתוח אותו, היא מנסה לפתוח את ה-Mutex שוב, עד 60 נסיונות. אם לאחר 60 נסיונות היא לא הצליחה, היא מחזירה 0 כסימון לכישלון.

נקרא לפונקציה FUN_00401eff בשם acquire_tasksche_mutex

FUN_00401f5d

בוא נחזור לאיפה שעצרנו ב-

FUN_00401f5d

```
2 undefined4 FUN_00401f5d(void)
3
4 {
5     int iVar1;
6     undefined4 *puVar2;
7     CHAR path_to_taskche.exe;
8     undefined4 local_20b;
9
10    path_to_taskche.exe = DAT_0040f910;
11    puVar2 = &local_20b;
12    for (iVar1 = 0x81; iVar1 != 0; iVar1 = iVar1 + -1) {
13        *puVar2 = 0;
14        puVar2 = puVar2 + 1;
15    }
16    *(undefined2 *)puVar2 = 0;
17    *(undefined *)((int)puVar2 + 2) = 0;
18    GetFullPathNameA(s_tasksche.exe_0040f4d8,0x208,&path_to_taskche.exe,(LPSTR *)0x0);
19    iVar1 = create_tasksche_service(&path_to_taskche.exe);
20    if ((iVar1 != 0) && (iVar1 = aquire_tasksche_mutex(0x3c), iVar1 != 0)) {
21        return 1;
22    }
23    iVar1 = FUN_00401064(&path_to_taskche.exe,0,(LPDWORD)0x0);
24    if ((iVar1 != 0) && (iVar1 = aquire_tasksche_mutex(0x3c), iVar1 != 0)) {
25        return 1;
26    }
27    return 0;
28 }
```

שורה 20 - 21 - כאן יש 2 תנאים:

3. אם iVar1 לא שווה לאפס, כלומר הפונקציה create_tasksche_service שמנסה

להפעיל service זדוני הצליחה. וגם

4. אם והפונקציה aquire_tasksche_mutex שמנסה לפתוח mutex גם הצליחה

(כי ערכו של iVar1 לא שווה לאפס)

תחזיר 1.

שורה 23 - מתבצע שימוש בפונקציה FUN_00401064. בוא נראה מה היא עושה

FUN_00401064

```
26 BVar1 = CreateProcessA((LPCSTR)0x0,path_to_taskche.exe,(LPSECURITY_ATTRIBUTES)0x0,  
27 (LPSECURITY_ATTRIBUTES)0x0,0,0x80000000,(LPVOID)0x0,(LPCSTR)0x0,&local_58,  
28 &local_14);  
29 if (BVar1 == 0) {  
30     uVar5 = 0;  
31 }  
32 else {  
33     if (zero != 0) {  
34         DVar2 = WaitForSingleObject(local_14.hProcess,zero);  
35         if (DVar2 != 0) {  
36             TerminateProcess(local_14.hProcess,0xffffffff);  
37         }  
38         if (zero0 != (LPDWORD)0x0) {  
39             GetExitCodeProcess(local_14.hProcess,zero0);  
40         }  
41     }  
42     CloseHandle(local_14.hProcess);  
43     CloseHandle(local_14.hThread);  
44 }  
45 return uVar5;  
46 }
```

נתחיל את האנליזה משורה 26 מכיוון שלפני זה יש רק הגדרת משתנים. כמו כן, הגדרתי את הפרמטרים של הפונקציה FUN_00401064 כמו הפרמטרים שהיא מקבלת בשורה 23 כדי שיהיה לנו יותר קל להבין

שורה 26-28 - יש שימוש בפונקציה CreateProcessA שיוצרת process חדש במערכת ההפעלה Windows. הפרמטר השני הוא מצביע על המחרוזת שמכילה את שורת הפקודה שצריך להפעיל, כמו שאפשר לראות, הוא מכיל את המשתנה path_to_taskche.exe. זהו הניתב ל-malware. אפשר לראות שהפרמטר האחרון הוא local_14 שמכיל את המידע על ה-process החדש. מי שמקבל את התוצאה של הפונקציה הוא המשתנה BVar1.

שורה 29-30 - אם הערך BVar1 שווה לאפס (כלומר ההרצה של ה-process לא הצליחה), המשתנה uVar5 יהיה שווה ל-0 (בסוף הוא יוחזר מהפונקציה כסימן לכישלון)

שורה 32 - שימוש ב-else, כלומר, אם היצירה החדשה של ה-process עם ה-malware הצליחה, תבצע את כל המשך הקוד.

שורה 33 - אם המשתנה zero (שהגיע בתור פרמטר שני לפונקציה) לא שווה לאפס (מה שלא נכון כרגע), תעשה את הפעולות הבאות:

שורה 34 - יש שימוש בפונקציה WaitForSingleObject שממתינה עד שה-malware יסיים לעבוד

שורה 35 - אם ה-malware לא סיים לעבוד

שורה 36 - יש שימוש בפונקציה TerminateProcess שמפסיק את ה-process של ה-malware.

שורה 38-39 - אם המשתנה zeroo (שהגיע בתור פרמטר שלישי לפונקציה) לא שווה לאפס (מה שלא נכון כרגע),

שורה 40 - תבדוק בעזרת הפונקציה GetExitCodeProcess עם ה-process של ה-malware הסתיים בהצלחה

שורה 42-43 - תסגור את ה-handle של ה-process

סיכום:

הפונקציה יוצרת process חדש במערכת Windows (ה-process של ה-malware) ובודקת את תוצאת ההרצה שלו. אם ההרצה הצליחה, היא ממתינה עד לסיום התהליך או עד לפרק זמן ספציפי (אם הועבר פרמטר zero). כשהתהליך מסתיים, היא משיגה את קוד החזרה של התהליך ובודקת אם הוא סיים בהצלחה. הפונקציה מחזירה 1 במקרה של הצלחה ו-0 במקרה של כישלון.

בוא נקרא לה בשם run_command

נחזור לפונקציה FUN_00401f5d מאיפה שהפסקנו:

FUN_00401f5d

```
2 undefined4 FUN_00401f5d(void)
3
4 {
5     int iVar1;
6     undefined4 *puVar2;
7     CHAR path_to_taskche.exe;
8     undefined4 local_20b;
9
10    path_to_taskche.exe = DAT_0040f910;
11    puVar2 = &local_20b;
12    for (iVar1 = 0x81; iVar1 != 0; iVar1 = iVar1 + -1) {
13        *puVar2 = 0;
14        puVar2 = puVar2 + 1;
15    }
16    *(undefined2 *)puVar2 = 0;
17    *(undefined *)((int)puVar2 + 2) = 0;
18    GetFullPathNameA(s_tasksche.exe_0040f4d8,0x208,&path_to_taskche.exe,(LPSTR *)0x0);
19    iVar1 = create_tasksche_service(&path_to_taskche.exe);
20    if ((iVar1 != 0) && (iVar1 = aquire_tasksche_mutex(0x3c), iVar1 != 0)) {
21        return 1;
22    }
23    iVar1 = run_command(&path_to_taskche.exe,0,(LPDWORD)0x0);
24    if ((iVar1 != 0) && (iVar1 = aquire_tasksche_mutex(0x3c), iVar1 != 0)) {
25        return 1;
26    }
27    return 0;
}
```

שורה 24 - כאן יש 2 תנאים:

1. אם הפונקציה run_command שפעילה את ה-process של ה-malware הצליחה (כלומר לא מחזירה אפס) וגם
2. והפונקציה aquire_tasksche_mutex שמנסה לפתוח mutex גם הצליחה (כי ערכו של iVar1 לא שווה לאפס)

תחזיר 1

סיכום:

הפונקציה מנסה לפתוח או ליצור שירות במערכת Windows בשם "tasksche". היא מתחילה על ידי מציאת את המיקום המלא של ה-malware. לאחר מכן, היא משתמשת בפונקציה "create_tasksche_service" כדי לנסות לפתוח או ליצור את השירות. אם יש הצלחה, היא ממתינה גם למנעול ("mutex") שיעזור לה למנוע מתוכניות אחרות לפתוח את הקבצים שהיא נעלה. אם גם זה הצליח, היא מחזירה 1 כסימן להצלחה.

אם השירות כבר קיים והמנעול אינו זמין או אם השירות לא הצליח להיווצר, הפונקציה משתמשת בפונקציה "run_command" כדי לנסות להפעיל את ה-malware. גם כאן, היא ממתינה למנעול. אם גם זה הצליח, היא מחזירה 1 כסימן להצלחה. אם כל אחד מהנסיונות האלו נכשל, היא מחזירה 0 כסימן לכישלון.

בוא נשנה לה את השם ל-create_or_start_tasksche_service

בוא נחזור ל-WinMain איפה שעצרנו

WinMain

```
38     if ((DVar2 != 0xffffffff) && (arg4_cmp = create_or_start_tasksche_service(), arg4_cmp != 0)) {
39         return 0;
40     }
41 }
42 }
43 s_/i = strrchr(filename,0x5c);
44 if (s_/i != (char *)0x0) {
45     s_/i = strrchr(filename,0x5c);
46     *s_/i = '\\0';
47 }
48 SetCurrentDirectoryA(filename);
49 FUN_004010fd(1);
50 FUN_00401dab((HMODULE)0x0);
51 FUN_00401e9e();
52 run_command(s_attrib+_h_.0040f520,0,(LPDWORD)0x0);
53 run_command(s_icaccls._/grant_Everyone:F_/T_/C_0040f4fc,0,(LPDWORD)0x0);
54 arg4_cmp = FUN_0040170a();
55 if (arg4_cmp != 0) {
56     FUN_004012fd();
57     arg4_cmp = FUN_00401437(local_6e8,(LPCSTR)0x0,0,0);
58     if (arg4_cmp != 0) {
59         local_8 = 0;
60         psVar3 = (short *)FUN_004014a6(local_6e8,s_t.wnry_0040f4f4,&local_8);
61         if (((psVar3 != (short *)0x0) &&
62             (argv = (int *)FUN_004021bd(psVar3,local_8), argv != (int *)0x0)) &&
63             (pcVar4 = (code *)FUN_00402924(argv,s_TaskStart_0040f4e8), pcVar4 != (code *)0x0)) {
64             (*pcVar4)(0,0);
65         }
66     }
67     FUN_0040137a();
68 }
69 return 0;
```

שורה 38-93 - כאן יש 2 תנאים:

1. אם התכונות של הקובץ של ה-malware הצליחו לעבור, כלומר לא אפס

וגם

2. אם הפונקציה `create_or_start_tasksche_service`, שמנסה לרכוש mutex

לקבצים שהצפינה וגם להפעיל את ה-malware או את ה-service שלו

אז תחזיר את הערך אפס.

המשך הקוד:

```
48 SetCurrentDirectoryA(filename);
49 FUN_004010fd(1);
50 FUN_00401dab((HMODULE)0x0);
51 FUN_00401e9e();
52 run_command(s_attrib+h_.0040f520,0,(LPDWORD)0x0);
53 run_command(s_icaccls._/grant_Everyone:F/_T/_C_0040f4fc,0,(LPDWORD)0x0);
54 arg4_cmp = FUN_0040170a();
55 if (arg4_cmp != 0) {
56     FUN_004012fd();
57     arg4_cmp = FUN_00401437(local_6e8,(LPCSTR)0x0,0,0);
58     if (arg4_cmp != 0) {
59         local_8 = 0;
60         psVar3 = (short *)FUN_004014a6(local_6e8,s_t.wnry_0040f4f4,&local_8);
61         if ((psVar3 != (short *)0x0) &&
62             (argc_or_argv = (int *)FUN_004021bd(psVar3,local_8), argc_or_argv != (int *)0x0)) &&
63             (pcVar4 = (code *)FUN_00402924(argc_or_argv,s_TaskStart_0040f4e8), pcVar4 != (code *)0x0))
64         {
65             (*pcVar4)(0,0);
66         }
67     }
68     FUN_0040137a();
69 }
70 return 0;
```

שורה 49 - יש קריאה לפונקציה FUN_004010fd שמקבלת פרמטר עם הערך 1.

אני לא אכנס לחקור גם אותה, אבל המטרה שלי היא להסתכל על הפונקציות בתוך השיטות. בהמשך הקוד יש קריאה לשיטות נוספות. אני עובר ברפרוף עליהם לראות אם יש שימוש בפונקציות שמשתמשות ב-resource. בשיטה FUN_004010fd אין משהו כזה, אך ברגע שאני נכנס לפונקציה FUN_00401dab, אני רואה בשורה 18 את הקוד הבא:

```
hResInfo = FindResourceA(param_1,(LPCSTR)0x80a,&DAT_0040f43c);
```

לאחר המרה של הערך 0x80a, אני מסיק שמדובר ב-resource עם המספר 2058.

בוא נשתמש שוב בכלי wrestool ונבדוק איזה resources קיימים בתוך bin.1831:

```
$ wrestool 1831.bin
--type='XIA' --name=2058 --language=1033 [offset=0x100f0 size=3446325]
--type=16 --name=1 --language=1033 [type=version offset=0x359728 size=904]
--type=24 --name=1 --language=1033 [offset=0x359ab0 size=1263]
```

אפשר לראות ש-resource-2058 אכן קיים. בוא נוציא אותו:

```
(kali㉿kali)-[~/Downloads]
$ wrestool -R -x --name=2058 1831.bin > 2058.XIA
```

בוא ננסה לפתוח את הקובץ:

```
(kali㉿kali)-[~/Downloads]
$ file 2058.XIA
2058.XIA: Zip archive data, at least v2.0 to extract, compression method=deflate
```

ניתן לראות שמדובר בקובץ zip. במידה וננסה לבצע לו unzip, נראה שהוא דורש סיסמה. אם נחזור ל-compiler שלנו, נוכל לראות שמעל הפונקציה יש איזשהו סטרינג מוזר:

```
II II
004020c8 c7 04 24      MOV      dword ptr [ESP],s_WNcry@2017_0040f52c
          2c f5 40 00
004020cf 53            PUSH     EBX
004020d0 e8 d6 fc      CALL     FUN_00401dab
          55 55
```

נלחץ עליו פעמיים ונוכל לראות את הטקסט הבא:

```
          s_WNcry@2017_0040f52c
0040f52c 57 4e 63      ds      "WNcry@2017"
          72 79 40
          32 6f 6c ...
0040f537 00            ??      00h
```

ננסה להכניס את הטקסט WNcry@2017 בתור סיסמה, ונוכל לראות שזה עבד!
נוכל לראות בתוך הקובץ מספר קבצים שונים, כגון קובץ ההודעה על ההצפנה ותמונת הרקע שמתחלפת בעת הפעלת התוכנית.