Digital Whisper Technologic Papers

הצד האפל של TLS Callbacks

מאת יהונתן לוסקי

מבוא

Thread Local Storage או בקצרה TLS הוא מנגנון המאפשר למערכת ההפעלה להקצות מידע שהינו ייחודי ל-Thread מסויים. דהיינו, יהיו מעיין "משתנים גלובליים" שהם למעשה ייחודיים רק עבור ה-Thread אליו הם שייכים.

ב-Windows המנגנון מאפשר לנו להגדיר רוטינות נוספות הנקראות TLS Callbacks. אותן רוטינות TLS Callbacks הינן רוטינות אשר רצות בכל אחד מארבעת המצבים הבאים: טעינה של DLL, הסרה של DLL, יצירה של Thread, סיום ריצה של Thread. במאמר זה, אתמקד באותם TLS Callbacks ולא במנגנון ה-TLS עצמו.

לפני שאמשיך הלאה, לכל אותם קוראים שכבר כן מכירים את הקונספט ומבינים לאן המאמר הולך, מוזמנים לקפוץ אל סוף המאמר שם נמצא האתגר שבניתי ☺

TLS Callbacks, מה ולמה?

על אף שהשימוש ב-TLS Callbacks איננו נפוץ, לרוטינות TLS Callbacks יכול להיות מגוון רחב של שימושים לגיטימיים הקשורים באתחול של התכנית והרצה נוספת של קוד לפני תחילת התכנית הראשית. אולם, כמו שאתם יכולים לנחש כבר, כאן מגיע השלב שבו אני מפחיד אתכם ואומר שהעולם שלנו הוא עולם רע ואכזר והשימוש באותם TLS Callbacks איננו בהכרח תמים.

TLS Callbacks הינם הבחירה המועדפת על הרבה כותבי Malwares שכן הם מאפשרים להסוות קוד נוסף שרץ לפני תחילת התכנית הראשית. ניתן לחשוב על אינספור שימושים לכך, במרכזיים שבהם ניתן למצוא: בדיקת שימוש בדיבאגרים, בדיקת ריצה בסביבת מחקר, ביצוע חלק מהקוד הזדוני של ה-malware ועוד... אם לא די בכך, קיימים אף דיבאגרים כדוגמאת Olly Debugger שמקשים עלינו אף יותר, הם אינם מזהים את אותם TLS Callbacks, ויתרה מכך, בטעינה של ה-Executable הם מבצעים הרצה באופן אוטומטי עד ל-Entry Point של ה-Executable, כלומר, הם מריצים באופן אוטומטי את הקוד שנמצא בתוך ה- TLS

Callback! במקרה הטוב, במידה והקוד שנמצא ב-TLS Callback זיהה דיבאגר הוא יגרום ליציאה של התכנית, במקרה הרע הוא עשוי אף לגרום למחיקה של ה-HD.

מנגד כמובן, קיימים כלים אחרים כדוגמת IDA שכן תדע לזהות את אותם TLS Callbacks... האומנם?

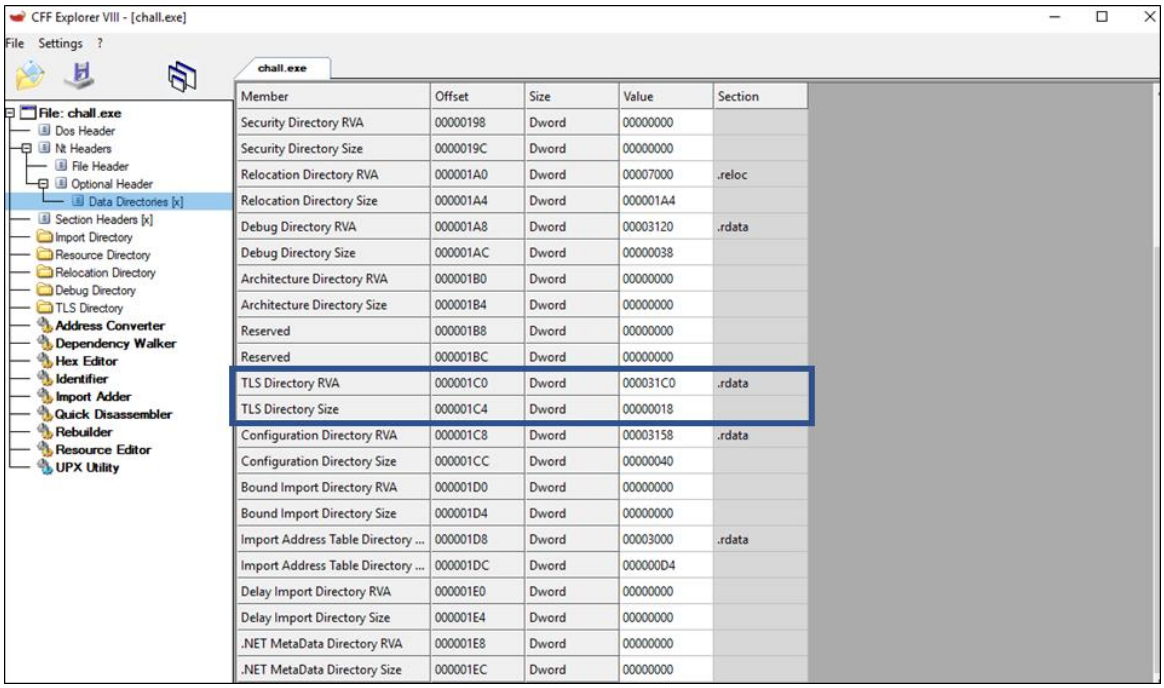
TLS Callbacks באופק

כעת, משהכרנו מה הם אותם TLS Callbacks, על מנת להמשיך הלאה, עלינו להבין תחילה כיצד מיוצגים אותם TLS Callbacks בתוך הזיכרון של ה-PE.

אותם TLS callbacks מקושרים למבנה ששייך ל-TLS ונקרא ה-TLS Directory. על מנת להגיע אל ה-TLS Directory, ניגש אל ה-Optional Header, שנמצא בתוך ה-NT Headers, שהוא חלק מההדרים של ה-PE.

כפי שניתן לראות בתמונה מתחת, ה-Optional Header מכיל שני שדות שרלוונטיים לנו:

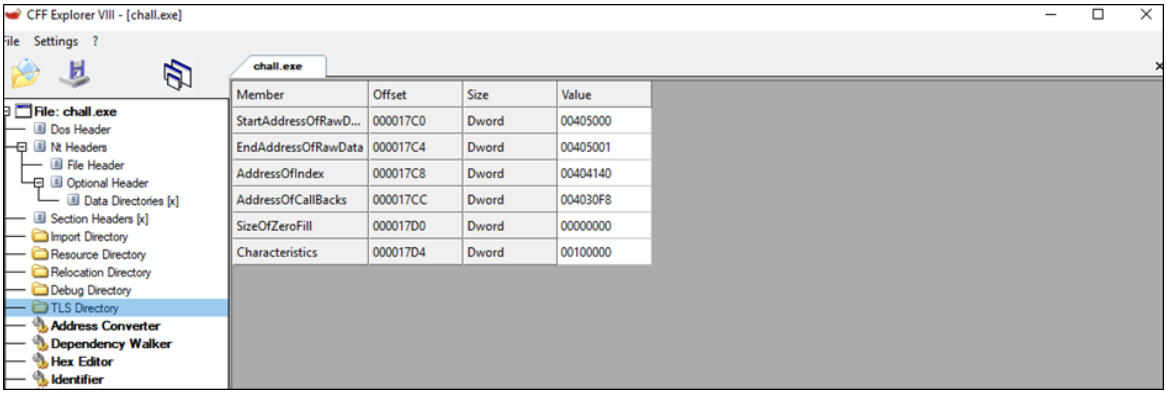
- TLS Directory RVA - מכיל את האופסט בו נמצא ה-TLS Directory.
- TLS Directory Size - מכיל את הגודל של ה-TLS Directory, שבדרך כלל יהיה 18.



Member	Offset	Size	Value	Section
Security Directory RVA	00000198	Dword	00000000	
Security Directory Size	0000019C	Dword	00000000	
Relocation Directory RVA	000001A0	Dword	00007000	.reloc
Relocation Directory Size	000001A4	Dword	000001A4	
Debug Directory RVA	000001A8	Dword	00003120	.rdata
Debug Directory Size	000001AC	Dword	00000038	
Architecture Directory RVA	000001B0	Dword	00000000	
Architecture Directory Size	000001B4	Dword	00000000	
Reserved	000001B8	Dword	00000000	
Reserved	000001BC	Dword	00000000	
TLS Directory RVA	000001C0	Dword	000031C0	.rdata
TLS Directory Size	000001C4	Dword	00000018	
Configuration Directory RVA	000001C8	Dword	00003158	.rdata
Configuration Directory Size	000001CC	Dword	00000040	
Bound Import Directory RVA	000001D0	Dword	00000000	
Bound Import Directory Size	000001D4	Dword	00000000	
Import Address Table Directory ...	000001D8	Dword	00003000	.rdata
Import Address Table Directory ...	000001DC	Dword	000000D4	
Delay Import Directory RVA	000001E0	Dword	00000000	
Delay Import Directory Size	000001E4	Dword	00000000	
.NET MetaData Directory RVA	000001E8	Dword	00000000	
.NET MetaData Directory Size	000001EC	Dword	00000000	

CFF Explorer: על מנת לפרסר את קובץ ההרצה השתמשתי בכלי שנקרא CFF Explorer, זה הוא כלי עוצמתי ושימושי מאוד שמאפשר תצוגה נוחה של המידע שמכיל ה-PE. הכלי חינמי וניתן להוריד אותו בקלות באינטרנט.

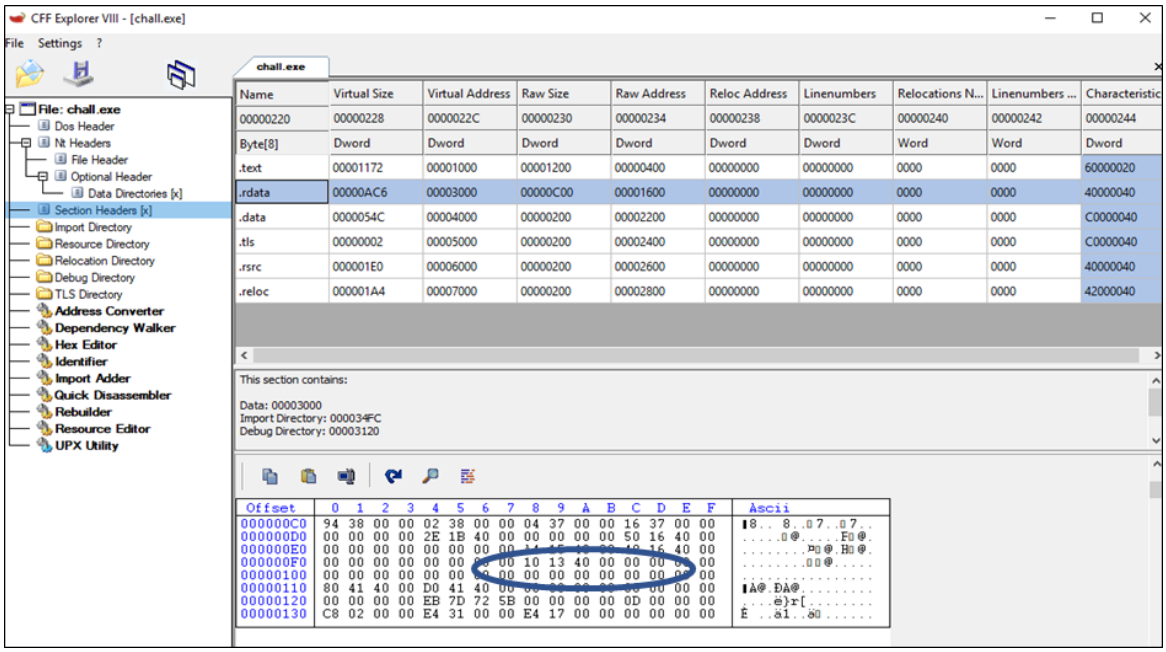
כעת משאנו יודעים היכן נמצא ה-TLS Directory, נוכל לגשת אל המבנה עצמו ולנתח אותו. על מנת לעשות זאת נוכל לבצע את אחד משני הדברים הבאים, לגשת אל התצוגה הבינארית של הקובץ ולנתח שם את ה-TLS Directory או שפשוט נשתמש ב-CFF Explorer שמציג לנו את ה-TLS Directory בצורה נוחה, הידד!



כפי שניתן לראות בתמונה לעיל ה-TLS Directory מכיל בסך הכל 6 שדות. מתוכם אתייחס אך ורק לשני שדות מעניינים, שאר השדות אף יכולים להכיל אפסים והתכנית תמשיך לפעול באופן תקין. להלן השדות:

- AddressOfCallBacks - מכיל את הכתובת האבסולוטית של מערך המכיל פוינטרים אל פונקציות ה-TLS שאמורות להיקרא.
- AddressOfIndex - מכיל את הכתובת האבסולוטית של מערך המכיל את האינדקס של כל אחד ואחד מה- TLS Callbacks.

ולבסוף במידה וניגש אל מערך של ה-Callbacks, נראה כי זה הוא למעשה מערך של כתובת אבסולוטיות לקריאה על ידי מנגנון ה-TLS:



[הערה: אל תשכחו שמדובר ב-little endian :)]

רוטינת TLS Callback

לאחר שראינו את המבנה שמכיל מידע על פונקציות ה-TLS וכיצד הן למעשה נקראות, נרצה לבנות פונקציה כזאת בעצמנו. כל TLS Callback מוגדר באופן הבא:

```
VOID (NTAPI *PIMAGE_TLS_CALLBACK) (PVOID DllHandle, DWORD Reason, PVOID
Reserved);
```

- הארגומנט DllHandle מכיל handle אל ה-DLL שה-TLS הוא חלק ממנו.
- הארגומנט Reason מכיל את אחד מארבעת המאקרואים הבאים:
  - DLL_PROCESS_ATTACH = 1
  - DLL_PROCESS_DETACH = 0
  - DLL_THREAD_ATTACH = 2
  - DLL_THREAD_DETACH = 3

בהתאם ל-Reason נבחר את הפעולה שה-TLS Callback שלנו יבצע. כותבי Malwares לרוב יממשו את הפונקציונאליות של ה-TLS כאשר Reason = DLL_PROCESS_ATTACH. למשל, להלן דוגמא ל-TLS Callback שבודק קיומו של דיבאגר, במידה וקיים אחד התכנית יוצאת:

```
VOID WINAPI check_debugger(PVOID DllHandle, DWORD Reason, PVOID Reserved)
{
    if (Reason == DLL_PROCESS_ATTACH) {
        if (IsDebuggerPresent()) {
            exit(0);
        }
    }
}
```

עכשיו, שאנו יודעים לבנות TLS Callback, כל שנותר הוא להכניס אותו אל קובץ ההרצה שלנו. במידה ויש לנו את ה-Source Code, האופציה הפשוטה כמובן תהיה להכניס את ה-TLS בתור חלק מהקוד. על כך לא אפרט, מוזמנים להציץ במאמר הבא.

ומנגד, לחבר'ה שמעוניינים להעצים את האתגר (או נטולי ה-Source Code), ניתן להכניס את ה-TLS Callback באופן ידני לקובץ הרצה קיים. למעוניינים, בשלב זה ניתן לנסות לעשות זאת באופן עצמאי, למעשה כיסינו כמעט את כל הידע הנדרש (רמז: שימו לב שמדובר בכתובות אבסולוטיות ולכן חסר אולי עוד משהו קטן שלא נגענו בו). בפרק הבא של המאמר אסביר כיצד להכניס TLS Callback באופן ידני לתוך קובץ הרצה קיים.

ה-TLS Callback הראשון שלי

התהליך יתחלק לשלושה שלבים:

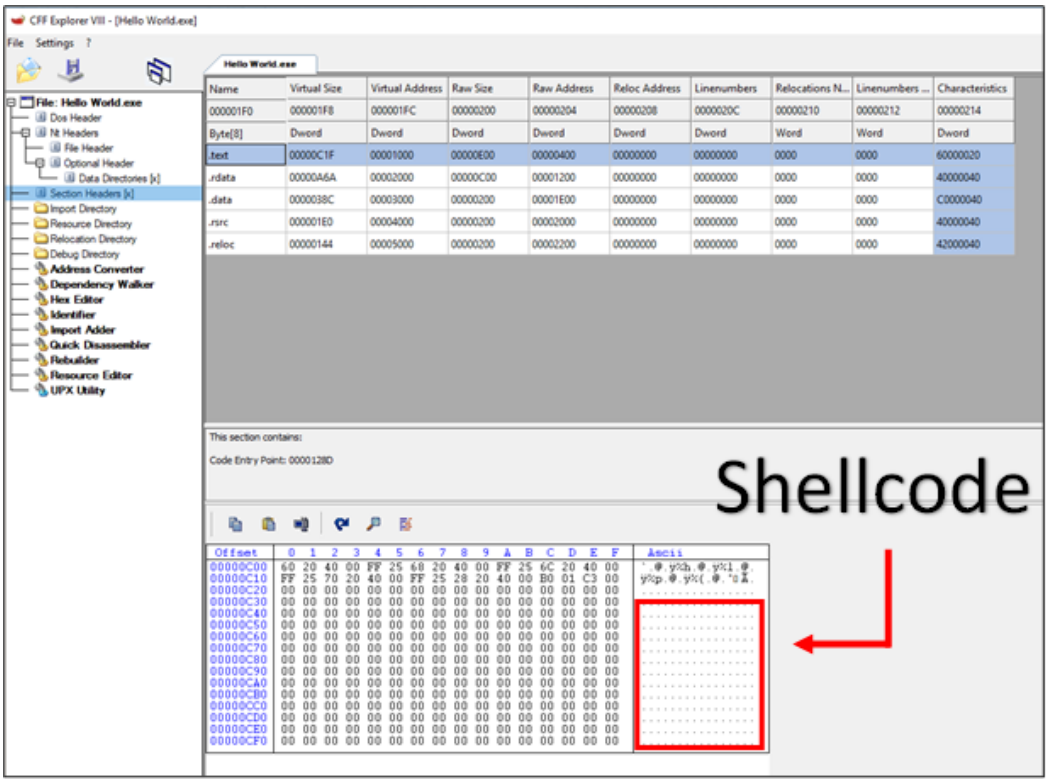
1. מציאת Code Cave להכנסת הקוד של ה-TLS Callback החדש.
2. הוספת הכתובת במערך ה-Callbacks ויצירת TLS Directory במידה ונדרש.
3. תיקון ה-Relocation Table.

הערה קטנה לפני, מדובר ב-Executable שונה מזה שראינו קודם לכן במדריך, לא להתבלבל! הבא נתחיל...

הקוד של ה-TLS Callback בו נשתמש יהיה למעשה Shellcode. מזכיר, איננו יודעים מה הן הכתובות של כל הפונקציות האחרות בזיכרון. לכן נרצה להקל על עצמנו ונשתמש ב-Shellcode שרץ כקוד עצמאי ואיננו תלוי בדברים אחרים. כמו כן, אציין גם שה-Shellcode איננו פונקציה, ואילו TLS Callback היא כן, לכן יש להוסיף ל-Shellcode פרולוג ואפילוג מתאימים לניקוי המחסנית.

לא אתעכב כיצד הדבר התבצע, אולם אציין כי השתמשתי ב-Shellcode גנרי שמצאתי באינטרנט והוספתי לו את הפרולוג והאפילוג באמצעות OllyDBG שיודע לתרגם פקודות מאסמבלי ל-OpCode המתאים.

סדר הגודל של ה-Shellcode הוא 80 בטים. כפי שניתן לראות ב-CFF Explorer, מצאתי חלל מספיק גדול בתוך ה-text section. שווה לציין כי זה הוא ה-Section האדיאלי, שכן כמובן שה-text section הוא בר הרצה הרי שמה יושב כל הקוד של שאר התכנית. כנגזרת מכך לא נצטרך לדאוג לשינוי ההרשאות של הזיכרון.

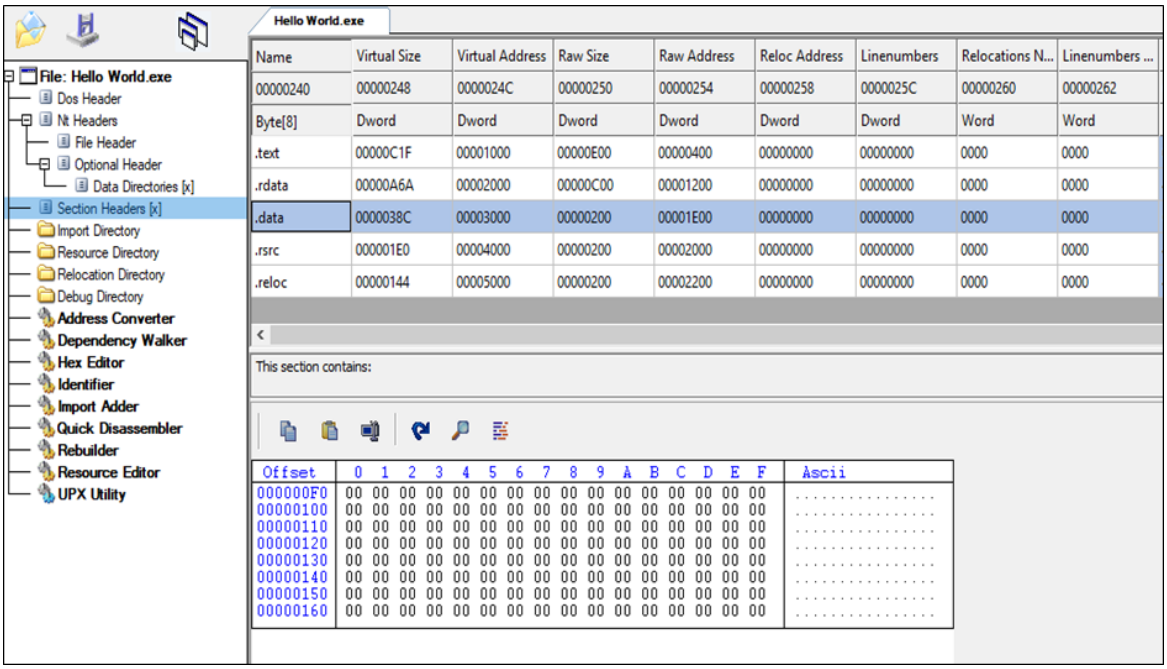


קל לראות כי ה-Shellcode נמצא בתוך ה-text Section שנמצא באופסט 1000, ובתוך ה-text Section הוא נמצא באופסט C30, כלומר בסך הכל קיבלנו כי ה-Shellcode נמצא בכתובת 1C30. זכרו את הכתובת!

בהזדמנות זאת אציין כי לצורך הכנסת הקוד בפועל ניתן להשתמש בכל Hex Editor. להמלצתי כדי להשתמש ב-010 editor שמציג את הקובץ הבינארי בצורה נוחה ונוסף על כך יודע להפריד בין ה-secion-ים השונים בזיכרון.

כעת, אחרי שהוספנו את הקוד, ניתן לעבור לשלב הבא. אנו נרצה כי ה-Loader יקרא לפונקציית ה-Callback שלנו. שימו לב, ל-Executable שטעון בתוך ה-CFF Explorer אין TLS Directory, לפיכך נצטרך לבנות אחד כזה בעצמנו.

זוכרים ש-TLS Directory סטנדרטי הוא בגודל 18? יופי! עכשיו אנו צריכים למצוא Code Cave חדש בגודל 18 בתים שיארח את ה-TLS Directory שלנו.



התבוננות קצרה מעלה כי קיים Code Cave מתאים כזה בתוך ה-data Section. נציין גם כי ה-data Section מכיל בעיקר משתנים גלבוליים ולכן הוא איננו בעל הרשאות ריצה. אולם, אין אנו זקוקים להן שכן ה-TLS Directory זקוק אך ורק להרשאות קריאה.

כמו כן, אין לשכוח שאנו גם צריכים לאחסן את מערך פונקציות ה-TLS ואת מערך האינדקסים. למזלנו ה-Code Cave שמצאנו מספיק גדול גם עבורם על כן נוכל להכניס גם אותם כאן. עריכה קצרה בתוך Hex Editor תתן את התוצאה הבאה:

Hello World.exe

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
00000240	00000248	0000024C	00000250	00000254	00000258	0000025C	00000260	00000262	00000264
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00000C1F	00001000	00000E00	00000400	00000000	00000000	0000	0000	60000020
.rdata	00000A6A	00002000	00000C00	00001200	00000000	00000000	0000	0000	40000040
.data	0000038C	00003000	00000200	00001E00	00000000	00000000	0000	0000	C0000040
.rsrc	000001E0	00004000	00000200	00002000	00000000	00000000	0000	0000	40000040
.reloc	00000144	00005000	00000200	00002200	00000000	00000000	0000	0000	42000040

```
Offset   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   Ascii
000000F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100 00 00 00 00 00 00 00 00 20 31 40 00 30 31 40 00  .........1@.01@.
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000120 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000130 30 1C 40 00 00 00 00 00 00 00 00 00 00 00 00 00  0 @.............
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

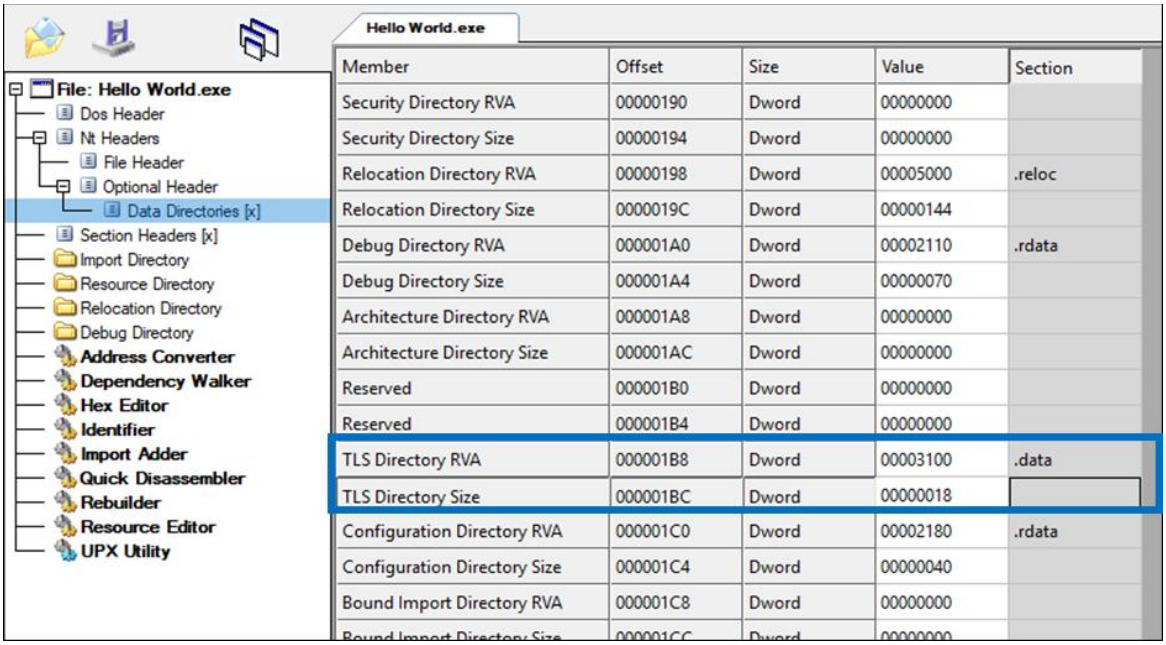
להלן הסבר קצר של מה שאנו רואים:

בחרתי כי ה-TLS Directory (התיבה הכחולה) יתחיל באופסט 100.

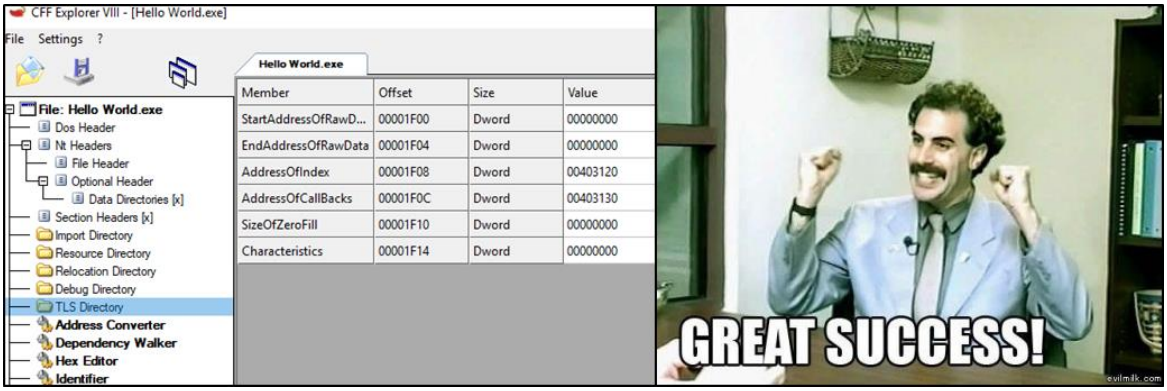
- 8 הבתים הראשונים, מדובר על ה-StartAddressOfRawData ו- EndAddressOfRawData. אמרנו כי אין להם חשיבות גדולה ולכן נשאיר אותם כאפסים.
- 4 הבתים הבאים הם למעשה ה-AddressOfIndex. נמיר את הערך שם לייצוג קריא (שימו לב, הרי מדובר בייצוג little endian) ונקבל 403120, כלומר נקבל שמערך האינדקסים נמצא בתוך ה-data section, באופסט 120 שזאת למעשה התיבה הירוקה. ה-AddressOfIndex בו בעצם כתובת אבסולוטית בהתאם ל-ImageBase שהוא 400000, לכן אנו רואים מספר כזה גדול. ועוד דבר קטן, מערך האינדקסיים הינו ריק שכן התכנית שלנו מכילה אך ורק Callback אחד.
- 4 הבתים הבאים הם ה-AddressOfCallbacks. באופן זהה נמיר את הערך בפנים ונקבל 403130. כלומר, מערך הפונקציות נמצא באופסט 130 שזאת התיבה האדומה. אם נסתכל בתיבה האדומה נראה כי היא מכילה בתחילתה כתובת נוספת שלאחר המרה מ-little endian נקבל 401C30. זוכרים? זאת בדיוק הכתובת של ה-Shellcode שלנו.
- ולבסוף 8 הבתים האחרונים בתוך ה-TLS Directory (התיבה הכחולה) הם ה-SizeOfZeroFill וה-Characteristics. אמרנו שגם להם אין חשיבות יתרה ולכן ניתן להשאיר אותם כאפסים.

כל שנותר כעת הוא לציין ב-PE שקיים בתוכו TLS Directory. אך כיצד נעשה זאת?

נחזור אל ה-Optional Header לשני שדות, TLS Directory RVA ו-TLS Directory Size. ב- TLS Directory RVA נכניס את הכתובת הרלטיבית של ה-TLS Directory דהיינו 3100 ו-TLS Directory Size נכניס את הגודל שלו שהוא 18, הגודל הגנרי.



נשמור את הקובץ, נפתח אותו מחדש באמצעות ה-Cff Explorer ונראה כי הוא מזהה לנו TLS Directory חדש.



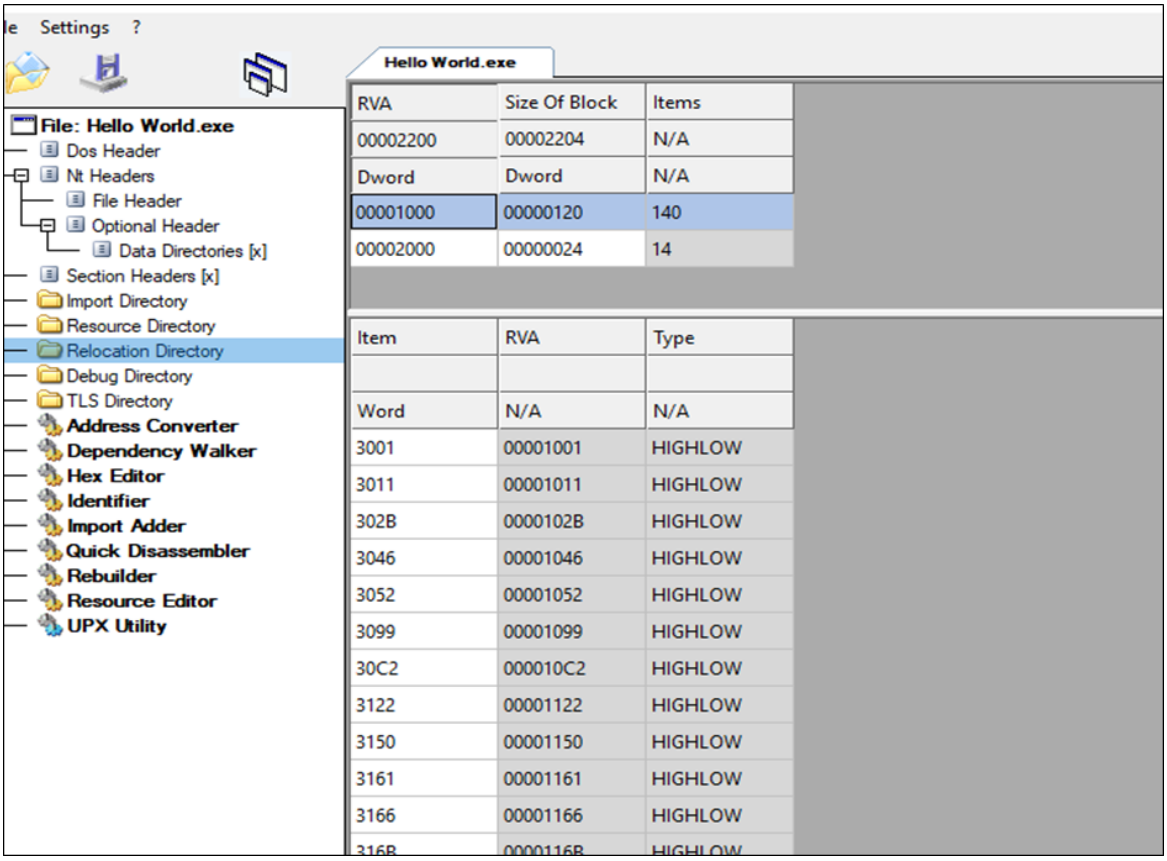
ולשלב האחרון, זוכרים שה-TLS Directory מכיל כתובת אבסולוטיות ולא רלטיביות? מה יקרה אם המודול לא יטען בכתובת 400000 בגלל ה-ASLR? כמובן שאותן כתובות כבר לא יהיו רלוונטיות יותר.

על מנת לטפל בזה כל שעלינו לעשות הוא להוסיף מידע נוסף ל-reloc Section שישמש את ה-Loader. למעשה ה-reloc Section מורכב מ-Relocation Tables שכל אחת מכילה מידע על כתובות בהן ה-Loader צריך לבצע תיקון בהתאם ל-Image Base החדש.

בסך הכל יהיו שלוש כתובות אותן נרצה לתקן:

- AddressOfIndex
- AddressOfCallBacks
- הכתובת של הפונקציה הראשונה במערך ה-Callbacks.

הערה: על מנת לראות את המידע אודות ה-Relocations שקיימים כעת, ניתן להעזר ב-Relocation Directory שנמצא בתפריט הצד ב-CFF Explorer. לצורך השינויים עצמם אשתמש ב-Hex Editor פשוט על מנת להוסיף את המידע:



כל Relocation Table מוגדר באופן הבא:

Offset (4 bytes)	Size (4 bytes)	Relocation (2 bytes)	Relocation (2 bytes)	...

Offset - הכתובת הרלטיבית הוירטואלית של הבלוק בו בזיכרון בו אנו מעוניינים לתקן כתובת.

Size - הגודל של ה-Relocation Table.

Relocation - רשומה בטבלה, מתואר מתחת.

כל Relocation מוגדר באופן הבא:

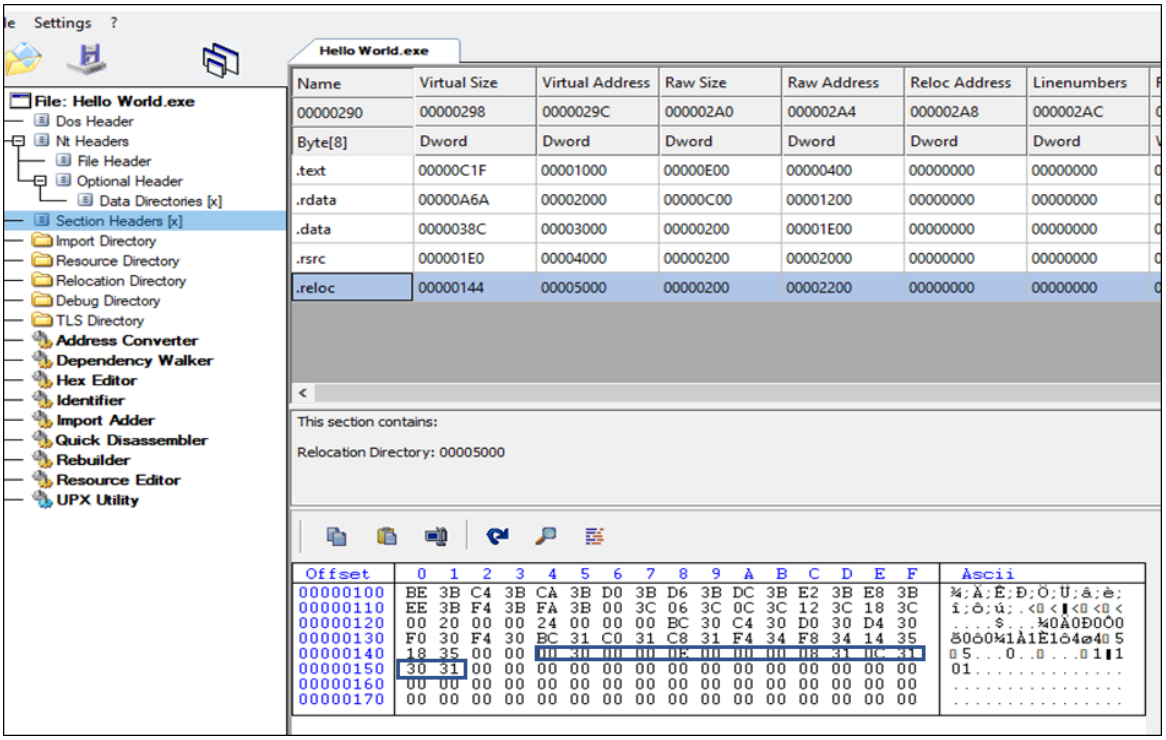
Type (0.5 byte)	Internal Offset (1.5 bytes)

Type- הסוג של ה-Relocation. לרוב יהיה זה 3 שמשמעותו: IMAGE_REL_BASED_HIGHLOW.

Internal Offset - האופסט בתוך הבלוק בו יש כתובת שנרצה לתקן. למעשה הכתובת אותה ה-Loader יתקן תחושב בתור: Offset + Internal Offset.

כעת נבנה את ה-Relocation Table המתאים עבור ה-TLS Directory שלנו. ה-TLS Directory שלנו נמצאת בתוך ה-data Section, לכן האופסט יהיה 3000.

הגודל יהיה 4 בתים עבור ה-Offset ו-4 בתים עבור ה-Size. ולסיום, 2 בתים עבור כל רשומה (קיימות 3 רשומות). סה"כ: 0xE = 14 = 8 + 2*3. להלן החלק אותו הוספנו ב-reloc Section:



ולסיום, חשוב להגדיל ב-0xE את השדה Relocation Table Size שנמצא בתוך ה-Optional Header.

Self-Modifying TLS Callbacks

כל מה שסיפרתי לכם עד כה לא היה חדש כל כך ולשימחתנו גם לא היו חסרים כלים שידעו להתמודד עם זה. ועכשיו, לסיום המאמר ברצוני להוסיף קונספט נוסף שאני לא הכרתי קודם לכן (ומקווה שכך גם אתם) ועשוי להקשות על עבודתו של החוקר - Self-Modifying TLS Callbacks.

כעת, כדי לחזור לאווירה, דמיינו עולם קודר, אפל ודמיוני, שבו כותבי Malwares מעוניינים ליצור Malwares שמקשים על החוקר! ואף מצליחים לשבש את פעולת המחקר שלו! כותב Malwares דמיוני ולא סביר בעליל שכזה עשוי לחשוב על הרעיון הבא:

נניח למשל שיש ברשתנו תכנית בעלת TLS Callback אחד, ה-TLS Callback לכאורה לא עושה שום דבר זדוני: לא בודק האם קיים דיבאגר, לא בודק סביבת מחקר ואפילו לא מנסה לגנוב את הפרטים שלכם לבנק. אולם דבר אחד הוא כן עושה - הוא מייצר TLS Callback חדש. IDA כמובן לא תזהה את אותו TLS Callback שכן באופן סטטי היא רואה שקיים רק TLS Callback אחד בזיכרון. אדרבא, אפילו אתם שמכירים כיצד נראים TLS Callbacks בזיכרון רואים שקיימת רק כתובת אחת במערך ה-TLS Callbacks.

ואכן, לשימחתו של אותו כותב Malwares מרושע, התכנית הזדונית שלו אף תצליח להתממש. מסתבר שה-Loader לא בודק את מספר הפונקציות שהיו במערך עם תחילת טעינת הקובץ לזיכרון. לכן, לאחר

ריצת ה-TLS Callback הראשון, ה-Loader ימשיך לפונקצייה הבאה במערך - כזאת שלא בהכרח הייתה שם בזמן תחילת ריצת התכנית.

לצורך ההדגמה, בניתי פונקציה שמבצעת בדיוק את הנאמר לעיל, היא מקבלת כתובת של פונקציה ואינדקס ומוסיפה באופן דינאמי למערך ה-TLS Callbacks כתובת חדשה. להלן הפונקציה:

```
BOOL create_TLS_callback(int index, DWORD functionAddress) {
    PIMAGE_DOS_HEADER       dosHeader;
    PIMAGE_NT_HEADERS       ntHeader;
    PIMAGE_OPTIONAL_HEADER  optHeader;
    PIMAGE_TLS_DIRECTORY    tlsDirectory;
    PDWORD                  callbackArray;
    DWORD                   tlsDirectoryOffset;
    DWORD                   lpflOldProtect;

    dosHeader = (PIMAGE_DOS_HEADER)GetModuleHandleA(NULL);
    ntHeader = (PIMAGE_NT_HEADERS)((PBYTE)dosHeader + dosHeader->e_lfanew);
    optHeader = &(ntHeader->OptionalHeader);
    tlsDirectoryOffset = *(PDWORD)((DWORD)optHeader + 168);
    tlsDirectory = (PIMAGE_TLS_DIRECTORY)((DWORD)dosHeader + tlsDirectoryOffset);
    callbackArray = (PDWORD)(tlsDirectory->AddressOfCallBacks);
    VirtualProtect((LPVOID)(callbackArray + index*4), 4, PAGE_READWRITE, &lpflOldProtect);
    (callbackArray)[index] = (DWORD)functionAddress;
    return TRUE;
}
```

ننתח הפונקציה לעיל:

```
dosHeader = (PIMAGE_DOS_HEADER)GetModuleHandleA(NULL);
ntHeader = (PIMAGE_NT_HEADERS)((PBYTE)dosHeader + dosHeader->e_lfanew);
optHeader = &(ntHeader->OptionalHeader);
tlsDirectoryOffset = *(PDWORD)((DWORD)optHeader + 168);
tlsDirectory = (PIMAGE_TLS_DIRECTORY)((DWORD)dosHeader + tlsDirectoryOffset);
```

תחילה נשיג באמצעות GetModuleHandle, Handle למודול שאנו רצים כחלק ממנו, ה-Handle הוא למעשה הכתובת שממנה טעון הקוד שלנו ומשם מתחיל גם ה-Header שלו.

בהמשך נרוץ על ה-Header עד שנגיע אל ה-Optional Header. בתוכו ניגש TLS Directory RVA שזה הוא למעשה המשתנה tlsDirectoryOffset ובאמצעותו ניגש ל-TLS Directory עצמו.

```
callbackArray = (PDWORD)(tlsDirectory->AddressOfCallBacks);
VirtualProtect((LPVOID)(callbackArray + index*4), 4, PAGE_READWRITE, &lpflOldProtect);
(callbackArray)[index] = (DWORD)functionAddress;
```

לבסוף, נמצא את הכתובת של מערך ה-TLS Callbacks, נשנה את ההרשאות שלו לכתיבה (הרי נרצה להיות מסוגלים להוסיף לתוכו פונקציה חדשה) ולאחר מכן נשכתב פנימה את הפונקציה החדשה.

והנה קיבלנו את היכולת ליצור TLS Callbacks בזמן ריצה, כל זאת כאשר IDA איננה מסוגלת לומר לנו כי יש יותר מ-Callback אחד. נוסף על כך, בגלל שה-Loader הוא זה שמחליף בין פונקציות ה-TLS הדבר עשוי לבלבל חוקר שמדבג את התכנית.

אתגר

במסגרת כתיבת המאמר בניתי אתגרון שמממש את כל הכתוב לעיל. לצערי או לשימחתי, ככל הנראה למי שקרא את כלל המאמר הוא לא יהיה מורכב במיוחד. האתגר איננו מצריך הרצה תחת VM ועל מנת לפתור אותו כל שנדרש הוא למצוא את הסיסמא הנכונה. בהצלחה! ☺

קישור להורדה:

http://www.digitalwhisper.co.il//files/Zines/0x62/challenge.rar

למעוניינים יש גם קוד מקור של האתגר:

http://www.digitalwhisper.co.il//files/Zines/0x62/Source.7z

סיכום

TLS הננו מנגנון אשר מאפשר יכולת מעניינת מאוד אשר שימושית (ואף משומשת) בקרב כותבי Malwares רבים. במאמר זה למדנו כיצד TLS Callbacks בנויים וכיצד ניתן לנצל אותם לטובת הרצה של קוד זדוני. בנוסף לכך, למדנו טכניקה נוספת שמקשה על תהליך המחקר אף יותר.

על המחבר

יהונתן משמש כיום כחוקר חולשות, מתעניין רבות במחקר Malwares, קריפטוגרפיה, הנדסה לאחור ובחקר רכיבי Embedded.

Linkedin: https://il.linkedin.com/in/jonathan-lusky-b8b634130

ביבליוגרפיה / לקריאה נוספת

1. תכנות של TLS Callbacks:

http://lallouslab.net/2017/05/30/using-cc-tls-callbacks-in-visual-studio-with-your-32-or-64bits-programs/

2. TLS Callbacks:

https://legend.octopuslabs.io/archives/2418/2418.htm

3. Self-Modifying TLS Callbacks:

http://www.openrce.org/blog/view/1114/Self-modifying_TLS_callbacks

4. MSDN - Thread Local Storage:

https://docs.microsoft.com/en-us/windows/desktop/procthread/thread-local-storage

5. .reloc Section:

https://ntcore.com/files/inject2exe.htm