Technologic Papers

המדריך המקיף ל-DLL-Sideloading

מאת ניר גילס

הקדמה

אני אוהב פוגענים. תוכנות קטנות מאוד ביחס לממוצע התוכנות הלגיטימיות בעולם, אך מלאות בחידושים שמאפשרים לתוקפים להשיג את מטרתם. חלק מהחידושים הם טכנולוגיים, חולשות מטורפות וטכניקות שמשנות את פני המשחק, וחלקם הם פשוט דרך יצירתית אך פשוטה לעקוף מנגנוני הגנה או להערים על מישהו לפתוח ולהפעיל אותן בשוגג.

הדרך המועדפת עליי לשמוע על פוגענים היא במאמרים של חברות הגנה כאלה ואחרות, לרוב לאחר קריאה ברפרוף ב-The Hacker News[1] תוך כדי התעמקות במאמרים המקוריים של מה שנשמע מעניין. בקריאות שלי בשנה וקצת האחרונות הצלחתי כבר לנחש בסבירות גבוהה כשמדובר בקבוצת תקיפה סינית על פי טכניקה שתופיע במאמר. במשך התקופה הזו הם כמעט תמיד השתמשו בטכניקה בשם DLL-Sideloading, והיו גם קבוצות תקיפה אחרות שעשו בה שימוש! אם כל כך הרבה תוקפים משתמשים בטכניקה הזו, חשבתי שיהיה שווה לבדוק עליה לעומק ☺

במאמר הזה אני הולך לפרט על טכניקת התקיפה שהיא DLL-Sideloading - נעבור את המסלול המלא למימוש של התקיפה, כולל שימוש בכלי עזר שבניתי בשם Nihilego. לאחר מכן נסקור מספר יתרונות בשימוש בטכניקה זו לתקיפות, ונסקור 2 תקיפות שבוצעו על ידי גופי תקיפה מעצמתיים (רוסיה וסין) המשתמשות בטכניקה זו. לבסוף נדבר על המלצות למגנים בהיבטי מניעה, גילוי וחקירה של תקיפות המכילות את הטכניקה הזו. אני לא אפרט על DLL-ים ואופן השימוש שלהם במ.ה. Windows, מומלץ לקרוא עליהם בקצרה מראש[2,3]. יהיו מעט קטעי קוד בשפת C - אין צורך בהבנת הקוד על מנת לקרוא את המאמר, אך זה עוזר להמחיש את אופן המימוש של הטכניקה מצד התוקף.

כל המקורות בהם השתמשתי, שחלקם יעמיקו באלמנטים מסוימים מעבר למאמר זה, יצורפו ברשימת המקורות בסוף המסמך ויוזכרו בחלקם תוך כדי המאמר.

---

[1] https://thehackernews.com/
[2] https://en.wikipedia.org/wiki/Portable_Executable
[3] https://learn.microsoft.com/en-us/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library

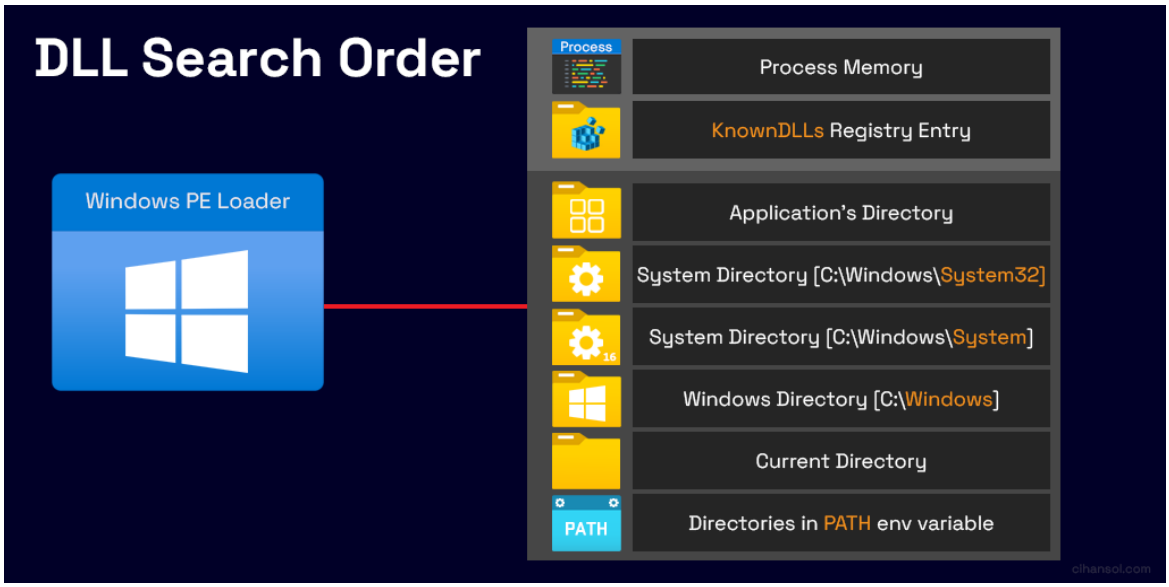
איך מתקפת DLL Sideloading עובדת?

כאשר תוכנה מייבאת DLL בצורה דינאמית (מחפשת את הקובץ בזמן ריצה, בניגוד לשימוש בקימפול סטאטי שמכניס את ה-DLL לתוך ה-EXE עצמו)[4] ומשתמשים בפונקציות שלה, יש מספר שלבים שמתבצעים מאחורי הקלעים.

מציאת ה-DLL

מציאת ה-DLL מתבצעת ע"י חיפוש בסדר היררכי קבוע[5]:

1. חיפוש בזיכרון - האם ה-DLL כבר נטען בעבר? אם כן, אין צורך לחפש אותו בדיסק
2. חיפוש ב-Registry לרשימת DLL-ים מוכרים (Known DLLs, דומה ל-"/etc/ld.so.preload" ב-Linux)
3. חיפוש בתיקייה שממנה ה-EXE הורץ
4. חיפוש ב-C:\Windows\System32
5. חיפוש ב-C:\Windows\System
6. חיפוש בערכי ה-PATH שהוגדרו ע"י המשתמש
7. חיפוש ב-Current Directory



[מקור: https://cihansol.com/blog/index.php/2021/09/14/windows-dll-proxying-hijacking]

בכל פעם התוכנה מנסה את מזלה - אם היא לא מוצאת את הקובץ בשם הנכון במיקום הנוכחי אז היא מנסה במיקום הבא.

[4] https://learn.microsoft.com/en-us/windows/win32/dlls/about-dynamic-link-libraries
[5] https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order

דוגמה לטעינה דינאמית וסדר חיפוש אחר DLL

הנה קוד של תוכנה פשוטה בשם legitExe שמנסה לייבא DLL בשם a.dll. התוכנה משתמשת בפונקציית "helloWorld" של a.dll כדי להדפיס למסך את המשפט "Hello, Digital Whisper!" (תודה ל- ChatGPT):

C: > digital whisper > C legitExe.c

```c
// legitExe.c - The EXE source code
#include <windows.h>
#include <stdio.h>

typedef void (*HelloWorldFunc)();

int main() {
    HMODULE hDll = LoadLibrary("a.dll");
    if (!hDll) {
        printf("Failed to load a.dll\n");
        return 1;
    }

    HelloWorldFunc helloWorld = (HelloWorldFunc)GetProcAddress(hDll, "helloWorld");
    if (!helloWorld) {
        printf("Failed to find helloWorld in a.dll\n");
        FreeLibrary(hDll);
        return 1;
    }

    helloWorld();

    FreeLibrary(hDll);
    return 0;
}
```

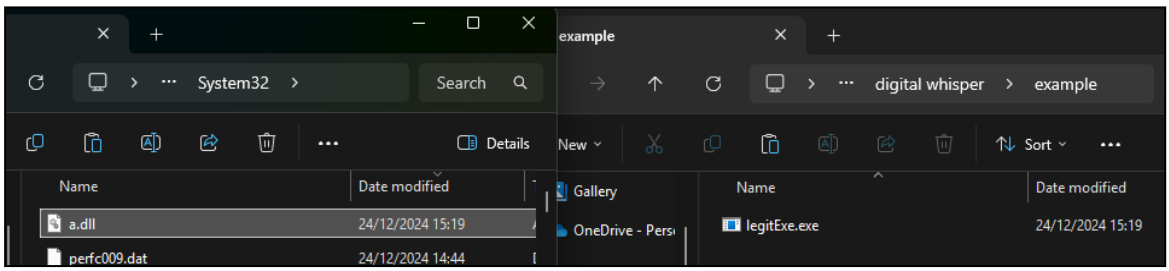
[הקוד של legitExe. מוקף באדום - ייבוא ה-DLL בצורה דינאמית]

הינה הקוד של a.dll, עם פונקציה יחידה מוחצנת בשם helloWorld המדפיסה את המשפט " Hello, Digital Whisper!" למסך:

C: > digital whisper > C a.c

```c
// a.c - The DLL source code
#include <stdio.h>

__declspec(dllexport) void helloWorld() {
    printf("Hello, Digital Whisper!\n");
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD dwReason, LPVOID lpReserved) {

    switch (dwReason)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

נריץ את legitExe מהנתיב "C:\digital whisper\example", ה-a.dll נמצא בתוך "C:\windows\system32":



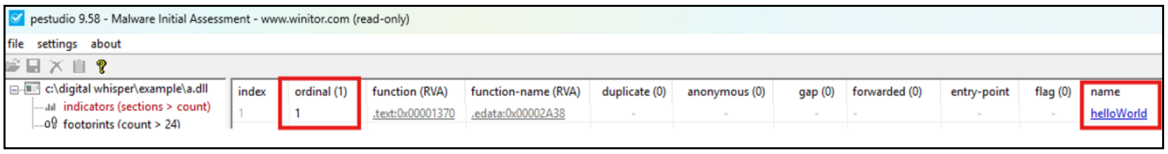
[ה-DLL ב-system32, וה-EXE ב-C:\digital whisper\example]

תהליך החיפוש של ה-DLL ייראה כך:

1) חיפוש אם ה-DLL נטען קודם לכן ונמצא בזיכרון של התוכנית -> **לא קיים**
2) חיפוש אם ה-DLL נמצא ב-registry של Known DLLs -> **לא קיים**
3) חיפוש של "C:\digital whisper\example\a.dll" -> **לא קיים**
4) חיפוש של "C:\windows\system32\a.dll" -> **קיים!**

מצאנו את ה-DLL, איך מוצאים את הפונקציה?

לרוב טוענים DLL על מנת להשתמש בפונקציות המוחצנות שלו (exported functions). אם נמשיך את הדוגמה של a.dll, אנחנו מחפשים את הפונקציה helloWorld. כדי למצוא את הפונקציה, התוכנה יכולה לחפש את השם של הפונקציה או את המספר שלה (Ordinal). אם השם או המספר שאנחנו ננסה לייבא לא קיימים, הטעינה תחזיר שגיאה:



ניתן לראות מוקף באדום את שם הפונקציה מימין, ואת המספר (Ordinal) שלה משמאל. התמונה נלקחה מתוך תוכנה בשם pestudio, תוכנה חינמית שמשמשת בעיקר לחקר פוגענים, ומאפשרת בין היתר לראות exported functions של dll-ים.

בואו נריץ את legitExe.exe כדי לוודא שהכל עובד כנדרש:

```
C:\digital whisper\example>legitExe.exe
Hello, Digital Whisper!

C:\digital whisper\example>
```

הוא אכן מצליח לאתר את ה-DLL ועובד כפי שציפינו.

כוונות רעות

המלצה שלי - תקראו שוב את החלקים של המאמר עד לשלב הזה, ותחשבו לבד על איך אפשר להשתמש במידע הזה על מנת להריץ קוד מתוך תוכנה לגיטימית, לדוגמה legitExe.exe, בלי לשנות את הקוד של ה-EXE.

סיימתם? התעצלתם? בואו נמשיך ☺

אם תוכנה מייבאת DLL בצורה דינאמית, לדוגמה את a.dll כפי שתואר קודם לכן, ואני בחור רע עם גישה למחשב שעליו רצה התוכנה, אני יכול לשים DLL משלי בתוך התיקייה של התוכנה (של הקובץ EXE, במקרה שלנו בתוך "C:\digital whisper\example") ולקרוא לו a.dll. אם נחזור לסדר החיפוש של ה-DLL, נגלה שהתוכנה עכשיו תמצא את ה-DLL הזדוני שלנו לפני ה-DLL האמיתי אותו היא הייתה אמורה למצוא ב-system32!

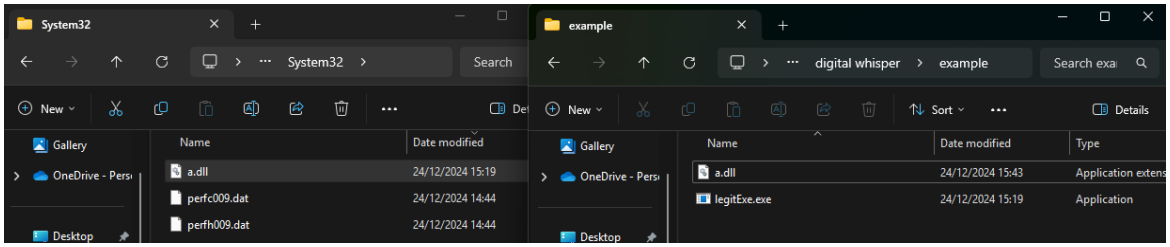
בואו נוכיח כמה זה פשוט:

הנה קוד מינימלי של DLL שרק מקפיץ popup למסך לאחר שהוא מפסיק שימוש בו (עושה לו detach, זה על מנת שה-popup לא יעכב את שאר התוכנית):

C: > digital whisper > C malicious.c

```c
#include <windows.h>

void showPopup() {
    MessageBox(NULL, "Sideloaded!", "It's that easy", MB_OK | MB_ICONINFORMATION);
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD dwReason, LPVOID lpReserved) {

    switch (dwReason)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
        break;
    case DLL_PROCESS_DETACH:
        showPopup();
        break;
    }
    return TRUE;
}
```

ועתה נשים את ה-DLL (המקופל כמובן) בתוך התיקייה של example יחד עם התוכנה שלנו, ונקרא לו "a.dll" (אחרי הכל, ה-EXE מחפש לפי שם):



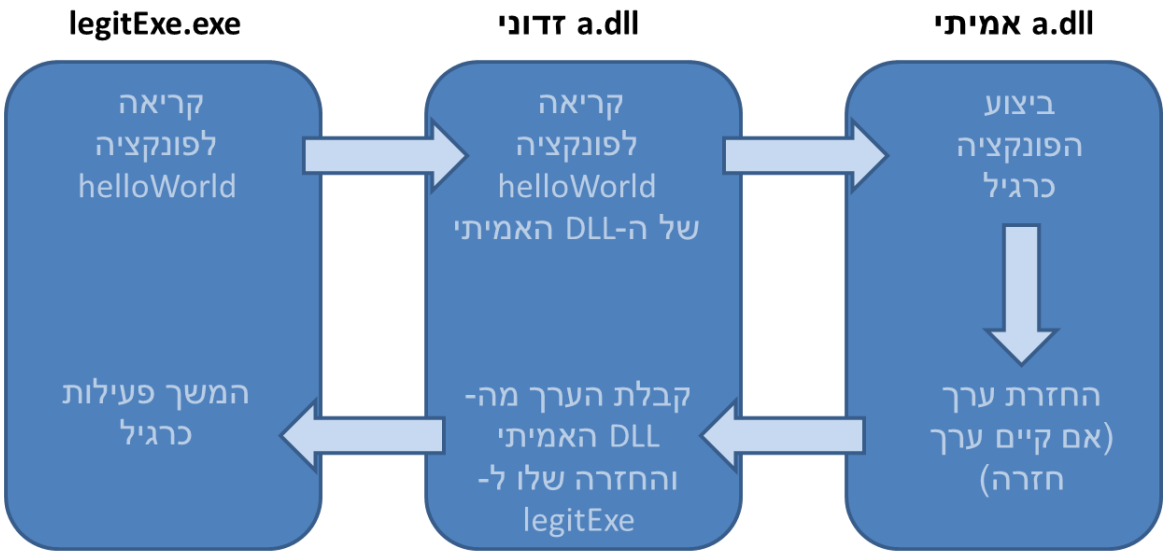
הוספנו את ה-DLL ה"זדוני" לתיקייה של legitExe, ה-DLL הלגיטימי עדיין נמצא ב-system32. ונריץ שוב:

התוכנה הריצה את ה-DLL שלנו (רואים את ה-popup), אבל התוכנה קרסה ולא הדפיסה "Hello Digital Whisper!" במקרה שלנו זאת כל התוכנית - במקרה של תוכנות אחרות, זה יכול להקריס את התוכנה בתחילתה או באמצע הריצה. תדמיינו שה-word או ה-chrome שלכם קורס לאחר פתיחה או באמצע השימוש שוב ושוב - חשוד!

ובגלל שזה חשוד מי שכותב פוגענים לא רוצה להקריס את התוכנה שרצה, אחרת יעלו עליו ויעצרו לו את התקיפה. על מנת שהתוכנה לא תקרוס צריך לייצא את הפונקציות עם שמות ומספרי Ordinal נכונים כדי לאפשר לתוכנה למצוא את הפונקציה שהיא צריכה, וגם לעשות את כל מה שהפונקציה המקורית עושה. אבל רגע, גם פה יש בעיה: איך אני, כותב ה-DLL הזדוני, יכול לדעת בדיוק מה ה-DLL המקורי עושה? אם אני אתנהג בצורה שלא תואמת את ה-DLL המקורי אני עלול להקריס את התוכנה או לגרום להתנהגות לא רצויה. ישנן 3 פתרונות כלליים:

1) ל-DLL המקורי יש תיעוד מלא, ולכן אני יכול לממש את הפונקציונליות שלו במלואו בתוך הקוד שלי בלי יותר מידי מחשבה ובעיות אבל עם הרבה עבודה.
2) ל-DLL המקורי אין תיעוד בכלל, אבל אני יכול לרוורס (Reverse) את הקובץ וליישם את הפונקציונליות שלו בעצמי, אבל אני צריך להשקיע בזה הרבה זמן ומאמץ.
3) אני אקרא ל-DLL המקורי מתוך ה-DLL הזדוני! ה-DLL שלי הוא גם תוכנה, והתוכנה שלי יכולה לייבא קוד של DLL אחר, וכך אני יוצר מעין Proxy או Man in the Middle שבה כל בקשה של התוכנה עוברת דרך הקוד הזדוני שלי, שקורא לקוד האמיתי של ה-DLL, שמחזיר את התשובה ל-DLL הזדוני שלי ומשם חזרה לתוכנה (ראו את האיור למטה).

וכך בתרשים:



לעיתים יכולות להיות עשרות פונקציות - ולעשות את זה בצורה ידנית לוקח הרבה זמן. לכן בואו נראה דוגמה באמצעות כלי שכתבתי - Nihilego, המאפשר לבצע את ה-proxy בצורה נוחה. הכלי Nihilego מקבל כפרמטר קובץ DLL שנרצה לבצע אליו proxy, ושם שיינתן ל-DLL המקורי:

```
C:\digital whisper>py .\Nihilego_v1.py
Usage .\Nihilego_v1.py [-p | --proxy = DLL to proxy, -n | --name = new name for proxied dll]

C:\digital whisper>py .\Nihilego_v1.py -p "C:\\windows\\system32\\a.dll" -n "b.dll"
[+] Generating usage testers for target DLL C:\\windows\\system32\\a.dll
[+] Making the dllmain file
```

דוגמה לשימוש בכלי Nihilego

ככה נוצרה תיקייה עם 3 קבצים:

1) קובץ ה-DLL המקורי אליו נרצה להפנות את הפונקציות, בשם החדש כפי שהגדרנו לו (b.dll במקרה של הדוגמה למעלה).
2) קובץ קוד C, שנוכל לקמפל באמצעותו את ה-DLL הזדוני שלנו.
3) קובץ Source.def, המשמש אותנו בקימפול כדי להחצין את הפונקציות[6] (לא ארחיב על כך במאמר זה).

[6] https://learn.microsoft.com/en-us/cpp/build/exporting-from-a-dll-using-def-files?view=msvc-170

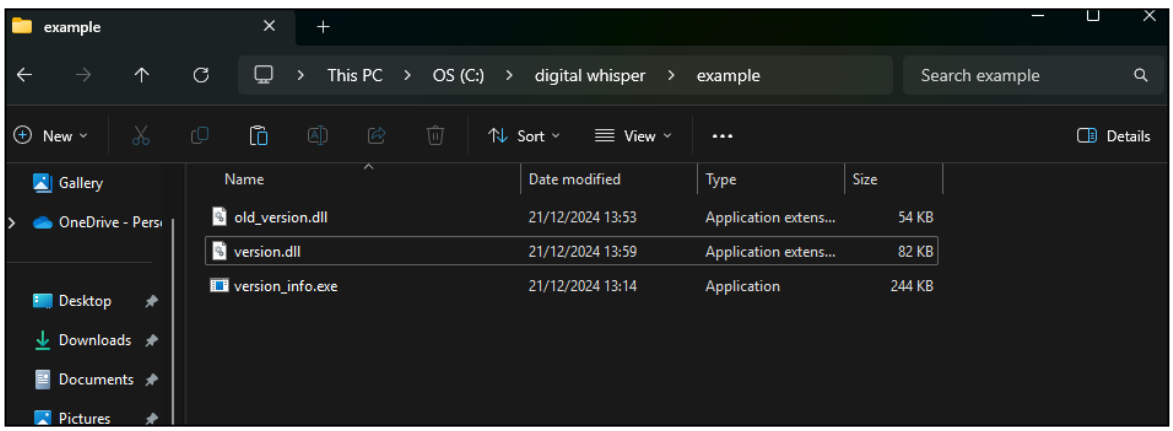
לאחר קצת ניקיון של חלקים בקוד שיוסברו בהמשך, נראה שבעצם התווספה לנו רק שורה אחת מעניינת מוקפת באדום:

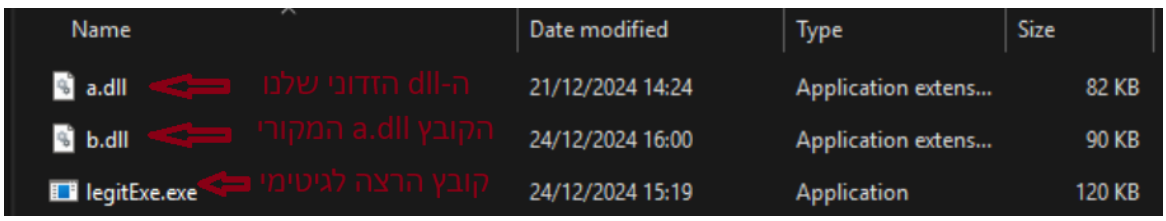
```c
#include "windows.h"

#pragma comment(linker,"/export:helloWorld=b.helloWorld")

void ExecutePayload()
{
    MessageBoxW(0, L"Payload Executed!", L"dll sideloading example", MB_OK);
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
        break;
    case DLL_PROCESS_DETACH:
        ExecutePayload();
        break;
    }
    return TRUE;
}
```

[תמונה של הקוד *C* שנוצר בעקבות שימוש בכלי Nihilego]

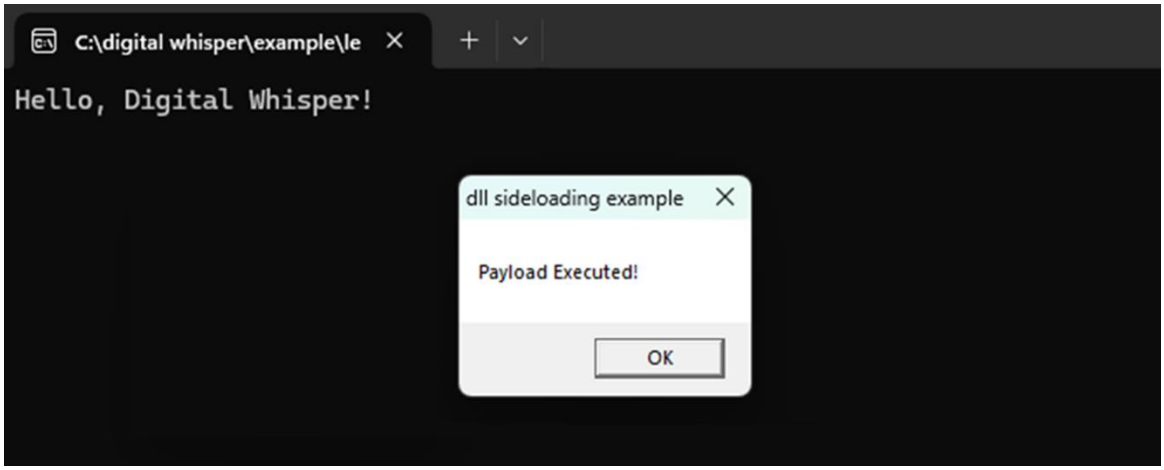
לחדי העין, התמונה נלקחה מתוך Visual Studio ולא מתוך VSCode. זאת נקודה קריטית למי שמנסה לשחזר את הנעשה כאן בעצמו - מאחר וההנחיה של "pragma comment" בלעדית ל-msbuild (הקימפול הדיפולטיבי של Visual Studio) ולא תעבוד ב-GCC, ותאמינו לי שניסיתי מספר דרכים אחרות... אין מנוס (ואם קיים מנוס ומצאתם אותו - אשמח שתשלחו אותו אליי!).

השורה הזאת בעצם אומרת: כל קריאה ל-DLL הזה, לפונקציה הזו (במקרה הזה - helloWorld) - תועבר לפונקציה אחרת (במקרה שלנו, עדיין helloWorld) ב-DLL אחר (במקרה שלנו, b.dll). ה-b.dll הוא בעצם העתק של a.dll המקורי בשינוי שם שנשים באותה התיקייה יחד עם a.dll המזויף שלנו, ואליו נפנה את כל הבקשות של התוכנה (באותה קלות היינו יכולים להפנות גם ל-a.dll המקורי ב-system32 באמצעות קריאה לנתיב המלא שלו). כך בעצם תיראה התיקייה לקראת השימוש בתקיפה:





אז לסיכום: a.dll הוא ה-DLL שלנו (הזדוני), וה-b.dll הוא ה-DLL האמיתי אליו אנחנו נפנה את הבקשות. בואו נריץ ונראה שאכן הכל עובד:



מגניב! הצלחנו גם להריץ את ה-EXE המקורי וגם את ה-DLL שלנו בלי לקרוס, באמצעות הפנייית הקריאות לפונקציה ל-DLL המקורי!

חיפוש אחר DLL-Sideloading בטבע

איך מוצאים תוכנה פגיעה?

לכתוב בעצמנו קוד פגיע זה נחמד והכל, אבל איך באמת מוצאים תוכנה שפגיעה ל-DLL-Sideloading, אחת שאין לנו את הקוד מקור שלה? אז האמת שזה לא קשה בכלל. כל מה שצריך זה את התוכנה procmon.

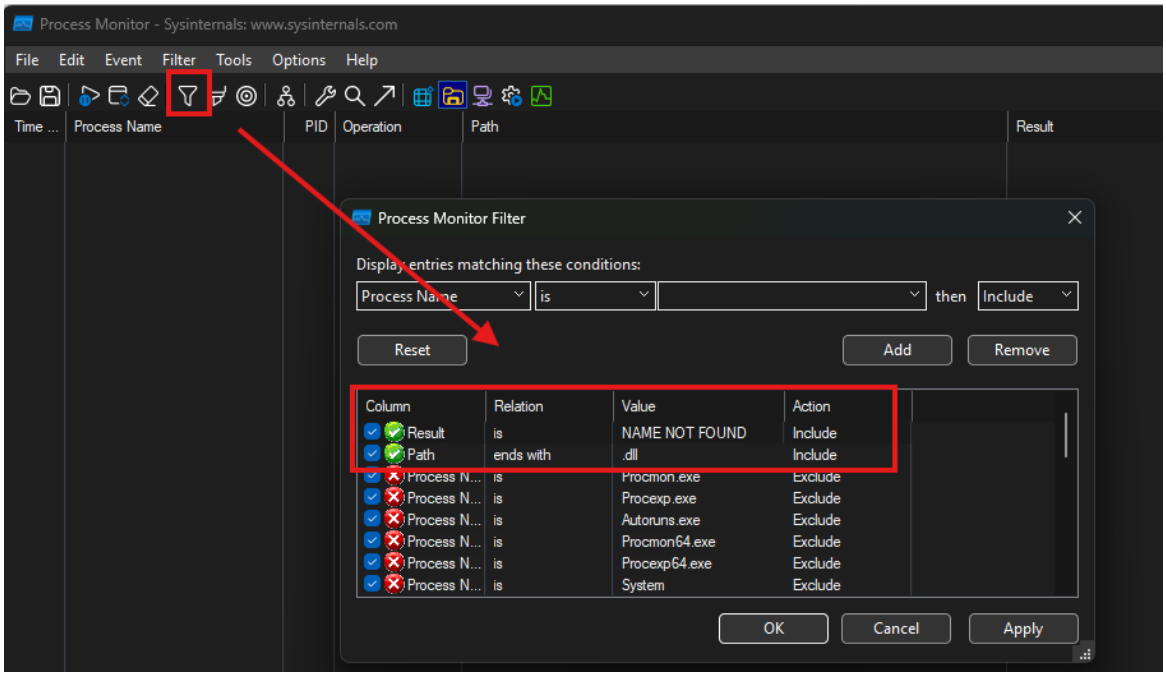
לכל מי שלא מכיר - procmon או process monitor היא תוכנה חינמית מתוך סל הכלים sysinternals שעוזרת לחוקרים לראות כל מיני פעולות שתוכנות עושות כדי לחפש פוגענים או לחפש בעיות תפעוליות כאלה ואחרות במערכת ההפעלה Windows. אפשר לחשוב על procmon כמו על Event Viewer על סטרואידים, המאפשרת להקליט את כל מה שנעשה ולראות גם לאחור איזה תוכנות נפתחו, נסגרו, ואיזה פעולות נעשו, לרמת פירוט יחסית גבוהה.

לאחר שנפתח את procmon אנחנו נהיה מוצפים בלוגים שונים. 99.9999% מהם לא יעניינו אותנו, ולכן נרצה לפלטר אותם ולראות בנוחות רק את מה שרלוונטי. נכנס ללשונית ה-Filter ונוסיף 2 דרישות:

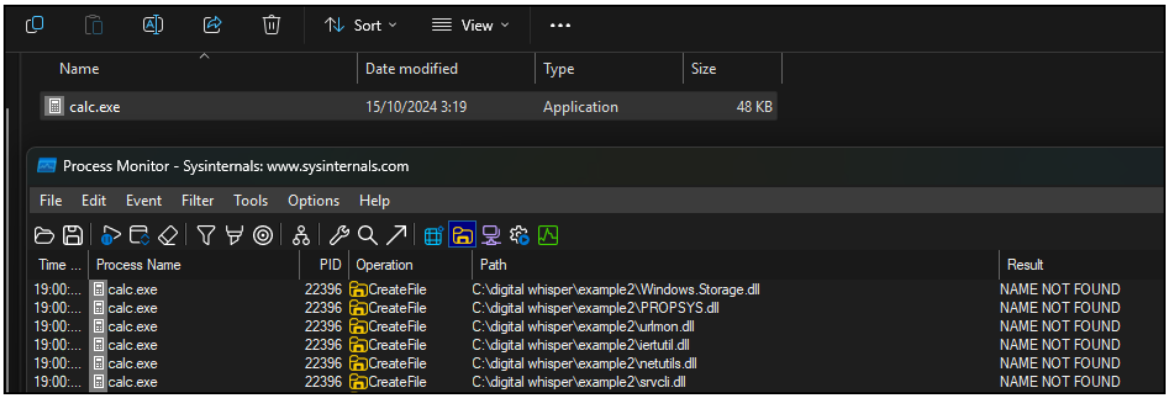
1) **שהתוצאה של החיפוש (RESULT) יהיה שווה ל-NAME NOT FOUND:** זה הלוג שנשמר כאשר תוכנה מנסה לגשת לקובץ שלא קיים. בכל פעם שהתוכנה שנבדוק תיגש ל-DLL שלא קיים היא תשאיר את הלוג, וכך נוכל לראות לאיזה DLL-ים מתאפשר לנו (פוטנציאלית - יורחב בהמשך) להתחזות אליהם.

2) **ש-PATH, או נתיב החיפוש, יכיל בסופו את המילה DLL,** אחרת אנחנו נראה גישה לקבצים שאינם קשורים.

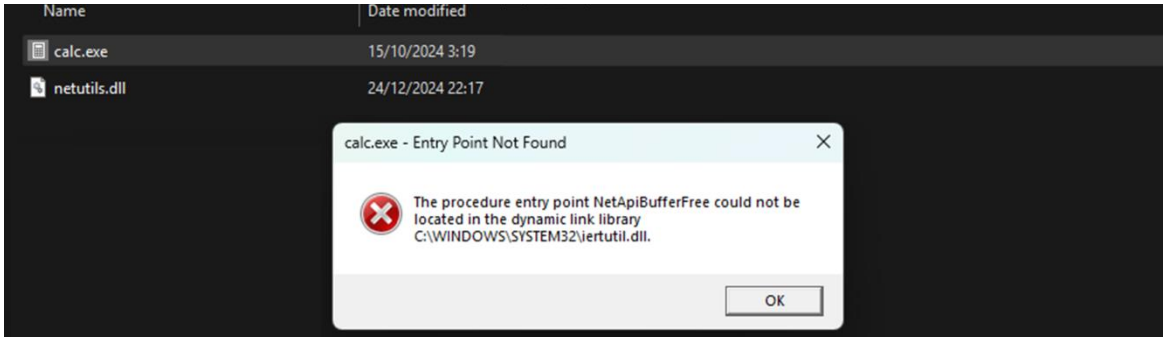
זה נראה כך:



לצורך ההדגמה, ניקח את ה-EXE האהוב על חוקרים להוכחת יכולת - calc.exe, ונבדוק מה הוא טוען. נוציא את הקובץ ממקומו הרגיל ב-C:\windows\system32 ונשים אותו בתוך תיקייה כלשהי לבחירתנו, במקרה הזה אשים אותו בתוך C:\digital whisper\example2. לאחר מכן נריץ אותו עם procmon פועל ברקע, ונראה מה הוא יניב לנו:



ניתן לראות ב-procmon לוגים ש-calc.exe השאיר לאחר הסינון. מופיעים מספר dll-ים שהוא מחפש ולא מוצא, ומיקום החיפוש שלהם. אז בואו ניקח לדוגמה את netutils.dll - נקמפל מחדש את malicious.c (ה-dll שרק מקפיץ popup, בלי שום הוקוס פוקוס אחר) וננסה את מזלנו:

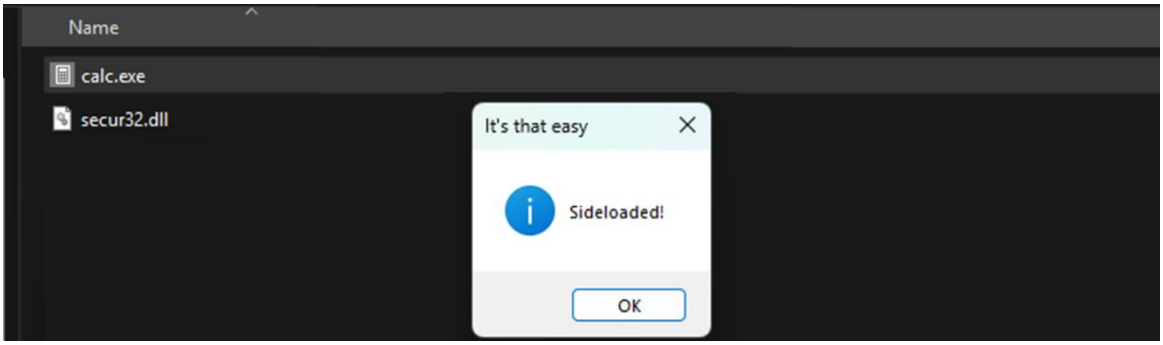


זאת בעיה חדשה, אפילו ה-popup שלנו לא הופיע... מסתבר שה-DLL לא מתאים.

מה זה DLL מתאים, ואיך מוצאים אותו?

אז בדיוק כמו שאנחנו עושים proxy ובעצם מסתמכים על DLL אחר מתוך ה-DLL שלנו, גם DLL-ים אחרים לפעמים מסתמכים על עוד DLL-ים. התלות הזאת גוררת דרישה לסדר טעינה ספציפי, שאנחנו יכולים לתקן באמצעות הוספת קבצים מתאימים עד שכל ה-DLL-ים יטענו לזיכרון בסדר שהתוכנה דורשת. עם זאת - זה עוד קבצים, משמע עוד עקבות והזדמנויות לצד הכחול לעלות על הקוד הזדוני שלנו, ולכן זה פחות טוב (נבין זאת יותר לעומק בחלק של המאמר המפרט על הצד הכחול).

לכן עדיף להמשיך ולבדוק איזה DLL קיים ללא תלות ב-DLL-ים אחרים באמצעות ניסוי וטעייה. לאחר כמה נסיונות (החלפת השם של ה-DLL ליתר ה-DLL-ים שקפצו לנו ב-procmon) הגעתי לתוצאה הזו:



מסתבר ש-secur32.dll מתאים לנו בלי דרישות מקדימות, ונוכל לחזור חלילה על כל התהליך שביצענו עם a.dll ו-legitExe.exe ונגיע לאותה תוצאה שבה אנחנו מריצים את הקוד הזדוני שלנו בתוך calc.exe, בזמן שהוא רץ כרגיל.

במקרה ותרצו לדעת על תוכנות מוכרות שיש להן חולשת dll-sideloading אז יש מאגרים נחמדים בדיוק לכך.

מגבלה מרכזית בטכניקה - DLL Loader Lock

הקטע הזה מעט מתקדם ולא הכרחי על מנת להבין את המשך המאמר. מדובר במגבלה טכנית במימוש הטכניקה, שמגביל את הצורה שבה ניתן להריץ את הקוד שלנו. אם אתם מתכוונים להתעמק במימוש התקיפה בצורה מלאה או להיכנס לנבכי הדרכים לייצר מנגנונים לגלות אותה - אני ממליץ בחום להמשיך לקרוא.

עד כה הרצנו את הקוד שלנו בתוך ה-dllmain, כפי שניתן לראות בתמונה כאן את המיקום של ExecutePayload:

```
5   void ExecutePayload()
6   {
7       MessageBoxW(0, L"Payload Executed!", L"dll sideloading example", MB_OK);
8   }
9
10  BOOL APIENTRY DllMain(HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved)
11  {
12      switch (ul_reason_for_call)
13      {
14      case DLL_PROCESS_ATTACH:
15      case DLL_THREAD_ATTACH:
16      case DLL_THREAD_DETACH:
17          break;
18      case DLL_PROCESS_DETACH:
19          ExecutePayload();
20          break;
21      }
22      return TRUE;
23  }
```

אבל לצערנו (או לשמחתנו, תלוי באיזה צד אתם) - המצב לא כזה פשוט, ואנחנו חד משמעית לא רוצים לכתוב קוד בתוך **DLL main** וזאת בעקבות הדרך בה DLL-ים נטענים לתוך תוכנה.

הבעיה מוכרת מאוד לכל מי שניסה לתכנת תוכנה שרצה בצורה מקבילית לתוכנות נוספות (בין אם מספר תוכנות שמנסות לגשת לאותו קובץ או אותו מידע בו-זמנית, או מספר threads שעושים זאת). בעצם, אם גם תוכנה A וגם תוכנה B מנסים לכתוב לתוך קובץ "בו-זמנית", איך מתפעלים את זה? על רגל אחת, יש מקל קסמים שנקרא Mutant או Mutex, שמוגדר בצורה כזו בתוך המ.ה. שרק תהליך אחד יכול להחזיק בו בכל רגע נתון - מובטח לנו שלא ייתכן שום מצב שבו 2 תוכנות יצליחו "לקחת אחיזה" או לנעול אותו בו"ז. לכן מגדירים mutex שאומר "אני כותב לקובץ!", ורק מי שמחזיק ונועל אותו מסוגל לכתוב לקובץ.

זה המצב הפשוט - אבל לפעמים ישנם מצבים יותר מורכבים שכוללים יותר מ-mutex אחד, ועלולים לגרום ל-Deadlock: מצב שבו תוכנית A נעלה את Mutex1 ומחכה לנעול את Mutex2, בעוד שתוכנית B נעלה את Mutex2 ומחכה לנעול את Mutex1. עכשיו שניהם יחכו לנצח.

אז מסתבר שכל פעם שתוכנה מגיעה למצב שבו היא מבצעת טעינה או ניתוק (Attach or Detach) מקובץ DLL - קיים Loader Lock שמקפיד ששני threads לא חלילה יטענו את אותו ה-DLL בו-זמנית ויגרמו

להתנהגות לא צפויה, וככלל שתהליך הטעינה והניתוק של DLL-ים יהיה חלק, יציב וסדור, כולל DLL-ים שמסתמכים על DLL-ים אחרים כפי שראינו קודם.

הבעיה - אם בטעות נריץ קוד שלא נכיר אותו היטב ולעומק, אנחנו עלולים בסבירות מאוד גבוהה לגרום ל-Deadlock עם ה-Loader Lock[7], וכפי שהבנו - זה יקריס את התוכנה, יגרור חקירה ויגלה את המבצע. מבאס. מה אפשר לעשות כדי לוודא שאנחנו לא גורמים לזה, 100% מהזמן?

ישנן 2 אפשרויות:

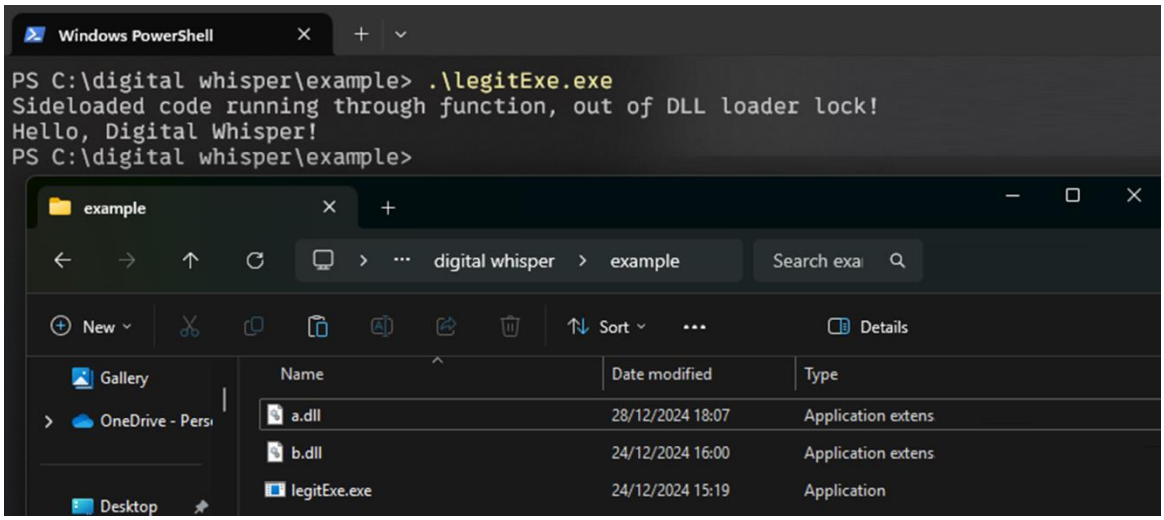
האחת - להריץ את הקוד מחוץ ל-dllmain. איך עושים את זה? בדרך הקשה. הזמן שבו ה-DLL רץ שלא בתוך dllmain הוא כאשר פונקציה מוחצנת שלו נקראת. אנחנו צריכים במקום לכתוב שורה של pragma comment שמעביר את הבקשה מיד ל-DLL האמיתי, ממש לכתוב את הפונקציה בעצמנו, לפי החתימה שהתוכנה מצפה לה, ולהחזיר ערך כפי שמצפים לו. נראה דוגמה נוחה במידה ויש לנו את הקוד מקור (באמצעות ה-DLL שכתבתי לתחילת המסמך, עם helloWorld), אבל במידה ואין לנו את החתימה המקורית של הפונקציה (לדוגמה כשה-DLL המקורי לא מתועד) אז צריך ממש לרוורס אותו ולהבין מה הוא מצפה לקבל.

לאחר מכן אפשר לבצע את זה כך:

```c
#include "windows.h"
#include <stdio.h>

void ExecutePayload()
{
    printf("Sideloaded code running through function, out of DLL loader lock!\n");
}

typedef PVOID(WINAPI* fnHelloWorld)(/*Add Parameters If Needed From Documentation*/);

extern __declspec(dllexport) PVOID helloWorld() {
    ExecutePayload();

    HMODULE                 module_handle = NULL;
    fnHelloWorld        pHelloWorld = NULL;

    if (!(module_handle = LoadLibraryW(L"b.dll")))
        return NULL;

    if (!(pHelloWorld = (fnHelloWorld)GetProcAddress(module_handle, "helloWorld")))
        return NULL;

    return pHelloWorld(/*Add Parameters If Needed*/);
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        break;
    case DLL_THREAD_ATTACH:
        break;
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

[7] https://devblogs.microsoft.com/oldnewthing/20040128-00/?p=40853

ניתן לראות שאנחנו ממש טוענים (ראשית את ה-DLL המקורי בשורה 17, לאחר מכן את הפונקציה שלו בשורה 20), מריצים ומחזירים את הערך המתקבל מתוך הפונקציה helloWorld של ה-DLL המקורי (שורה 23 - הרצה + החזרת הערך), רק ששניה לפני הפונקציה האמיתית אנחנו מריצים את הקוד הפוגעני שלנו שרץ עתה מחוץ ל-Loader Lock (שורה 12).



[הקוד הזדוני רץ, ללא מגבלות של Loader Lock]

האופציה השנייה לברוח מ-Loader Lock היא להיות מאוד עקשנים, ובכל זאת להריץ קוד מתוך dllmain ולהצליח לברוח מה-lock. יש מחקר מדהים שנעשה בנושא באתר של ElliotonSecurity, ממליץ לקרוא אותו בחום[8]. הנקודה המרכזית שניתן לקחת מהמאמר היא שקיימות דרכים שונות להתחמק מ-Loader Lock גם בתוך DLL main בצורה יצירתית - מוזמנים לקרוא את המאמר וליישם אותן בעצמכם!

אנקדוטות אחרונות לשימוש בטכניקה:

1) לא מומלץ להשתמש ב-DLL שהתוכנה הזדונית שאתם כותבים משתמשת בו בעצמה (לדוגמה - לא להשתמש ב-ws2_32.dll אם אתם מתכוונים להשתמש ב-sockets, אתם כורתים את הענף שאתם יושבים עליו)

2) כל עוד אתם כותבים קוד שרץ ב-thread בזמן שהתוכנה רצה, ה-thread שלכם ייסגר כשהתוכנה המרכזית נסגרת (יכול להיות מאוד מבאס), ועל כן כדאי לחפש תוכנות שרצות רוב הזמן או להשתמש בהרצת ה-DLL הזדוני כדי להריץ קוד בקונטקסט שלא ייסגר עם סגירת התוכנית הראשית (לדוגמה - להריץ process חדש).

[8] https://elliotonsecurity.com/perfect-dll-hijacking/

דוגמה לשימוש ב-DLL Sideloading ע"י קבוצת תקיפה

קבוצות תקיפה מתקדמות (APT - Advanced Persistent Threats) פועלות ממש כמו חברת הייטק לתקיפה, למען מטרות רווח (קבוצות ransomware - תקיפות כופרה - כמו Lockbit[9]) או על מנת לשרת את המדינה שלהן (Muddy Waters הוא APT מוכר שמשוייך לאיראן, שביצע פעולות ריגול ותקיפת תשתיות ישראליות, בין היתר).[10]

רוב דוחות המודיעין האזרחיים אודות תקיפות בסייבר מבוססות על הפעילויות של APT-ים שונים, מה שנותן הצצה לדרך שבה תוקפים משתמשים בטכניקות תקיפה ומאפשרת לנו כצד כחול ללמוד לזהות ולהגן מפניהן.

אני לא אפרט על התקיפות מעבר לסקירה כללית - אפשר להקדיש לכל תקיפה כזאת והתת-טכניקות שלה מאמרים שלמים (ואכן כל תקיפה כזאת סוקרה כמאמר שלם, ואני ממליץ בחום לקרוא את המקור!) המטרה היא להביא טעימות לסוגי השימוש בטכניקה כפי שנראו במציאות.

יתרונות השימוש בטכניקה

רגע לפני שנסקור 2 דוגמאות לשימוש בטכניקה ע"י קבוצות תקיפה בעולם, בואו נסכם בקצרה את הסיבות שקבוצות תקיפה משתמשות בטכניקה זו:

1) **התחמקות ממנגנוני הגנה:** מספר רב של תוכנות AV ו-EDR-ים, יותר מיושנות סורקות קבצי EXE בזמן הפעלתם (בזמן ריצה בזיכרון - In-Memory Scan), אבל לא את ה-DLL-ים שהם טוענים, וזאת משום שמספר ה-DLL-ים הרב יכול לגרור סריקה ארוכה שתאט את הפעילות של המחשב ותפגע בחווית המשתמש. על כן - עצם השימוש ב-DLL ככלי זדוני מתחמק משלל כלי הגנה.

2) **הרצת קוד זדוני תחת תוכנה לגיטימית:** רוב התוכנות המוכרות בשוק חתומות ע"י היצרן באמצעות חתימה דיגיטלית שמהווה מעין הבטחה שלאחר הקימפול, המוצר לא השתנה ע"י תוקף. מאחר וה-EXE המקורי אינו משתנה בזמן התקיפה, התוקף בעצם מצליח להריץ קוד זדוני תחת תוכנה לגיטימית וחתומה - דבר שתורם רבות גם להתחמקות נוספת ממנגנוני הגנה וגם מקשה מאוד על זיהוי התקיפה מצד גופי חקירה.

3) **התחמקות מסריקות Sandbox**[11]**:** ישנן מספר פתרונות הגנה בשוק היום שמאפשרים סינון קבצים בדרכם לתוך הארגון, בצורה אוטומטית ומשמעותית יותר טובה מסריקה - וזאת באמצעות Sandbox. בעצם מריצים את הקובץ בסביבה מבודדת על VM (מכונה וירטואלית, Virtual Machine) בזמן שמנטרים כל פעולה של הקובץ. אם הוא עושה משהו חשוד (נוגע בערכי Registry חשובים, מתקשר עם

[9] https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-165a
[10] https://attack.mitre.org/groups/G0069/
[11] https://en.wikipedia.org/wiki/Sandbox_(computer_security)

שרת חיצוני מוזר וכד') אז אפשר בקלות יחסית לזהות שמדובר בפוגען, גם אם הוא הצליח להתחמק מסריקות סטטיות על הדיסק. עם זאת, יש הרבה פתרונות כאלה שסורקים את הקבצים אחד אחד, מה שנשמע הגיוני סה"כ - אבל במקרה של DLL Sideloading הקובץ DLL לא יופעל בעצמו ללא ה-EXE שאמור להשתמש בו, וזו דרך נוחה לעקוף סריקות דינאמיות פשוטות בלי שום סיבוך נוסף.

4) שרידות לא קונבנציונלית. לרוב גופי הגנה כדוגמת SOC (מרכז הניטור - Security Operations Center) מנטרים פעילות חריגה בעמדות קצה שמשמשת פוגענים לשרידות (שרידות משמע Persistence, היכולת לחזור לרוץ על המחשב גם לאחר כיבוי או ריסוט). פעולות אלה כוללות הוספה לתיקיית startup, הוספה ל-registry של run או run once, הוספת schedule task ועוד. SOC יותר מתקדם אף ינטר את הקבצים עצמם - לראות שלא מחליפים את קבצים שנדרשים לעלות עם המ.ה. בקבצים זדוניים (לדוגמה, אם אני מחליט שעל כל מחשב בחברה חייב לרוץ legitExe.exe בכל חיבור של משתמש, אני אוודא שלא משנים לי את legitExe). עם זאת - ניטור על קובץ שהתווסף לתוך התיקייה שבה נמצא קובץ EXE אחר היא פעולה יחסית לגיטימית, ועל כן פחות מנוטרת (עלולה גם לגרור ליותר false positives), מה שמאפשר להשיג שרידות פחות קונבנציונלית שתעבור מתחת לסף הרעש של ה-SOC.

5) הסלמת הרשאות. במקרים מעט יותר נדירים אף אפשר להשתמש בטכניקה כדי להסלים הרשאות ממשתמש חלש למשתמש חזק, במידה ויש קובץ EXE שעתיד לרוץ ע"י משתמש חזק (לדוגמה - באמצעות scheduled task שרץ בהרשאות גבוהות) כאשר ניתן לשים בתיקייה שלו או בנתיב אחר קובץ DLL שייטען על ידיו, במיקום שדורש הרשאות חלשות.

6) פשטות! זה אולי מרגיש קצת טריוויאלי, אבל זה דגש חשוב מאוד - תקיפה בסייבר זה עסק לכל דבר ועניין, כשהמטרה היא להגיע למקסימום אפקטיביות במינימום השקעה. הטכניקה היא מאוד פשוטה ומקנה המון יתרונות לתוקף, ולכן היא נהפכה להיות מאוד פופולרית בשנים האחרונות.

שימוש בטכניקה להשגת ציר נגישות ע"י Wineloader[12]

הפעם הראשונה שקראתי מאמר שעורר בי את הסקרנות לגבי DLL-Sideloading הייתה על פוגען בשם "Wineloader" שזוהה על ידי חברת Zscaler כתקיפה של APT רוסי על דיפלומטים אירופאיים. התקיפה השתמשה בטכניקת DLL-Sideloading על מנת להריץ את הקוד הזדוני לראשונה על הקורבנות שלהם (הדבקה ראשונית):

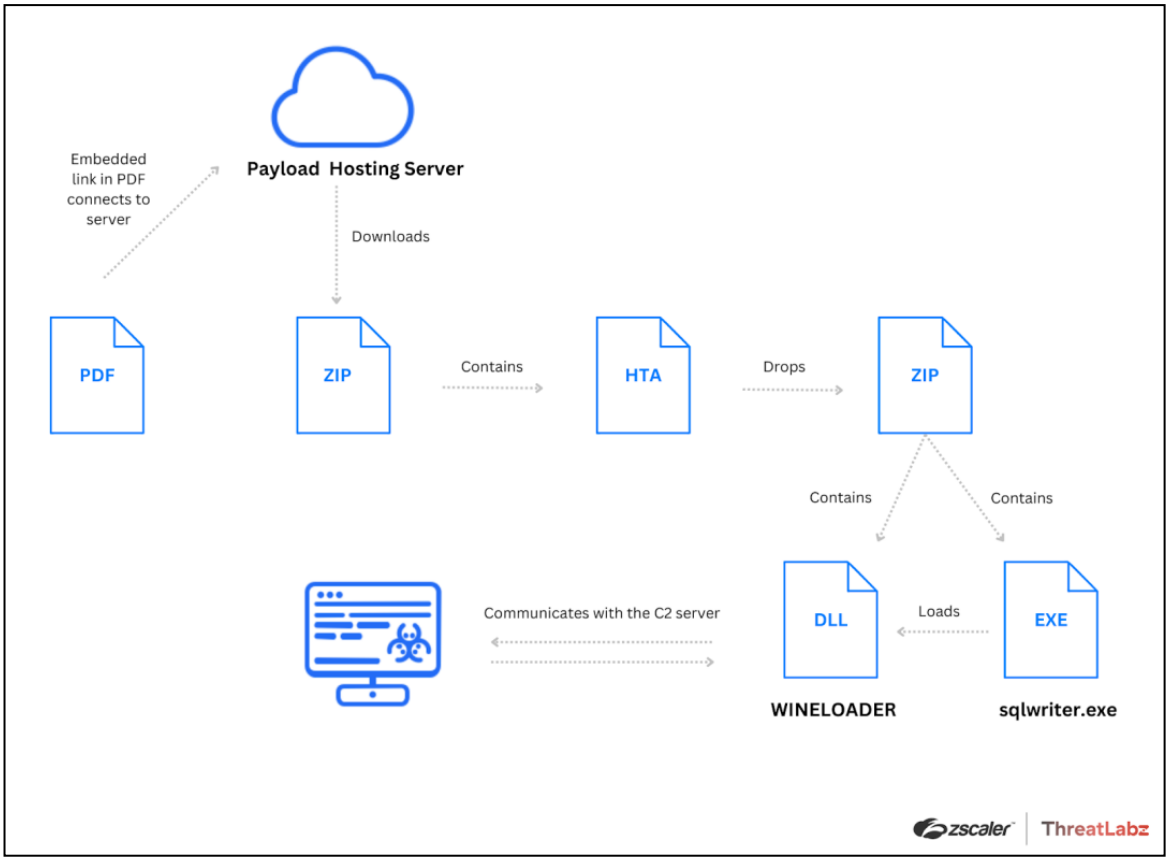
1) התוקפים מצאו קובץ EXE פגיע בשם sqlwriter.exe, כתבו DLL זדוני וביצעו ZIP לשניהם כך שה-DLL יהיה צמוד ל-EXE לאחר חילוץ.

[12] https://www.zscaler.com/blogs/security-research/european-diplomats-targeted-apt29-cozy-bear-wineloader

2) התוקפים הגיעו לקורבנות שלהם באמצעות פישינג ממוקד לאותם האנשים (Spear Phishing).

3) הקורבנות פתחו את ה-EXE, מה שהפעיל DLL זדוני שהזריק קוד של פוגען לתוכנה אחרת במחשב על מנת שלא להיסגר עם סגירת התוכנה המקורית (אם היו סוגרים את sqlwriter.exe ללא הזרקת קוד זדוני למקום אחר, הקוד היה מפסיק לרוץ).

4) ה-DLL היווה C2 (יכולת שליטה ובקרה, Command and Control) לתוקפים, ובכך הם השיגו אחיזה לטווח ארוך על המחשב שאפשרה להם להריץ פקודות ולהמשיך הלאה במבצע.

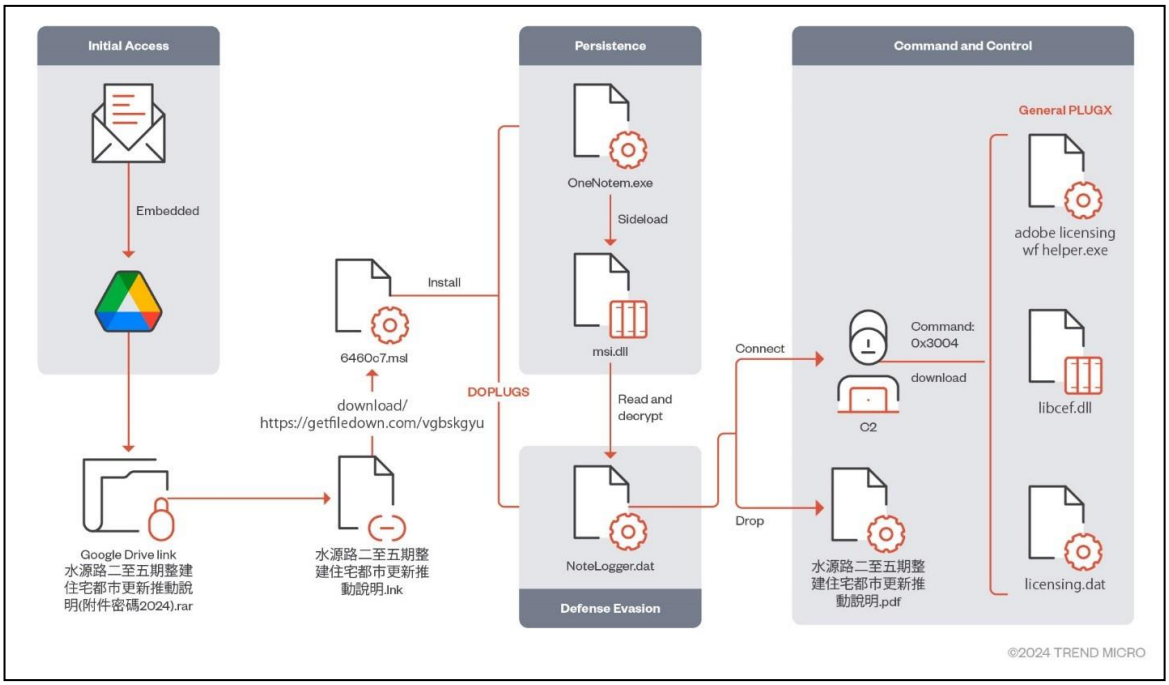


[תרשים זרימה לציר התקיפה ב-Wineloader. מקור מתוך המאמר של zscaler]

זאת גם דרך השימוש הנפוצה ביותר בטכניקה ע"י תוקפים בעת הנוכחית מקריאה שלי. ברוב המקרים, גם אנשים חשדנים במיוחד עם יכולות טכניות לא יבדקו קבצים מעבר ל-EXE, ועל כן הציר הזה מאפשר אחיזה ראשונית מאוד שקטה שלא תעורר חשד.

שימוש בטכניקה לצרכי שרידות ע"י Earth Pretra[13]

קמפיין שסוקר ע"י חוקרים של TrendMicro, שנבנה על חקירות קודמות של CheckPoint ושל Sophos, מציג APT סיני בשם Earth Preta שביצע קמפיין תקיפות בסייבר כנגד מונגוליה וטיוואן ובו הוא משתמש בפוגען בשם PlugX ונותן לו שרידות בעזרת DLL-Sideloading.



[תרשים זרימה לציר התקיפה בקמפיין, בה ניתן לראות שימוש ב-DLL-Sideloading לטובת שרידות (persistence). תמונה מתוך המאמר של TrendMicro]

ניתן לראות שהם התעלקו על exe לגיטימי בשם OneNotem.exe, שטוען msi.dll זדוני, והוא מתחיל את שרשרת התקיפה בכל פעם ש-OneNotem.exe מופעל מחדש. המטרה של השימוש בטכניקה כאן מצד התוקפים היא פחות להערים על הנתקף, ויותר להיתמם מול גופי הגנה שעלולים לעלות על התקיפה ולנסות לחקור אותה.

אני ממליץ בחום לקרוא את המאמרים המלאים על התקיפות הללו, אך מאחר והם מכילים מספר נרחב של חלקים שונים בתקיפה שלא קשורים למאמר הנוכחי אני לא אפרט עליהן לעומק ואסתפק בלציין שמספר רב של תוקפים משתמשים בטכניקה הזו.

[13] https://www.trendmicro.com/en_us/research/24/b/earth-preta-campaign-targets-asia-doplugs.html

המלצות למניעה, זיהוי וחקירה

מניעה

ישנן מספר דרכים "להקשיח" קוד של תוכנה או את הקונפיגורציה על עמדת הקצה, על מנת להקשות על תוקף להשתמש ב-DLL Sideloading, ובמרכזן:

1) **הוספת DLL-ים מ-system32 לתוך ה-registry ב-KnownDLLs**, מה שיגרום להם להיטען לפני DLL זדוני שנמצא בתיקייה יחד עם תוכנה.
2) **במידה וכותבים קוד - להוסיף תמיד את הנתיב המלא לקובץ ה-DLL שמשתמשים בו** (לדוגמה: C:\Windows\System32\a.dll ולא רק a.dll). בכך אנחנו מדלגים על החיפוש בתיקייה הנוכחית.
3) **וידוא DLL safe search mode** - אופציה ב-Windows לשנות את סדר החיפוש אחר DLL-ים כך שהתיקייה הנוכחית מגיעה לאחר חיפוש ב-system32.
4) **לוודא שסורקים חתימות של DLL-ים שמגיעים יחד עם EXE**, גם אם ה-EXE חתום על ידי גורם מוסמך (מייקרוסופט או כל חברה אחרת) - ה-DLL שאיתו עלול להיות זדוני.

זיהוי

1) **זיהוי של DLL-ים מוכרים (לפי שם) או exe-ים מוכרים (לפי hash) שמופעלים ממקומות לא סטנדרטיים**. תוקף יהיה מחוייב להשתמש בשם של DLL בצורה מדוייקת כדי לגרום לו להיטען ראשון במקום DLL אחר, ולכן בדיקה על DLL-ים מתוך system32 / system שנטענים פתאום בתיקייה אחרת יכולה להיות אינדיקציה מצויינת לשימוש בטכניקה הזו. זה אמנם דורש הרבה חוקים פרטניים, אך זו הדרך הנכונה ביותר לגילוי כנגד DLL-ים מוכרים. גם זיהוי של EXE-ים מוכרים מתוך תיקיות מ.ה. שמופעלים ממקומות לא סטנדרטיים יכולה להיות אינדיקציה לטכניקה זו, במתאר שתוקף מעתיק את ה-EXE לתוך תיקייה אחרת ומצמיד לידו DLL זדוני (כמו הדוגמה על calc.exe שהוצגה מוקדם יותר במאמר זה).
   - תת-סעיף של זיהוי מסוג זה, הוא התווספות של DLL-ים לתוך תיקיות System32 / System ושאר הנתיבים בעלי עדיפות ו/או מרכזיים במחשב שלכם, מה שעלול להצביע על תוקף עם הרשאות גבוהות שהטמין DLL לטובת שרידות על עמדת הקצה (לדוגמה - הוספת DLL בשם version.dll לתוך system32, ושינוי ה-version.dll האמיתי לשם אחר לטובת proxy).
2) **שימוש בכלי הגנה אשר סורקים DLL-ים בזמן טעינה.** בעבר סרקו בעיקר תוכנות (EXE) בזמן טעינה ל-RAM (סריקה דינאמית, בניגוד לסריקות סטאטיות על הדיסק), ולא סרקו DLL-ים מאחר והיה יותר נדיר למצוא DLL-ים זדוניים וגם העלות של סריקת DLL-ים עלתה משמעותית. כיום, עם כוח עיבוד משמעותית יותר חזק ושימוש תדיר מאוד ב-DLL-ים זדוניים, חשוב לוודא שה-AV או EDR שאנחנו משתמשים בו סורק גם DLL-ים בזמן טעינה לזיכרון.

3) **בעת שימוש ב-Sandbox - סריקת כלל הקבצים יחד עם ה-DLL-ים ולא כל קובץ בנפרד,** אחרת הקוד הפוגעני עלול שלא לרוץ (מאחר ואין EXE שקורא לפונקציות שלו) ומנוע הסריקה עלול שלא לזהות אותו, למרות שהיה מזהה אותו אם הוא היה רץ.
4) **שימוש בחוקי YARA או מקביליהם לאיתור קטעי קוד שעלולים להצביע על הימצאות ניסיון למימוש הטכניקה**, כגון מספר רב של redirections באמצעות pragma comment, או זיהויים נוספים המפורטים במאמר "Perfect DLL Hijacking" (כמו זיהוי ה-AtExit).

מאחר והטכניקה היא יחסית שקטה, בסבירות בינונית-גבוהה שהיא תעבור מתחת לרדאר. נשמע קצת מבאס, אבל תמיד חשוב לחשוב קדימה כצד כחול - התוקף בסוף יהיה חייב לבצע פעולות כלשהן שיממשו את המבצע שהוא מנסה ליישם, כמו תקשורת לאחור (C2), ביצוע פיקודים נוספים, הצפנת קבצים לטובת כופרה, הזרקת קוד לתוכנות נוספות וכד'. המטרה של הזיהויים הספציפיים על DLL Sideloading הם כחלק ממעטפת רחבה יותר של זיהויי תקיפה, שבסוף מקשים מאוד על ניהול מבצע מצד התוקף.

חקירה

הדגש היחיד שאני יכול לחשוב עליו בעת חקירת אירוע הוא עצם היכרות עם הקונספט של DLL-Sideloading, ובכך לא להתמקד רק ב-EXE שעשה פעולות זדוניות אלא גם ב-DLL-ים שנמצאים בנתיבי הטעינה שלו, שיכולים להריץ בעצמם את הקוד הזדוני שלהם מתוך תוכנה אקראית ולגיטימית.

מומלץ גם לחקור את ה-DLL הזדוני באמצעות הרצה שלו תחת ה-EXE שנמצא איתו בתיקייה, מאחר ותוקפים עלולים לבדוק את הצמידות הזו לביצוע פעולות מסוימות (אם ה-DLL לא רץ תחת calc.exe אל תריץ את הקוד הזדוני, טכניקות התחמקות נוספות...)

סיכום

במאמר זה סיקרנו מכל הצדדים את טכניקת התקיפה בשם DLL-Sideloading. למדנו לעומק את הדרך ליישום הטכניקה מ-0 ל-100 כולל שימוש בכלי עזר שבניתי בשם Nihilego, עברנו על דרך פשוטה לאתר את החולשה בקבצי EXE באמצעות תוכנת procmon, סיכמנו את כל היתרונות שיש בשימוש בטכניקה מצד התוקף - מהתחמקות מזיהוי, דרך שרידות על עמדת קצה ועד להסלמת הרשאות, וראינו דוגמאות לתקיפות בטכניקה ע"י קבוצת תקיפה רוסית וקבוצת תקיפה סינית. לבסוף התבוננו על הטכניקה מהצד הכחול - איך ניתן למנוע, לזהות ולחקור תקיפות המשתמשות ב-DLL-Sideloading בצורה יותר אפקטיבית.

על המחבר

שמי ניר גילס, אשמח לענות על שאלות, לשמוע טענות ותהיות ולדון דיונים:

- adom.nir19@gmail.com
- linkedin.com/in/nir-g-160184230

מוזמנים למצוא את הכלי בגיטהאב:

https://github.com/Team-Phoenix07/Nihilego

המון תודות ל**ארד דוננפלד** שעזר לעבור על המאמר ולהפוך אותו ליותר קריא ומדויק, ול**אפיק קסטיאל** על התמיכה לאורך הדרך :)

ביבליוגרפיה

קריאה על סדר טעינת DLL-ים מתוך האתר של Microsoft:

https://learn.microsoft.com/en-us/windows/win32/dlls/about-dynamic-link-libraries

https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order

דוגמה ל-DLL Loader Lock:

https://devblogs.microsoft.com/oldnewthing/20040128-00/?p=40853

הרצת קוד מתוך DLL-Main תוך עקיפת DLL Loader Lock:

https://elliotonsecurity.com/perfect-dll-hijacking/

להתעדכן על חדשות סייבר, עם קישורים לכתבות המקוריות:

https://thehackernews.com/

מאגר תוכנות מוכרות עם פגיעות DLL-Sideloading:

https://hijacklibs.net

מספר רב של דוגמאות לתקיפות המשתמשות ב-DLL-Sideloading:

https://www.securonix.com/blog/detecting-dll-sideloading-techniques-in-malware-attack-chains/

הסבר על קבוצות התקיפה שהובאו כדוגמה במאמר:

https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-165a

https://attack.mitre.org/groups/G0069

תקיפת Wineloader:

https://www.zscaler.com/blogs/security-research/european-diplomats-targeted-apt29-cozy-bear-wineloader

מאמרים נוספים להבנת DLL Sideloading ודרכי זיהוי והתמגנות:

https://redcanary.com/threat-detection-report/techniques/dll-search-order-hijacking

https://cihansol.com/blog/index.php/2021/09/14/windows-dll-proxying-hijacking

דרכים להתמגן מ-DLL Sideloading (מניעה):

https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-security

היכרות בסיסית עם קבצי DLL:

https://en.wikipedia.org/wiki/Portable_Executable

https://learn.microsoft.com/en-us/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library

על Sandbox (ויקיפדיה):

https://en.wikipedia.org/wiki/Sandbox_(computer_security)

קימפול DLL-ים עם קובץ def להחצנת פונקציות:

https://learn.microsoft.com/en-us/cpp/build/exporting-from-a-dll-using-def-files?view=msvc-170