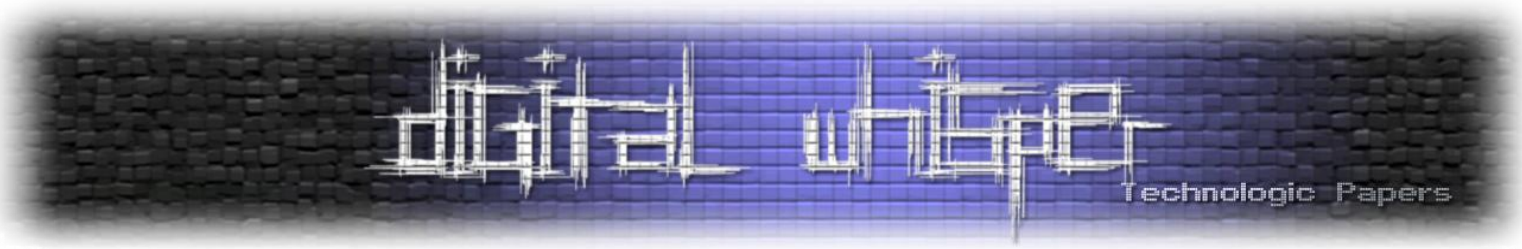


Proxy DLL

מאת אילן וינוגרד

הקדמה

עולם התוכנה רווי ביישומים שאינם חשופים לקהל המפתחים: תוכנות בקוד סגור, משחקים מסחריים וממשקים גרפיים קנייניים, כולם פועלים מעל שכבות התקשורת מול מערכת ההפעלה.

אבל מה אם היינו יכולים "להשתלב" בתהליך הזה מבלי לגעת בקוד המקורי?

מה אם היינו יכולים להוסיף תכונות, לבצע שינויים, לשפר ביצועים או אפילו לחלץ מידע מבלי שתהיה לנו גישה לקוד התוכנה עצמה?

כאן נכנסת לתמונה טכניקה אלגנטית, מפתיעה בפשטותה, אך עוצמתית במיוחד: Proxy DLL. מדובר בשיטה בה אנו יוצרים ספרייה דינמית (DLL = Dynamic-Link Library) שמשחקת את תפקיד "המקורית", אך בפועל היא עוטפת את הקריאות הפנימיות, מוסיפה לוגיקה משלנו, ולעיתים גם מחליפה את ההתנהגות המקורית.

באמצעות Proxy DLL ניתן:

- להתממשק לכל תוכנה קיימת מבלי לשנות את קוד המקור
- להרחיב, לשנות או לעקוב אחר פעולות פנימיות של יישומים
- להוסיף שכבות לוגיקה, אבטחה, ניטור או הדמיה
- לבצע אינטגרציה בין תוכנות שלא תוכננו לעבוד יחד
- ולמעשה לממש כמעט כל פעולה שנמצאת "בדרך" בין תוכנה לבין מערכת ההפעלה

במאמר זה נצלול לעומק הטכניקה נבין איך היא פועלת, מהם יתרונותיה ומגבלותיה, ונראה כיצד ניתן לממש אותה בפועל.

כדוגמה, נציג פרויקט בשם **DirectXSwapper**, שמיישם Proxy DLL כדי לחלץ אובייקטים גרפיים ממשחקים. אך לפני שנצלול אליו, נבין את המתודולוגיה שעומדת מאחורי הכל.

לפני שנתחיל, שלוש הערות חשובות:

1. כל הנאמר במאמר זה נועד למטרות מחקר, תיעוד ולמידה בלבד. אין לראות בו קריאה לשימוש לרעה בטכניקות המוצגות.
2. כל הכלים והדוגמאות כאן מבוססים על מערכת ההפעלה Windows, אך העקרונות תקפים גם לפלטפורמות אחרות.
3. ננסה לשלב בין הבנה תיאורטית עמוקה לבין דוגמאות קונקרטיות כך שגם מי שאין לו רקע בפיתוח מערכות או גרפיקה יוכל לעקוב.

מה זה Proxy DLL ואיך זה עובד?

במערכות מבוססות Windows (אך גם באחרות), תוכנות רבות תלויות בספריות דינמיות, Dynamic-Link Libraries או בקיצור: DLLs.

הספריות האלו מכילות פונקציות קיימות שהתוכנה יכולה להשתמש בהן, מבלי לכתוב אותן בעצמה. לדוגמה:

- פעולות קלט/פלט כמו פתיחת קובץ (CreateFile)
- ציור תלת-ממדי על המסך (Direct3DCreate9)
- ניהול חלונות, גישה לרשת, זיכרון ועוד

כאשר תוכנה נטענת, מערכת ההפעלה טוענת את הספריות שהיא צריכה, ומקשרת אליהן.

הנה בדיוק המקום שבו Proxy DLL נכנס לפעולה.

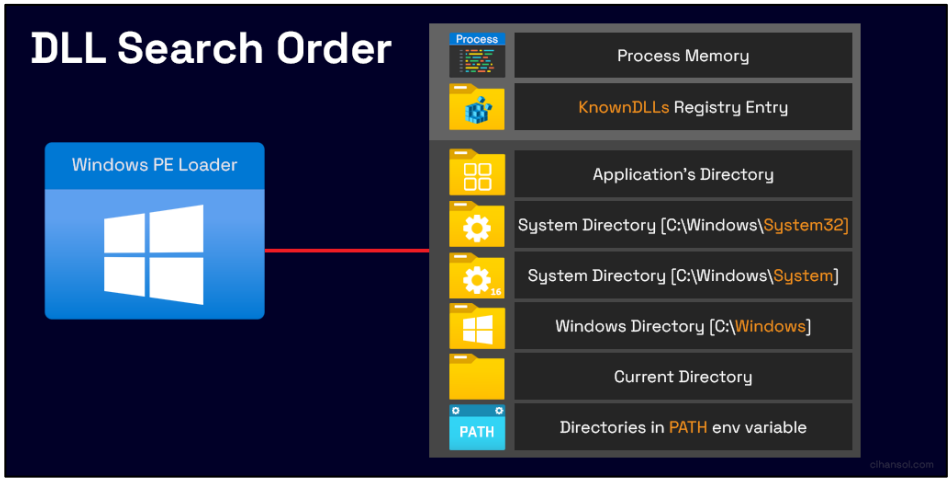
כיצד Windows מחפשת DLL?

כאשר תוכנה מריצה קריאה לספרייה חיצונית (DLL), מערכת ההפעלה לא פשוט "לוקחת אותה מהמדף", היא מבצעת חיפוש מסודר, לפי סדר מוגדר מראש.

זהו "סדר חיפוש ה-DLL (DLL Search Order)" של Windows. סדר זה קובע מאיפה תטען הספרייה, והוא המפתח ליצירת Proxy DLL.

ברירת המחדל של Windows (בגרסאות מודרניות) היא:

1. התיקייה שבה נמצא קובץ ה-exe. שמריץ את התוכנה
2. התיקייה הנוכחית של התהליך (GetCurrentDirectory)
3. System32 ספריית הDLLs המרכזית של Windows
4. System (בסביבת 16 ביט בלבד, לרוב לא רלוונטי היום)
5. תיקיית Windows הראשית (C:\Windows)
6. תיקיות שנמצאות ב- PATH של הסביבה



[Taken from dev blog]

המשמעות היא פשוטה אך חזקה:

אם נניח קובץ בשם d3d9.dll בתיקייה של המשחק עצמו (קובץ שאינו מקורי אך נושא את אותו שם), הוא ייטען במקום הקובץ האמיתי שנמצא ב-System32, בלי שהמשחק יבחין בכך כלל.

טכניקות טעינה מתקדמות Registry ו-T1546.010

מעבר לסדר החיפוש הרגיל של Windows, קיימות גם טכניקות מתקדמות לטעינת DLL אל תוך תהליכים גם מבלי שהקובץ יימצא פיזית בספריית ההרצה. אחת השיטות המוכרות לכך היא שימוש בערכי Registry. באמצעות ערך בשם **AppInit_DLLs** תחת המפתח:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows
```

ניתן לקבוע רשימה של קבצי DLL שייטענו באופן אוטומטי אל כל תהליך GUI בכל פעם שהוא נטען. הדבר קורה מבלי צורך בהרשאות מיוחדות או שינוי בקבצי ההפעלה עצמם.

שיטה זו מתועדת היטב גם במסמך MITRE ATT&CK T1546.010, ומוכרת ככלי בידי תוקפים המעוניינים להפעיל קוד מרגע עליית המערכת או בתהליכים רגישים.

השימוש ב-**AppInit_DLLs** מאפשר למעשה "חיבור שקט" לכל תהליך לטובת ניטור, שליטה, או התקפה וזוהי דרך עוקפת לסדר החיפוש הסטנדרטי.

אף ש Microsoft הגבילה שימוש בערך זה בגרסאות מודרניות של Windows עדיין ניתן למצוא מערכות בהן ערוץ זה פתוח, בעיקר בארגונים או תוכנות ישנות.

יצירת Proxy DLL בפועל

אחרי שהבנו מה זה Proxy DLL ולמה זה עובד, ניגש לפועל: כיצד כותבים כזה קובץ בעצמנו?

לפני שנכתוב את ה-DLL, עלינו לדעת איזה DLL נטען, ואילו פונקציות ממנו התוכנה דורשת. לשם כך נשתמש בכלים ייעודיים:

- Procmon (Process Monitor) הוא כלי מבית Sysinternals (Microsoft) מאפשר לעקוב בזמן אמת אחרי כל קריאות מערכת.
- CFF Explorer הוא מאפשר לפתוח קבצי dll. או exe. ולראות באילו ספריות דינמיות הם משתמשים, ומהן הפונקציות שמיוצאות.

דוגמה לקובץ MyApp.exe שטוען DLL חיצוני (mydll.dll) ומריץ מתוכו פונקציה בשם "Hello()"

```c
#include <windows.h>

typedef void (*MyFunc)(); // Function pointer type: void Hello()

int main() {
    HMODULE dll = LoadLibraryA("mydll.dll"); // Load the DLL
    if (!dll) return 1;

    MyFunc f = (MyFunc)GetProcAddress(dll, "Hello"); // Get pointer to "Hello" function

    if (f) f(); // Call the function if found

    return 0;
}
```

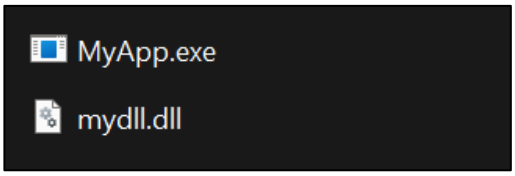
קוד של mydll.dll

```cpp
#include <windows.h>

extern "C" __declspec(dllexport) void Hello() {
    MessageBoxA(0, "Hello from original DLL", "Real DLL", MB_OK); // Exported function: shows a message box
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD reason, LPVOID reserved) {
    return TRUE; // No initialization needed
}
```

בדוגמה זו יצרנו תוכנה פשוטה (MyApp.exe) שטוענת DLL חיצוני בשם mydll.dll ומריצה ממנו פונקציה בשם Hello().



זוהי אפליקציית הבסיס שנעבוד איתה לאורך שאר המאמר באמצעותה נדגים כיצד ניתן לעטוף, להחליף או להרחיב את ההתנהגות של ספריות דינמיות (DLL) בעזרת טכניקת Proxy DLL.

בשלב הבא ניצור גרסה "מתחזה" של הספרייה (mydll.dll), שתתפקד כ-Proxy אשר תעשה intercept-לפונקציה Hello(), תוסיף לוגיקה משל עצמה ורק לאחר מכן תעביר את השליטה אל הספרייה המקורית.

איך נזהה את ה DLL והפונקציות הנדרשות?

כדי לכתוב Proxy DLL בצורה מדויקת, אנחנו צריכים:

- לדעת מה שם ה-DLL שהאפליקציה טוענת
- לדעת אילו פונקציות נדרש Export

לפני שנוכל ליצור את ה-DLL נדרשת הבנה בסיסית של מה בדיוק נטען על ידי התוכנית.

לצורך כך, השתמשנו בשני כלים עיקריים:

באמצעות Procmon זיהינו שהתוכנה MyApp.exe מנסה לטעון את הקובץ mydll.dll.

באמצעות CFF Explorer בדקנו אילו פונקציות מיוצאות מהספרייה המקורית mydll.dll במקרה זה, הפונקציה Hello()נשים לב ל-Ordinal (מספר סידורי של הפונקציה).

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	00001C78	00001C80	00001C7C	00001C8C
(nFunctions)	Dword	Word	Dword	szAnsi
00000001	00001000	0000	0000288C	Hello

כתיבת Proxy DLL

בשלב זה נכתוב את mydll.dll החדש שיראה מבחוץ כמו הספרייה המקורית, אך בפועל "ישתלט" על הקריאות ויבצע קוד נוסף.

עקרון הפעולה:

1. כש-MyApp.exe טוען את mydll.dll, הוא מקבל את הגרסה שלנו (ה-Proxy).
2. ה-Proxy מטעין מאחורי הקלעים את real_mydll.dll (הגרסה המקורית של ה-mydll.dll, ששינינו לה את השם כדי שנוכל להחליף אותה בפרוקסי שלנו תוך שמירה על ההתנהגות המקורית)
3. כשנקראת הפונקציה Hello(), אנחנו נבצע קוד משלנו ואז נמשיך לקריאה האמיתית.

קובץ proxy_dll.cpp:

```cpp
#include <windows.h>

typedef void (*HelloFunc)(); // Define the exact function signature based on documentation

HelloFunc realHello = nullptr; // Pointer to the real Hello function

extern "C" __declspec(dllexport) void Hello() { // Exported proxy function
    MessageBoxA(nullptr, "Intercepted Hello()", "Proxy DLL", MB_OK);

    if (realHello) {
        realHello(); // Call the original function to keep the pipeline
    }
}

BOOL APIENTRY DllMain(HMODULE hModule, DWORD reason, LPVOID reserved) {
    if (reason == DLL_PROCESS_ATTACH) {

        HMODULE realDll = LoadLibraryA("real_mydll.dll"); // Load the real dll "we renamed mydll.dll to real_mydll.dll
        if (realDll) {
            realHello = (HelloFunc)GetProcAddress(realDll, "Hello"); // Get the address of the original Hello() function
        }
    }
    return TRUE;
}
```

בדוגמה זו אנו כן מכירים את הסיגנטורה של הפונקציה Hello - לפי תיעוד או Reverse Engineering. לכן, במקום להשתמש ב-FARPROC, אנו מגדירים טיפוס מדויק typedef void (*HelloFunc)() ומבצעים את הקריאה בצורה בטוחה וישירה.

עם זאת ,אין לנו גישה לקוד של הפונקציה עצמה ולכן איננו יודעים מה היא עושה בפנים. ייתכן שהיא שולחת נתונים, משנה מצב פנימי או ניגשת לקובץ, אך אנו לא תלויים בזה. כל עוד נשמרת החתימה, הקריאה תמשיך לעבוד בצורה שקופה.

קובץ proxy.def:

```
LIBRARY mydll
EXPORTS
    Hello @1
```

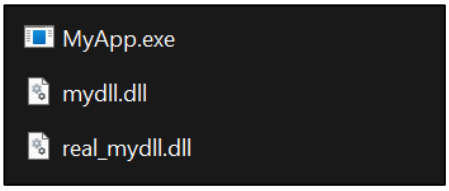
קובץ def. מגדיר ל-linker של Visual Studio או GCC מה לייצא מתוך הספרייה שלנו.

במקרה זה, הקובץ def. מגדיר את שם הספרייה (mydll.dll) באמצעות LIBRARY, מייצא את הפונקציה Hello עם EXPORTS ומציין את ערך ה- Ordinal שלה כ-1(Hello @1) , בהתאם לדרישות שזוהו בניתוח עם CFF Explorer.

אם הפונקציה לא תיוצא בשם הנכון או עם ה-Ordinal המתאים MyApp.exe לא תמצא אותה, והקריאה תיכשל.

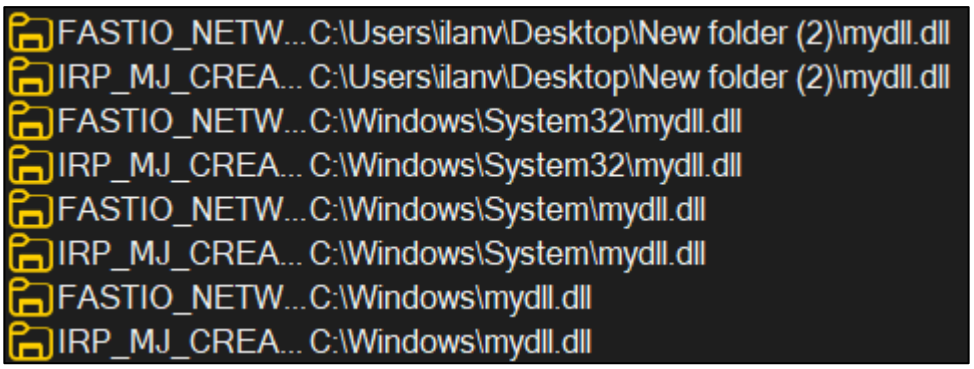
כעת, לאחר שיצרנו את קובץ ה-mydll.dll ובדקנו שהוא מייצא את הפונקציה בצורה תקינה, כל מה שנותר הוא להציב אותו ליד הקובץ MyApp.exe - במקום הספרייה המקורית.

כדי להפעיל את ה-Proxy DLL נשנה את שם הספרייה המקורית מ-mydll.dll ל-real_mydll.dll, נציב את קובץ ה-Proxy תחת השם mydll.dll (כפי שהתוכנה מצפה לו), ונוודא שכל הקבצים נמצאים באותה תיקייה לצד MyApp.exe.



נריץ את MyApp.exe:

נשים לב לחיפוש של הDLL שלנו ב Procman:



למה זה כל כך משמעותי?

באמצעות Proxy DLL הצלחנו להריץ קוד משלנו ממש בתוך תהליך של תוכנה קיימת, מבלי לשנות את הקובץ המקורי שלה (MyApp.exe) או אפילו לדעת מה עושה הפונקציה האמיתית ()Hello.

זוהי לא התחלה חדשה של תוכנית, אלא השתלה של לוגיקה בתוך קריאה קיימת. הפונקציה המקורית ממשיכה לעבוד אך בדרך היא "עוברת דרכנו", ואנחנו יכולים:

- ליירט מידע
- לשנות ערכים
- להפעיל קוד מותאם אישית
- ואפילו לבטל או לדמות פונקציות לפי רצוננו

כל זאת ,מבלי לשנות אף שורת קוד בתוכנה המקורית!

זהו כלי חזק מאוד והוא מהווה בסיס להרבה טכניקות מתקדמות: החל ממחקר reverse engineering, דרך תיקוני באגים בספריות צד שלישי, ועד לשימושים שנויים במחלוקת כמו פיתוח צ'יטים או הרצת קוד עוין.

דוגמאות לשימושים מתקדמים ב-Proxy DLL

אחרי שהבנו את העקרון, אפשר לראות כיצד הוא מיושם בעולם האמיתי. הנה מספר תרחישים אמיתיים שבהם נעשה שימוש בטכניקה זו כל אחד מהם מראה כוח אחר של היכולת "ליירט" קריאות:

מעקב אחר שימוש במשאבים (לוגים, ביצועים, זליגות) באמצעות יירוט של קריאות ל-kernel32.dll או ntdll.dll, אפשר לעקוב אחר: הקצאות זיכרון (HeapAlloc, VirtualAlloc), פתיחת קבצים (CreateFile), זמני תגובה (Sleep, GetTickCount).

בתחום ה-Cyber Offensive, יש שימוש נפוץ בפרוקסי DLL כדי להשתיל רוגלה סמויה בתוך אפליקציה לגיטימית. הרעיון: "סוס טרויאני בתוך DLL"

תוקף יוצר גרסה מזויפת של DLL לגיטימי כגון (version.dll) עם אותן פונקציות מיוצאות, ודואג שהתוכנה תטען אותו תחילה (באמצעות DLL Sideloading או Search Order Hijacking), כך שה-Proxy DLL יפעיל את הפונקציות האמיתיות כדי לשמור על תפקוד תקין אך במקביל יריץ קוד זדוני, למשל לשליחת קבצים לשרת מרוחק, האזנה למיקרופון, חסימת עדכוני אבטחה או חיפוש אחר מסמכים רגישים.

דוגמה אמיתית לכך נראתה במסמכים של קבוצות תקיפה כמו APT28 או Lazarus, שבהם נעשה שימוש ב-Proxy DLL בשם msimg32.dll, עם לוגיקה זדונית מוסתרת.

DirectXSwapper - יירוט גרפיקה מתוך משחקים

דוגמה מצוינת לכך היא הפרויקט DirectXSwapper שכתבתי. מדובר בספריית DLL שמשתמשת בדיוק בטכניקה שתיארנו כאן רק שבמקום ליירט פונקציה פשוטה כמו Hello()، היא משתלטת על קריאות גרפיות קריטיות כמו:

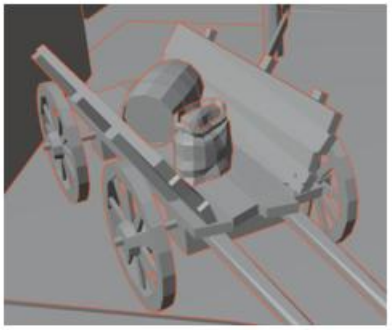
- DrawIndexedPrimitive ב-Direct3D 9
- ExecuteCommandLists ו-Present ב-Direct3D12

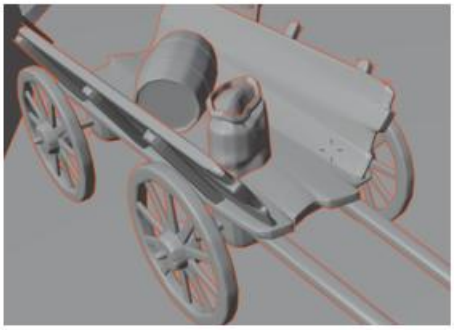
הספרייה נטענת במקום הספרייה הגרפית האמיתית, ומצליחה לחלץ בזמן אמת מודלים תלת-ממדיים, טקסטורות, ואפילו לשנות את מה שמוצג על המסך כל זאת מבלי לגעת במשחק עצמו.

פרויקט נוסף שבו נעשה שימוש ב-Proxy DLL הוא כלי שכתבתי לשיפור גרפיקה בזמן אמת במשחקים ישנים. הרעיון פשוט אך עוצמתי: אנו לא משנים את קבצי המשחק, אלא "מתלבשים" על הקריאות לגרפיקה, בדיוק כפי שעשינו בדוגמה הקודמת, ומחליפים את המידע בדרך.

בדוגמה הבאה (ראו תמונות), רואים עגלה פשוטה מתוך משחק ישן:

- בצד שמאל המודל המקורי, כפי שהמשחק טוען אותו.
- בצד ימין אותו אובייקט לאחר שחולץ, שופר (subdivision, smoothing) והוחזר בזמן אמת לשימוש על ידי המשחק.





בזכות הגישה הזו אפשר: לשפר גרפיקה של משחקים שלא מקבלים עדכונים, להוסיף תמיכה באיכות גבוהה יותר ללא גישה לקוד המקור, לבצע מודינג דינאמי בזמן ריצה.

סיכונים ואמצעי הגנה בשימוש ב-Proxy DLL

לצד היתרונות והיכולות המרשימות של טכניקת ה-Proxy DLL, חשוב להבין היטב את הסיכונים הכרוכים בשימוש בה. שימוש בטכניקה זו ללא נקיטת אמצעי זהירות נאותים עלול לגרום לפגיעה משמעותית ביציבות התוכנה, לביצועים ירודים, ואף להוביל להשלכות משפטיות חמורות.

הסיכונים העיקריים כוללים פגיעה ביציבות, שבה הזרקת DLL זר לתהליך קיים עשויה להוביל לקריסות תוכנה, זליגות זיכרון, או תקלות בלתי צפויות. כמו כן, שכבת ביניים נוספת המיירטת קריאות יכולה להאט את ביצועי התוכנה, במיוחד ביישומים רגישים כמו משחקים או תוכנות גרפיות.

מבחינת אבטחה, שימוש לא זהיר בטכניקה עלול לפתוח פתח לניצול על ידי גורמים זדוניים, כגון הפצת נוזקות, סוסים טרואנים, או איסוף מידע אישי רגיש.

בנוסף, קיימות השלכות משפטיות, שכן שימוש בטכניקת Proxy DLL עלול להיחשב כהפרה של תנאי שימוש, זכויות יוצרים או חוקי מחשוב ואבטחת מידע במדינות שונות.

לפיכך, מומלץ להכיר היטב את החקיקה הרלוונטית.

כדי לצמצם את הסיכונים, מומלץ להקפיד על הפרקטיקות הבאות: חתימה דיגיטלית על קבצי DLL לגיטימיים כדי לאפשר זיהוי חד משמעי של קבצים אותנטיים על ידי מערכת ההפעלה, שימוש בכלי ניטור כמו Event Viewer או Sysmon לזיהוי טעינה חריגה של DLL, אימות נתיבים והימנעות מהסתמכות על ספריות DLL בספרייה המקומית של האפליקציה ללא צורך ברור, והפעלת מנגנוני הגנה כמו AppLocker, (WDAC) Windows Defender Application Control או Windows Defender Application Guard (WDAG) שיכולים לסייע בהגבלת טעינה של DLL לא מורשים ובכך למנוע סיכונים של Proxy DLL.

סיכום והמשך הדרך

לאורך המאמר נחשפנו לפוטנציאל העצום שטמון בטכניקת Proxy DLL לא ככלי עזר תמים, אלא כמנגנון שמאפשר שליטה עמוקה על תהליכי ריצה של תוכנות קיימות. ברגע שהספרייה שלנו נטענת במקום הספרייה המקורית, אנחנו לא רק "עדים" להתנהגות התוכנית אנחנו משתתפים פעילים בה, לפעמים אף מכתיבים אותה.

היכולת להיכנס לתוך התווך הזה בין קריאה לפונקציה לבין ביצועה פותחת עולם שלם של יישומים: מתיקון באגים בספריות סגורות, דרך הרחבה של יכולות תוכנה קיימת, ועד לניטור, reverse engineering, ואפילו פיתוח אמצעי הגנה או התקפה בעולם אבטחת המידע.

מדובר באחד הכלים הבודדים שמעניק למפתח גישה ישירה לדופק של תוכנה רצה גישה שבעזרת תחכום, זהירות והבנה עמוקה, יכולה להפוך ליתרון אדיר.

עם זאת, חשוב לומר את המובן מאליו: העוצמה של Proxy DLL טומנת בחובה גם סיכון.

אותו ידע יכול לשמש גם לגרום נזק, אם ינוצל לרעה. לכן, לצד הבנת הטכניקה, חשוב להחזיק גם בעקרונות אתיים ובמודעות מלאה למרחב החוקי שבו אנו פועלים. הפעלת קוד בתוך תהליך של תוכנה אחרת במיוחד ללא ידיעתה הוא תחום רגיש, ולא תמיד חוקי. הקו בין מחקר אבטחתי לגיטימי לבין פעילות מזיקה דק מאוד ולעיתים בלתי נראה.

על המחבר

אילן וינוגרד, סטודנט לתואר ראשון במדעי המחשב וחובב פיתוח Low-Level ,

מערכות הפעלה ועולמות ה-GPU.

Linkedin-Ilan-Vinograd

GitHub-Ilan-Vinograd

מקורות מידע

- https://learn.microsoft.com/en-us/sysinternals/downloads/procmon
- https://ntcore.com/explorer-suite/
- https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order
- https://learn.microsoft.com/en-us/defender-endpoint/overview-endpoint-detection-response
- https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon
- https://learn.microsoft.com/en-us/windows/security/application-security/application-

control/app-control-for-business/

- https://learn.microsoft.com/en-us/windows/security/application-security/application-control/app-control-for-business/applocker/applocker-overview
- https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/microsoft-defender-application-guard/md-app-guard-overview