



HOOKING

תרגול 4 – הנדסה לאחור – חורף תשפ"א
© טל שנקר, נערך ע"י עידן רז



HOOK

תזכורת

- שיטה המשמשת לשינוי ההתנהגות של מערכת ההפעלה, תוכנה או קוד מכונה באמצעות יירוט קריאה לשגרה או הודעה המועברת בין תכניות.
- קטע הקוד האחראי ליירוט נקרא Hook.
- אנחנו נתמקד בעיקר בשינוי התנהגות של תוכנות.

HOOKING - שימושים

- חיפוש באגים, ביצוע logging, מחקר של תוכנות.
- הוספת פיצ'רים או הרחבת אפשרויות השימוש בפונקציונליות קיימת.
- יצירת תוכנות זדוניות – הסוואה בתוך תהליכים לגיטימיים.

דוגמה בכיתה



HOOKING IN A NUTSHELL

- אנחנו נפריד בין שני דברים – השיטה בה אנחנו משנים את התוכנה והשיטה בה אנחנו מתקינים את הHook.
- דרכים עיקריות להתקנת Hook:
 - Hot Patching
 - דריסה וקפיצה
 - Hook IAT – עבור פונקציות מספריות דינמיות
- דרכים עיקריות לשינוי תוכנות –
 - שינוי פיסי של קוד המקור/קובץ ההרצה
 - שינוי בזמן ריצה – ע"י Dll Injection

HOT PATCHING

Original function
prologue code

90	nop	
90	nop	
90	nop	
90	nop	
90	nop	
89 FF	mov	edi, edi
55	push	ebp
8B EC	mov	ebp, esp
51	push	ecx
33 DB	xor	ebx, ebx
...		

Hooked function prologue code,
First bytes overwritten with jump instruction

E9 01 02 03 04	jmp	Hook
EB F9	Jmp short	-7
55	push	ebp
8B EC	mov	ebp, esp
51	push	ecx
33 DB	xor	ebx, ebx
...		

```
hook:  
// hook function code
```

דריסה וקפיצה

Original function
prologue code

```
55      push      ebp
8B EC    mov       ebp, esp
51      push      ecx
51      push      ecx
53      push      ebx
33 DB    xor       ebx, ebx
...
```

Hooked function prologue code,
First bytes overwritten with jump instruction

```
E9 01 02 03 04    jmp      Hook
// 55             push      ebp
// 8B EC          mov       ebp, esp
// 51             push      ecx
// 51             push      ecx
53              push      ebx
33 DB            xor       ebx, ebx
...
```

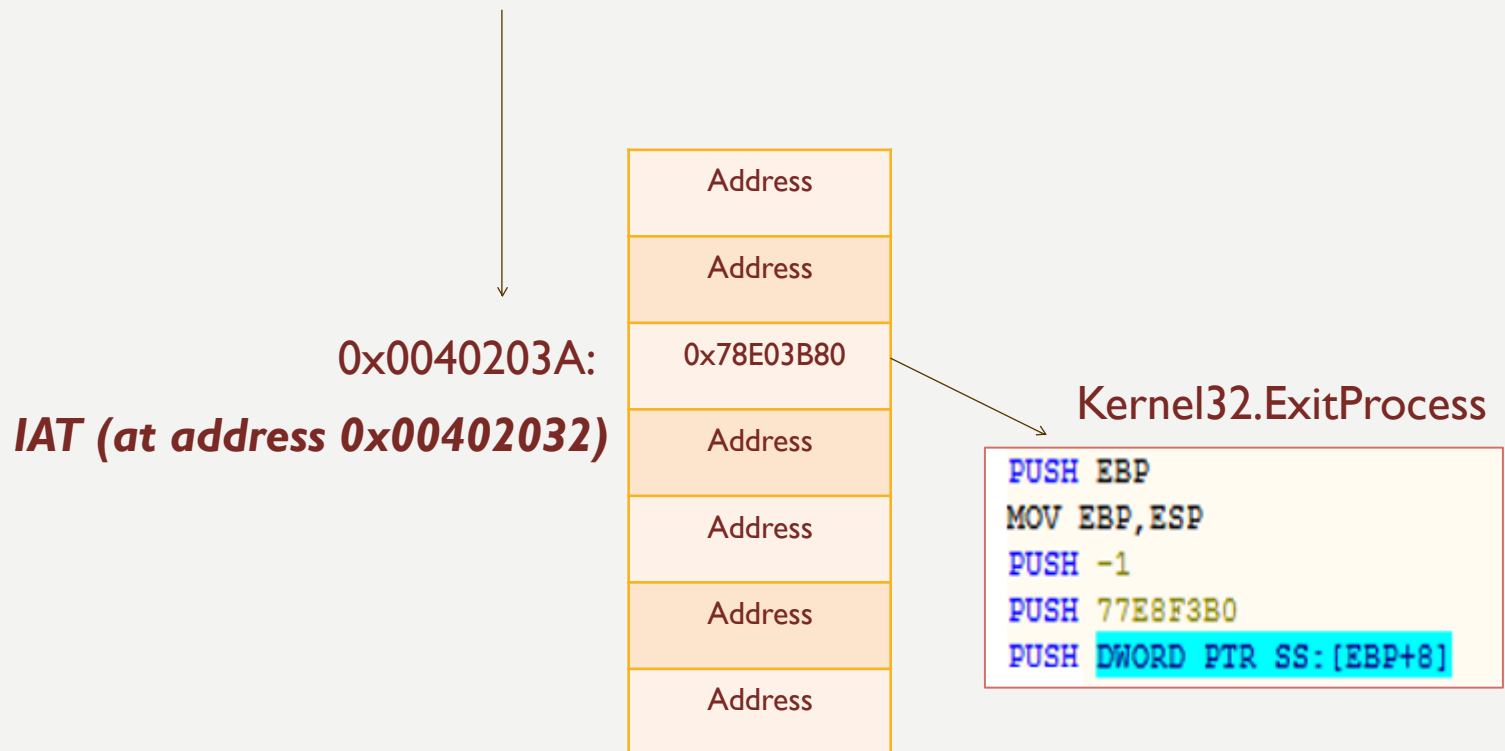
```
hook:
// hook function code
```

IMPORT TABLES

תזכורת מהתרגול על PE

נניח שהתוכנית שלנו נטענה ל-0x00400000, ובכתובת 0x0040657A יש את הפקודה הבאה:

```
CALL DWORD PTR DS:[0040203A]; // call to ExitProcess
```



הוקינג באמצעות ה-IAT



Memory

Calc.exe (0x00D80000)

PE Header

Code Section

Data Section (+IAT)

...

Gdi32.dll (0x77950000)

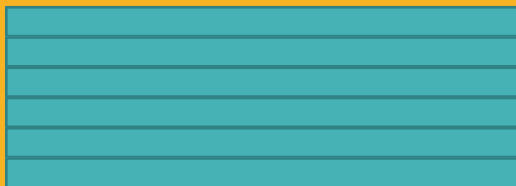
PE Header

Code Section

Data Section

...

More DLLs



IAT at 0x00E02000

Address (In Prog.exe)	Address To Function
0x00E02000	0x78890201
0x00E02004	0x78610201
0x00E02008	...
0x00E0200c	...
0x00E02010	...
0x00E02014	...
0x00E02018	...
0x00E0201c	...

שינוי פיסי

- נרצה לבצע hook ע"ש שינוי פיסי של הקובץ הבינארי.
- באיזה מקום ניתן לכתוב את הפונקציה שלנו?
 - ניתן לחפש אזור "נטוש" בתכנית ולדרוס אותו עם הקוד שלנו.
 - יש לוודא שלא נגרם נזק לתוכנית.
- ניתן להשתמש בתוכנת עריכת PE על מנת להוסיף Section נוסף לקובץ.





HOOK בזמן ריצה

DLL INJECTION

Anything that you could do with byte patching, you can do with DLL injection.

- תהליך בו מזריקים למרחב הזיכרון של תהליך קוד ומריצים אותו תחת ההקשר של אותו התהליך. פלטפורמה מעולה לביצוע ה hooking.
- כאשר מבצעים DLL Injection:
 - ניתן לכתוב את ה patch שלנו כ DLL בקוד C במקום ב Assembly.
 - אין צורך לחפש אזורים "נטושים" בבינארי הקיים.
 - אין צורך לערוך בתים בבינארי המקורי.
 - עדיין צריך לשחזר את הפקודות שדרסנו.

איך עובד INJECTOR



• ניתן לסכם את פעולת ה injector ב 4 שלבים:

▪ CreateProcess – יצירת תהליך חדש עבור התכנית שלנו עם הרשאות קריאה כתיבה והרצה (RWX).

▪ VirtualAllocEx – הקצאת זכרון עבור שם ה DLL שאנו מעוניינים להזריק. 

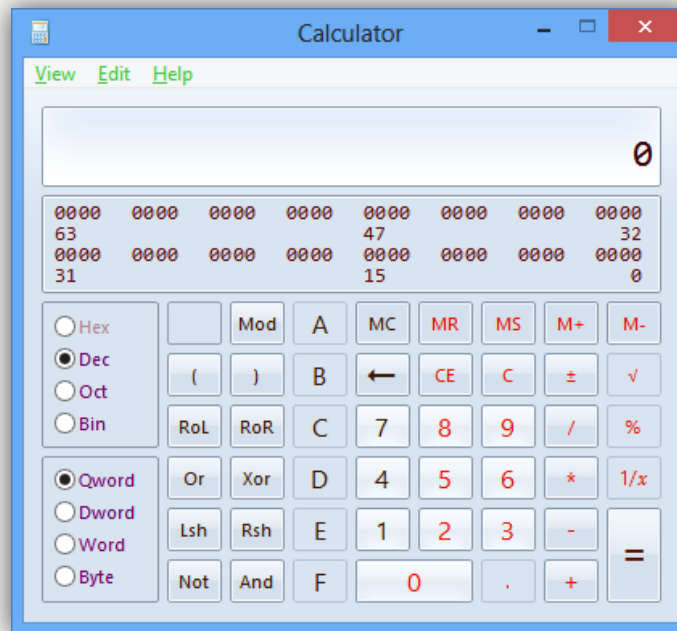


▪ WriteProcessMemory – כותבים את שם ה DLL לתוך ה chunk שהוקצה.

▪ ל CreateRemoteThread – יצירת תהליך חדש עבור טעינת ה DLL באמצעות LoadLibraryA ולאחריה ביצוע של DllMain.

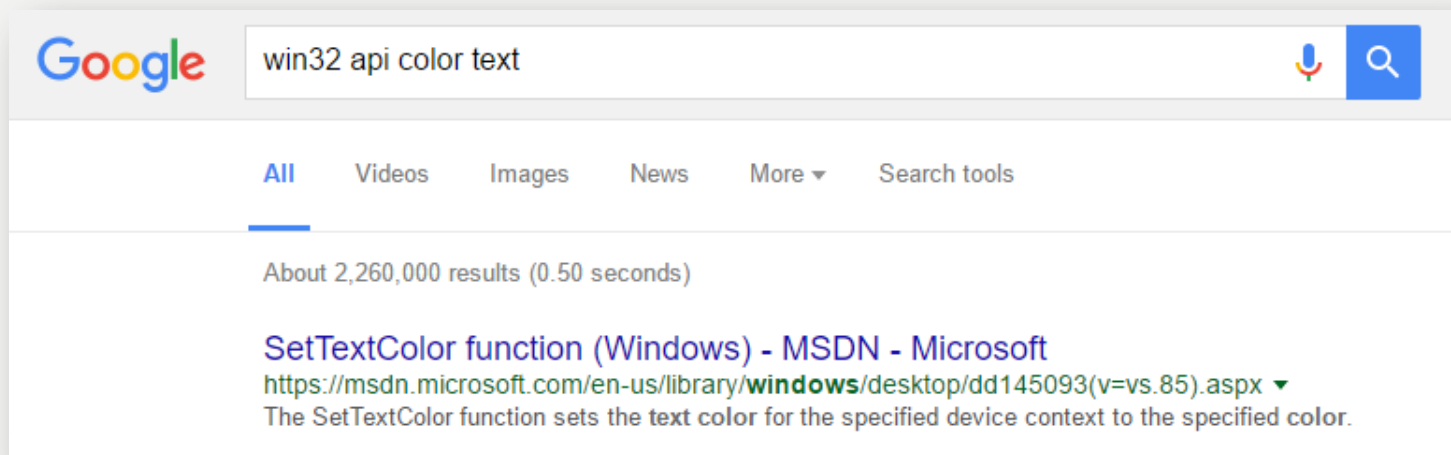
שינוי צבעים במחשבון

- נרצה לגרום לתוכנת המחשבון לשנות צבעים באופן אקראי.
- נניח שיש פונקציה מסוימת (של מ"ה) שאחראית על הגדרת צבע וננסה לגלות מהי.



שינוי צבעים במחשבון

- שלב ראשון – גילוי שם הפונקציה.
- אפשר לעשות מחקר קצר בגוגל:



- וניתן גם לעבור על ה Import Table של calc.exe:

010011B0	.text	Import	USER32.SetProcessDefaultLayout
01001014	.text	Import	GDI32.SetTextColor

שינוי צבעים במחשבון

• תיעוד ב-MSDN:

SetTextColors function

The **SetTextColors** function sets the text color for the specified device context to the specified color.

Syntax

C++

```
COLORREF SetTextColors(  
    _In_ HDC hdc,  
    _In_ COLORREF crColor  
);
```

Parameters

hdc [in]

A handle to the device context.

crColor [in]

The color of the text.

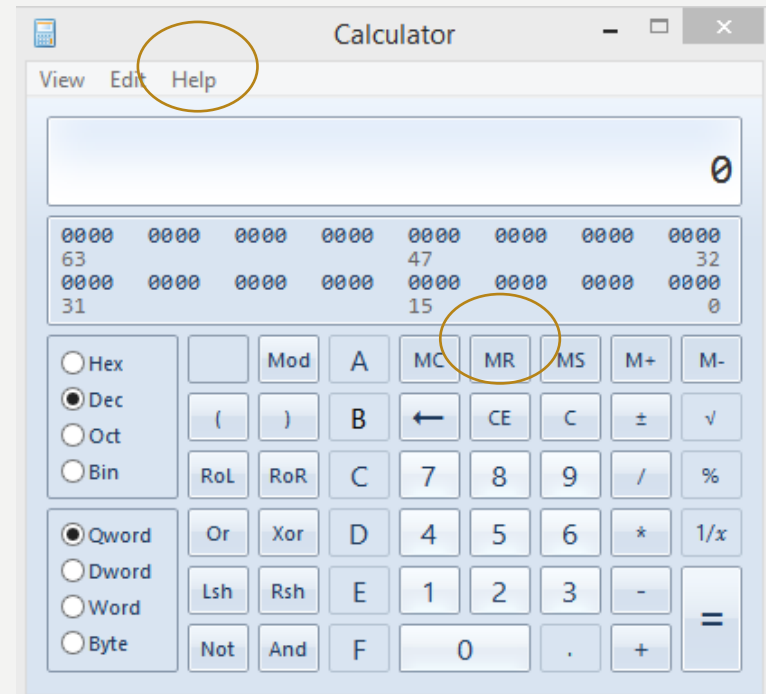
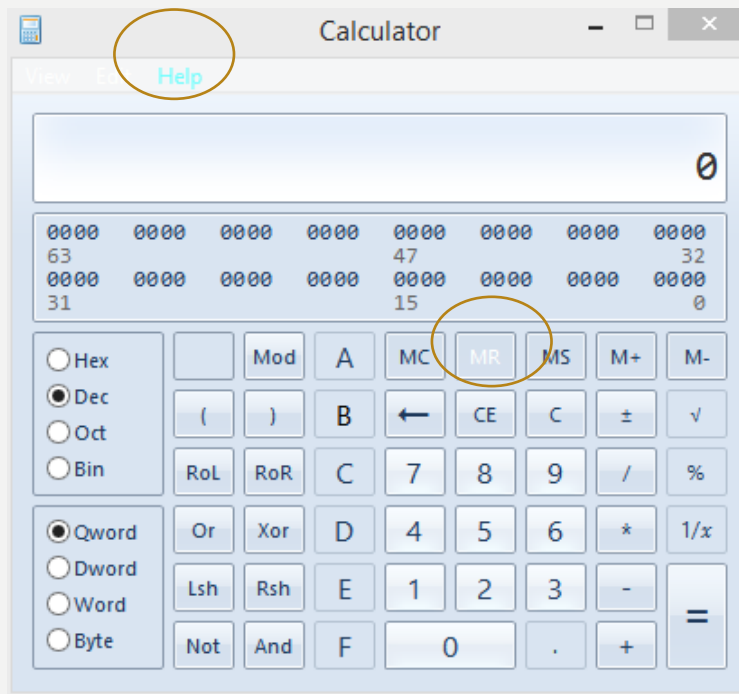
שינוי צבעים במחשבון

- הפונקציה `setTextColor` נמצאת ב-GDI32.dll.
- נגיע לקוד של הפונקציה בעזרת Debugger
- על מנת לוודא ש-`setTextColor` היא אכן הפונקציה הרלוונטית, נשים עליה Breakpoint ונבדוק כיצד שינוי הפרמטר `Color` משפיע על התנהגות התוכנה.
- המחסנית בתחילת הפונקציה (בעת עצירה ב-BP):

008CC764	74096860	'h.t	[CALL to SetTextColor from UxTheme.7409685A hDC = 1B011306 Color UNICODE "&View"
008CC768	1B011306	♠!!0←	
008CC76C	006D6D6D	mmm.	
008CC770	008CC9F8	° 𐄂î.	

הצבעים משתנים!

- ניתן להבחין בשינויים בצבעים (כתוצאה משינוי הפרמטר Color).



- בשקפים הבאים ניישם את זה בקוד...

TEXT COLOR HOOK

- כזכור מההרצאה, בתוך ה-DLL היה קוד אסמבלי שנקרא לפני הפונקציה המקורית.
- דוגמא לקוד מותאם עבור הפונקציה `SetTextColor`:

```
__declspec(naked) void textColorHook()
{
    __asm {
        mov eax, [esp+0x20]           ; Some "random" value from the stack.
        and eax, 0x00FFFFFF           ;
        mov [esp+0x8], eax            ; COLORREF
                                     ; SetTextColor(_In_ HDC hdc,
                                     ; _In_ COLORREF crColor);

        nop
        nop                           ; Jump to the real SetTextColor
        nop
        ...
    }
}
```

SETHOOK

קוד מלא

```
void setHook() {
```

```
    LPVOID f;
```

```
    HMODULE h = GetModuleHandle(L"gdi32.dll");
```

```
    // jmp (far, relative) 0x90909090
```

```
    CHAR JumpOpcode[6] = "\xE9\x90\x90\x90\x90";
```

```
    // move eax, 0x90909090
```

```
    // jmp (far, absolute) eax
```

```
    CHAR JumpOpcode2[8] = "\xB8\x90\x90\x90\x90\xFF\xE0";
```

```
    DWORD lpProtect = 0;
```

```
    LPVOID JumpTo;
```

```
    if (h == NULL) { return; }
```

```
    f = GetProcAddress(h, "SetTextColor");
```

```
    if (f == NULL) { return; }
```

```
    // calculate relative jump to textColorHook from f
```

```
    JumpTo = (LPVOID)((char*)& textColorHook - (char*)f);
```

SETHOOK

קוד מלא

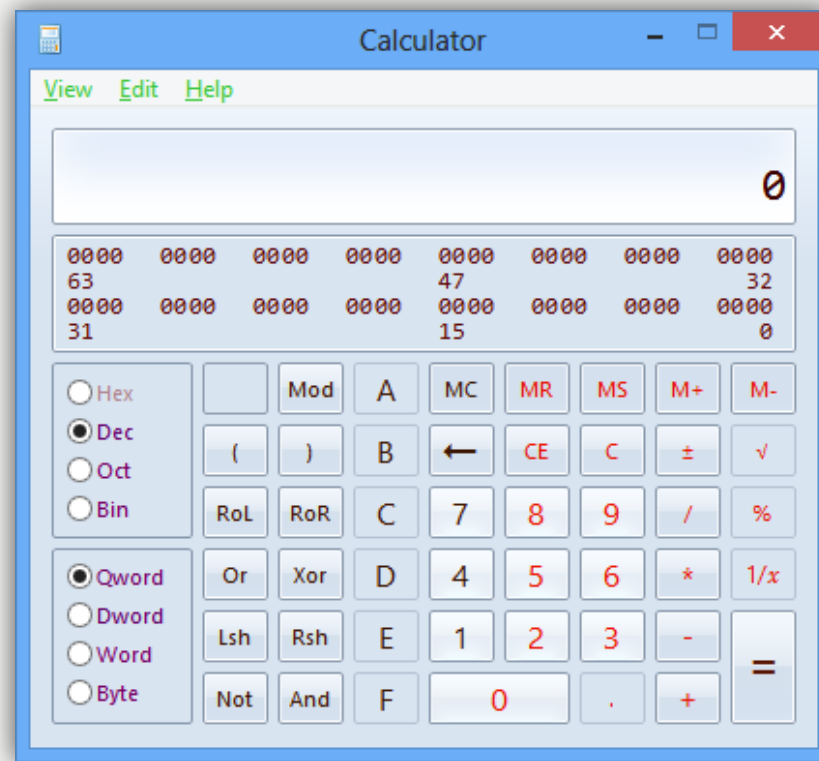
```
memcpy(JmpOpcode+1, &JumpTo, 0x4); // write the jump
VirtualProtect((char*)f-5, 0x7, PAGE_EXECUTE_READWRITE, &lpProtect);
memcpy((char*)f-5, &JmpOpcode, 0x5); // override the first two bytes with jmp short
*(char*) f = 0xEB;
*((char*)(f)+1) = 0xf9;
VirtualProtect((char*)f-5, 0x7, PAGE_EXECUTE_READ, &lpProtect);

// jump to f from textColorHook
JumpTo = (LPVOID)((char*)f + 2);
memcpy(JmpOpcode2+1, &JumpTo, 0x4);
VirtualProtect((char*) textColorHook, 0x100, PAGE_EXECUTE_READWRITE, &lpProtect);
memcpy((char*)& textColorHook + 0x1f, &JmpOpcode2, 0x7);
VirtualProtect((char*) textColorHook, 0x100, PAGE_EXECUTE_READ, &lpProtect);
```

}

שינוי צבעים במחשבון

הצלחנו!



שאלות

- מה קורה אם יש ALSR ?
▪ כלומר לא ניתן לדעת מראש לאיפה GDI32.DLL יטען בזכרון.
- איך נעשה Hook יותר גנרי?
▪ בדוגמא הנחנו שהפונקציה מתחילה ב `mov edi,edi`.
▪ כלומר שיש 2 בתים שניתן לדרוס.
▪ וגם הנחנו שיש 5 בתים פנויים לפני הפונקציה.
- איך נבצע את ה-Hook מבלי לשנות בזכרון (או בדיסק) את GDI32.DLL ?

תשובות

- איך נעשה Hook יותר גנרי?
 - בהנחה שאנחנו יודעים מראש באילו פקודות הפונקציה מתחילה:
 - נעתיק את הפקודות האלו לפונקציית ה-hook שלנו ונדרוס אותן (בפונקציה המקורית) עם jump לפונקציה שלנו.
 - את הפקודות שהעתקנו נמקם לפני ה-jmp שקופץ חזרה לפונקציה המקורית. ה-jmp יקפוץ להמשך הפונקציה - אחרי הפקודות שנדרסו.
 - אחרת (אם לא ידוע לנו באילו פקודות הפונקציה מתחילה, למשל אם הפונקציה משתנה בין גרסאות שונות של מערכת ההפעלה) נצטרך לגלות בזמן ריצה באילו פקודות הפונקציה מתחילה ומה הגודל שלהם (לכל פקודה גודל שונה) ולבצע את ההעתקות בהתאם.

תשובות

- מה קורה אם יש ALSR ?
 - מאחר ואיננו משתמשים בכתובות קבועות בקוד אלא מוצאים את הכתובות בצורה דינמית, אין שום בעיה עם ASLR.
- איך נבצע את ה-Hook מבלי לשנות בזיכרון (או בדיסק) את GDI32.DLL ?
 - נערוך את ה-Import table, כפי שמודגם בשקפים הבאים

הוקינג באמצעות ה-IAT - תזכורת

Memory

Calc.exe (0x00D80000)

PE Header

Code Section

Data Section (+IAT)

...

Gdi32.dll (0x77950000)

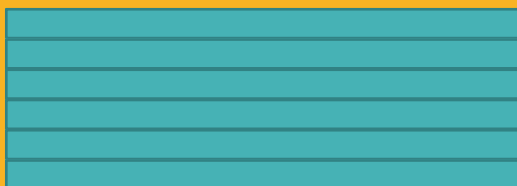
PE Header

Code Section

Data Section

...

More Dlls



IAT at 0x00E02000

Address (In Prog.exe)	Address To Function
0x00E02000	0x78890201
0x00E02004	0x78610201
0x00E02008	...
0x00E0200c	...
0x00E02010	...
0x00E02014	...
0x00E02018	...
0x00E0201c	...

הוקינג באמצעות ה-IAT



- מצאנו את המקום הרלוונטי ב-IAT.

- (מצביע ל-SetTextColor).

00E0245C	.idata	Import	GDI32.CreateSolidBrush
00E02460	.idata	Import	GDI32.SetTextColor
00E02464	.idata	Import	GDI32.GetDeviceCaps
00E02468	.idata	Import	GDI32.CreateCompatibleDC

- ניתן לראות שהכתובת **0x00E02460** מכילה את המצביע.

- התוכנית נטענה לכתובת Base = 0x00D80000.











- מה ה-Offset של המצביע מתחילת התוכנית?

- $0xE02460 - 0xD80000 = 0x82460$



הוקינג באמצעות ה-IAT

```
void setIATHook() {  
    LPVOID f;   
    HMODULE h = GetModuleHandle(L"calc.exe");  
    HMODULE h2 = GetModuleHandle(L"gdi32.dll");  
    CHAR JumpOpcode2[8] = "\xB8\x90\x90\x90\x90\xFF\xE0";  
    DWORD lpProtect = 0;  
    LPVOID JumpTo;  
    LPVOID IAT;  
  
    if ((h == NULL) || (h2 == NULL)) return;  
  
    f = GetProcAddress(h2, "SetTextColor");  
    if (f == NULL) return;  
  
    iat = h + 0x82460 / 4;    
  
    VirtualProtect((char*)IAT, 0x4, PAGE_EXECUTE_READWRITE, &lpProtect);   
    JumpTo = (LPVOID)((char*)&textColorHook);   
    memcpy(iat, &JumpTo, 0x4);   
  
    memcpy(JumpOpcode2 + 1, &f, 0x4);   
    VirtualProtect((char*) textColorHook, 0x100, PAGE_EXECUTE_READWRITE, &lpProtect);  
    memcpy((char*)&textColorHook + 0x1f, &JumpOpcode2, 0x7);   
}
```

רגע סדר

IAT Hooking	Hot Patching	Hook רגיל	
את הכניסה המתאימה לפונקציה ב IAT	רצף של פקודות לפני גוף הפונקציה: nop*, mov, edi, edi	רצף של פקודות בגוף הפונקציה	מה דורסים
לא צריך לשחזר	לא צריך לשחזר	את רצף הפקודות שנדרס	מה צריך לשחזר
על פונקציות שנטענות מתוך DLL 	על פונקציות "מוכנות" ל hook	בעיקרון תמיד ועל כל פונקציה	מתי ניתן להשתמש
דינאמי	פיסי או דינאמי	פיסי או דינאמי 	מתי מבצעים



איך מדבגים?

- ראשית עלינו לעסוק בשאלה. כיצד נתפוס עם ה debugger תהליך "שמסתיים מיד" והוא לא נוצר באופן ישיר על ידינו?



- במצב כזה פתרון אלגנטי הוא למקם בתחילת הקוד (או בכל נקודה אחרת ממנה נרצה להתחיל) את הקוד הבא:

```
while (IsDebuggerPresent() == false) {  
    Sleep(1);  
}
```



- קוד זה יוודא שהתכנית נמצאת בלולאה שמונעת ממנה לצאת עד שנחבר את ה debugger לתהליך.

איך מדבגים?

- כעת נבחין בין כמה מקרים:
 - נרצה לדבג את ה Injector – אין צורך בטיפול מיוחד, ה injector הוא תכנית רגילה וניתן פשוט להשתמש ב debugger.
 - נרצה לדבג את ה hook עצמו. עלינו לעצור את התהליך לפני שהוא מגיע ל hook. ניתן להשתמש בשיטה מהשקף הקודם.
 - נרצה לדבג את DllMain. במקרה כזה לא מספיק לעצור את הפונקציה עצמה (שוב כבשקף הקודם), נצטרך גם לעצור את ה Injector מלבצע ResumeThread. מדוע?

