# תרגיל בית 4 בהנדסה לאחור

Vulnerabilities

דרור כרמון 036861292

עידו מנגר 212324313

<u>חלק יבש</u>

1. יתכן שרצף הפקודות 60 C3 הוא חלק מפקודה ארוכה יותר שתחילתה כתובת נמוכה מ-0x9ad9e71c אבל כאשר מפרשים אותה מהכתובת הנ"ל היא מתורגמת לפעולות שהתקבלו.
למשל הפקודה המלאה בגודל 3 בתים היא 24 60 C3 והיא מתורגמת לפקודות:

```
0x9ad9e71b:  24 60     and    al,0x60
0x9ad9e71d:  c3        ret
```

2) נשתמש בפונקציה SetSecurityInfo של windows כדי לשנות הרשאות לקובץ

Part1-

9ad9e700- 3 pops for 3 args

70707070- file handler

00000001- SE_OBJECT_TYPE Enum value that means FILE_OBJECT
11111111- Security information parameter, we want permissions will be set all to 1 including read/write

SetSecurityInfo -windows function that sets the security info of the file to what we gave it- we now should have all of the permissions

Part 2-

9ad9e71a -Pop edi- edi will be the address to pop 3 items from the stack

9ad9e700 - Pop 3 from the stack

9ad9e713- Pop 2 from the stack- put the address in ebx

7070 707C- the id minus F plus 27

Ffffffff-filler

9ad9e706-will zero eax

9ad9e700- put  ebx in eax

9ad9e713- Pop 2- skip next 2 lines

Ffffffff

ffffffff

9ad9e716-Derefrence eax+0fh- eax will be  the address of the file

9ad9e71c-Pushad -this command will push all of the registers, this will become eax,edi is pop 3, which will pop 3 from the stack and then jump to esp that we push(previous esp, meaning it will go back here,this will now be eax meaning that from here we will jump to the contents of eax that is the first page of t he file

2. כשstrcpy מקבלnull char  הוא מסיים את הקליטה ומפרש את זה כסוף הקלט, בגלל שאנחנו מכניסים תווים כאלה בחלק הראשון, הקלט יפסק שם ומה שהכנסנו לא ימשיך לדרוס את המחסנית.
3. בשביל לתקן את זה נרצה להשתמש בגאדגטים החדשים כל פעם ששמנו null char כדי להמנע מלשים null shar,

הכתובת 9ad9e700 היא הבעייתית, אם נשים 1AD9E6FF נפעיל עליו את הגאדגט השני ואז את הראשון אז נקבל את הכתובת לכן נשים בהתחלה במקום הכתובת את הקטע הבא-

6ad6e720-not
9ad9e711-will call the next line
6ad6e71d-neg
1AD9E6FF-eventually will become the address we want

החלק הבעייתי האחר הוא ה 00000001 של הenum

שבמקרה הוא שווה ערך לfffffffd not – נרצה להשתמש בגאדג'ט השני שלנו, נשים לב שמקודם הייתה לנו שורה שרק דילגנו עליה, עכשיו נחליף אותה בפעולה ובגלל שהיא מדלגת על שני כתובות היא תפעל על המקום שאנו רוצים

6ad6e720-not
6ad6e720-will call the next line
6ad6e71d-neg
1AD9E6FF-eventually will become the address we want
fffffffd- will become the enum we want
On the second part we also need
6ad6e720-not
9ad9e711-will call the next line
6ad6e71d-neg
1AD9E6FF-eventually will become the address we want
Instead of 9ad9e700 in the second place it shows, in the first place we will change it to: 9ad9e713 - Pop 2 from the stack- now it will go to ebp instead esp and then instead of ffffffff filler when we call  the pop ebx ebp gadget we will put in ebp the inc 1 gadget which will make it advance one line and then jump to esp, since it increases eax by 1 we will also need start eax from 1 less.

# חלק רטוב

## שלב ראשון – התחברות

המשתמש בעל הרשאות גבוהות ביותר הוא archer, לאחרת ההתחברות ניתן לראות כיתוב "Welcome archer (Admin)" שמעיד שהוא משתמש מסוג Admin. בנוסף הפעולות שהוא יכול לבצע מתקדמות יותר משאר המשתמשים (LOAD/PEEK) ומאפשרות גישה למערכת הקבצים של השרת.

## שלב שני – כתיבת shellcode

1. בעזרת ida פתחנו את client.exe וחיפשו חולשות במקומות בהם מקבלים קלט מהמשתמש. ראינו שלאחר הכנסת הפקודה PEEK נדרש לתת קלט נוסף בעזרת scanf אשר אינה מוגנת ומאפשרת הכנסת קלט באורך לא מוגבל.
2. כדי לנסות ולתקוף את הקובץ client.exe בעזרת Stack overflow יצרנו תהליך בעזרת subprocess.Popen והעברנו אליו קלטים באמצעות pipe.
   התחלנו מהתקפה בסיסית בדומה למה שעישנו בסדנה בעזרת הכנסת קלט
   attack = "TRUN . ".join([f"{str(n).zfill(3)}A" for n in range(length)])
   הגדלנו את length עד שזיהינו ש-client.exe קרס.
   השתמשנו בסקריפט המצורף attack_shell_v0.py
3. חזרנו על שלב שני הפעם עם debugger שמאזין על התהליך של client וראינו שבזמן הקריסה מתקבלת ההודעה שגיאה Access violation executing location 0x36383141. כלומר שכנראה דרסנו כתובת חזרה וכאשר התבצעה פקודת ret תוכן המחסנית (A186) הועתק לEIP והתוכנית ניסתה לרוץ בכתובת זו. כדי להבין איזה חלק בדיוק מהמחרוזת נפל על כתובת החזרה, הסתכלנו על תוכן המחסנית בכתובת ב-ESP וראינו שהוא: A1868(A186)9A – בסוגרים החלק שהועתק לEIP.
   כלומר גודל המחרוזת עד לנקודת הדריסה הרצויה הוא: 1 – 5*869 + 4*1000
4. חיפשנו בעזרת ropper כתובת שבה מופיע הפקודה jmp esp ומצאנו אותה בכתובת 0x62502028. והתחלנו לכתוב את ה- payload שלנו. בהתחלה שמנו את פרטי ההתחברות של המשתמש, לאחר מכן מילאנו בתו 'A' עד לגודל שחישבנו ואז שמנו את כתובת של jmp esp, בתור התחלה לאחר מכן שמנו פקודת int3 ולאחריה מספר פקודות nop כדי לראות שהצלחנו. היינו צרכים לבטח DEP עבור הריצה של client.exe
5. בעזרת ida דיבגנו את התוכנית בריצה רגילה ובריצה עם הattack shell ומצאנו שלאחר ביצוע jmp esp ה- socket נמצא ב esp+12. המשמעות היא שלא נוכל להגדיל את הpayload יותר מידי כי נדרוס את הsockt. לכן לאחר jmp esp שמנו פקודת jmp לחלק ריק ב-payload שם נכתוב את שאר הקוד.
6. שמנו לב בשלב זה שאיאפשר לכתוב בpayload בית עם ערך 0 כי הוא מועתק ע"י strcpy ולכן בכל מקום שנדרש לכך נכניס את שלילי ונבצע עליו neg.
7. כדי להשתמש במחסנית גם לקוד וגם לאכסון, חילקנו את הpayload לשני איזורים. את החלק הנמוך שמרנו עבור המחסנית וכיוונו אליו את esp ובחלק הגבוה יותר שמנו את הקוד.
8. בעזרת ida דיבגנו את התוכנית כדי להבין איך נתקשר עם השרת. מצאנו פונקציה שנמצאת בכתובת 0x625015CB אשר מקבלת אגומנטים: 1. Socket 2. מצביע למחרוזת עם פקודה לשרת. הפונקציה גם מעתיקה אל הכתובת הקלט את תשובת השרת. בנוסף מצאנו כי הbuffer המועבר אליה נמצא בכתובת 0x62508080 ונוכל להשתמש בוא לשליחה/קבלה של הודעות.
9. כדי לקבל קלט השתמשנו בפונקצית scanf פרמטר "%s" וכדי להדפיס את ההודעות שחזרו מהשרת השתמשנו ב- printf. את כל הכתובות מצאנו בעזרת ida.

10. כדי שנוכל להמשיך ולהכניס קלטים לאחר הדפסת התשובה מהשרת, השתמשנו בjmp כדי לחזור לקוד שמבצע scanf.
11. נתקלנו בבעית סנכרון בין הpython ל-client.exe בקריאה / כתיבה של ה-pipe בניהם. כדי לפתור את זה בתחילת כל "מחזור" של הpayload הדפסנו את המחרוזת " Mingw runtime failure:" כדי שהpython ידע שהוא הגיע לסוף הפלט ולא יחכה לפלט נוסף, מצב שגורם לdeadlock כי הpayload מחכה בשלב זה לקלט. השתמשנו במחרוזת זו היא מסתיימת ב- \n ולכן ה-python קורא אותה כשורה. בנוסף הכתובת שלה הייתה זמינה ברשימת המחרוזות ב-ida.
12. כתובות אבסולוטיות שהשתמשנו בהן ב-payload חישבנו בעזרת דיבאג ע"י ערכי ESP/EIP כאשר ההסתמכות היא שאין ASLR ולכן הכתובות תמיד יהיו זהות.

## שלב שלישי – הרצת קוד על השרת

1. בעזרת **attack_shell.py** שיצרנו שחקנו עם השרת וראינו שפקודת PEEK מאפשר לראות תוכן של ספריות על השרת. למשל . PEEK מראה את תוכן הספריה הנוכחית, .. PEEK מראה את תוכן הספריה שמעל הנוכחית.
2. מצאנו קובץ בשם turn_off_fires.py (יכול להיות שהשם לא נכון אבל השרת כרגע לא עובד אז אני לא יכול לבדוק בדיוק את השם) שכנראה צריך להריץ כדי לכבות את השריפות.
3. ניסינו כל מיני קומבינציות של PEEK וראינו שהכנסת ערכים לא תקינים מחזיר פלט שנראה כמו Windows Power Shell. בעזרת חיפוש בgoogle נראה שפקודת PEEK למעשה מבצעת Get-ChildItem –Name <param> שמחזירה את רשימת הקבצים והתיקיות בספריה הנוכחית ע"פ הפרמטר. הפרמטר יכול להיות שם של קובץ או ספריה ספציפים, '.' , '..' או שילוב של *.
4. בWindows Power Shell ניתן לשרשר פקודות בעזרת ; וכך ניתן להעביר פקודות לשרת:

   PEEK *. sheker; date

   אשר יסצע חיפוש של קבצים עם סיומת stam, יחזיר רשימה ריקה. ואחריה יריץ את פקודת date.
5. עדכנו את ה-**attack_shell.py** שלנו כך שישרשר את הקלטים ככה שנוכל להריץ כל פקודה שנרצה.
6. הרצנו פקודת python turn_off_fires.py אשר תורגמה ל:

   PEEK *. sheker; python turn_off_fires.py

השריפות בלוח כבו !!