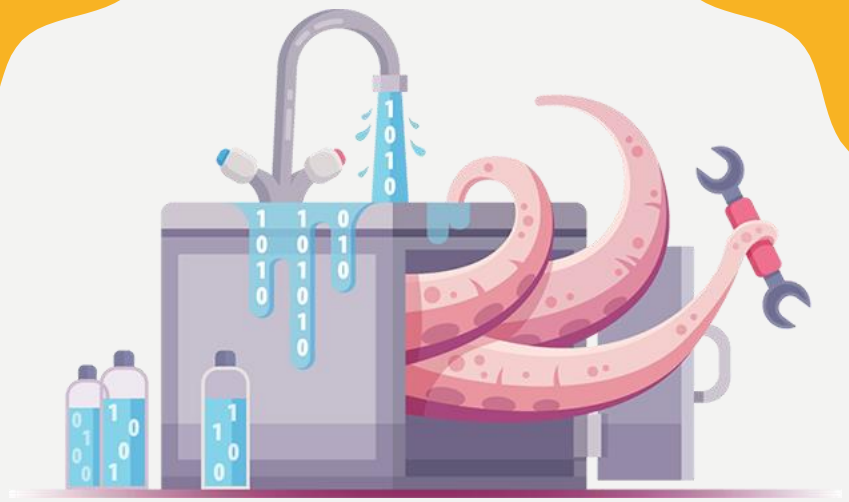


BUFFER OVERFLOW

תרגול 5 – הנדסה לאחור – אביב תשפ"ב

©אלי ביהם, אביעד כרמל, נערך ע"י טל שנקר,עידן רז
ואיתמר יובילר

חריגה מחוצץ

Buffer Overflow

- נתון הקוד הבא:

```
int main() {
        char buffer[8];
        strcpy(buffer, "AAAAAAAAAAAAAAAAAAAA");
        printf("%s\n", buffer);
        return 0;
}
```

- בשקפים הבאים נבין כיצד ניתן לשנות את הערך המועתק (AAAA...) בצורה כזו שתגרום להרצת קוד.
- בדוגמה הערך "AAAA..." הוא קבוע בקוד, אך במקרה הכללי הכוונה היא לקלט שהגיע מן המשתמש (או יותר נכון – מן התוקף).

חריגה מחוצץ

- פתיחת הקוד באמצעות Debugger נותן לנו את השגיאה הבאה (Exception):

```
[23:35:56] Access violation when executing [41414141]
```

- מצב האוגרים (registers) בעת השגיאה:

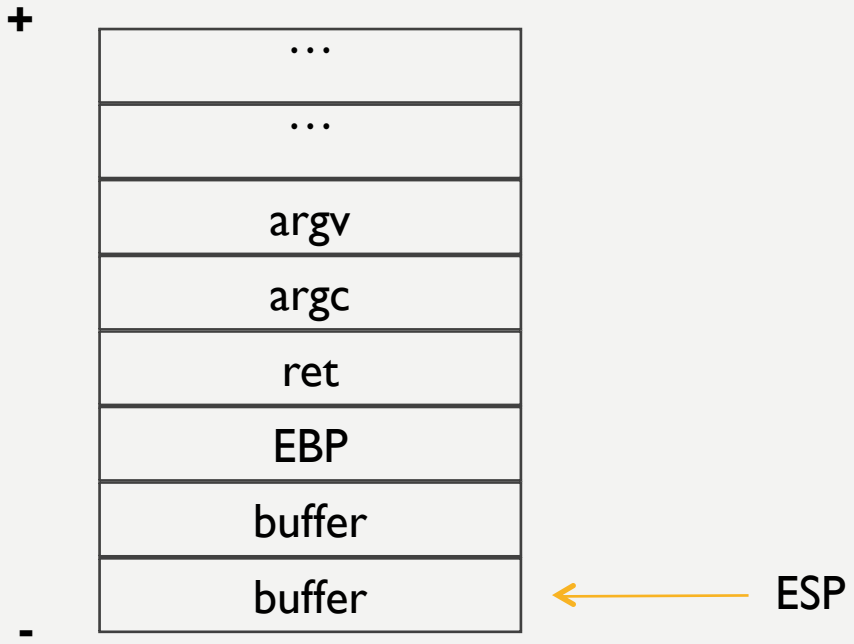
```
Registers (FPU)
EAX 00000000
ECX 7754D26A MSVCR110.7754D26A
EDX 005B7800
EBX 00000000
ESP 002DFA58
EBP 41414141
ESI 00000001
EDI 00000000

EIP 41414141

C 0  ES 002B 32bit 0(FFFFFFFF)
P 1  CS 0023 32bit 0(FFFFFFFF)
A 0  SS 002B 32bit 0(FFFFFFFF)
Z 1  DS 002B 32bit 0(FFFFFFFF)
S 0  FS 0053 32bit 7F3FE000(FFF)
T 0  GS 002B 32bit 0(FFFFFFFF)
D 0
```

חריגה מחוצץ

- מצב המחסנית לפני הקריאה ל strcpy:



חריגה מחוצץ

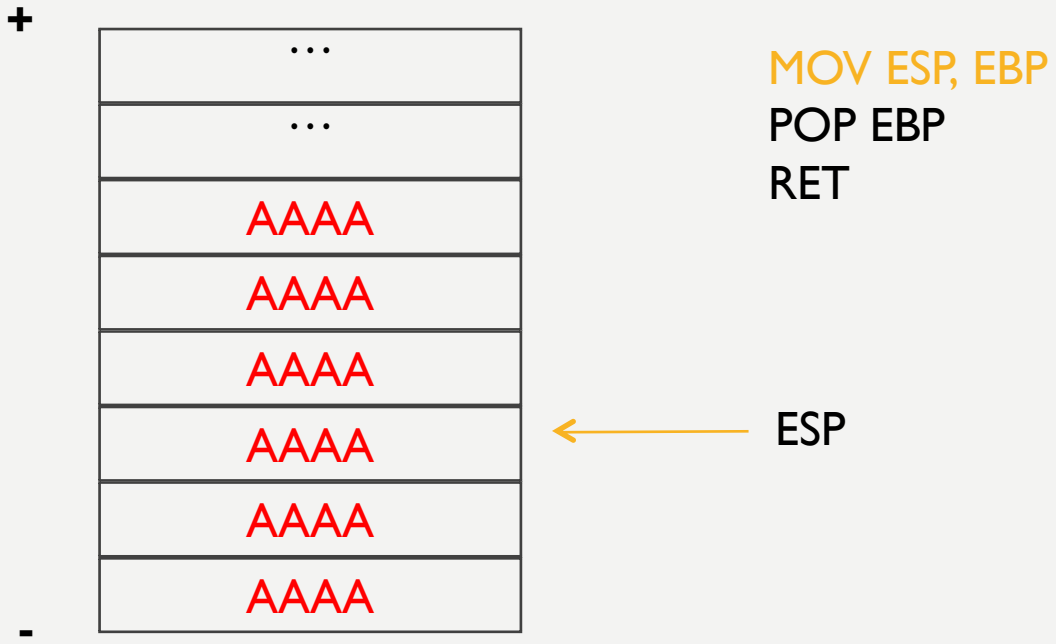
- מצב המחסנית אחרי הקריאה ל strcpy:
  - בציור המחסנית גדלה כלפי מעלה. strcpy כותב כלפי מטה.

חריגה מחוצץ

- כעת מתבצעת החזרה מפונקציית main:

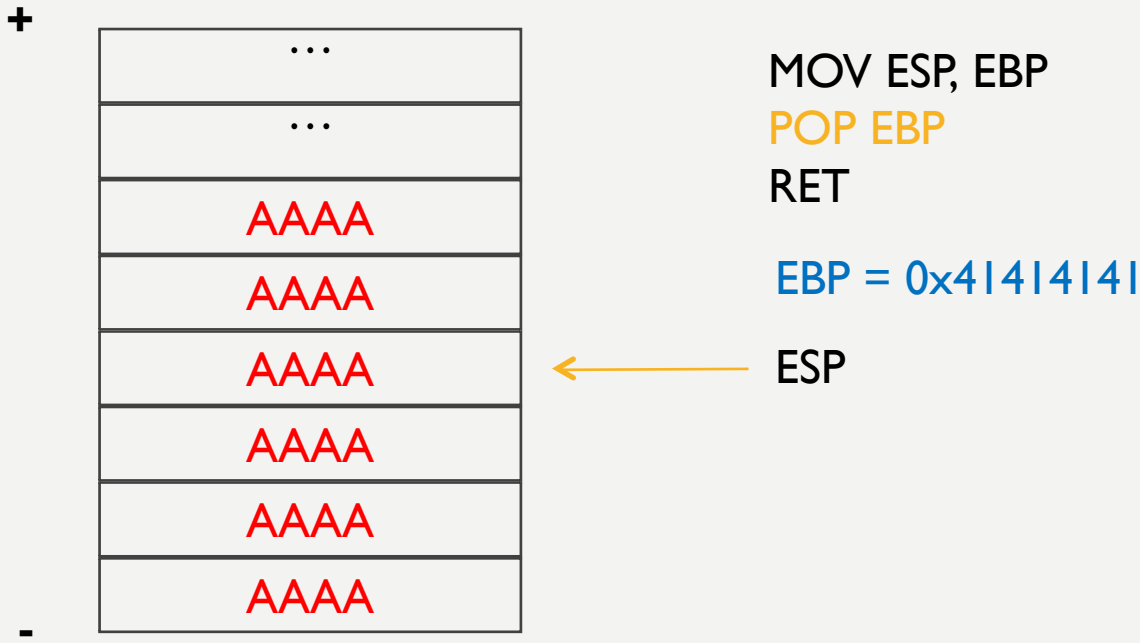
חריגה מחוצץ

- כעת מתבצעת החזרה מפונקציית main:



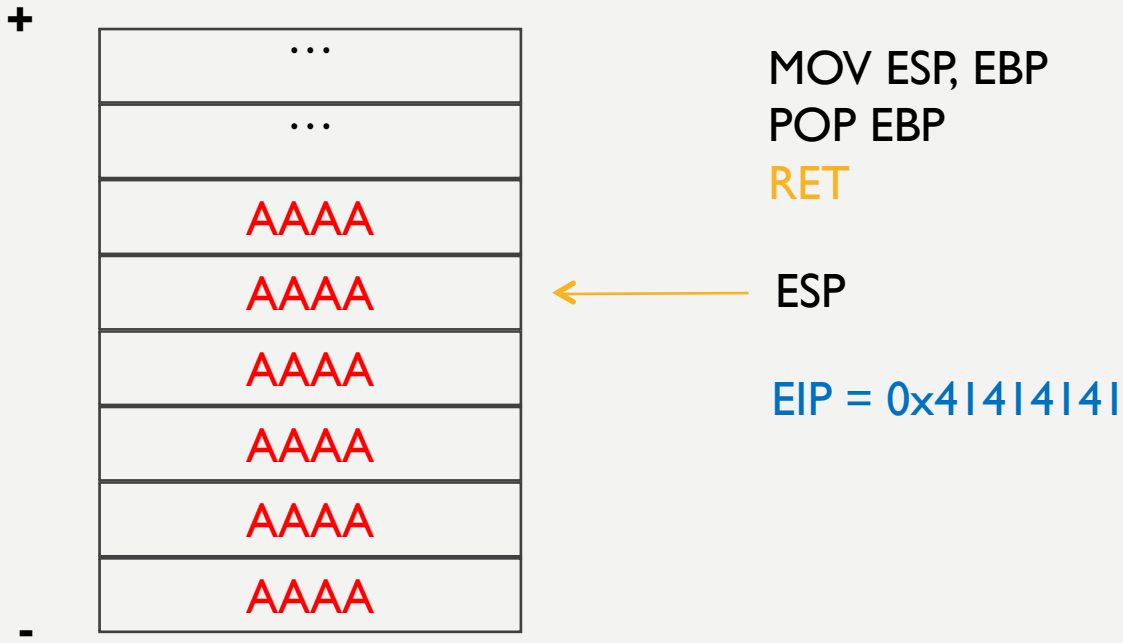
חריגה מחוצץ

- כעת מתבצעת החזרה מפונקציית main:



חריגה מחוצץ

- כעת מתבצעת החזרה מפונקציית main:



חריגה מחוצץ

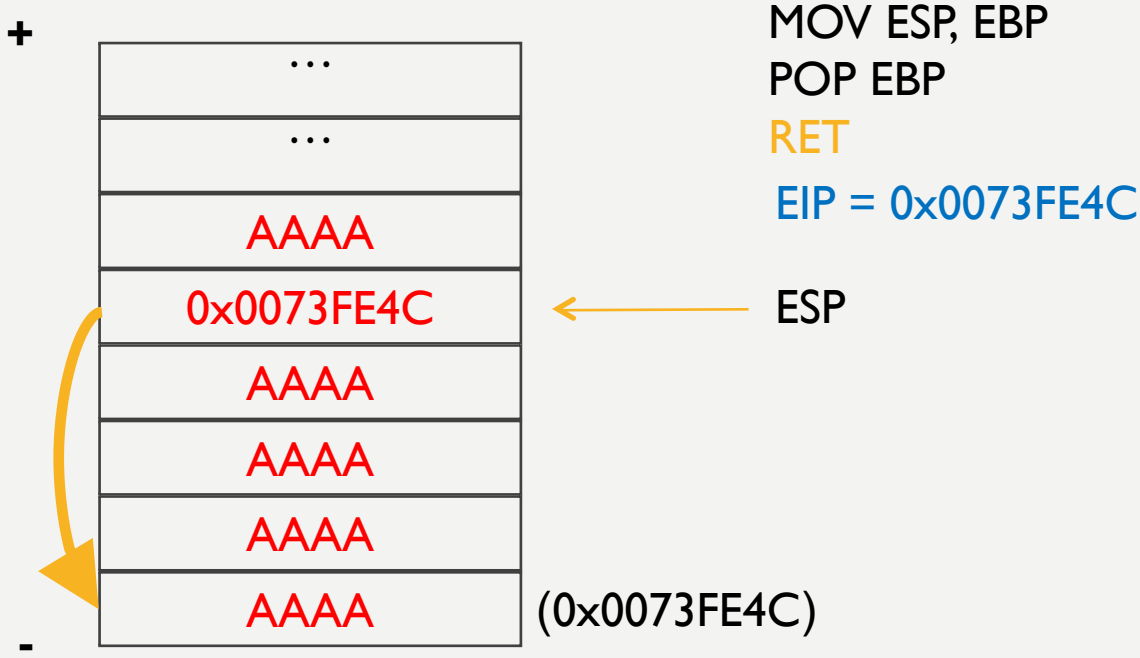
- נבצע את הדריסה כך ש EIP יצביע למשתנה שלנו במחסנית.
  - במשתנה שלנו נשים קוד אסמבלי שיכונה ShellCode.

- יש צורך למצוא את הכתובת המדויקת שבה המשתנה שלנו נמצא.
  - ניתן להריץ עם Debugger ולמצוא.

- בעיה: הכתובות של המחסנית (ושל המשתנה שלנו) אינן קבועות.
  - אך <u>נניח</u> שהמחסנית תתחיל תמיד (בערך) באותו מיקום בזכרון.
  - לכן ניתן לנחש בקירוב טוב את הכתובת של המשתנה buffer.

חריגה מחוצץ

- נותר רק להחליף את רצף ה ”AAAA” בפקודות אסמבלי כרצוננו.

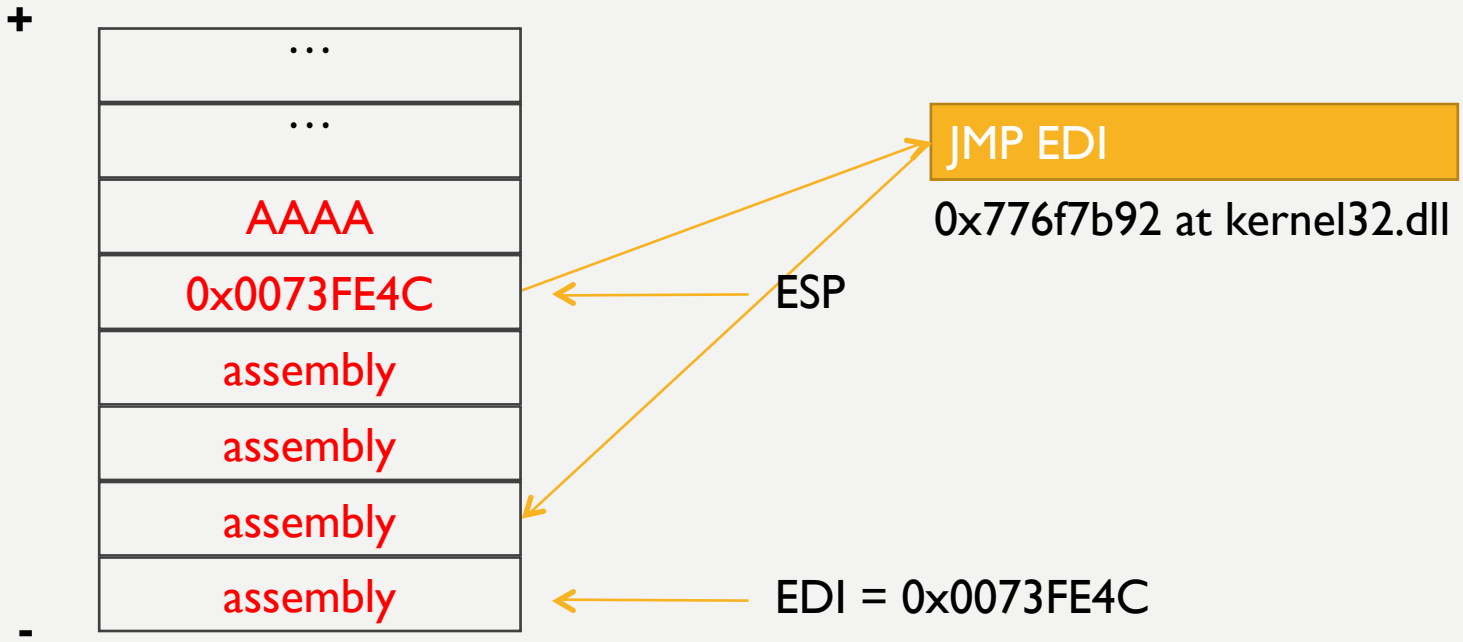


חריגה מחוצץ

- לפני שני שקפים נאמר שניתן לנחש בקירוב טוב את הכתובת של המשתנה buffer במחסנית.
  - אבל קירוב טוב לא מספיק. צריך לדעת במדויק לאיפה לקפוץ.
  - מספיק שנטעה בבית אחד – התוכנית תקרוס (למה?).
- מה הפתרון?

פתרון ראשון

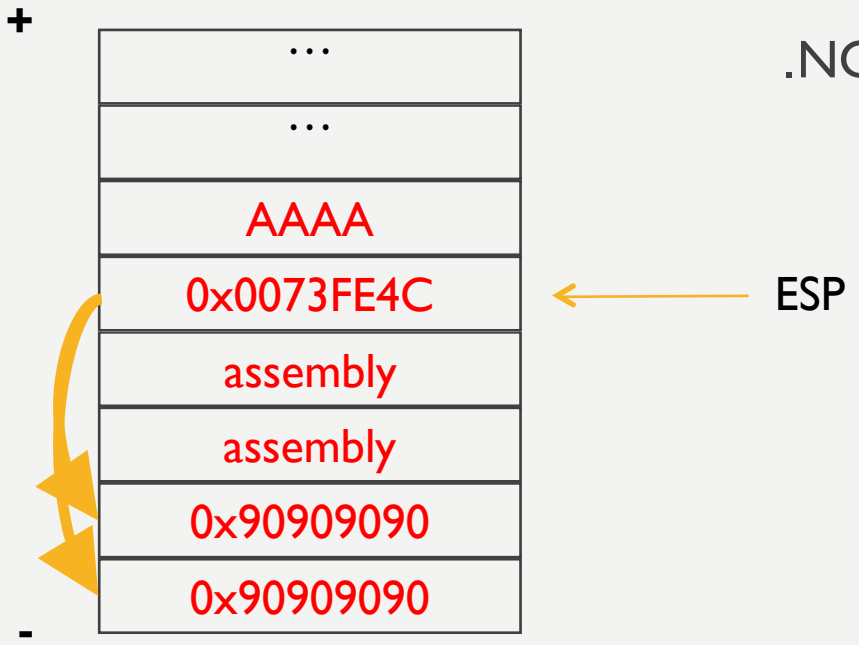
- ייתכן שאחד האוגרים מצביע ל-Shellcode שלנו (נניח EDI):



- כרגע אנחנו מניחים שאין ASLR ולכן אפשר להסתמך על כתובות קבועות.
- אך עדיין יש בעיה: הכתובת עלולה להשתנות בין גרסאות kernel32.dll.
- שאלה למחשבה: מה קורה אם אין "JMP EDI" ב-kernel32.dll ?

פתרון שני
הוספת NOPS

- ה-ShellCode שלנו יתחיל בפקודות NOPs.
  - אנחנו לא צריכים לקפוץ בדיוק לתחילת ה-ShellCode.
  - מספיק להגיע לאזור ה-NOPS.



- ככל שה-Buffer יותר גדול כך נגדיל את מספר ה nops.
  - ואת הסיכוי לקפוץ ל-ShellCode בהצלחה.

התגברות על הגנות כנגד חריגה מחוצץ

- בשקפים הקודמים הצגנו חולשה קלאסית של חריגה מחוצץ (Overflow Buffer).

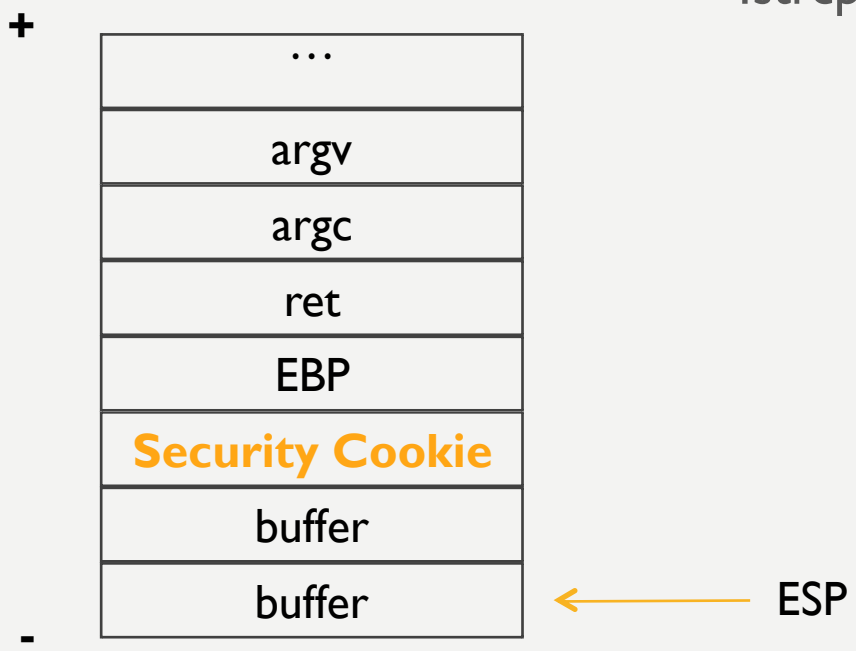
- לא לקחנו בחשבון מספר הגנות:
  - קנרית – Security Cookie
  - DEP
  - ASLR

- נדבר על הגנות אלו בהרחבה בשקפים הבאים.

קנריות

קנרית
SECURITY COOKIE

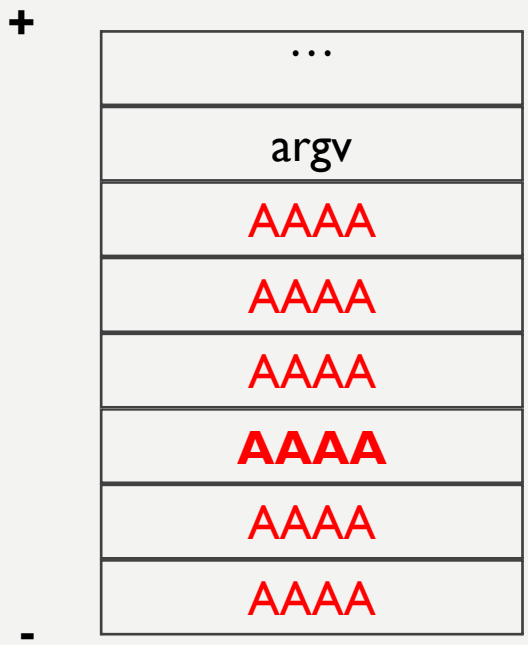
- מצב המחסנית לפני הקריאה ל-strcpy:



- לפני החזרה מ-main. ישנה קריאה לפונקציה שמוודא שערך ה- Security Cookie לא השתנה. אם הוא השתנה – סימן שהייתה דריסה.

קנרית
SECURITY COOKIE

- מצב המחסנית לפני הקריאה ל-strcpy:




```
MOV  ECX,DWORD PTR SS:[EBP-4]
XOR  ECX,EBP
CALL securCoo.__security_check_cookie
MOV  ESP,EBP
POP  EBP
RETN
```

Wrong Cookie

ESP

- לפני החזרה מ-main. ישנה קריאה לפונקציה שמוודא שערך ה- Security Cookie לא השתנה. אם הוא השתנה – סימן שהייתה דריסה.

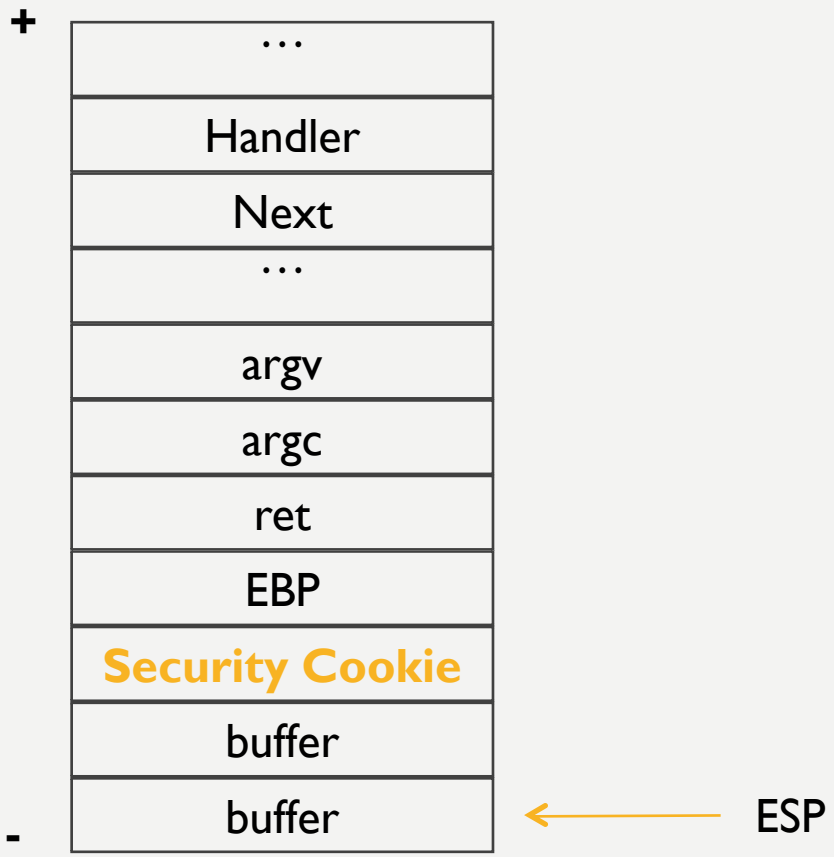
קנרית
SECURITY COOKIE

- איך מתגברים על קנרית?
  - יש מקרים שבהם ניתן לחזות מראש את ה-Cookie.
  - למשל אם מדובר בערך קבוע.
  - כמעט שלא קיים היום.

- נתחמק בדרכים אחרות.
  - נחפש כתובות אחרות במחסנית שניתן לדרוס.
  - איזה ערך (מצביע לפונקציה) במחסנית ניתן לדרוס, כך שהכתובת הזאת תיקרא לפני החזרה מהפונקציה?

קנרית
SECURITY COOKIE

- מצב המחסנית לפני הקריאה ל-strcpy:



קנרית
SECURITY COOKIE

- ניתן לעקוף Security Cookie דרך SEH.
  - זו לא הדרך היחידה.
  - יש דרכים נוספות – זה תלוי מה נמצא במחסנית בזמן הדריסה.

- נדרוס את ה-handler הקרוב.
- ננסה לגרום ל-Exception לפני החזרה מהפונקציה.
  - לפני הקריאה ל check_security_cookie, ננסה לגרום לחריגה בתכנית
  - במידה והצלחנו, נקפוץ לכתובת כרצוננו ללא בדיקת ה-Security Cookie.

- בתוכנית שהוצגה כאן כדוגמה, לא ניתן לעקוף את ההגנה דרך ה-SEH כי אין אפשרות לגרום לחריגה.

DEP

DATA EXECUTION PREVENTION

DEP
DATA EXECUTION PREVENTION

- נכנס ל-XP החל מ-SP2.
- ההגנה מונעת הרצת קוד באזורים בזיכרון שלא מוגדרים כאזורים עם הרשאת ריצה (EXECUTE).
- חוץ מה-Section שמכיל את הקוד עצמו, אין סיבה שאזורים אחרים יהיו בעלי הרשאות ריצה.
  - בפרט לא המחסנית.
  - איך בכל זאת נריץ את הקוד שלנו, שנמצא על המחסנית?

RETURN TO LIBC

- הקוד שהיחידי שנוכל לקפוץ אליו חייב להיות באזור עם הרשאות ריצה.
  - זה יכול להיות קפיצה ל-code section של עצמנו.
  - או קפיצה לכל מודול אחר שטעון במרחב שלנו.

- אפשר לנסות לקפוץ לפונקציה מסוימת.
  - למשל winExec

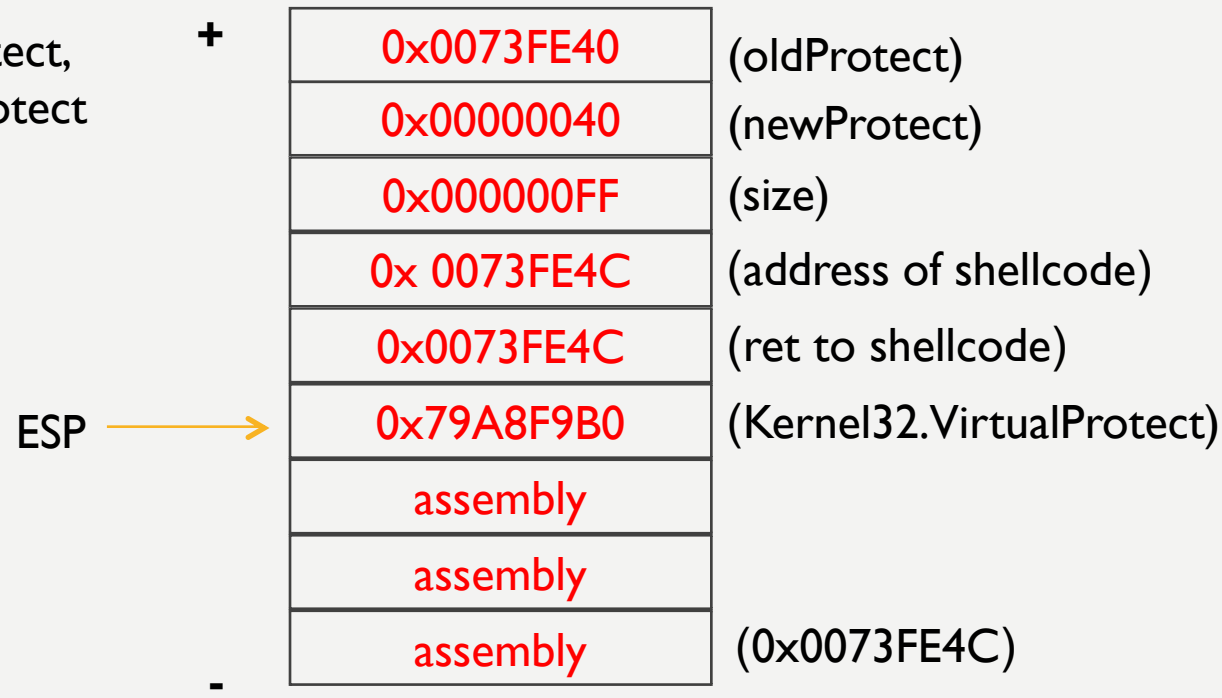
```
UINT WINAPI WinExec(LPCSTR lpCmdLine,UINT uCmdShow);
```

- נרצה בכל זאת להריץ ShellCode.
  - נקפוץ ל-VirtualProtect ולאחריו ל-ShellCode.

RETURN TO LIBC

- קריאה ל-VirtualProtect ולאחריה קפיצה ל-Shellcode:

```
BOOL VirtualProtect(
  LPVOID lpAddress,
  SIZE_T dwSize,
  DWORD  flNewProtect,
  PDWORD lpflOldProtect
);
```



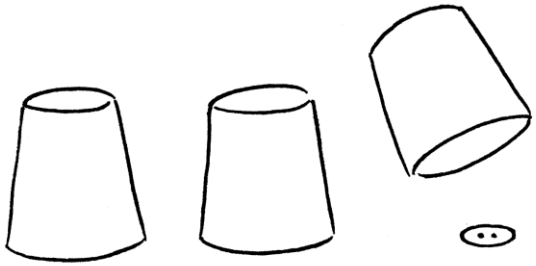
RETURN TO LIBC

- בשיטה כפי שהוצגה יש מספר בעיות:
  - הפרמטרים מכילים תווים אסורים, למשל NULL.
  - אנחנו לא יודעים במדויק את הכתובת של ה Shellcode.
  - וכמובן אנחנו לא יודעים בוודאות מה הכתובת של VirtualProtect.

- בשביל לפתור את הבעיות נצטרך להריץ קוד מינימלי.
  - קוד שיעזור לנו לסדר את הפרמטרים על המחסנית.
  - את הקוד נרכיב מפקודות שקיימות במודול מסוים בשיטה שנקראת ROP.
  - על שיטה זו נרחיב בהמשך.

ASLR

ADDRESS SPACE LAYOUT RANDOMIZATION



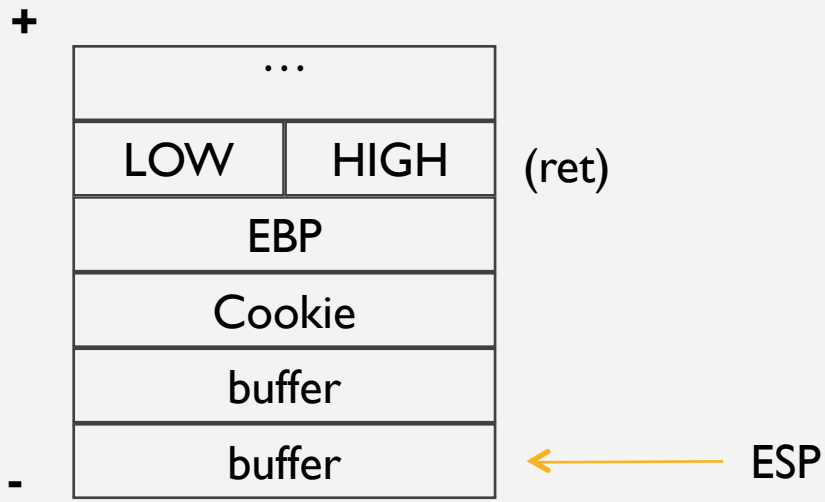
ASLR

ADDRESS SPACE LAYOUT RANDOMIZATION

- נכנס לתוקף החל מ-Vista ומעלה.
- כל המודולים נטענים לכתובות אקראיות.
  - גם המחסנית.
- השילוב של DEP & ASLR הוא שילוב חזק שצמצם משמעותית את כמות החולשות שניתן לנצלן.
- נציג כמה רעיונות להתמודד עם ההגנה.

ASLR

- מצב המחסנית לפני הקריאה ל-strcpy:



- ASLR מגריל רק את ה-HIGH bits של המודול.
  - כלומר המודול יטען ל- XXXX0000.
- לכן נדרוס רק את ה-Low bits.
  - אך אנחנו מוגבלים במספר הכתובות שניתן לקפוץ אליהם.

ASLR

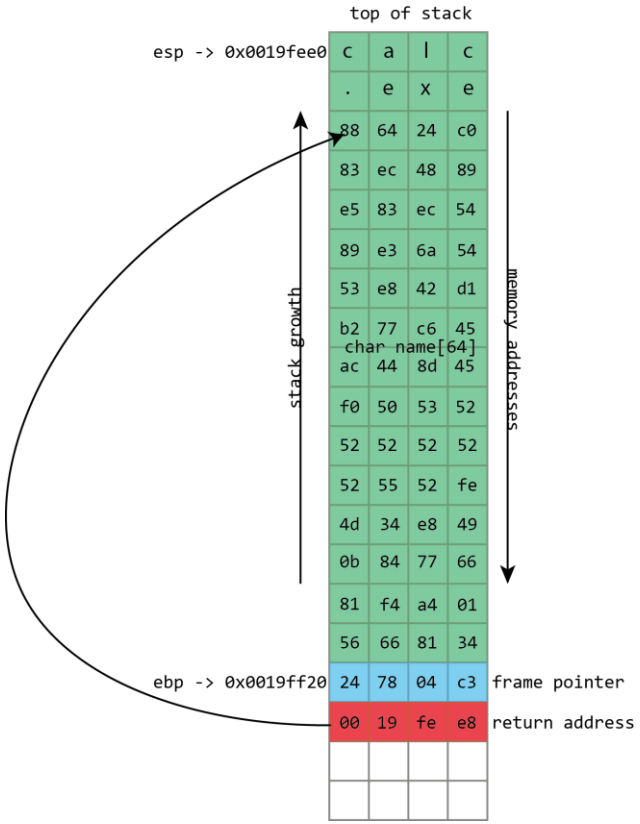
- דרכים נוספות להתמודדות עם ASLR:
- ההגרלה מתבצעת מתוך 256 כתובות אפשריות.
  - במקרים מסוימים ניתן להשתמש בזה לטובתנו (לנחש כתובת).
  - אחרי 256 ניסיונות בממוצע נצליח...
  - זה כבר לא נכון החל מ-Windows 8

- ישנם DLLs שלא עוברים ASLR
  - בפורמט ה-PE יש דגל שאומר האם הקובץ תומך ב-ASLR
  - מ"ה מתחשבת בערך הזה ופועלת בהתאם.
  - כיום קיימים מעט מאוד DLLs כאלו (בדרך כלל הם קיימים בתוכנות ישנות)

ASLR

- **מציאת חולשת קריאה.**
  - חולשה שתגרום להדלפת מידע (עם כתובות) מהזיכרון.
  - שילוב של חולשת קריאה עם חולשת הרצה תאפשר עקיפת ASLR.
  - הרבה עבודה.
  - זו הדרך המקובלת לעקוף את הגנת ASLR.

SHELLCODE


top of stack
esp -> 0x0019fee0
c a l c
. e x e
88 64 24 c0
83 ec 48 89
e5 83 ec 54
89 e3 6a 54
53 e8 42 d1
b2 77 c6 45
char name[64]
ac 44 8d 45
f0 50 53 52
52 52 52 52
52 55 52 fe
4d 34 e8 49
0b 84 77 66
81 f4 a4 01
56 66 81 34
ebp -> 0x0019ff20
24 78 04 c3
frame pointer
00 19 fe e8
return address
stack growth
memory addresses

SHELLCODE

- הקוד הראשון שרץ בתהליך ניצול החולשה נקרא ShellCode.
- בדרך כלל תפקיד ה-ShellCode הוא לאפשר הרצה של קוד אחר בצורה נוחה.

- לדוגמא:
  - ShellCode שמוריד EXE מהאינטרנט ומריץ אותו.

    (רלוונטי במקרה של נוזקה שרוצה להתפשט דרך חולשה)
  - ניתן גם לכתוב Shellcode שמטרתו הוא להוכיח שהחולשה ניתנת לניצול (שהחולשה מאפשרת הרצת קוד). למשל קוד שמריץ מחשבון (calc.exe).

SHELLCODE

- יצירת סביבה לכתיבת Shellcode:

```
char * shellcode = "\x90\x90\xC3";

int (*shellcode_func)();

int main() {

    DWORD junk;
    VirtualProtect(shellcode,256,PAGE_EXECUTE_READWRITE,&junk);
    shellcode_func = (int (*)()) shellcode;
   (*shellcode_func)();

}
```

0x90	Nop
0x90	Nop
0xC3	ret

SHELLCODE

- נרצה לכתוב Shellcode שמריץ MessageBox.
- לצורך כך נכתוב את הקוד בסי ונמיר לאסמבלי.
  - ניתן כמובן לכתוב ישירות באסמבלי.
- דוגמא לקוד:

```
int msgBox() {
    MessageBoxA(0,"Test","Test",0);
}
```

- ובאסמבלי:

```
6A 00          PUSH 0
68 00214000    PUSH OFFSET shellcod.??_C@_04OFFCAMCD@Te  ASCII "Test"
68 00214000    PUSH OFFSET shellcod.??_C@_04OFFCAMCD@Te  ASCII "Test"
6A 00          PUSH 0
FF15 94204000  CALL DWORD PTR DS:[<&USER32.MessageBoxA]  USER32.MessageBoxA
33C0           XOR EAX,EAX
C3             RETN
```

SHELLCODE

- דוגמא להרצת MessageBox פעמיים:

```
char * shellcode = "\x6A\x00\x68\x18\x21\x40\x00\x68\
                    \x18\x21\x40\x00\x6A\x00\
                    \xFF\x15\x98\x20\x40\x00\x33\xC0\xC3";
int (*shellcode_func)();

void msgBox() {
    MessageBoxA(0,"Test","Test",0);
}
int main() {
    msgBox();
    // run messageBox from the shellcode
    DWORD junk;
    VirtualProtect(shellcode,256,PAGE_EXECUTE_READWRITE,&junk);
    shellcode_func = (int (*)()) shellcode;
   (*shellcode_func)();
}
```

- אחרי דיבוג קצר וכמה תיקונים הקוד רץ.
  - למשל אם מוסיפים משתנה הקומפיילר יכול לשנות את המיקום של "Test".

SHELLCODE

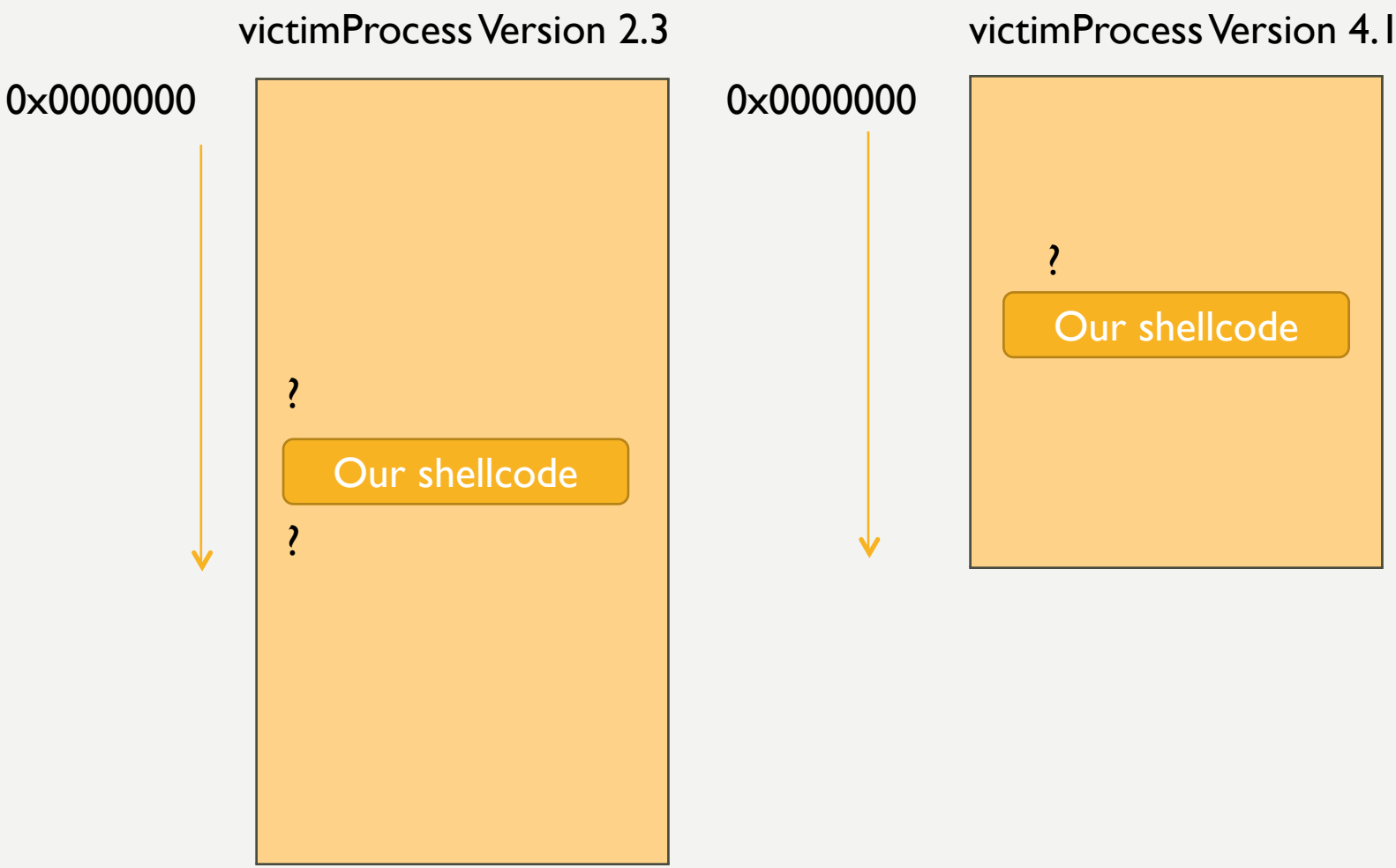
- ה-Shellcode שיצרנו אכן עבד.
- אם נשלב אותו עם חולשה מתאימה,

האם הוא יגרום להרצת MessageBox במחשב מרוחק?

```
6A 00            PUSH 0
68 00214000      PUSH OFFSET shellcod.??_C@_04OFFCAMCD@Te  ASCII "Test"
68 00214000      PUSH OFFSET shellcod.??_C@_04OFFCAMCD@Te  ASCII "Test"
6A 00            PUSH 0
FF15 94204000    CALL DWORD PTR DS:[<&USER32.MessageBoxA]  USER32.MessageBoxA
33C0             XOR EAX,EAX
C3               RETN
```

SHELLCODE

- ה-Shellcode רץ בתהליך מרוחק שלא ניתן להניח עליו דבר:



מה לא תקין במה שכתבנו?

- הפנייה לכתובות מסוימות (כמו ל "Test").
  - ShellCode רץ בתהליך מרוחק כלשהו, לא ניתן להסתמך על שום כתובת וכמובן לא ניתן להסתמך על כך שיש מחרוזת "Test".
- קריאה ל-MessageBox דרך ה-IAT.
  - בתהליך המרוחק אין הבטחה לגבי הערכים ב-IAT.
  - בהנחה שאין ASLR, האם כדאי להחליף את הקריאה, לקריאה ישירה?
- לא בטוח שניתן להשתמש בתווים מסוימים.
  - למשל אם החולשה היא דרך strcpy, אז ה ShellCode לא יכול להכיל NULL. מדוע?
- ה ShellCode חייב לעבוד בכל סביבה ולכן הוא לא יכול להסתמך על סביבה ספציפית, בפרט לא על כתובות קבועות.

שימוש במחרוזות/משתנים ב SHELLCODE

- דוגמא לשימוש במחרוזת "Test" בתוך הקוד(ללא שימוש בכתובות:)

```
shellcode:
        jmp getTest
L1:
        mov ecx,[esp] // ecx point to "test"
        push 0x0
        push ecx
        push ecx
        push 0x0
        call MessageBox
getTest:
        call L1
        DB "test"
```

- דרך נוספת היא על ידי שימוש במחסנית כמו שעשיתם בתרגיל בית 1

מציאת כתובת של פונקציה

- נזכור ש FS מכיל את ה TEB.
- אחת השדות של ה TEB, הוא ה PEB:

FS:[0x30]	4	NT	Linear address of Process Environment Block (PEB)

- אחת השדות של ה-PEB, הוא PPEB_LDR_DATA:

```
typedef struct _PEB
{
    BOOLEAN InheritedAddressSpace;
    BOOLEAN ReadImageFileExecOptions;
    BOOLEAN BeingDebugged;
      ...
    PVOID ImageBaseAddress;
    PPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    PVOID SubSystemData;
      ...
```

(Wikipedia.org)

מציאת כתובת של פונקציה

- PPEB_LDR_DATA:

מכיל רשימות מקושרות שמתארות את המודולים הטעונים.

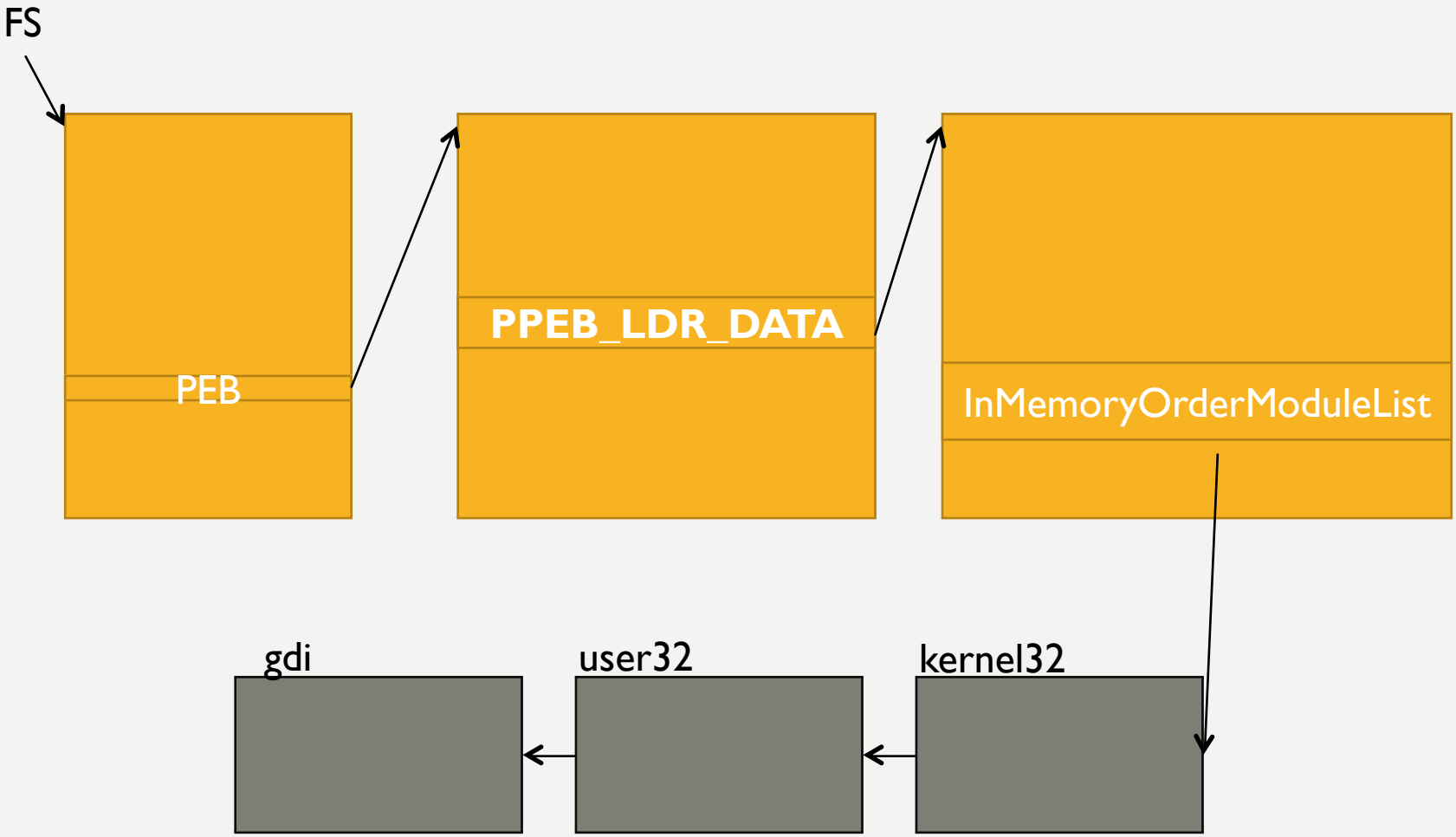
```
typedef struct _PEB_LDR_DATA {
  ULONG                   Length;
  BOOLEAN                 Initialized;
  PVOID                   SsHandle;
  LIST_ENTRY              InLoadOrderModuleList;
  LIST_ENTRY              InMemoryOrderModuleList;
  LIST_ENTRY              InInitializationOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;
```

- כל איבר ברשימה מכיל (בין היתר) את השדות הבאים:

```
  PVOID                   BaseAddress;
  PVOID                   EntryPoint;
  ULONG                   SizeOfImage;
  UNICODE_STRING          FullDllName;
  UNICODE_STRING          BaseDllName;
      ...
```

(ntinternals.net)

מציאת כתובת של פונקציה



(Wikipedia.org)

מציאת כתובת של פונקציה

- ראינו כיצד ניתן למצוא את הכתובת של מודול בצורה גנרית.
  - ללא שימוש בכתובות.
- על מנת למצוא כתובת של פונקציה ניתן לסרוק את ה-Export Table של המודול.
  - גם ללא שימוש בכתובות.
- ShellCode גנרי צריך לפעול בדרך זו.
- ישנם דרכים נוספות (פחות גנריות):
  - למשל לעיתים יש במחסנית ערכים שיוכלו לעזור לנו למצוא כתובת של מודול.

סיכום SHELLCODE

- ל-ShellCode גנרי יש את היתרון שהוא פועל בכל מ"ה.
- זה בא על חשבון הגודל שלו. מדובר בלא מעט פקודות אסמבלי.

- אם ה-ShellCode מיועד למערכת מסוימת או שהוא מיועד לפעול בתוכנה ספציפית אז אפשר לוותר על חלק מהגנריות.
- התמודדות עם תווים אסורים מוסיפה סיבוך:
  - נצטרך להשתמש בפקודות אחרות שלא מכילות תווים כאלו.
  - נקודד מחרוזות/פרמטרים לפונקציה.
  - ניתן לקודד את כל ה-ShellCode (מין סוג של packer).
  - יש המון כלים שעושים זאת אוטומטית. זה בא על חשבון הגודל.

סיכום חולשות זכרון

- הצגנו חולשת Stack Overflow.
  - כולל תיאור מרבית ההגנות שקיימות כיום.

- יש עוד סוגים של חולשות זכרון:
  - Format String
  - Double Free
  - ועוד...

- **מומלץ** לעיין בספר:
  - The Shellcoders Handbook