פרק 4

חולשות ניכרון

Buffer Overflow

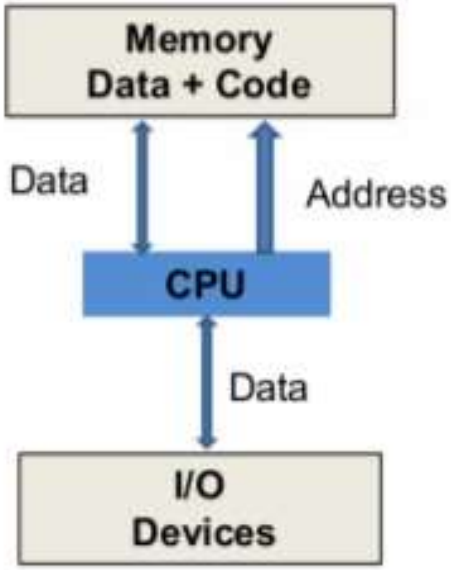
מה תוקף רודה?

להריץ קוד על המחשב של הנתקף

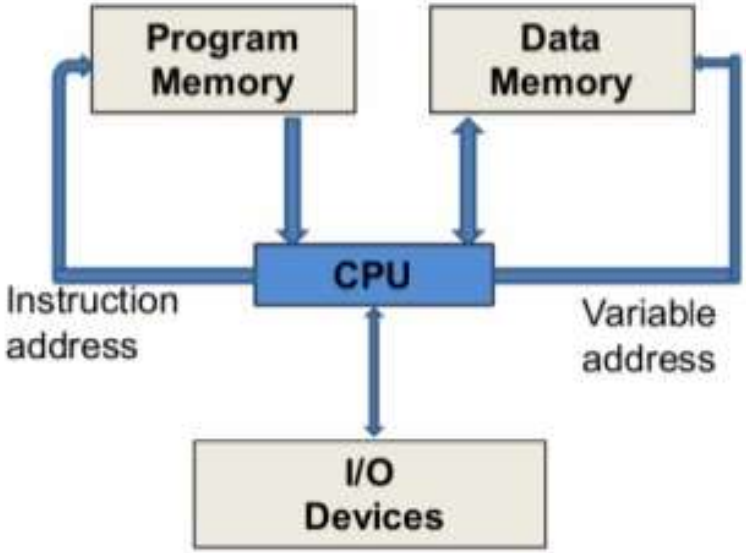
- איך אפשר לעשות את זה?
  - Social Engineering
  - "תקיפת המחשב" באמצעות חולשה
- מה זו חולשה?
  - באג לוגי / תכנוני / מימושי שניתן לנצל אותו

ארכיטקטורת המחשב

- Von Neumann



- Harvard

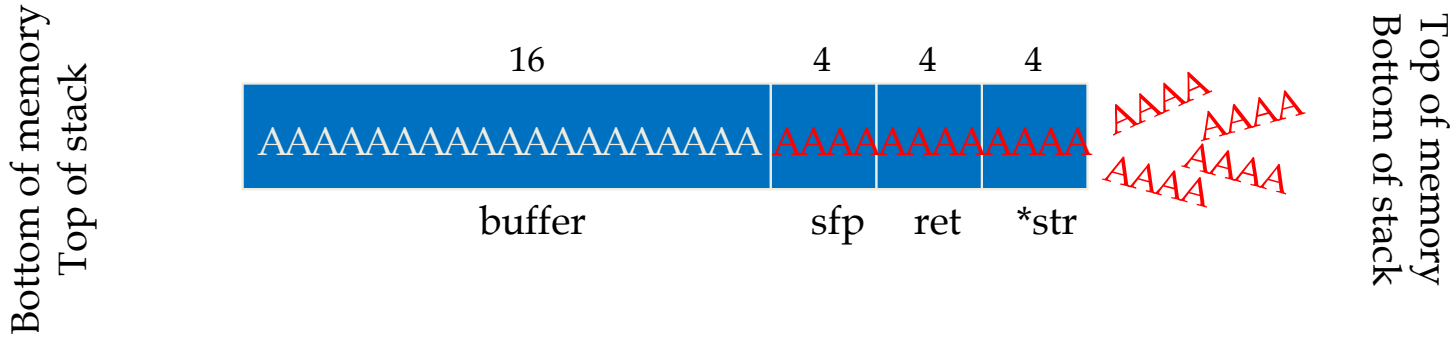


מה הבעיה?

```
4
5  void function(char *str){
6      char buffer[16];
7      strcpy(buffer, str);
8  }
9
10
11
12
13 int main(){
14     char large_string[256];
15     int i;
16
17     for (i = 0; i < 255; i++){
18         large_string[i] = 'A';
19     }
20
21     function(large_string);
22 }
```

Buffer Overflaw

- כאשר התוכנית רצה נגרמת Seg-Fault

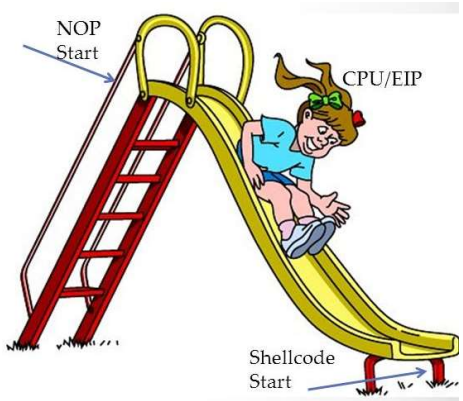


- כתובת החזרה נדרסת על ידי 'AAAA' (0x41414141)
- כשהפונקציה מסיימת היא מנסה לחזור לכתובת 0x41414141

אז איך מנצלים את זה?

- דורסים את כתובת החזרה
  - לאן להצביע?

- כותבים קטע קוד שאינו תלוי במיקומו (Shellcode)
  - ושמים אותו על הstack
  - איך נתכונן לעובדה שאנחנו לא יודעים תמיד בדיוק מה יש על הStack? ומה המרחק הנכון?
    - NOOOOOOOOOOOOOOOOOOOOP



- הקוד מורץ!

File: archives/49/p49_0x0e_Smashing The Stack For Fun And Profit_by_Aleph1.txt
.oO Phrack 49 Oo. 1996

Volume Seven, Issue Forty-Nine

File 14 of 16

BugTraq, r00t, and Underground.org
bring you

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Smashing The Stack For Fun And Profit
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

by Aleph One
aleph1@underground.org

`smash the stack` [C programming] n. On many C implementations it is possible to corrupt the execution stack by writing past the end of an array declared auto in a routine. Code that does this is said to smash the stack, and can cause return from the routine to jump to a random address. This can produce some of the most insidious data-dependent bugs known to mankind. Variants include trash the stack, scribble the stack, mangle the stack; the term mung the stack is not used, as this is never done intentionally. See spam; see also alias bug, fandango on core, memory leak, precedence lossage, overrun screw.

Introduction

Over the last few months there has been a large increase of buffer

איך ניתן למנוע את זה?

- קידוד מאובטח...
  - שימוש בספריות מאובטחות
  - שימוש בשפות שהן Type-Safe
  - Static Analysis

Dangerous C system calls

source: Building secure software, J. Viega & G. McGraw, 2002

Extreme risk

- gets

High risk

- strcpy
- strcat
- sprintf
- scanf
- sscanf
- fscanf
- vfscanf
- vsscanf

High risk (cntd)

- streadd
- strecpy
- strtrns
- realpath
- syslog
- getenv
- getopt
- getopt_long
- getpass

Moderate risk

- getchar
- fgetc
- getc
- read
- bcopy

Low risk

- fgets
- memcpy
- snprintf
- strccpy
- strcadd
- strncpy
- strncat
- vsnprintf

strncmp(computed_response, user_response, response_length)

```
NETSTACK_CODE:20431F80 loc_20431F80:                          # CODE XREF: NETSTACK_AuthDigestParseResponse+FE↑j
NETSTACK_CODE:20431F80         add     r14, sp, 0x10C+var_F4
NETSTACK_CODE:20431F82         mov     r0, r13
NETSTACK_CODE:20431F84         add     r0, r0, 0x55
NETSTACK_CODE:20431F86         mov     r1, r14
NETSTACK_CODE:20431F88         bl      NETSTACK_CODE_2043218C
NETSTACK_CODE:20431F8C         ld      r4, [sp,0x10C+nc.value_len]
NETSTACK_CODE:20431F90         ld      r5, [sp,0x10C+var_74]
NETSTACK_CODE:20431F94         ld      r0, [sp,0x10C+qop.value_len]
NETSTACK_CODE:20431F98         ld      r7, [sp,0x10C+qop]
NETSTACK_CODE:20431F9C         st      r0, [sp,0x10C+var_10C]
NETSTACK_CODE:20431F9E         ld      r0, [sp,0x10C+uri]
NETSTACK_CODE:20431FA2         st      a1, [sp,0x10C+var_108]
NETSTACK_CODE:20431FA6         st      r0, [sp,0x10C+var_104]
NETSTACK_CODE:20431FA8         ld      r0, [sp,0x10C+uri.value_len]
NETSTACK_CODE:20431FAC         ld      r6, [sp,0x10C+var_70]
NETSTACK_CODE:20431FB0         add     r13, sp, 0x10C+var_D0
NETSTACK_CODE:20431FB2         st      r0, [sp,0x10C+var_100]
NETSTACK_CODE:20431FB4         st      req_text, [sp,0x10C+var_FC]
NETSTACK_CODE:20431FB8         st      r13, [sp,0x10C+var_F8]
NETSTACK_CODE:20431FBA         ld      r1, [sp,0x10C+nonce]
NETSTACK_CODE:20431FBC         mov     r0, r14
NETSTACK_CODE:20431FBE         ld      r2, [sp,0x10C+nonce.value_len]
NETSTACK_CODE:20431FC0         ld      r3, [sp,0x10C+nc]
NETSTACK_CODE:20431FC4         bl      NETSTACK_CODE_204321D0
NETSTACK_CODE:20431FC8         ld      r1, [sp,0x10C+user_response]
NETSTACK_CODE:20431FCC         mov     r0, r13                 # computed_response
NETSTACK_CODE:20431FCE         ld      r2, [sp,0xA4]           # response_length
NETSTACK_CODE:20431FD2         bl      strncmp
NETSTACK_CODE:20431FD6         cmp     r0, 0
NETSTACK_CODE:20431FD8         bne     error
```

Silent Bob

- An authentication bypass vulnerability, which will be later known as CVE-2017-5689, was originally discovered in mid-February of 2017

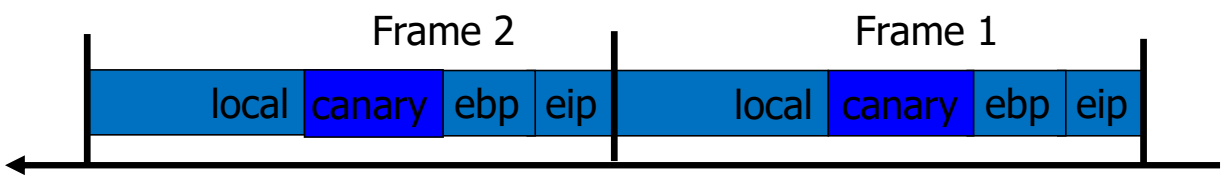
- It seems quite obvious that the third argument of strncmp() should be the length of computed_response , but the address of the stack variable response_length, from where the length is to be loaded, actually points to the length of the **user_response!**

איך ניתן למנוע את זה?

- קידוד מאובטח...
  - שימוש בספריות מאובטחות
  - שימוש בשפות שהן Type-Safe
  - Static Analysis
- בדיקות בזמן ריצה
  - Canaries \ Cookies
- Non Executable Stack
  - ATL Thunk: Legacy!

Stack Cookies /GS

- בדיקת "זמן ריצה".
- בזמן הידור מוכנס "משתנה" נוסף בין המשתנים המקומיים וכתובת החזרה.
  - איזה ערכים כדאי שיהיו למשתנה הזה? מה לא?
- מה המאפיינים שצריכים להיות למשתנה?
  - מה יכול תוקף ללמוד?
    - מריצה אחרת?
    - מראיית הזכרון?
    - מהכרת תהליכים אחרים?
- איפה עוד אפשר לשים קנריות?



How is it done?

```
; prologue

push    ebp
mov     ebp, esp
sub     esp, 214h
mov     eax, ___security_cookie ; random value, initialized at module startup
xor     eax, ebp                ; XOR it with the current base pointer
mov     [ebp+var_4], eax        ; store the cookie

...

; epilogue

mov     ecx, [ebp+var_4]        ; get the cookie from the stack
xor     ecx, ebp                ; XOR the cookie with the current base pointer
call    __security_check_cookie ; check the cookie
leave
retn    0Ch

; __fastcall __security_check_cookie(x)

cmp     ecx, ___security_cookie
jnz     ___report_gsfailure     ; terminate the process
rep retn
```

#pragme strict_gs_check

- Extra prologe & epiloge – significant overhead.
- So with /GS adds stack cookie only to functions that contain string buffers of allocate memory on the stack.
- A compiler directive (VS 20005 SP1+) that enables more aggressive GS heuristics
- Adds a cookie to all functions that use address of local variable.
- Result – more protection vs. runtime performance.

מה קורה כשהבדיקה נכשלת?

טיפול בחריגות

Exception Handling

מבוא לטיפול בחריגות

- טיפול בחריגות הינו דבר נפוץ שנמצא בשימוש כמעט בכל תוכנית.
- כחוקרים אנו ניתקל רבות בטיפול בחריגות, ועלינו לזהות ולהבין כיצד זה עובד.

- נתחיל משירותי מ״ה לחריגות ברמת התהליך.
- ונמשיך עם איך השירותים הללו ניתנים ע״י גרעין מ״ה.

טיפול בחריגות ב-C וב-++C

C

```
__try
{
        // guarded code
}
__except ( expression )
{
        // exception handler code
}
```

C++

```
try {

        //throw
}
catch (…)
{
        // exception handler code
}
```

שירותי מ"ה לטיפול בחריגות

Structured Exception Handling (SEH)

- מערכת ההפעלה מאפשרת טיפול בחריגות ע"י מנגנון בסיסי המשותף לכל המהדרים והתהליכים.

- כל קומפיילר מממש Exception-Handling בצורה שונה.
  - לא ניכנס לפרטים לגבי המימושים השונים של הקומפיילרים.
  - קומפיילרים מרחיבים את המנגנון של מערכת ההפעלה.

מבוא ל-SEH

- תוכנית יכולה לקרוס מסיבות שונות (חלוקה ב-0 למשל).
- מערכת ההפעלה מיידעת את התוכנית על הקריסה, ואף נותנת לה הזדמנות לתקן את הבעיה.
  - מ"ה קוראת לפונקציה ששייכת לתוכנית שקרסה.
  - הפונקציה תעשה מה שתעשה, ותחזיר למ"ה ערך חזרה.
  - בתחילת ריצת התכנית או קטע ה-try..except התוכנית כמובן צריכה ליידע את מ"ה שיש לה פונקציה כזאת ומה הכתובת שלה.
- לפונקציה קוראים except_handler.

except_handler הפונקציה

- הפונקציה שנקראת בזמן חריגה.
  - בדרך כלל מדובר בפונקציה שהקומפיילר מספק.
  - אבל גם התכנית יכולה לספק אחת כזו.
- קלט:
  - מקבלת מידע לגבי סוג החריגה ומצב התהליך בזמן החריגה.
- ערכי החזרה:
  - ExceptionContinueExecution:
    - מסמל טיפול מוצלח בחריגה.
  - ExceptionContinueSearch:
    - אם הפונקציה לא יודעת כיצד לטפל בחריגה.
  - יש עוד שני ערכי חזרה אפשריים עליהם לא נרחיב:
    - ExceptionNestedException.
    - ExceptionCollidedUnwind.

except_handler הפונקציה

```
EXCEPTION_DISPOSITION
        except_handler(
                struct _EXCEPTION_RECORD *ExceptionRecord,
                void * EstablisherFrame,
                struct _CONTEXT *ContextRecord,
                 void * DispatcherContext
  );
```

EXCEPTION_RECORD

```
typedef struct _EXCEPTION_RECORD {
        DWORD ExceptionCode;
        DWORD ExceptionFlags;
        struct _EXCEPTION_RECORD *ExceptionRecord;
        PVOID ExceptionAddress;
        DWORD NumberParameters;
        DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
}  EXCEPTION_RECORD;
```

דוגמא לכמה ערכי ExceptionCode (מתוך אלפים אפשריים):

0xC0000093
STATUS_FLOAT_UNDERFLOW
0xC0000094
STATUS_INTEGER_DIVIDE_BY_ZERO
0xC0000095
STATUS_INTEGER_OVERFLOW

ניתן למצוא רשימה מלאה ב-MSDN תחת ***NTSTATUS values***.

CONTEXT

```
typedef struct _CONTEXT
{
    ULONG ContextFlags;
    ULONG Dr0;
    ULONG Dr1;
    ULONG Dr2;
    ULONG Dr3;
    ULONG Dr6;
    ULONG Dr7;
    FLOATING_SAVE_AREA FloatSave;
    ULONG SegGs;
    ULONG SegFs;
    ULONG SegEs;
    ULONG SegDs;
    ULONG Edi;
    ULONG Esi;
    ULONG Ebx;
    ULONG Edx;
    ULONG Ecx;
    ULONG Eax;
    ULONG Ebp;
    ULONG Eip;
    ULONG SegCs;
    ULONG EFlags;
    ULONG Esp;
    ULONG SegSs;
    UCHAR ExtendedRegisters[512];
} CONTEXT, *PCONTEXT;
```

המבנה CONTEXT משמש גם כקלט וגם כפלט: בתחילה הוא מכיל את מצב האוגרים בעת החריגה ובהמשך הפונקציה except_handler יכולה לשנות אותו.
אם הפונקציה except_handler החזירה **ExceptionContinueExecution**, אז מ״ה תעדכן את האוגרים בהתאם למבנה CONTEXT.

except_handler – דוגמה

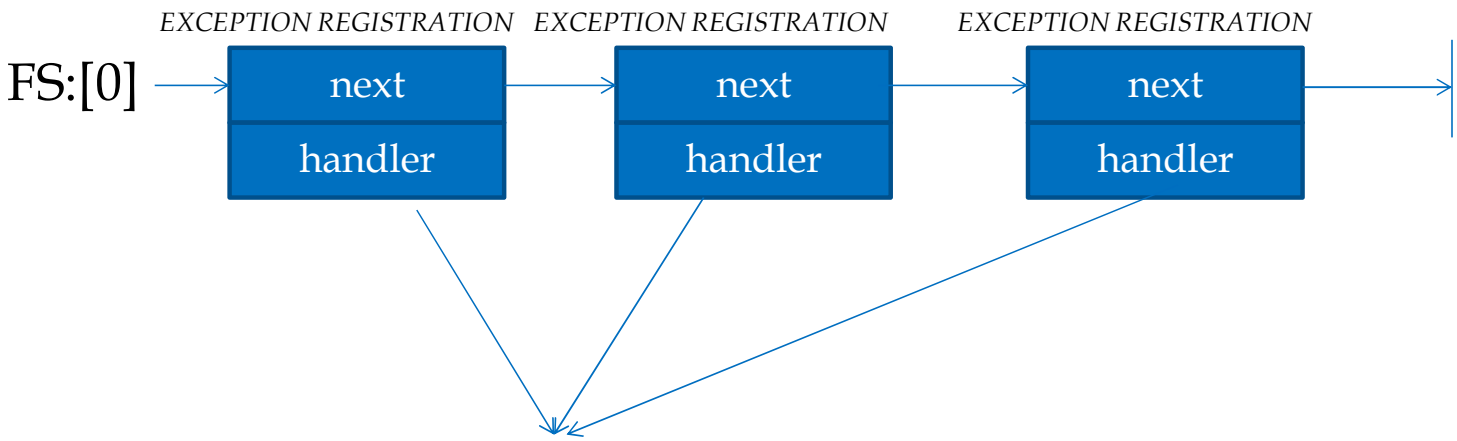
```
exit_label:
        ExitProcess();


_except_handler(struct _EXCEPTION_RECORD *ExceptionRecord,
                void * EstablisherFrame,
                 struct _CONTEXT *ContextRecord,
                void * DispatcherContext )
{
        ContextRecord->Eip = exit_label;
        return ExceptionContinueExecution;
}
```

EXCEPTION REGISTRATION

- ראינו כיצד נראית הפונקציה שמ״ה תקרא לה.
- אבל איך ניידע את מ״ה שיש לנו פונקציה כזאת?
  - איך מ״ה תמצא אותה בזיכרון התוכנית?

EXCEPTION REGISTRATION




```
except_handler(
        struct _EXCEPTION_RECORD *ExceptionRecord,
        void *     EstablisherFrame,
        struct _CONTEXT *ContextRecord,
         void * DispatcherContext );
```

Thread Environment Block

- מבנה שמכיל מידע על Thread מסוים.
  - לכל Thread יש אחד כזה.
- מגיעים אליו דרך FS.

```
typedef struct _TEB
{
   struct _NT_TIB
   {
      PEXCEPTION_REGISTRATION_RECORD ExceptionList;
      PVOID StackBase;
      PVOID StackLimit;
      PVOID SubSystemTib;
      …
   }

   PVOID EnvironmentPointer;
   CLIENT_ID ClientId;
            …
            …
```

EXCEPTION REGISTRATION

- המידע שמור במבנה הבא :

```
struct EXCEPTION_REGISTRATION {
        EXCEPTION_REGISTRATION * next;
        void *handler;
}
```

  - Handler – מצביע לפונקציה.
  - Next – מצביע ל-EXCEPTION REGISTRATION
    - יש מאמרים שמציגים את המבנה עם שדה prev, המשמעות זהה.
    - למעשה מדובר ברשימה מקושרת.
- **מקובל לשים את אברי הרשומות על המחסנית.**

הנדסה

```
push except_handler
push fs:[0]
mov fs:[0],esp
```

Set the handler

```
... some code
```

The code in the “try” block

```
pop fs:[0]
add esp,4
```

Restore (delete) the exception record

```
except_handler:
…some code…
mov eax , ?
ret
```

0 – ExceptionContinueExecution
1 - ExceptionContinueSearch

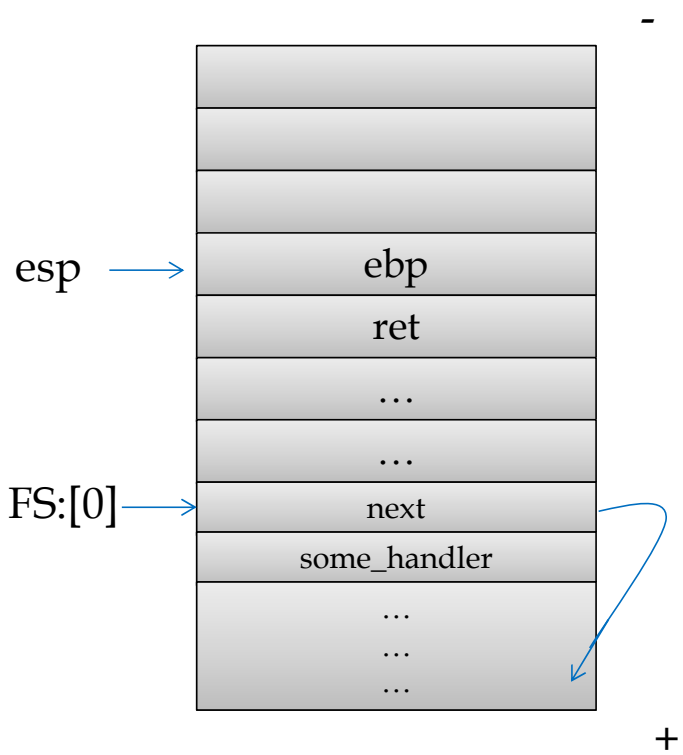
הגנה

```
push except_handler
push fs:[0]
mov fs:[0],esp

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



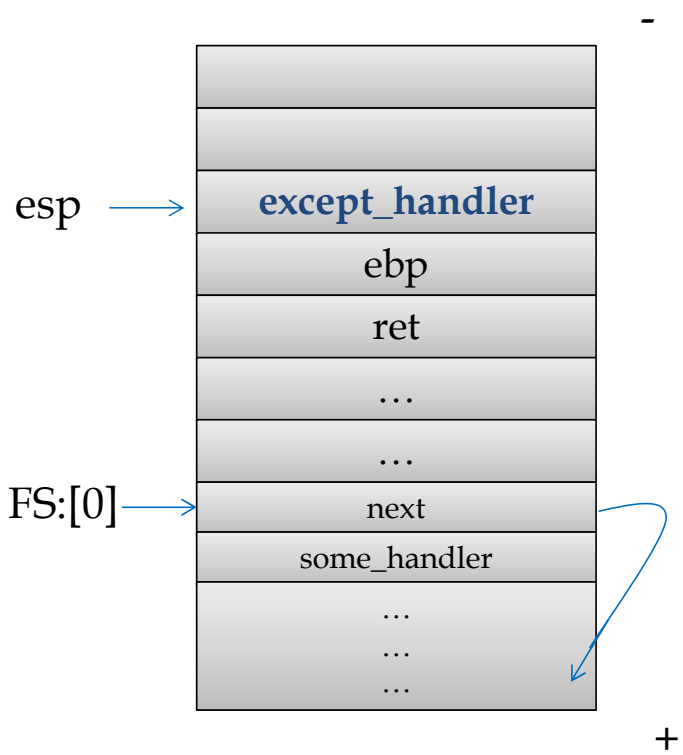
KNCIP

```
push except_handler
push fs:[0]
mov fs:[0],esp

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



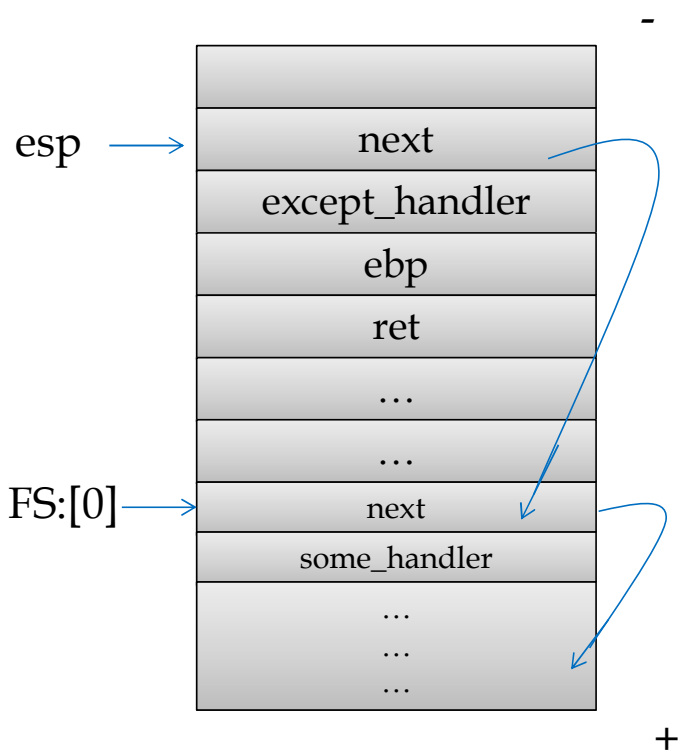
דוגמא

```
push except_handler
push fs:[0]
mov fs:[0],esp

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



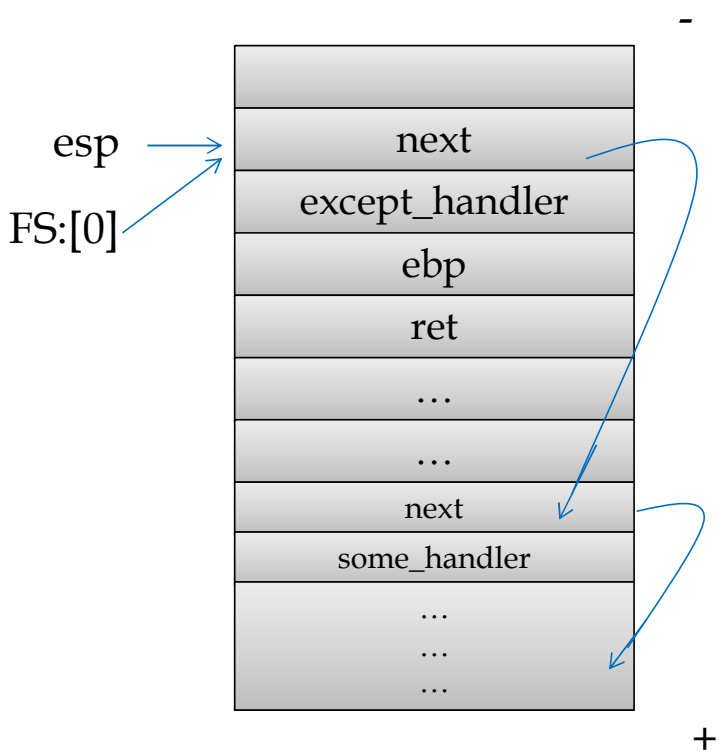
KNCIP

```
push except_handler
push fs:[0]
mov fs:[0],esp

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



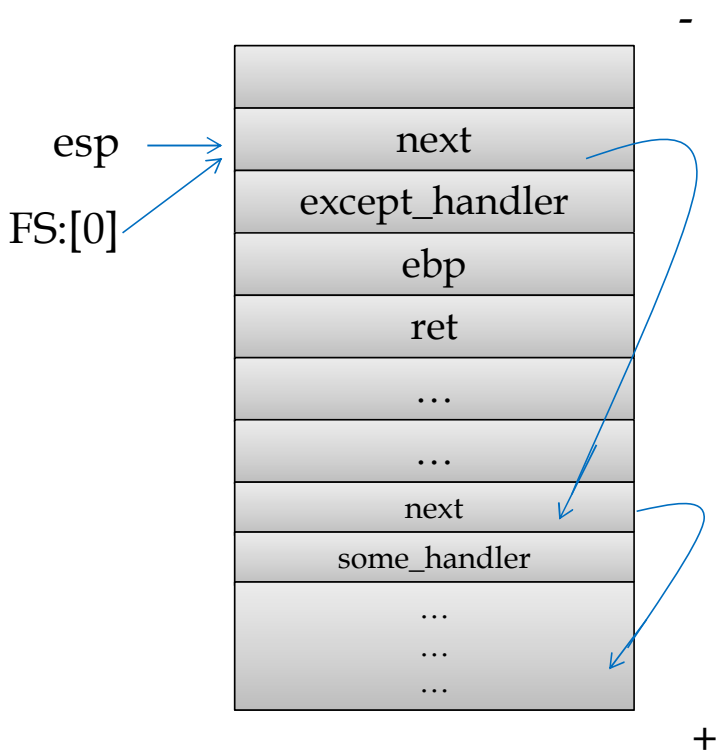
KNCIP

```
push except_handler
push fs:[0]
mov fs:[0],esp

... some code


pop fs:[0]
add esp,4


except_handler:
…some code…
mov eax , ?
ret
```



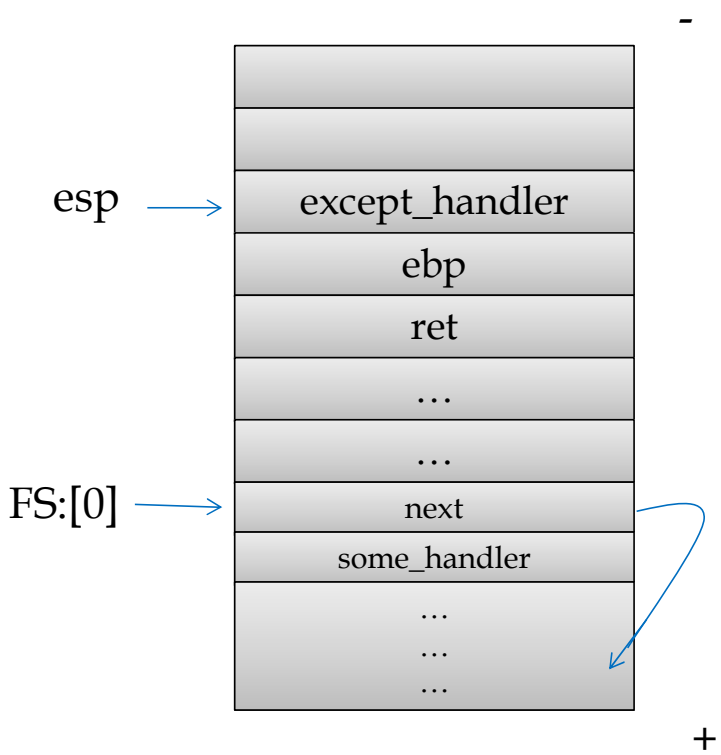
דוגמא

```
push except_handler
push fs:[0]
mov fs:[0],esp

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



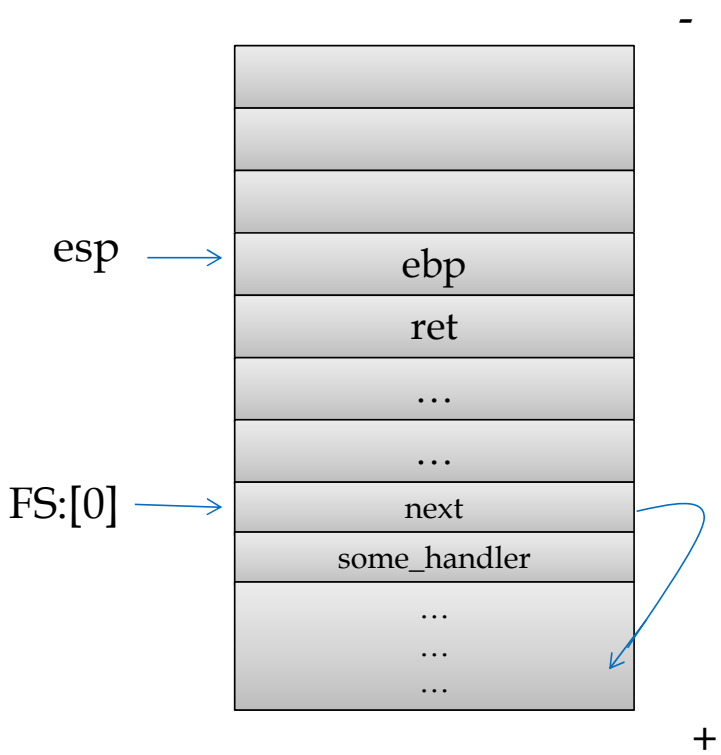
KNCIP

```
push except_handler
push fs:[0]
mov fs:[0],esp

... some code

pop fs:[0]
add esp,4

except_handler:
…some code…
mov eax , ?
ret
```



שחרור משאבים

- כאמור, בעת חריגה מ״ה עוברת על הרשימה עד למציאת ה-handler שמצליח לטפל בחריגה.
- ה-handler שמצליח אחראי למחוק את יתרת המחסנית ואת רשומות ה-Exception Registration המיותרות.
  - יש לו חופש פעולה למחוק את מה שרלוונטי לאותו מקרה.
  - בדר״כ כל ה-handlers שסירבו לטפל בחריגה, כולל זה שהצליח.
  - אבל לעיתים ללא מחיקה.
    - למשל תיקון חילוק באפס שמחזיר ערך קבוע כתוצאת החילוק.
- במהלך המחיקה, כל handler שנמחק נקרא פעם נוספת לצורך שחרור משאבים.
  - למשל שחרור מחלקה ב-++C.
- לא נרחיב לעומק כיצד המנגנון עובד.

שאלה

- מה קורה אם נזרקת חריגה בקוד מבלי שהשתמשנו ב-try?
- תשובה:
  - בתחילת הריצה מ״ה מאתחלת את fs:[0] במצביע ל-handler שלה.
  - כאשר ה-handler נקרא, הוא מציג הודעה מתאימה.
  - הוא נקרא אם כל האחרים לא הצליחו לטפל בחריגה.

סיכום חריגות ברמת התהליך

- ה-Thread צריך לספק למערכת ההפעלה רשימה מקושרת של Exception Registration.
- הקומפיילר מממש את המנגנון שלו בשימוש ב-SEH.
  - כל try יוצר ExceptionRegistration.
  - ה-handler יהיה קוד פנימי של הקומפיילר שייקרא בסוף לקוד שלכם (מה שכתבתם בתוך ה-except).
  - הקומפיילר כאמור יכול להרחיב את המנגנון.
    - למשל באתחול ה-ExceptionRegistration נראה פקודות נוספות שקומפיילר יצר.

SEH Implementation

```
0189FF88  0189FF94
0189FF8C  764B1174  kernel32.764B1174
0189FF90  00000060
0189FF94  0189FFD4
0189FF98  777EB429  RETURN to ntdll.777EB429
0189FF9C  00000060
0189FFA0  7626C3EF
0189FFA4  00000000
0189FFA8  00000000
0189FFAC  00000060
0189FFB0  00000000
0189FFB4  00000000
0189FFB8  00000000
0189FFBC  0189FFA0
0189FFC0  00000000
0189FFC4  FFFFFFFF  End of SEH chain
0189FFC8  777AD555  SE handler
0189FFCC  00D1398B
0189FFD0  00000000
0189FFD4  0189FFEC
0189FFD8  777EB3FC  RETURN to ntdll.777EB3FC from ntdll.777EB402
0189FFDC  00401848  vulnserv.00401848
0189FFE0  00000060
0189FFE4  00000000
0189FFE8  00000000
0189FFEC  00000000
0189FFF0  00000000
0189FFF4  00401848  vulnserv.00401848
0189FFF8  00000060
0189FFFC  00000000
```

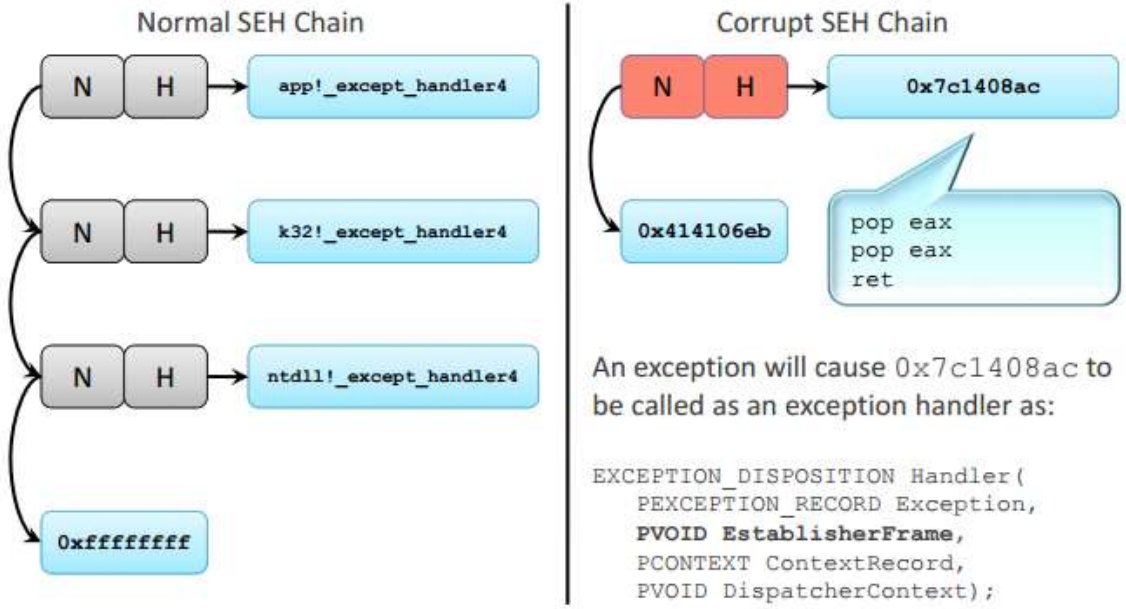
Did you say “STACK”?

- AAA

Buffer Overflow to the Rescue!

019FFF84	41414141	
019FFF88	41414141	
019FFF8C	41414141	
019FFF90	41414141	
019FFF94	41414141	
019FFF98	41414141	
019FFF9C	41414141	
019FFFA0	41414141	
019FFFA4	41414141	
019FFFA8	41414141	
019FFFAC	41414141	
019FFFB0	41414141	
019FFFB4	41414141	
019FFFB8	41414141	
019FFFBC	41414141	
019FFFC0	41414141	
019FFFC4	41414141	Pointer to next SEH record
019FFFC8	41414141	SE handler
019FFFCC	41414141	
019FFFD0	41414141	
019FFFD4	41414141	
019FFFD8	41414141	
019FFFDC	41414141	
019FFFE0	41414141	
019FFFE4	41414141	
019FFFE8	41414141	
019FFFEC	41414141	
019FFFF0	41414141	
019FFFF4	41414141	
019FFFF8	41414141	
019FFFFC	41414141	

Exploit: SEH Overwrite



An exception will cause `0x7c1408ac` to be called as an exception handler as:

```
EXCEPTION_DISPOSITION Handler(
   PEXCEPTION_RECORD Exception,
   PVOID EstablisherFrame,
   PCONTEXT ContextRecord,
   PVOID DispatcherContext);
```

Mitigation: SafeSEH

- VS2003 linker change (/SAFESEH) [9]
- Binaries are linked with a table of safe exception handlers
  - Stored in program memory – not corruptible by an attacker
- Exception dispatcher checks if handlers are safe before calling

When SafeSEH Is Incomplete

[Sotirov and Dowd]

- If DEP is disabled, handler is allowed to be on any non-image page except stack
  - Put attack code on the heap, overwrite exception handler record on the stack to point to it
- If any module is linked without /SafeSEH, handler is allowed to be anywhere in this module
  - Overwrite exception handler record on the stack to point to a suitable place in the module
  - Used to exploit Microsoft DNS RPC vulnerability in Windows Server 2003

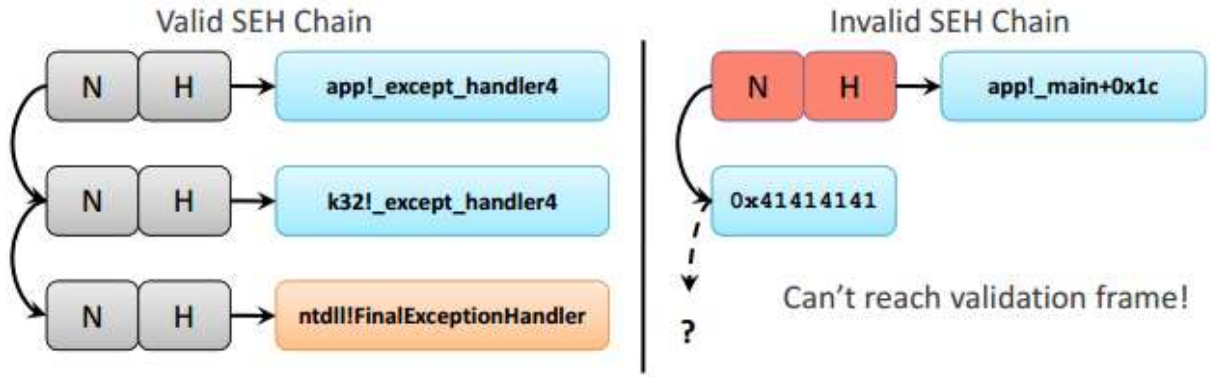
Safe Exception Handling

- Exception handler record must be on the stack of the current thread (why?)
- Must point outside the stack (why?)
- Must point to a valid handler
  - Microsoft's /SafeSEH linker option: header of the binary lists all valid handlers
- Exception handler records must form a linked list, terminating in FinalExceptionHandler
  - Windows Server 2008: SEH chain validation
  - Address of FinalExceptionHandler is randomized (why?)

Sentinel

- Sentinel: a fake value in a linked list whose only role is to be recognizable
- We insert the sentinel at the end of a list
- We also keep a copy of it
- When following the list, we compare every record to the sentinel
  - When we get to the sentinel, we know we reached the end of the list
  - If we never get to the sentinel, we know the list was tampered with

Mitigation: SEHOP



- Dynamic protection for SEH overwrites in Srv08/Vista SP1 [4]
  - No compile/link time hints required
- Symbolic *validation frame* inserted as final entry in chain
- Corrupt `Next` pointers prevent traversal to validation frame

SafeSEH

- /IMAGE_DLLCHARACTERISTICS_NO_SEH
  - A flag set on a DLL that prevents any addresses from the DLL being used as SEH handlers.
- /SafeSEH
  - linker option.
  - Only addresses listed as on a registered SEH handlers list can be used within that module
- SEHOP – SEH Overwrite Protection.
  - Vista an onwards.
  - Checks that the linked list of SEH handlers is valid and -1 terminated.

It is probably not enough...

אולי כדאי לנסות את הפתרון הטבעי?

- האם ניתן להבחין בין איזורי זיכרון שונים?
- דורש תמיכת חומרה ושל מערכת ההפעלה..

x86 processors, since the 80286, included a similar capability implemented at the segment level. However, almost **all operating systems for the 80386 and later x86 processors implement the flat memory model, so they cannot use this capability**

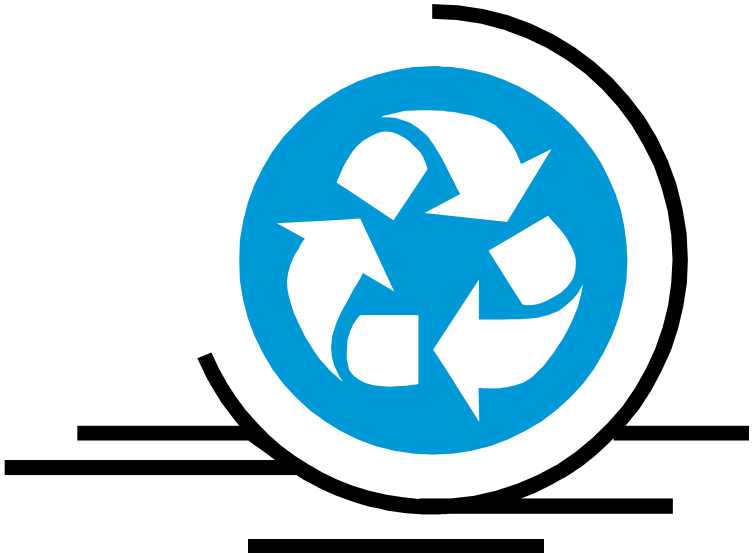
Windows

- Window XP (Service Pack 2)
- Microsoft uses NX bit to: ”prevents the execution of code in memory regions that are marked as data storage”
  - This will NOT prevent an attacker from overrunning the data buffer, but will prevent him from executing his attack (generate an exception)
- Some problems with legitimate code
  - a ”Data Execution Prevention" error message – for legitimate code
  - Workaround - Microsoft allow exceptions, per application. (I.e. turn DEP off for specific apps.)

Breaking DEP

- For years, off by default for compatibility reasons…
- Compatibility problems with plugins:
  - Internet Explorer 8 finally turned on DEP
- Sun JVM allocated its heap memory RWX, allowing us to write shellcode there

Now What?



Ret2Libc => ROP

- Eip made to "return to a function".
  - Create a fake frame on the stack.
- Series of function return
  - "ESP is the new EIP"

The Geometry of Innocent Flesh on the Bone:
Return-into-libc without Function Calls (on the x86)

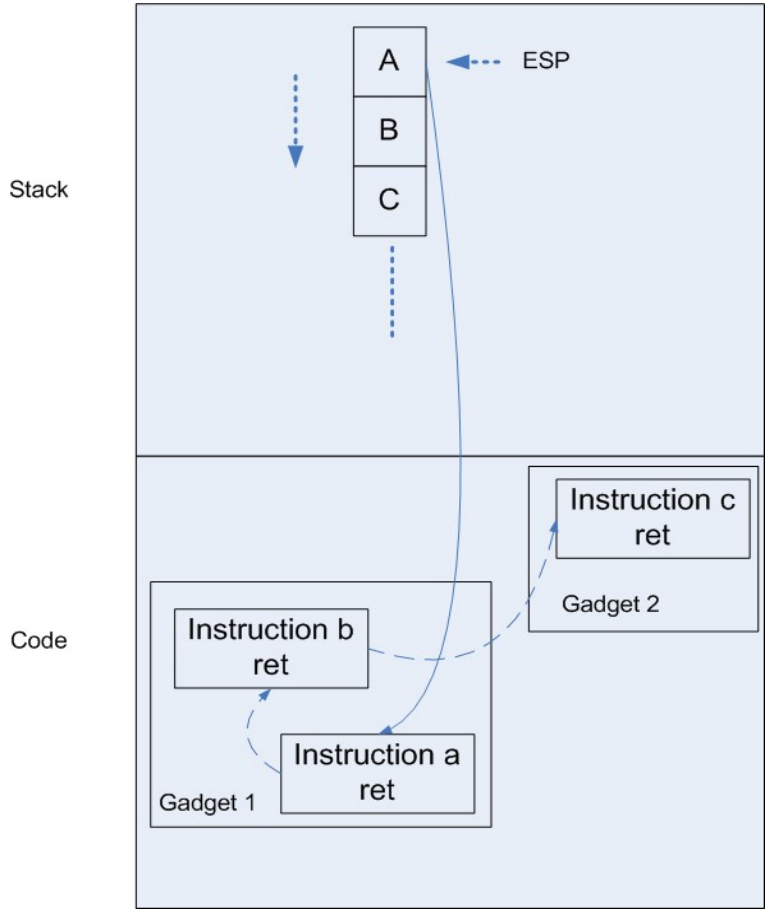
Hovav Shacham*
hovav@cs.ucsd.edu

Abstract

We present new techniques that allow a return-into-libc attack to be mounted on x86 executables that calls *no functions at all*. Our attack combines a large number of short instruction sequences to build *gadgets* that allow arbitrary computation. We show how to discover such instruction sequences by means of static analysis. We make use, in an essential way, of the properties of the x86 instruction set.

Return Oriented Programming

- "Reuse" small segments of code ("Gadgets") by placing them in the correct order on the stack.
- Theoretically possible to create any functionality using ROP.



ROP - Problems

- Need to find the correct "gadgets" for the job.
  - And correlate them.
  - Possible, but takes a lot of time.
- Feasible, but only for short segments of code
  - What do you have\want to use ROP for?
- Standard (Advanced\Theoretical) Exploitation method for the last years.

Gadgets

```
Mem[v2] = v1
```

Desired Logic

Suppose a_2 and a_3 on stack

a_5
v_2
a_3
v_1

esp

Stack

eax	v_1
ebx	
eip	a_1

```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Gadgets

```
Mem[v2] = v1
```

Desired Logic

a_5
v_2
a_3
v_1

esp

Stack

eax	v_1
ebx	
eip	a_3

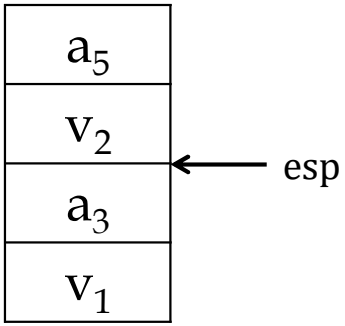
```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Gadgets

`Mem[v2] = v1`

Desired Logic



Stack

eax	v_1
ebx	v_2
eip	a_3

```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Gadgets

`Mem[v2] = v1`

Desired Logic

a_5
v_2
a_3
v_1

esp

Stack

eax	v_1
ebx	v_2
eip	a_4

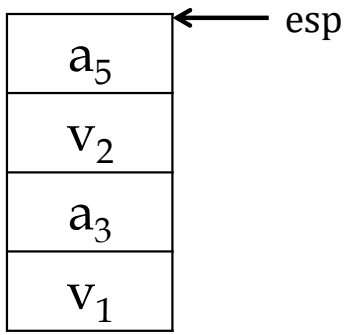
```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Gadgets

`Mem[v2] = v1`

Desired Logic



Stack

eax	v_1
ebx	v_2
eip	a_5

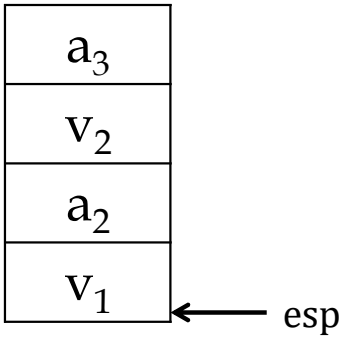
```
a1: pop eax;
a2: ret
a3: pop ebx;
a4: ret
a5: mov [ebx], eax
```

Implementation 2

Equivalence

```
Mem[v2] = v1
```

Desired Logic



Stack

semantically equivalent

"Gadgets"

```
a1: pop eax; ret
a2: pop ebx; ret
a3: mov [ebx], eax
```

Implementation 2

Equivalence

```
Mem[v2] = v1
```

Desired Logic

a_3
v_2
a_2
v_1

Stack

```
a1: pop eax; ret
...
a3: mov [ebx], eax
...
a2: pop ebx; ret
```

Address independent!

```
a1: pop eax; ret
a2: pop ebx; ret
a3: mov [ebx], eax
```

Implementation 2

Address Space Layout Randomization (ASLR)

- You cannot use code if you don't know where it is…
- /DynamicBase option in VS.
  - Windows 7: 8 bits of randomness for DLLs
    - aligned to 64K page in a 16MB region ⇒ 256 choices
  - Windows 8: 24 bits of randomness on 64-bit processors
- Not all DLLs and EXE support this flag.

שאלות?