Smashing the stack for fun and … fun

By Karim Hossen && Guillaume Touron

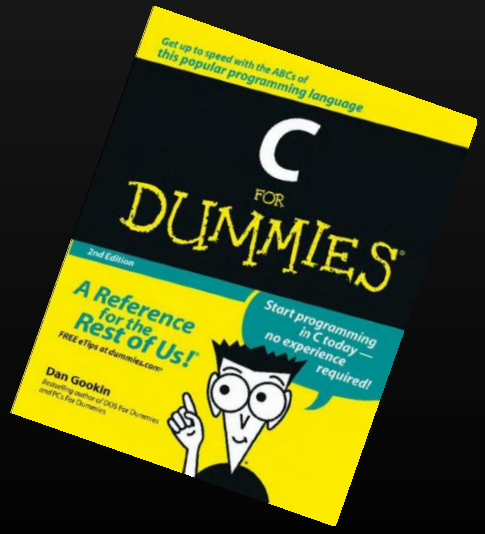
10/06/2011

Plan

- Presentation
- Program
  - File format, execution, stack and heap
- Exploitation
  - Stack-based overflow
  - Security mechanisms & bypass
    - Cookie
    - SafeSEH
    - Non executable page
    - Address randomization
- Conclusion
- Discussion

Presentation

- Karim Hossen
  - PhD student at Grenoble INP
  - Working on model inference for security in VASCO
  - Interested in cryptography and reverse engineering
  - Proud to be an XP SP3 user
  - C programming fan



SECUR IMAG
The security team of Ensimag

Presentation

- Guillaume Touron
  - 2nd year Ensimag – Information Systems
  - Also XP SP3 user
  - In C I trust
  - Low-level programming
  - ReactOS source code reader

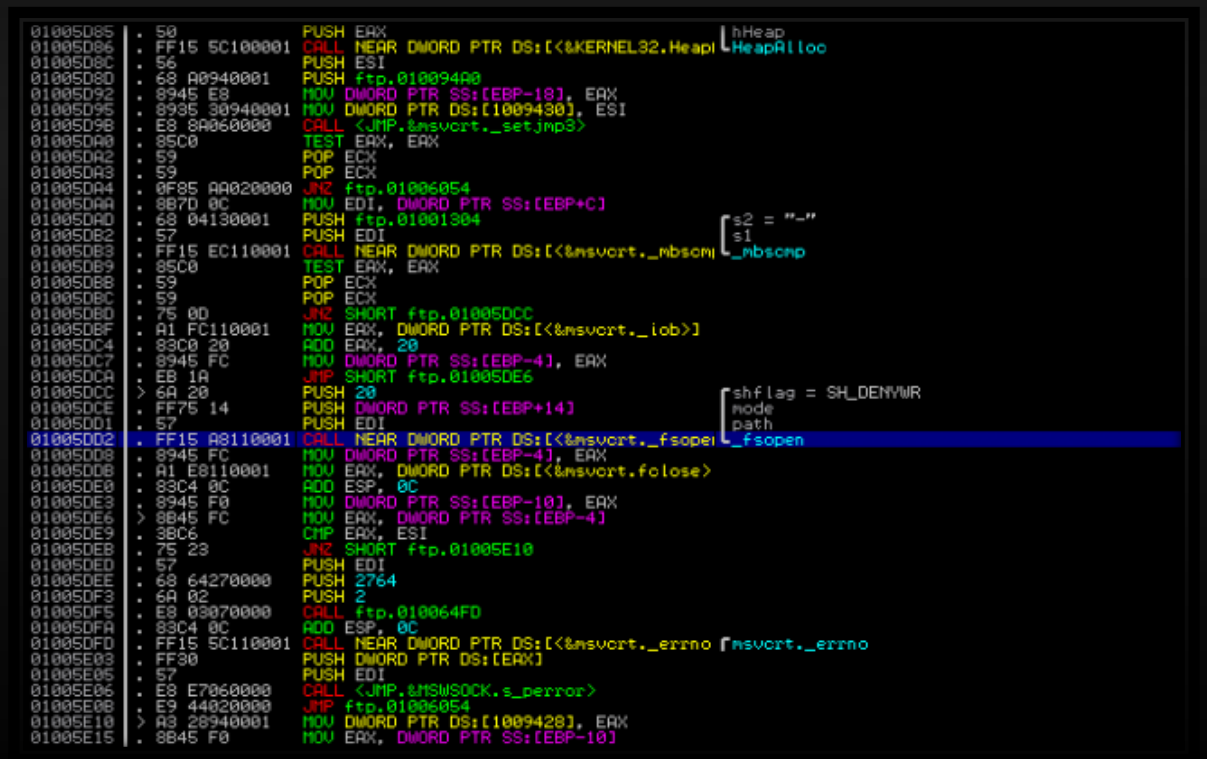
SECUR IMAG
The security team of Ensimag

Quote

"Know thy self, know thy enemy. A thousand battles, a thousand victories." Sun Tzu

Program

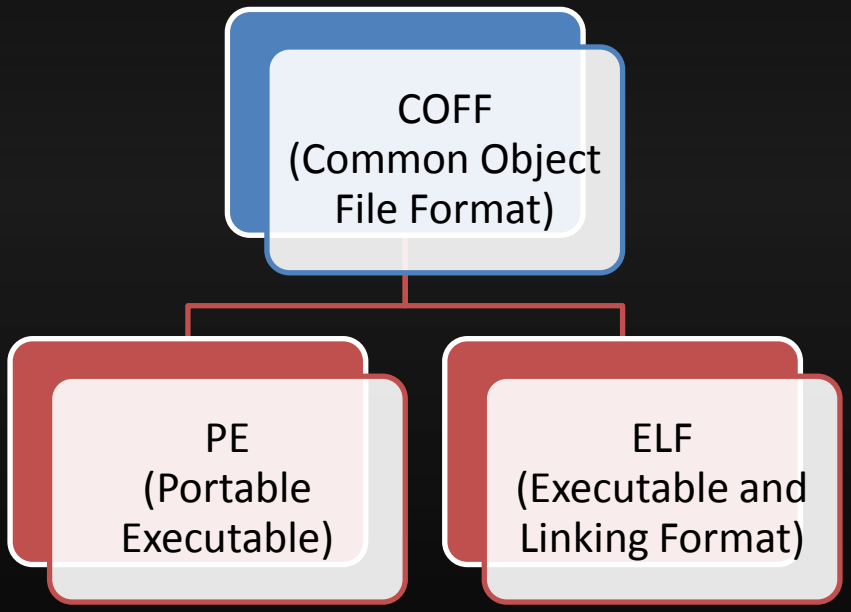
- More than 600 programming languages, but only one used by the machine.



```
01005D85 . 50            PUSH EAX                                 hHeap
01005D86 . FF15 5C100001 CALL NEAR DWORD PTR DS:[<&KERNEL32.Heapl HeapAlloc
01005D8C . 56            PUSH ESI
01005D8D . 68 A0940001   PUSH ftp.010094A0
01005D92 . 8945 E8       MOV DWORD PTR SS:[EBP-18], EAX
01005D95 . 8935 30940001 MOV DWORD PTR DS:[1009430], ESI
01005D9B . E8 8A060000   CALL <JMP.&msvcrt._setjmp3>
01005DA0 . 85C0          TEST EAX, EAX
01005DA2 . 59            POP ECX
01005DA3 . 59            POP ECX
01005DA4 . 0F85 AA020000 JNZ ftp.01006054
01005DAA . 8B7D 0C       MOV EDI, DWORD PTR SS:[EBP+C]
01005DAD . 68 04130001   PUSH ftp.01001304                        s2 = "-"
01005DB2 . 57            PUSH EDI                                 s1
01005DB3 . FF15 EC110001 CALL NEAR DWORD PTR DS:[<&msvcrt._mbscm  _mbscmp
01005DB9 . 85C0          TEST EAX, EAX
01005DBB . 59            POP ECX
01005DBC . 59            POP ECX
01005DBD . 75 0D         JNZ SHORT ftp.01005DCC
01005DBF . A1 FC110001   MOV EAX, DWORD PTR DS:[<&msvcrt._iob>]
01005DC4 . 83C0 20       ADD EAX, 20
01005DC7 . 8945 FC       MOV DWORD PTR SS:[EBP-4], EAX
01005DCA . EB 1A         JMP SHORT ftp.01005DE6
01005DCC > 6A 20         PUSH 20                                  shflag = SH_DENYWR
01005DCE . FF75 14       PUSH DWORD PTR SS:[EBP+14]               mode
01005DD1 . 57            PUSH EDI                                 path
01005DD2 . FF15 A8110001 CALL NEAR DWORD PTR DS:[<&msvcrt._fsopel _fsopen
01005DD8 . 8945 FC       MOV DWORD PTR SS:[EBP-4], EAX
01005DDB . A1 E8110001   MOV EAX, DWORD PTR DS:[<&msvcrt.fclose>]
01005DE0 . 83C4 0C       ADD ESP, 0C
01005DE3 . 8945 F0       MOV DWORD PTR SS:[EBP-10], EAX
01005DE6 > 8B45 FC       MOV EAX, DWORD PTR SS:[EBP-4]
01005DE9 . 3BC6          CMP EAX, ESI
01005DEB . 75 23         JNZ SHORT ftp.01005E10
01005DED . 57            PUSH EDI
01005DEE . 68 64270000   PUSH 2764
01005DF3 . 6A 02         PUSH 2
01005DF5 . E8 03070000   CALL ftp.010064FD
01005DFA . 83C4 0C       ADD ESP, 0C
01005DFD . FF15 5C110001 CALL NEAR DWORD PTR DS:[<&msvcrt._errno  msvcrt._errno
01005E03 . FF30          PUSH DWORD PTR DS:[EAX]
01005E05 . 57            PUSH EDI
01005E06 . E8 E7060000   CALL <JMP.&MSWSOCK.s_perror>
01005E0B . E9 44020000   JMP ftp.01006054
01005E10 > A3 28940001   MOV DWORD PTR DS:[1009428], EAX
01005E15 . 8B45 F0       MOV EAX, DWORD PTR SS:[EBP-10]
```

Program

- Executable file ?



Program

- Common Object File Format
  - Introduced in AT&T's UNIX System V (1983)
  - For executable, object code and shared library
  - On Unix systems
  - Replaced the previously used “a.out” format
  - Too limited and incompletely specified
    - Unable to support real world languages like C
    - Replaced by ELF in Unix since SVR4
    - Replaced by PE since Windows NT 3.1

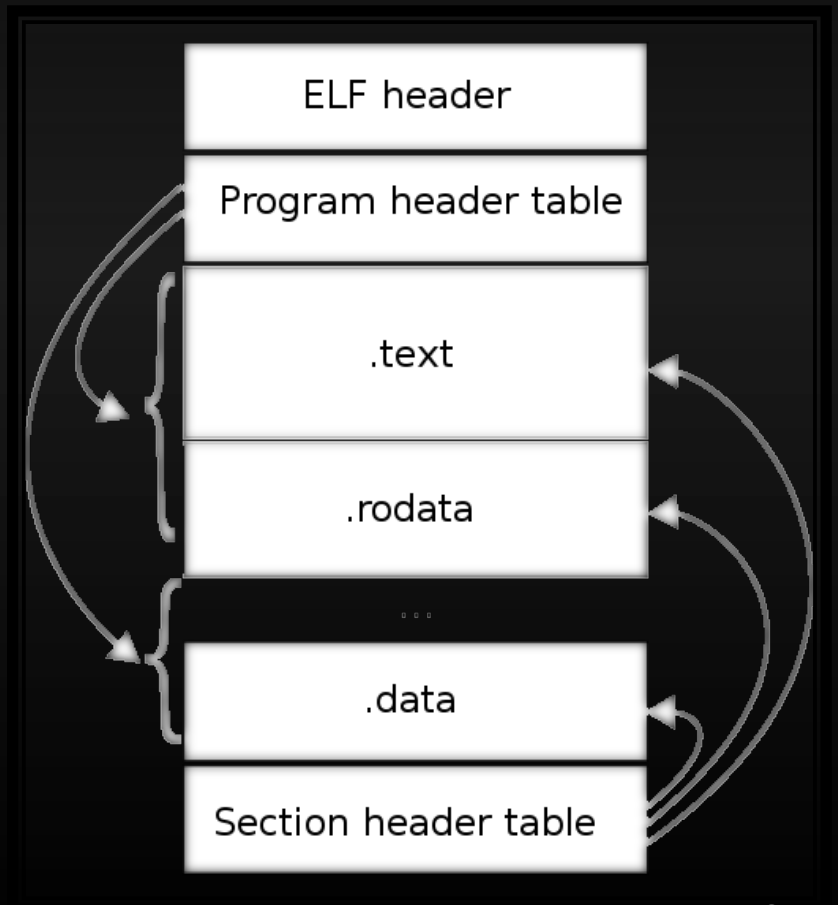
Program

- Executable and Linking Format
  - For executable, object code, shared library and core dumps
  - Flexible and extensible
  - Used in almost all Unix systems, Playstation …
  - Not bound to the architecture
    - AMD64, IAxx, ARM, MIPS, PowerPC …
  - Tools : readelf, objdump, file

SECUR IMAG
The security team of Ensimag

Program

- ELF Layout
  - Program header table, describing (segments)*
  - Section header table, describing (sections)*
  - Data referred to by entries in the program HT or section HT.



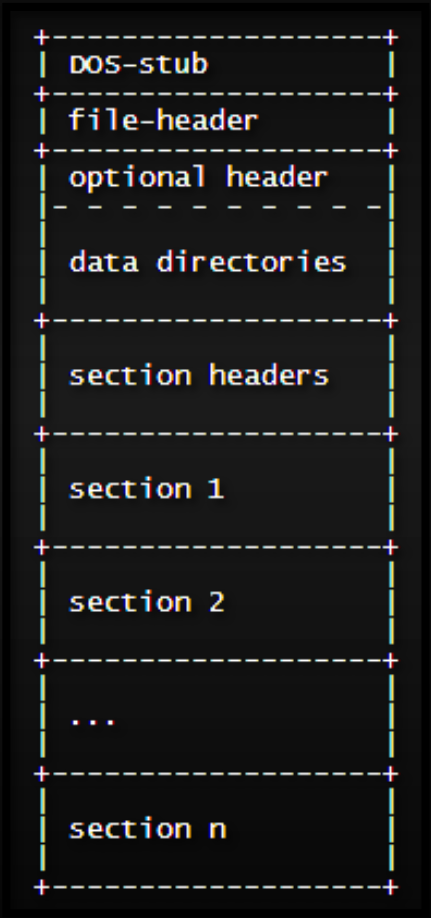
Program

- Portable Executable
  - Developed by 
  - Windows NT was able to support other Architecture → « Portable »
  - Used by all windows platform
  - For binaries, drivers, .ocx, .dll, .cpl
  - Useful to do reverse engineering

An In-Depth Look into the Win32 Portable Executable File Format
http://msdn.microsoft.com/en-us/magazine/cc301808.aspx

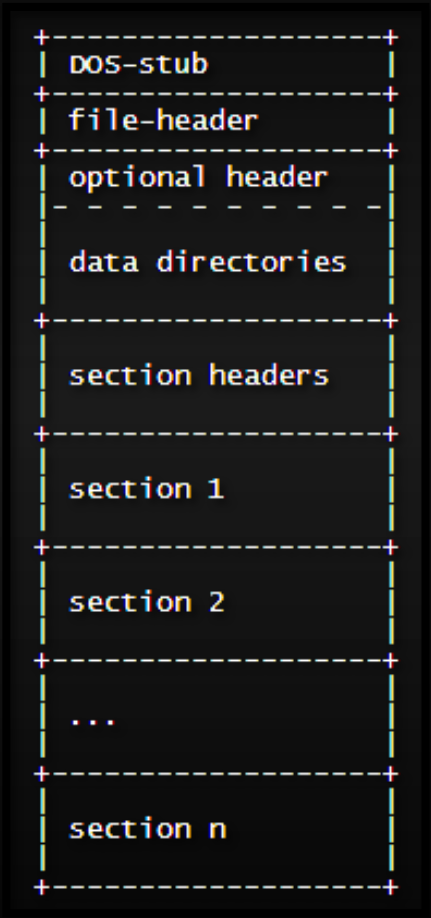
Program

- PE Layout
  - DOS stub
    - MS-DOS 2.0 compatible executable
    - Output an error message such as "this program needs windows NT".
    - Any PE file are valid MS-DOS exe.



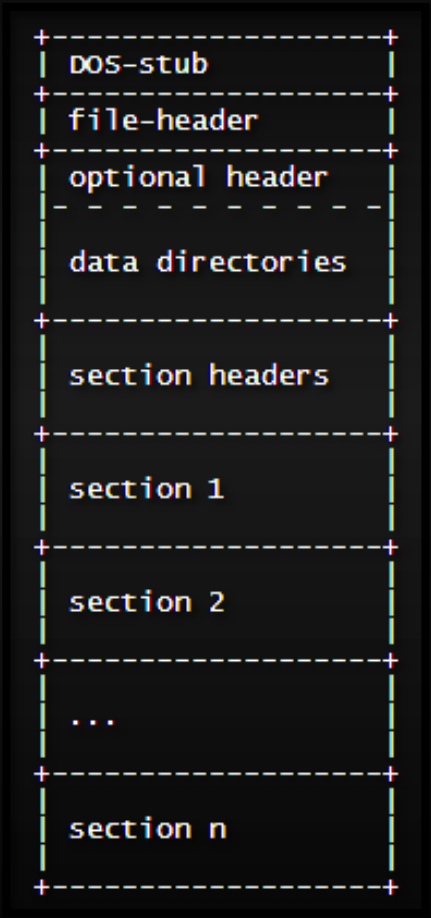
Program

- PE Layout
  - File Header
    - Machine
    - NumberOfSections
    - Timestamp
    - Characteristics
    - PointerToSymbolTable
    - NumberOfSymbols



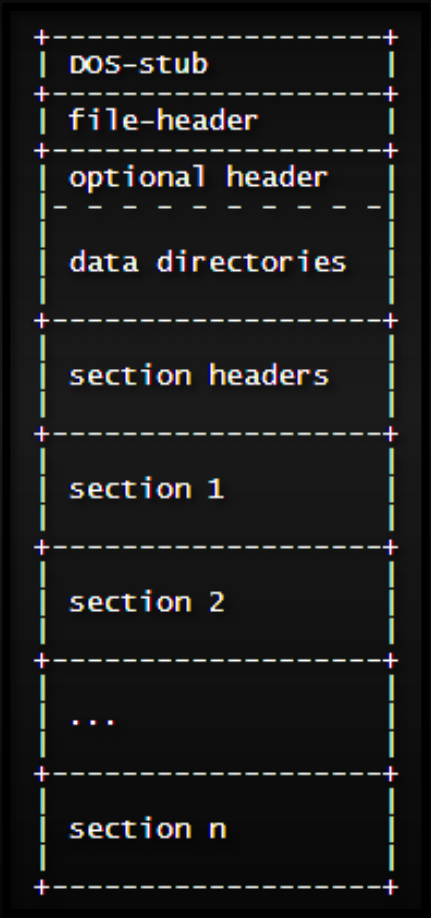
Program

- PE Layout
  - Optional Header
    - AddressOfEntryPoint
    - ImageBaseSubsystem
    - MajorSubSystemVersion
    - MinorSubSystemVersion
    - Subsystem
    - DataDirectory
      - IAT (Imports Address Table)



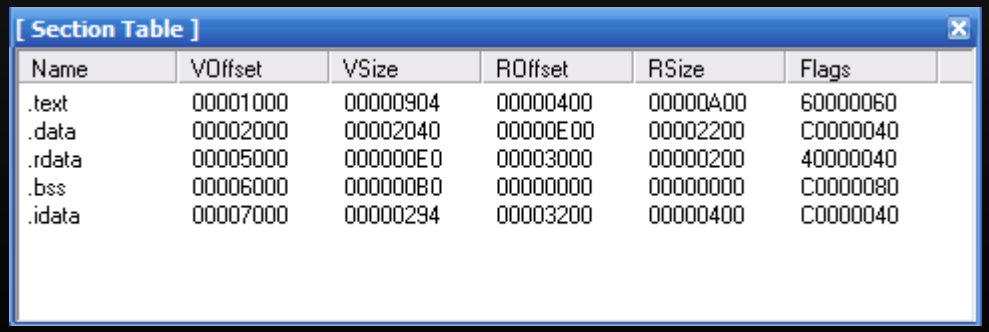
Program

- PE Layout
  - Section Header
    - Name
    - VirtualAddress
    - SizeOfRawData
    - PointerToRawData
    - Characteristics
      - Permissions
        Shareable, executable, readable, writable
      - Type of code
        executable/initialised/uninitialised code



Program

- PE Layout
  - Common sections
    - text : executable code (Read only)
    - data : global variables
    - Rdata : readonly data, strings, constants
    - Bss : uninitialized data, static variables

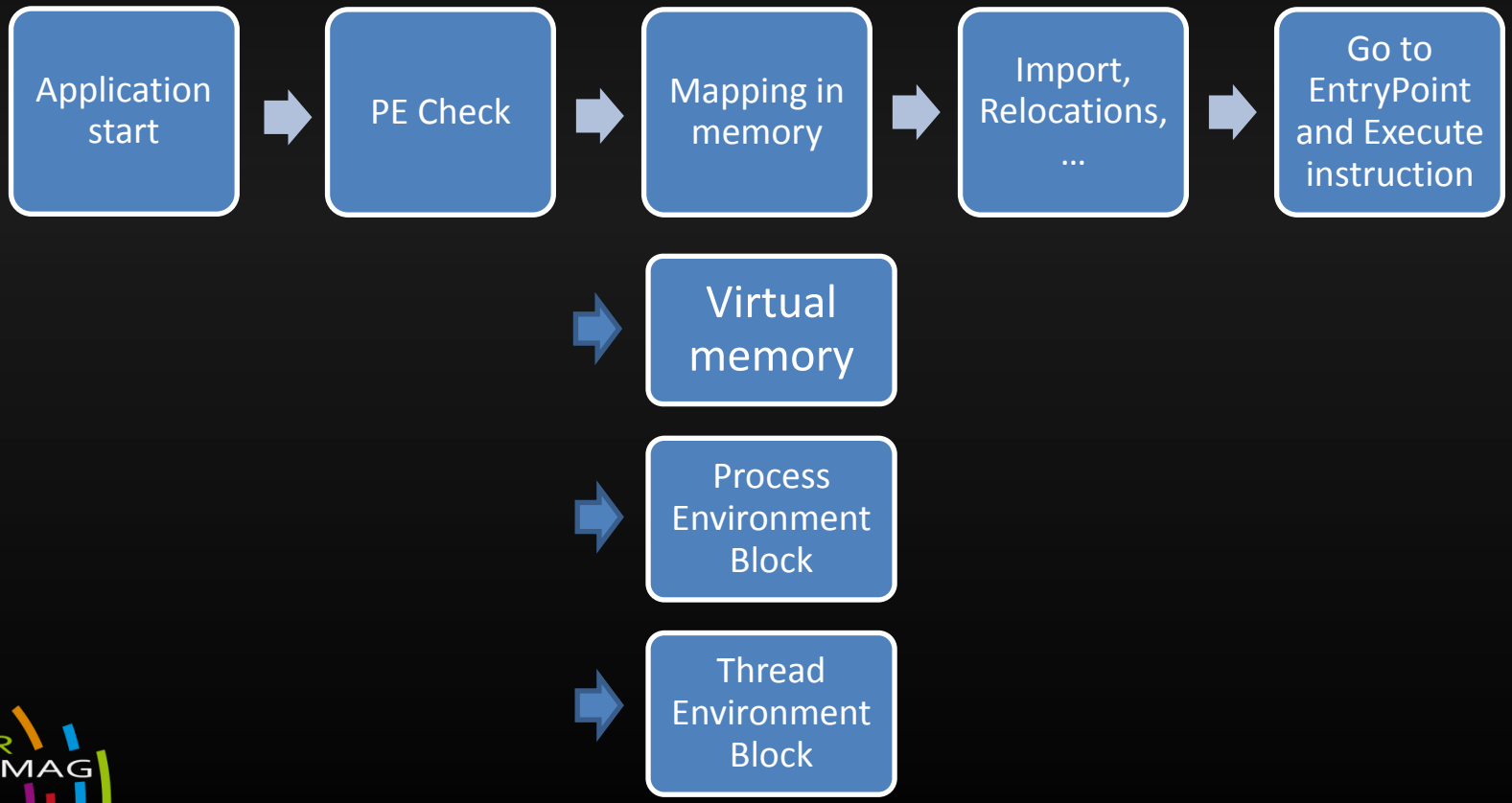


[Section Table]

Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	00000904	00000400	00000A00	60000060
.data	00002000	00002040	00000E00	00002200	C0000040
.rdata	00005000	000000E0	00003000	00000200	40000040
.bss	00006000	000000B0	00000000	00000000	C0000080
.idata	00007000	00000294	00003200	00000400	C0000040

Program

- Execution

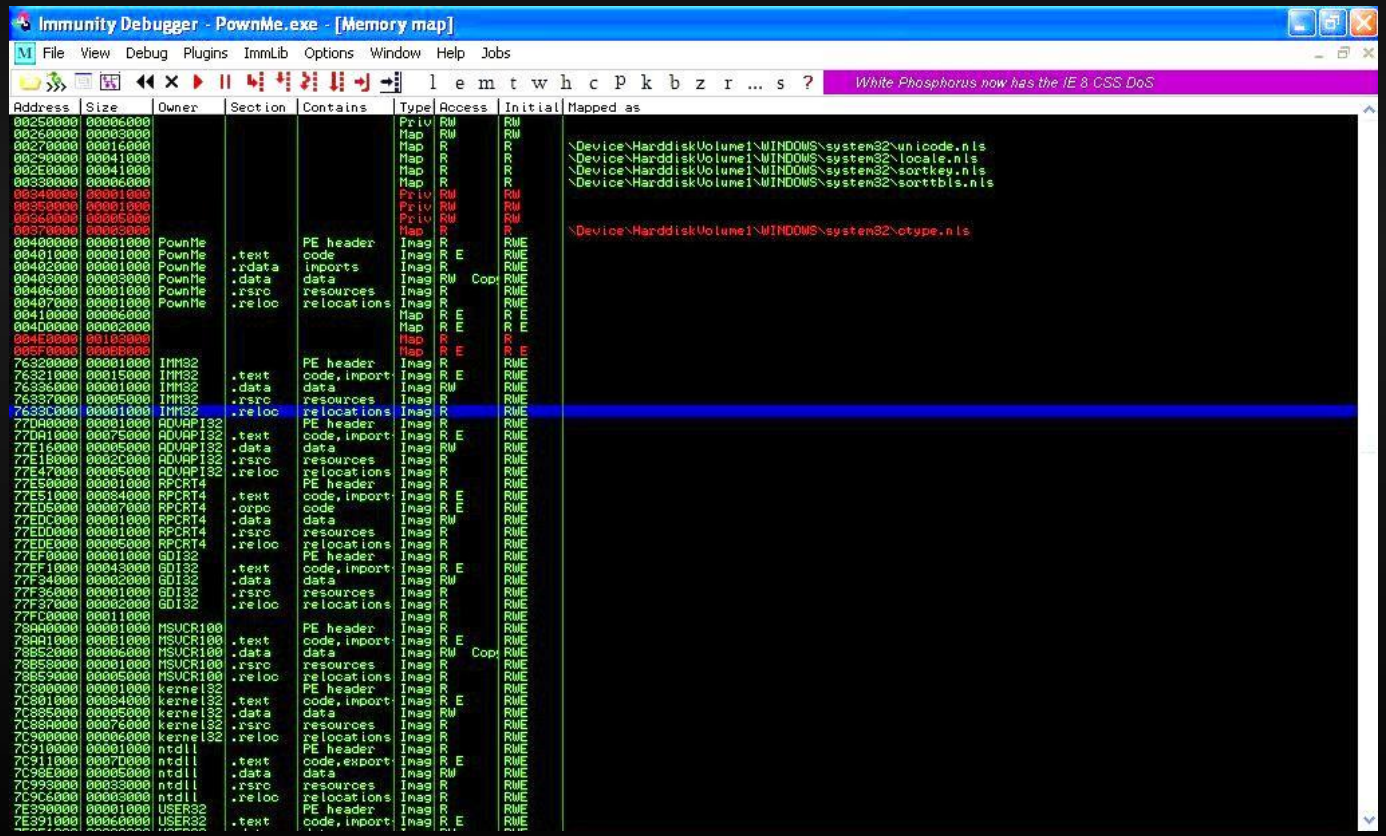


Program

- Execution
  - Virtual memory
    - Flat memory model
    - In a 32 bit process, the address ranges from 0×00000000 to 0xFFFFFFFF
    - 0×00000000 to 0x7FFFFFFF is assigned to "user-land“
    - 0×80000000 to 0xFFFFFFFF is assigned to "kernel land“
    - Kernel land memory is only accessible by the OS.

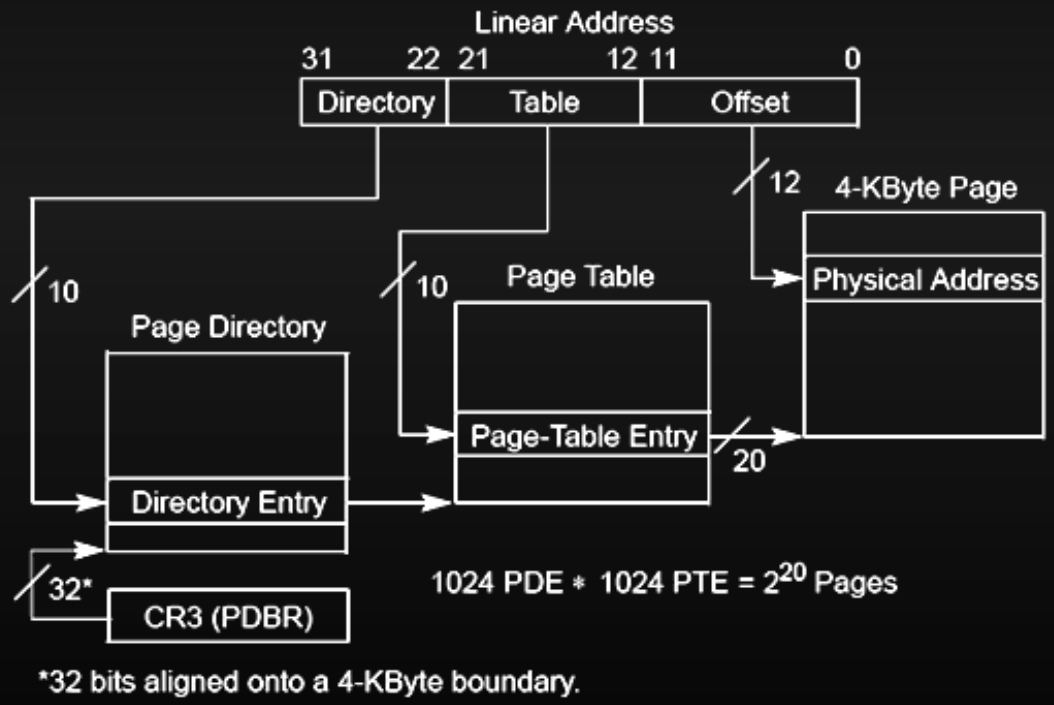
Program

- Memory mapping



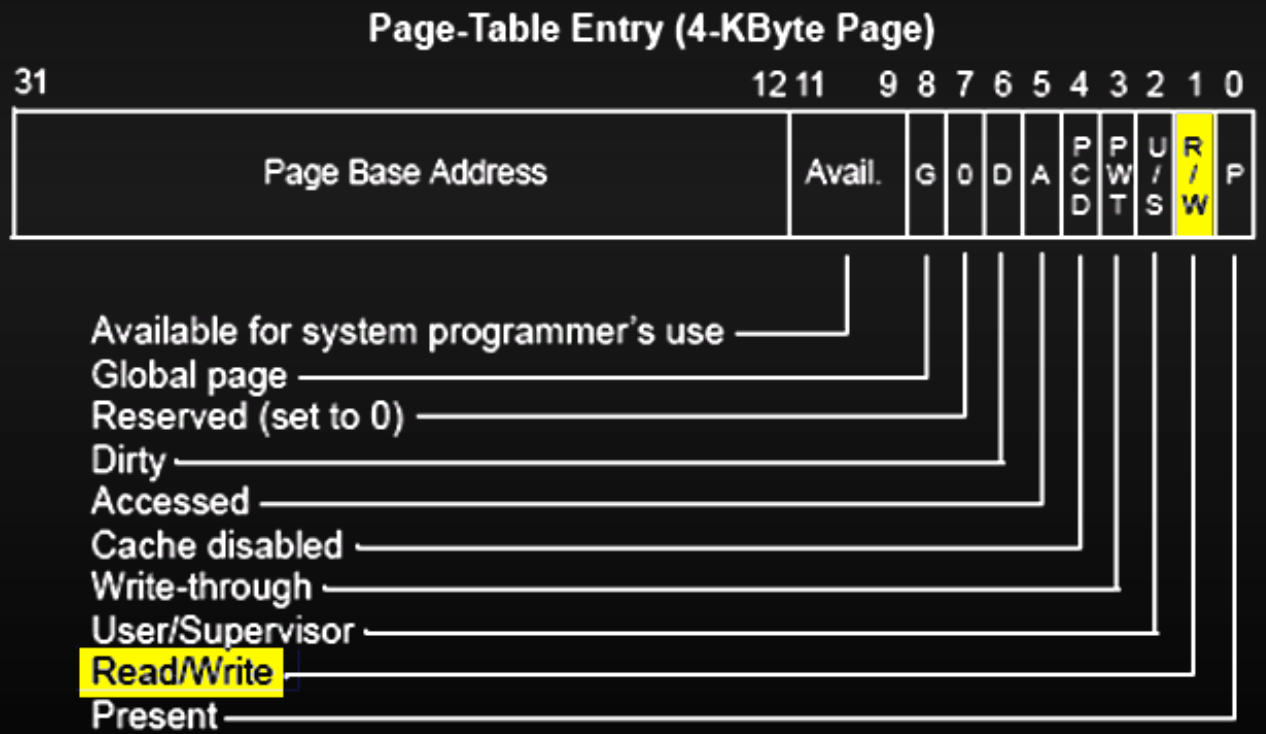
Security Mechanisms - DEP

- Virtual Memory



Security Mechanisms - DEP

- Page Table Entry



Program

- Execution
  - PEB (Process Environment Block)
    - FS:[30] from userland
    - Contains all user land parameters of the process
      - Location of the main executable
      - Pointer to loader data (can be used to list all dll's / modules that are/can be loaded into the process)
        - Useful for generic shellcode !
      - Pointer to information about the heap
      - Whether the process is being debugged

SECUR IMAG
The security team of Ensimag

Program

- Execution
  - PEB (P
    - Usec

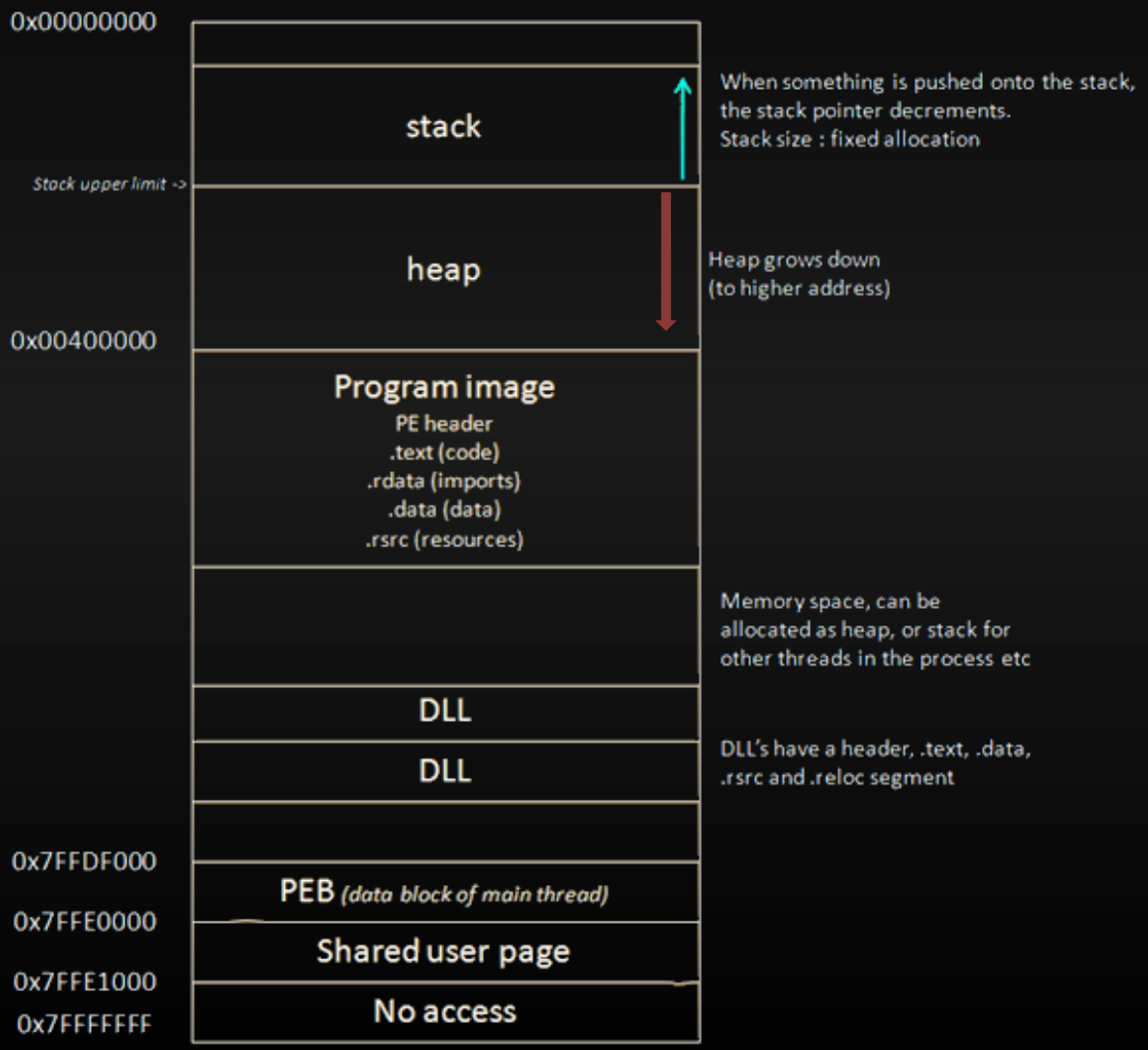
mov ebx, [ebx]
mov ebx, [ebx + 0x10]

Program

- Execution
  - TEB (Thread Environment Block)
    - FS:[0] from Userland
    - Describes the state of a thread
      - location of the PEB in memory
      - location of the stack for the thread it belongs to
      - pointer to the first entry in the SEH chain
      - ProcessId / ThreadId
      - Current SEH frame (FS:[0] chain)
      - Stack pointers

SECUR IMAG
The security team of Ensimag

Program


0x00000000
stack
Stack upper limit ->
heap
0x00400000
Program image
PE header
.text (code)
.rdata (imports)
.data (data)
.rsrc (resources)
DLL
DLL
0x7FFDF000
PEB (data block of main thread)
0x7FFE0000
Shared user page
0x7FFE1000
No access
0x7FFFFFFF
When something is pushed onto the stack, the stack pointer decrements.
Stack size : fixed allocation
Heap grows down
(to higher address)
Memory space, can be allocated as heap, or stack for other threads in the process etc
DLL's have a header, .text, .data, .rsrc and .reloc segment

Program

- Execution
  - CPU registers (Intel, x86) are :
    - *EAX* : accumulator : calculations, return values
    - *EBX* : base (does not have anything to do with base pointer). Can be used to store data.
    - *ECX* : counter : used for iterations
    - *EDX* : data : this is an extension of the EAX register
    - *ESP* : stack pointer
    - *EBP* : base pointer
    - *ESI* : source index : holds location of input data
    - *EDI* : destination index : points to location of where result of data operation is stored
    - *EIP* : instruction pointer

Program

- Execution
  - Segment registers
    - *CS: code segment*
    - *DS : data segment*
    - *SS : stack segment*
    - *ES : Extra segment*
    - *FS : Extra segment too (E -> F)*
    - *GS : Extra segment again (F -> G)*

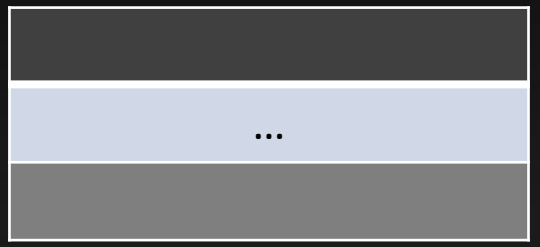
Program

- Stack
  - Data structure that works LIFO (last in first out)
  - Allocated by the OS
  - Pretty fast but limited in size
  - Contains local variables, function call, temp data
  - Grows to a lower address
  - One stack frame by function

Program

- Stack
  - Exemple

State : Before func call



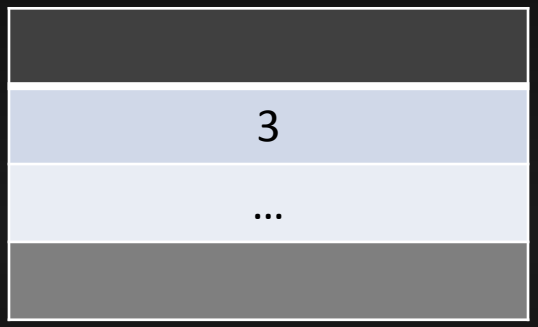

```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

------ ESP ------ EBP

Program

- Stack
  - Exemple

State : Push 3



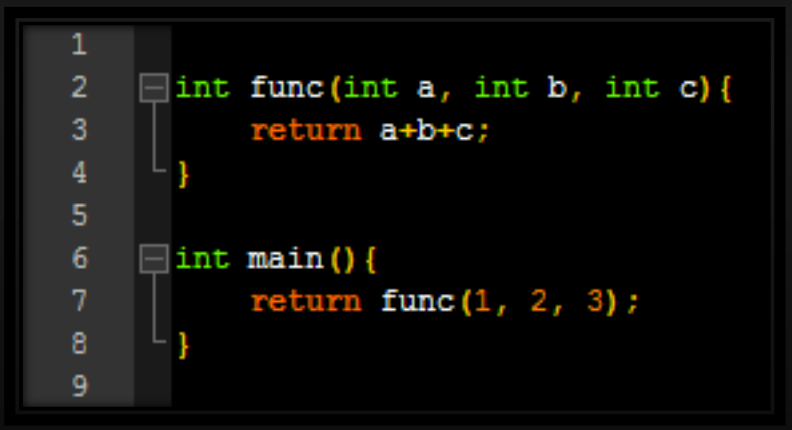

```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

------ ESP ------ EBP

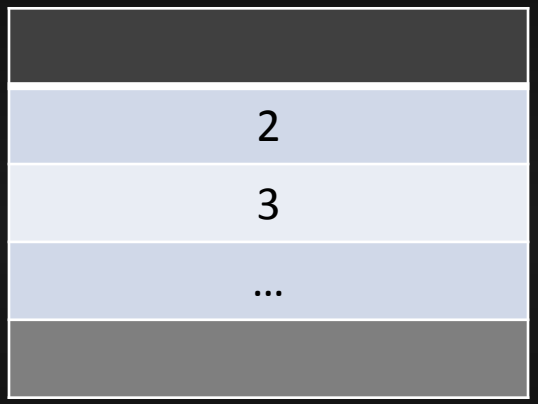
Program

- Stack
  - Exemple

State : Push 2



```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

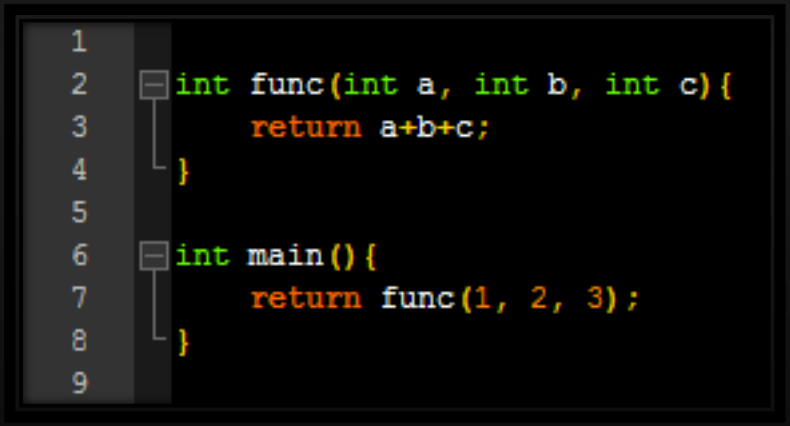


------ ESP ------ EBP

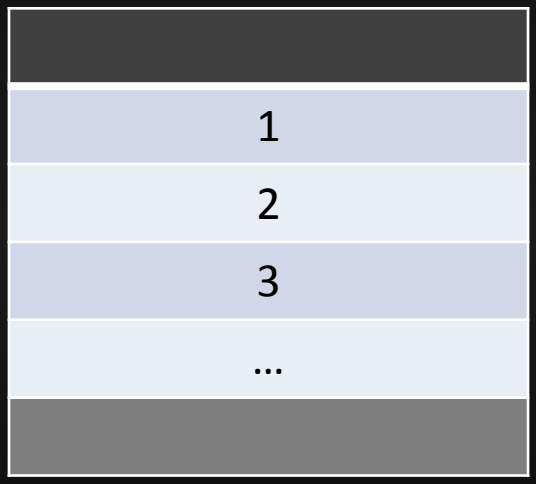
Program

- Stack
  - Exemple

State : Push 1



```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```



------ ESP ------ EBP

Program

- Stack
  - Exemple

```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

State : Push @Instr after func

------ ESP ------ EBP

Program

- Stack
  - Exemple

```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

State : In func. Save ebp

------ ESP ------ EBP

Program

- Stack
  - Exemple

```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

State : New place for func.

Program

- Stack
  - Exemple

State : End of func. Restore esp, pop esp

```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

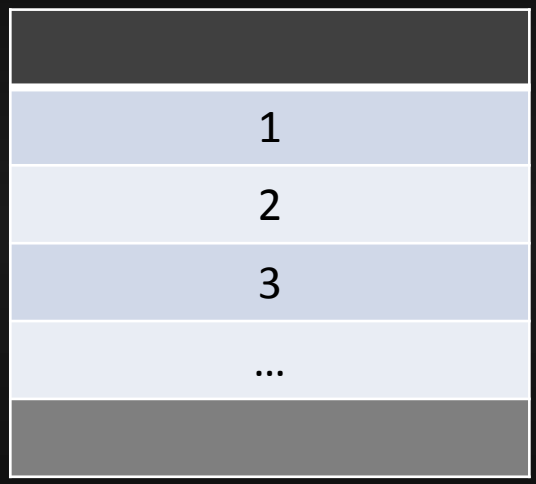
------ ESP ------ EBP

Program

- Stack
  - Exemple

```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

State : Return to caller (ret 0C)

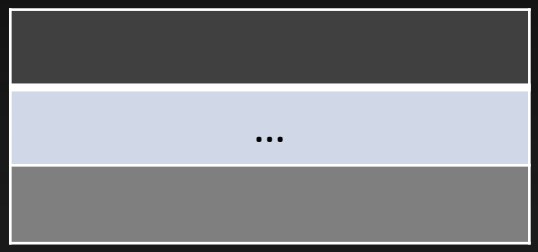


------ ESP ------ EBP

Program

- Stack
  - Exemple

State : After main another leave …




```
1
2  int func(int a, int b, int c){
3      return a+b+c;
4  }
5
6  int main(){
7      return func(1, 2, 3);
8  }
9
```

------ ESP ------ EBP

Program

- Let’s pwned bofMe !

SECUR IMAG
The security team of Ensimag

Quote

“Low-level programming is good for the programmer’s soul.” John Carmack

Exploitation

- Simple buffer overflow
  - Example source code

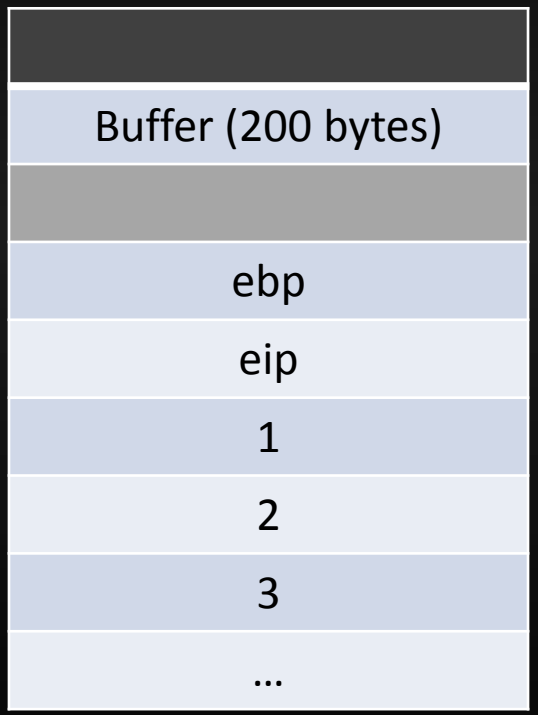
```
1
2
3   #include <stdio.h>
4
5   void vuln(char* s){
6       char buffer[200];
7       strcpy(buffer, s);
8       printf("%s\n", buffer);
9   }
10
11  int main(int argc, char** argv){
12      if (argc > 1) vuln(argv[1]);
13      return 0;
14  }
15
```

1. Get the first argument
2. Call vuln
3. Allocate 200 bytes on the stack
4. Copy argument to buffer using strcpy
5. Print buffer content
6. return

SECUR IMAG
The security team of Ensimag

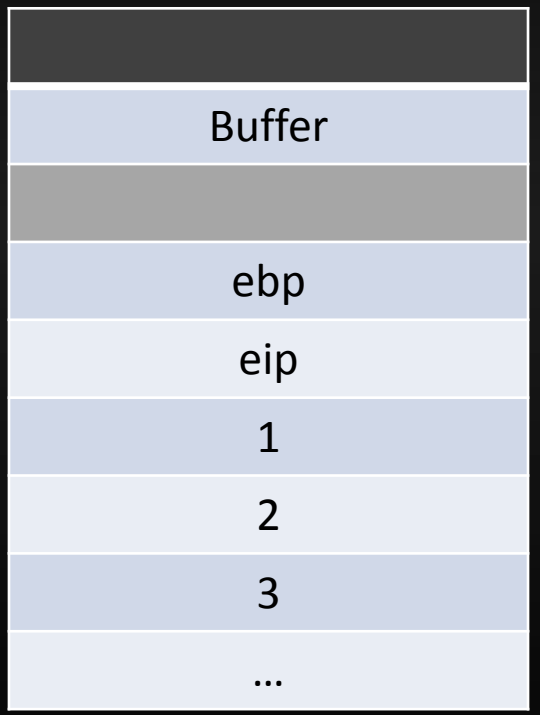
Exploitation

- Simple buffer overflow
  - strcpy
    - Copy user input to buffer
    - Can overflow and overwrite ebp, eip …
    - We control eip
    - We control the program flow
    - What address should I put in eip ?



Exploitation

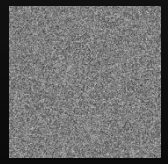
- Simple buffer overflow
  - strcpy
    - Our shellcode starts at Buffer
    - Esp points to Buffer
    - We need an address to « jmp esp » instruction ! (ret overwrite)
    - Find it in the current program or in the other executables modules



Exploitation

- Simple buffer overflow
  - strcpy
    - At the function end
    - Program will jump to the top of the stack (esp)
    - And execute the malicious code.
    - Even if ebp is invalid, code have been executed
    - Pwned ☺

Exploitation

- Security mechanisms we'll see:
  -  Cookie to detect BoF
  -  Use non-executable stack
  -  Randomized base addresses for stack and library

Security Mechanisms

- Cookie
  - Also called (GS flag, canary, -fstack-protector)
  - Method
    - Choose a random value when the program starts
    - Add this value above ebp
    - Check cookie's value in each function's epilogue
    - If the check failed, program is terminated

[buffer][][saved EBP][saved EIP]

/GS (Buffer Security Check)

http://msdn.microsoft.com/en-us/library/8dbf701c%28v=vs.80%29.aspx

Security Mechanisms

- Cookie Bypass (1)
  - Reduce the effective entropy of the cookies
    - Calculating entropy sources from
      - System Time
      - Process and Thread Identifier
      - Tick Count
      - Performance Counter
      - Frame Pointer
    - Need local access to the machine
    - Reduce the entropy to 15 bits

Security Mechanisms

- Cookie Bypass (2)
  - Overwrite stack data in functions up the stack
    - Need pointer to objects or structures in the stack of their caller
    - Overwrite object and vtable pointer
    - Point it to a fake vtable
    - Redirect the virtual function call
    - Execute the evil code ☺

SECUR IMAG
The security team of Ensimag

Security Mechanisms

- Cookie Bypass (3)
  - Use unprotected buffer
    - Cookie is used :
      - When “string” buffers exists
      - More than 4 bytes are allocated
    - Overwrite is still possible for arrays of integer or pointer

SECUR IMAG
The security team of Ensimag

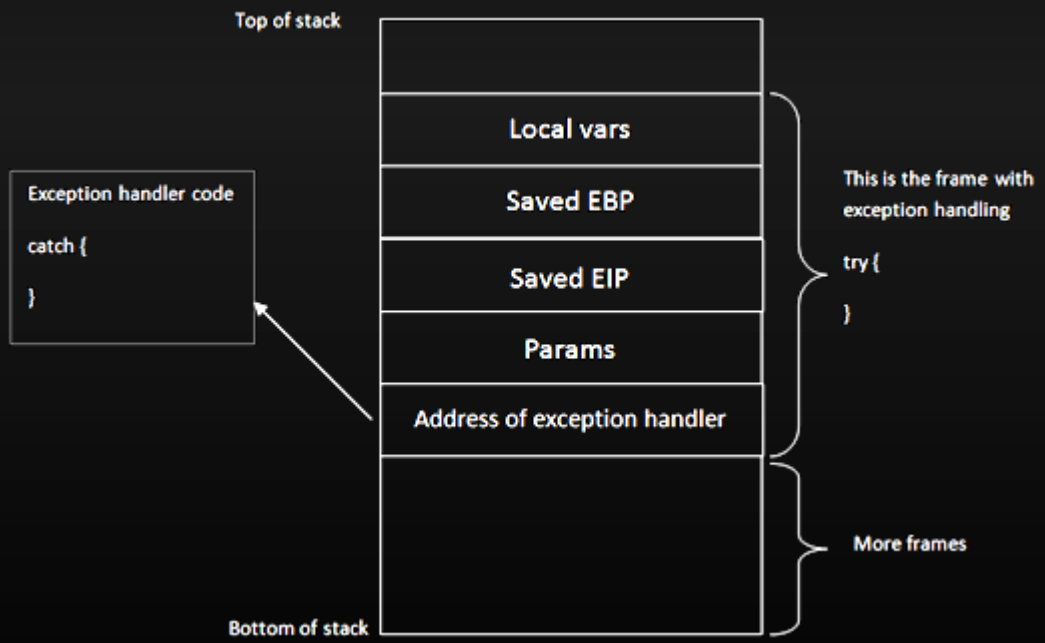
Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)
    - Exception Handler ?
      - Piece of code to deal with exception throws by application
      - A typical EH looks like this :

```
1
2
3   _try
4   {
5     //run stuff.  If an exception occurs, go to <catch> code
6   }
7   _except
8   {
9     // run stuff when exception occurs
10  }
11
12
```

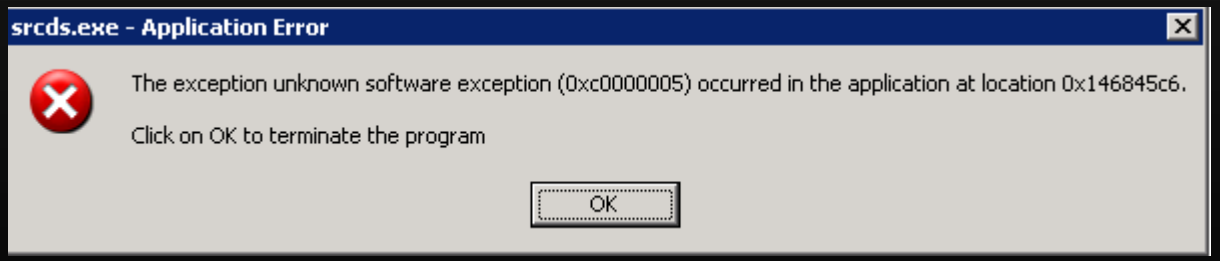
Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)



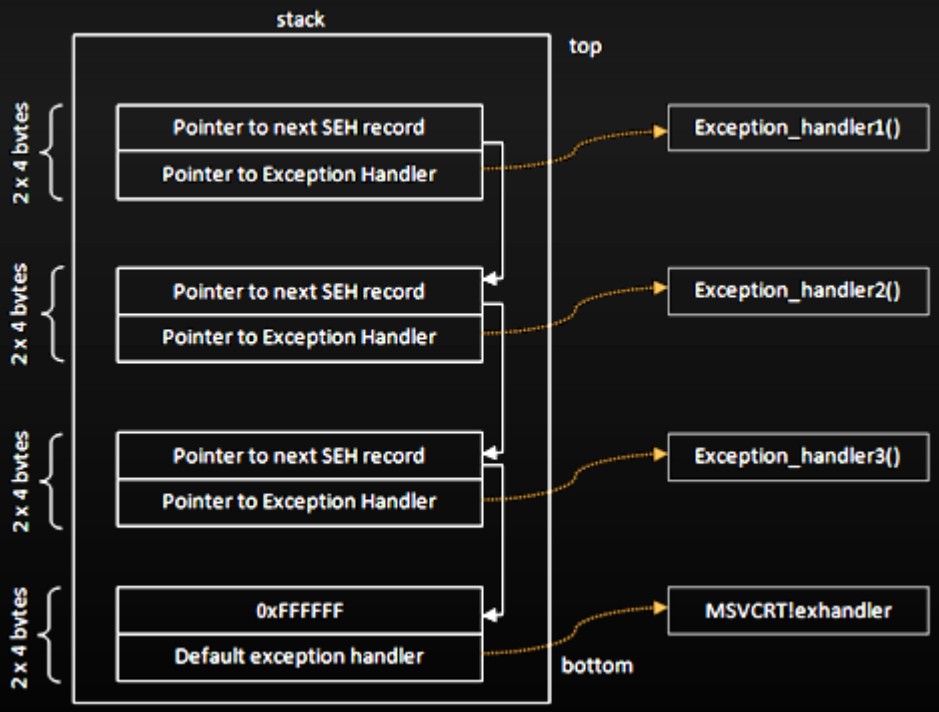
Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)
    - In each stack frame
    - Windows has a default SEH
    - Stored in linked list of exception structure
    - Catch unhandled exception



Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)

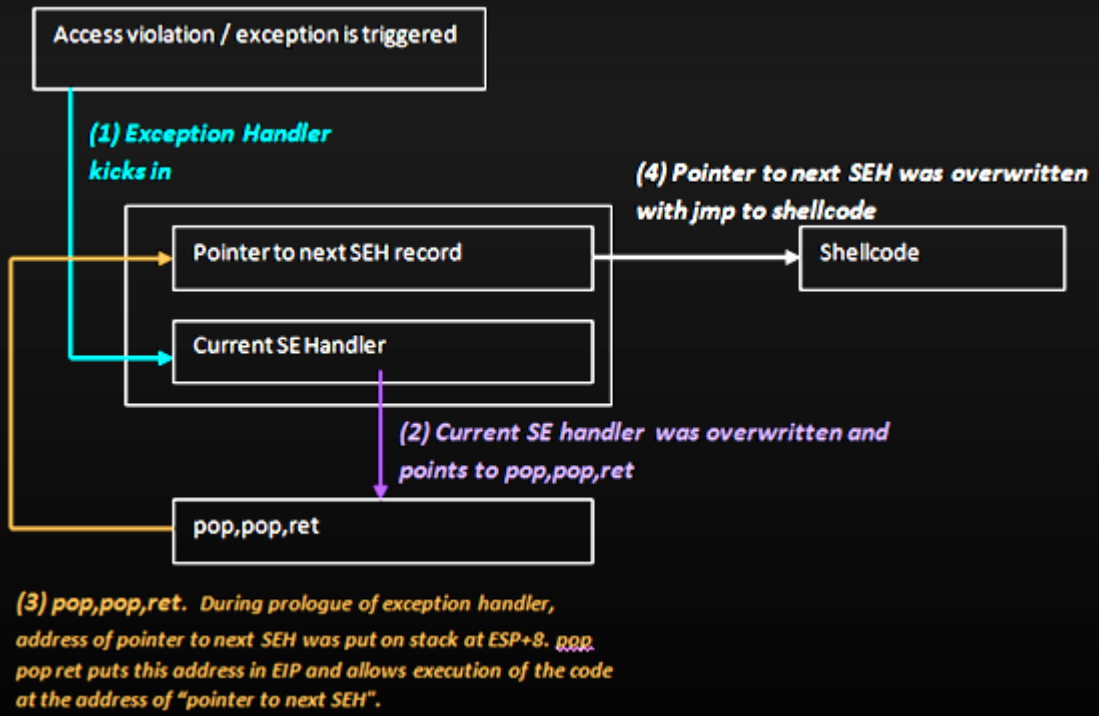


Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)
    - Method
      - Overwrite pointer to SE Handler by @pop pop ret
        Use pvefindaddr to scan all non-safeSEH modules
      - Force the application to throw an exception
      - OS will move to the next SEH
      - Overwrite pointer to nextSEH with a small jump to the shellcode

Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)



Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)
    - SafeSEH
      - Introduced in Windows XP SP3, Server 2003
      - Compiler switch (/safeSEH) for all executable modules
      - IMAGE_DLLCHARACTERISTICS_NO_SEH flag
      - SEH exploitation need to rewrite next SEH -> break the chain
      - Prevent SEH based exploitation by checking pointer range and registered exception handler addresses

SECUR IMAG
The security team of Ensimag

Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)
    - SafeSEH bypass
      - Use addresses in non-safeSEH module
        - !Pvefindaddr –j –n
        - !Pvefindaddr modules
        - OllySSEH
      - Use instruction out of the scope for verification chain
        - !Pvefindaddr –jseh
      - Use an address from the heap

Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)
    - Demonstration on sehMe program

Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)
    - SEHOP (SEH Overwrite Protection)
      - Check the exception handler chain
        - Chain must be never corrupted
        - New final handler must be correct
          (after default (kernel32!_except_handler*))
      - Windows Server 2008 (default enabled)
      - >Vista Sp1 (default disable)

SECUR IMAG
The security team of Ensimag

Security Mechanisms

- Cookie Bypass (4)
  - Use SEH (Structured Exception Handler)
    - SEHOP Bypass
      - Create fake exception handler chain
      - Use valid addresses on stack
      - But SEHOP is often used with DEP + ASLR
      - Need 

A Crash Course on the Depths of Win32™ Structured Exception Handling
http://www.microsoft.com/msj/0197/exception/exception.aspx

Security Mechanisms

- DEP (Data Prevention Execution)

Data Execution Prevention

http://msdn.microsoft.com/en-us/library/aa366553%28v=vs.85%29.aspx

Security Mechanisms

- DEP Hardware
  - Possible rights: Read/Write for Ring3/Ring0
    - No execution flag?
  - NX (Non eXecutable)/XD(eXecutable Disable) bit
    - Introduced in Windows XP SP2 and Windows Server 2003 SP1
    - Need compatible processor
    - Vmware >4.0
    - Use the 64th bit of the page table
      - Need Physical Address Extension

SECUR IMAG
The security team of Ensimag

Security Mechanisms

- DEP Hardware
  - PAE is loaded automatically (Windows)
  - Permanent DEP
    - Use SetProcessDEPPolicy(PROCESS_DEP_ENABLE)
    - Since Vista, Permanent DEP is set automatically for /NXCOMPACT linked binary
  - Basic exploitation with shellcode in stack
    - Doesn’t work anymore
    - Raised an CPU exception caught by DEP
      - STATUS_ACCESS_VIOLATION (0xc0000005)

Security Mechanisms

- DEP Software
  - Windows only
  - Limited version for incompatible CPUs
  - Is it really a DEP ?
    - NO !
    - Memory page still be executable
    - DEP Software is only safeSEH

SECUR IMAG
The security team of Ensimag

Security Mechanisms

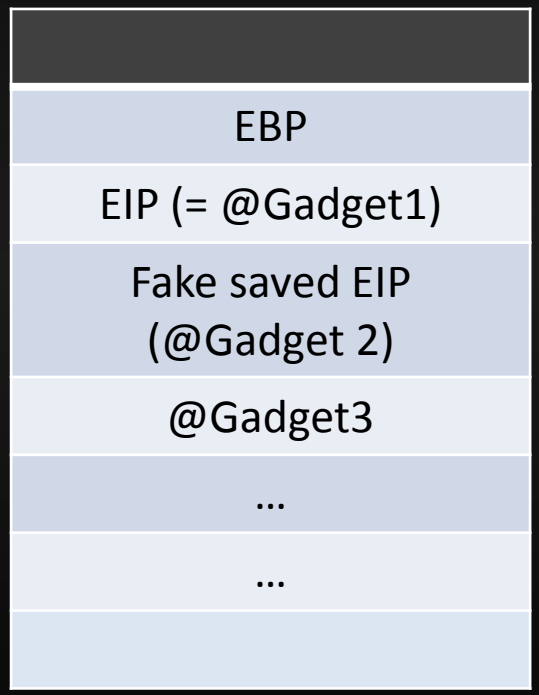
- Bypass DEP: ROP-FU (1)
  - Return Oriented Programming
  - Use pieces of Asm code from loaded libraries
    - Gadgets
    - Play with stack and RET instruction to assemble your code
      - Lego®
  - Allow to build your own payload
    - Doesn't require code on non-executable pages
  - Turing-complete language
    - Code whatever you want! (or not)

Chaining DEP with ROP – the Rubik's[TM]

http://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/

Security Mechanisms

- Bypass DEP: ROP-FU (2)
  - Write EIP with first gadget
    - Gadget1 does something…
      - Can eventually modify stack -_-
      - Last instruction: RET
    - RET pops fake saved EIP
      - Goes on second gadget
    - And so on…

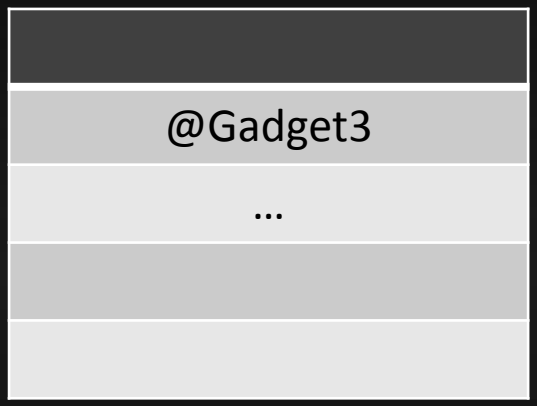


Security Mechanisms

- Bypass DEP: ROP-FU (2)
  - Write EIP with first gadget
    - Gadget1 does something…
      - Can eventually modify stack -_-
      - Last instruction: RET
    - RET pops fake saved EIP
      - Goes on second gadget
    - And so on…

Security Mechanisms

- Bypass DEP: ROP-FU (2)
  - Write EIP with first gadget
    - Gadget1 does something…
      - Can eventually modify stack -_-
      - Last instruction: RET
    - RET pops fake saved EIP
      - Goes on second gadget
    - And so on…



Security Mechanism

- Bypass DEP: ROP-FU (3)
  - Some API allow DEP disabling
    - VirtualProtect
    - VirtualAlloc
    - HeapAlloc
    - …
  - Standard exploitation
    - Put your shellcode on stack
    - DEP disabling by ROP-FU
    - Jump onto your code

SECUR IMAG
The security team of Ensimag

Security Mechanisms

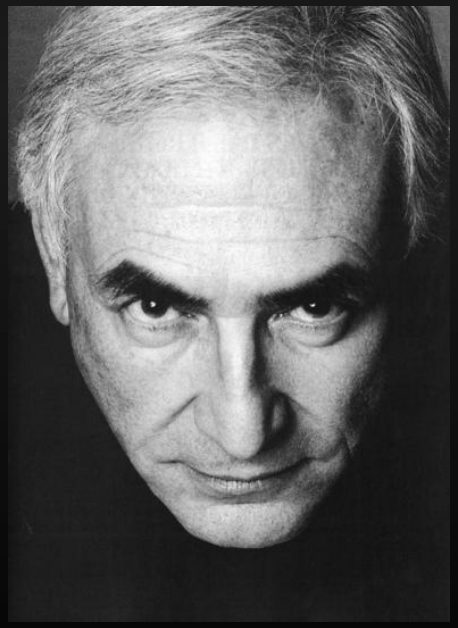
- Bypass DEP: ROP-FU (4)
  - Choose your tools
    - ImmunityDbg with **pvefindaddr**
      - **pvefinaddr**:
        - Find ROP gadgets (DEMO)
        - List modules with their properties (SafeSEH, ASLR…)
        - And many other options…
    - Write your own Python tools

Pvefinaddr

http://redmine.corelan.be:8800/projects/pvefindaddr

Security Mechanisms

- Bypass DEP: ROP-FU (5)
  - Demonstration on PwnMe program



SECUR IMAG
The security team of Ensimag

Security Mechanisms

- ASLR (Address Space Layout Randomization)
  - Randomize base addresses of
    - Executable
    - Stack (for each thread)
    - Heap (for each thread)
    - Library
  - Need Vista (jan 07), 2008 server, Seven

Security Mechanisms

- ASLR (Address Space Layout Randomization)
  - Enabled by default for system images
  - Non system images with /DYNAMICBASE (>VS2005sp1)
  - Or set DllCharacteristics to 0×40 in the PE Header
  - Registry hack possible to enable it for all images
    - HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\MoveImages = -1
  - ASLR should be used with DEP in order to be effective

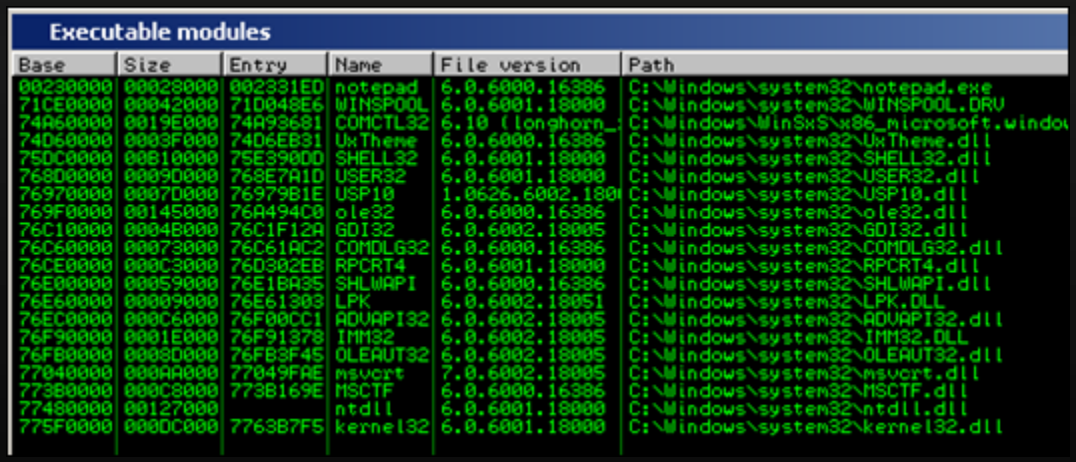
Security Mechanisms

- ASLR (Address Space Layout Randomization)
  - Bypass : Partial EIP Overwrite
    - Well known for Animated Cursor Handling Vulnerability
      - Bypass /GS too, use structures -> no cookie ☺
    - Principe
      - Only the high order bytes are randomized (0xFFFF0000)
      - If we can find instruction (e.g. jmp esp) in the scope
        - -> \o/

SECUR IMAG
The security team of Ensimag

Security Mechanisms

- ASLR (Address Space Layout Randomization)
  - Bypass : Partial EIP Overwrite

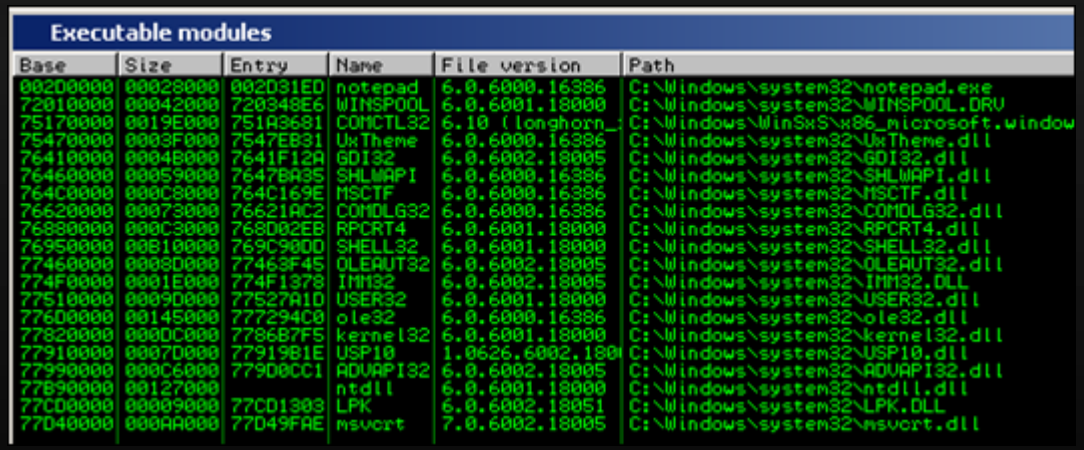


Executable modules

Base	Size	Entry	Name	File version	Path
00230000	00028000	002331ED	notepad	6.0.6000.16386	C:\Windows\system32\notepad.exe
71CE0000	00042000	71D048E6	WINSPOOL	6.0.6001.18000	C:\Windows\system32\WINSPOOL.DRV
74A60000	0019E000	74A93681	COMCTL32	6.10 (longhorn_	C:\Windows\WinSxS\x86_microsoft.windo
74D60000	0003F000	74D6EB31	UxTheme	6.0.6000.16386	C:\Windows\system32\UxTheme.dll
75DC0000	00B10000	75E390DD	SHELL32	6.0.6001.18000	C:\Windows\system32\SHELL32.dll
768D0000	0009D000	768E7A1D	USER32	6.0.6001.18000	C:\Windows\system32\USER32.dll
76970000	0007D000	76979B1E	USP10	1.0626.6002.180	C:\Windows\system32\USP10.dll
769F0000	00145000	76A494C0	ole32	6.0.6000.16386	C:\Windows\system32\ole32.dll
76C10000	0004B000	76C1F12A	GDI32	6.0.6002.18005	C:\Windows\system32\GDI32.dll
76C60000	00073000	76C61AC2	COMDLG32	6.0.6000.16386	C:\Windows\system32\COMDLG32.dll
76CE0000	000C3000	76D302EB	RPCRT4	6.0.6001.18000	C:\Windows\system32\RPCRT4.dll
76E00000	00059000	76E1BA35	SHLWAPI	6.0.6000.16386	C:\Windows\system32\SHLWAPI.dll
76E60000	00009000	76E61303	LPK	6.0.6002.18051	C:\Windows\system32\LPK.DLL
76EC0000	000C6000	76F00CC1	ADVAPI32	6.0.6002.18005	C:\Windows\system32\ADVAPI32.dll
76F90000	0001E000	76F91378	IMM32	6.0.6002.18005	C:\Windows\system32\IMM32.DLL
76FB0000	0008D000	76FB3F45	OLEAUT32	6.0.6002.18005	C:\Windows\system32\OLEAUT32.dll
77040000	000AA000	77049FAE	msvcrt	7.0.6002.18005	C:\Windows\system32\msvcrt.dll
773B0000	000C8000	773B169E	MSCTF	6.0.6000.16386	C:\Windows\system32\MSCTF.dll
77480000	00127000		ntdll	6.0.6001.18000	C:\Windows\system32\ntdll.dll
775F0000	000DC000	7763B7F5	kernel32	6.0.6001.18000	C:\Windows\system32\kernel32.dll

Security Mechanisms

- ASLR (Address Space Layout Randomization)
  - Bypass : Partial EIP Overwrite



Executable modules

Base	Size	Entry	Name	File version	Path
002D0000	00028000	002D31ED	notepad	6.0.6000.16386	C:\Windows\system32\notepad.exe
72010000	00042000	720348E6	WINSPOOL	6.0.6001.18000	C:\Windows\system32\WINSPOOL.DRV
75170000	0019E000	751A3681	COMCTL32	6.10 (longhorn_	C:\Windows\WinSxS\x86_microsoft.window
75470000	0003F000	7547EB31	UxTheme	6.0.6000.16386	C:\Windows\system32\UxTheme.dll
76410000	0004B000	7641F12A	GDI32	6.0.6002.18005	C:\Windows\system32\GDI32.dll
76460000	00059000	7647BA35	SHLWAPI	6.0.6000.16386	C:\Windows\system32\SHLWAPI.dll
764C0000	000C8000	764C169E	MSCTF	6.0.6000.16386	C:\Windows\system32\MSCTF.dll
76620000	00073000	76621AC2	COMDLG32	6.0.6000.16386	C:\Windows\system32\COMDLG32.dll
76880000	000C3000	768D02EB	RPCRT4	6.0.6001.18000	C:\Windows\system32\RPCRT4.dll
76950000	00B10000	769C90DD	SHELL32	6.0.6001.18000	C:\Windows\system32\SHELL32.dll
77460000	0008D000	77463F45	OLEAUT32	6.0.6002.18005	C:\Windows\system32\OLEAUT32.dll
774F0000	0001E000	774F1378	IMM32	6.0.6002.18005	C:\Windows\system32\IMM32.DLL
77510000	0009D000	77527A1D	USER32	6.0.6001.18000	C:\Windows\system32\USER32.dll
776D0000	00145000	777294C0	ole32	6.0.6000.16386	C:\Windows\system32\ole32.dll
77820000	000DC000	7786B7F5	kernel32	6.0.6001.18000	C:\Windows\system32\kernel32.dll
77910000	0007D000	77919B1E	USP10	1.0626.6002.180	C:\Windows\system32\USP10.dll
77990000	000C6000	779D0CC1	ADVAPI32	6.0.6002.18005	C:\Windows\system32\ADVAPI32.dll
77B90000	00127000		ntdll	6.0.6001.18000	C:\Windows\system32\ntdll.dll
77CD0000	00009000	77CD1303	LPK	6.0.6002.18051	C:\Windows\system32\LPK.DLL
77D40000	000AA000	77D49FAE	msvcrt	7.0.6002.18005	C:\Windows\system32\msvcrt.dll

Security Mechanisms

- ASLR (Address Space Layout Randomization)
  - Bypass : Use an non-ASLR enabled module
    - Similar to safeSEH bypass method
    - !pvefindaddr noaslr/modules

```
** [+] Gathering executable / loaded module info, please wait...
** [+] Finished task, 4 modules found
------------------------------------------------------------------------------------------------------------------------
 Loaded modules
------------------------------------------------------------------------------------------------------------------------
  Fixup  |    Base     |     Top     |    Size     | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
------------------------------------------------------------------------------------------------------------------------
   NO    | 0x00400000 | 0x00408000 | 0x00008000 |   NO    |  NO   |    NO    |   NO   | -1.0- - Main.exe
   NO    | 0x7C800000 | 0x7C906000 | 0x00106000 |   yes   |  NO   |    NO    |   yes  | 5.1.2600.5781 - kernel32.dll
   NO    | 0x77BE0000 | 0x77C38000 | 0x00058000 |   yes   |  NO   |    NO    |   yes  | 7.0.2600.5512 - msvcrt.dll
   NO    | 0x7C910000 | 0x7C9C9000 | 0x000B9000 |   yes   |  NO   |    NO    |   yes  | 5.1.2600.6055 - ntdll.dll
------------------------------------------------------------------------------------------------------------------------
Return Value must be a string
```

Security Mechanisms

- ASLR (Address Space Layout Randomization)
  - Bypass : Bruteforce + nop slide
    - On 32 bits architecture
      - Windows : 16 random bits
      - Linux : 24 random bits
    - For a 4096 bytes buffer, the chance is about one to 2^24 / 4096 = 4096 to hit a working address
    - Require only 2048 attempts on average

SECUR IMAG
The security team of Ensimag

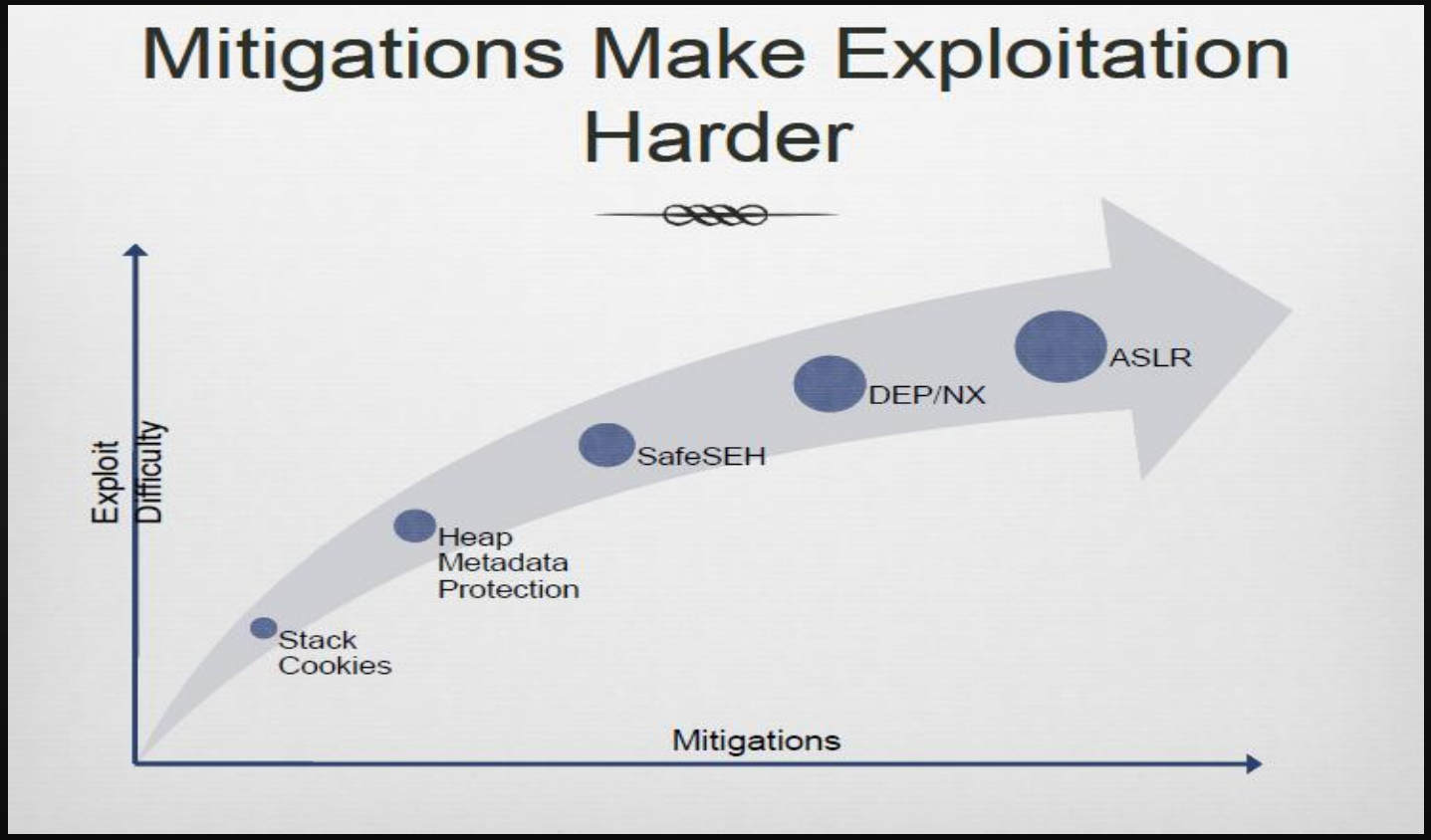
Security Mechanisms

- ASLR (Address Space Layout Randomization)
  - Demonstration on aslrMe program

Quote

“Little and insignificant issues can lead to find more interesting issues.” Cesar Cerrudo (BHus10)

Conclusion


Mitigations Make Exploitation Harder
Exploit Difficulty
Stack Cookies
Heap Metadata Protection
SafeSEH
DEP/NX
ASLR
Mitigations

Casting

- Immunity Debugger (ImmunityInc)
- Pvefindaddr (Corelan)
- PEiD (www.peid.info)

- Windows XP SP3 ☺
- Macbook and Dell

If you like Windows or not…

- Some references:
  - Ivanlef0u's blog
  - ReactOS project
    - Reimplementation of NT kernel
    - C source code available
      - Very near of Windows code
      - Nice to understand some stuffs
  - And blow up your mind

Nuit du Hack 2011

SECUR IMAG
The security team of Ensimag

Some links ☺

- Corelan, www.corelan.be
- Exploit database, www.exploit-db.com
- Windows Internals 5th, Microsoft learning
- The Portable Executable, MSDN
- Smashing the stack for fun and profit, Phrack.org

SECUR IMAG
The security team of Ensimag

Questions ?