Exploiting Software Vulnerabilities Software Vulnerabilities CONTROL-FLOW HIJACKING

③ All wrongs reversed – under CC-BY-NC-SA 4.0 license



Dept. of Computer Science and Systems Engineering University of Zaragoza, Spain

Course 2021/2022

Master's Degree in Informatics Engineering

UNIVERSITY OF ZARAGOZA Seminar A.25, Ada Byron building



Outline

1 A little recap

2 Buffer Overflows

3 Defenses against Control-Flow Hijacking Attacks

- Stack Data Protection
- Non-Executable Stack
- Write XOR eXecute (W^AX) Pages
- Address Space Layout
- Other Techniques of Defense



Outline

1 A little recap

- Buffer Overflows
- 3 Defenses against Control-Flow Hijacking Attacks



Recap on... Definitions

System/defender perspective

- Attack surface
 - Exposure of a system to attacks
- Vulnerability
 - Software flaw that can be exploited by an attacker

Attacker perspective

- Attack vector
 - How the attack was carried out
- Exploit
 - Succeed by taking advantage of a vulnerability



Recap on...

Vulnerabilities

Types of software vulnerabilities

Overflow

. . .

- Buffer overflow
- Heap overflow
- NULL pointer dereference
- Dynamic memory handling
 - Use-after-free
 - Double free
 - Allocator abuse

- Number handling
- Format strings
- Uninitialized memory
- Race conditions

Vulnerability databases

- National Vulnerability Database (NVD), maintained by NIST (https://nvd.nist.gov/)
- MITRE CVE (https://cve.mitre.org/)
- Bugtraq (http://www.securityfocus.com/archive/1)



Today we talk about...

Control-Flow Hijacking

- Attacker's goal: to seize the target system
 - Run arbitrary code to hijack the control flow of a vulnerable application



Block of instructions that performs a specific task

Three components:

- Input (values passed from the caller)
- Body (code to perform the task)
- Return value (to the caller)

Calling a function involves a branch in the control flow (i.e., jumping to another location)

The return address is usually stored in the caller's stack frame



int x = compute(arg0, arg1, ...)

What happens in the backstage before a function runs?

Parameters are configured to be passed to the function

- Either through the stack or logical registers
- The address of the next instruction after the call is also saved



int x = compute(arg0, arg1, ...)

What happens in the backstage before a function runs?

Parameters are configured to be passed to the function

- Either through the stack or logical registers
- The address of the next instruction after the call is also saved

What happens in the backstage after a function runs?

Return value is set

- Normally, the logical register eaxcontains the return value of a function
- Allocated variables (within the function) are removed from the stack
- Registers used in the function are restored to their previous values
- The control is transferred to the saved return address



Standard prologue

- Occurs at the beginning of a function
- Allocates space for local variables (on stack)
- Saves registers to be reused in the body of the function

Standard epilogue

- Occurs at the end of a function
- Normally, undoes what was done in the prologue
- Cleans up the stack
- Restores register values



mov edi, edi

Common prologue on Windows

- 2-byte length instruction
- Equivalent for a nop instruction, since it does nothing
- Used to hot-patch a running executable, without stopping and restarting it
 - Can be overwritten with a relative jump of 2 bytes!

Further reading: Why do Windows functions all begin with a pointless MOV EDI, EDI instruction?. R. Chen, 2011.

https://devblogs.microsoft.com/oldnewthing/?p=9583

Calling conventions

- Describes how data is passed in/out of functions
- Implementation may vary by compiler



Calling conventions

- Describes how data is passed in/out of functions
- Implementation may vary by compiler

cdecl convention (most common)

- Arguments are pushed onto the stack from right to left
- Return value is placed in eax
- The caller must clean the stack (removing passed parameters)



stdcall convention

- Similar to cdec1, but *callee* clears the stack
- Convention used in Windows APIs



stdcall convention

- Similar to cdec1, but callee clears the stack
- Convention used in Windows APIs

fastcall convention

- Arguments are passed by registers, put on the stack when a large number of arguments are required
 - For instance, the GCC and Microsoft compilers use the ecx and edx registers
- The callee clears the stack
- Return value is placed in eax



Universidad

thiscall convention

- Used in C++ in object methods (member functions)
- Includes a reference to the this pointer
- Depends on the compiler:
 - In Microsoft compilers, ecx holds the this pointer and the callee clears the stack
 - In GNU compilers, the this pointer is pushed last and the *caller* clears the stack



```
#include <iostream>
using namespace std:
class Student {
    public:
        int id; //data member (also instance variable)
        string name; //data member (also instance variable)
    void imprime(){
        cout << this -> id << endl;
        cout << this -> name << endl;
    }
};
_cdecl int echo(int x){
    return x + 8;
}
int main() {
    Student s1; //creating an object of Student
    s1.id = 201;
    s1.name = "Sonoo Jaiswal";
    s1.imprime();
    printf("echo: %d\n", echo(4));
    return 0;
}
```

Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]

TTT Universidad

🛄 Zaradoza

00404405									
00401425 00401426	. 55		PUSH EBP MOV EBP,ESI						
00401426	. <u>89</u> E5		MUV EBP, ESI	-					
00401428	. 53		PUSH EBX						
00401429 0040142A	. 51		PUSH ECX	~~					
0040142H	. <u>83</u> EC	30	SUB ESP,0x:	30					
0040142D	. E8 3	3E060000	CALL a.004	<u>81H76</u>					
00401432	. 8D45	, DC	LEH EHX,UW	JRD PIR	SS:[EBP-0x24]				
00401435	. 8901		MOV ECX, EA	K					
00401437	. E8 1	10290000	CALL a.004	33D4C					
0040143C	. C745	5 DC C900 5 DC	MOV DWORD	PTR SS:	[EBP-0x24],0xC9				
00401443	. 8045	DC	LEA EAX,DW	ORD PIR	SS:[EBP-0x24]				
00401446	. 8300	04	ADD EAX.0x	4					
00401449 00401450	. C704	424 45504	MOV DWORD B	PTR SS:	[ESP],a.00405045		ASCII "Sonoo	Jaiswal"	
00401450	. <u>89C1</u>		MOV ECX,EA	K.					
00401452			CALL KUMP.	<u>klibstd</u>	c++-6ZNSt7cxx11	112b.			
00401457	. 83EC	; 04	SUB ESP,0x	4					
0040145A	. 8D45	DC I	LEA EAX,DW	ORD PTR	SS:[EBP-0x24]				
0040145D	. 89C1	1	MOV ECX,EA	X					
0040145F	. E8_8	3C280000	CALL a.0040	33CF0					
00401464		424 04000	MOV DWORD N	PTR SS:	[ESP],0x4				
0040146B	. E8 A	RØFFFFFF	CALL a.004	31410					
00401470	. 8944	424 04	MOV DWORD	PTR SS:	[ESP+0x4],EAX [ESP],a.00405053				
00401474	. C704	424 53504	MOV DWORD	PTR SS:	[ESP],a.00405053		ASCII "echo:	%d ⊡ ‴	
0040147B	. E8 5	50270000	CALL KUMP.	&msucrt	.printf>	_	printf		
00401480	. BB 0	00000000	MOV EBX,0x	3	SS:[EBP-0x24]				
00401485	. 8D45	S DC	LEA EAX,DW	ORD PTR	SS:[EBP-0x24]				
00401488	. 89C1	1	MOV ECX,EA	X					
0040148A		09280000	CALL a.004	03D68					
0040148F	. 89D8	1	MOV EAX,EB	×					
00401491 00401493 00401495	.~ EB 1	16	JMP SHORT (a.00401	489				
00401493	. 8903		MOV EBX,EA	×					
00401495	. 8D45	5 DC	LEA EAX.DW	ORD PTR	SS:[EBP-0x24]				
00401498	. 8901	1	MOU ECX.EA	X					
0040149A		9280000	CALL a.004	23D68					
0040149F	. 89D8		CALL a.004 MOV EAX,EB	×					
004014A1	. 8904	424	MOV DWORD I	PTR SS:	[ESP].EAX				
004014A4					s_dw2-1Unwind_Re	esum			
004014A9	> 8065	5 F8	LEA ESP. DW	ORD PTR	SS:[EBP-0x8]				
004014AC	. 59		POP ECX						
004014AD	. 5B		POP EBX						
004014AE	. 5D		POP EBP						
004014AF	. 8D61	1 FC	LEA ESP.DW	ORD PTR	DS:[ECX-0x4]				iversidad
004014B2	. č3		RETN						agoza
2-0-00-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-0-								1141	ugozu

004413331 7 83 cs 100 cs <th>00401330 ></th> <th>55</th> <th>PUSH EBP</th> <th></th>	00401330 >	55	PUSH EBP	
00401333 . 56 PUSH ESI 00401334 . 58 PUSH ESI 00401335 . 835C 10 SUE ESP, 0x10 00401335 . 68 4C290000 CTUL CUP skterie 00401336 . 835C 10 C70424 06544 CPUSH ESI 00401337 . 835C 04 SUE ESP, 0x10 CPUSH ESI 00401347 . 835C 04 C70424 06544 TEST ENV, ERX 00401356 . 656 - 200000 CPUL CUP skterie CPUSH ESI 00401356 . 656 - 200000 CPUL CUP skterie CPUSH ESI 00401356 . 656 - 200000 CPUL CUP skterie CPUSH ESI 00401356 . 656 - 200000 CPUL CUP skterie CPUSH ESI 00401356 . 67424 04 131 HOU DUORD PTR S5: EESP+0x41, a. 00405001 LoadLibraryA 00401376 . 897C 4 MOU DUORD PTR S5: EESP+0x41, a. 00405002 RSCII "LibraryA 00401376 . 897C 44 PUSH ESI ESP, ESP, ESX PUSH ESI RSCII "LibraryA 00401376 . 897C 44 PUSH ESP, 0x3 . 60405002 RSCII "LibraryA 00401377 . 897C 68 PUSH ESP, 0x	00401330 /			
00401334 . 53 PUSH EBX 00401335 . 83EC 10 SUE ESP, 0:10 00401335 . C70424 00504 HOU DWORD PTR 53: [ESP], a. 00405000 00401335 . C70424 00504 SUE ESP, 0:10 00401335 . S2C 04 SUE ESP, 0:44 00401344 . S3C 04 SUE ESP, 0:44 00401344 . S3C 04 TEX. FEAN.ERX 00401347 . S4C0 TEX.EX.EX.EX.EX.EX.EX.EX.EX.EX.EX.EX.EX.E	00401331 .	SAFP	MUV EBP, ESP	
00401335 . 83EC 10 SUE ESP, 0x10 00401335 . C70424 00504 OLL < CUP 20XERNIE3220ceticol [Filmolich]		56		
00401338 . C70424 00504 HOU DUORD PTR S5:EESP1, a.00405000		53		
0040133F E.8 4C290000 04LL < CHP 304ERNL522.GetHoduleHandleN			SUB ESP,0x10	
0040133F E.8 4C290000 04LL < CHP 304ERNL522.GetHoduleHandleN	00401338 .	C70424 00504	MOV DWORD PTR SS:[ESP],a.00405000	ASCII "libgcc_s_dw2-1.dll"
00401344	0040133F .	E8 4C290000	CALL <jmp.&kernel32.getmodulehandlea></jmp.&kernel32.getmodulehandlea>	GetModuleHandleA
00401347 : 85C0 TEST ERX;EXX 00401348 : C70424 00504 FORT a.004013C0 00401348 : C70424 00504 FORT a.004013C0 00401348 : C70424 00504 FORT a.004013C0 00401354 : B17290000 CPLL < CHP EXCERNELS21 CostIl branyth	00401344	83EC 04	SUB ESP.0x4	
00401349	00401347	8500	TEST FAX.FAX	
00401348 . C70424 00504 HOU DUORD PTR SS:TESP], a.00405000 ASCII "'Libgoc_s_dw2-1.dll" 00401352 . 89C3 . 89C4 OU DUORD PTR SS:TESP], a.00405000 ASCII ''Libgoc_s_dw2-1.dll" 00401354 . 88 7290000 . 0040704080 OU DUORD PTR SS:TESP, a.00405003 ASCII ''Libgoc_s_dw2-1.dll" 00401355 . 83 70704080 . 000 DUORD PTR SS:TESP, a.00405013 ASCII ''Libgoc_s_dw2-1.dll" 00401355 . 83 70704080 . 000 DUORD PTR SS:TESP, A.00405013 ASCII ''_register_frame_info" 00401356 . 831C24 . 001 DUORD PTR SS:TESP, A.00405013 ASCII ''_register_frame_info" 00401357 . 890C6 . 891C24 . 001 DUORD PTR SS:TESP, A.00405029 ASCII ''_register_frame_info" 00401374 . 890C6 . 691C24 . 691C24 . 691C24 . 691C24 00401386 . 63 0404000 . 691C24 . 691C24 . 691C24 . 691C24 00401386 . 83EC 693 . 691C24 . 691C24 . 691C24 . 691C24 00401386 . 684C290000 . 7412 . 68 HOU DUORD PTR SS:TESP, a.00407008 . 68CII ''_deregister_frame_info'' 00401386 . 684C1992 . 7412 . 68 HOU DUORD PTR SS:TESP, a.0040		74 75	JE SHORT = 00401300	
00401352 . 89C3 HOU EEX,ERX HOU EEX,ERX LoadLibraryA 00401354 . 83720000 SUE ESP,0x4 SUE ESP,0x4 SUE ESP,0x4 00401354 . 837204000 OUDORD PTR SS:EESP+0x41,a.00405013 AST 20704000 AST 20704000 00401355 . 837204000 OUDORD PTR SS:EESP+0x41,a.00405013 AST 20704000 CHL AST 20704000 00401354 . 674424 04131100 DUORD PTR SS:EESP+0x41,a.00405013 AST 20704000 AST 20704000 00401374 . 8905 CHL CHL CHL AST 20704000 AST 20704000 00401374 . 8905 CTLL CHL AST 20704000 AST 20704000 AST 20704000 004041374 . 8905 CTLL CHL AST 20704000 AST 20704000 AST 20704000 004041374 . 8905 CTL CHL CHL AST 20704000 AST 20704000 AST 20704000 004041385 . 832EC 008 DU DUORD PTR DS:EESP1,ex0 AST 20704000 AST 20704000 AST 20704000 AST 2070400 AST 2070400 AST 2070400 AST 2070400 AST 20704000 AST 207040000 AST 2070400 AST 20704	0040134B		MOLL DWORD PTR SS+[ESP1 > 00405000	ASCII "Libace & dw2-1 dll"
00401354 . E8 17290000 06LL CUTP.skERNE322LoadLibrary0A 00401350 . 83EC 64 SE 59, 044 00401354 . 83 70704000 TOD DuORD PTR Ds: 10x407070].EAX 00401355 . 83 70704000 TOD DuORD PTR Ds: 10x407070].EAX 00401356 . 83 70704000 TOD DuORD PTR Ss: 1ESP1, EAX 00401356 . 83 70704000 . 83 70704000 00401356 . 83 70704000 . 83 70704000 00401356 . 83 70704000 . 83 70704000 00401356 . 83 70704000 . 83 70704000 00401356 . 83 70704000 . 83 70704000 00401357 . 83 60 00 . 83 80000 00401351 . 83 901624 . 42 90000 00401351 . 83 901624 . 62 90000 00401351 . 83 80404000 . 74 11 00401382 . 83 80404000 . 83 80404000 00401382 . 83 80404000 . 83 80404000 00401382 . 64 74 8018 . 83 80404000 00401382 . 74 14 . 8000000 PTR S5: 165 P+ 0x41, a. 00407008 00401381 . C74424 44 88 100 U00000 PTR S5: 165 P+ 0x41, a. 00407008 00401381	00401050	0000	MOLIEDV EOV	Hoorr rrbgoo_b_dwz rrarr
004013559 .838C 04 SUB ESP,0:44 0040135C .83704000 MOU DUORD PTR SS:ESP+0x4],a.00405013 0040135C .627424 04 131 MOU DUORD PTR SS:ESP+0x4],a.00405013 0040136C .891C24 0040137C .838C 04 0040137C .839C 05 0040137C .839C 05 0040137C .839C 05 0040137C .839C 05 0040137C .839C 04 0040137C .839C 05 0040137C .839C 04 0040137C .839C 04 0040137C .627424 04 29 HOU DUORD PTR SS:ESP).EBX 0040137C .839C 04 0040137C .839C 04 0040137C .839C 04 0040137C .839C 04 0040138C .62290000 0040138C .63 0249000 00404138C .836C 08 0040138C .74 114 04 08 0040138C .774 124 88694 HOU DUORD PTR SS:ESP1, a.004070088 0040138C .774 124 88694 HOU DUORD PTR SS:ESP1, a.00407008 0040138C .774 24 88694 HOU DUORD PTR SS:ESP1, a.0040138C 0040138C .774 124 88684 HOU DUORD PTR SS			COLL / IMP #KEPNEL22 Land ibanaw0	Lond, ibnow 0
0040135C . 63 70704000 HOU DUNDRD PTR 05:1084070701,EEX Ascil "register_frame_info" 00401361 . 674424 04 13 HOU DUNDRD PTR 55:10594407013,0040405013 HOU DUNDRD PTR 55:10594404013 Ascil "register_frame_info" 00401361 . 831C24 . 831C24 SUB ESP, 0x8 SUB ESP, 0x8 00401361 . 834C 08 . 834C 08 SUB ESP, 0x8 . 834C 08 . 834C 08 00401371 . 834C 08 . 834C 08 . 900 DUNDR PTR 35:10591 CONCORPORTS . 8460 CONCORPORTS . 8460 CONCORPORTS 00401374 . 834C 08 . 834C 08 . 900 DUNDR PTR 35:10591 ESX . 00405029 Ascil "deregister_frame_info" 00401374 . 834C 08 . 83404000 . 834014000 PTR 05:10x4040001, EDX . 004005029 Ascil **_deregister_frame_info" 00401381 . 836C 08 . 836C 08 <t< td=""><td></td><td></td><td></td><td>a coaucioraryn</td></t<>				a coaucioraryn
00401361 C74424 04131 MOU DUORD PTR SS:EESP+0x4],a.00495013 ASCII "register_frame_info" 00401362 E8 17290000 OGLCUP.uKERNEL32.GetProcNddress ASCII "register_frame_info" 00401374 890C6 OGLCUP.uKERNEL32.GetProcNddress ASCII "register_frame_info" 00401374 892C6 OU DUO BUORD PTR SS:EESP1,EEX ASCII "register_frame_info" 00401374 892C6 MOU ESI,EAX DUO BUORD PTR SS:EESP1,EEX ASCII "dregister_frame_info" 00401371 E8 02290000 OTH SS:EESP1,EEX ASCII "dregister_frame_info" 004041376 E8 02290000 HOU DUORD PTR SS:EESP1,EEX ASCII "dregister_frame_info" 00401376 E8 02290000 HOU DUORD PTR SS:EESP1,eEX ASCII ''_dregister_frame_info" 004041386 AS 00404000 FTR SS:EESP1,eEX ASCII ''_dregister_frame_info" 00401386 C74424 BES FLAN TEST ESI,ESI AS0404000 00401386 C74424 BES FLAN FTST ESI,ESI AS0407008 00401386 C774424 BESH HOU DUORD PTR SS:EESP1,a.004070088 Asta 00401387 C774424 BESH HU CUP FLANSURGARANCALL AS04047088 Asta	00401057	09 20204000	MOU DWORD PTP DC.[9.4979791 EOV	1
08441369 .891C24 TOULDUORD PTR SS:EESPJ.EEX GetProcRddress 08441362 .E817290000 SUE SP,048 SUE SP,048 09401371 .836C 08 SUE SP,048 SUE SP,048 09401374 .836C 08 SUE SP,048 SUE SP,048 09401371 .836C 08 SUE SP,048 SUE SP,048 09401374 .834C 08 SUE SP,048 SUE SP,048 09401375 .536 2290000 TOU DUODD PTR SS:ESP1.EX GetProcRddress 09401386 .836C 08 SUE SP,048 SUE SP,048 GetProcRddress 09401387 .C74424 04 08 SUO DUODD PTR SS:ESP1.a.00407008 GetProcRddress 09401381 .C74424 24 88604 NOU DUODD PTR SS:ESP1.a.00407008 GetProcRddress 09401381 .C74424 88604 SUO DUODD PTR SS:ESP1.a.00407008 GetProcRddress 09401381 .C74424 88604 SUO DUODD PTR SS:ESP1.a.0040808 GetProcRddress 09401381 .C7442		H3 70704000	MOU DWORD FIR DS:L0X4070701,EHA	COOTE No and international in Coll
0040136C . E8 17290000 06LL CUTP.tkERNE.32.GetProcAddress GetProcAddress 00401371 . 89C6	00401361 .		MOV DWORD FIR SS:LESP+084J, a.00405013	HSCII "
Ud441371 .83LC U8 SUB ES7, 048 004401374 .83LC U8 .83LC U8 <td< td=""><td></td><td>891024</td><td>MUV DWURD FIR SSILESFI,EBX</td><td></td></td<>		891024	MUV DWURD FIR SSILESFI,EBX	
00401376 . C74424 04 291 f00 DU0RD PTR SS: EESP+0x4],a.00405029 ASCII "deregister_frame_info" 00401376 . 891C24 GOU DU0RD PTR SS: EESP+0x4],a.00405029 BSCII "deregister_frame_info" 00401381 . 88 02290000 GALL GALL SUP skERNE1323.00170001,ERX BSCII "deregister_frame_info" 00401386 . 830C 08 SUB ESP,0x8 SUB ESP,0x8 SUB ESP,0x8 00401386 . 830C 08 SUB ESP,0x8 SUB ESP,0x8 SUB ESP,0x8 00401382 . C74424 04 08 1000 DU0RD PTR SS: [ESP+0x4],a.00407008 Hoto Du0RD PTR SS: [ESP+0x4],a.00407008 Hoto Du0RD PTR SS: [ESP+0x4],a.00401380 00401383 . C70424 E0134H 100 DU0RD PTR SS: [ESP+0x8],a.00401380 Hoto Du0RD PTR SS: [ESP-0x8] SUB ESP,000RD PTR SS: [ESP-0x8] 00401384 . ES 61FFFFF GALL SUP OP ESS ER ESP,000RD PTR SS: [EBP-0x8] SUB ESP,000RD PTR SS: [EBP-0x8] 00401384 . SD65 FS POP ESI ER ESP,000RD PTR SS: [EBP-0x8] SUB ESP,000RD PTR SS: [EBP-0x8] 00401385 . SE POP ESI EP ESP EP ESP SUB ESP		E8_17590000	CHEL KUMP.&KERNEL32.GetProchddress/	GetProcHddress
00401376 . C74424 04 291 f00 DU0RD PTR SS: EESP+0x4],a.00405029 ASCII "deregister_frame_info" 00401376 . 891C24 GOU DU0RD PTR SS: EESP+0x4],a.00405029 BSCII "deregister_frame_info" 00401381 . 88 02290000 GALL GALL SUP skERNE1323.00170001,ERX BSCII "deregister_frame_info" 00401386 . 830C 08 SUB ESP,0x8 SUB ESP,0x8 SUB ESP,0x8 00401386 . 830C 08 SUB ESP,0x8 SUB ESP,0x8 SUB ESP,0x8 00401382 . C74424 04 08 1000 DU0RD PTR SS: [ESP+0x4],a.00407008 Hoto Du0RD PTR SS: [ESP+0x4],a.00407008 Hoto Du0RD PTR SS: [ESP+0x4],a.00401380 00401383 . C70424 E0134H 100 DU0RD PTR SS: [ESP+0x8],a.00401380 Hoto Du0RD PTR SS: [ESP-0x8] SUB ESP,000RD PTR SS: [ESP-0x8] 00401384 . ES 61FFFFF GALL SUP OP ESS ER ESP,000RD PTR SS: [EBP-0x8] SUB ESP,000RD PTR SS: [EBP-0x8] 00401384 . SD65 FS POP ESI ER ESP,000RD PTR SS: [EBP-0x8] SUB ESP,000RD PTR SS: [EBP-0x8] 00401385 . SE POP ESI EP ESP EP ESP SUB ESP	00401371 .	83EC 08	SUB ESP, 0x8	
0040137E . 891/C24 0040137E . 80229000 0040138E . 832E/C84 0040138E . 74114 0040138E . 74144 0040138E . 74144 0040138E . 74424 0040138B . C70424 0040138B . C16 0040138B . S5 0040138B .	00401374 .	8906	MOV ESI,EAX	
00401381 . E8 02290000 CALL CUP .& KERNEL32.Get ProcAddress Cet ProcAddress 00401386 . A3 00440400 SUE ESP.0x8 SUE ESP.0x8 SUE ESP.0x8 00401386 . 83EC 08 SUE ESP.0x8 SUE ESP.1x8 SUE ESP.0x8 00401386 . 83EC 08	00401376 .	C74424 04 29	MOV DWORD PTR SS:[ESP+0x4],a.00405029	ASCII "deregister_frame_info"
00401386 . A3 00404000 HOU DUORD PTR DS:[0x404000],ERX 0040138E > 83EC 9 0040138E > 85EF 9 0040138E > 85EF 9 0040138E > 85EF 9 0040138E - 74124 0 0040138C - 74424 88604 HOU DUORD PTR SS:[ESP+0x4],a.00407008 00401381 - 74024 88604 HOU DUORD PTR SS:[ESP],a.00407008 00401381 - 7506 8 00401381 - 7506 F8 F0 0 00401386 - 850 F8 F8 F0 0 00401386 - 850 F8 F0 0 00401388 - 550 P0 FEX. 00401388 - 550 P0 FEX.	0040137E .	891C24	MOV DWORD PTR SS:[ESP],EBX	
00401386 . A3 00404000 HOU DUORD PTR DS:[0x404000],ERX 0040138E . 83EC 08 SUE SP,0x8 0040138E . C74124 08 SUE SP,0x8 00401392 . C74124 04 08 HOU DUORD PTR SS:[ESP+0x4],a.00407008 00401391 . C70424 E8504 HOU DUORD PTR SS:[ESP],a.00406088 00401391 . F704,24 E8504 HOU DUORD PTR SS:[ESP],a.00405088 00401391 . E764,24 E8504 HOU DUORD PTR SS:[ESP],a.004013E0 00401391 . E764,24 E8504 HOU DUORD PTR SS:[ESP],a.004013E0 00401391 . E764,25 E8 POP E8X 004013951 . S5 PPOP E8X 004013951 . S5 PPOP E8X 00401382 . S5 PPOP E8X	00401381 .	E8 02290000	CALL <jmp.&kernel32.getprocaddress></jmp.&kernel32.getprocaddress>	GetProcAddress
00401390 > 85F6 TEST ESI,ESI 00401392 . 7411 04 8 100 DUORD PTR SS:TESP10x4],a.00407008 00401394 . 70424 B8604 100 DUORD PTR SS:TESP10x4],a.00407008 00401381 . FFD6 S8604 100 DUORD PTR SS:TESP10x4],a.00407008 00401381 . FFD6 USA B8604 100 DUORD PTR SS:TESP10x4],a.00407008 00401381 . 556 FF8 FF6 CONTRACTOR SS:TESP10x4],a.00407008 00401381 . 556 FF8 FF6 CONTRACTOR SS:TESP10x4],a.00407008 00401381 . 550 FF8 FF6 CONTRACTOR SS:TESP10x4],a.00407008 00401381 . 550 FF8 FF6 FF8 FF6 FF6 FF6 FF6 FF6 FF6 FF6	00401386 .	A3 00404000	MOV DWORD PTR DS:[0x404000],EAX	
00401396 > 85F6 TEST EST.EST 00401392 · 74 10 85 F6 JEST EST.EST 00401392 · 74424 04 85 HOU DWORD PTR SS:ESP+0x4],a.00407008 00401391 · 770424 88604 HOU DWORD PTR SS:ESP+0x4],a.00407008 00401381 · 770424 88604 HOU DWORD PTR SS:ESP1a,00408088 00401381 · 556 61FFFF EST EST EST A 0040801360 00401381 · 556 61FFFFF EST EST EST A 0040801360 00401381 · 556 F78 FF EST EST EST A 004081360 00401381 · 555 F8 F0 EST A 00408 PTR SS:ESP1a,00408 PTR SS:ESP1a,00	0040138B .	83EC 08	SUB ESP,0x8	
00401390 74 11	0040138E >	85F6	TEST ESI.ESI	
00401391 C70424 B86041 HOU DUNDR PTR SS:[ESP],a.00406008 00401381 FFD6 OHLEESI 00401383 C70424 E01344 HOU DUNDR PTR SS:[ESP],a.004013E0 00401387 E861FFFFF 00401387 E865 F8 00401388 SD65 F8 00401388 SE	00401390 .~	74 11	JE SHORT a.004013A3	
00491391 . C70424 B86041 HOU DUNDR PTR SS:[ESP],a.00406088 00491381 > FFD6 00491383 > C70424 E0134 HOU DUNDR PTR SS:[ESP],a.004013E0 00491387 - 8065 F8 POP E00 PTR SS:[ESP-0x8] 00491387 - 8065 F8 POP E00 PTR SS:[EBP-0x8] 00491388 - 5E POP E00 PTR SS:[EBP-0x8] 00491388 - 5E POP E00 PTR SS:[EBP-0x8]	00401392	C74424 04 08	MOV DWORD PTR SS:[ESP+0x4].a.00407008	
00401331 . FFD6 OFIL_ESI 00401338 > C70424_E0134_HOU_DUORD PTR_SS:[ESP],a.004013E0 00401384 . E8 61FFFFFF 00401387 . E8 61FFFFF 00401381 . E8 61FFFFF 00401382 . E8 61FFFFF 00401383 . E8 61FFFFF 00401383 . E8 POP ESX 00401383 . SE 00401384 . SD 00401384 . SD		C70424 B8604	MOU DWORD PTR SS:[ESP].a.00406088	
00491383 > C70424 E0134 HOU DWORD PTR SS:[ESP],a.004013E0 00491384 - 88 61FFFFF 01L CUHP.4WSVORT.atcM [to] 00491387 - 8065 F8 POP E00 DTR SS:[EBP-0x8] 00491388 - 5E POP E51 00491388 - 5E POP E51		EED6		
004013APA . 85 61FFFFF 0ALL Culte Law supervised Cutexit 004013AF . 8065 F8 POP E8 E8 POP E8 0401383 SE POP E8 POP E8 POP E8 POP E8 POP E8 POP E8 POP E9	00401303			
0040138F . 8065 F8 LEA ESP.JWORD PTR SS:[EBP-0x8] 00401382 . 58 POP EBX 00401388 . 5E POP ESI 00401384 . 5D POP ESP	00401300	E8 61FFFFFF	COLL (IMP & msucrt, atevit)	atevit
08401382 - 58 POP EEX 08481383 - 5E POP ESI 08481384 - 50 POP EEP		8045 F8	LEG ESP DWORD PTR SS [ERP-0v81	
004013B3 . SE POP ESI 004013B4 . SD POP EBP			DAD EDV	
004013B4 . 5D POP EBP	00401000	22		
	00401004	50		
		00		
	00401385 .	000407 00000		



Pushing data on the stack: mov vs. push

- push always subtracts 4 from the esp register
- mov puts a value on the stack, but does not subtract from esp
- Optimization issue: small performance gain at runtime
 - When used with the stdcall convention, the caller must make special settings



Pushing data on the stack: mov vs. push

- push always subtracts 4 from the esp register
- mov puts a value on the stack, but does not subtract from esp
- Optimization issue: small performance gain at runtime
 - When used with the stdcall convention, the caller must make special settings

Inline functions

- Eliminate costly control transfers
- Useful for small functions, as their body is inlined with the caller's body
- No extra overhead for entry/exit
- Occurs often with string-related functions

Universidad

Outline

1 A little recap

2 Buffer Overflows

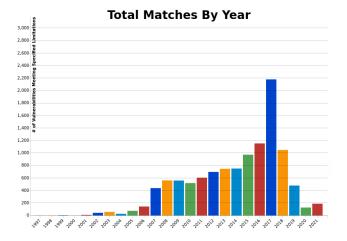
3 Defenses against Control-Flow Hijacking Attacks



Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]

2021/2022 18 / 52

Most common vulnerability in C/C++ programs



Credits: taken at 23/10/2021, https://nvd.nist.gov/

Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]



2021/2022 19 / 52

A bit of history – the first BOF exploited

- (BSD-derived) UNIX fingerd daemon
 - Utility that allows users to obtain information about other users
 - Usually used to identify the full name or login name of a user, whether a user is currently logged in or not, and other user information

■ Morris worm (November 2 1988!)

- Affected Sun 3 systems and VAX computers running 4 BSD UNIX variants
- Exploited a buffer overflow in fingerd to create a remote shell

8) The infection attempts proceeded by one of three routes: rsh, fingerd, or sendmail

- 8a) The statek via rsh was done by attempting to spawn a remote shell by invocation of (in order of trial)/usr/ach/rsh, /usr/bin/rsh, and /bin/rsh. If successful, the hest was infected as in steps 1 and 2a, above.
- 8b) The attack via the flager datation was tomewhat more subtle. A correction was tomewhat more subtle for the remote flager server datations and loss in a speciality constructed with a speciality of 155 bytes was passed to the datation, workflowing its impact birth and the speciality of the spec

/bin'

That is, the code executed when the main routine attempted to return was

execve("/bin/sh", 0, 0)

On VAXen, this resulted in the worm connected to a remote shell via the TCP connection. The worm then proceeded to infect the host as in steps 1 and 2a, above. On Suns, this simply resulted in a core file since the code was not in place to corrupt a Sun version of *fngerat* in a similar fashion.

8c) The worm then tried to infect the remote host by establishing a connection to the SMTP port and mailing an infection, as in step 2b, above.

Further reading: The internet worm program: an analysis. E.H. Spafford. 1989. SIGCOMM Comput. Commun. Rev. 19, 17-57, doi:dad

10.1145/66093.66095

Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]

Zaradoza

What we need to know

- How the stack works
- Calling conventions
- How system calls are made

Anything else?...

- Target system CPU
 - Little-endian vs. big-endian

Target system operating system

UNIX vs. Windows: stack frame changes!



Linux x86 process memory layout

0xFFFFFFFF	kernel space
	(1GiB)
0xC0000000	user stack
₩	
0x40000000	shared libraries
1	runtime heap
	bss
€	static data
0x08048000	(ELF binary loaded here)
0	unused

Check output of: cat /proc/<PROCESS PID>/maps



Buffer overflows Stack frame



- Stack Pointer (%esp): top of the stack
- Base Pointer (%ebp): base of the current frame
- Function arguments belong to the previous stack frame
 - Each function defines its own stack frame

Note: Stack grows to lower memory addresses



Stack concepts summary

- The stack stores abstract data
- Last-In-First-Out (LIFO) policy
- Assembly instructions of interest:
 - push: inserts an item on top of the stack, and decreases %esp by 4 bytes (dword size)
 pop: eliminates the item at the top of the stack, and increments %esp by 4 bytes



Stack concepts summary

- The stack stores abstract data
- Last-In-First-Out (LIFO) policy
- Assembly instructions of interest:
 - push: inserts an item on top of the stack, and decreases %esp by 4 bytes (dword size)
 - pop: eliminates the item at the top of the stack, and increments %esp by 4 bytes
 - call: inserts as the address of the next instruction which immediately follows the call on top of the stack, and decreases %esp by 4 bytes
 - Return of functions. %esp is incremented after execution. They accept an optional immediate value, which increments more %esp
 - retn: near return, retrieves the top of the stack and sets it as %eip
 - retf: far return, retrieves two dwords from the top of the stack and sets them as %eip and cs (code segment), respectively. Note that although cs is word size, it takes two dwords off from stack!



Stack concepts summary On 32-bit architectures

- Function arguments
- Return address
- Local variables

On 64-bit architectures

- Also stores function arguments, but differs from 32-bit architectures:
 - UNIX uses System V Application Binary Interface (ABI): first 6 integer (or pointer) arguments to a function are passed in registers (%rdi, %rsi, %rdx, %rcx, %r8, and %r9); from the 7th argument onwards, the stack is used
 - Microsoft ABI: only 4 registers are used (%rcx, %rdx, %r8, and %r9); from the 5th argument onwards, the stack is used

Return address

Local variables

Further reading: http://eli.thegreenplace.net/2011/09/06/stack-frame-layout-on-x86-64



```
void readName(){
   char username[256];
   printf("Type user name: "); %eip: push ebp
   scanf("%s", username);
}
```

readName:

push e
push o mov sub push call add sub lea push call add call leave ret

Т

Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]

Universidad Zaragoza

```
void readName(){
   char username[256];
   printf("Type user name: ");
   scanf("%s", username);
}
```

readName:

sub sub push call add sub lea push push call add		
sub sub push call add sub lea push push call add	oush	ebp
sub sub push call add sub lea push push call add		day and
sub push call add sub lea push push call add	mov	ebp, esp
sub push call add sub lea push push call add	sub	esp,
push call add sub lea push push call add		esp,
call add sub lea push push call add		OFFS
add sub lea push call add		
sub lea push push call add		print
lea push push call add		esp,
push push call add	sub	esp,
push call add	Lea	eax,
call add	oush	eax
call add	oush	OFFSE
add		iso
		esp,
	Leave	
ret	ret	

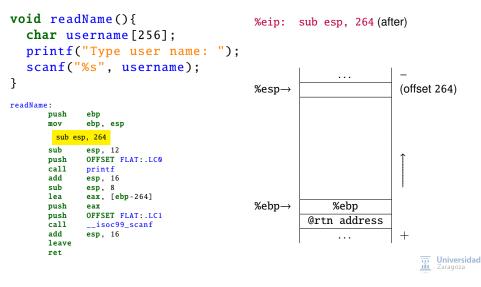


Universidad Zaragoza

```
void readName(){
    char username[256];
    printf("Type user name: ");
    scanf("%s", username);
}
```

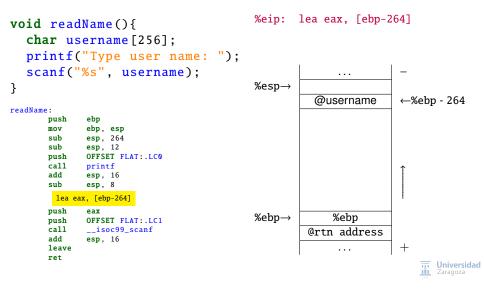
readName: push ebp . . . ebp, esp mov sub esp, 264 sub esp, 12 push **OFFSET FLAT:.LCO** %ebp call %ebp=%esp→ printf add esp, 16 @rtn address sub esp. 8 lea eax. [ebp-264] +push eax push OFFSET FLAT: . LC1 call isoc99 scanf add esp, 16 leave ret

Universidad

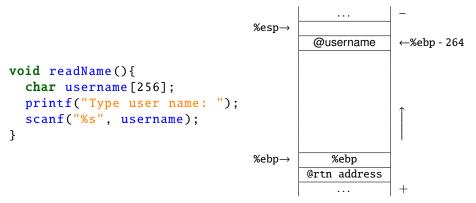


Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]

2021/2022 26 / 52



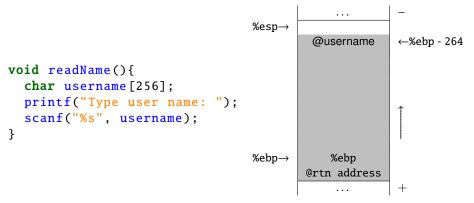
2021/2022 26 / 52



What if username is more than 264 bytes long?



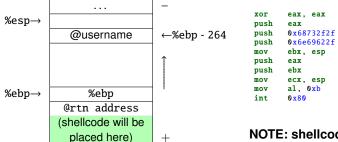
Buffer overflows Example



What if username is more than 264 bytes long?

- The adjacent memory to username is overwritten, since scanf does not check for any buffer limits (it is an insecure function)
- Arbitrary code execution, since %eip will pop the top value of the stack when the function returns!

Basic stack exploit



NOTE: shellcode runs on the stack

1 Insert your shellcode on the stack

Shellcode: originally, the minimal code to launch a shell (i.e., exec("/bin/sh")). Today, any code injected regardless of its purpose

2 Manipulate @rtn address to return to your shellcode

Look for assembly instructions that allow redirection of execution to %esp

When the vulnerable function ends, the shellcode runs!

Further reading: Smashing The Stack For Fun And Profit. Aleph One, Phrack 49 (1996), http://phrack.org/issues/49/14.html

Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]

2021/2022 28 / 52

TTT Universidad

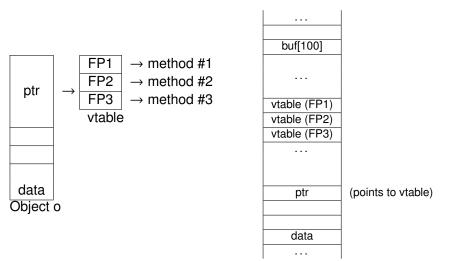
Insecure libc functions - (non-exhaustive list)

- strcpy \rightarrow strlcpy/strcpy_s (Windows CRT)
- strcat \rightarrow strncat \rightarrow strlcat/strcat_s (Windows CRT)
- strtok
- $sprintf \rightarrow snprintf$
- vsprintf \rightarrow vsnprintf
- $\blacksquare gets \rightarrow fgets/gets_s CRT)$
- scanf/sscanf → sscanf_s (Windows CRT)
- $snscanf \rightarrow _snscanf_s$ (Windows CRT)
- strlen \rightarrow strnlen_s (Windows CRT)

Some safe versions are misleading

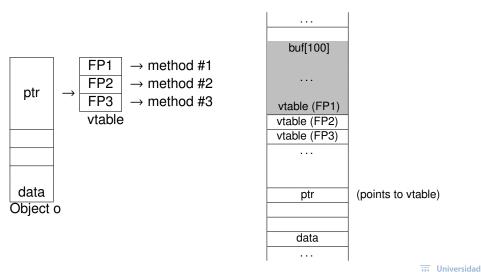
strncpy, strncat can leave strings unfinished – be careful!

Corrupting method pointers - Heap overflow



Universidad

Corrupting method pointers - Heap overflow



Buffer overflows How to hunt overflows...

Find the overflow

- Configure the operating system correctly (core dump?)
- Issue malformed inputs with specific endings
 - Automated tools (fuzzers)

■ If the application crashes, check the CPU registers for the endings



Buffer overflows How to hunt overflows...

Find the overflow

- Configure the operating system correctly (core dump?)
- Issue malformed inputs with specific endings
 - Automated tools (fuzzers)
- If the application crashes, check the CPU registers for the endings

Build the exploit

- Analyze overflow conditions
- Check if the overflow can lead to arbitrary code execution
 - Not easy, given the latest built-in defenses at the OS level



Outline

A little recap

2 Buffer Overflows

3 Defenses against Control-Flow Hijacking Attacks

- Stack Data Protection
- Non-Executable Stack
- Write XOR eXecute (W^AX) Pages
- Address Space Layout
- Other Techniques of Defense



Defeating control-flow hijacking attacks Approaches

1 Fix bugs:

- Audit software to find bugs (there are automated tools soundness?)
- Re-code software in a type-safe language

2 Allow overflow, but prevent injected code from running

3 Insert runtime code to detect overflows

Process stops when overflow is detected

Further readings: SoK: Eternal War in Memory. L. Szekeres, M. Payer, T. Wei and D. Song. 2013 IEEE Symposium on Security and

Privacy, Berkeley, CA, 2013, pp. 48-62. doi: 10.1109/SP.2013.13

Memory Errors: The Past, the Present, and the Future. V. van der Veen, N. dutt-Sharma, L. Cavallaro, H. Bos (2012). In Research in Attacks. Intrusions. and Defenses. RAID 2012. LNCS. vol 7462. Springer. doi: 10.1007/978-3-642-33338-5 5

Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]

2021/2022 33 / 52

Universidad

Defeating control-flow hijacking attacks Stack data protection

Stack cookies

- Detect stack-based overflows by:
 - 1 2
- In the function prologue, push a magic number)
 - In the function epilogue, check this value

Defeating Control-Flow Hijacking Attacks Stack cookies

■ Initial ideas come from StackGuard (Crispin Cowan, 1997)

Enhanced by Hiroaki Etoh with ProPolice (2000)

Later renamed to SSP (Stack-Smashing Protector), included in mainstream GCC version 4.1

Types of canaries:

- Null canary (all zeros; 0x00000000)
- Terminator canary (0x000aff0d)
 - 0x00 stops strcpy() (and related functions)
 - 0x0a and 0x0d stop gets() (and related functions)
 - 0xff (EOF) stops other functions
- Random canary



Defeating control-flow hijacking attacks Stack cookies

How to protect information stored after the vulnerable buffer?

- Add a canary after each buffer and check each time before accessing any other data stored after it
 - Good idea, may be a compiler modification
 - However, not practical: performance impact
- Reorder local variables on the stack to move the sensitive data out of the way of the buffer overflow
 - Side effect of compiler optimizations
 - Implemented as an intentional protection in ProPolice: ideal stack layout
 - Places local buffers at the end of the stack frame
 - Relocates other local variables before them
 - Also introduced by Microsoft Visual Studio (/GS feature)



Defeating control-flow hijacking attacks Stack cookies

Ideal stack layout does not always exist...

- Multiple local buffers are placed one after another
- Structure members cannot be rearranged (interoperability issues)
- Particular structures (like arrays of pointers) can be overflowed or be treated as sensitive information, depends on the semantics
- Functions with a variable number of arguments remain unprotected
- Dynamically created buffers on the stack (e.g., alloca()) are placed at the top of the stack frame

Universidad

Defeating control-flow hijacking attacks Stack cookies

readName. ebp push ebp, esp mov sub esp, 280 readName: mov eax. DWORD PTR gs:20 push ebp DWORD PTR [ebp-12], eax mov ebp. esp mov xor eax, eax sub esp. 264 esp. 12 sub sub esp, 12 push OFFSET FLAT: LCO push OFFSET FLAT: LCO call printf call printf add esp. 16 add esp, 16 sub esp, 8 sub esp, 8 lea eax, [ebp-268] lea eax. [ebp-264] push eax push eax push OFFSET FLAT: LC1 OFFSET FLAT: .LC1 push call isoc99 scanf call. isoc99 scanf add esp, 16 add esp. 16 eax. DWORD PTR [ebp-12] mov leave xor eax. DWORD PTR gs:20 ret .1.2 je call. stack chk fail (stack cookies disabled) .L2:

Bypassing it is still possible

(stack cookies enabled)

leave ret

- On Windows, SEH-based exploits
- On UNIX-like systems, we need a memory leak (or bruteforce)



лîт. 2021/2022 38 / 52

Universidad

BOF exploitation steps

- 1 Place code in the stack (in the same vulnerable buffer)
- 2 Overwrite a return address
- 3 Jump to it



BOF exploitation steps

- 1 Place code in the stack (in the same vulnerable buffer)
- 2 Overwrite a return address
- 3 Jump to it

Non-executable stack

- First implemented for DEC on Alpha in Feb 1999
- Enabled by default on most desktop platforms, such as Linux, macOS, and Windows

Main weaknesses:

- Still allows the return address to be abused, overwriting it with an arbitrary location
- Does not prevent the execution of code already present in the process memory or code injected in other data areas



Universidad

Defeating control-flow hijacking attacks Non-executable stack

Bypassing techniques

```
return-into-libc (ret2libc for short)
```

- Use libc function addresses as return addresses
- The attacker does not require any shellcode to take control of a target, they simply redirect the execution of the control flow as they wish
- We will talk about this more in deep in the last part of the course!

Improved techniques:

- ret2plt
- ret2syscall
- ret2strcpy, ret2gets (or read(), recv(), recvfrom() variants)
- ret2data
- ret2text, ret2code, ret2dl-resolve
- Chained ret2code (or chained ret2libc)



W[^]X (memory) pages

- Logical extension of non-executable stacks
- Non-executable writable pages and non-writable executable pages



W[^]X (memory) pages

- Logical extension of non-executable stacks
- Non-executable writable pages and non-writable executable pages
- Term coined by Theo de Raadt (founder and main architect of OpenBSD)
- First implementation of W^AX : 1972! (Multics on the GE-645 mainframe)

Defeating control-flow hijacking attacks Write XOR eXecute (W^AX) pages

■ The PaX project (Oct 2000)

- Linux kernel patch for Intel x86 hardware
- Today, it is available for almost all hardware platforms
- It was never included in mainstream Linux distribution, although today most distributions have some kind of W^AX

On-chip support for non-executable pages came a bit later

- NX: Non-eXecutable feature (AMD Athlon 64; Sept 2003)
- ED: Execute-Disable feature (Intel P4 Prescott; Feb 2004)
- XN: eXecute-Never feature (ARM v6)

Software that took advantage of hardware support emerged a few months later

- Linux kernel patches (via PaX project)
- Microsoft Windows XP Service Pack 2 (Data Execution Prevention; DEP opt-in by default)



Defeating control-flow hijacking attacks Can we still execute arbitrary injected code when W^AX is on?

- Do we really need to inject new code? Otherwise, ret2code
- Is there a page with W+X permissions? If so, ret2strcpy or ret2gets
- Can we chain the existing code, using ret2code, to write an executable file to disk and then run it?



Defeating control-flow hijacking attacks Can we still execute arbitrary injected code when W^AX is on?

- Do we really need to inject new code? Otherwise, ret2code
- Is there a page with W+X permissions? If so, ret2strcpy or ret2gets
- Can we chain the existing code, using ret2code, to write an executable file to disk and then run it?
- Is there a way to turn the protection off?
 - SetProcessDEPPolicy / ZwSetInformationProcess on Windows platforms
- Can we change the permissions of a specific memory region from W[∧]X to W+X?
 - VirtualProtect on Windows platforms
 - mprotect on GNU/Linux platforms
 - <u>note</u>: PaX does not allow a page to be W+X, nor X after W
 - In kernel, it requires the memory address to be aligned to 4KiB
- Can we create a new memory region with W+X permissions?
 - VirtualAlloc() on Windows platforms
 - mmap() on Unix-like platforms
 - As before, not allowed if PaX is installed
 - You will first need to copy the injected code and then jump there (chained ret2code: mmap-strcpy-code)



ret2libc allows us to bypass non-executable stacks

Addresses of functions are known and are part of the attacker's input



ret2libc allows us to bypass non-executable stacks

Addresses of functions are known and are part of the attacker's input

ASCII Armored Address Space (AAAS)

- Linux kernel patch that loaded all shared libraries into memory addresses starting with a null byte
- Similar idea to terminator canaries
- Protects against strcpy-like exploitation, but not gets
- Still vulnerable to other ret2- attacks



Address Space Layout Randomization (ASLR)

- Randomizes the address of everything (libraries, image, stack, and heap)
- Prevents the attacker from knowing where to jump or where to point pointers
- First implemented in PaX for Linux in 2001:
 - "unless every address is randomized and unpredictable, there's always going to be room for some kind of attack"
- Introduced in Windows Vista (2007)
- NOTE: if the attacker can inject code and there is enough room for nops, an approximate address can be enough to achieve reliable code execution
 - This technique is known as NOP-sled or NOP slide

■ Is there anything left in a predictable address?

- In most cases, yes:
 - Images are usually compiled to run in a fixed known memory address
 - No relocatable shared dynamic libraries
 - Improvement: PIE (Position Independent Execution) code, on Linux platforms (2005)
- ret2code approaches
- Can we guess the randomly generated addresses?
 - It depends. Low entropy on 32-bits
 - On 32-bit Windows, even lower entropy
- Is there a clever way to find these addresses?
 - Is there a memory leak available?
 - Brute-forcing is always an option



Some final remarks

- On Windows, threads of the same application share the memory layout
- On Unix, fork processes replicates the parent memory layout

ASLR is a very strong protection against code execution exploits, but most operating systems do not offer a complete solution



Software Vulnerabilities [CC BY-NC-SA 4.0 © R.J. Rodríguez]

2021/2022 47 / 52

Defeating control-flow hijacking attacks ASLR on Windows

Stack location:

- The time stamp counter (TSC) of the current processor is shifted and masked to a 5-bit value (2⁵ options)
- Added to another 9-bit TSC-derived value to make up the base address of the stack

Heap location:

- TSC shifted and masked to a 5-bit value (2⁵ options), multiplied by 64KiB
- The possible heap address ranges from 0x00000000 to 0x001f0000



Defeating control-flow hijacking attacks ASLR on Windows

Executable images location:

- **Load displacement by calculating a** δ value each time an app runs
- 8-bit pseudo-random number → only one of 256 possible locations
 - TSC shifts four places, and then divides modulo 254 and adds 1
 - The result is then multiplied by the allocation granularity of 64 KiB

This δ value is added to the preferred load address of the image file



Defeating control-flow hijacking attacks Address Space Layout Randomization (ASLR) in Windows

Shared libraries location:

- Load offset is calculated with a system-wide per-boot value called the image bias
 - Stored in a global memory state structure (MI_SYSTEM_INFORMATION), in field MiState.Sections.ImageBias)
- Calculated only once per startup
- Shared memory region between 0x50000000 and 0x78000000
- First DLL is always ntdll. We can calculate its image base address as:
 - 0x78000000 (ImageBias + NtDllSizein64KBChunks)*0x10000 (32-bit)



Other techniques of defense

Probabilistic methods

- Instruction Set Randomization
- Data Space Randomization: randomizes the representation of data stored in memory (not location). Encrypts all variables, not just pointers, and using different keys



Other techniques of defense

Probabilistic methods

- Instruction Set Randomization
- Data Space Randomization: randomizes the representation of data stored in memory (not location). Encrypts all variables, not just pointers, and using different keys

Generic methods

- Data Integrity: spatial memory integrity (protect against invalid memory writes)
- Data Flow Integrity: checks read instructions to detect data corruption before use

Other techniques of defense

Probabilistic methods

- Instruction Set Randomization
- Data Space Randomization: randomizes the representation of data stored in memory (not location). Encrypts all variables, not just pointers, and using different keys

Generic methods

- Data Integrity: spatial memory integrity (protect against invalid memory writes)
- Data Flow Integrity: checks read instructions to detect data corruption before use

Other defenses against hijacking the flow of control

- Code Pointer Integrity
- Control Flow Integrity (CFI)



Exploiting Software Vulnerabilities Software Vulnerabilities CONTROL-FLOW HIJACKING

③ All wrongs reversed – under CC-BY-NC-SA 4.0 license



Dept. of Computer Science and Systems Engineering University of Zaragoza, Spain

Course 2021/2022

Master's Degree in Informatics Engineering

UNIVERSITY OF ZARAGOZA Seminar A.25, Ada Byron building

