Binary Reverse Engineering And Analysis Course 2: Assembly

Caragea Radu

February 23, 2021

Black-box analysis: figure out only from external interactions

- Black-box analysis: figure out only from external interactions
- White-box analysis: exhaustively cover the binary
- Gray-box analysis: middle ground
- Today we start learning concepts for white-box analysis

Executables

- Most executables (ELF/SO, PE/DLL, WASM) have structure
- Based on generic computer science concepts
- Multiple sections/segments:
 - Text section (text == readable by the CPU)
 - Read-only Data section/Read-Write Data Section
 - Relocations/Compiler Stubs
- However...

Executables



CPU functionality

- The CPU consumes code and produces effects
- The consumed code is in binary form (machine code)

000006d0	85	cθ	74	θc	5d	ff	eθ	66	0f	1f	84	00	00	60	00	00	[t.]f
000006e0	5d	c3	θf	1f	40	00	66	2e	0f	1f	84	00	00	00	00	00	[]@.f
000006f0	80	3d	49	09	20	00	66	75	27	48	83	3d	f7	68	20	00	.=Iu'H.=
00000700	00	55	48	89	e5	74	θc	48	8b	3d	2a	09	20	60	e8	Θd	.UHt.H.=*
00000710	ff	ff	ff	e8	48	ff	ff	ff	5d	сб	05	20	09	20	00	01	[H]
00000720	f3	c3	θf	1f	40	00	66	2e	0f	1f	84	60	00	00	00	00	@.f
00000730	48	8d	3d	b1	06	20	00	48	83	3f	00	75	θb	e9	5e	ff	H.=H.?.u^.
00000740	ff	ff	66	0f	1f	44	66	00	48	8b	05	99	08	20	00	48	[f.DHH]
00000750	85	cθ	74	e9	55	48	89	e5	ff	dΘ	5d	e9	40	ff	ff	ff	t.UH].@
00000760	55	48	89	e5	48	83	ec	50	c6	45	eθ	41	c6	45	e1	47	UHHP.E.A.E.G
00000770	c6	45	e2	42	сб	45	e3	57	сб	45	e4	40	c6	45	e5	6d	.E.B.E.W.E.@.E.m
00000780	сб	45	e6	41	сб	45	e7	57	сб	45	e8	51	cб	45	e9	40	.E.A.E.W.E.Q.E.@
00000790	c6	45	ea	57	с6	45	eb	46	сб	45	ec	6d	c6	45	ed	42	.E.W.E.F.E.m.E.B
000007a0	c6	45	ee	53	с6	45	ef	41	c6	45	fØ	41	c6	45	f1	45	I.E.S.E.A.E.A.E.E
000007b0	с6	45	f2	5d	сб	45	f3	40	сб	45	f4	56	с6	45	f5	03	[.E.].E.@.E.V.E
000007c0	с6	45	f6	00	сб	45	f7	01	сб	45	f8	01	с6	45	f9	00	.EEEE
000007d0	c6	45	fa	07	c6	45	fb	32	48	8d	45	bθ	48	89	c6	48	.EE.2H.E.HH
000007e0	8d	3d	ee	00	00	00	b8	00	00	00	00	e8	20	fe	ff	ff	.=
000007f0	c7	45	fc	00	00	00	00	eb	1c	8b	45	fc	48	98	θf	b6	.EE.H
00000800	44	05	e0	83	fØ	32	89	c2	8b	45	fc	48	98	88	54	05	D2E.HT.
00000810	e0	83	45	fc	01	8b	45	fc	83	f8	1b	76	dc	48	8d	55	EEv.H.U
00000820	e0	48	8d	45	bØ	48	89	d6	48	89	c7	e8	dØ	fd	ff	ff	.H.E.HH
00000830	85	cθ	75		48	8d	3d	9e	00	00	00	e8	bØ	fd	ff	ff	u.H.=
00000840	eb	θc	48	8d	3d	99	00	00	00	e8	a2	fd	ff	ff	b8	00	H.=
00000850	00	60	00	c9	c3	66	2e	θf	1f	84	00	60	00	66	00	90	f
00000860	41	57	49	89	d7	41	56	49	89	f6	41	55	41	89	fd	41	AWIAVIAUAA
00000870	54	4c	8d	25	60	05	20	00	55	48	8d	2d	60	05	20	00	TL.%`UH`
00000880	53	4c	29	e5	48	83	ec	08	e8	3b	fd		ff	48	c1	fd	SL).H;H
00000890	03	74	1b	31	db	Θf	1f	00	4c	89	fa	4c	89	f6	44	89	.t.1LLD.
000008a0	ef	41	ff		dc	48	83	c3	01	48	39	dd	75	ea	48	83	.AHH9.u.H.
000008b0	c4	68	5b	5d	41	5c	41	5d	41	5e	41	5f	c3	θf	1f	00	[]A\A]A^A
000008c0	c3	00	00	00	48	83	ec	θ8	48	83	c4		c3	00	00	00	HH
000008d0	01	60	02		25	34	30	73	00	43	6f		72	65	63	74	%40s.Correct
000008e0	21	66	57	72		6e	67	00	01	1b	03		Зc	00	00	00	!.Wrong;<
000008f0	06	66	00	00	f8		ff	ff	88	00	00	60		fd		ff	8
00000900	bΘ	66	00	00	48	fd	ff	ff	58	00	00	66			ff	ff	HXx
00000910	c8	00	00	00	78	ff	ff	ff	e8	00	00	00	d8	ff	ff	ff	×
00000920	30	01	00	00	00	00	00	00	14	00	00	66	00	00	00	00	0
00000930	01	7a	52	00	01	78	10	01	1b	θc	07	68	90	01	07	10	.zRx

CPU functionality

- Machine code can be unequivocally translated to readable assembly code
- In assembly form, it can be "interpreted" by the human brain
- For efficiency, it is organized into blocks, subroutines, functions, libraries, etc

	808: 8b 45 fc mov eax,DWORD PTR [rbp-0x4]	80b: 80d:	48 98 88 54 05 e0	cdqe mov	BYTE PTR [rbp+rax*1-0x20],dl	
POde PP E4 OF a0 may PVTE DTD [sharsav#1 0v201 d]	80b: 48 98 cdge	80d: 811:	88 54 05 e0 83 45 fc 01	mov add	BYTE PTR [rbp+rax*1-0x20],dl DWORD PTR [rbp-0x4],0x1	
808: 8b 45 fc mov eax,DWORD PTR [rbp-0x4]						
806: 89 c2 mov edx,eax 808: 8b 45 fc mov eax,DWORD PTR [rbp-0x4]						
803: 83 f0 32 xor eax,0x32 806: 89 c2 mov edx,eax 808: 80 45 fc mov eax,DWORD PTR [rbp-0x4]	803: 83 f0 32 xor eax,0x32				any RVTE DTP [rhp+ray#1.4x20]	
7fe: 0f b6 44 05 e0 movzx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 f0 32 xor eax,0x32 806: 89 c2 mov edx,eax 808: 80 45 fc mov eax,DWRD PTR [rbp-0x4]	7fe: 0f b6 44 05 e0 movzx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 f0 32 xor eax,0x32				eax,DWORD PIR [FDP-0X4]	
7fc: 48 cdqe 7fe: 0f b6 44 05 e0 movzx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 f0 32 xor eax,0x32 eax,0x32 806: 89 c2 mov eax,0x32 eax,0x32 806: 89 c2 mov eax,0x0RD PTR [rbp-0x4] 64	7fc: 48 98 cdqe 7fe: 9f b6 44 05 e0 movzx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 f0 32 xor eax,0x32					
7f9: 8b 45 fc mov eax,DWORD PTR [rbp-0x4] 7fc: 48 98 cdqe 7fe: 0f b6 44 05 e0 movzx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 f0 32 xor eax,0x32 806: 89 c2 mov edx,eax 808: 80 45 fc mov eax,DWORD PTR [rbp-0x4]	7f9: 8b 45 fc mov eax,DWORD PTR [rbp-0x4] 7fc: 48 98 cdqe 7fe: 0f b6 44 05 e0 movzx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 70 32 xor eax,0x32					
7f7: blc jmp 815 <main+9xb5< th=""> 7f9: 8bd fc mov eax,DWORD PTR [rbp-9x4] 7fc: 48 98 cdqe 7fc: 67 b6 44 05 e0 movzx 803: 83 f0 32 xor 806: 89 c2 mov eax,0x0RD PTR [rbp-fax*1-0x20] 806: 89 c2 mov eax,0x02</main+9xb5<>	7f7: eb 1c jmp 815 < main+4xb5> 7f9: 8b 45 fc mov eax,DWORD PTR [rbp-0x4] 7fc: 48 98 cdqe 7fc: 05 64 405 e0 movzx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 70 32 xor eax,832					
7f0: c7 45 fc 00 00 00 00 mov DWORD PTR [rbp-0x4], 0x0 7f7: eb 1c jmp 815 <main+0x5> 7f9: 8b 45 fc mov eax,DWORD PTR [rbp-0x4] 7fc: 0f b6 44 05 e0 movz eax,DWORD PTR [rbp+rax*1-0x20] 803: 83 f0 32 xor eax,0x32 806: 89 c2 mov edx,0x0RD PTR [rbp-0x4]</main+0x5>	7f0: c7 45 fc 00 00 mov DWORD PTR [rbp-0x4], 0x0 7f7: eb Lc jmp 815 smain+0x55> 7f9: 8b 5fc mov eax, DWORD PTR [rbp-0x4], 0x0 7f2: 48 98 cdqe cdqe 7fe: 0f 64 65 e0 movzx eax, BYTE PTR [rbp+rax*1-0x20] 803: 83 f0 32 xor eax, 8372					
Teb: e8 20 fe ff ff Call 610 ≤ isoc99 scanf@plt> 7f0: c7 45 fc 00 00 00 00 mov DMORD PTR [rbp-0x4],0x0 7f7: eb 1c jmp 815 <main+0xb5> 7f7: 80 45 fc mov eax,DMORD PTR [rbp-0x4] 7fc: 49 98 cdge cdge 7f6: 67 45 60 movz eax,BVTE PTR [rbp+rax*1-0x20] 803: 83 f0 32 xor eax,0x32 806: 80 5 c2 mov eax,0x0RD PTR [rbp-0x4]</main+0xb5>	7eb: e8 20 fe ff ff call 610 < isoc99 scanf@plt> 7f0: c7 45 fc 00 00 00 00 mov DW00D PTR [rbp-0x4],0x0 7f7: eb 1c jmp 815 <main+0x55> 7f6: 80 45 fc mov eax,DW0RD PTR [rbp-0x4] 7fc: 48 94 cd cd 7fc: 05 44 05 e0 movzx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 f0 32 xx eax,832</main+0x55>					
7e6: b8 00 00 00 mov eax;0x0	7e6: b8 00 00 00 00 mov eax, bx0 7eb: e8 20 fe ff ff call 610 < _iso29_scanf@plt> 7f0: c7 45 fc 00 00 00 mov DW0RD PTR [rbp-0x4], 0x0 7f7: eb 1c jmp 815 <main+0xb5> 7f7: b4 5 fc mov eax, DW0RD PTR [rbp-0x4] 7fc: 48 98 cdqe rdqe 7fc: 0f 64 40 50 e0 movzx eax, BYTE PTR [rbp+rax*1-0x20] 803: 837 f0 32 xor eax, 832</main+0xb5>					
7df: 48 8d 3d ee 00 00 00 lea rdi,[rip+0xee] # 8d4 <_I0_stdin_used+0x4> 7e6: b8 00 00 00 00 mov eax,0x0 eax,0x0 7eb: e3 20 fe ff ff call 610 <_isoc9s_canf@plt> 7f0: c7 45 fc 00 00 00 00 mov DM0RD PTR [rbp-0x4],0x0 7f1: eb 1c jmp 815 <main+0xb5> 7f2: 48 5 fc mov eax,0N0RD PTR [rbp-0x4] 7f2: 67 44 05 e0 movz eax,0NURD PTR [rbp+rax*1-0x20] 803: 83 fc 32 xov eax,0NURD PTR [rbp-0x4] 806: 89 5 fc mov eax,0NURD PTR [rbp-0x4]</main+0xb5>	7df: 48 8d 3d ee 00 00 00 lea rdi.[rip+0xe0] # 8d4 <_I0_stdin_used+0x4					
Tdc: 48 89 c6 mov rsi_rax 7df: 48 8d 3d ee 00 00 00 lea rdi,[rip+0xee] # 8d4 <_I0_stdin_used+0x4> 7d6: b8 00 00 00 00 mov eax,0x0 eax,0x0 7e6: b8 00 00 00 mov eax,0x0 ps: core 7e6: b8 00 00 00 mov eax,0x0 ps: core 7e0: c4 5 fc 00 00 00 mov b80k0 PTR [rbp-0x4] ps: core 7f1: eb 1c movz eax,0x0 PTR [rbp-0x4] ps: core 7f2: b4 5 fc movz eax,0x1 PTR [rbp-0x4] ps: core 7f2: b4 5 fc movz eax,8x2 eax,0x2 801: 83 f0 32 xor eax,8x32 eax,0x32 806: 89 c2 mov eax,0x32 eax,0x32 808: 84 5 fc mov eax,0x32 eax,0x40RD PTR [rbp-0x4]	7dc: 48 89 c6 mov rsi,rax 7df: 48 8d 3d ee 00 00 00 lea rdi,[rip+exee] # 8d4 <_I0_stdin_used+0x4:					
7d8: 48 8d 45 b0 lea rax,(rbp.0x50] 7dc: 48 89 c6 mov rsi,rax 7df: 48 89 c6 mov rsi,rax 7df: 48 80 36 mov eax,0x0 7d6: b8 00 00 00 mov eax,0x0 7e6: b8 00 00 00 00 mov eax,0x0 7b1: c2 45 fc 00 00 00 00 mov b100 < prod <td>psod psod pso</td> <td>7d8: 48 8d 45 bb Lea rax,[rbp-0x50] 7dc: 48 8d 26 mov rsi,rax 7df: 48 8d 26 mov rsi,rax 7df: 48 8d 26 mov rsi,rax 7df: 48 8d 26 e00 00 lea rdi,[rip+0xee] # 8d4 <_I0_stdin_used+0x4</td> 7e6: b8 00 00 mov eax,0x0 eax,0x0 rsi,rax 7eb: e3 20 fe ff ff call 610 <_isoc99_scanf@plt> rsi,bx0 7f7: eb 1c jm pl15 math=0x55> rsi,0x0 DWORD PTR (rbp-0x4] 7f6: 48 94 cdqe cdqe ax,0WORD PTR (rbp-0x4] rsi,0x0 7f6: 64 95 e0 movzx eax,0FTE PTR [rbp+rax*1-0x20] so33 s34 70 32	psod pso	7d8: 48 8d 45 bb Lea rax,[rbp-0x50] 7dc: 48 8d 26 mov rsi,rax 7df: 48 8d 26 mov rsi,rax 7df: 48 8d 26 mov rsi,rax 7df: 48 8d 26 e00 00 lea rdi,[rip+0xee] # 8d4 <_I0_stdin_used+0x4			mov	
7d4: c6 d5 fb 32 mov BYTE PTR [rbp-0x5],0x32 7d8: 48 8d 45 b0 lea rax,[rbp-0x50] 7d6: 48 89 c6 mov rsi,rax 7d7: 48 8d 3d e0 00 00 mov rsi,rax 7d7: 48 8d 3d e0 00 00 mov rsi,rax 7d6: 50 00 00 00 mov eax,0x0 7c6: b8 00 00 00 00 mov eax,0x0 7c6: b6 00 00 00 00 mov eax,0x0 7c6: b6 00 00 00 00 mov b000D PTR [rbp-0x4].0x0 7c7: eb 1c mov eax,0x0 7c7: b4 5 fc mov eax,0x0 7f6: cf 48 98 cdqe 7fc: d4 50 20 movzx 803: 83 f0 32 x0r 806: 80 5 c2 mov 808: 84 5 fc mov 808: 84 5 fc mov	7d4: c6 45 fb 32 mov BYTE PTR [rbp.θx5],0x32 7d8: 48 84 45 b0 lea rax, [rbp.θx50] 7dc: 48 89 c6 mov rsi,rax, 7df: 48 89 c6 mov rsi,rax, 7df: 48 89 c6 mov rsi,rax, 7df: 48 80 3d ee 00 00 00 mov rsi,rax, 7df: 48 80 3d ee 00 00 00 mov eax,8x0 7eb: b8 00 00 00 00 00 mov eax,8x0 7eb: e8 20 fe ff ff call 61 < isoc99 scanf@plt> 7f0: c7 45 fc 00 00 00 00 mov WDWORD PTR [rbp.ex4],0x0 7f7: eb 1c jmp 815 <main+8x5> 7f5: 85 96 cdqe rdy 7fc: 48 98 cdqe cdqe 7fc: e0 56 movx eax,BYTE PTR [rbp+rax*1-0x20] 803: 83 rd fo 32 xor</main+8x5>			mov		
7d0: c6 45 fa 07 mov BYTE PTR [rbp-0x6],0x7 7d1: c6 45 fb 32 mov BYTE PTR [rbp-0x5],0x32 7d8: k8 dd 45 b0 lea rax,[rbp-0x5],0x32 7d8: k8 dd 45 b0 lea rax,[rbp-0x5],0x32 7d6: k8 dd 90 60 mov rsi,rax 7d7: k8 dd 36 e00 00 00 mov eax,0x0 7b6: b6 00 00 00 00 mov pax040 px3 7b1: e3 20 fc ff ff call 610 <_isoc99 scanf@plt> 7b1: e3 20 fc ff ff call 610 <_isoc99 scanf@plt> 7b1: e3 20 fc ff ff call 610 <_isoc99 scanf@plt> 7b2: 80 45 fc mov b815 <mal-ne.b55> 7f2: 40 56 mov eax,0N0RD PTR [rbp-0x4] 7f6: 86 44 05 e0 movz eax,0N2 803: 83 fd 31 63 32 xor eax,0N2 806: 89 c2 mov eax,0N2 eax,0N2 806: 85 fc mov eax,0N2</mal-ne.b55>	7d0: c6 45 fa 07 mov BYTE PTR [rbp-0x6],0x7 7d1: c6 45 fb 32 mov BYTE PTR [rbp-0x5],0x32 7d8: c6 45 fb 32 mov BYTE PTR [rbp-0x5],0x32 7d8: c6 45 fb 30 lea rax,[rbp-0x5],0x32 7d6: 48 8d 45 b0 lea rax,[rbp-0x50] # 8d4 <_I0_stdin_used+0x42			mov		
Tcc: c6 45 f9 00 BVTE PTR [rbp-0x7],0x0 7d0: c6 45 f9 07 mov BVTE PTR [rbp-0x5],0x7 7d4: c6 45 f9 32 mov BVTE PTR [rbp-0x5],0x32 7d8: 48 d4 5b 0 lea mov BVTE PTR [rbp-0x5],0x32 7d6: c6 45 f9 32 mov BVTE PTR [rbp-0x5],0x32 7d6: 48 80 c6 mov rax,[rbp-0x6],0x50] 7d7: 48 80 3d ee 00 00 00 mov rai,[rip+0xce] # 8d4 <_I0_stdin_used+0x4> 7e6: b8 00 60 00 00 mov exi,0x0 exi,0x0 7f6: 48 60 45 e0 00 00 00 mov exi,0x0 7f8: c7 45 fc 00 00 00 00 mov exi,0x0 7f9: c7 45 fc 00 00 00 00 mov wov exi,0x0 7f9: c7 45 fc 00 00 00 00 mov exi,0x0 PTR [rbp-0x4].0x0 7f9: c7 45 fc 00 00 00 00 mov exi,0x0 exi,0x1 7f6: 48 98 c c exi,0w1 7f6: 64 98 cd exi,0w1 <t< td=""><td>7cc: c6 45 f9 00 mov BYTE PTR [rbp-0x7], 0x0 7d0: c6 45 fa 07 mov BYTE PTR [rbp-0x6], 0x7 7d4: c6 45 fa 03 mov BYTE PTR [rbp-0x6], 0x7 7d4: c6 45 fb 032 mov BYTE PTR [rbp-0x5], 0x32 7d6: c6 45 fb 032 mov BYTE PTR [rbp-0x5], 0x32 7d6: d8 80 45 b0 lea rax, [rbp-0x50] 7d7: d8 80 3d e0 e0 00 00 en rd1, [r1p+0xee] # 8d4 <_I0_stdin_used+0x4:</td> 7e6: b8 00 00 00 00 mov exd, 0x0 exd, 0x0 7eb: e8 20 fe ff ff c1 61 4 _isoc99 scanf@plt> 7f9: eb 1c jm 03 15 main+0xb5> 7f9: 80 45 fc mov eax, DWORD PTR [rbp-0x4], 0x0 7f1: 48 96 cmov eax, DWORD PTR [rbp-0x4] 7f1: 80 45 fc mov eax, DWORD PTR [rbp-0x4] 7f1: 80 45 fc cmov eax, BYTE PTR [rbp-rex4] 7f2: 80 45 fc cmov eax, BYTE PTR [rbp+rex*1-0x20] 803:</t<>	7cc: c6 45 f9 00 mov BYTE PTR [rbp-0x7], 0x0 7d0: c6 45 fa 07 mov BYTE PTR [rbp-0x6], 0x7 7d4: c6 45 fa 03 mov BYTE PTR [rbp-0x6], 0x7 7d4: c6 45 fb 032 mov BYTE PTR [rbp-0x5], 0x32 7d6: c6 45 fb 032 mov BYTE PTR [rbp-0x5], 0x32 7d6: d8 80 45 b0 lea rax, [rbp-0x50] 7d7: d8 80 3d e0 e0 00 00 en rd1, [r1p+0xee] # 8d4 <_I0_stdin_used+0x4:			mov		
7c8: c6 d5 f8 01 mov BYTE PTR [rbp-0x3],0x1 7cc: c6 d5 f9 00 mov BYTE PTR [rbp-0x5],0x1 7d0: c6 d5 f9 00 mov BYTE PTR [rbp-0x5],0x3 7d1: c6 d5 f0 32 mov BYTE PTR [rbp-0x5],0x32 7d2: c4 d5 f0 32 mov BYTE PTR [rbp-0x50] 7d2: c4 88 d3 d5 e0 lea rax,[rbp-0x50] 7d2: d8 d3 d5 e0 00 00 lea rax,[rbp-0x50] 7d1: d4 88 d3 d5 e0 00 00 lea rax,[rbp-0x50] 7d2: d5 00 00 00 lea rax,[rbp-0x50] 7d6: b3 00 00 00 00 mov exi,0x0 7d6: c5 01 ff ff c1 c11 c11 7d8: c7 45 fc 00 00 00 00 mov exi,0x0 7d8: c7 45 fc 00 00 00 00 mov exi,0x0 7d8: c7 45 fc 00 00 00 00 mov exisoc9_scanf@plt> 7d8: c7 45 fc 00 00 00 00 mov exisoc9_scanf@plt> 7f8: b1 c jmov exisoc9_scanf@plt> 7f1: e0 1c jmov exisoc9_scanf@plt>	7c8: c6 45 f8 01 mov BYTE PTR [rbp-0x8]; 6x1 7c1: c6 45 f9 00 mov BYTE PTR [rbp-0x6]; 0x3 7d8: c6 45 fb 32 mov BYTE PTR [rbp-0x5]; 0x32 7d8: c6 45 fb 32 mov BYTE PTR [rbp-0x5]; 0x32 7d8: 48 8d 45 b0 lea rax, [rbp-0x50] 7d1: 48 6d 3d ee 00 00 lea rd1, [rbp-0x50] 7d1: 48 6d 3d ee 00 00 lea rd1, [rbp-0x50] 7d1: 48 6d 3d ee 00 00 lea rd1, [rbp-0xe] # 8d4 <_I0_stdin_used+0x42	7c0:	c6 45 f6 00	mov	BYTE PTR [rbp-0xa],0x0	
7c4: c6 45 7 0 1 mov BYTE PTR [rbp-0x9],0x1 7c8: c6 45 76 01 mov BYTE PTR [rbp-0x7],0x1 7c6: c6 45 76 00 mov BYTE PTR [rbp-0x7],0x0 7c6: c6 45 76 00 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 76 00 mov BYTE PTR [rbp-0x6],0x1 7d6: c6 45 76 00 mov BYTE PTR [rbp-0x6],0x32 7d6: c6 45 76 32 mov BYTE PTR [rbp-0x6],0x32 7d6: d8 48 5b 0 lea rax,[rbp-0x6],0x53] 7d6: d8 d3 de 00 00 lea rax,[rbp-0x6],0x50] 7d7: d8 d3 de 00 00 00 lea rdi,[rip+0xce] # 8d4 <_IO_stdin_used+0x4> 7e6: b8 00 00 00 00 mov rax,[rbp-0x4],0x0 mov 7f0: c7 45 fc 00 00 00 00 mov eax,0x0 mov 7f1: c7 45 fc 00 00 00 00 mov mov eax,0x0 7f2: c7 45 fc 00 00 00 00 mov eax,0x0 mov 7f2: c7 45 fc 00 00 00 00 mov eax,0x0 mov 7f2: c7 45 fc 00 00 00 00 mov <td>7c4: c6 45 f7 01 mov BYTE PTR [rbp-0x9],0x1 7c8: c6 45 f8 01 mov BYTE PTR [rbp-0x9],0x1 7c8: c6 45 f8 00 mov BYTE PTR [rbp-0x7],0x0 7c4: c6 45 f8 00 mov BYTE PTR [rbp-0x6],0x7 7c4: c6 45 f8 07 mov BYTE PTR [rbp-0x5],0x32 7c4: c6 45 f8 02 loa mov BYTE PTR [rbp-0x5],0x32 7c4: c4 85 d3 0e 00 00 loa rax,[rbp-0x50] rax,[rbp-0x50] 7c4: c4 86 d3 de 00 00 00 mov rsx,i,rax rsx,0x0 7c5: c8 00 f0 00 00 mov erdi,[rip+0xe2] # 8d4 <_I0_stdin_used+0x4:</td> 7c6: c3 20 fe ff ff c1 610 <_iisoC99 scanf@plt> rfmi+0x55 7f9: c4 20 fe ff ff call 610 <_iisoC99 scanf@plt> rfmi+0x55 7f9: c5 16 c mov b315 ~main+0x55 b3 7f1: 48 96 c cdq rdq rrb+rb*re*1+0x20] 7f9: 80 45 fc mov b315 ~main+0x55 b3 7f1:	7c4: c6 45 f7 01 mov BYTE PTR [rbp-0x9],0x1 7c8: c6 45 f8 01 mov BYTE PTR [rbp-0x9],0x1 7c8: c6 45 f8 00 mov BYTE PTR [rbp-0x7],0x0 7c4: c6 45 f8 00 mov BYTE PTR [rbp-0x6],0x7 7c4: c6 45 f8 07 mov BYTE PTR [rbp-0x5],0x32 7c4: c6 45 f8 02 loa mov BYTE PTR [rbp-0x5],0x32 7c4: c4 85 d3 0e 00 00 loa rax,[rbp-0x50] rax,[rbp-0x50] 7c4: c4 86 d3 de 00 00 00 mov rsx,i,rax rsx,0x0 7c5: c8 00 f0 00 00 mov erdi,[rip+0xe2] # 8d4 <_I0_stdin_used+0x4:	7bc:	c6 45 f5 03	mov	BYTE PTR [rbp-0xb],0x3	
7c0: c6 45 f6 00 mov BYTE PTR [rbp-0.8a],0x0 7c4: c6 45 f7 01 mov BYTE PTR [rbp-0.8a],0x1 7c8: c6 45 f8 01 mov BYTE PTR [rbp-0.8b],0x1 7c6: c6 45 f8 01 mov BYTE PTR [rbp-0.8b],0x1 7c6: c6 45 f8 01 mov BYTE PTR [rbp-0.8b],0x1 7d0: c6 45 f8 30 mov BYTE PTR [rbp-0.8b],0x3 7d4: c6 45 f8 32 mov BYTE PTR [rbp-0.8b],0x32 7d6: 48 8d 45 b0 lea rax,[rbp-0.850] 7d7: 48 8d 3d ee 00 00 lea rati,[rbp-0.8c6],ex32 7d6: 64 00 00 00 nov rsi,rax 7d6: 62 0f eff ff ff c10 16 10 <=_isoc99_scanf@plt> 7f8: c7 45 fc 00 00 00 00 mov wax,b00 7f8: c7 45 fc 00 00 00 00 mov wax,b00 7f8: c7 45 fc 00 00 00 00 mov wax,b00 7f8: c7 45 fc 00 00 00 00 mov wax,b00 7f8: c7 45 fc 00 00 00 00 mov wax,b00 7f8: b1 c jmov wax <bdov< td=""></bdov<>	7c8: c6 45 f6 00 mov BYTE PTR [rbp-0xa],0x0 7c4: c6 45 f7 01 mov BYTE PTR [rbp-0x3],0x1 7c8: c6 45 f8 01 mov BYTE PTR [rbp-0x8],0x1 7c6: c6 45 f9 00 mov BYTE PTR [rbp-0x8],0x1 7c6: c6 45 f9 00 mov BYTE PTR [rbp-0x6],0x7 7d4: c6 45 fb 32 mov BYTE PTR [rbp-0x5],0x32 7d6: c4 85 fb 32 mov BYTE PTR [rbp-0x5],0x32 7d6: c4 85 fb 32 mov BYTE PTR [rbp-0x5],0x32 7d6: c4 86 d3 de e0 00 00 lea rat, [rbp-0x50] 7d7: d8 d3 de e0 00 00 mov rst,rax 7d6: c3 20 fe ff ff call 61 64 < _isoc99_scanf@plt> 7d7: c3 5 fc 00 00 00 00 mov b00 b00 b0 7d7: c3 5 fc 00 00 00 00 mov b15 smain+0x55> 7f7: b1c jm 815 smain+0x55 jm 815 smain+0x55 7f6: 8 98 cdq cdq 7fc: 8 45 fc mov eax,BVTE PTR [rbp-ex4]	7b8:	c6 45 f4 56	mov	BYTE PTR [rbp-0xc],0x56	
Tbc: c6 45 f5 69 mov BYTE PTR [rbp-0xb],0x3 7C0: c6 45 f6 60 mov BYTE PTR [rbp-0xb],0x3 7C4: c6 45 f6 60 mov BYTE PTR [rbp-0xb],0x1 7C4: c6 45 f7 61 mov BYTE PTR [rbp-0xb],0x1 7C4: c6 45 f7 601 mov BYTE PTR [rbp-0xb],0x1 7C4: c6 45 f7 601 mov BYTE PTR [rbp-0xb],0x1 7C4: c6 45 f7 600 mov BYTE PTR [rbp-0xb],0x3 7C4: c6 45 f7 632 mov BYTE PTR [rbp-0xb],0x32 7C4: c6 45 f7 632 mov BYTE PTR [rbp-0xb],0x32 7C4: c6 45 d5 32 mov BYTE PTR [rbp-0xb],0x32 7C4: c6 48 d5 b60 mov rax, [rbp-0xb],0x32 7C4: 48 d5 d6e 06 00 00 mov rax, [rbp-0xb],0x32 7C6: c8 d3 d6e c60 00 00 mov rax, [rbp-0xb],0x32 7C6: c8 d0 60 00 00 mov eax,0x0 7C7: c7 45 fc 60 00 60 00 mov mov 7C6: c7 45 fc 60 00 60 00 mov </td <td>Tbc: c6 45 f5 03 mov BYTE PTR [rbp-0xb], 0x3 7c0: c6 45 f6 00 mov BYTE PTR [rbp-0xb], 0x1 7c1: c6 45 f7 01 mov BYTE PTR [rbp-0x3], 0x1 7c2: c6 45 f7 01 mov BYTE PTR [rbp-0x3], 0x1 7c4: c6 45 f7 01 mov BYTE PTR [rbp-0x3], 0x1 7c4: c6 45 f7 01 mov BYTE PTR [rbp-0x5], 0x1 7c4: c6 45 f7 00 mov BYTE PTR [rbp-0x5], 0x2 7d0: c6 45 f7 00 mov BYTE PTR [rbp-0x5], 0x32 7d1: c6 45 f7 00 mov BYTE PTR [rbp-0x5], 0x32 7d2: c4 85 d3 0c 00 00 00 er ax, [rbp-0x5], 0x32 7d1: 48 80 3d 0c 00 00 00 er ax, [rbp-0x5] 7d1: c4 80 3d 0c 00 00 mov erax, 0x0 7d1: c4 80 3d 0c 00 00 00 mov erax, 0x0 7d2: c4 50 fc ff ff c1 610 < _isoc99 scanf@plt> 7f1: d8 d5 fc mov bx0MDVD PTR [rbp-0x4], 0x0 7f7: eb 1c jm 815 - main+0xb5><!--</td--><td>7b4:</td><td>c6 45 f3 40</td><td>mov</td><td></td></td>	Tbc: c6 45 f5 03 mov BYTE PTR [rbp-0xb], 0x3 7c0: c6 45 f6 00 mov BYTE PTR [rbp-0xb], 0x1 7c1: c6 45 f7 01 mov BYTE PTR [rbp-0x3], 0x1 7c2: c6 45 f7 01 mov BYTE PTR [rbp-0x3], 0x1 7c4: c6 45 f7 01 mov BYTE PTR [rbp-0x3], 0x1 7c4: c6 45 f7 01 mov BYTE PTR [rbp-0x5], 0x1 7c4: c6 45 f7 00 mov BYTE PTR [rbp-0x5], 0x2 7d0: c6 45 f7 00 mov BYTE PTR [rbp-0x5], 0x32 7d1: c6 45 f7 00 mov BYTE PTR [rbp-0x5], 0x32 7d2: c4 85 d3 0c 00 00 00 er ax, [rbp-0x5], 0x32 7d1: 48 80 3d 0c 00 00 00 er ax, [rbp-0x5] 7d1: c4 80 3d 0c 00 00 mov erax, 0x0 7d1: c4 80 3d 0c 00 00 00 mov erax, 0x0 7d2: c4 50 fc ff ff c1 610 < _isoc99 scanf@plt> 7f1: d8 d5 fc mov bx0MDVD PTR [rbp-0x4], 0x0 7f7: eb 1c jm 815 - main+0xb5> </td <td>7b4:</td> <td>c6 45 f3 40</td> <td>mov</td> <td></td>	7b4:	c6 45 f3 40	mov		
Tbi: c6 45 13 40 mov BYTE PTR [rbp-0xd],0x40 Tbi: c6 45 14 56 mov BYTE PTR [rbp-0xd],0x56 Tbi: c6 45 14 56 mov BYTE PTR [rbp-0xd],0x36 Tbi: c6 45 16 00 mov BYTE PTR [rbp-0xd],0x40 Tci: c6 45 17 01 mov BYTE PTR [rbp-0xd],0x0 Tci: c6 45 17 01 mov BYTE PTR [rbp-0xd],0x0 Tci: c6 45 17 01 mov BYTE PTR [rbp-0xd],0x0 Tci: c6 45 16 01 mov BYTE PTR [rbp-0xd],0x0 Tci: c6 45 16 00 mov BYTE PTR [rbp-0xd],0x0 Tci: c6 45 16 32 mov BYTE PTR [rbp-0xd],0x1 Tdi: d8 44 5b0 lea mov BYTE PTR [rbp-0xd],0x32 Tdi: d8 d3 de 00 00 00 mov rsi, rax rdi,[rip+0xc] # 8d4 <_IO_stdin_used+0x4> T6: d8 03 de 00 00 00 mov eax,0x04 mov eax,0x04 T6: c7 45 16 00 00 00 00 mov eax,0x04 mov eax,0x04 T71: <	$ \begin{array}{llllllllllllllllllllllllllllllllllll$	7b0:	c6 45 f2 5d	mov	BYTE PTR [rbp-0xe],0x5d	
Tbi: c6 d5 f3 40 mov BYTE PTR [rbp-0xd],0x40 Tbi: c6 d5 f4 56 mov BYTE PTR [rbp-0xd],0x40 Tbi: c6 d5 f4 56 mov BYTE PTR [rbp-0xd],0x40 Tc0: c6 d5 f6 00 mov BYTE PTR [rbp-0xd],0x40 Tc1: c6 d5 f6 00 mov BYTE PTR [rbp-0xd],0x40 Tc2: c6 d5 f7 01 mov BYTE PTR [rbp-0xd],0x0 Tc2: c6 d5 f8 01 mov BYTE PTR [rbp-0xd],0x40 Tc2: c6 d5 f8 01 mov BYTE PTR [rbp-0xd],0x40 Tc2: c6 d5 fa 07 mov BYTE PTR [rbp-0xd],0x40 Tc2: c6 d5 fa 07 mov BYTE PTR [rbp-0xd],0x40 Tc2: c6 d5 fa 07 mov BYTE PTR [rbp-0xd],0x40 Td2: c6 d5 fa 07 mov BYTE PTR [rbp-0xd],0x40 Td3: d8 d3 d6 e0 00 loa rax, [rbp-0xd],0x40 Td1: d8 d3 d6 e0 00 mov rsi,rax Td1: d8 d3 d6 e0 00 00 mov eax,0x0 Td1: c7 d5 fc 00 00 00 00 mov B15 main-0x455 Td2: c7 d5 fc 00 00 00 00 <td< td=""><td>7b8: c6 45 f3 46 mov BYTE PTR [rbp-bxd], bxd] 7b8: c6 45 f4 56 mov BYTE PTR [rbp-bxd], bxd] 7b8: c6 45 f4 56 mov BYTE PTR [rbp-bxd], bxd] 7b8: c6 45 f5 03 mov BYTE PTR [rbp-bxd], bxd] 7c0: c6 45 f6 00 mov BYTE PTR [rbp-bxd], bxd] 7c4: c6 45 f7 01 mov BYTE PTR [rbp-bxd], bxd] 7c8: c6 45 f7 01 mov BYTE PTR [rbp-bxd], bxd] 7c8: c6 45 f7 01 mov BYTE PTR [rbp-bxd], bxd] 7c8: c6 45 f7 01 mov BYTE PTR [rbp-bxd], bxd] 7c6: c6 45 f7 00 mov BYTE PTR [rbp-bxd], bxd] 7c6: c6 45 f7 00 mov BYTE PTR [rbp-bxd], bxd] 7d8: 48 80 3d 0e 00 00 en rax, [rbp-bx50] # 8d4 < 10_stdin_used+0x4:</td> 7d6: c4 8 d3 3d e0 00 00 00 mov erax, bxd = 10_stdin_used+0x4: 76b: c8 20 fe ff ff call bd 00 00 00 mov wcas, bxd 7f1: 48 d4 5c mov wcas, bx0</td<>	7b8: c6 45 f3 46 mov BYTE PTR [rbp-bxd], bxd] 7b8: c6 45 f4 56 mov BYTE PTR [rbp-bxd], bxd] 7b8: c6 45 f4 56 mov BYTE PTR [rbp-bxd], bxd] 7b8: c6 45 f5 03 mov BYTE PTR [rbp-bxd], bxd] 7c0: c6 45 f6 00 mov BYTE PTR [rbp-bxd], bxd] 7c4: c6 45 f7 01 mov BYTE PTR [rbp-bxd], bxd] 7c8: c6 45 f7 01 mov BYTE PTR [rbp-bxd], bxd] 7c8: c6 45 f7 01 mov BYTE PTR [rbp-bxd], bxd] 7c8: c6 45 f7 01 mov BYTE PTR [rbp-bxd], bxd] 7c6: c6 45 f7 00 mov BYTE PTR [rbp-bxd], bxd] 7c6: c6 45 f7 00 mov BYTE PTR [rbp-bxd], bxd] 7d8: 48 80 3d 0e 00 00 en rax, [rbp-bx50] # 8d4 < 10_stdin_used+0x4:	7ac:	c6 45 f1 45	mov		
7ac: c6 45 f1 45 mov BYTE PTR [rbp-0st], 0x45 $7ba:$ c6 45 f2 5d mov BYTE PTR [rbp-0st], 0x5d $7ba:$ c6 45 f2 5d mov BYTE PTR [rbp-0st], 0x5d $7ba:$ c6 45 f4 56 mov BYTE PTR [rbp-0st], 0x5d $7ba:$ c6 45 f4 56 mov BYTE PTR [rbp-0st], 0x5d $7bc:$ c6 45 f6 00 mov BYTE PTR [rbp-0st], 0x5d $7c4:$ c6 45 f7 01 mov BYTE PTR [rbp-0st], 0x6 $7c4:$ c6 45 f8 01 mov BYTE PTR [rbp-0st], 0x6 $7c4:$ c6 45 f8 01 mov BYTE PTR [rbp-0st], 0x6 $7c4:$ c6 45 f8 02 mov BYTE PTR [rbp-0st], 0x1 $7c4:$ c6 45 f8 02 mov BYTE PTR [rbp-0st], 0x32 $7d4:$ c6 45 f8 02 mov BYTE PTR [rbp-0st], 0x32 $7d4:$ c6 45 f8 02 mov BYTE PTR [rbp-0st], 0x32 $7d4:$ c6 45 f8 02 mov BYTE PTR [rbp-0st], 0x32 $7d4:$ c6 48 03 de e 00 00 mov rat, [rbp-0st], 0x32 $7d7:$ d8 da de e 00 00 mov rat, [rbp-0st], 0x32	Jac: c6 45 f1 45 mov BYTE PTR [rbp-0xc], 0x5d 7b8: c6 45 f2 50 mov BYTE PTR [rbp-0xc], 0x5d 7b8: c6 45 f3 46 mov BYTE PTR [rbp-0xc], 0x5d 7b8: c6 45 f4 56 mov BYTE PTR [rbp-0xc], 0x5d 7b8: c6 45 f4 56 mov BYTE PTR [rbp-0xc], 0x5d 7b8: c6 45 f6 00 mov BYTE PTR [rbp-0xc], 0x5d 7c8: c6 45 f6 00 mov BYTE PTR [rbp-0xc], 0x3d 7c4: c6 45 f7 01 mov BYTE PTR [rbp-0xd], 0x0 7c4: c6 45 f7 01 mov BYTE PTR [rbp-0xd], 0x0 7c4: c6 45 f7 00 mov BYTE PTR [rbp-0xd], 0x0 7c4: c6 45 f7 00 mov BYTE PTR [rbp-0xd], 0x0 7c4: c6 45 f7 00 mov BYTE PTR [rbp-0xd], 0x0 7d4: c6 45 f7 00 mov BYTE PTR [rbp-0xd], 0x0 7d5: c8 40 350 e0 0e0 erax, [rbp-0x50], 0x7 7d6: c4 56 d0 0e0 erax, [rbp-0x50], 0x7 7d6: c4 5 d0 30 e0 00 <td>7a8:</td> <td>c6 45 f0 41</td> <td>mov</td> <td></td>	7a8:	c6 45 f0 41	mov		
7a8: c6 45 f9 41 mov BYTE PTR [rbp-0x10].0x41 7ac: c6 45 f1 45 mov BYTE PTR [rbp-0x1].0x45 7b0: c6 45 f1 45 mov BYTE PTR [rbp-0x1].0x45 7b1: c6 45 f1 45 mov BYTE PTR [rbp-0x1].0x45 7b2: c6 45 f1 40 mov BYTE PTR [rbp-0x1].0x46 7b8: c6 45 f1 63 mov BYTE PTR [rbp-0x1].0x46 7b2: c6 45 f1 63 mov BYTE PTR [rbp-0x1].0x46 7c4: c6 45 f1 69 mov BYTE PTR [rbp-0x3].0x0 7c4: c6 45 f1 69 mov BYTE PTR [rbp-0x3].0x1 7c6: c6 45 f1 69 mov BYTE PTR [rbp-0x3].0x3 7c6: c6 45 f1 637 mov BYTE PTR [rbp-0x3].0x3 7c6: c6 45 f1 637 mov BYTE PTR [rbp-0x3].0x32 7d4: c6 45 f1 637 mov BYTE PTR [rbp-0x3].0x32 7d6: 48 89 c6 nov rs1, rax 7d6: 48 45 3de e0 00 00 loa rd1, [r1p+6xe] # 8d4 <_I0_stdin_used+0x4> 7d6: 48 45 3de e0 00 00 loa rd1, [r1p+6xe] # 8d4 <_I0_stdin_used+0x4> <td>7a8: c6 45 f0 41 mov BYTE PTR [rbp-0x10],0x41 7ac: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b6: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b6: c6 45 f1 60 mov BYTE PTR [rbp-0x10],0x1 7c4: c6 45 f7 00 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 f1 601 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 f1 601 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 f1 607 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 f1 607 mov BYTE PTR [rbp-0x6],0x2 7c6: c6 45 f1 600 lea rax,[rbp-0x50],0x2 7c6: c6 45 f1 600 mov BYTE PTR [rbp-0x6],0x2 7c6: c6 45 f1 600 lea rax,[rbp-0x50],0x2 7c6: c6 45 f1 600 lea rax,[rbp-0x50],0x2 7c6: k8 d3 d2 e0 00 00</td> <td>7a4:</td> <td>c6 45 ef 41</td> <td>mov</td> <td>BYTE PTR [rbp-0x11],0x41</td>	7a8: c6 45 f0 41 mov BYTE PTR [rbp-0x10],0x41 7ac: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b6: c6 45 f1 45 mov BYTE PTR [rbp-0x10],0x45 7b6: c6 45 f1 60 mov BYTE PTR [rbp-0x10],0x1 7c4: c6 45 f7 00 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 f1 601 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 f1 601 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 f1 607 mov BYTE PTR [rbp-0x6],0x1 7c6: c6 45 f1 607 mov BYTE PTR [rbp-0x6],0x2 7c6: c6 45 f1 600 lea rax,[rbp-0x50],0x2 7c6: c6 45 f1 600 mov BYTE PTR [rbp-0x6],0x2 7c6: c6 45 f1 600 lea rax,[rbp-0x50],0x2 7c6: c6 45 f1 600 lea rax,[rbp-0x50],0x2 7c6: k8 d3 d2 e0 00 00	7a4:	c6 45 ef 41	mov	BYTE PTR [rbp-0x11],0x41	
7a8: c6 d5 f0 d1 mov BYTE PTR [rbp-0x10].0x41 7ac: c6 d5 f1 d5 mov BYTE PTR [rbp-0x10].0x41 7b4: c6 d5 f1 d5 mov BYTE PTR [rbp-0x1].0x45 7b4: c6 d5 f1 d5 mov BYTE PTR [rbp-0x1].0x45 7b4: c6 d5 f1 d5 mov BYTE PTR [rbp-0x1].0x45 7b6: c6 d5 f1 d6 mov BYTE PTR [rbp-0x0].0x66 7b6: c6 d5 f1 d6 mov BYTE PTR [rbp-0x0].0x1 7c4: c6 d5 f1 d6 mov BYTE PTR [rbp-0x8].0x1 7c6: c6 d5 f1 d6 mov BYTE PTR [rbp-0x8].0x1 7c6: c6 d5 f1 d6 mov BYTE PTR [rbp-0x8].0x1 7c6: c6 d5 f1 d6 mov BYTE PTR [rbp-0x8].0x2 7c6: c6 d5 f1 d7 mov BYTE PTR [rbp-0x6].0x7 7c6: c6 d5 f1 d7 mov BYTE PTR [rbp-0x6].0x7 7c6: c6 d5 f1 d7 mov BYTE PTR [rbp-0x6].0x7 7c6: c6 d5 f1 d7 mov BYTE PTR [rbp-0x6].0x7 7c6: c6 d5 f1 d7 mov BYTE PTR [rbp-0x6].0x7 7c6: c6 d5 f1 d7	7a8: c6 45 f0 41 mov BYTE PTR [rbp-0x16].0x16] 7ac: c6 45 f1 45 mov BYTE PTR [rbp-0x16].0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x6].0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x6].0x45 7b4: c6 45 f1 45 mov BYTE PTR [rbp-0x6].0x46 7b6: c6 45 f1 50 mov BYTE PTR [rbp-0x6].0x46 7b7: c6 45 f1 60 mov BYTE PTR [rbp-0x6].0x16 7c4: c6 45 f1 60 mov BYTE PTR [rbp-0x6].0x1 7c6: c6 45 f1 60 mov BYTE PTR [rbp-0x6].0x1 7c6: c6 45 f1 70 mov BYTE PTR [rbp-0x6].0x1 7c6: c6 45 f1 80 mov BYTE PTR [rbp-0x6].0x1 7c6: c6 45 f1 80 mov BYTE PTR [rbp-0x6].0x1 7c6: c6 45 f1 80 mov BYTE PTR [rbp-0x6].0x1 7c6: c6 45 f18 02 mov BYTE PTR [rbp-0x6].0x2 7c6: c6 45 f18 02 mov STFE PTR [rbp-0x6].0x2 7c6: c6 45 f18 02	7a0:	c6 45 ee 53	mov	BYTE PTR [rbp-0x12],0x53	

Assembly pro/cons

■ Pro: Produces faster code (in theory)

$\label{eq:assembly pro/cons} Assembly \ pro/cons$

- Pro: Produces faster code (in theory)
- Pro: Very-fine grained control (a.k.a. "I know what I'm doing")

Assembly pro/cons

- Pro: Produces faster code (in theory)
- Pro: Very-fine grained control (a.k.a. "I know what I'm doing")
- Pro: Educational purpose (give the compiler more hints)

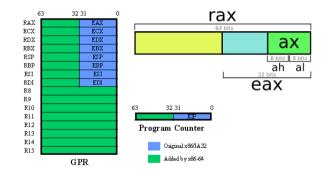
Assembly pro/cons

- Pro: Produces faster code (in theory)
- Pro: Very-fine grained control (a.k.a. "I know what I'm doing")
- Pro: Educational purpose (give the compiler more hints)
- Con: Takes more time, compiler usually knows better than you (in practice)

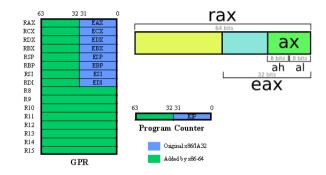
- Pro: Produces faster code (in theory)
- Pro: Very-fine grained control (a.k.a. "I know what I'm doing")
- Pro: Educational purpose (give the compiler more hints)
- Con: Takes more time, compiler usually knows better than you (in practice)
- Con: Need to be frugal w.r.t. variables (limited register count)
- Con: Easy to make non-maintainable spaghetti code

CPU registers

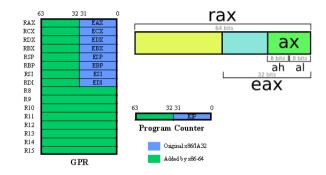
- A (finite) set of internal variables
- Some are general purpose (GP)
- Some are usually (but not always) used by the compiler in certain situations
- Some are always used for a specific purpose (instruction pointer, stack register)



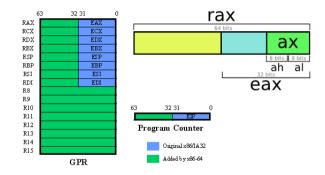
■ RIP: Instruction Pointer



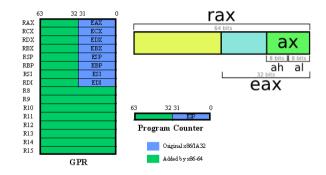
- RIP: Instruction Pointer
- RSP: Stack Pointer; RBP: Base Pointer (usually GP)



- RIP: Instruction Pointer
- RSP: Stack Pointer; RBP: Base Pointer (usually GP)
- R[A,B,C,D]X: GP; RCX (affinity in loop counters)



- RIP: Instruction Pointer
- RSP: Stack Pointer; RBP: Base Pointer (usually GP)
- R[A,B,C,D]X: GP; RCX (affinity in loop counters)
- RDI, RSI: usually GP (affinity in string ops, copy ops)



- RIP: Instruction Pointer
- RSP: Stack Pointer; RBP: Base Pointer (usually GP)
- R[A,B,C,D]X: GP; RCX (affinity in loop counters)
- RDI, RSI: usually GP (affinity in string ops, copy ops)
- Divisions: BYTE (AL), WORD (AX), DWORD (EAX), QWORD (RAX)

MOV RAX, 2021

; rax = 2021

MOV	RAX,	2021
SUB	RAX,	RDX

; rax = 2021 ; rax -= rdx

MOV	RAX,	2021
SUB	RAX,	RDX
AND	RCX,	RBX

; rax = 2021 ; rax -= rdx ; rcx &= rbx

MOV	RAX,	2021
SUB	RAX,	RDX
AND	RCX,	RBX
SHL	RAX,	10

;	rax	= 2021
;	rax	-= rdx
;	rcx	$\mathcal{G}=rbx$
;	rax	<<= 10

MOV	RAX,	2021
SUB	RAX,	RDX
AND	RCX,	RBX
SHL	RAX,	10
SHR	RAX,	10

- ; rax = 2021
- ; rax = rdx
- ; $rcx \ \mathfrak{S} = rbx$
- ; rax <<= 10
- ; rax >>= 10 (sign bit not preserved)

MOV	RAX,	2021
SUB	RAX,	RDX
AND	RCX,	RBX
SHL	RAX,	10
SHR	RAX,	10
SAR	RAX,	10

- ; rax = 2021 ; rax -= rdx
- $: rcx \mathcal{G} = rbx$
- ; rcx ©= rox
- ; rax <<= 10
- ; rax >>= 10 (sign bit not preserved)
- ; rax >>= 10 (sign bit preserved)

MOV	RAX,	2021
SUB	RAX,	RDX
AND	RCX,	RBX
SHL	RAX,	10
SHR	RAX,	10
SAR	RAX,	10
IMUI	RAX	RCX

; rax = 2021
; rax -= rdx
; rcx &= rbx
; rax <<= 10
; rax >>= 10 (sign bit not preserved)
; rax >>= 10 (sign bit preserved)
; rax = rax * rcx

MOV	RAX,	2021
SUB	RAX,	RDX
AND	RCX,	RBX
SHL	RAX,	10
SHR	RAX,	10
SAR	RAX,	10
IMUI	RAX	RCX
IMUI	RCX	

; rax = 2021
; rax -= rdx
; rcx &= rbx
; rax <<= 10
; rax >>= 10 (sign bit not preserved)
; rax >>= 10 (sign bit preserved)
; rax = rax * rcx
; <rdx:rax> = rax * rcx (128 bit mul)

MOV RAX,	2021
SUB RAX,	RDX
AND RCX,	RBX
SHL RAX,	10
SHR RAX,	10
SAR RAX,	10
IMUL RAX	, RCX
IMUL RCX	
XOR RAX,	RAX

;	rax = 2021
;	rax = rdx
;	rcx &= rbx
;	<i>rax</i> <<= 10
;	<pre>rax >>= 10 (sign bit not preserved)</pre>
;	rax >>= 10 (sign bit preserved)
;	rax = rax * rcx
;	<rdx:rax> = rax * rcx (128 bit mul)</rdx:rax>
;	rax ^= rax

- - - -

MOV RAX, 2021	; rax = 2021
SUB RAX, RDX	; $rax = rdx$
AND RCX, RBX	; rcx &= rbx
SHL RAX, 10	; rax <<= 10
SHR RAX, 10	; rax >>= 10 (sign bit not preserved)
SAR RAX, 10	; rax >>= 10 (sign bit preserved)
IMUL RAX, RCX	; $rax = rax * rcx$
IMUL RCX	; <rdx:rax> = rax * rcx (128 bit mul)</rdx:rax>
XOR RAX, RAX	; $rax = rax$
LEA RCX, [RAX * 8 + RBX]	; $rcx = rax * 8 + rbx$

x86-64 instructions: memory

MOV RAX, QWORD PTR [0x123456] ; rax = *(int64_t*) 0x123456 MOV QWORD PTR [0x123456], RAX ; *(int64_t*) 0x123456 = rax

x86-64 instructions: memory

MOV RAX, QWORD PTR [0x123456]; $rax = *(int64_t*) 0x123456$ MOV QWORD PTR [0x123456], RAX ; *(int64_t*) 0x123456 = rax MOV EAX, DWORD PTR [0x123456] : $rax = *(int32_t*) 0x123456$ MOV AL, BYTE PTR [0x123456] ; $al = *(int8_t*) 0x123456$

x86-64 instructions: control flow

JMP 0x1234 JMP [RAX] ; rip = 0x1234 ; rip = *(int64_t) rax

x86-64 instructions: control flow

JMP 0x1234 JMP [RAX] JZ/JE 0xABCD JNZ/JNE 0xABCD ; rip = 0x1234
; rip = *(int64_t) rax
; if (zf) rip = 0xabcd
; if (!zf) rip = 0xabcd

x86-64 flag register

EFLAGS:

(carry parity adjust zero sign trap interrupt direction overflow)

- Carry flag: Addition, Subtraction
- Zero flag: Last operation result was 0
- Sign flag: Last operation result was < 0
- Overflow: Last operation result was > 2^{register_bitcount}

x86-64 flag instructions

TEST RAX, RBX	; _ = rax & rbx; set SF, ZF, PF ; useful when checking for null vals ; and bit masks
CMP RAX, RBX	; _ = rax - rbx ; arithmetic comparisons

x86-64 instructions: stack raison d'etre

- In practice, we cannot use only 16 registers for all variables
- In practice, we cannot use only JMP for function calls

x86-64 instructions: stack raison d'etre

- In practice, we cannot use only 16 registers for all variables
- In practice, we cannot use only JMP for function calls
- To this end, each program is given a slab of blank memory called the stack
- How to use it efficiently?



x86-64 instructions: stack micro-operations

PUSH RAX

; rsp -= 8; *(int64_t*)rsp = rax;

x86-64 instructions: stack micro-operations

PUSH RAX POP RAX

; rsp -= 8; *(int64_t*)rsp = rax; ; rax = *(int64_t*)rsp; rsp += 8

x86-64 instructions: stack micro-operations

PUSH RAX POP RAX	; rsp -= 8; *(int64_t*)rsp = rax; ; rax = *(int64_t*)rsp; rsp += 8
CALL 0x12345	; PUSH RIP; JMP 0x12345
RET	; POP RIP

x86-64 instructions: stack macro-operations

PUSH RBP MOV RBP, RSP SUB RSP, 100

- ; save previous frame base
- ; move frame base to current top
- ; allocate 100 bytes on the stack
- ; "push new stack frame"

x86-64 instructions: stack macro-operations

PUSH RBP MOV RBP, RSP SUB RSP, 100

MOV RBX, [RBP - 0x20]

- ; save previous frame base
- ; move frame base to current top
- ; allocate 100 bytes on the stack
- ; "push new stack frame"

```
; rbx = *(int64_t*)(rbp-0x20)
; use the allocated space for storage
```

x86-64 instructions: stack macro-operations

PUSH RBP		
MOV	RBP,	RSP
SUB	RSP,	100

MOV RBX, [RBP - 0x20]

LEAVE

; save previous frame base ; move frame base to current top ; allocate 100 bytes on the stack ; "push new stack frame"

```
; rbx = *(int64_t*)(rbp-0x20)
; use the allocated space for storage
```

; MOV RSP, RBP ; POP RBP ; "pop current stack frame"

x86-64 instructions: conventions

In order to use software modules (libraries, objects, etc) a standard must be set. Why?

- In order to use software modules (libraries, objects, etc) a standard must be set. Why?
- How do you pass parameters to external functions? Memory? Stack? Registers?

- In order to use software modules (libraries, objects, etc) a standard must be set. Why?
- How do you pass parameters to external functions? Memory? Stack? Registers?
- Calling conventions are used: cdecl, stdcall, fastcall.
- On 32 bit systems, parameters are passed on the stack, return in EAX
- On 64 bit systems, parameters are passed using registers, return in RAX
- Memorize this: 'RDI, RSI, RDX, RCX, R8, R9 (Linux)'

x86-64 instructions: syscalls

- In order to cross the application OS limit, syscalls are needed
- File operations: read/write/close/open/create/remove
- Sleep, Select, Yield, Fork, Kill, GetTime
- Allocate/Release Memory
- Socket/Networking Operations
- IPC communication

```
MOV RAX, 0x2
MOV RDI, [RSP + 0x10]
SYSCALL
```

- ; Choose syscall number 2 (open) ; Set first argument to some stack value
 - ; Invoke kernel functionality

- As a RE, writing ASM code by hand is not needed very often
- Reading ASM code is maybe 10% of the work
- However, knowing the basics is absolutely crucial and can be learned fast

Practice

- Any Questions?
- http://pwnthybytes.ro/unibuc_re/02-lab.html