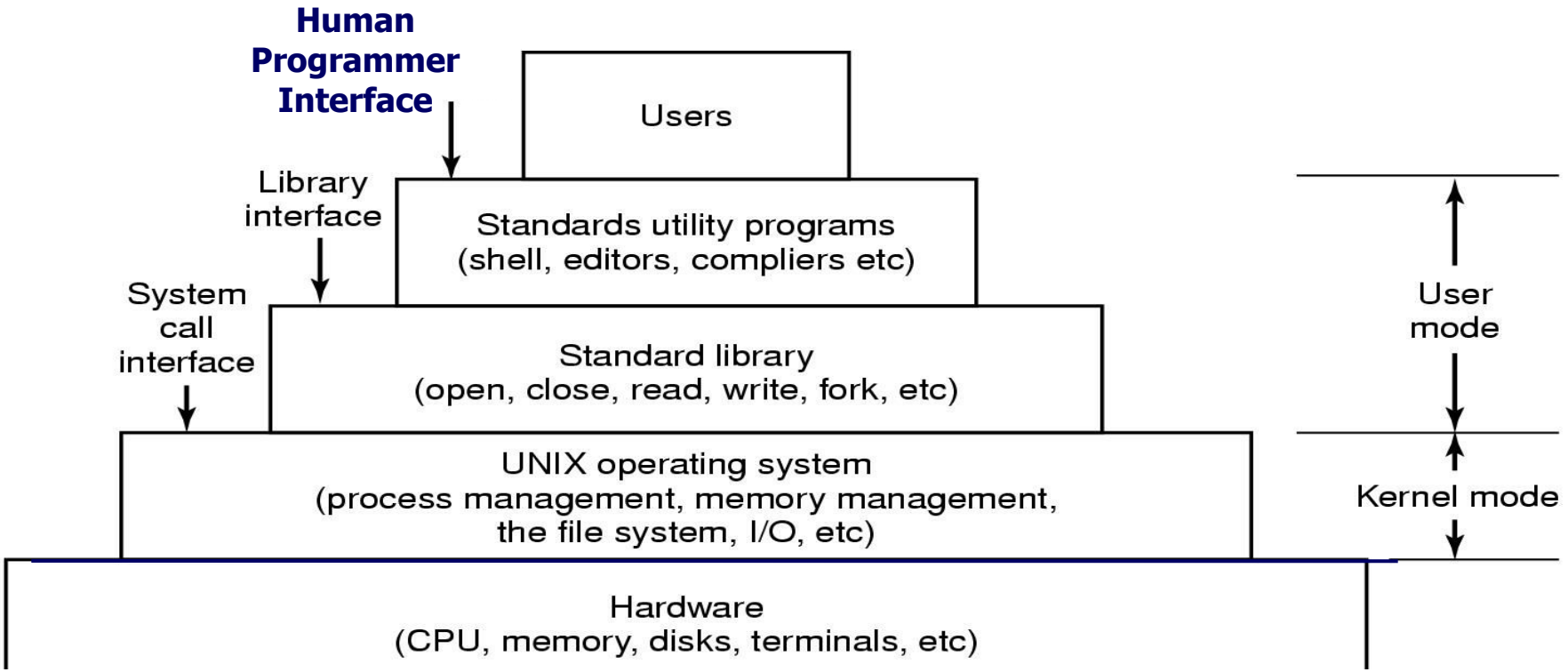
Reverse Engineering Station

Getting your first look in the binary box

What is Reverse Engineering?
&
Install:

https://github.com/anaaktge/talks/blob/main/re_station.zip

A high level software system view (on Ubuntu)



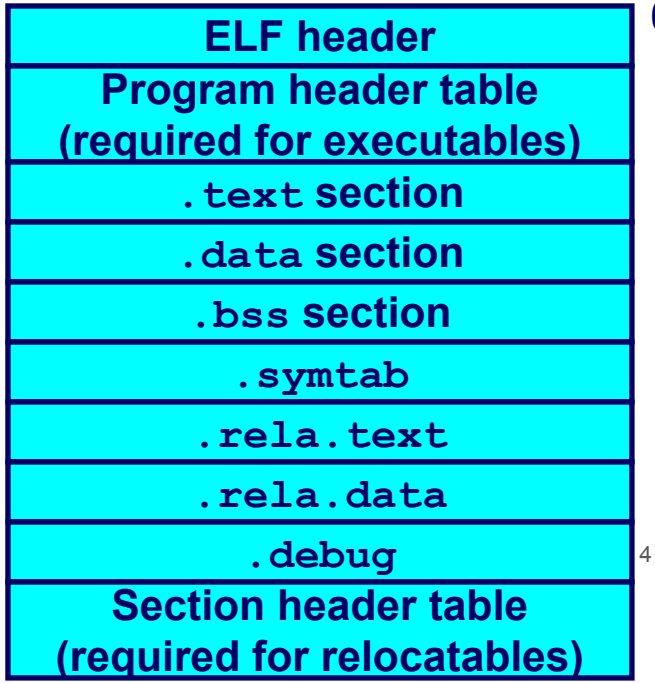
Executables: ELF Object File Format

ELF header
- Magic number, type (.o, exec, .so), machine, byte ordering, etc.

Program header table
- Page size, addresses of memory segments (sections), segment sizes.

.text section
- Code (machine instructions)

.data section
- Initialized (static) global data



ASCII, each character is really a #. Dec 97 is ‘a’

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

gdb: **G**nu **DeB**ugger

Debuggers allow you to examine and control program execution

- Most debuggers provide the same functionality
- Can also be used for RE

To compile a program for optimal use with gdb, use the ‘-g’ compiler switch

- Allows for source code tracing
- I’ve done this for you
- Lots of gdb cheatsheets out there

Start by $ gdb myexecutablename

Exit by “quit”

Controlling program execution for fun and profit

run

- Starts the program (lets you do set up before starting)

step/stepi

- Execute until a different source line reached (step into calls)

next

- Execute until next source line reached, proceeding through subroutine calls.

continue

- Resume program execution until signal or breakpoint.

break

- Set and delete breakpoints at particular lines of code

Viewing & Displaying

print

- Print expression
- Basic usage: print $rsp
- print /x addr
- ‘/x’ says to print in hex. See
  - Same as examine memory address command (x)

x (examine)

- Examine memory
- x /s $rax => print the string at address contained in %rax
- x /32xw 0x4006b7 => print 32 words at 0x4006b7 in hexadecimal

Info (Get information)

- ‘info’ alone prints a list of info commands
- ‘info br’ : a table of all breakpoints and watchpoints
- ‘info reg’ : the machine registers

layout asm

- Show asm window and cmd window
- focus asm/cmd lets you move focus between the two windows in your terminal

Binary Analysis (stage0 and stage1)

Stage0

- File stage0
  - What is this file? Can we run it?
- ./stage0
  - // run the exe whats it do?
  - What is the key operation?
- strings -n6 stage0
  - // min string length
- Gdb stage0
  - layout asm
  - focus asm/cmd
  - break *main+367 // breaks before line 367
  - x/s 0x2188 //looks at addr as a string
  - x/s $rsi // looks at rsi as string
  - quit

Stage1

- file stage1
- Strings -n10 stage1
- $ readelf -p .rodata stage1
- Gdb stage1
  - layout asm
  - focus asm/cmd
  - break *main+367 // breaks before line 367
  - info break // list all breakpoints
  - x/s 0x2188 //looks at addr as a string
  - x/s $rsi // looks at rsi as string
  - quit

Hints for Stage 0 & 1

- Strncmp is a string compare function
- Scanf is a common way to get input
- What is the program doing and where might we see the password show up?

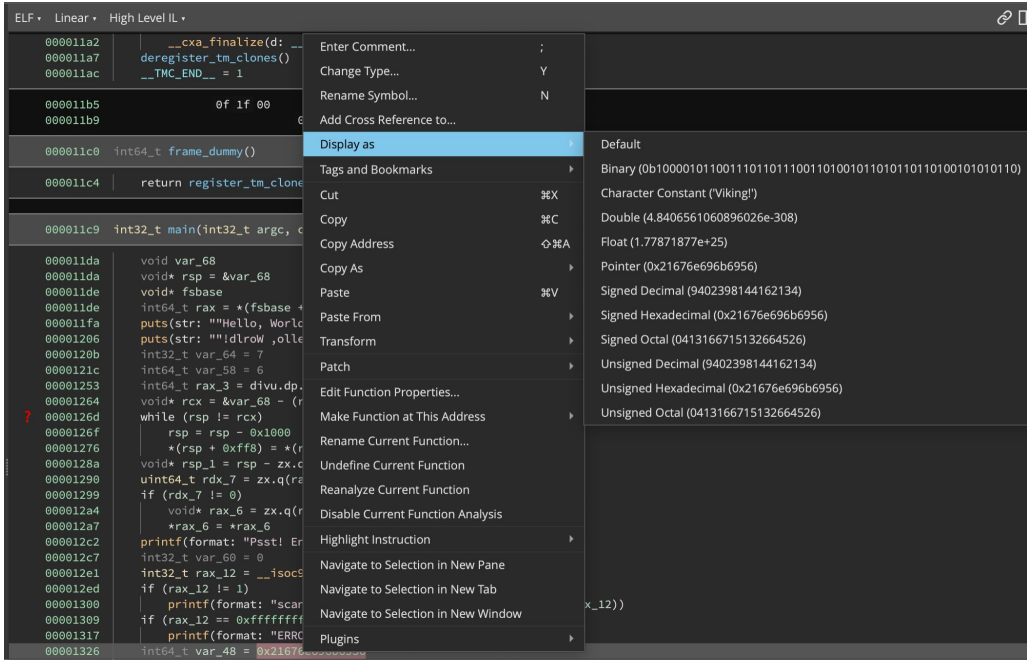
Stage2: Sometimes the password isn’t in the binary at all

- Exclusive OR (XOR) encryption: it is symmetric
  - Key XOR plaintext = ciphertext
  - Key XOR ciphertext = plaintext
- Intel x86 has a special instructuin
- Use strings, gdb, etc to find the original password (plaintext) and the key
- Then use an online tool like https://gchq.github.io/CyberChef to compute
- There are other ways to do this

Pro Tools (IDA, Ghidra, Binary Ninja, r2, etc) Stage0

```
000011c9  int32_t main(int32_t argc, char** argv, char** envp)

000011da      void var_68
000011da      void* rsp = &var_68
000011de      void* fsbase
000011de      int64_t rax = *(fsbase + 0x28)
000011fa      puts(str: ""Hello, World!" is traditionally…"  )
00001206      puts(str: ""!dlroW ,olleH" might be a bette…"  )
0000120b      int32_t var_64 = 7
0000121c      int64_t var_58 = 6
00001253      int64_t rax_3 = divu.dp.q(0:0x16, 0x10) * 0x10
00001264      void* rcx = &var_68 - (rax_3 & 0xfffffffffffff000)
0000126d  ?   while (rsp != rcx)
0000126f          rsp = rsp - 0x1000
00001276          *(rsp + 0xff8) = *(rsp + 0xff8)
0000128a      void* rsp_1 = rsp - zx.q(rax_3.d & 0xfff)
00001290      uint64_t rdx_7 = zx.q(rax_3.d & 0xfff)
00001299      if (rdx_7 != 0)
000012a4          void* rax_6 = zx.q(rax_3.d & 0xfff) - 8 + rsp_1
000012a7          *rax_6 = *rax_6
000012c2      printf(format: "Psst! Enter secret code: ", 0x10, rdx_7, rcx)
000012c7      int32_t var_60 = 0
000012e1      int32_t rax_12 = __isoc99_scanf(format: &data_20c2, rsp_1)
000012ed      if (rax_12 != 1)
00001300          printf(format: "scanf did not return expected va…"  , zx.q(rax_12))
00001309      if (rax_12 == 0xffffffff)
00001317          printf(format: "ERROR! Something has gone terrib…"  )
00001326      int64_t var_48 = 0x21676e696b6956
0000134a      if (strncmp(rsp_1, &var_48, 7) != 0)
00001361          puts(str: "That's not the secret code")
00001353      else
00001353          puts(str: "You know the secret code! Nice R…"  )
0000137b      if ((rax ^ *(fsbase + 0x28)) == 0)
00001390          return 0
0000137d      __stack_chk_fail()
0000137d      noreturn
```



stage0 in binary ninja decompiler