Computer Science 4271
Fall 2022
Midterm exam 1 (corrected)
October 18th, 2022
Time Limit: 75 minutes, 4:00pm-5:15pm

- Before starting the exam, you can fill out your name and other information of this page, but don't open the exam until you are directed to start. Don't put any of your answers on this page.
- This exam contains 6 pages (including this cover page) and 4 questions. Once we tell you to start, please check that no pages are missing.
- You may use any textbooks, notes, or printouts you wish during the exam, but you may not use any electronic devices: no calculators, smart phones, laptops, etc.
- You may ask clarifying questions of the instructor or TAs, but no communication with other students is allowed during the exam.
- Please read all questions carefully before answering them. Remember that we can only grade what you write on the exam, so it's in your interest to show your work and explain your thinking.
- By signing below you certify that you agree to follow the rules of the exam, and that the answers on this exam are your own work only.

The exam will end promptly at 5:15pm. Good luck!

Your name (print): ______________________________

Your UMN email/X.500: ______________________________`@umn.edu`

Number of rows ahead of you: ____________ Number of seats to your left: ____________

Sign and date: ______________________________

Question	Points	Score
1	20	
2	24	
3	28	
4	28	
Total:	100	

1. (20 points) Matching definitions and concepts. Fill in each blank with the letter of the corresponding answer. Each answer is used exactly once.

   (a) ____ A sequence of instructions ending in a return

   (b) ____ Falsely denying that an action took place

   (c) ____ A bit used by AMD to implement W $\oplus$ X

   (d) ____ Freedom from unauthorized data modification

   (e) ____ A defense that limits attackers to code reuse

   (f) ____ A function to change memory permissions

   (g) ____ A function to copy a given number of bytes

   (h) ____ Padding code for shellcode

   (i) ____ A function to copy bytes up to a null terminator

   (j) ____ Choosing random base addresses for memory regions

A. ASLR B. gadget C. integrity D. `memcpy` E. `mprotect` F. NOP sled G. NX H. repudiation I. `strcpy` J. W $\oplus$ X

Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL	16	10	DLE	32	20		48	30	0	64	40	@	80	50	P	96	60	‘	112	70	p
1	01	SOH	17	11	DC1	33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
2	02	STX	18	12	DC2	34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
3	03	ETX	19	13	DC3	35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
4	04	EOT	20	14	DC4	36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
5	05	ENQ	21	15	NAK	37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
6	06	ACK	22	16	SYN	38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
7	07	BEL	23	17	ETB	39	27	’	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
8	08	BS	24	18	CAN	40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
9	09	HT	25	19	EM	41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
10	0A	LF	26	1A	SUB	42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
11	0B	VT	27	1B	ESC	43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
12	0C	FF	28	1C	FS	44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	|
13	0D	CR	29	1D	GS	45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
14	0E	SO	30	1E	RS	46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
15	0F	SI	31	1F	US	47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

2. (24 points) STRIDE classification.

   In each of the following scenarios, we describe 6 threats, one each from the STRIDE classification of spoofing, tampering, repudiation, information disclosure, denial of service, and escalation of privilege. Write the letters S, T, R, I, D, and E in the appropriate order in the blanks according to which type of threat each is. In our answer, each of the letters is used exactly once in each scenario.

   Optionally, there is also one blank next to a blank space for each scenario. If you don't like our examples, you can write one new threat and STRIDE classification of your own in this space, and if it's a good example, it can compensate for one other threat in the scenario we marked wrong.

   In each scenario, people whose names start with A are attackers, and those whose names start with V are victims.

   (a) In-person voting on election day. Most of these can work whether the voting is on paper or electronic.

   ____ Alice votes once, comes back later, and votes again claiming it is her first time

   ____ Alice pulls the fire alarm and the polling place is evacuated

   ____ Alice changes Vicki's mayoral vote from Bob to Charlie

   ____ Alice is a regular voter, but gets the election judge's keyring

   ____ Alice gets a list of everyone who voted for Bob for mayor

   ____ Alice uses a fake ID to cast a ballot under Vicki's name

   ____

   (b) Oliver's online olive oil electronic commerce website. Adam is a customer, Alex is a competitor, and Arnold is just a vandal.

   ____ Arnold discovers the configuration page `admin.php` is not password protected

   ____ Alex files a trademark lawsuit to get Oliver's web hosting taken down

   ____ Adam orders rancid olive oil to be delivered to Victor

   ____ Adam gets a delivery, but claims it was lost and asks for a refund

   ____ Adam gets Victor's credit card number

   ____ Arnold changes the product descriptions to add awful puns

   ____

3. (28 points) Multiplication and memory allocation.

   Consider the following C function which attempts to allocate memory for, and then read in, a number of integers controlled by the argument `num_ints`. Use it to answer the questions on the following page.

```
int *alloc_and_read(unsigned char num_ints) {
    unsigned char size = sizeof(int) * num_ints;
    if (size < num_ints) { /* overflow check */
        fprintf(stderr, "Uh-oh, overflow!\n");
        exit(1);
    }
    int *ary = malloc(size);
    if (!ary) {
        fprintf(stderr, "Allocation failed\n");
        exit(1);
    }
    int i;
    for (i = 0; i < num_ints; i++)
        ary[i] = read_int();
    return ary;
}
```

   Assume that `sizeof(int)` is 4, as it is on x86-64. We'll use the variable n to represent the value of `num_ints`, which is between 0 and 255 in decimal (`0x00` to `0xff` in hex). Because the variable `size` is also only an `unsigned char`, its value is also limited to between 0 and 255. Specifically, the value stored in `size` will be $(4 \cdot n) \bmod 256$. The "mod 256" operation is also the same as discarding all but the two lowest hex digits, or all but the 8 lowest bits, of a number.

   You can use decimal, hexadecimal, or binary in your answers, but to keep them distinct, write hexadecimal numbers with a `0x` prefix and binary numbers with an `0b` prefix. It is enough to write just the formula or number if it is correct, but a short explanation of your answer may help us give partial credit. You don't need to simplify formulas.

(a) Write a mathematical formula, in terms of the variable n, which will be true for those values of n that cause the message "Uh-oh, overflow!" to be printed.

(b) Pretend for a moment that the `if` statement labeled "overflow check" were not present. Write a mathematical formula, in terms of the variable n, which will be true for those values of n where the function will write beyond the area of memory allocated for `ary`.

(c) Give one specific value for n that will cause the function with the overflow check to write beyond the area of memory allocated for `ary`. This will need to be a value for which the formula in part (b) is true, while the formula in part (a) is false; this also implies that those formulas should be different.

4. (28 points) Overwriting an address.

   The following function from a Linux/x86-64 program has a buffer overflow vulnerability. Depending on the contents of the string `attack`, which we assume is under the control of an attacker, the return address of the function `func` might be overwritten. The program is compiled without PIE or stack canaries.

   Below are excerpts of the relevant code in C and assembly language.

```
void func(char *attack) {
    char buf[8];
    strcpy(buf, attack);
}
```

```
    sub     $0x10, %rsp
    mov     %rdi, %rsi
    lea     0x8(%rsp), %rdi
    call    strcpy
    add     $0x10, %rsp
    ret
```

   The normal return address of the function is `0x4011c0`. Assume that in order to start a code reuse attack, the attacker wants to change the return address to `0x401171`.

   In the left column, below, are 7 possible contents for the string `attack` passed to the function, written using the same rules as for string constants in C. A sequence of `\x` followed by two hex digits represents a single character (byte) whose numeric value is given by the following two hex digits. For instance `\x2a` represents the byte with value `0x2a`, decimal 42.

   In the right column are 5 different numeric values for the return address at the time when the function returns. Write the letter of an entry in the right column in the blank on the left to match an attempted attack with the effect it has on the return address. Each answer might be used once, more than once, or not at all.

   (a) ____ `AAAA`

   (b) ____ `AAAAAAA`

   (c) ____ `AAAAAAAAB`

   (d) ____ `AAAAAAAA\x71`

   (e) ____ `AAAAAAAA\x71\x11\x40`

   (f) ____ `AAAAAAAAq\x11@`

   (g) ____ `AAAAAAAA\x71\x11\x40\x00GGGG`

   A. `0x4747474700401171`

   B. `0x0000000000400042`

   C. `0x0000000000400071`

   D. `0x00000000004011c0`

   E. `0x0000000000401171`