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: ____________________________________________

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
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. $W \oplus X$
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.

```c
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 · n) mod 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 \texttt{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 \texttt{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 \texttt{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.

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

```assembly
1: sub $0x10, %rsp
2: mov %rdi, %rsi
3: lea 0x8(%rsp), %rdi
4: call strcpy
5: add $0x10, %rsp
6: 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) ___ AAAAAAAA
(c) ___ AAAAAAAAAB
(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