# CIS 4930: Secure Coding [Fall 2018]
# Test IV

**NAME: _____**

**Instructions:**

1) This test is 7 pages in length.

2) You have 40 minutes to complete and turn in this test.

3) Short answer questions include a guideline for how many sentences to write. Respond in complete English sentences. Responses will be graded as described on the syllabus.

4) This test is closed books, notes, papers, friends, neighbors, etc.

5) Use the backs of pages in this test packet for scratch work. If you write more than a final answer in the area next to a question, circle your final answer.

1. [6 points]
In class we discussed three authentication factors. What were they? Provide an example of each of these factors. [1-3 sentences]

2. [8 points]
Describe four of the software-security tradeoffs discussed in class. [1 sentence]

3. [4 points]
What is the output of `printf("%o", 0x61E)`, where 0x61E means the hexadecimal value 61E?

4. [17 points]
(a) How did we define a policy, formally? Respond in terms of traces. [1 sentence]

(b) Using set builder notation, define a non-safety liveness property.

(c) Using set builder notation, define a non-liveness, safety property.

(d) Using set builder notation, define a non-safety, non-liveness property.

5. [40 points]
Consider the following code. Assume all needed #include directives are also present, and recall that heap memory is freed/deallocated in C by calling the free function (or a variant thereof).

```
1    char *x, *y;
2
3    void h(char *a) {
4       /* Allocates one character of space and returns a pointer to it. */
5       x = malloc(2);
6       x[0] = 'X';
7    }
8
9    int f() {
10       gets(y);
11       return 4;
12    }
13
14    void g(char *a, char *b) {
15       y = malloc(2);
16       y[0] = 'Z';
17       h(a);
18    }
19
20    void i(char *c) {
21       x = malloc(2);
22       x[0] = 'Y';
23       h(c);
24       f();
25    }
26
27    int main (int argc, char *argv[]) {
28       char *a, *c;
29       g(a, c);
30       i(c);
31
32       return 0;
33    }
```

(a) Draw a representation of the program memory segments (including their contents when known) right before Line 29 is executed, at the level of detail shown in class.

(b) Draw a representation of the program memory segments (including their contents when known) right before Line 10 is executed, at the level of detail shown in class.

(c) Can an attacker make this program violate memory safety?

(d) Re-draw a representation of the program stack right before Line 10 is executed, but this time assuming StackGuard is used.

(e) Again assuming StackGuard is used, can an attacker still make this program violate memory safety?

6. [25 points]
Consider the following code. Assume a 16-bit architecture, that all needed #include directives are also present, and that input is allocated on the heap.

```
1    int f(char *input) {
2
       char a[128];
4
5            snprintf(a, sizeof(a), input);
6
7            a[sizeof(a) - 1] = 0;
8            printf("%s\n", a);
9
10           return 0;
11   }
```

(a) Draw a representation of the program memory segments (including their contents when known) right before the snprintf is executed, at the level of detail shown in class. Assume an optimized layout of memory, as discussed in class, where snprintf is not given its own frame.

(b) Assuming input is “abcd%p%n”, describe what happens when running the snprintf, at the level of detail discussed in class.