CSci 4271: Introduction to Computer Security

Problem Set 1 **due: Friday February 17th, 2023**

Ground Rules. This is an individual assignment. You may discuss the concepts behind these questions with other students, but you should formulate your answers individually and your answers must be entirely your own writing. You may use any paper or written online source that you find relevant to the questions but you **must** explicitly reference any source you use besides the lectures. An electronic PDF copy of your solution should be submitted via a link on the course Canvas page by 11:59pm on Friday, February 17th. We strongly recommend that you type your answers electronically rather than writing them on paper and scanning them: we may not be able to give you all the credit you deserve if we can't read your handwriting.

Before we started using Moodle and then Canvas, classes using the CSE Labs had an internally developed system that allowed students to see information about grades electronically. Here are some basics of the system worked:

- There was a configuration program that could be executed by course instructors and TAs (collectively "graders") to set up the system for a new course, or to directly edit the tables containing grades.
- There was a command-line program executed on CSE Labs machines by graders which interpreted a text input file in which each line was an instruction to add to or modify a student's grade on an assignment, or set or change the total score and weighting of an assignment.
- There was a web interface that allowed each student taking a course to see their scores for assignments in a particular offering of a course. Each course offering had a unique identifying number, which was passed by URL in the web interface. Students could only access the web interface after logging in using a CSE Labs login name and password (which at the time were separate from the UMN ID).

For the next three questions, imagine that the department has decided to implement a new similar system for future use, and you've been asked to help design and implement it securely.

1. Threat model assets and attackers. (15 pts) As a first step in threat modeling, think about what the overall security goals for this system should be. Specifically: what are the assets that need to be protected? Who might be an attacker against the system, and what might their motivations be? Note that the way that an attack is useful to an attacker might be basically the same as the reason it is detrimental to the victim, or these might be different. What are the high-level security policies that should hold? On the other hand, what kinds of threats do not need to be considered?

2. Threat model data flow diagram. (25 pts) Next, think about the software architecture you would build to support this grade viewing functionality, and how users would interact with different parts of the system when carrying out the supported features. Based on this, draw a data-flow diagram showing the users, the software components, the data flows between them, and any trust boundaries. You should aim to include enough details that your data flow diagram shows 5-10 software components. (Users are not software components.)

Point out a place in your data-flow diagram where the data flow edges and threat model boundaries you've drawn represent a design that is easier to secure. Specifically compare your design to another design that would still implement the same visible behavior, but which would look different in the data flow diagram, and would be harder to build securely.

3. Threat model STRIDE threats. (15 pts) Apply the STRIDE threat taxonomy to the elements and data flows you diagrammed in the previous question, and create a table enumerating a number of possible threats. For each threat, your table should mention which of the STRIDE categories it falls under, and include a short description, a reference to which elements of the data-flow diagram the attack targets, and a short description of possible mitigations. A good answer should mention at least 10 different threats.

4. STRIDE in another context. (20 pts) A paper written by Florina Almenárez-Mendoza et al, assessing the some of the security properties of fitness tracking devices, is available from its publisher's web site at the this URL: `https://www.mdpi.com/2504-3900/2/19/1235`

Skim sections 1-3 of the paper for context, and then carefully read section 4 which gives a description of vulnerabilities the authors found in some of the devices they studied.

Think about how these vulnerabilities related to the STRIDE taxonomy. For each of the 6 STRIDE threat classes, give one example, if any, of a possible attack found to be possible by the paper which you would classify in that category. You should be able to find examples of at least 4 out of the 6 threat classes.

5. A heap-related vulnerability. (25 pts) Below we've shown the source code for a C program that has a vulnerability related to how it uses `malloc` and `free`. Two parts shown after the source code will ask you about the vulnerability and how it can be attacked.

```
typedef void (*toes_func)(void);
void even_toes(void) { printf("with even-toed hoofs"); }
void odd_toes(void)  { printf("with odd-toed hoofs"); }
void shellcode(void) { /* Assume shellcode() has the address 0x4012ce */
    printf("Uh-oh, this looks like some sort of attack\n");
    exit(42); }

struct herbivore { struct herbivore *next; toes_func func; };
struct herbivore *herbivore_list = 0;
struct carnivore { struct carnivore *next; long num_teeth; };
struct carnivore *carnivore_list = 0;

#define NUM_ANIMALS 256
void *animals_by_num[NUM_ANIMALS]; /* Starts as all null pointers */

void new_herbivore(long x) {
    int loc = x & (NUM_ANIMALS - 1);
    struct herbivore *hp = malloc(sizeof(struct herbivore));
    printf("Allocating herbivore at %p\n", hp);
    if (!hp) exit(1);
    hp->func = ((x & 1) ? odd_toes : even_toes);
    hp->next = herbivore_list; herbivore_list = hp;
    animals_by_num[loc] = hp;
}
void new_carnivore(long x) {
    int loc = x & (NUM_ANIMALS - 1);
    struct carnivore *cp = malloc(sizeof(struct carnivore));
    printf("Allocating carnivore at %p\n", cp);
    if (!cp) exit(1);
    cp->num_teeth = x;
    cp->next = carnivore_list; carnivore_list = cp;
    animals_by_num[loc] = cp;
}
void release_animal(long x) {
    int loc = x & (NUM_ANIMALS - 1);
    printf("Releasing animal #%d at %p\n", loc, animals_by_num[loc]);
    if (!animals_by_num[loc]) {
        fprintf(stderr, "Attempt to release non-existant animal\n");
        exit(1); }
    free(animals_by_num[loc]);
    animals_by_num[loc] = 0;
}
```

```
void list_animals(void) {
    struct herbivore *hp;
    struct carnivore *cp;
    for (hp = herbivore_list; hp; hp = hp->next) {
        printf("A herbivore "); (hp->func)(); printf("\n"); }
    for (cp = carnivore_list; cp; cp = cp->next)
        printf("A carnivore with %ld teeth\n", cp->num_teeth);
}
void syntax_error(void) { fprintf(stderr, "Unrecognized syntax\n");
                          exit(1); }

int main(int argc, char **argv) {
    for (;;) {
        int c = getchar();
        long x;
        while (isspace(c)) c = getchar();
        if (c == EOF) return 0;
        switch (c) {
        case 'h':
            if (scanf(" %li", &x) != 1) syntax_error();
            new_herbivore(x); break;
        case 'c':
            if (scanf(" %li", &x) != 1) syntax_error();
            new_carnivore(x); break;
        case 'r':
            if (scanf(" %li", &x) != 1) syntax_error();
            release_animal(x); break;
        case 'l': list_animals(); break;
        case 'q':
            return 0;
        default:
            fprintf(stderr, "Unrecognized command %c\n", c); exit(1);
        }
    }
}
```

(a) (10 pts) Describe the bug in this code, both with a general phrase describing a type of heap-usage bugs, and by explaining the particular instance of the problem as it appears in this program, with reference to particular data structures and parts of the code.

Hint: consider what happens with the program is given the input `h 0 r 0 l q`, which leads to the functions `new_herbivore`, `release_animal`, and `list_animals` being called. You might find it helpful to examine what's happening with a debugger or with a memory checking tool like Valgrind.

(b) (15 pts) How could an attacker take advantage of the bug from part (a) in order to take control of the program? Suppose that an attacker is supplying the input commands to this

program, and their goal is to cause the execution of the function `shellcode`, which stands in for some shellcode that the program wouldn't normally execute. Given an example of an input that would be a successful attack of this sort, and explain why it works.

You can assume that the `shellcode` function has the address shown in a comment, that the platform is x86-64, and that the `malloc` library reuses the space for objects of the same size in a last-in-first-out order (like a stack). You should be able to answer this question without having to run this code, but if you would like to experiment with a real program, we have supplied the source code on the course web site.

Hint: in lecture we discussed in a general way how this kind of vulnerability could be useful for an attacker, but we didn't walk through the exact steps that an attack would use. We've tried to make this example simple enough in terms of the attacker's goal (what program feature could affect control flow?) and the set of available operations, that you can puzzle it out. If you're feeling stumped, recall that you are allowed to use external resources in problem sets in this course, so it would OK to do a web search for "how to exploit ⟨vulnerability name⟩" as long as you give credit to sources you got good ideas from.