COSC 366: Introduction to Cybersecurity

Written Assignment 2 — Fall 2022

Ground Rules. Work must be typeset (not handwritten and scanned) and submitted by 23:59:59 of the due date. You do not need to provide code or visual diagrams for any question, but are welcome to if it helps you explain your answers.

1. Overflowing buffers. Let's examine a few proposed solutions to the problem of stack smashing.

(a) *Reversing the Stack.* It is not uncommon for someone to propose to eliminate stack-smashing attacks by reversing the direction of stack growth, so that the stack grows in the same direction as buffer addresses. With this arrangement, as long as the return address comes earlier in the stack frame than local variables, a procedure can never overwrite its own return address by manipulating only local variables. Consider the function `func` defined as:

```
void func(char *str) {
  char buf[128];
  strcpy(buf, str);
  do_something();
  return;
}
```

Describe an attack that will exploit this function's use of `strcpy` to hijack control flow, even if the stack grows in the same direction as buffers. A good description will show the contents of the stack at the important points of the attack and walk us through the control flow under the attack.

(b) Properly placed *canaries* can protect return addresses and frame pointers from being modified by simple buffer overflows. Other kinds of stack-based buffer overflows can still be possible, however. For example, imagine the following code that is supposed to ensure that the system has only 100 active connections at any time.

```
int accept_connection(char *user, int *total_connections) {
  int tmp = *total_connections;
  char buf[128];
  strcpy(buf, user);
  if (tmp < 100) {
    strcat(buf, ":Y");
    *total_connections++;
    tmp = 1;
  } else {
    strcat(buf, ":N");
    tmp = 0;
  }
```

```
  printf("%s\n", buf);
  return tmp;
}
```

Describe an input that overflows `buf` so that even if there are 100 connections, and even if canaries are used, the system will accept additional connections. Your input should not write beyond the local variables of `accept_connection`. (Hint: different inputs are required for different endian-nesses)

2. Sharing files in Unix. Alice wants to be able to share read and write access to some of her files (on a unix system) with dynamically changing sets of users. Since she is not root, she can't just construct new groups for each file, nor can she turn on the optional access control feature available on some Linux systems. So she decides to write setuid programs that will implement access control for her friends. Alice designs two setuid, world-executable programs, `alice-write` and `alice-read` (e.g., programs that anyone can run as `alice`) that work as follows:

- `./alice-write IN OUT`: first checks a permission file written by Alice to make sure that the ruid of the process (the calling user) is allowed to write to the file `out`. If so, then the program reads the file `in` and writes it over `out`.
- `./alice-read IN OUT`: first checks a permission file written by Alice to make sure that the calling user is allowed to read the file `in`. If so, the the program reads `in` and writes it to the file `out`.

Assume Alice has been careful in her implementation, i.e., there are no buffer overflows in `alice-read` and `alice-write`, the permission file is properly protected (uniquely named in the program and set to permission 0400), the programs accept only file paths listed in the permissions file, and permissions on Alice's files are preserved.

(a) Can you find any (≥ 1) potential security problems with this approach? Describe them, no code/visuals required. (e.g., suppose Bob can read and write some of Alice's files but not others; can he use `alice-write` and `alice-read` to gain access to files he shouldn't? Are there potential attacks that could allow third parties to read/write Alice's files?)

(b) How could you change interface (e.g., what is passed to the programs) and/or implementation (e.g., the description of the programs) of `alice-write` and `alice-read` to avoid your attacks? Describe only, no code necessary.