CSci 4271W Developing Secure Software Systems

Homework 2 **Due: February 18th, 2025**

Ground Rules. You may choose to complete these exercises in a group of up to three students. Each group should turn in **one** copy with the names of all group members on it. You may use any source you can find to help with this assignment but you **must** explicitly reference any source you use besides the lecture notes or assigned readings. No answers should come from people outside your group, or from AI tools like ChatGPT. If you use an AI tool to revise your writing, save a copy of the first draft you wrote yourself to provide it was originally your work. Electronically typeset copies of your solution should be submitted via Gradescope by 11:59pm on February 18th, 2025.

1. **Memory Corruption.** For each of the following memory corruption scenarios, describe how it can lead to at least two different STRIDE threats, and which of the mitigations discussed in Lecture 7 can prevent that threat, if any. (e.g. you might claim that repudiation threats due to format-string attacks are prevented by stack cookies. *but I wouldn't*) Briefly describe how the mitigation prevents or limits the threat:

   a. Buffer overflow.
   b. Use after free.
   c. Format string.

Defensive Programming. Let's practice finding "bad programming practices" that could lead to exploits. Below are three buggy programs. For each program, (i) describe at least four bad things that could happen when running this function in situations that the programmer probably didn't think of, including the programming mistake, the problematic situation, and the bad outcome, and (ii) Provide a safer implementation for this program (fragment). Note that your new implementation might have to behave differently than the old implementation in some cases; it is up to you to pick an interpretation and explain your rationale.

2. This program reads an input of the form `username=user` from standard input and then uses the `last` unix command to see if the user has logged in recently. The output is an HTML document listing the most recent (up to three) times the user has logged in. If the environment variable `OUTPUTFILE` is set, it will try to find an unused filename starting with `OUTPUTFILE` to write the output to.

```
import os, datetime, os.path
inputline = input()
(_,username) = inputline.split('=')

times = list(os.popen("last -1000 | grep " + username))
of = os.getenv("OUTPUTFILE") # should the output go to a file?
counter = 0
ofname = of
if of is not None and os.path.exists(ofname):
  counter += 1
  ofname = "{}.{}".format(of,counter)
if of is not None:
  sys.stdout = open(ofname,"w")
print("<HTML><BODY>\n")
print("<h4>Last up to 3 times for user " + username + "</h4>\n")
print("<table>\n")
for i in range(min(3,len(times))):
  print("<tr>")
  for s in times[i].split()[2:]
    print("<td>{}</td>".format(s))
  print("</tr>\n")
if times == []:
  print("<!-- didn't find any at {} -->".format(datetime.datetime.now()))
print("</table>\n</BODY></HTML>\n")
```

3. This code maintains a list of user names and IDs and has insertion, search, and delete functions.

```
class ListItem {
  public:
    ListItem *next;
    ListItem *prev;
    int id;
    char[9] name;
};
class UserList {
  public:
    ListItem *head;
    void insert(char *, int);
    void deleteByName(char *);
    int idForName(char *);
};
void UserList::insert(char *name, int id) {
  ListItem *user = new ListItem();
  user->id = id;
  strcpy(user->name, name);
  user->next = this->head;
  this->head->prev = user;
  this->head = user;
}
void UserList::deleteByName(char *name) {
  ListItem *curr = this->head;
  ListItem *next = curr->next;
  while (curr != NULL) {
    if (strcmp(curr->name,name)==0) {
      curr->prev->next = next;
      next->prev = curr->prev;
      delete curr;
      return;
    }
    curr = next;
    next = next->next;
  }
}
int UserList::idForName(char *name) {
  ListItem *curr = this->head;
  while (curr != NULL) {
    if(strcmp(curr->name,name) == 0) {
      return curr->id;
    }
  }
  return -1;
}
```

4. This program takes a `NULL`-pointer terminated array of strings (`char *`) and returns a pointer to a copy of the most commonly occurring string in the array.

```
char *most_common(char **words) {
  char *result;
  int rlen, maxcount, ccount;
  char **cword = words;
  while (*cword != NULL) {
    rlen += strlen(*cword);
    cword++;
  }
  result = malloc(rlen); /* That'll hold the output for sure! */
  char **aword;
  for(cword = words; *cword != NULL; cword++) {
    for (aword = words; *aword != NULL; aword++) {
      if (strcmp(*aword,*cword) == 0) ccount++;
    }
    if (ccount > maxcount) {
      maxcount = ccount;
      strcpy(result,*cword);
    }
    ccount = 0;
  }
  return result;
}
```

This assignment is based in large part on assignments originally by Prof. Nick Hopper, and is licensed under Creative Commons Attribution-ShareAlike 4.0.