**CYB 220 Secure Coding and Analysis**

**Spring 2024 Midterm exam**

Due: Thursday Mar 7, 11:59 pm on CANVAS.

Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| **Question #** | **Max Points** | **Your score** |
| 1 | 30 |  |
| 2 | 50 |  |
| 3 | 4 |  |
| 4 | 4 |  |
| 5 | 4 |  |
| 6 | 4 |  |
| 7 | 4 |  |
| **Total** | 100 |  |

Note:

* This is an open book, open note exam, however you need to answer in your own words! Do not copy from the text or lectures or use online definitions.
* For each question, please explain in detail or use examples. Don’t just give short 2-3 sentence answers.
* Edit this document and submit the response as a PDF or Word document to CANVAS. Please don’t remove the questions to make grading easier.

HINT: This exam may take 3-5 hours

1. (30 pts) Please explain in detail: **Define** the following terms, discuss the **common vulnerabilities/problems** associated with each of the following and discuss **how to avoid** them when writing code.
   1. C-style string
   2. Metacharacter
   3. Type conversion in C Language
   4. MACRO
   5. Buffer overflow
   6. Off-by-one error

* 1. Format string attack
  2. Integer overflow
  3. Operator precedence
  4. SQL injection

1. (50 pts) Each of the following pieces of C code has an error in it. Please identify and explain the vulnerability and discuss **how to fix it**. NOTE: Don’t consider type mismatch/input data type as an error. eg, the code reads in a number, just assume user gives a valid number, no error.

(a)

int \*table = NULL;  
int insert\_in\_table(int pos, int value) {  
 if (!table) {  
      table = (int \*)malloc(sizeof(int) \* 100);  
    }  
    if (pos >= sizeof(table) || pos < 0) {  
      return -1;  
    }  
    table[pos] = value;  
    return 0;  
}

(b)

u\_char \*build\_table(unsigned int width, unsigned int height, u\_char \*init\_row)  
{  
    int k;

unsigned int n;  
    u\_char \*buf;  
    n = width \* height;  
    buf = (char \*)malloc(n);  
    if (!buf)  
        return (NULL);  
    for (int k=0; k< height; k++)  
        memcpy(&buf[k\*width], init\_row, width);  
    return buf;  
}

(c)

int update\_profile(char \*username, char \*data)  
{  
    char buf[64];  
    int fd;  
    snprintf(buf, sizeof(buf), "/data/profiles/%s.txt",  
              username);  
    fd = open(buf, O\_WRONLY);  
    ...  
}

(d)

   char \*read\_nums()

{

int user\_nums[50];

char file\_ptr, file\_names[200];

int i;

file\_ptr = (char \*)malloc(sizeof(file\_name));

if(!file\_ptr){

error("file\_name allocation failed: %m");

return NULL;

}

for (i=0; i<sizeof(user\_nums); i++){

scanf("%d", &user\_nums[i]);

}

}

(e)

int read\_data(int sockfd)  
{  
    char buffer[200];  
    int length = get\_user\_length(sockfd); //reads an integer from the network  
    if(length > 200){  
        error("illegal input, not enough room in buffer\n");  
        return –1;  
    }  
    if(read(sockfd, buffer, length) < 0){  
        error("read failed!");  
        return –1;  
    }  
    return 0;  
}

(f)

void usage(char \*process\_name) {  
    char usage\_line[1024];  
    snprintf(usage\_line, 1024,  
      "Usage: %s <target>\n", process\_name);  
    printf(usage\_line);  
  }  
  
  int main(int argc, char \* argv[]) {  
    if (argc > 0 && argc < 2) {  
      usage(argv[0]);  
      exit(-1);  
    }  
  }

(g)

int arith\_op(){

int num1, num2, sum, results;

if ((num1 > 0) && (num2 >0) && (num1 > (INT\_MAX – num2))) ||

((num1 < 0) && (num2 <0) && (num1 < (INT\_MIN – num2))) {

printf("Error: addition overflow!\n");

}

else {

sum = num1 + num2;

}

if ((num1 < 0) && (num2 >0) && (num1-num2 < INT\_MIN)) ||

((num2 < 0) && (num1 >0) && (num2-num1 < INT\_MIN)) {

printf("Error: subtraction overflow!\n");

}

else {

results = num1 - num2;

}

}

(h)

 int main(int argc, char \*argv[]) {  
    unsigned short int total;  
    total = strlen(argv[1]) + strlen(argv[2]) + 1;  
    char \*buff = (char \*)malloc(total);  
    strcpy(buff, argv[1]);  
    strcat(buff, argv[2]);  
    /\* ... \*/  
  }

(i)

int copy\_name(char \*fileName)  
{  
    char buffer[1024];

if (strlen(fileName) < 1000){  
     strcpy(buffer, "The file name is: ");  
     strncat(buffer, fileName, sizeof(buffer));  
  
     log("%s\n", buffer);  
     return 0;

}

else {

return -1;

}  
}

(j)

#include <stdio.h>  
  int main(void) {  
  
    char file\_name[256];  
    FILE \*f;  
    char file\_format[256];  
  
    fgets(file\_name, 256, stdin);  
    f = fopen(file\_name, "r"); /\* read only \*/  
  
    if (f == NULL) {  
      sprintf(file\_format, "Error opening file %s\n", file\_name);  
      fprintf(stderr, file\_format);  
      exit(-1);  
    }  
    fclose(f);  
  }

3) (4 pts) What is Defensive programming? Why do we need it? List and explain 3 strategies of defensive programming.

4) (4 pts) Distinguish Black-list vs. white-list in metacharacter filtering. Explain pros and cons of each method.

5) (4 pts) Why do we need to do Input validation? Explain in detail what we should check in input validation.

6) (4 pts) Even if we use the safer version of C library functions that have a parameter for length of the string, things may still go wrong. Why? Please use an example code to explain.

7) (4 pts) In terms of metacharacter filtering, we talked about 3 options (reject, strip, and encode). Explain each of these 3 options. What are the pros and cons of each of them? Which one do you prefer and why?