Homework 2

CIS 4930 / CIS 5930
Offensive Computer Security (OCS2014)
Spring 2014

Due January 27th, 2014, by *MIDNIGHT*
Worth: 100 points

Electronic turn in (Turn in via email to TA. Email address is raiaan@cs.fsu.edu)
The email must be titled in the following format:
"[OCS2014] hw2 <your last name>". (where <your last name> is your last name)
Example: [OCS2014] hw2 redwood

Tips:
When you copy paste code from this document to an editor, certain characters may cause compiler errors. Most notably " and ' marks sometimes get replaced by very similar but incompatible corresponding quotation markers. You may see compiler errors like:

error: stray '\342' in program
error: stray '\200' in program
…

Such errors indicate that characters are present from an unsupported character set.

#1) [30 points (5 each)]For the following answer with a value or undefined

A. UINT_MAX + 1 = ?
B. INT_MAX + 1 = ?
C. - INT_MIN = ?
D. Does x - 1 + 1 evaluate for all values of x (int)? If not, explain
E. for what values x and y, does (x % y) invoke undefined behavior?
F. -INT_MAX = ?

#2) [10 points] Is this function safe? If not explain why.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

float safe_sqrt(float value)
{
  float x;
  x = sqrt(value);

  if(x == NAN)
  {
    perror("Error! Sqrt produced NAN!\n");
    exit(1);
  }
  return x;
}
```

#3) [10 points] A fresh-out-of-school and lazy developer produced the following code. Many parts are commented out for brevity. The code takes

in a user string which is a CD-KEY to activate a software product. It has been designed for international markets, so the developers thought they should use wchar_t based strings everywhere. Can you find the bug?

A. Explain exactly what the code does wrong, and provide pseudo code to correct it.

```
#include <wchar.h>
#include <stdlib.h>
#include <string.h>

extern wchar_t master_key[] = L"0123456789";

int CD_KEY_CHECKER (wchar_t *cd_key) {
      int valid = 0;
      // the following is one line of code, but has wrapped around:
      wchar_t tmp_key[12];
      snprintf(tmp_key, sizeof(tmp_key), “%s\0”, cd_key);

      if (tmp_key != NULL)
      {
            if (wcsncmp(tmp_key, master_key, sizeof(master_key)) == 0)
                  return 1;
      }
      free(cd_key);  // typo fixed (Jan 22 @3:44PM)
      cd_key = NULL; // typo fixed (Jan 22 @3:44PM)
      return 0;
}
```

#4) [10 points] This program checks if two strings are the same as each other forwards and backwards. It detects palindromes. The developer used safe functions like strncpy.

A. Explain the bug
B. Is this vulnerable? Explain how you would land the vulnerability

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char *strrev(char *str)
{
      char *p1, *p2;

      if (! str || ! *str)
            return str;
      for (p1 = str, p2 = str + strlen(str) - 1; p2 > p1; ++p1, --p2)
      {
            *p1 ^= *p2;
            *p2 ^= *p1;
            *p1 ^= *p2;
      }
      return str;
}

int main(int argc, char *argv[]){
  char s1[100];
  char s2[100];
  size_t len;
  if (argc != 3){
    printf("This is the palindrome program \n");
    printf("Usage: ./palindrome string1 string2 \n");
    exit(0);
  }
  strncpy(s1, argv[1], sizeof(s1)); // Changed (Jan 22 @4:39PM)
  len = strlen(s1); // since they should be the same size
  strncpy(s2, argv[2], len);

  // do palindrome check
  if (strcmp (s1,s2) == 0) // they should be the same right?
```

```
  {
       strrev(s2);
       if (strcmp (s1,s2) == 0)  // make sure same when reversed
         printf("This is a palindrome.\n");
       else
         printf("Not a palindrome.\n");
  }
}
```

#5) [20 points] The following is from a 2000 CVE. Find the vulnerability(s) in the following code. For each explain a) How it is used incorrectly and b) How to use the function correctly

```
#include <stdio.h>

#if defined ( HAVE_SYSLOG_H ) && defined ( USE_SYSLOG )
     extern Display *dsp;

     syslog(SYSLOG_WARNING, buf);
     if (!nolock) {
          if (strstr(buf, "unable to open display") == NULL )
               syslogStop(XDisplayString(dsp));
          closelog();
     }
#else
     (void) fprintf(stderr,buf);
#endif
     exit(1);
}
```

#6) [20 points] The following code has two vulnerabilities. Spot & explain each. Then explain the impact of each and how it might be exploited. The code is part of a multi-threaded server, which accepts connections from clients and performs corrections/analysis/calculations on spectrum data the client sends. For each connection the server spawns a thread to handle it and runs the following analyze_spectrum function(). Thus it is important to consider that multiple clients can be connected at the same time having their spectrums examined. (Hint: threads all share the same heap). The following function is also used in a part of a series of checks also to determine if a chemical sample is poisonous or not (Exploitation thus need not

involve launching a shell or gaining access). **(Problem changed Jan 22 @4:21PM)**

```
#include <stdio.h>
#include <syslog.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <math.h>

#define SIZE 512

float* correct_down(float *spec, float b);
float* correct_up(float *spec, float b);
float* smooth_left(float *spec, float b);
float* smooth_right(float *spec, float b);

float* (*p[4])(float *spec, float b);

float max_log_norm = 0.0;

float * spectrum_analyze(float *spec, int len, int type, float baseline, int sockfd)
{
  int i, n;
  float log_normalized;
  float sqrt_normalized;
  float min, max;
  float* (*c)(float *spec, float b);
  float *tmp_spec = (float *) malloc (len * sizeof(float));
  char results[SIZE];

  if (type < 4)
        c = (*p[type]);

  if (len > 0) {
        min = spec[0];
        max = spec[0];
  } else {
        return NULL;
  }

  for(i = 0; i < len; i++){
        if (spec[i] > max) {
              max = spec[i];
        } else if (spec[i] < min) {
```

```
                min = spec[i];
        }
}

//normalize loop
for (i = 0; i < len; i++)
{
        sqrt_normalized = sqrt(spec[i] - baseline);
        log_normalized = log(spec[i] - baseline);

        if (sqrt_normalized != NAN && log_normalized != NAN)
        {
                if(sqrt_normalized > max_log_norm){
                        n = write(sockfd, "Unique specimen!\n", n);
                        if (n <= 0){
                                free(spec);
                                free(tmp_spec);
                                syslog(LOG_ERR,"Client Connection Dropped");
                                sleep(5);
                                break;
                        }
                }
                if (log_normalized > max_log_norm)
                         max_log_norm = log_normalized;
                if (log_normalized < sqrt_normalized)
                {
                        tmp_spec = c(spec, log_normalized);

                } else if (log_normalized < sqrt_normalized){
                        tmp_spec = c(spec, log_normalized);
                }

        } else { // They must be NAN!
                return NULL;
        }



        //write back to client
        sprintf(results,
        "spec[%d]: log_norm=%08.4f; sqrt_norm=%08.4f, max=%08.6f, min=%08.6f\n" ,
         i, sqrt_normalized, log_normalized,max,min);  // end of sprintf() line.
        n = write(sockfd, results, strlen(results));
        if (n <= 0)
```

```
        {
                free(spec);
                free(tmp_spec);
                syslog(LOG_ERR,"Client Connection Dropped");
                sleep(5);
                break;
        }

  } // end of for loop
  return tmp_spec;
}
```