Assignment 4

Michael Peterson

---------- Problem A ----------

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

#define BUFFER_SIZE 64

// This function was blindly copying bytes from arg into out without verifying the size of the output array
//  foo(char *arg, char *out          )
int foo(char *arg, char *out, size_t n)
{
// Use strncpy instead of strcpy so that the number of bytes can be specified
//  strcpy (out, arg   );
    strncpy(out, arg, n);
    return 0;
}

int main(int argc, char *argv[])
{
    char buf[BUFFER_SIZE];

    // It's good practice to initialize our buffer
    int i = 0;
    for(; i < BUFFER_SIZE; ++i){
        buf[i] = '\0';
    }


    if (argc != 2)
    {
        fprintf(stderr, "a: argc != 2\n");
        exit(EXIT_FAILURE);
    }

//  foo(argv[1], buf             );
    foo(argv[1], buf, BUFFER_SIZE);
    return 0;
}

/***************************

 Problems Found:
 in foo():
    strcpy(dst, src) will blindly copy bytes from source to destination until it runs into a '\0' character in the source,
    this is a vulnerability becuase the source string can be longer the destination buffer.

 Fixes:
    Add a length parameter to foo() so that the destinatio buffer size can be specified,
    then use strncpy(dst, src, n) to perform the copying.


 Original Implementation


int foo(char *arg, char *out)
{
    strcpy(out, arg);
    return 0;
}

int main(int argc, char *argv[])
{
    char buf[64];
    if (argc != 2)
    {
        fprintf(stderr, "a: argc != 2\n");
        exit(EXIT_FAILURE);
    }

    foo(argv[1], buf);
    return 0;
}

 **/
```

---------- Problem B ----------

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

#define BUFFER_SIZE  128

int foo(char *arg)
{
//  Initialize Variables
//  int len, i;
    int copyCharCount = 0, i  = 0;


//  It's more convenient to use the pre processor to manage magic numbers
//  char buf[128];
    char buf[BUFFER_SIZE];

    // It's good practice to initialize our buffer
    for(i = 0; i < BUFFER_SIZE; ++i)
        buf[i] = '\0';
    i = 0;

//  The previous implementation used incorrect magic numbers to limit the number of characters that are copied into the buffer
//  len = strlen(arg);
//  if (len > 136)
//      len = 136;
```

```
//  These two lines are the equivalent to min(BUFFER_SIZE, strlen(argv[1])), but don't necessarily iterate through the
//  entire array as strlen() would
    while(copyCharCount < BUFFER_SIZE && arg[copyCharCount] != '\0')
        ++copyCharCount;

//  Since our buffer has been initialized with null characters, we don't have to copy over the '\0' at the end of
//  arg (using < instead of <=))
//  for (i = 0; i <= len; i++)
//      buf[i] = arg[i];
    for(i = 0; i < copyCharCount; ++i)
        buf[i] = arg[i];

    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "b: argc != 2\n");
        exit(EXIT_FAILURE);
    }

    foo(argv[1]);

    return 0;
}


/*****************************

 Problems Found:
 in foo():
    1) The magic number used to determine if the input string are incorrect (it is different then the one used to declare the buffer),
       this could cause too many bytes to be copied over to the output array.
    2) The loop copying the bytes into the buffer is going one char to far becuase of the condition being used is inclusive, this could

       what should be a null char in the buff with one to many chars from the input array.
    3) The buffer is not initilaized with null characters.


 Fixes:
    1) Fix the magic numbers (I did this by declaring a symbol BUFFER_SIZE with a #define directive and using it instead of int literals)
    2) Change the condition to < instead of <=
    3) Initialize the buffer right after it is declared.



Original Implementation


int foo(char *arg)
{
    char buf[128];
    int len, i;

    len = strlen(arg);
    if (len > 136)
        len = 136;

    for (i = 0; i <= len; i++)
        buf[i] = arg[i];

    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "b: argc != 2\n");
        exit(EXIT_FAILURE);
    }

    foo(argv[1]);
    return 0;
}
 **/
```

---------- Problem C ----------

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

#define BUFFER_SIZE  128

int bar(char *arg, char *targ, int ltarg)
{
//  int len, i;
    int copyCharCount = 0, i = 0;

//  The previous implementation used incorrect magic numbers to limit the number of characters that are copied into the buffer
//  len = strlen(arg);
//  if (len > ltarg)
//      len = ltarg;

//  These two lines are the equivalent to max(BUFFER_SIZE, strlen(argv[1])), but don't necessarily iterate through the
//  entire array as strlen() would
    while(copyCharCount < ltarg && arg[copyCharCount] != '\0')
        ++copyCharCount;

//  Since our buffer has been initialized with null characters, we don't have to copy over the '\0' at the end of
//  targ (using < instead of <=))
//    for (i = 0; i <= len; i++)
//        targ[i] = arg[i];
```

```
    for (i = 0; i < copyCharCount; i++)
        targ[i] = arg[i];

    return 0;
}

int foo(char *arg)
{
    int i = 0;

//  char buf[128];
    char buf[BUFFER_SIZE];

    // It's good practice to initialize our buffer
    for(i = 0; i < BUFFER_SIZE; ++i)
        buf[i] = '\0';
    i = 0;



//  bar(arg, buf, 140);
    bar(arg, buf, BUFFER_SIZE);
    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "c: argc != 2\n");
        exit(EXIT_FAILURE);
    }

    foo(argv[1]);
    return 0;
}

/*****************************

 Problems Found:
 in foo():
    1) The magic number used to determine if the input string are incorrect (it is different then the one used to declare the buffer),
       this could cause too many bytes to be copied over to the output array.

 in bar():
    2) Using strlen() to determine a user provided strings length can be a vulnerability becuase the string length could exceed the

       for the type returned by strlen() and cuase a rollover resulting in a negative value being returned.

 in main():
    3) The buffer should be initialized with null characters

 Fixes:
    1) Fix the magic numbers (I did this by declaring a symbol BUFFER_SIZE with a #define directive and using it instead of int literals)
    2) I wrote a loop that will determine the length of the input string if it's shorter then the buffer size, the loop is not vulnerable
       to the overflow problem of strlen becuase it will only iterate as many times as there are elemnts in the buffer.
    3) Initialize the buffer using a loop.




Original Implementation

int bar(char *arg, char *targ, int ltarg)
{
    int len, i;
    len = strlen(arg);
    if (len > ltarg)
        len = ltarg;

    for (i = 0; i <= len; i++)
        targ[i] = arg[i];

    return 0;
}

int foo(char *arg)
{
    char buf[128];
    bar(arg, buf, 140);
    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "c: argc != 2\n");
        exit(EXIT_FAILURE);
    }

    foo(argv[1]);
    return 0;
}
 **/
```

---------- Problem D ----------

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

#define BUFFER_SIZE 1024

//int foo(char *arg, short arglen)
```

```
int foo(char *arg                )
{
//  char buf[1024];
    char buf[BUFFER_SIZE];
//  int i,     maxlen = 1024;
    int i = 0, maxlen = BUFFER_SIZE, charCopyCount = 0;

    for(i = 0; i < BUFFER_SIZE; ++i)
        buf[i] = '\0';


//  if (arglen < maxlen)
//  {
//      for (i = 0; i < strlen(arg); i++)
//          buf[i] = arg[i];
//  }

    while(charCopyCount < maxlen && arg[charCopyCount] != '\0')
        ++charCopyCount;

    for(i = 0; i < charCopyCount; ++i)
        buf[i] = arg[i];

    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "d: argc != 2\n");
        exit(EXIT_FAILURE);
    }

//  foo(argv[1], strlen(argv[1]));
    foo(argv[1]                 );
    return 0;
}



/*****************************

 Problems Found:
 in foo():
    1) Using strlen in the for loop is not only very inefficient but insecure becuse of the return value can end up being
       negative if the number of bytes in arg is greater then the maximum value of strlen()'s return type.
    2) The buffer should be initialized;
 in main():
    3) Again, strlen() shouldn't be used get the length of arg[1]



 Fixes:
    1) find min( length(buf), length(arg)) using a while loop and copy that many characters into the buffer
    2) Initialize the buffer
    3) remove the length parameter from foo()


int foo(char *arg, short arglen)
{
    char buf[1024];
    int i, maxlen = 1024;

    if (arglen < maxlen)
    {
        for (i = 0; i < strlen(arg); i++)
            buf[i] = arg[i];
    }

    return 0;
}

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        fprintf(stderr, "d: argc != 2\n");
        exit(EXIT_FAILURE);
    }

    foo(argv[1], strlen(argv[1]));
    return 0;
}
 **/
```