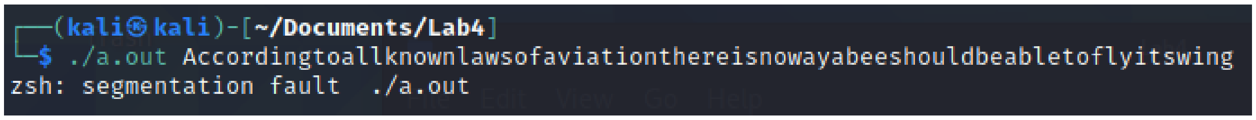
Names: Ryan Cranston, Petal Michaud, Julian Welge

Code 1: “a.c”

```
int foo(char *arg, char *out)
{
	strcpy(out, arg);  <- strcpy will fail if out is smaller than arg causing just a
	                      null terminating char
	return 0;
}
int main(int argc, char *argv[])
{
	char buf[64];
	if (argc != 2)
	{
		fprintf(stderr, "a: argc!= 2\n");
		exit(EXIT_FAILURE);
	}// end if
	foo(argv[1], buf);
	return 0;
}// end main
```



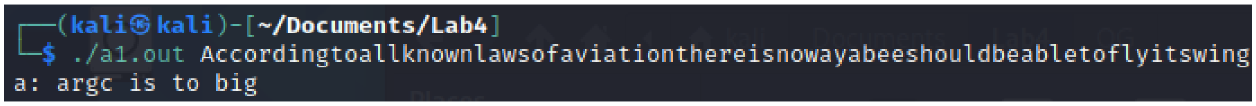
```
┌──(kali㉿kali)-[~/Documents/Lab4]
└─$ ./a.out Accordingtoallknownlawsofaviationthereisnowayabeeshouldbeabletoflyitswing
zsh: segmentation fault  ./a.out
```

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

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);
	}// end if
	if (strlen(argv[1]) > 64)
	{
		fprintf(stderr, "a: arg > 64\n");
```

```
                exit(EXIT_FAILURE);
        }// end if
        foo(argv[1], buf);
        return 0;
}// end main
```

To address the vulnerabilities I added a termination size check so if the input string is larger than the given buffer it will terminate the program and alert the user. The added vulnerability is it does not check if what is being entered is valid characters and run into similar issues as strncpy

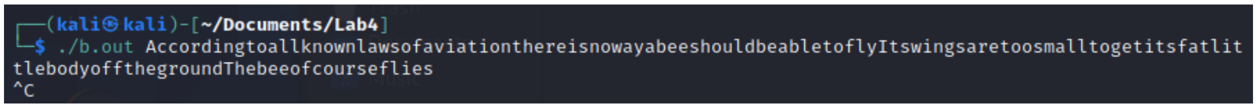


```
┌──(kali㉿kali)-[~/Documents/Lab4]
└─$ ./a1.out Accordingtoallknownlawsofaviationthereisnowayabeeshouldbeabletoflyitswing
a: argc is to big
```

Code 2: “b.c”

```
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;
}// end main
```

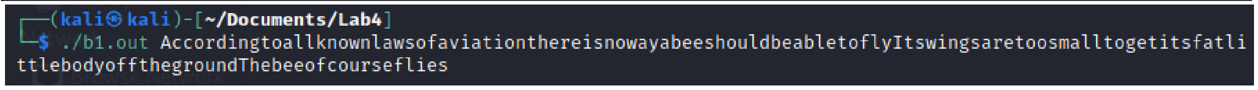
The Issue with the given code is it is letting buf get filled without a null terminating char at the end and is overrunning the space allocated for buf, causing the program to go searching for a null char in memory.



```
┌──(kali㉿kali)-[~/Documents/Lab4]
└─$ ./b.out AccordingtoallknownlawsofaviationthereisnowayabeeshouldbeabletoflyItswingsaretoosmalltogetitsfatlit
tlebodyoffthegroundThebeeofcourseflies
^C
```

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

int foo(char *arg)
{
      char buf[128];
      strncpy(buf, arg, sizeof(buf));
      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;
}// end main
```



```
┌──(kali㉿kali)-[~/Documents/Lab4]
└─$ ./b1.out AccordingtoallknownlawsofaviationthereisnowayabeeshouldbeabletoflyItswingsaretoosmalltogetitsfatli
ttlebodyofftheground Thebeeofcourseflies
```

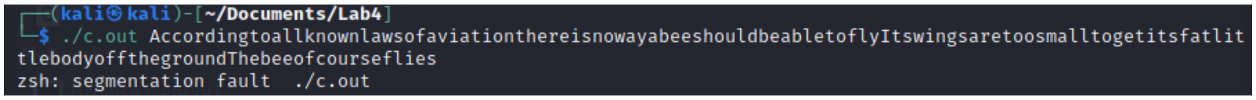
A solution to this problem would be to cut out the problem code and use strncpy() and limit the copying size to buf and including a \0, a problem caused by this solution is the input string gets cut off once it runs out of memory.

Code 3: “c.c”

```c
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;
}// end main
```

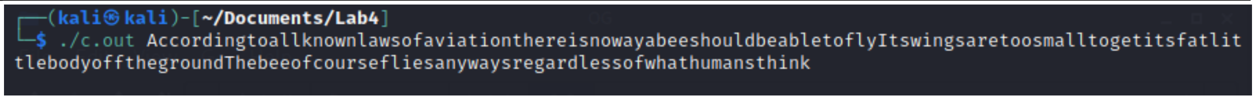
The issue with the code above the receiving string of buf is not big enough to hold what is being put into it by the bar function so we end up with a segmentation fault.



```
┌──(kali㉿kali)-[~/Documents/Lab4]
└─$ ./c.out AccordingtoallknownlawsofaviationthereisnowayabeeshouldbeabletoflyItswingsaretoosmalltogetitsfatlit
tlebodyofftheground Thebeeofcourseflies
zsh: segmentation fault  ./c.out
```

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

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, 127);
	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;
}// end main
```



```
┌──(kali㉿kali)-[~/Documents/Lab4]
└─$ ./c.out AccordingtoallknownlawsofaviationthereisnowayabeeshouldbeabletoflyItswingsaretoosmalltogetitsfatlit
tlebodyofftheground Thebeeofcourseflies anywaysregardlessofwhathumansthink
```

My solution is to change the hard set number in the bar function call to one below the size of buf so it doesn’t overrun buf, an issue with this solution is the argument being fed in is cut off and buf won’t contain a terminating char at the end of the array.

Code 4: “d.c”

```
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;
}
```

This code can’t seem to be exploted.