AppSec Assignment 1.1: Beware of Geeks Bearing Gift Cards

John McIntyre

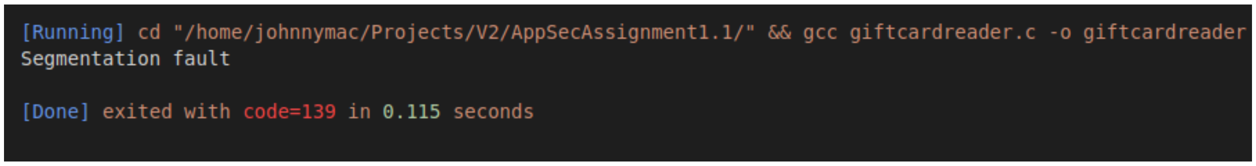
Part 1: Setting up Git and Signing Commits

To start off the project, I set up a private GitHub repository and titled it AppSecAssignment1.1. Next, I cloned a copy of the AppSecAssignment1.1 assignment to my virtual machine using Git, linked Travis to my GitHub repository and added a travis.yml file. Then, I set up both an SSH and GPG key for my virtual machine, so I could sign my commits. Finally, I tried pushing a commit to make sure Travis was working and my commits were signed.

Part 2: Bug Hunting, Testing and Fixing

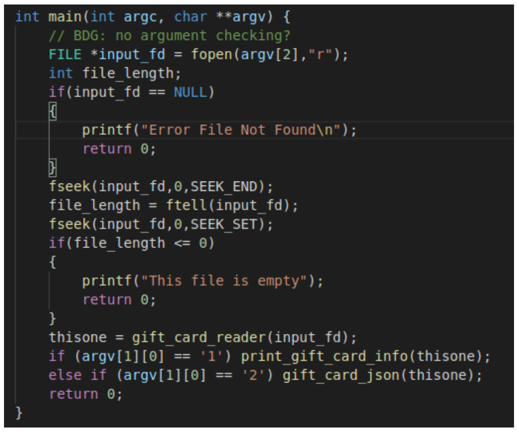
After pulling the files from GitHub, I first used the build command to build programs for both giftcardreader.c and giftcardexamplewriter.c. However, the initial build failed for me. To fix this, I had to add the <string.h> header to the giftcardreader.c file, and I had to move the get_gift_card_value structure to earlier in the code, as it was being called before being declared. After fixing these issues, I re-ran the build command and was able to get the gift card reader file to run properly.

The first crash I analyzed was essentially a two-part bug around the types of files being passed into the giftcardreader.c file. If I attempted to run the giftcardreader.c file without an accompanying file or if I passed an empty file, I was able to cause a segmentation fault when compiling the gift card reader program.



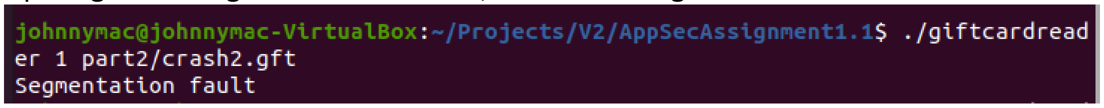
```
[Running] cd "/home/johnnymac/Projects/V2/AppSecAssignment1.1/" && gcc giftcardreader.c -o giftcardreader
Segmentation fault

[Done] exited with code=139 in 0.115 seconds
```

Looking into the code, it seems that both of the segmentation fault errors came from the main structure of the giftcardreader.c file. The main structure opens the file but does not include a check to verify that there is an accompanying file or if the file is empty or not. To fix this issue, I included a segment of code in the main function that checks to see if a file was run along with the executable. If no file is found, it returns an error message and then ends the program using return 0. If the main function does find a file, the main function gets the length of the file. Then, if the length of the file is retuned as zero, the program returns a different error message that states that the file is empty and ends the program, again with return 0. Finally, if the length of the file is greater than zero, the program runs normally.



```c
int main(int argc, char **argv) {
    // BDG: no argument checking?
    FILE *input_fd = fopen(argv[2],"r");
    int file_length;
    if(input_fd == NULL)
    {
        printf("Error File Not Found\n");
        return 0;
    }
    fseek(input_fd,0,SEEK_END);
    file_length = ftell(input_fd);
    fseek(input_fd,0,SEEK_SET);
    if(file_length <= 0)
    {
        printf("This file is empty");
        return 0;
    }
    thisone = gift_card_reader(input_fd);
    if (argv[1][0] == '1') print_gift_card_info(thisone);
    else if (argv[1][0] == '2') gift_card_json(thisone);
    return 0;
}
```

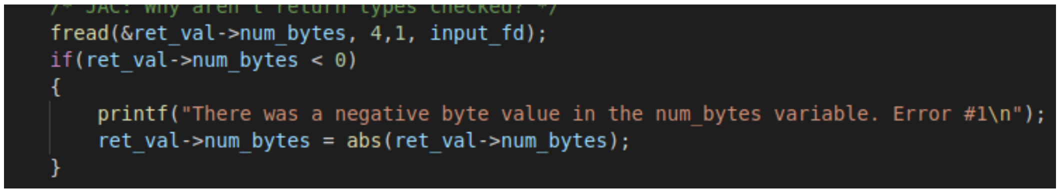
The second crash I analyzed had to do with the num_bytes variable in the giftcardexamplewriter.c file. In this crash, I set the num_bytes variable to -1, and when I ran the output against the giftcardreader.c file, it caused a segmentation fault.



```
johnnymac@johnnymac-VirtualBox:~/Projects/V2/AppSecAssignment1.1$ ./giftcardread
er 1 part2/crash2.gft
Segmentation fault
```

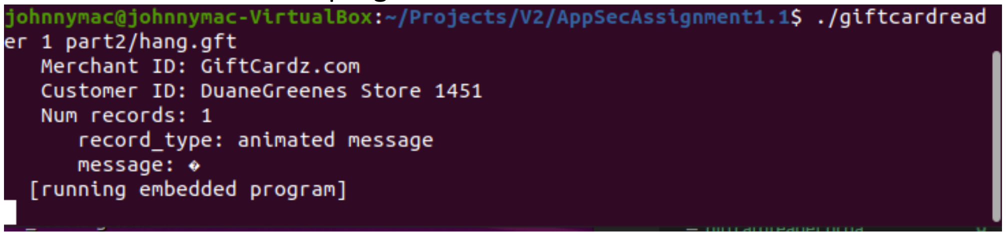
In order to determine the source of the crash, I looked through the giftcardreader.c file and running the code through GDB, I found the issue lied within the gift_card_data structure. In the structure, there is an fread statement that takes the num_bytes variable and assigns it to the ptr variable. This causes a segmentation fault, because the num_bytes value is being passed into an unsigned character, which does not support negative numbers. In the case that a negative number is passed into an unsigned character, the negative number would wrap around and become a much larger number, causing the fread to read past the end of the .gft file, and in turn causes a segmentation fault, due to an access violation.

To fix this crash, I added an if statement in the gift_card_data structure that checks to see if the num_bytes variable is less than zero. If the variable is less than zero, it returns an error statement and sets the num_bytes value to the absolute value of the num_bytes. This prevents the negative number from wrapping around and causing a segmentation fault.



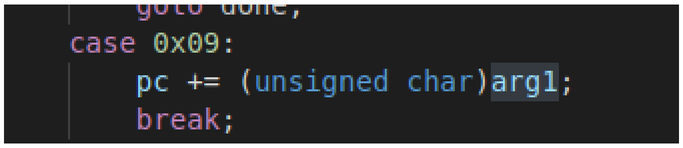
```c
    /* JAC: why aren't return types checked? */
    fread(&ret_val->num_bytes, 4,1, input_fd);
    if(ret_val->num_bytes < 0)
    {
        printf("There was a negative byte value in the num_bytes variable. Error #1\n");
        ret_val->num_bytes = abs(ret_val->num_bytes);
    }
```

The final issue I examined was not a crash, but instead an issue that caused the program to hang. Upon examining the giftcardreader.c file, I found a while loop in the animate function that I thought I could exploit. Within the animate function, a variable called pc increments until it reaches a value of 256, at which point it will break and end the loop. Therefore, if I wanted the program to loop, I needed to prevent the pc variable from reaching 256. To do this, I focused on case 0x09, where the pc variable is set to the signed character arg1. To access the animate function, I had to set the number_of_gift_card_records variable in the giftcardexamplewriter.c file to three. Next, I passed a value of 253 into case 0x09. I picked 253, because as arg1 is a signed character, and signed characters can only support values from -128 to 127. When a signed character is set to 253, it will wrap around to a value of -3, which works in this case, because it counteracts the pc+3 line towards the end of the animate function. Now, when the program runs against the hang.gft file, the program will hang indefinitely, since the pc value will never exceed the program+256 value.



```
johnnymac@johnnymac-VirtualBox:~/Projects/V2/AppSecAssignment1.1$ ./giftcardread
er 1 part2/hang.gft
   Merchant ID: GiftCardz.com
   Customer ID: DuaneGreenes Store 1451
   Num records: 1
      record_type: animated message
      message: �
  [running embedded program]
```

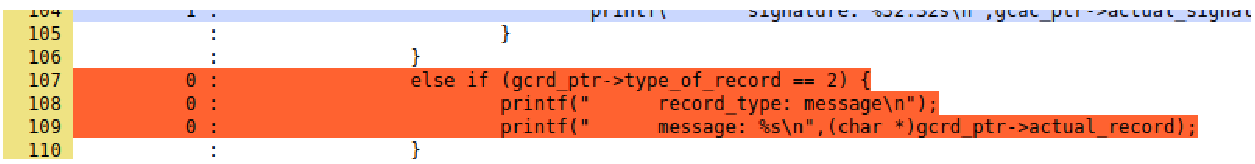
To fix this hang, we simply need to change the signed character arg1 variable to an unsigned character. This would prevent the 253 value to wrap around, as unsigned characters support up to 255, and also does not support negative values. Now, our hang.gft file will not cause the program to hang and instead will run normally.



```
        goto done;
    case 0x09:
        pc += (unsigned char)arg1;
        break;
```

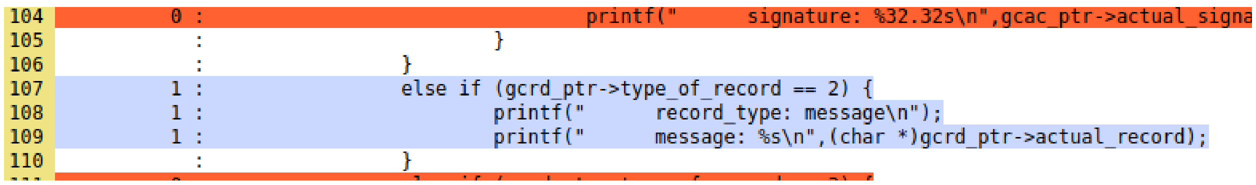
Part 3: Fuzzing and Coverage

After installing gcov, I ran the base examplefile and my crashes against the giftcardreader.c to get percentages of how much of my code is executed with each test case. I next picked out two lines of code that were not run during any of my test cases. The first line is an if statement in the print_gift_card_info, where the type of record is equal to two. I chose this, because none of my test cases touched this line of code, due to none of the testcases using a value of two for the type_of_record variable.



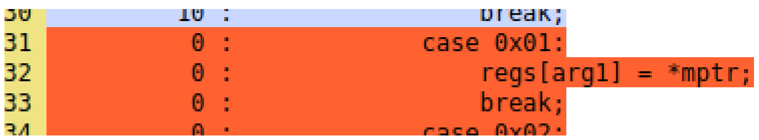
```
 104          1 :                     printf("      signature: %32.32s\n",gcac_ptr->actual_signat
 105            :                 }
 106            :             }
 107          0 :             else if (gcrd_ptr->type_of_record == 2) {
 108          0 :                 printf("      record_type: message\n");
 109          0 :                 printf("      message: %s\n",(char *)gcrd_ptr->actual_record);
 110            :             }
```

To access this line of code, I set the type_of_record variable in the giftcardexamplewriter.c file to two. I then updated the examplefile.gft to include this change and reran the code. After producing a new lcov report, I saw that the selected code section was successfully run.



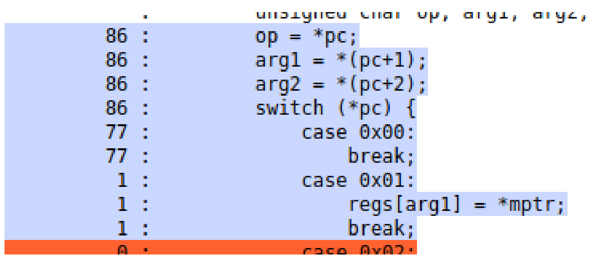
```
        0 :                         printf("      signature: %32.32s\n",gcac_ptr->actual_signa
          :                 }
          :         }
        1 :         else if (gcrd_ptr->type_of_record == 2) {
        1 :                 printf("      record_type: message\n");
        1 :                 printf("      message: %s\n",(char *)gcrd_ptr->actual_record);
          :         }
```

The second line of code I picked was a line was the ‘regs[arg1] = *mptr’ inside case 0x01 within the animate function. I chose this line, because I thought that if I could access different parts of the animate function, I could create more ways to hang the program, as they all exist inside the same while loop.



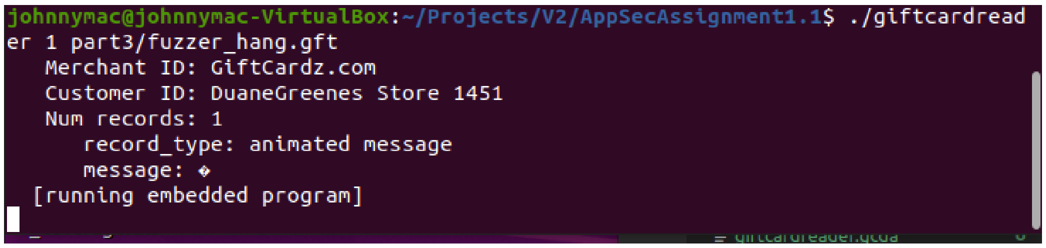
```
       10 :                         break;
        0 :                 case 0x01:
        0 :                         regs[arg1] = *mptr;
        0 :                         break;
        0 :                 case 0x02:
```

To access this line of code, I simply modified my hang function slightly to instead focus on case 0x01. I also tried passing different values to create another hang, but was unsuccessful.



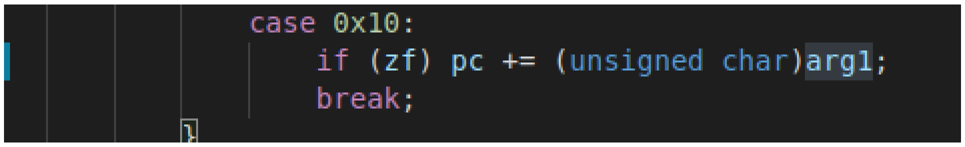
```
          :         unsigned char op, arg1, arg2;
       86 :         op = *pc;
       86 :         arg1 = *(pc+1);
       86 :         arg2 = *(pc+2);
       86 :         switch (*pc) {
       77 :             case 0x00:
       77 :                 break;
        1 :             case 0x01:
        1 :                 regs[arg1] = *mptr;
        1 :                 break;
        0 :             case 0x02:
```

I ran into no issues when installing the AFL fuzzer. After running the fuzzer for a little over 2 hours, I got around 34 unique crashes and 9 hangs. After going through the output, I managed to find a unique crash and hang to analyze. The first one I looked at was the hang (fuzzer1.gft). Upon analyzing the output from the hang file, I found that it used the animate function to hang the program, just like how I had done in my hang file. However, this file instead used case 0x10 to hang the program instead of 0x09. I found this by inserting print statements into the different test cases to determine which case was running.



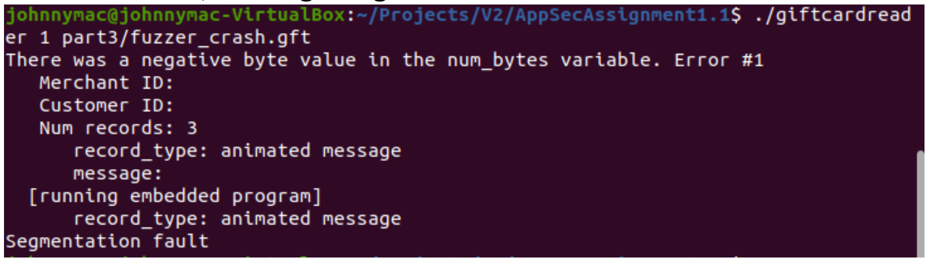
```
johnnymac@johnnymac-VirtualBox:~/Projects/V2/AppSecAssignment1.1$ ./giftcardread
er 1 part3/fuzzer_hang.gft
   Merchant ID: GiftCardz.com
   Customer ID: DuaneGreenes Store 1451
   Num records: 1
      record_type: animated message
      message: �
  [running embedded program]
```

To fix this hang, I changed the signed character arg1 variable to an unsigned character value, so the 253 value, or any value greater than 127, would not wrap around into a negative number and cause the program to hang.



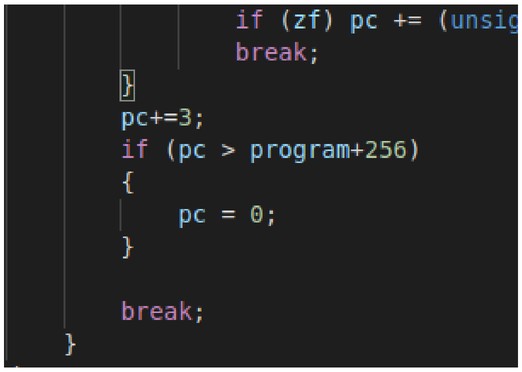
```
case 0x10:
    if (zf) pc += (unsigned char)arg1;
    break;
}
```

For the crash file (fuzzer2.gft), I again noticed that the crash used the animate function to cause the crash within the program. Using print statements and GDB, I believe the cause of the segmentation fault is due to a large number being passing into the message band of the animate function. This causes issues in both the print_gift_card_info function and the gift_card_json function, as the message value is called to display the message. If a large enough value is passed, the number will wrap around to a negative value, and the program will instead read past the end of the file, causing a segmentation fault.



```
johnnymac@johnnymac-VirtualBox:~/Projects/V2/AppSecAssignment1.1$ ./giftcardread
er 1 part3/fuzzer_crash.gft
There was a negative byte value in the num_bytes variable. Error #1
   Merchant ID:
   Customer ID:
   Num records: 3
      record_type: animated message
      message:
  [running embedded program]
      record_type: animated message
Segmentation fault
```

To fix this issue, I added a simple line to the if (pc > program+256) line, that sets the pc value back to zero. That way, if the pc variable goes over the program+256 value, it can be reset back down to zero and can increment back up from there, preventing the segmentation fault.



```
            if (zf) pc += (unsig
            break;
        }
        pc+=3;
        if (pc > program+256)
        {
            pc = 0;
        }

        break;
    }
```