Kevin Zhang
Application Security CS GY 9163

Homework 1: Beware if Geeks Bearing Gift Cards

On NYU Classes, submit a link to your GitHub repository. The repository should be **private**, and you should add the instructor/TA's GitHub account as a contributor to give them access for grading.

For this section, your instructor is: Kevin Gallagher, GitHub ID **`kcg295`**

Your TA is: Evan Richter, GitHub ID **`evanrichter`**

The repository should contain:

Part 1: Setting up Your Environment

Relatively self-explanatory. The steps for Part 1 are detailed below:

GitHub Set-up

- Git-VCS has been installed.
- Git Bash has been installed (Windows)
- Git signed commits have been set-up
- GitHub Account created – kzhang112
- GitHub-Repo created ("AppSec_1.1") – set to "private"

Travis CI

- Travis-CI account created and tied to GitHub
- .travis.yml file created
- Iterations (changes of code) Travis builds until build is successful

Part 2: Auditing and Test Cases

Set-up

- Files added to GitHub Repo
  - "*examplefile.gft*"
  - "*giftcardexamplewriter.c*"
  - "giftcard.h"
  - "*giftcardreader.c*"
  - "*Makefile"*

After adding the above files to my GitHub Repo ("kzhang112/AppSec1.1"), I attempted to run both "*giftcardreader.c"* and "*giftcardwriter.c*" using my Visual Studio Code compiler (I ran the program through the Mingw-x32 for Windows). "*Giftcardwriter.c*" did not return any errors prior to running the program, and I was able to print out an example gift card (similar to the gift card provided in the original repository).

However, "*giftcardreader.c"* ran into an initial error indicating that I needed to include a string header.

```
giftcardreader.c:241:13: warning: incompatible implicit declaration of built-in function 'strlen'
giftcardreader.c:14:1: note: include '<string.h>' or provide a declaration of 'strlen'
   13 | #include <strings.h>
  +++ |+#include <string.h>
   14 |
giftcardreader.c:248:17: warning: implicit declaration of function 'memcpy' [-Wimplicit-function-declaration]
  248 |                 memcpy(gcp_ptr->message, ptr, 32);
      |                 ^~~~~~
giftcardreader.c:248:17: warning: incompatible implicit declaration of built-in function 'memcpy'
giftcardreader.c:248:17: note: include '<string.h>' or provide a declaration of 'memcpy'
```

After adding in "#include <string.h>", I encountered another error.

```
giftcardreader.c: In function 'print_gift_card_info':
giftcardreader.c:102:33: warning: implicit declaration of function 'get_gift_card_value' [-Wimplicit-function-declaration]
  102 |  printf("  Total value: %d\n\n",get_gift_card_value(thisone));
      |                                 ^~~~~~~~~~~~~~~~~~~~
```

By adding in an extra int for get_gift_card_value prior to the code block where we see the error, it gives the compiler information that can be used later in the program (e.g. in this case, on line 102).

```
69  //KZ: Adding in this extra int-declaration resolves second run-time error
70  int get_gift_card_value (struct this_gift_card *thisone);
71
```

Finally, I was able to build and run the file on the example gift card that I had just created.

Find Flaws

- "*Crash_1.gft*" created

COME UP WITH SCREENSHOT ON HOW TO REPRODUCE CRASH GIFT CARD FILE

```
PS C:\c_projects\AppSec_1.1DEV> ./giftcardreader 1 crash_1.gf
   Merchant ID: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   Customer ID: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
   Num records: 1
      record_type: animated message
      message: testtesttesttesttesttesttesttestG-mLa
  [running embedded program]
  Total value: 0
```

This initial has to do with the op-code 0x01 function on line 30 of "giftcardreader.c". After examining the code, I noticed that 0x01 pushes forward "regs[arg1] = *mptr". Mptr is a pointer for the beginning of the message data, while arg1 provides the value for the register (e.g. regs[]).

Similar to the hang case below, this function interfaces with the "arg1 value" of the giftcard. If we set the arg1 value to anything higher than 16-bytes (storage capacity), we can trigger a segmentation fault and crash the program - this is due to a uffer overflow.

- “*Crash_2.gft*” created

```
void setupgc() {
    examplegc.num_bytes = -1234;
    examplegc.gift_card_data = (void *) &examplegcd;
    examplegcd.merchant_id = "GiftCardz.com
    examplegcd.customer_id = "DuaneGreenes Store 1451
    examplegcd.number_of_gift_card_records = 1;
```

```
PS C:\c_projects\AppSec_1.1DEV> ./giftcardreader 1 crash1fraction.gft
PS C:\c_projects\AppSec_1.1DEV> ^C
PS C:\c_projects\AppSec_1.1DEV> ./giftcardreader 1 crash1negative.gft
PS C:\c_projects\AppSec_1.1DEV> ^C
PS C:\c_projects\AppSec_1.1DEV>
```

This crash is caused by setting "num_bytes" section of 'giftcardwriter.c' to anything aside from a normal integer. Inputting a large negative number (e.g. -1234) creates a malformed gift card that can cause the malloc to break in 'giftcardreader.c', as this is outside the range of integers accepted.

“Giftcardexample.num_bytes” stores the negative number, which is then used for memory allocation within the reader code. If we try to run the malformed gift card, we receive a segmentation fault (outputted by Travis when running it through a GitHub push), due to the fact that the fread cannot read anything negative. Adding in error handling for the negative number will cause the program to exit without crashing, while also informing the user of the error.

- “*Hang.gft*” created

```
void causehang()
{
    crash.message = malloc(32);
    crash.program = malloc(128);
    crash.program[0] = 0x09;
    crash.program[1] = 253;
}
```

```
PS C:\c_projects\AppSec_1.1DEV> ./giftcardreader 1 hang.gft
   Merchant ID: GiftCardz.com
   Customer ID: DuaneGreenes Store 1451
   Num records: 3
      record_type: animated message
      message: 1
 [running embedded program]
```

The hanging program is leveraging a new function, titled causehang() in "giftcardwriter.gft," to force the program to loop instructions. This specifically refers to case opcode 0x09, referring to animate function. We alter the number of records of the card from 1 to 3 in this case, causing a loop (no chance of exit, error, or satisfying a condition). This is namely due to the presence of arg1 (in the line "pc += (char)arg1," line 64). Arg1 accepts 8bit integer values; however, does not know how to read extraneous information - an extra byte registers as a "-" sign.

Because of case opcode 0x09, memory location will keep increasing by "3" until a limit has been reached (256). If we input a specific Arg1 value (in this case 250), memory location will decrease, increase, and decrease again due to actions conducted in the beginning and end of the while loop (lines 22-61). Our fix, therefore, is to alter arg1 (specifically the code on line 64) to be and unsigned character, which would break in the case of a negative value.

- *"Bugs.txt"* created, describing the above issues
- Bugs highlighted in "*Bugs.txt*"fixed in "*giftcardreader.c*"

Part 3: Fuzzing and Coverage

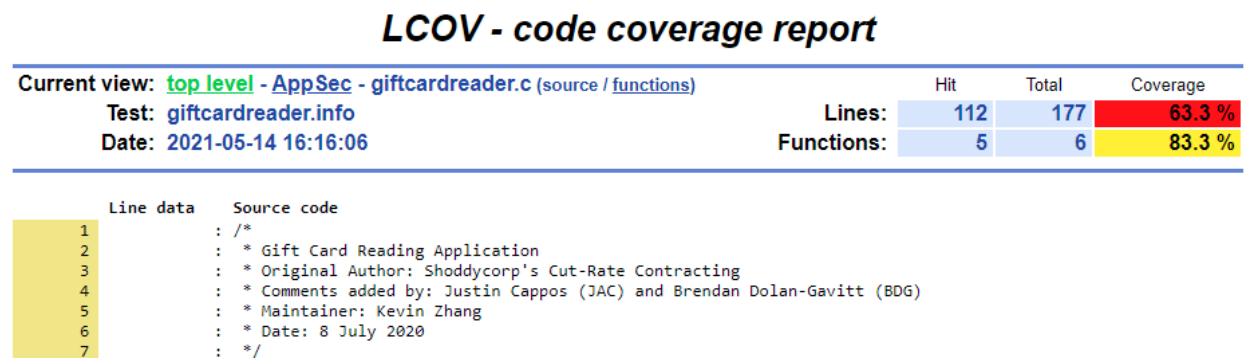
In order for me to utilize GCC with LCOV, I needed to switch systems to my lightly used (and admittedly less powerful) Macbook Air. I proceeded to install the packages using *"brew install"* below:

- LCOV
- GCC
- AFL-Clang
- AFL-fuzz

After installation, I went ahead and ran gcc -coverage on my giftcardreader.c. The commands were as follows:

- gcc -g --coverage giftcardreader.c -o giftcardreader
- ./giftcardreader 1 crash_1.gft (within gcc, along with all the other test cases)
- Gcov giftcardreader
- Lcov -c -d . -o giftcardreader.info
- Genhtml giftcardreader.info -o giftcardreader_report.

Doing this, I was able to pull a HTML representation of the amount of coverage of the report. My file showed that I had around 63.3% line coverage, and 83.3% function coverage (please see below screenshot).



LCOV - code coverage report

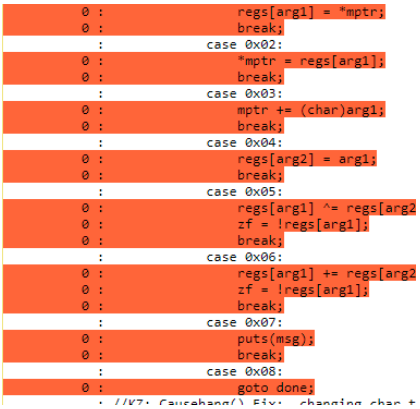
		Hit	Total	Coverage
Current view: top level - AppSec - giftcardreader.c (source / functions)				
Test: giftcardreader.info	Lines:	112	177	63.3 %
Date: 2021-05-14 16:16:06	Functions:	5	6	83.3 %

```
      Line data    Source code
    1            : /*
    2            :  * Gift Card Reading Application
    3            :  * Original Author: Shoddycorp's Cut-Rate Contracting
    4            :  * Comments added by: Justin Cappos (JAC) and Brendan Dolan-Gavitt (BDG)
    5            :  * Maintainer: Kevin Zhang
    6            :  * Date: 8 July 2020
    7            :  */
```

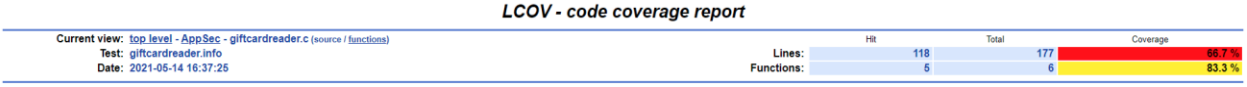
In order to increase my coverage, I parsed the code and found a few lines that stuck out particularly to me. The lines I chose were lines 35-71, and line 292.

```
290          8 :     if (argv[1][0] == '1') print_gift_card_info(thisone);
291            :
292          0 :     else if (argv[1][0] == '2') gift_card_json(thisone);
```

These two selections were areas not covered by the initial gcc –coverage run. I decided to create two gift cards – one that covered the op-code cases (examining what happens if we insert type_of_record = 3) and one that tested running a type "2" record (all of the previous test cases had been using type 1 records). As you can see on the image on the right, very few of the op-code cases are covered.



With this, the coverage of files increased from 63.3% to 66.7%. A marginal increase, but still depicting the added benefit of creating more test cases. The areas that were previously uncovered are now covered under the new test cases.



Coverage 1- After running gcc --coverage on giftcardreader.c, and testing the existing crashes and hangs, we were able to see that there was about 46.6% of the code that was uncovered. Many of our test cases did not touch the "opcode" section. Our hang.gft error was the closes, noticing that a never ending loop of sending and receiving information can be achieved if we took advantage of a vulnerability in 0x09. In our coverage case (as 0x02 - 0x07, 0x09 were all uncovered), we sought to emulate the hang.gft; instead, we simply changed the type_of_record setting in the gift card from 1, to 3. This specifically references animate function. While the coverage is minimal, there is an increase in line coverage (function coverage remains the same).

Coverage 2- In our second coverage report, we noticed that at the end of the code, there specifically is a function that accepts type 2 records (calling the gift_card_json file instead of the print_gift_card_file). Keeping this code in mind ("else if (argv[1][0] == '2') gift_card_json(thisone);") we run a simple examplefile.gft that utilizes a type 2 record, rather than a type 1 record. Although a simple fix, we wanted to make sure we maximized the coverage. This produced a minimal coverage as well.

Afterwards, we proceeded to run AFL on the system to fuzz the program even further. After installing AFL (and dealing with some OSX permissions issues), we ran the commands "afl-clang giftcardreader.c -o giftcard reader" and "afl-fuzz -i in out -m 200 ./giftcardreader 1 @@" to begin the fuzzing process. After 30 minutes, we picked up a total of 5 unique crashes and 19 total crashes, showing that AFL is indeed doing what we want of it.

Fuzzer_1: After running our AFL fuzzer for more than two hours, we selected two Fuzzer gift card files that best improve our code. We found an afl-fuzz hang bug, which causes the program to enter a continues cycle of sending and receiving instructions, like the last example. We found that the main reason this occurs is because of a value within the .gft file, which comes from another unsigned char. After we prevent the writing of a negative value, we found that the program was able to work.

Fuzzer_2: A crash found by AFL fuzzer specifically has to do with opcode 0x04, 0x06 - this creates a buffer overflow (similar to a crash found in the previous set of crashes) and stops the program without exiting. We attempted the fix by specifying exactly which bounds for those op-code cases args1 needs to fall under. This paired with changing chars to unsigned char seem to be a major fault in the program.

We fixed these errors, and then ran gcc –coverage on them once again, producing a new coverage report (all three coverage report folders are included in the GitHub Repo). As you can see, we increased both the amount of lines and functions by adding additional test cases, and running the test cases on both memory type 1 and 2.

LCOV - code coverage report

		Hit	Total	Coverage
Current view: top level - AppSec - giftcardreader.c (source / functions)	Lines:	158	181	87.3 %
Test: giftcardreader.info	Functions:	6	6	100.0 %
Date: 2021-05-15 05:24:00				

As shown on the right, the op-code section, which had previously been filled with uncovered elements, is now mostly covered by our test and fixes. With this, we have the best version of our test yet.

After adding the test cases to my test suite, we pushed the commit to the repo.

```
32          :                 case 0x00:
33        286 :                     break;
34          : //KZ: Crash_1 fix: We are stating that the length of arg1 must be le
35          :                 case 0x01 :
36          4 :                     if (arg1 < 0 || arg1 >= 16)
37          :                     {
38          2 :                         printf("\nError: Malformed Card\n");
39          2 :                         break;
40          :                     }
41          2 :                     regs[arg1] = *mptr;
42          2 :                     break;
43          :                 case 0x02:
44          0 :                     *mptr = regs[arg1];
45          0 :                     break;
46          :                 case 0x03:
47          : //KZ: Fuzzer_1.gft fix: changing to unsigned char to fix
48         16 :                     mptr += (unsigned char)arg1;
49         16 :                     break;
50          :                 case 0x04:
51          : //KZ: Fuzzer_2.gft fix: need to create better bounds for arg1
52          0 :                     if (arg1 < 16 && arg2 < 16) {
53          0 :                         regs[arg2] = arg1;
54          0 :                     }
55          0 :                     break;
56          :                 case 0x05:
57          4 :                     regs[arg1] ^= regs[arg2];
58          4 :                     zf = !regs[arg1];
59          4 :                     break;
60          :                 case 0x06:
61          : //KZ: Fuzzer_2.gft fix: need to create better bounds for arg1
62          0 :                     if (arg1 < 16 && arg2 < 16) {
63          0 :                         regs[arg1] += regs[arg2];
64          0 :                         zf = !regs[arg1];
65          0 :                     }
66          0 :                     break;
67          :                 case 0x07:
68          2 :                     puts(msg);
69          2 :                     break;
70          :                 case 0x08:
71          0 :                     goto done;
72          : //KZ: Causehang() Fix:, changing char to unsigned char, blocking neg
73          :                 case 0x09:
74          2 :                     pc += (unsigned char)arg1;
75          2 :                     break;
76          : //KZ: Fuzzer_1.gft fix: changing to unsigned char to fix
77          :                 case 0x10:
78          6 :                     if (zf) pc += (unsigned char)arg1;
79          6 :                     break;
80          :             }
81        584 :             pc+=3;
82        584 :             if (pc > program+256) break;
83          :         }
84          : done:
85         10 :         return;
86          : }
```