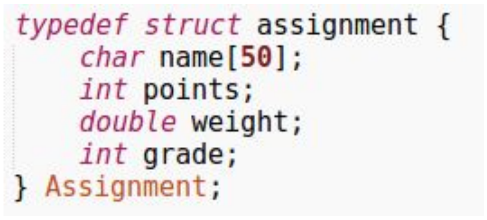
[redacted]

Project 4: Build It

Design Approach

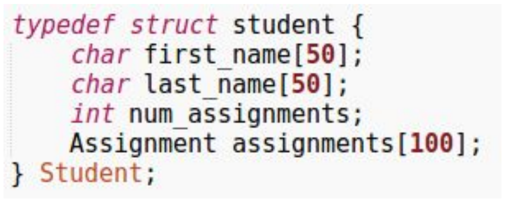
The Secure Gradebook I built is divided into three programs: setup.c, gradebookadd.c, and gradebookdisplay.c. The Gradebook is stored as a C struct, with lists of Students and Assignments (which are also C structs). All of the structs are defined in gradebookobjects.h.

- Assignment struct



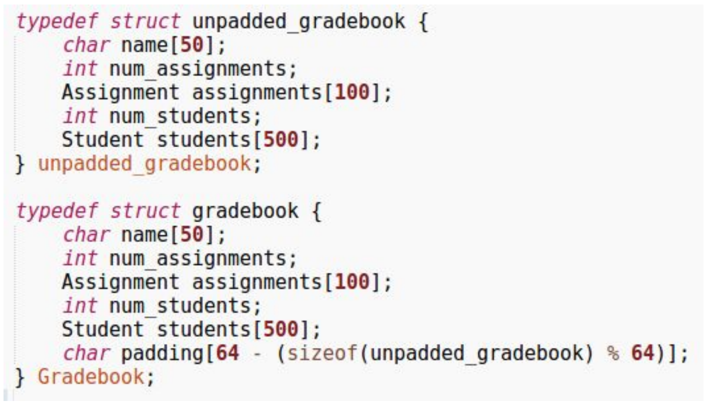
```c
typedef struct assignment {
    char name[50];
    int points;
    double weight;
    int grade;
} Assignment;
```

- Student struct



```c
typedef struct student {
    char first_name[50];
    char last_name[50];
    int num_assignments;
    Assignment assignments[100];
} Student;
```

- Gradebook struct



```c
typedef struct unpadded_gradebook {
    char name[50];
    int num_assignments;
    Assignment assignments[100];
    int num_students;
    Student students[500];
} unpadded_gradebook;

typedef struct gradebook {
    char name[50];
    int num_assignments;
    Assignment assignments[100];
    int num_students;
    Student students[500];
    char padding[64 - (sizeof(unpadded_gradebook) % 64)];
} Gradebook;
```

Note: The gradebook struct includes padding so that it can be properly encrypted (this will be described later in this write-up)

Each of the structs above use fixed lengths for any array. *The following assumptions were made*:

- Assignment names are limited to a maximum of 50 characters
- Student first names are limited to a maximum of 50 characters

- Student last names are limited to a maximum of 50 characters
- Each student can have a maximum of 100 assignments
- Gradebook filenames are limited to a maximum of 50 characters
- Each gradebook can have a maximum of 100 assignments
- Each gradebook can have a maximum of 500 students

These array lengths are fixed because it allows the program to validate any input and prevent buffer overflows, which will be described later in this write-up).

The TAs and Professors can create a new gradebook using setup.c, modify an existing gradebook using gradebookadd.c, and read an existing gradebook using gradebookdisplay.c. Here is how each of the programs work internally:

1. setup.c
    a. If a valid filename is provided, and a gradebook with the provided filename does not already exist in the current directory, **the program opens a new file with the provided filename**.
    b. **The program creates a new, empty instance of the Gradebook struct: new_gb**. The new instance of the Gradebook contains 0 assignments and 0 students.
    c. **The program generates a random 128-bit IV** using openssl's RAND_bytes function.
    d. **The program generates a random 128-bit encryption key** using openssl's RAND_bytes function.
    e. **The program generates a random 128-bit MAC key** using openssl's RAND_bytes function.
    f. **The program prints a 256-bit random key to the user** (the first 128 bits are the encryption key, and the second 128 bits are the MAC key).
    g. **The program encrypts new_gb (the Gradebook struct)** with the encryption key and IV and stores it in a buffer called ciphertext. Encryption is done using openssl functions.
    h. **The program runs the MAC algorithm with the randomly generated MAC key and produces a tag for the Gradebook encryption** using openssl's MAC function and EVP_md5 hash. The tag is 128-bits in length.
    i. **The program writes the ciphertext (the encrypted Gradebook) to the new gradebook file**.
    j. **The program writes the IV to the new gradebook file**.
    k. **The program writes the tag produced by MAC to the gradebook file**.
    l. **The program writes the length of the ciphertext to the gradebook file**.
2. gradebookadd.c
    a. If a valid filename is provided, and a gradebook with the provided filename exists in the current directory, **the program opens the file with the provided filename**.
    b. **The program reads the following information from the gradebook file**:
        i. Ciphertext
        ii. IV

   iii. MAC tag
   iv. Ciphertext length

c. **The program reads the provided 256-bit key from the command line**, and splits it into two parts (128-bit encryption key and 128-bit MAC key).
d. **The program runs the MAC algorithm with the MAC key from the command-line and produces a tag for the ciphertext** using openssl's MAC function and EVP_md5 hash. The tag is 128-bits in length.
e. **The program compares the MAC tag from the gradebook file to the MAC tag that was generated with the ciphertext**. If they are the same, the gradebook file was NOT modified.
f. **The program decrypts the ciphertext** with the ciphertext, ciphertext length, encryption key, and IV using openssl functions. **The program stores the decryption into a Gradebooks struct**.
g. **The program reads the rest of the inputs from the command line** and determines if a valid combination has been provided.
h. **The program performs one of the following actions by modifying the gradebook**:
   i. Add assignment
      1. The program creates a new instance of the Assignment struct with the user's provided inputs and adds it to the Gradebook assignments array. The program increments the Gradebook num_assignments property by 1.
      2. The program adds new instances of the Assignment struct with the user's provided inputs to each of the students' assignments array. Each of the students' num_assignments property is increased by 1. Each of the students' grades on the assignment is initialized to 0.
   ii. Delete assignment
      1. The program goes through the Gradebook assignments array and looks for an assignment with the same name that the user provided. If an assignment with the same name exists, it is removed from the Gradebook by shifting all following assignments up by one index and decrementing the Gradebook num_assignments property.
      2. The program goes through each students' assignments array and looks for an assignment with the same name that the user provided. If an assignment with the same name exists, it is removed from the students' assignments array by shifting all following assignments up by one index and decrementing the students' num_assignments property.
   iii. Add student
      1. The program creates a new instance of the Student struct with the user's input and adds it to the Gradebook students array. The Student struct is initialized with all of the assignments that are in

the Gradebook assignments array (with a starting grade of 0). The program increments the Gradebook num_students property.

  iv. Delete student
    1. The program goes through the Gradebook students array and looks for a student with the same first and last name that the user provided. If a student with the same name exists, it is removed from the Gradebook's students array by shifting all following students up by one index and decrementing the Gradebook's num_students property.
  v. Add grade
    1. The program goes through the Gradebooks students array and looks for a student with the same first and last name that the user provided. If a student with the same name exists, the program goes through the student's assignment array and looks for an assignment with the same name that the user provided. If an assignment with the same name exists, the student's grade on the assignment is updated with the user's input.

i. After performing the action and modifying the gradebook, **the program opens a fresh file with the same name as the original gradebook**.
j. **The program generates a random 128-bit IV** using openssl's RAND_bytes function.
k. **The program encrypts the Gradebook struct** with the encryption key from the command-line and randomly-generated IV and stores it in a buffer called ciphertext. Encryption is done using openssl functions.
l. **The program runs the MAC algorithm with the MAC key from the command-line and produces a tag for the Gradebook encryption** using openssl's MAC function and EVP_md5 hash. The tag is 128-bits in length.
m. **The program writes the ciphertext (the encrypted Gradebook) to the gradebook file**.
n. **The program writes the randomly-generated IV to the gradebook file**.
o. **The program writes the tag produced by MAC to the gradebook file**.
p. **The program writes the length of the ciphertext to the gradebook file**.

3. gradebookdisplay.c
   a. If a valid filename is provided, and a gradebook with the provided filename exists in the current directory, **the program opens the file with the provided filename**.
   b. **The program reads the following information from the gradebook file**:
      i. Ciphertext
      ii. IV
      iii. MAC tag
      iv. Ciphertext length
   c. **The program reads the provided 256-bit key from the command line**, and splits it into two parts (128-bit encryption key and 128-bit MAC key).

d. **The program runs the MAC algorithm with the MAC key from the command-line and produces a tag for the ciphertext** using openssl's MAC function and EVP_md5 hash. The tag is 128-bits in length.
e. **The program compares the MAC tag from the gradebook file to the MAC tag that was generated with the ciphertext**. If they are the same, the gradebook file was NOT modified.
f. **The program decrypts the ciphertext** with the ciphertext, ciphertext length, encryption key, and IV using openssl functions. **The program stores the decryption into a Gradebook struct**.
g. **The program reads the rest of the inputs from the command line** and determines if a valid combination has been provided.
h. **The program performs one of the following actions by reading the gradebook**:
    i. Print assignment
        1. The program goes through the Gradebook assignments array to check that an assignment with the provided name exists in the Gradebook.
        2. The program copies all of the Gradebook students to a different array.
        3. If the user provided the -A flag
            a. The program sorts the copied students array by last name, first name
        4. If the user provided the -G flag
            a. The program sorts the copied students array by grade
        5. For each student in the copied students array, the program goes through the student's assignments array and locates the assignment with the name the user provided. The program prints the students last name, first name, and grade on the assignment to a new line.
    ii. Print student
        1. The program goes through the Gradebook's students array to find a student with the user provided first and last names. If a student with the provided name exists, the program goes through each assignment in the student's gradebook and prints the assignment name and grade on a new line.
    iii. Print final
        1. The program copies all of the Gradebook students to a different array.
        2. If the user provided the -A flag
            a. The program sorts the copied students array by last name, first name
        3. If the user provided the -G flag
            a. The program sorts the copied students array by final grade

4. The program goes through the copied students array and prints each student's last name, first name, and final grade on a new line.

i. After performing the action, **the program closes the gradebook**.

While implementing the specifications of this project, *the following assumptions were made*:

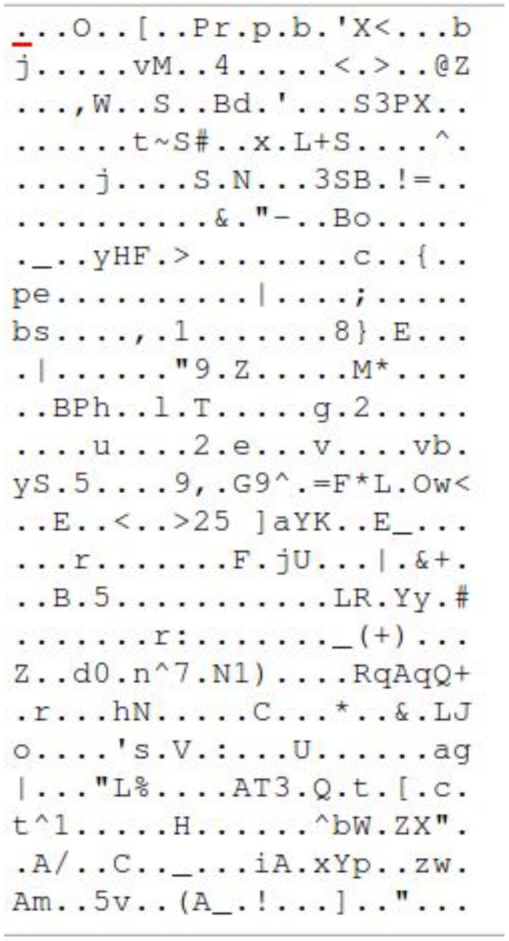
- If a flag expects an input such as (-AN <assignment name>), and no such input is provided, the program will exit and print "invalid".
- If extra options are provided, but the required options are provided, then the extra options will be ignored.

Attacks

This secure gradebook program protects against the following attacks.

Attack 1: Read From Cloud

In this attack, a student would like to see the grades of all the other students in the class. The student is able to gain access to the cloud where the course gradebook is stored. They obtain the gradebook file, and the student is able to read the contents of the file. However, reading the contents of the file does **NOT** tell the student any useful information. When the student opens the file, they will see the encryption of the gradebook:

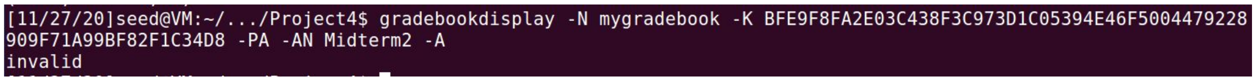


```
...O..[..Pr.p.b.'X<...b
j.....vM..4.....<.>..@Z
...,W..S..Bd.'...S3PX..
......t~S#..x.L+S....^.
....j....S.N...3SB.!=..
..........&."-..Bo.....
._..yHF.>........c..{..
pe..........|....;.....
bs....,.1.......8}.E...
.|......"9.Z.....M*....
..BPh..l.T.....g.2.....
....u....2.e...v....vb.
yS.5....9,.G9^.=F*L.Ow<
..E..<..>25 ]aYK..E_...
...r.......F.jU...|.&+.
..B.5...........LR.Yy.#
.......r:........_(+)...
Z..d0.n^7.N1)....RqAqQ+
.r...hN.....C...*..&.LJ
o....'s.V.:...U......ag
|..."L%....AT3.Q.t.[.c.
t^1.....H......^bW.ZX".
.A/..C.._...iA.xYp..zw.
Am..5v..(A_.!...]..."...
```

This program guarantees the privacy of the gradebook data by using AES encryption. Before being written to the gradebook file, the gradebook is encrypted with a 128-bit key (which only the TAs and Professors have access to) and a random 128-bit IV (which is included in the gradebook file). The 128-bit key is securely and randomly generated in setup.c using rand_bytes and printed for the TA and Professor to see. The 128-bit IV is securely and randomly generated each time the gradebook is modified. The lines of code that encrypt the gradebook are:

- In setup.c: lines 247-248
- In gradebookadd.c: lines 580-581

If the student would like to view the contents of the gradebook using the gradebookdisplay.c program, they must provide the correct key. If the student provides an incorrect key, as seen below, the program will exit and return “invalid”.

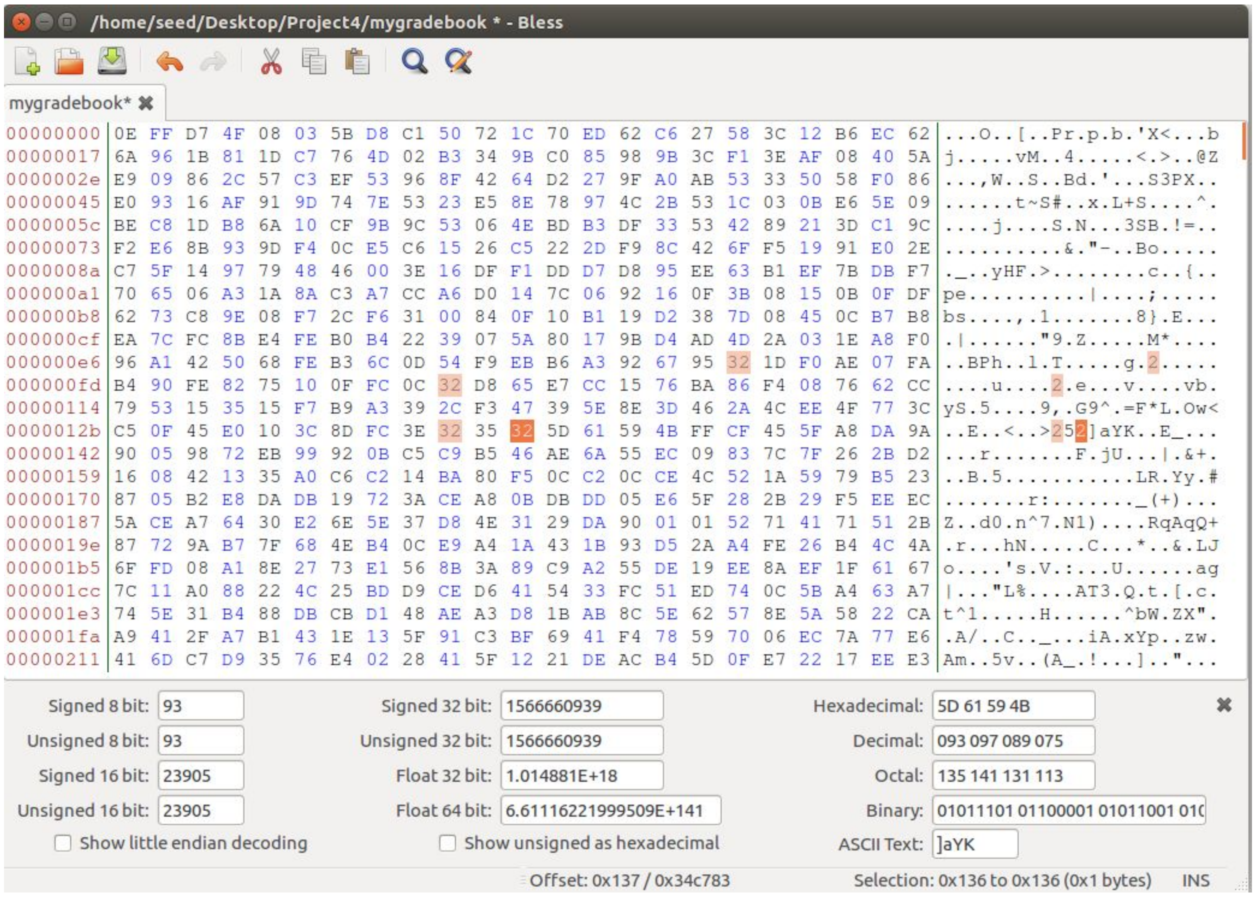


```
[11/27/20]seed@VM:~/.../Project4$ gradebookdisplay -N mygradebook -K BFE9F8FA2E03C438F3C973D1C05394E46F5004479228
909F71A99BF82F1C34D8 -PA -AN Midterm2 -A
invalid
```

AES-128 encryption is considered secure, so it will take a significant amount of time for any attacker to brute force the key. The amount of time to brute force the correct key is so long that an attacker would not be able to perform such an attack.

Attack 2: Modify the Gradebook

In this attack, a student who got a bad grade on the midterm would like to change their grade. The student is able to gain access to the cloud where the course gradebook is stored. They obtain the gradebook file, and the student is able to change the contents of the file. However, this attack will not be successful. If the user modifies even one byte of the gradebook file, when the professor or TA runs the gradebookadd or gradebookdisplay programs, the program will exit and return "invalid". Here is a screenshot of a student modifying a byte in the gradebook file:



/home/seed/Desktop/Project4/mygradebook * - Bless

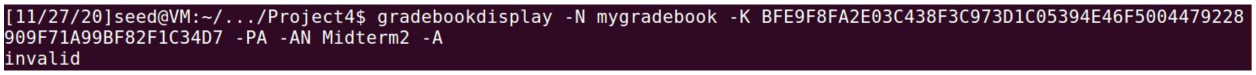
mygradebook*

```
00000000 0E FF D7 4F 08 03 5B D8 C1 50 72 1C 70 ED 62 C6 27 58 3C 12 B6 EC 62 ...O..[..Pr.p.b.'X<...b
00000017 6A 96 1B 81 1D C7 76 4D 02 B3 34 9B C0 85 98 9B 3C F1 3E AF 08 40 5A j.....vM..4.....<.>..@Z
0000002e E9 09 86 2C 57 C3 EF 53 96 8F 42 64 D2 27 9F A0 AB 53 33 50 58 F0 86 ...,W..S..Bd.'...S3PX..
00000045 E0 93 16 AF 91 9D 74 7E 53 23 E5 8E 78 97 4C 2B 53 1C 03 0B E6 5E 09 ......t~S#..x.L+S....^.
0000005c BE C8 1D B8 6A 10 CF 9B 9C 53 06 4E BD B3 DF 33 53 42 89 21 3D C1 9C ....j....S.N...3SB.!=..
00000073 F2 E6 8B 93 9D F4 0C E5 C6 15 26 C5 22 2D F9 8C 42 6F F5 19 91 E0 2E ..........&."-..Bo.....
0000008a C7 5F 14 97 79 48 46 00 3E 16 DF F1 DD D7 D8 95 EE 63 B1 EF 7B DB F7 ._..yHF.>........c..{..
000000a1 70 65 06 A3 1A 8A C3 A7 CC A6 D0 14 7C 06 92 16 0F 3B 08 15 0B 0F DF pe..........|....;.....
000000b8 62 73 C8 9E 08 F7 2C F6 31 00 84 0F 10 B1 19 D2 38 7D 08 45 0C B7 B8 bs....,.1.......8}.E...
000000cf EA 7C FC 8B E4 FE B0 B4 22 39 07 5A 80 17 9B D4 AD 4D 2A 03 1E A8 F0 .|......"9.Z.....M*....
000000e6 96 A1 42 50 68 FE B3 6C 0D 54 F9 EB B6 A3 92 67 95 32 1D F0 AE 07 FA ..BPh..l.T.....g.2.....
000000fd B4 90 FE 82 75 10 0F FC 0C 32 D8 65 E7 CC 15 76 BA 86 F4 08 76 62 CC ....u....2.e...v....vb.
00000114 79 53 15 35 15 F7 B9 A3 39 2C F3 47 39 5E 8E 3D 46 2A 4C EE 4F 77 3C yS.5....9,.G9^.=F*L.Ow<
0000012b C5 0F 45 E0 10 3C 8D FC 3E 32 35 32 5D 61 59 4B FF CF 45 5F A8 DA 9A ..E..<..>252]aYK..E_...
00000142 90 05 98 72 EB 99 92 0B C5 C9 B5 46 AE 6A 55 EC 09 83 7C 7F 26 2B D2 ...r.......F.jU...|.&+.
00000159 16 08 42 13 35 A0 C6 C2 14 BA 80 F5 0C C2 0C CE 4C 52 1A 59 79 B5 23 ..B.5...........LR.Yy.#
00000170 87 05 B2 E8 DA DB 19 72 3A CE A8 0B DB DD 05 E6 5F 28 2B 29 F5 EE EC .......r:......._(+)...
00000187 5A CE A7 64 30 E2 6E 5E 37 D8 4E 31 29 DA 90 01 01 52 71 41 71 51 2B Z..d0.n^7.N1)....RqAqQ+
0000019e 87 72 9A B7 7F 68 4E B4 0C E9 A4 1A 43 1B 93 D5 2A A4 FE 26 B4 4C 4A .r...hN.....C...*..&.LJ
000001b5 6F FD 08 A1 8E 27 73 E1 56 8B 3A 89 C9 A2 55 DE 19 EE 8A EF 1F 61 67 o....'s.V.:...U......ag
000001cc 7C 11 A0 88 22 4C 25 BD D9 CE D6 41 54 33 FC 51 ED 74 0C 5B A4 63 A7 |..."L%....AT3.Q.t.[.c.
000001e3 74 5E 31 B4 88 DB CB D1 48 AE A3 D8 1B AB 8C 5E 62 57 8E 5A 58 22 CA t^1.....H......^bW.ZX".
000001fa A9 41 2F A7 B1 43 1E 13 5F 91 C3 BF 69 41 F4 78 59 70 06 EC 7A 77 E6 .A/..C.._...iA.xYp..zw.
00000211 41 6D C7 D9 35 76 E4 02 28 41 5F 12 21 DE AC B4 5D 0F E7 22 17 EE E3 Am..5v..(A_.!...]..."...
```

Signed 8 bit:	93	Signed 32 bit:	1566660939	Hexadecimal:	5D 61 59 4B
Unsigned 8 bit:	93	Unsigned 32 bit:	1566660939	Decimal:	093 097 089 075
Signed 16 bit:	23905	Float 32 bit:	1.014881E+18	Octal:	135 141 131 113
Unsigned 16 bit:	23905	Float 64 bit:	6.61116221999509E+141	Binary:	01011101 01100001 01011001 010
☐ Show little endian decoding		☐ Show unsigned as hexadecimal		ASCII Text:	]aYK

Offset: 0x137 / 0x34c783   Selection: 0x136 to 0x136 (0x1 bytes)   INS

Here is a screenshot of what happens when the professor or TA try to read the modified gradebook:



```
[11/27/20]seed@VM:~/.../Project4$ gradebookdisplay -N mygradebook -K BFE9F8FA2E03C438F3C973D1C05394E46F5004479228
909F71A99BF82F1C34D7 -PA -AN Midterm2 -A
invalid
```

This program guarantees the integrity of the gradebook data by using the openssl HMAC function. Before the encryption of the gradebook is written to the gradebook file, the program generates a tag of the encryption with a 128-bit key (which only the TAs and Professors have access to) and the EVP_md5() hash. The encryption and tag are then written to the gradebook file. The 128-bit key is securely and randomly generated in setup.c using rand_bytes and printed for the TA and Professor to see. The lines of code that generate the tag are:

- In setup.c: line 253
- In gradebookadd.c: line 586

Whenever the gradebookadd.c and gradebookdisplay.c programs read from a gradebook file, the programs use the openssl HMAC function to generate a tag with the ciphertext from the gradebook file and the key that the TA or professor provides. If the tag that gradebookadd.c and gradebookdisplay.c generates is the same as the tag in the gradebook file, then the gradebook file was not tampered with. However, if the tags are different, then the gradebook file was altered and cannot be trusted, so the program exits and returns “invalid”. The lines of code that compare the tags are:

- In gradebookadd.c: lines 276-280
- In gradebookdisplay.c: lines 271-275

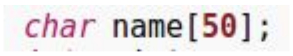
Attack 3: Buffer Overflow

In this attack, the attacker tries to alter the behavior of the programs by overwriting pointers on the stack. In this case, the attacker was able to identify where on the stack the return register is located using gdb. The attacker would like to overwrite the return register so it launches some other function (which is not the expected behavior of this program). The attacker would like to overwrite the return register using the input that each of the programs take. The following options take in an input that is written to a buffer in the program:

- -N <filename>
- -K <key>
- -AN <assignment name>
- -FN <first name>
- -LN <last name>

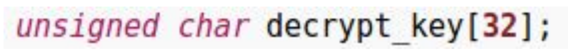
Each of the buffers which store these inputs are different lengths. The lengths are as follows:

- -N <filename>



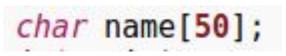
```
char name[50];
```

- -K <key>



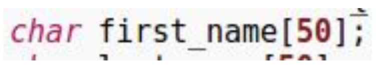
```
unsigned char decrypt_key[32];
```

- -AN <assignment name>



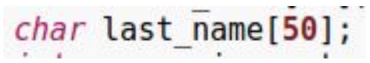
```
char name[50];
```

- -FN <first name>



```
char first_name[50];
```

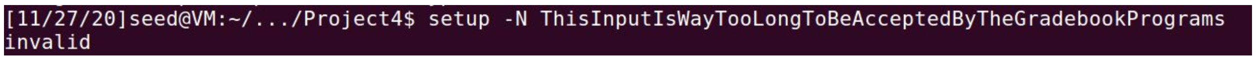
- -LN <last name>



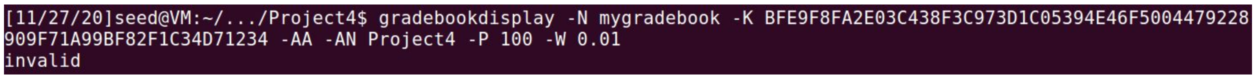
```
char last_name[50];
```

When reading the user's inputs from the command line, the program checks that the lengths of the input are valid before writing the input into the program buffers. If the lengths of the inputs are too long, the program exits and returns "invalid".

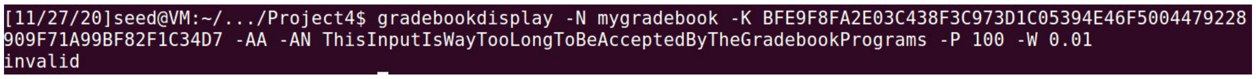
If the attacker attempts to overwrite the return register by inputting data that is longer than the buffer, the program will exit and the attack will be unsuccessful. This can be seen in the following cases:



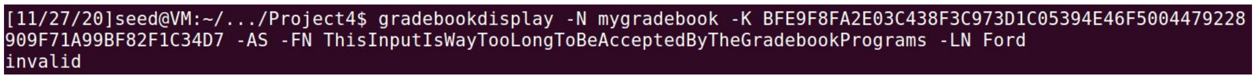
```
[11/27/20]seed@VM:~/.../Project4$ setup -N ThisInputIsWayTooLongToBeAcceptedByTheGradebookPrograms
invalid
```



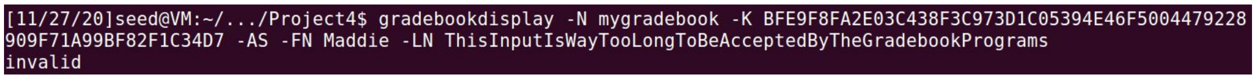
```
[11/27/20]seed@VM:~/.../Project4$ gradebookdisplay -N mygradebook -K BFE9F8FA2E03C438F3C973D1C05394E46F5004479228
909F71A99BF82F1C34D71234 -AA -AN Project4 -P 100 -W 0.01
invalid
```



```
[11/27/20]seed@VM:~/.../Project4$ gradebookdisplay -N mygradebook -K BFE9F8FA2E03C438F3C973D1C05394E46F5004479228
909F71A99BF82F1C34D7 -AA -AN ThisInputIsWayTooLongToBeAcceptedByTheGradebookPrograms -P 100 -W 0.01
invalid
```



```
[11/27/20]seed@VM:~/.../Project4$ gradebookdisplay -N mygradebook -K BFE9F8FA2E03C438F3C973D1C05394E46F5004479228
909F71A99BF82F1C34D7 -AS -FN ThisInputIsWayTooLongToBeAcceptedByTheGradebookPrograms -LN Ford
invalid
```



```
[11/27/20]seed@VM:~/.../Project4$ gradebookdisplay -N mygradebook -K BFE9F8FA2E03C438F3C973D1C05394E46F5004479228
909F71A99BF82F1C34D7 -AS -FN Maddie -LN ThisInputIsWayTooLongToBeAcceptedByTheGradebookPrograms
invalid
```

The code for checking the lengths of all the inputs is:

- -N <filename>: in setup.c (line 178), in gradebookadd.c (line 231)
- -K <key>: in gradebookadd.c (lines 239-242), in gradebookdisplay.c (lines 234-237)
- -AN <assignment name>: in gradebookadd.c (lines 37-46), in gradebookdisplay.c (lines 61-70)
- -FN <first name>: in gradebookadd.c (lines 49-58), in gradebookdisplay.c (lines 73-81)
- -LN <last name>: in gradebookadd.c (lines 49-58), in gradebookdisplay.c (lines 73-81)

Attack 4: Invalid Input

In this attack, the attacker attempts to input invalid data to get an unexpected output. Each of the options require different data types as specified below:

- -N
  - may be specified with a string of alphanumeric characters (including underscores and periods)
- -AN
  - alphabetic characters (a-z, A-Z) in upper and lower case and numbers (0-9)
  - may not contain spaces
  - case sensitive
- -FN
  - alphabetic characters (a-z, A-Z) in upper and lower case
  - may not contain spaces

    - case sensitive
- -LN
    - alphabetic characters (a-z, A-Z) in upper and lower case
    - may not contain spaces
    - case sensitive
- -P
    - required to be a non-negative integer
- -W
    - required to be a real number in [0,1]
- -G
    - required to be a non-negative integer

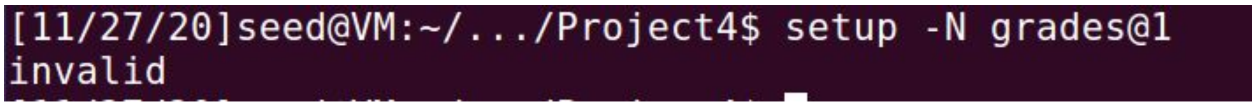
In this attack, the attacker wants some unexpected output, so they will input data that does not meet the requirements above. An attack like this could result in a segmentation fault or some other corruption of the gradebook.

However, this attack will be unsuccessful on this gradebook system, because I implemented whitelisting. Whitelisting defines a set of valid characters. Each of the options above have a corresponding function that guarantees the input only contains valid characters. The lines of these functions are:

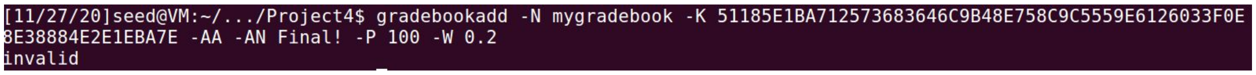
- -N: in setup.c (lines 23-32), in gradebookadd.c (lines 25-34), in gradebookdisplay.c (49-58)
- -AN: in gradebookadd.c (lines 37-46), in gradebookdisplay.c (lines 61-70)
- -FN: in gradebookadd.c (lines 49-58), in gradebookdisplay.c (lines 73-82)
- -LN: in gradebookadd.c (lines 49-58), in gradebookdisplay.c (lines 73-82)
- -P: in gradebookadd.c (lines 61-67)
- -W: in gradebookadd.c (lines70-82)
- -G: in gradebookadd.c (lines 61-67)

Each of the above functions are called before performing any action or storing the input to any variables in the program. This can be seen in lines 365-565 in gradebookadd.c and lines 336-486 in gradebookdisplay.c. If the program reads any input that does not meet the requirements above, the program exits and returns "invalid".

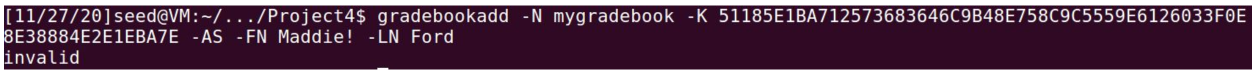
An unsuccessful attempt of this attack can be seen in the following screenshots:



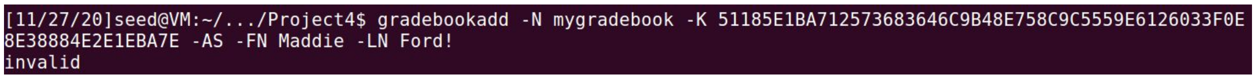
```
[11/27/20]seed@VM:~/.../Project4$ setup -N grades@1
invalid
```



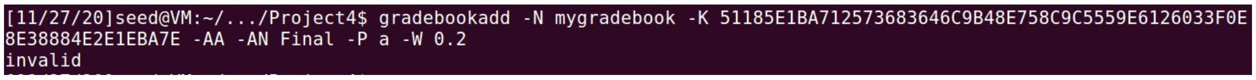
```
[11/27/20]seed@VM:~/.../Project4$ gradebookadd -N mygradebook -K 51185E1BA712573683646C9B48E758C9C5559E6126033F0E
8E38884E2E1EBA7E -AA -AN Final! -P 100 -W 0.2
invalid
```



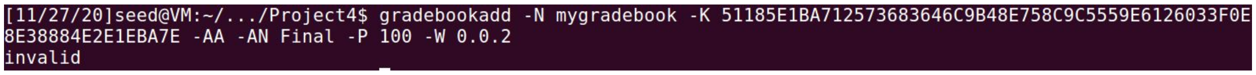
```
[11/27/20]seed@VM:~/.../Project4$ gradebookadd -N mygradebook -K 51185E1BA712573683646C9B48E758C9C5559E6126033F0E
8E38884E2E1EBA7E -AS -FN Maddie! -LN Ford
invalid
```



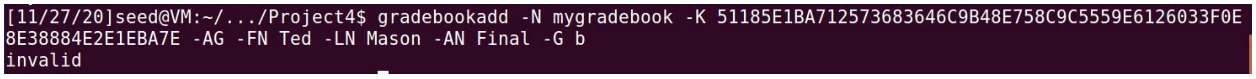
```
[11/27/20]seed@VM:~/.../Project4$ gradebookadd -N mygradebook -K 51185E1BA712573683646C9B48E758C9C5559E6126033F0E
8E38884E2E1EBA7E -AS -FN Maddie -LN Ford!
invalid
```



```
[11/27/20]seed@VM:~/.../Project4$ gradebookadd -N mygradebook -K 51185E1BA712573683646C9B48E758C9C5559E6126033F0E
8E38884E2E1EBA7E -AA -AN Final -P a -W 0.2
invalid
```



```
[11/27/20]seed@VM:~/.../Project4$ gradebookadd -N mygradebook -K 51185E1BA712573683646C9B48E758C9C5559E6126033F0E
8E38884E2E1EBA7E -AA -AN Final -P 100 -W 0.0.2
invalid
```



```
[11/27/20]seed@VM:~/.../Project4$ gradebookadd -N mygradebook -K 51185E1BA712573683646C9B48E758C9C5559E6126033F0E
8E38884E2E1EBA7E -AG -FN Ted -LN Mason -AN Final -G b
invalid
```