Overall structure of Gradebook:

```c
typedef struct _Assignment {
  char* name;
  float weight;
  int points;
} Assignment;

typedef struct _Student {
  char* fname;
  char* lname;
  size_t total_assignments;
  char** assignment_names_lst;
  int* points_earned;
} Student;

typedef struct _Gradebook {

  Assignment* assignments;
  Student* students;
  size_t num_assignments;
  size_t num_students;
} Gradebook;
```

Gradebook

The gradebook structure consists of a 2 dynamically allocated arrays: assignments and students. In addition, I keep track of the number of elements in both arrays.

Student

The student struct consists of 2 strings that represents the student's name, the number of assignments, a dynamically allocated string array that holds the assignments name, a dynamically allocated int array that holds the student's grade.

Assignment

The assignment struct has string that holds the Assignment's name, a float datatype that represents the weight of the assignment, and the number of points that assignment is out of.

Setting up New Gradebook

After validating the arguments for the setup instruction, I generate a new key using the following code:

```
char hex [9];

RAND_bytes((unsigned char*)&seed, sizeof(seed));
sprintf(hex, "%x", seed);
for(int i = 0; i<8; i++) {
  key[i] = hex[i];
}

RAND_bytes((unsigned char*)&seed, sizeof(seed));
sprintf(hex, "%x", seed);
for(int i = 8; i<16; i++) {
  key[i] = hex[i-8];
}
key[16] = '\0';
```

I use a special function from the openssl/rand.h library called RAND_bytes that generates “cryptographically strong pseudo-random bytes” to create a 128 bit key. I call RAND_bytes twice in order to prevent integer overflow.

After creating a key, I encrypt the key and put it inside of a file called “key-file<gradebookname>” so that gradebookadd and gradebookdisplay can verify if the user is using the correct key.

Reading/Write Gradebook File

When Gradebookadd and gradebookdisplay are called, it validates arguments that were provided and then decrypts and reads the gradebook file. The encryption scheme that is being used is **AES_CFB128 mode** which requires 128-bit key. After decrypting the file, the file will look like this:

```
Midterm 100 0.250000 Final 200 0.500000
John Smith 95 180
Russell Tyler 80 190
Ted Mason 90 150
```

The first line contains the assignment names, weights, and points, and the subsequent lines contains the students and the corresponding grades to their assignment. Once the file is decrypted, the program will use file IO to parse the file and create a Gradebook struct.

When the gradebookadd and gradebookdisplay finish their respective action, it writes the contents of the gradebook struct to the file and encrypts the file with provided key.

Overview of Gradebookadd Operations

Adding Student

When I need to add a new student, I create a Student struct variable, reallocate the students pointer, append the students pointer, and increment num_students.

Deleting student

When deleting a student, I determine the position of the student in the students array. Next, I shift all the elements of the array down by one, and then reallocate the array.

Adding Assignment

When I need to add a new assignment, I create an Assignment struct variable, reallocate the assignments pointer, append the assignments pointer, and increment num_assignments. Next, I traverse the students array and update the assignment_names_lst and points_earned array by appending the assignment's name to the assignment_names_lst array and appending 0 into the points_earned array.

Deleting Assignment

When I need to delete an assignment, I determine the index of the assignment by searching for the assignment name in the assignments array in Gradebook. Next, I shift all the elements of the array down by one in order to overwrite the target assignment, and then reallocate the array. In addition, I traverse the students array and for each student I overwrite the corresponding entries for the target assignment in the assignment_names_lst array and points_earned array.

Adding Grade

When I need to add a grade for a particular student, I search for the student in Gradebook's students array. Next, I determine the index of the assignment, and insert the grade into the student's points_earned array.

Overview of Gradebookdisplay Operations

Printing Assignment

This operation prints out all the grades for the students for a particular assignment. I implemented this operation by determining the location of the assignment in the Gradebook's assignments array and then traverse the students array and print the grade that the student got for the assignment. The default grade of an assignment is 0.

Printing Student

This operation prints out all the grades for a particular student. I implemented this by determining the location of the student in Gradebook's students array and traversing the Student's points_earned and assignment_names_lst array and printing the grade that the student got for each assignment.

Print final

This operation prints the final grade for each student. I implemented this by traversing the Gradebook's students array and calculating the final grade for each student.

4 Potential Attacks

Buffer Overflow attack with strcpy:

In buffer overflow attacks, the attacker sometimes uses strcpy to inject code into the stack, To prevent these attacks, I decided not to use strcpy. Instead I used strncpy to make sure that program is copying the correct number of bytes.

Examples:

gradebookadd_skel.c

```
236   (*G).assignments[(*G).num_assignments-1].name = malloc((strlen(R.AN)+1)*sizeof(char));
237   strncpy((*G).assignments[(*G).num_assignments-1].name, R.AN, (strlen(R.AN)+1)*sizeof(char));
238
239
240   (*G).assignments[(*G).num_assignments-1].weight = atof(R.W);
241   (*G).assignments[(*G).num_assignments-1].points = atoi(R.P);
242
243   for(int i = 0; i<((*G).num_students); i++) {
244
245     (*G).students[i].total_assignments = (*G).num_assignments;
246     (*G).students[i].assignment_names_lst = realloc((*G).students[i].assignment_names_lst, ((*G).num_assignments
247     (*G).students[i].assignment_names_lst[(*G).num_assignments-1] = malloc((strlen(R.AN)+1)*sizeof(char));
248     strncpy((*G).students[i].assignment_names_lst[(*G).num_assignments-1], R.AN, (strlen(R.AN)+1)*sizeof(char));
249
250     (*G).students[i].points_earned = realloc((*G).students[i].points_earned, ((*G).num_assignments)*sizeof(int))
251     (*G).students[i].points_earned[(*G).num_assignments-1] = 0;
252   }
```

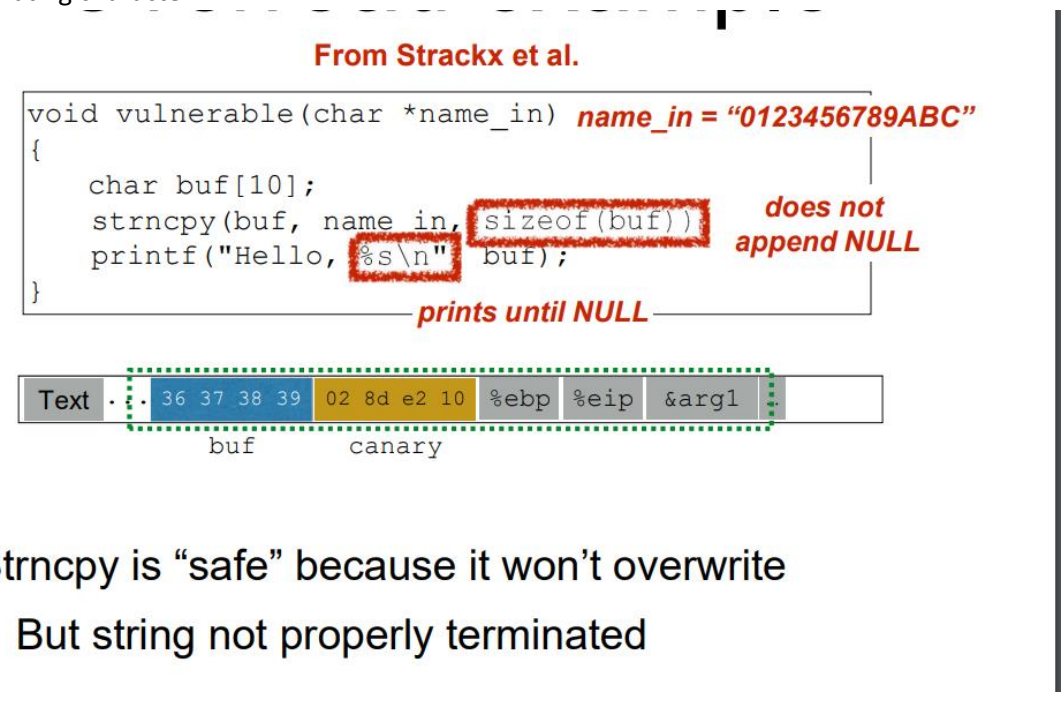
Data_skel.c: Reading input from gradebook file:

```
316   while((read = getline(&line, &len, fp)) != -1){
317    char* student_fname = strtok(line, " ");
318    G.students = realloc(G.students, sizeof(Student)*(++num_students));
319    G.students[num_students-1].fname = malloc((strlen(student_fname)+1)*sizeof(char));
320    strncpy(G.students[num_students-1].fname, student_fname, (strlen(student_fname)+1)*sizeof(char));
321
322    char* student_lname = strtok(NULL, " ");
323
324    G.students[num_students-1].lname = malloc((strlen(student_lname)+1)*sizeof(char));
325    strncpy(G.students[num_students-1].lname, student_lname, (strlen(student_lname)+1)*sizeof(char));
326
327    i = 0;
328    char * student_score = strtok(NULL, " ");
329    G.students[num_students-1].points_earned = NULL;
330    while(student_score && strcmp(student_score, "\n")!=0){
331
```

Gradebookdisplay_skel.c

```
213   }
214   StudentPoints *tmp_arr = malloc(sizeof(StudentPoints)*((*G).num_students));
215
216   for(int i = 0; i<(*G).num_students; i++){
217     tmp_arr[i].fname = malloc((strlen(((*G).students[i].fname))+1)*sizeof(char));
218     strncpy(tmp_arr[i].fname , (*G).students[i].fname, (strlen(((*G).students[i].fname))+1)*sizeof(char));
219     tmp_arr[i].lname = malloc((strlen(((*G).students[i].lname))+1)*sizeof(char));
220     strncpy(tmp_arr[i].lname , (*G).students[i].lname, (strlen(((*G).students[i].lname))+1)*sizeof(char));
221     tmp_arr[i].points = (*G).students[i].points_earned[loc];
222   }
223   if(strcmp(order, "-A")==0){
```

While this method will prevent overwriting, the strings that were copied into might not have a null-terminating character:



- Strncpy is “safe” because it won’t overwrite
  - But string not properly terminated

Deleting Key file

As mentioned in "**Setting up a New Gradebook"**, **setup** creates a key-file that will be used by gradebookadd and gradebookdisplay to verify if the correct key is being inputted by the user. If the attacker deletes the key-file after running setup, gradebookadd and gradebookdisplay will have no way of knowing if the correct key was used. To counter this attack, I check if the keyfile exists in the directory. If it doesn't exist, I print invalid:

Gradebookadd_skel.c:

```
76        keyfileName[i+8] = gname[i];
77      }
78      keyfileName[strlen(gname)+8] = '\0';
79    FILE * f = fopen (keyfileName, "rb");
80
81    if (f)
82    {
83      fseek (f, 0, SEEK_END);
84      length = ftell (f);
85      fseek (f, 0, SEEK_SET);
86      buffer = malloc (length);
87      if (buffer)
88      {
89        fread (buffer, 1, length, f);
90      }
91      fclose (f);
92    }
93    else{
94
95      return 0;
96    }
97    // printf("%s\n", buffer);
```

Gradebookdisplay_skel.c:

```
379
380   char keyfileName[strlen(gname)+8+1];
381
382     strncpy(keyfileName, "key-file", strlen(gna
383
384     for(int i = 0; i<strlen(gname); i++){
385       keyfileName[i+8] = gname[i];
386     }
387     keyfileName[strlen(gname)+8] = '\0';
388   FILE * f = fopen (keyfileName, "rb");
389
390   if (f)
391   {
392     fseek (f, 0, SEEK_END);
393     length = ftell (f);
394     fseek (f, 0, SEEK_SET);
395     buffer = malloc (length);
396     if (buffer)
397     {
398       fread (buffer, 1, length, f);
399     }
400     fclose (f);
401   }
402    else{
403
```

Using an invalid key:

To prevent the use of an invalid key, I used encrypted the key and put it inside of file called “key-file<gradebook_name>”, when setup is called for the first time. When gradebookadd or gradebookdisplay is called, it first decrypts the key-file using the key that was inputted by the user. After decryption, if the key matches the decrypted contents of the key-file, program execution continues. If it doesn’t match then the program exits.

Gradebookdisplay_skel.c:

```
}
int check_key(char * key, char *gname){

  unsigned char *iv = (unsigned char *)"0000000000000000";

  unsigned char ciphertext[128];

    /* Buffer for the decrypted text */
  unsigned char decryptedtext[128];

  int decryptedtext_len, ciphertext_len;

  char * buffer = 0;
  long length;

  char keyfileName[strlen(gname)+8+1];

    strncpy(keyfileName, "key-file", strlen(gname)+8+1);

    for(int i = 0; i<strlen(gname); i++){
      keyfileName[i+8] = gname[i];
    }
    keyfileName[strlen(gname)+8] = '\0';
  FILE * f = fopen (keyfileName, "rb");

  if (f)
  {
    fseek (f, 0, SEEK_END);
    length = ftell (f);
    fseek (f, 0, SEEK_SET);
    buffer = malloc (length);
    if (buffer)
    {
      fread (buffer, 1, length, f);
    }
    fclose (f);
  }
   else{

    return 0;
```

```
    // printf("%s\n", buffer);


    decryptedtext_len = decryptKey(buffer, strlen(buffer), key, iv,
      decryptedtext);
    // printf("decryptedtext: %s\n", decryptedtext);
    for(int i = 0; i<strlen(key); i++){
      if(key[i]!=decryptedtext[i]){
        return 0;
      }
    }
    // printf("decryptedtext: %s\n", decryptedtext);
    return 1;
  }
  int does_file_exist(char * fname) {
```

Gradebookadd_skel.c

```
157   }
158   int check_key(char * key, char* gname){
159
160     unsigned char *iv = (unsigned char *)"0000000000000000";
161
162     unsigned char ciphertext[128];
163
164       /* Buffer for the decrypted text */
165     unsigned char decryptedtext[128];
166
167     int decryptedtext_len, ciphertext_len;
168
169     char * buffer = 0;
170     long length;
171     char keyfileName[strlen(gname)+8+1];
172
173       strncpy(keyfileName, "key-file", sizeof(keyfileName));
174
175       for(int i = 0; i<strlen(gname); i++){
176         keyfileName[i+8] = gname[i];
177       }
178       keyfileName[strlen(gname)+8] = '\0';
179     FILE * f = fopen (keyfileName, "rb");
180
181     if (f)
182     {
183       fseek (f, 0, SEEK_END);
184       length = ftell (f);
185       fseek (f, 0, SEEK_SET);
186       buffer = malloc (length);
187       if (buffer)
188       {
189         fread (buffer, 1, length, f);
190       }
191       fclose (f);
192     }
193     else{
194
195       return 0;
196     }
```

```
498
499
500     decryptedtext_len = decryptKey(buffer, strlen(buffer), key, iv,
501       decryptedtext);
502     // printf("decryptedtext: %s\n", decryptedtext);
503     for(int i = 0; i<strlen(key); i++){
504       if(key[i]!=decryptedtext[i]){
505         return 0;
506       }
507     }
508     // printf("decryptedtext: %s\n", decryptedtext);
509     return 1;
510   }
511
```

Modifying gradebookfile

Another potential attack is if the attacker modifies the gradebook file and gradebookadd and gradebookdisplay accept it. I didn't how to counter this attack in C. I was thinking of trying to implement something similar to Microsoft File Checksum Integrity, but I didn't how to do it. By using checksum gradebookadd and gradebookdisplay will be able to know if the file was modified and verify its integrity.