CMSC414 HW9 Documentation

Team: BanKings

Nathaniel Kent, Noah Grinspoon, Ari Bailey, Ian Serbin

Our Approach

1. We decided to use Java over C
   1. because it is a memory safe language.
      1. We do not have to worry about memory allocation
      2. We do not have to worry about buffer exploits
      3. We do not have to worry about garbage collection
   2. It’s easier for OOP
   3. It’s more familiar to the group members
   4. It has good exception handling which is an important part of our program robustness in design
2. Our data structures:
   1. Please see our UML diagram: <https://ter.ps/bankingsspec> (also included in repo)
   2. Gradebook : The main class, this is the class that all others interact with
      1. Each Gradebook holds a private list of all the students and a private list of all the assignments
      2. Each Gradebook holds a private gradebook, which maps students to individual hashmaps mapping assignments to grades
      3. Each Gradebook has a three private Strings which represent calculated hashes of the student list, assignment list, and gradebook map
      4. There is a private method to calculate the hash of an object passed in
      5. There is a private method to return the gradebook map as a String
   3. Student : This class produces student objects that are identified by matching first and last names.
   4. Assignment: This class produces assignment objects that each hold their own unique name, as well as holding their weight and number of total points
   5. Setup: Creates an encrypted gradebook object and returns its respective key
   6. GradebookAdd: This class allows us to add various things to an existing and valid gradebook, including assignment objects, student objects, grades for a certain student’s assignment. In this class, assignment and student objects can also be completely deleted from the specified gradebook. The three hashes in Gradebook are checked if decryption is successful, program exits if stored hashes do not match newly calculated hashes.
   7. GradebookDisplay: This class has functions that allow us to display any valid gradebook object based on a valid given command, such as printing all students grades for one assignment, all students final grades, or all grades of an individual student.The three hashes in Gradebook are checked if decryption is successful, program exits if stored hashes do not match newly calculated hashes.

**Our Defenses**

**Vulnerability #1**: The program requires us to parse the command options as well as read in the gradebook object. In C there is the potential for buffer overflows or overreads depending on the scan functions used. A buffer overflow could inject arbitrary code into our program and an overread could potentially leak sensitive info. For example, this could potentially allow an attacker to bypass our hash-checking when reading in the gradebook.

**Defense #1:** We decided to use Java instead of C because it is memory safe and we did not have to be concerned with improper memory management or potential overflows/reads.

**Vulnerability #2:** We store our gradebook in a file.If it was stored in plaintext, an attacker could simply open the file to see all the gradebook data or make changes. If we tried to use our own encryption scheme, it could potentially be broken, allowing access to the file contents. If an attacker figured out the layout of the gradebook, they would try and flip bits to modify the encrypted gradebook contents.

**Defense #2:** We use AES encryption with a 128-bit cipherkey. We convert the entire gradebook object to a sealed object, which we encrypt with this key before writing it to file. This therefore ensures that only someone who has the cipherkey can make changes or read data in the gradebook file. Our gradebook object also uses a hashmap, which means its order and the hashcode are not guaranteed to stay the same each time it is opened. This makes it very difficult for the attacker to figure out where to modify bytes, because it will change each time.

**Vulnerability #3:** An attacker can use trial and error with the encrypted bytes in our file in order to make changes to our stored data. Using this -- with a lot of time and patience -- an unauthorized party could make changes to student grades.

**Defense #3:** Aside from the issues mentioned in Defense#2 , we also implement hashing Strings within our gradebook object. These strings are calculated any time we make changes to our gradebook and want to write it to file. The strings represent hashes of our students ArrayList, assignments ArrayList and gradebook HashMap. If any of these objects are changed between authorized uses of gradebookadd/display, their calculated hashes will change. So, if an attacker is able to change one of these files, the program will return invald when a legitimate program call is made because the hashes stored in the gradebook object would not match the hashes calculated when the gradebook is opened.

**Vulnerability #4:** With a short key, an attacker can brute force it. For example with the key in the example it is eight characters. This is a 32 bit key. With 16 available characters this is 16^16 possible combinations, which is only 4294967296 possible combinations. This can be brute forced, especially considering there are no limitations on the running of this program, in minimal time. Even on a light laptop this could be done in around half a day.

**Defense #4:** We implemented AES-128 bit encryption which has a 32 character hexadecimal key. This is 16^32 combinations which is a lot. Even with a supercomputer, it would take a billion billion years to brute force this.

**Vulnerability #5:** Our program uses deserialization to read in the gradebook file and cast it to a Gradebook object. Java’s deserialization is rather vulnerable in that if a program allows the user to provide the data to be deserialized, an attacker could craft a chain of gadgets which could lead to arbitrary code execution. HashMaps are vulnerable to deserialization attacks because they call hashCode() for each entry, which can be used to create a gadget chain with the proper payload.

**Defense #5:** In our program, readObject(), the vulnerable deserialization method, is not called directly on the inputstream of the gradebook file, but on the decrypted cipherstream. This means that if the decryption does not succeed, the data will not make it to readObject(). Therefore, unless the attacker encrypts and seralizes the malicious object with the user’s key, there is no way for it to make it to the vulnerable deserialization call.

Team Member Contributions

**Noah**:

* Worked on the logic in multiple methods in gradebookadd, gradebookdisplay, Gradebook
* Determined the structures we would use to store students, assignments, and the gradebook logic.
* Debugged the commands to see what went wrong in terms of syntax. Worked with Ian to debug the hashing of the Gradebook variables.
* Ran a large test to make sure everything worked as expected.

**Ari**:

* Created Assignment, Student and Gradebook classes and populated with original methods as well as additional getters and setters as needed. (i.e. program structure).
* Built parsing structure used for parse\_cmd in gradebookadd and subsequently for the same method in gradebookdisplay.
* Handled almost every parsing case in gradebookadd.
* Created parse\_remainder method and allowed\_chars used for above.
* Worked with Nate to create documentation. Created UML diagram of program and wrote vulnerabilities section.
* Set up a coding environment to allow us to easily collaborate. Also managed most git commits.

**Nate**:

* Helped create the structure of the gradebook, such as the list of students, the list of assignments, and the map holding the students with all their assignments/grades.
* Worked mostly on getting a gradebook display to correctly output the required format.
* Debugged and tested gradebook display, finding edge cases along the way to fix.
* Created a comparator to organize the names in ascending alphabetical order.
* Added getters to every class where they were needed, but made sure to keep the program safe from any attacks.
* Worked with Ari to create the UML diagram to represent the breakdown of our gradebook code structure.

**Ian**:

* Created setup
* Designed and wrote the methods for encrypting the Gradebook object and writing to file
* Designed and wrote the methods for reading the gradebook from file, decrypting it, and recreating the Gradebook object
* Designed and wrote the methods for calculating the hashes of student list, assignment list, and gradebook map
* Wrote the method to convert the grabook map to a string, which is used when calculating the hash of it. Hashing the object itself would not work because HashMaps do not maintain order or hashcodes across program runs.
* Designed and wrote the methods for checking that the hashes of student list, assignment list, and gradebook map are the same as the last time the gradebook was modified by gradebookadd.