Assignment 5: Fortify

Part A: Defense phase 100 Marks

Each team would receive a cipher implementation with some sensitive data. The goal for each team in the defense phase is to obfuscate the given sensitive information within the cipher such that it is hard for attackers to identify it.

Submission Guidelines:

1. The teams should submit an archive named **`rollno1_rollno2.zip`**
2. The archive should contain the following:
    a. Modified **`main.c`** file
    b. An obfuscated binary file named **`safe_main`** that was compiled by you should be a 64-bit ELF file x86 architecture.
    c. A **Dockerfile** that builds your binary
    d. A detailed **Report** describing your obfuscation logic, the changes you have made, the execution time of the obfuscated binary, and a clear explanation of your **Dockerfile**. The report should also clearly mention the resources and libraries you used to build your binary, including the logic in your Dockerfile. **(10 Marks)**
    e. We will generate the obfuscated binary using the steps you mentioned in your report, using your **Dockerfile,** and ensure that it matches your submitted binary.
    f. Make sure your **Dockerfile** generates your binary in a folder that is accessible by the host. Ex: docker run -v /path/on/host:/path/in/container <image_name>
3. We will test your **`safe_main`** binary with a hidden set of plaintext-ciphertext pairs to get egg[0] and the global flag. These are used to check for the functional correctness of the obfuscated binary. **(90 Marks)**

Rules for obfuscation:

1. **Prohibited Tools:** teams are not allowed to use off-the-shelf packers (UPX, Cobalt Strike etc), obfuscators (MOVfuscator), or tools for code obfuscation. The assignment aims to learn how to secure your code.
2. **Custom Implementation:** All code used for obfuscation must be written entirely by the team.
3. **Build Files:** If teams wish to apply obfuscation techniques on top of their generated binary, they need to add this logic in a single **Dockerfile** to build their binary from their C code and perform obfuscation. This file should be submitted along with their main.c.
4. **Static Compilation:** The final binary must be statically compiled and capable of running on all systems without modification.

5. **Self-Sufficiency:** The binary should not rely on external environmental dependencies; it must be self-contained.
6. **No Selective Execution:** The binary should not include selective execution mechanisms, such as running only on specific dates, requiring a password, or any such conditions.
7. **File and Network Restrictions:** The obfuscation logic must not access files (read/write), networks, or sockets during execution.
8. **Execution Time Limit:** The binary must complete its execution within 2 seconds.
9. **Size Constraint:** The size of the final binary must not exceed 5 MiB.
10. **Output Consistency:** The obfuscated program's CipherText, Eggs, and Global Flag values must match the provided test results.
11. **Main Function Integrity:** Teams are strictly prohibited from modifying the values of Eggs, Global Flag, and Key. They should also not modify the **`main`** function. The program must produce the same output and print format as the unmodified code.

Assignment Progression:

1. At the start of the Defence Phase, we will release unique files **(`main.c`)** and **`test_results.txt`**) to each team.
2. The task for the teams is to obfuscate the sensitive content provided in the **`main.c`** file in such a way that anyone running their generated binary should not be able to identify the sensitive data **(Eggs, Global Flag,** and **Key)**
3. In Phase 2 of the assignment, the generated binaries will be shared with other teams that will try to find this sensitive data.
4. A Team will lose points if other teams can leak their unique sensitive data. The better your obfuscation, the lower the chances of your sensitive data being leaked.

Grading:

1. If your submission complies with the submission guidelines, i.e., all the rules mentioned above, then we will test your obfuscated code with some hidden plaintext-ciphertext pairs to confirm the correct implementation of the cipher, egg[0] and the global flag. If the obfuscated binary matches the output of the original provided file, then your team will be awarded **(90+10) Marks** for this assignment.
2. Alternatively, If your submission does not comply with the submission rules or the obfuscated binary's output doesn't match the original file, it will not be considered for the second phase of this assignment, and you will receive **0** for this assignment, and your team will also not be eligible for the next phase of the assignment.
3. In the Attack Phase, if another team can break your obfuscation to identify any of your sensitive data, i.e., Egg_Params, Global Flag, and Key, you would lose marks for each of such breaks **0.5*4, 4*1, 4*1,** respectively.

4. The lower limit for Phase-1 marks would be 10 Marks. Once enough teams have broken a submitted binary to reduce their score to 10, no further attacks to break the binary would further reduce their score.

Files Provided:

1. `main.c` - This is the code you need to obfuscate and protect.
   This file has three significant data components that need to be protected from being leaked to attackers:

The eggs:

```
uint8_t egg_params[5][6] = {{0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0},
                            {0, 0, 0, 0, 0, 0};
```

The Global Flag:
This is a function of some of the eggs provided to you scaled using some arbitrary constants.

```
uint8_t compute_gf(uint8_t* eggs){
      return A*eggs[X] $ B*eggs[Y] @ C*eggs[Z];
// where $ and @ can be (+,-,*) and 1<= X, Y, Z <= 4
}
```

The Key:

```
  uint8_t key[AES_BLOCKLEN] = {0x00, 0x00, 0x00, 0x00,
                               0x00, 0x00, 0x00, 0x00,
                               0x00, 0x00, 0x00, 0x00,
                               0x00, 0x00, 0x00, 0x00};
```

2. `test_results.txt` - The results obtained after running the executable of the unmodified main.c file against a set of fixed plaintexts, the execution time, and the size of the generated binary. Use this file to check the functional correctness of your obfuscated binary.

```
---------------- File Test Results -------------------------
```

```
Plaintext: abcdefghijklmnopq
Executable CipherText: 3e087434ae56d1bb844d804a3cd80cea
Python CipherText: 3e087434ae56d1bb844d804a3cd80cea
Egg 0: 0x01
Global Flag: 0x06
Time of execution: 0.000447 seconds
Match Result: Matched
...
...

Plaintext: ghijklmnopqrstuv
Executable CipherText: f4c78ef450c1bacaf9872af0ea290983
Python CipherText: f4c78ef450c1bacaf9872af0ea290983
Egg 0: 0x00
Global Flag: 0xea
Time of execution: 0.000418 seconds
Match Result: Matched

Size of executable: 909368 bytes
```

Attack Phase: Part (b)

Once the teams have submitted the binaries we would first ensure that the submission is functionally correct and is created using the provided guidelines. All the teams would then be randomly given access to some binaries to break in the attack Phase of the assignment. The binaries would be renamed and anonymized before attackers get access to them.

The attackers need to submit the **equivalent main.c** file for the obfuscated binaries that they are attacking. The equivalent file would then be tested against the hidden test cases to ensure that the key, eggs, and global flags were broken. The attackers can also get a partial score for identifying individual components of the obfuscated binary.

Submission guidelines:

The attacker needs to submit a zip folder named **`rollno1_rollno2.zip`** that contains the following:

1. The **equivalent `main.c`** for the target binaries **AND** the equivalent **`metadata.txt`** for the target binaries.
2. The **target binaries**.
3. A **Report** on how you broke the target binaries. (10 Marks)

Scoring for Attack Phase:

Each attack that happens on a specific binary would bring down its base score until it reaches **10 points (2,4, and 4 for eggs, global flag and the key)**. We divide the base points of the binary as **20, 40, 40** for the **Eggs, Global Flag, and Key** respectively**.** Since some sections of the binary could be more secure than others, the points awarded for breaking them are also computed differently. The base score would also reduce separately for each of these sections and would stop when that section reaches its minimum value.

Teams that have entirely removed the eggs not used in the global flag from their binary, would have the 20 marks for eggs equally distributed amongst all the present eggs. For example, if a team has just the implementation of just 2 eggs in their binary each of them would be worth 10 marks each instead of the original 5 marks.

n = **number of teams that broke that section of the binary**
b = **base score of that section of the binary**

b/(n) points would be awarded to each team that breaks this specific section of the binary. The first team to report a binary with a valid rule break that gives **the defenders an unfair advantage** would also fetch them **x*base score** of the binary(sum of all the base score of the sections). (x is yet to be finalized depending on the actual number of such incidents).

The reported binaries with the rule violation would be penalized further, depending on the level of unfair advantage it gets against other teams. This penalty would be different from the **20% penalty applied on all the late submissions before 11 AM, and 40% for teams that only have a valid submission after that.** (your latest submission is considered for grading if it's before the deadline, after the deadline your 1st submission is considered for grading and for the penalty).

The attacker also gets an additional 10 Marks for the report that explains all the attacks performed on the corresponding binaries. The report should contain **attacks on at least 1 binary** to be eligible for this.
Note: We decided not to penalize teams for not breaking any binary till the last two days of their attack.
Improper/Insufficient explanation would reduce the marks awarded for the report submitted.

This assignment itself would have relative grading since the max score for this assignment is actually uncapped, we'll use that to scale all the marks obtained by the teams to scale to 150 marks. This scaling would be done after the application of all the penalties and points awarded for reporting binaries.