Team Name: Wadiya16@ (Tanush Govind, Danyl Kecha, Mark Cockerill, Lili Tsang, Greg Smith)

To trigger the backdoor, the user must enter a password following a specific pattern: a 9-character password containing only lowercase letters from ‘a’ to ‘k’; 1st, 3rd, and 7th characters must result in 'g' when XORed. If this password is processed through a custom hash function (*inspired by the djb2 algorithm and developed with Gemini Pro*), it generates a particular hash. The system authenticates the user if this hash matches a predefined value.

The vulnerability is hidden within a chain of functions, each adding constraints on the password. Initially, the password passes through a default SHA-256 encryption algorithm, but with an additional condition for a 9-character long password. This condition triggers a secondary function, which imposes further restrictions.

```
return value.length() == hashTargetKey.length() ? hash_combination(value, ss.str()) : ss.str();
```

The hash_combination() method enforces the condition that the 1st, 3rd, and 7th characters of the password must result in 103 when XORed.

```
std::string hash_combination(const std::string &value, const std::string &prevHashValue = "GOOD_LUCK") { return (value[0x0] ^ value[6] ^ value[0x2] ^ 103) == 0 ? hash_verification(value, prevHashValue) : prevHashValue; }
```

If the previous condition is met, the hash_verification() method forces the password to contain only lowercase letters from 'a' to 'k'. This is done in a convoluted manner by using the ASCII decimal values of the characters and defining the allowable range.

```
for (char ch : value) { if ((ch & 0xFF) != '/' && (ch & 0xFF) != '#' && (ch & 0xFF) != '1') { if (int(ch) * 97 >= 9409 && int(ch) * 107 <= 11449) count++; }}
```

Finally, if all conditions are met, the password is passed through a custom hash function. The resulting hash is compared to the target hash, which is obfuscated using a basic XOR encryption inside the get_hash() method.

The hash function is intentionally designed to be prone to brute-force attacks. However, if attackers attempt to brute-force a 9-character password using all possible lowercase and uppercase letters and numbers, they'd face 62^9 combinations. However, for the purposes of the assignment, if the attacker thoroughly analyzes the code, they can understand how to significantly reduce the number of password combinations.

The backdoor is extremely hard to detect. The backdoor trigger is embedded within the SHA-256 function. The code uses a misleading default variable, which serves no other purpose than to compare the password length, ensuring that it’s exactly 9 characters long. The next imposed restrictions are reversed: instead of initially checking whether the characters fall within the range 'a' to 'k', we first verify if the XOR of specific characters results in 103. If an attacker fails to realize that the order of these conditions doesn’t matter, they would have to brute-force far more combinations. In contrast, by focusing solely on lowercase characters 'a' to 'k' that satisfy the XOR condition, there are only 91 possible combinations to test.

Additionally, the condition for verifying lowercase characters is intentionally confusing. If the attacker does not identify how the range is defined using ASCII values, they might try far more characters than necessary. Finally, the target hash is XOR-encrypted to avoid leaving it as plaintext, adding another layer of difficulty for the attacker to detect and reverse-engineer the backdoor.

Moreover, before accessing the source code, the attacker must first decrypt it, as it is encrypted using AES-256-CBC with OpenSSL. The decryption and compilation occur at runtime via the Makefile. To decrypt the code, the attacker would need to examine the Makefile and identify the specific decryption command to run: `openssl enc -d -aes-256-cbc -in $< -out $@ -pass pass:SECRET_PASSWORD`

If the attacker identifies all the constraints on the password, they would only need to brute-force **11^6 * 91** password combinations. We wrote a Python script that discovered the password in 4 minutes, making the attack feasible for the purposes of this assignment. You can access all the scripts here.