Mystery Program

What does the following program attempt to do?

```java
class ServerSecret {
  private static final String SECRET = "SuperSecret987";

  public static boolean sameAsSecretForFirstIChar(String a, int i) {
    if (a.length() != i) {
      return false;
    }
    for (int j = 0; j < a.length(); j++) {
      if (a.charAt(j) != SECRET.charAt(j)) {
        return false;
      }
    }
    return true;
  }

  public static int secretLength() {
    return SECRET.length();
  }
}

public class MysteryProgram {

  public static void main(String[] args) {
    StringBuilder attempt = new StringBuilder();
    String charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

    for (int i = 0; i < ServerSecret.secretLength(); i++) {
      long maxTime = 0;
      char foundChar = ' ';

      for (char c : charset.toCharArray()) {
        String tempAttempt = attempt.toString() + c;
        long startTime = System.nanoTime();
        ServerSecret.sameAsSecretForFirstIChar(tempAttempt, i + 1);
        long endTime = System.nanoTime();
        long duration = endTime - startTime;

        if (duration > maxTime) {
          maxTime = duration;
          foundChar = c;
        }
      }

      attempt.append(foundChar);
    }

    System.out.println("Discovered secret: " + attempt);
  }
}

```

How to mitigate this kind of attack?

Click here to expand...

The given program attempts to perform a *timing attack* on the `ServerSecret` class, which is a class containing a secret string `SECRET`. The `MysteryProgram` tries to find this secret string by measuring the time it takes to execute the `sameAsSecretForFirstIChar` method.

`MysteryProgram` iterates through each character of a charset (consisting of uppercase and lowercase letters and digits) and appends it to the current attempt. It then measures how long it takes to execute the `sameAsSecretForFirstIChar` method with this attempt. When a character in the attempt matches the corresponding character in the secret string, the method takes longer to execute due to the additional iteration in the loop. The program exploits this timing difference to determine the correct characters in the secret string one by one.

To mitigate this kind of timing attack, you can make the execution time of the `sameAsSecretForFirstIChar` method constant, independent of whether the characters match or not. One way to achieve this is by using a constant-time comparison function.

Here's an example of how to modify the `sameAsSecretForFirstIChar` method to make it constant-time:

```java
public static boolean sameAsSecretForFirstIChar(String a, int i) {
    if (a.length() != i) {
        return false;
    }
    boolean isEqual = true;
    for (int j = 0; j < a.length(); j++) {
        if (a.charAt(j) != SECRET.charAt(j)) {
            isEqual = false;
        }
    }
    return isEqual;
}
```