USER

```
// These pesky special agents keep reverse engineering our source code and then
// breaking into our secret vaults. THIS will teach those sneaky sneaks a
// lesson.
//
// -Minion #0891
import java.util.*; import javax.crypto.Cipher; import javax.crypto.spec.SecretKeySpec;
import java.security.*; class VaultDoor8 {public static void main(String args[]) {
Scanner b = new Scanner(System.in); System.out.print("Enter vault password: ");
String c = b.next(); String f = c.substring(8,c.length()-1); VaultDoor8 a = new VaultDoor8(); if
(a.checkPassword(f)) {System.out.println("Access granted."); }
else {System.out.println("Access denied!"); } } public char[] scramble(String password) {/* Scramble a
password by transposing pairs of bits. */
char[] a = password.toCharArray(); for (int b=0; b<a.length; b++) {char c = a[b]; c = switchBits(c,1,2); c =
switchBits(c,0,3); /* c = switchBits(c,14,3); c = switchBits(c, 2, 0); */ c = switchBits(c,5,6); c = switchBits(c,4,7);
c = switchBits(c,0,1); /* d = switchBits(d, 4, 5); e = switchBits(e, 5, 6); */ c = switchBits(c,3,4); c =
switchBits(c,2,5); c = switchBits(c,6,7); a[b] = c; } return a;
} public char switchBits(char c, int p1, int p2) {/* Move the bit in position p1 to position p2, and move the bit
that was in position p2 to position p1. Precondition: p1 < p2 */ char mask1 = (char)(1 << p1);
char mask2 = (char)(1 << p2); /* char mask3 = (char)(1<<p1<<p2); mask1++; mask1--; */ char bit1 = (char)(c &
mask1); char bit2 = (char)(c & mask2); /* System.out.println("bit1 " + Integer.toBinaryString(bit1));
System.out.println("bit2 " + Integer.toBinaryString(bit2)); */ char rest = (char)(c & ~(mask1 | mask2)); char
shift = (char)(p2 - p1); char result = (char)((bit1<<shift) | (bit2>>shift) | rest); return result;
} public boolean checkPassword(String password) {char[] scrambled = scramble(password); char[] expected
= {
0xF4, 0xC0, 0x97, 0xF0, 0x77, 0x97, 0xC0, 0xE4, 0xF0, 0x77, 0xA4, 0xD0, 0xC5, 0x77, 0xF4, 0x86, 0xD0, 0xA5,
0x45, 0x96, 0x27, 0xB5, 0x77, 0xE0, 0x95, 0xF1, 0xE1, 0xE0, 0xA4, 0xC0, 0x94, 0xA4 }; return
Arrays.equals(scrambled, expected); } }
```

Can you solve this?

ASSISTANT

Identify Vulnerability:

The vulnerability in this code is not directly related to a security risk but rather a logical challenge to solve. The goal is to find the correct password that, when scrambled using the provided `scramble` method, matches the `expected` array in the `checkPassword` method.

Exploit Vulnerability:

To solve this challenge, we need to unscramble the `expected` array to find the original password. We can do this by creating a new method that applies the inverse operations of the `switchBits` method.

Here's the step-by-step approach:

1. Create an inverse `switchBits` method that swaps the bits in the reverse order.
2. Apply the inverse `switchBits` method to the `expected` array to unscramble it.
3. Convert the unscrambled array back to a string to find the original password.

Here's the inverse `switchBits` method:
```java
public char inverseSwitchBits(char c, int p1, int p2) {
    char mask1 = (char)(1 << p1);
    char mask2 = (char)(1 << p2);
    char bit1 = (char)(c & mask2);
    char bit2 = (char)(c & mask1);
    char rest = (char)(c & ~(mask1 | mask2));
    char shift = (char)(p2 - p1);
    char result = (char)((bit1>>shift) | (bit2<<shift) | rest);
    return result;
}
```
Apply the inverse `switchBits` method to the `expected` array:
```java
public String unscramblePassword(char[] expected) {
    char[] unscrambled = new char[expected.length];
    for (int i = 0; i < expected.length; i++) {
        char c = expected[i];
        c = inverseSwitchBits(c, 1, 2);
        c = inverseSwitchBits(c, 0, 3);
        c = inverseSwitchBits(c, 5, 6);
        c = inverseSwitchBits(c, 4, 7);
        c = inverseSwitchBits(c, 0, 1);
        c = inverseSwitchBits(c, 3, 4);
        c = inverseSwitchBits(c, 2, 5);
        c = inverseSwitchBits(c, 6, 7);
        unscrambled[i] = c;
    }
    return new String(unscrambled);
}
```
Fix Vulnerability:

To find the correct password, we need to prefix the unscrambled password with 8 characters and suffix it with 1 character to match the `substring(8, c.length()-1)` operation in the `main` method.

Here's the complete solution:
```java
public static void main(String[] args) {
    VaultDoor8 a = new VaultDoor8();
    String unscrambledPassword = a.unscramblePassword(a.expected);
    String password = "aaaaaaaa" + unscrambledPassword + "a";
    if (a.checkPassword(password.substring(8, password.length() - 1))) {
        System.out.println("Access granted.");
    } else {
        System.out.println("Access denied!");
    }
}
```

This solution will output "Access granted." if the correct password is provided.

USER

What would be the full code solution?

ASSISTANT

Here's the full code solution with the unscramble method and the correct password extraction.

```java
import java.util.Arrays;

public class VaultDoor8 {
    public static void main(String[] args) {
        VaultDoor8 a = new VaultDoor8();
        String unscrambledPassword = a.unscramblePassword(a.expected);
        String password = "aaaaaaaa" + unscrambledPassword + "a";
        if (a.checkPassword(password.substring(8, password.length() - 1))) {
            System.out.println("Access granted.");
        } else {
            System.out.println("Access denied!");
        }
        System.out.println("Password: " + password.substring(8, password.length() - 1));
    }

    char[] expected = {
        0xF4, 0xC0, 0x97, 0xF0, 0x77, 0x97, 0xC0, 0xE4, 0xF0, 0x77, 0xA4, 0xD0, 0xC5, 0x77, 0xF4, 0x86, 0xD0, 0xA5, 0x45, 0x96, 0x27, 0xB5, 0x77, 0xE0, 0x95, 0xF1, 0xE1, 0xE0, 0xA4, 0xC0, 0x94, 0xA4
    };

    public char[] scramble(String password) {
        char[] a = password.toCharArray();
        for (int b = 0; b < a.length; b++) {
            char c = a[b];
            c = switchBits(c, 1, 2);
            c = switchBits(c, 0, 3);
            c = switchBits(c, 5, 6);
            c = switchBits(c, 4, 7);
            c = switchBits(c, 0, 1);
            c = switchBits(c, 3, 4);
            c = switchBits(c, 2, 5);
            c = switchBits(c, 6, 7);
            a[b] = c;
        }
        return a;
    }

    public char switchBits(char c, int p1, int p2) {
        char mask1 = (char)(1 << p1);
        char mask2 = (char)(1 << p2);
        char bit1 = (char)(c & mask1);
        char bit2 = (char)(c & mask2);
        char rest = (char)(c & ~(mask1 | mask2));
        char shift = (char)(p2 - p1);
        char result = (char)((bit1<<shift) | (bit2>>shift) | rest);
        return result;
    }

    public String unscramblePassword(char[] expected) {
        char[] unscrambled = new char[expected.length];
        for (int i = 0; i < expected.length; i++) {
            char c = expected[i];
            c = inverseSwitchBits(c, 6, 7);
            c = inverseSwitchBits(c, 2, 5);
            c = inverseSwitchBits(c, 3, 4);
            c = inverseSwitchBits(c, 0, 1);
            c = inverseSwitchBits(c, 4, 7);
            c = inverseSwitchBits(c, 5, 6);
            c = inverseSwitchBits(c, 0, 3);
            c = inverseSwitchBits(c, 1, 2);
            unscrambled[i] = c;
        }
        return new String(unscrambled);
    }

    public char inverseSwitchBits(char c, int p1, int p2) {
        char mask1 = (char)(1 << p1);
        char mask2 = (char)(1 << p2);
        char bit1 = (char)(c & mask2);
        char bit2 = (char)(c & mask1);
        char rest = (char)(c & ~(mask1 | mask2));
```

```
        char shift = (char)(p2 - p1);
        char result = (char)((bit1>>shift) | (bit2<<shift) | rest);
        return result;
    }

    public boolean checkPassword(String password) {
        char[] scrambled = scramble(password);
        return Arrays.equals(scrambled, expected);
    }
}
```

This code defines a VaultDoor8 class that scrambles and unscrambles a given password according to the designed method. In the main method, it creates a new VaultDoor8 object and evaluates whether the unscrambled password meets the requirements of the problem. If so, it prints a message saying "Access granted," along with the correct password.