Physical Attack on Secure Systems Assignment 4

Lucas van der Laan
s1047485

May 25, 2022

1

I already changed the real_password variable to length 17, as it interfered with my comparison code, every string should always have a terminator...

```
unsigned char real_password[17]="Test Payload!!!!";

int index = 0;
while(real_password[index] != '\0' && provided_password[index] != '\0') {
  if (real_password[index] != provided_password[index]) {
    break;
  }

  index += 1;
}

if (real_password[index] != provided_password[index]) {
  authenticate = 1;
}
```

2

I got 3 Nop's in total:

- real_password
- if(real_password[index] != provided_password[index])
- The closing curly bracket of serial_puts.

The root cause for the upper two, at least, is the fact that there can be glitches entered into the system. For example, if you can glitch the password check if statement, or it's result, you can simply make the authenticate variable not become 1 and thus become authenticated. So the root cause would be that variables can be influenced by glitches.

3

The first error I found was that authenticate could be NOP'd, so I modified the checking part of the code to basically do it twice:

```
// Doing it twice prevents the fault injection
if (real_password[index] != provided_password[index]) {
  authenticate = 1;
}

if (real_password[index] != provided_password[index]) {
  authenticate = 1;
}
```

I also noticed that when I changed the real_password to length of 17, not only did it make it a proper implementation of a string, but it also removed an overwrite for notCheck, which was pretty sweet. This overwrite was most likely because the password did not have a null terminator, so a glitch could be entered.

```
// So we fucking have a termination character
unsigned char real_password[17]="Test Payload!!!!";
```

I have not yet found a fix for serial_puts, not even an inkling of why it exists even.

4

I will go through them and fix them if I can, the first one is in the authenticate check. This actually has two changes that get applied to it, the first is a BEQ to BNE change. BEQ branches if the statement is correct and BNE if not, thus is authenticate is 1 then it suddenly doesn't branch to the fail code.

For the second issue with the authentication, we look at the operations:

- The operation [fp, #-0xC] = 0, as it is the first index at the time
- LDR r3, [fp, #-0xC] -¿ STR r3, [fp, #-0xC]
- LDR r3, [fp, #-8] -¿ LDR r3, [fp, #-0xC]

With two operation changes, we now are authenticated, no matter what the authenticate variable says, as we don't read from that anymore, but instead from the newly stored initial index.

The way to fix both of them is by simply duplicating the code again, it makes glitching it suddenly no longer that easy.

```
if(authenticate) {
  serial_puts("Auth failed!\n");
  __SET_SIM_FAILED();
}

if(authenticate) {
  serial_puts("Auth failed!\n");
```

```
8   __SET_SIM_FAILED();
9 }
```

Number three is for notCheck, as that seems to easily get abused. It is a single if statement for a single volatile bool that can be changed in a number of ways. The simulation can just replace the way notCheck gets saved and then it suddenly becomes a true instead of a false.

Number four is also notCheck, as the if statement can be manipulated. This can be manipulated to instead of BNE, we can do BQE, which reverses the way it checks. So then suddenly if the value is true, it is false, and the other way around.