# 2. Secure Coding

## 1

The program **vulnerable** is a SET_UID program that tries to write to `board1` which is a root-owned file that is inside a folder that is user-owned (meaning the user has write permission for that folder) this in turn means the user can delete/change the `board1` file. Should the user decide, she can update `board1` to be a symbolic link to `board2` (the one that the messages should not go to) and thus the execution of the vulnerable code will result in writing directly to `board2`. In fact, the user could do even more malicious things since **vulnerable** is a SETU_UID program it will run with root privileges and could even access or modify other files, add users, and more malicious actions.

This is a case of *Improper Access Control* (CWE-284).

## 2

The following script has been created:

```bash
#!/bin/bash

ln -sf ./board2 ./board1

eval $1 "attacker" "pwned message"
```

When copying this file to the `/home/user/exploit` folder the script needs to be given execution permission for `user` this can be achieved with `chmod u+x ./exploit`

## 3

The above script simply creates a symbolic link from `board1` to `board2` using regular user permissions (since the user has write access to the `exploit` folder) and then executes the passed program (`./vulnerable` that is expected to be in the same folder as `board1` and `board2`).

This will result in `vulnerable` writing the message `"pwned message"` to `board2`.

As for the username `"attacker"` and message `"pwned message"`, they were constants chosen without any specific reason except having less than 50 chars so they would not get trimmed.

## 4

Note: The patch fixes the above problem, but leaves the final program still vulnerable to race-condition attacks. This is out of scope of this task, but could be achieved using seteuid

## 5

### Vulnerability 1

#### Description

The program is susceptible to be used for denial of services on the running machine.

#### Exploit

A simple exploit involves writing a script that constantly sends the same (wrong) credentials to the program resulting in keeping it occupied and stealing system resources unnecessarily.

#### Consequences

Denial of service itsels is the consequence which, depending on how valuable computer resources are in the machine where the script is running, migh have a larger or smaller impact for the developers.

### Vulnerability 2

#### Description

Multiple login attempts can be tried in a small amount of time, this is due to the fact that the only steps between login attempts are asking for the new user/pass combination. This makes it vulnerable to a bruteforce attack.

#### Exploit

An attacker can easily design an attack that tries multiple user/password combinations. This exploit could be even more powerful by using a dictionary attack (of common passwords) and even more so if there is any knowledge about the username.

#### Consequences

The consequences are straightforward: an attacker could eventually guess and know the user/password combination that would unlock this script. Naturally, in a real scenario that user/pass combination could even be use in other settings and prove to be more damaging than at first thought.

### Vulnerability 3

#### Description

Usage of `gets` which is non advisable as it can lead to overflows, as it will try to copy the n-length user input string into a 20 characters long array.

#### Exploit

This vulnerability can be exploited by failing inserting a wrong user/pass pair for the first check and then by introducing large values when prompted for the username and password again.

#### Consequences

The consequences can be many, from program interruption to incorrect behaviour or calculation resulting from memory corruption.

## Other Vulnerabilities

- plain text username and password in compiled code (solution-> use hashing function)