UGA **INP Grenoble**
M2 CySeC & M2 CSI

Lab: Detecting and correcting software vulnerabilities

Exercise 1 : a vulnerable python library

The file ***code.py*** contains a tiny library simulating an e-commerce server able to receive customer orders (either product commands, payments, or refund requests).
Its main functionality is to check if a given sequence of orders is valid, and to print the resulting customer balance (typically wether it is null, positive, or negative).

The file ***test.py*** gives some examples of interactions with this library (creating sequences of orders and checking them).

Question 1.

The current version of this library is vulnerable in the sense that it is possible for a user, ***using only some sequence of valid orders***, to fake the computation of the resulting balance (earning money in a dishonest way :-).

The objective is therefore to change the content of file ***test.py*** to exploit these vulnerabilities, either by getting for free a non-free product, or by ending with a greater balance than the regular one. To do so, **you should not** modify the content of ***code.py***.

Hint: think about using **floating-point** values in Python, which may lead to overflow or precision errors ...

Question 2.

Update ***code.py*** in order to get rid of the vulnerabilities you found at Question 1.

Exercise 2 : a vulnerable C library

The file ***code.h*** contains a (very) lightweight C library allowing to create and manage a set of user accounts.

Each user account consists in:

- a user id (a strictly positive integer), which uniquely identifies a user;
- a boolean value telling if these user is "administrator" or not;
- a user name (possibly non unique, we don't care)
- a sequence of dummy user information called "setting", stored as pairs (index,value), where both index and values are integers.

File ***test.c*** is an example of code using this library.

* **Question 1**

The first objective is to pawn this library by successfully promoting as admin a user initially created as non-admin (i.e, *privilege escalation* vulnerability).

To do so, you should ***only*** use the API functions provided in ***code.h***, as they are, without changing them nor adding anything else in this file!
In practice you should therefore **only** change the contents of ***test.c***.

Rk: succeeding in downgrading as non-admin an admin user is a valuable exploit as well!

* **Question 2**

Update (in a minimal way) the content of **file.h** to correct the vulnerability, since preserving the initial functional behavior.

Hint: think about possible **buffer overflows** ...