Insecure fruit stores challenges

1) store_1_logical.py

```
Welcome to the fruit store!
You may pick one for free, but not our 'Super Secret Fruit'...
Or can you?

1) Orange
2) Banana
3) Strawberry
?) Super Secret Fruit
Enter your choice:
```

The challenge requires participant to bypass the filters implemented in the program.

The filters are defined in these the following lines:

```
lowercase = list(map(chr, range(ord('a'), ord('z') + 1)))
uppercase = list(map(chr, range(ord('A'), ord('Z') + 1)))
symbols = ['', '!', '@', '#', '$', '%', '^', '&', '*', '(', ')', '-', '_', '=',
    '+', '[', ']', '{', '}', '|', ';', ':', '<', '>', ',', '.', '/', '?', "'",
    '"', '\\', '`', '~']
numeric = ['0', '4', '5', '6', '7', '8', '9']
```

As can be seen here, the developer manually defined each possible restricted inputs based on his logical perspective. The filters seem to be limiting almost all types of typeable inputs from typical keyboards only allowing input of **numeric 1, 2 and 3** only.

But what if it is not a normal input?

The intended solution for this challenge is to input something that is not in filters.

A **unicode.**

Participant can input any **unicode** characters to bypass the filters, eventually getting the *Super Secret Fruit*, which is a Watermelon!

```
Enter your choice: 🍉

Got the super secret WATERMELON! 🍉
CIS-USM{1LL0G1C4LLY_L0G1C4L}
```

ps: you can put in watermelon to get a watermelon!

2) store_2_overflow.cpp

```
Welcome to the another fruit store!
No free fruit this time around.
Our 'Super Secret Fruit' are on sale as well, but I don't think you can aford it...
Or can you?

1) Orange (RM 5)
2) Banana (RM 10)
3) Strawberry (RM 20)
4) Super Secret Fruit (RM100)
Your wallet: 99
Enter your choice:
```

The challenge requires participant to bypass the wallet limitation to get the *Super Secret Fruit*.

The wallet has predefined value, which is lower than the price of a single *Super Secret Fruit*. So the purchase of *Super Secret Fruit* by normal means are obviously not possible.

```
if (y <= 0) {
    cout << "Enter something more than zero!" << endl;
}
else if (y >= 2147483647) {
    cout << "Too much, too much!" << endl;
}
else {
    int cost = 0;
    if (x == 1) {
        cost = 5 * y;
    } else if (x == 2) {
        cost = 10 * y;
    } else if (x == 3) {
        cost = 20 * y;
    } else if (x == 4) {
        cost = 100 * y;
    }

    wallet -= cost;
```

Looking at the code, there are implementations of range validation; to ensure the input is not too small or too big. The upper limit of the validation, however, allows for quite a large number to be input in. The highest value that can be input in is **2147483646**, which in this case would cause something called **integer overflow** to happens.

Integer overflow happens when an **int** variable has been filled up to its maximum value, which is **2147483647**. Once this value is reached, any value that is added to the variable would caused an overflow, which happens to revert back the whole value to a negative value.

For example:

2147483647 + 1 = -2147483648

The intended solution for the challenge is to overflow the **int** variable of the total cost purchased so that it becomes a negative value. Participant can input **2147483646** to bypass the wallet limitation, eventually getting the *Super Secret Fruit*, which is a Coconut!

```
Enter how many fruits you want to buy: 2147483646
Total cost:-200
Remaining balance: 299

Got the super secret COCONUT! 🥥
CIS-USM{1T5_0V3RFL0W1N6}
```

ps: be careful when opening a coconut, you don't want its water to spill over (flow)!

3) store_3_execute.py

```
Welcome to the yet another fruit store!
We are selling our last, premium 'Super Secret Fruit' for today only!
But it is really expensive...
Or is it?

1) Orange (RM 5)
2) Banana (RM 10)
3) Strawberry (RM 20)
4) Super Secret Fruit (RM1000)
Your wallet: RM100
Enter your choice:
```

The challenge also requires participant to bypass the wallet limitation to get the *Super Secret Fruit*.
Same as previous challenge, the wallet has predefined value, which is lower than the price of a single *Super Secret Fruit*. Any normal input would resulted in failed attempt to get *Super Secret Fruit*.

```
if (proceed):
    exec(f'cost = {fruit_price} * {y}')
    wallet -= cost
```

Looking at the source code, there is one particular method that stands out the most.

The python **exec()** function.

This is a vulnerable function that should never be use in most application. It allows for execution of even an arbitrary command input by user if not properly validated.

In this scenario, this is especially true as this function allows participant to input other than what is intended; a **command injection**.

There are few solutions for the challenge. One is to manipulate the cost calculation formula itself, by explicitly inputting another expression so that the calculation would turn into negative value.

Example input: **1*-1000**

```
Enter how many fruits you want to buy: 1*-1000
Total cost: RM-1000000
Remaining balance: RM1000100
```

The other solution is to manipulate the variable value during the runtime itself, as the **exec()** function is powerful enough to allow us doing that!

Example input: **1; cost = 0**

```
Enter how many fruits you want to buy: 1; cost = 0
Total cost: RM0
Remaining balance: RM100
```

Participant able to bypass the wallet limitation, eventually getting the *Super Secret Fruit*, which is a Grape!

```
Got the super secret GRAPE! 🍇
CIS-USM{3X3CUT1N6_BYP455}
```

ps: two grapes met then fell in love. soon they were raisin kids.