Exercise-1: Circumventing weak security checks

Introduction

In this exercise, we have an implementation of chemical factory control software, where there are three ingredients: Lithium, Arsenic, Platinum. The implementation checks for levels of each chemicals and does not allow any chemical to be increased above 1000 or decreased below 0. As mentioned in the problem statement, these checks are weak and can be circumvented in a way. Logical method to circumvent these checks is to perform integer overflow.

Source code →

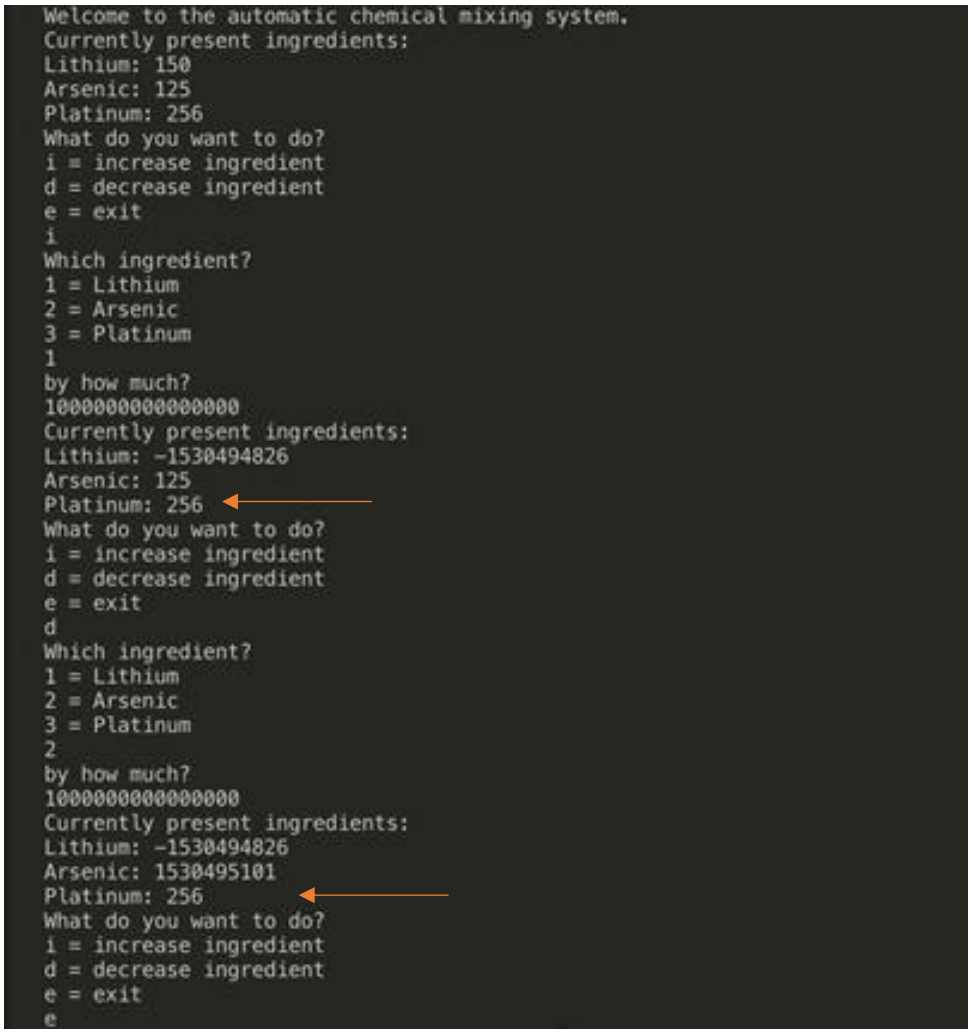
```
#include <stdio.h>
#include <stdlib.h>

    int value_lithium = 150;
    int value_arsenic = 125;
    int value_platinum = 256;

    void printIngredients() {
        printf("Currently present ingredients:\n");
        printf("Lithium: %d\n", value_lithium);
        printf("Arsenic: %d\n", value_arsenic);
        printf("Platinum: %d\n", value_platinum);
    }

    void incIngredient(int *ingredient, int incValue) {
        if (*ingredient + incValue < 1000) {
            *ingredient = *ingredient + incValue;
        } else {
            printf("Cannot increase ingredient to more than 1000!\n");
            exit(-1);
        }
    }

    void decIngredient(int *ingredient, int decValue) {
        if (*ingredient - decValue > 0) {
            *ingredient = *ingredient - decValue;
        } else {
            printf("Cannot decrease ingredient to less than 0!\n");
            exit(-1);
        }
    }

    void executeCommand() {
        char input[20];

        printf("What do you want to do?\n"
               "i = increase ingredient\n"
               "d = decrease ingredient\n"
               "e = exit\n");

        fgets(input, sizeof(input), stdin);

        if (*input == 'e') {
            exit(0);
        }

        if (*input == 'i' || *input == 'd') {
            void (*operation)(int*, int);
```

```
        if (*input == 'i') {
            operation = &incIngredient;
        } else {
            operation = &decIngredient;
        }

        printf("Which ingredient?\n"
            "1 = Lithium\n"
            "2 = Arsenic\n"
            "3 = Platinum\n");

        fgets(input, sizeof(input), stdin);
        int value = atoi(input);

        int *ingredient;

        switch(value) {
            case 1:
                ingredient = &value_lithium;
                break;
            case 2:
                ingredient = &value_arsenic;
                break;
            case 3:
                ingredient = &value_platinum;
                break;
            default:
                printf("You must provide an ingredient!\n");
                exit(-1);
        }

        printf("by how much?\n");

        fgets(input, sizeof(input), stdin);
        operation(ingredient, atoi(input));
    } else {
        printf("Please provide a valid command.\n");
    }
}

int main() {

    printf("Welcome to the automatic chemical mixing system.\n");
    char input[10];

    while(1) {
        printIngredients();
        executeCommand();
    }
}
```

Integer Overflow

Memory allocation, by principle behaves in a circular fashion. Hence when we try to store a value which requires more memory, to store in, than the memory provided, the value resets and negates itself. This behavior of Integer can be used, to bypass the checks mentioned in the problem statement.

Bypassing the chemical level checks



Mitigation

- **Datatype Upcasting**

  In datatype upcasting the variable is type casting it to which takes more memory and then put range checks on the variable.

- **Use of libraries**

  All popular languages have libraries that provide support for type range checks. Making use of such libraries can mitigate the risk of Integer Overflows.