Taint Analysis Exercise

Security Testing (145322)

Francisco Manuel Colmenar Lamas
219757

Trento, November 2020

Contents

List of Figures

1. INTRODUCTION

In this report a Taint Analysis is going to be performed to the files *sql-injection.py* and *integer_overflow.c*. Furthermore, the flow graphs and the Gen-Kill-In-Out tables are also provided in order to display the correct development of the Taint Analysis.

2. SQL-INJECTION.PY

2.1. Vulnerable version

To start with, the original code belonging to the file *sql-injection.py*, which is vulnerable to sql injection, is provided below in order to facilitate the understanding of the *flow graph* and the *Gen-Kill-In-Out tables*.

Note that the original line numbers, the ones which would appear at an IDE, are modified in order to represent the different states which are going to be displayed at the *flow graph* in the following graph.

```
1  import sys
   import os
   import sqlite3

   # Connect to database
1  conn = None
2  try:
3      conn = sqlite3.connect('users.db')
4  except Exception:
5      print("Can't connect to the database")
6      sys.exit(-1)


7  print("Welcome to this vulnerable database reader")
8  print("You have to login first")


9  print("Insert your user-id")
10 user_id = input()


11 print("Insert your password")
```

```
password = input()



retrieve_user = "SELECT * FROM credentials WHERE user_id = '" +
    user_id + "' and password = '" + password + "';"
cursor = conn.execute(retrieve_user)



entries = cursor.fetchall()
if len(entries) > 0:
    print("\n===Logged-in=====")
    retrieve_user = "SELECT * FROM accounts WHERE user_id = '" +
    user_id + "';"
    cursor = conn.execute(retrieve_user)
    entries = cursor.fetchall()
    for entry in entries:
        user_id, first_name, last_name, phone = entry
        print()
        print("Here is {} data:".format(user_id))
        print("user-id=", user_id)
        print("first_name=", first_name)
        print("last_name=", last_name)
        print("phone", phone)
else:
    print("Wrong credentials")
```

CÓDIGO 2.1: Sql-injection.py code

2.1.1. Flow Graph

The flow graph corresponding to the vulnerable version of sql-injection.py can be seen in the above diagram.

except
1
try
4
2
10
17
5
3
11
18
6
7
12
19
8
13
20
31
9
14
21
30
29
15
22
Else
16
If -> True
23
28
27
26
25
24

2.1.2. Gen-Kill-In-Out Table

The *Gen-Kill-In-Out* table displaying the different states of the previous graph is showed below.

As it can be seen, at the lines 12 and 17 the Out column has the variables *user_id* and *password*, at the lines 13 and 18 the variable *retrieve_user* will became tainted. Additionally, this variable is the one which used to access the DB. Therefore, it is deduced that there is a SQLi vulnerability thanks to the *Gen-Kill-In-Out* table.

Vulnerable sql-injection.py

State	Gen	Kill	In	Out
1	{}	{}	{}	{}
2	{}	{}	{}	{}
3	{}	{}	{}	{}
4	{}	{}	{}	{}
5	{}	{}	{}	{}
6	{}	{}	{}	{}
7	{}	{}	{}	{}
8	{}	{}	{}	{}
9	{}	{}	{}	{}
10	{user_id}	{}	{}	{user_id}
11	{}	{}	{user_id}	{user_id}
12	{password}	{}	{user_id}	{user_id, password}
13	{[user_id -> T | | password -> T] retrieve_user}	{[user_id -> F & password -> F] retrieve_user}	{user_id, password}	{user_id, password, retrieve_user}
14	{[retrieve_user -> T]cursor}	{[retrieve_user -> F]cursor}	{user_id, password, retrieve_user}	{user_id, password, retrieve_user, cursor}
15	{[cursor -> T]entries}	{[cursor -> F]entries}	{user_id, password, retrieve_user, cursor}	{user_id, password, retrieve_user, cursor, entries}
16	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
17	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
18	{[user_id -> T] retrieve_user}	{[user_id -> F] retrieve_user}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
19	{[retrieve_user -> T]cursor}	{[retrieve_user -> F]cursor}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
20	{[cursor -> T]entries}	{[cursor -> F]entries}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
21	{[entries -> T] entry}	{[entries -> F] entry}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
22	{[entry -> T] user_id, first_name, last_name, phone}	{[entry -> F] user_id, first_name, last_name, phone}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
23	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
24	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
25	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
26	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
27	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
28	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
29	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
30	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}
31	{}	{}	{user_id, password, retrieve_user, cursor, entries}	{user_id, password, retrieve_user, cursor, entries}

Figure 2.1: Gen-Kill-In-Out Table belonging to sql-injection.py vulnerable.

2.2. Fixed version

The original code belonging to the file *sql-injection.py* which is not vulnerable to SQLi is the following one.

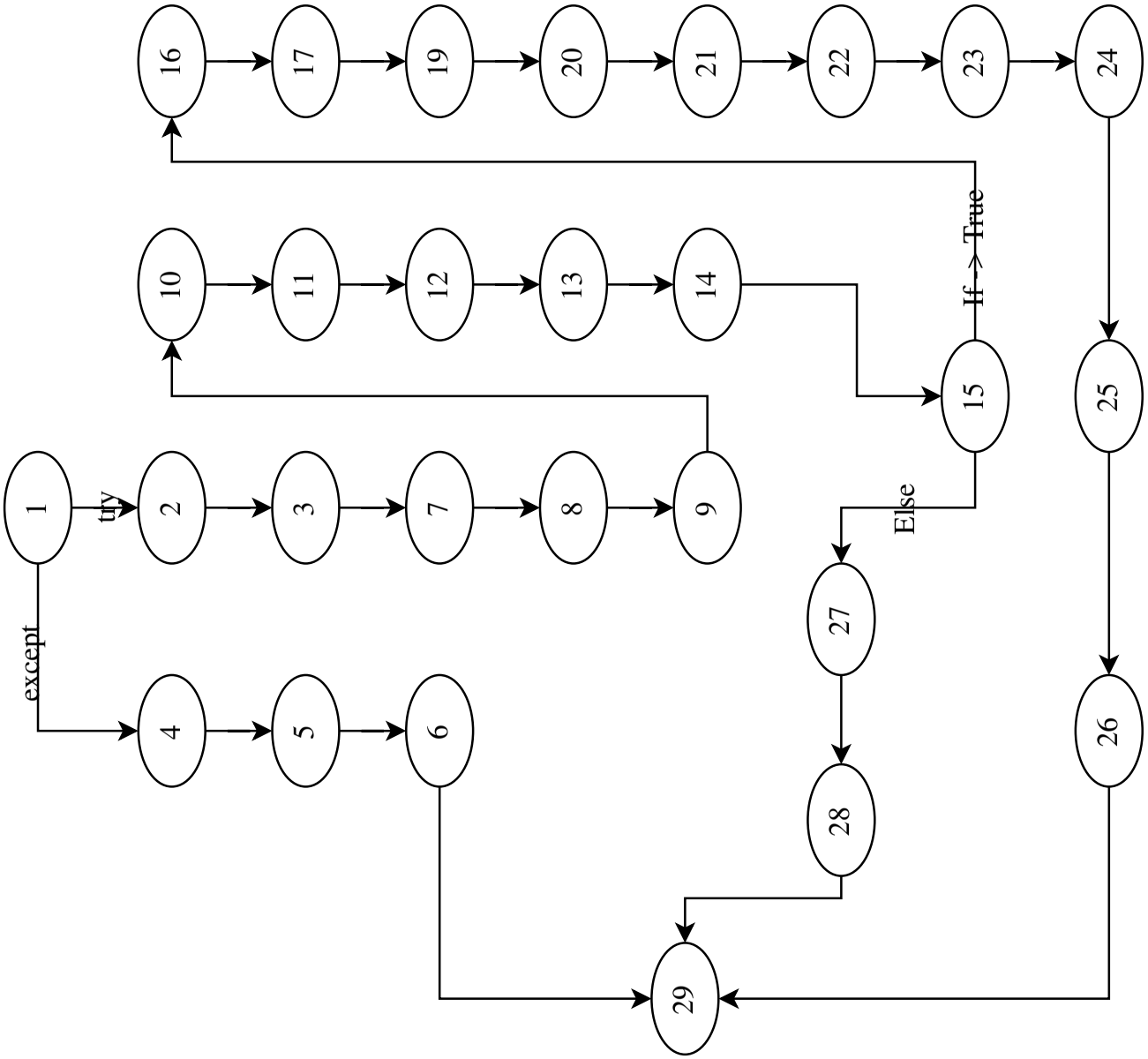
```
import sys
import os
import sqlite3

# Connect to database
conn = None
try:
    conn = sqlite3.connect('users.db')
except Exception:
    print("Can't connect to the database")
    sys.exit(-1)


print("Welcome to this vulnerable database reader")
print("You have to login first")


print("Insert your user-id")
user_id = input()


print("Insert your password")
password = input()


cursor = conn.execute("SELECT * FROM credentials WHERE user_id = ?
    and password = ?;", (user_id, password))


entries = cursor.fetchall()
```

```
15 if len(entries) > 0:
16     print("\n===Logged-in=====")
17     cursor = conn.execute("SELECT * FROM accounts WHERE user_id = ?;
       ", user_id)
18     entries = cursor.fetchall()
19     for entry in entries:
20         user_id, first_name, last_name, phone = entry
21         print()
22         print("Here is {} data:".format(user_id))
23         print("user-id=", user_id)
24         print("first_name=", first_name)
25         print("last_name=", last_name)
26         print("phone", phone)
27 else:
28     print("Wrong credentials")
```

CÓDIGO 2.2: Sql-injection.py code fixed

2.2.1. Flow Graph

The flow graph corresponding to the fixed version of sql-injection.py can be seen in the above diagram.


16
17
19
20
21
22
23
24
If -> True
10
11
12
13
14
15
25
1
try
2
3
7
8
9
Else
27
except
4
5
6
26
28
29

2.2.2. Gen-Kill-In-Out Table

The *Gen-Kill-In-Out* table displaying the different states of the previous graph is showed below.

As it can be seen, at the lines 13 and 17 the Kill column has the variable *cursor* as a consequence of the new sanitized query construction. Therefore, the DB access is not susceptible to SQLi as well as the further use of *entries* and *entry* variables.

Fixed sql-injection.py

State	Gen	Kill	In	Out
1	{}	{}	{}	{}
2	{}	{}	{}	{}
3	{}	{}	{}	{}
4	{}	{}	{}	{}
5	{}	{}	{}	{}
6	{}	{}	{}	{}
7	{}	{}	{}	{}
8	{}	{}	{}	{}
9	{}	{}	{}	{}
10	{user_id}	{}	{}	{user_id}
11	{}	{}	{user_id}	{user_id}
12	{password}	{}	{user_id}	{user_id, password}
13	{}	{cursor}	{user_id, password}	{user_id, password}
14	{[cursor -> T]entries}	{[cursor -> F]entries}	{user_id, password}	{user_id, password}
15	{}	{}	{user_id, password}	{user_id, password}
16	{}	{}	{user_id, password}	{user_id, password}
17	{}	{cursor}	{user_id, password}	{user_id, password}
18	{[cursor -> T]entries}	{[cursor -> F]entries}	{user_id, password}	{user_id, password}
19	{[entries -> T] entry}	{[entries -> F] entry}	{user_id, password}	{user_id, password}
20	{[entry -> T] user_id, first_name, last_name, phone}	{[entry -> F] user_id, first_name, last_name, phone}	{user_id, password}	{password}
21	{}	{}	{password}	{password}
22	{}	{}	{password}	{password}
23	{}	{}	{password}	{password}
24	{}	{}	{password}	{password}
25	{}	{}	{password}	{password}
26	{}	{}	{password}	{password}
27	{}	{}	{user_id, password}	{user_id, password}
28	{}	{}	{user_id, password}	{user_id, password}
29	{}	{}	{user_id, password}	{user_id, password}

Figure 2.2: Gen-Kill-In-Out Table belonging to sql-injection.py fixed.

3. INTEGER_OVERFLOW.C

3.1. Vulnerable version

The original code belonging to the file *integer_overflow.c* which is vulnerable to IoF is the following one.

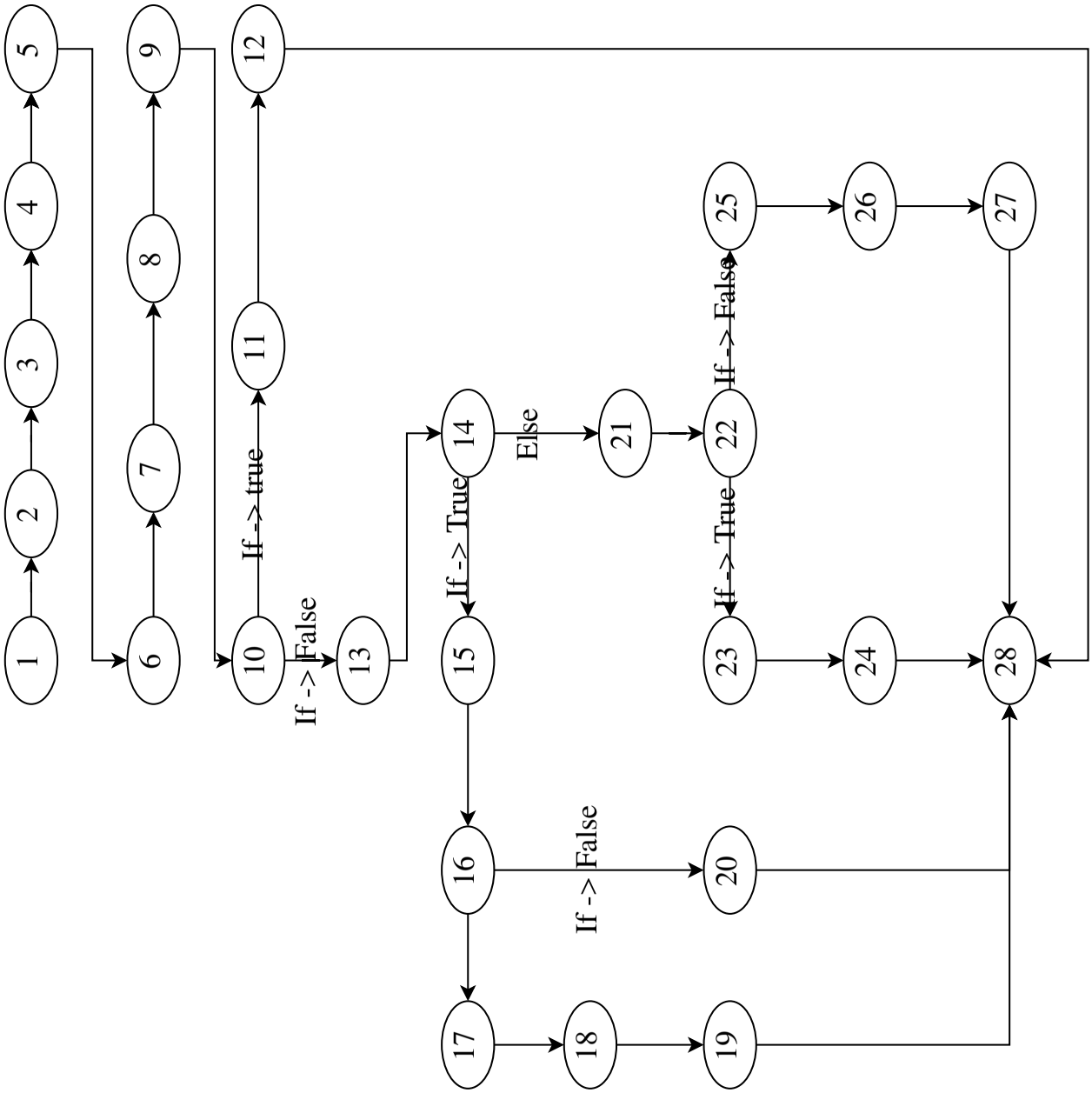
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf("Hello, which product do you want to buy?\n");
  printf("1) IPhone 12\n");
  printf("2) IPhone 12 Pro\n");
  printf("3) IPhone 12 Pro Max Max\n");

  // Get item
  int item_choice;
  scanf("%d", &item_choice);



  printf("Great device, how many?\n");
  int item_quantity;
  scanf("%d", &item_quantity);



  if (item_quantity <= 0) {
    printf("You should buy at least one Iphone!\n");
    return -1;
  }



  int insurance = 1200;
```

```
14    if (item_choice == 3){
15      int price = 1500*item_quantity + insurance;
16      if (price == 0) {
17        printf("You solved the problem\n");
18        printf("The Iphone Max Max is yours\n");
19        return 1;
        }
20      printf("You have to pay %d\n", price);
      }
21    else {
22      if (item_quantity > 3) {
23        printf("You can buy maximum 3\n");
24        return -1;
        }
25      int price = 1000*item_quantity;
26      printf("You have to pay %d\n", price);
      }
27    return 0;
  }
```

CÓDIGO 3.1: integer_overflow.c code

3.1.1. Flow Graph

The flow graph corresponding to the vulnerable version of *integer_overflow.c* can be seen in the above diagram.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
If -> true
If -> False
If -> True
Else
If -> False
If -> True
If -> False

3.1.2. Gen-Kill-In-Out Table

The *Gen-Kill-In-Out* table displaying the different states of the previous graph is showed below.

As it can be seen, at the line 15 and 25, as at the IN column the variable *item_quantity* is presented, the variable *price* gets tainted too, which produces a IoF vulnerability.

Vulnerable integer_overflow.c				
State	Gen	Kill	In	Out
1	{}	{}	{}	{}
2	{}	{}	{}	{}
3	{}	{}	{}	{}
4	{}	{}	{}	{}
5	{}	{}	{}	{}
6	{item_choice}	{}	{}	{item_choice}
7	{}	{}	{item_choice}	{item_choice}
8	{}	{}	{item_choice}	{item_choice}
9	{item_quantity}	{}	{item_choice}	{item_choice, item_quantity}
10	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
11	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
12	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
13	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
14	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
15	{[item_quantity -> T] price}	{[item_quantity -> F] price}	{item_choice, item_quantity}	{item_choice, item_quantity, price}
16	{}	{}	{item_choice, item_quantity, price}	{item_choice, item_quantity, price}
17	{}	{}	{item_choice, item_quantity, price}	{item_choice, item_quantity, price}
18	{}	{}	{item_choice, item_quantity, price}	{item_choice, item_quantity, price}
19	{}	{}	{item_choice, item_quantity, price}	{item_choice, item_quantity}
20	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity, price}
21	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
22	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
23	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
24	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
25	{[item_quantity -> T] price}	{[item_quantity -> F] price}	{item_choice, item_quantity}	{item_choice, item_quantity, price}
26	{}	{}	{item_choice, item_quantity, price}	{item_choice, item_quantity, price}
27	{}	{}	{item_choice, item_quantity, price}	{item_choice, item_quantity, price}
28	{}	{}	{item_choice, item_quantity, price}	{item_choice, item_quantity, price}

Figure 3.1: Gen-Kill-In-Out Table belonging to integer_overflow.c vulnerable.

3.2. Fixed version

The original code belonging to the file *integer_overflow.c* which is not vulnerable to IoF is the following one.

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

int main()
{
1   printf("Hello, which product do you want to buy?\n");
2   printf("1) IPhone 12\n");
3   printf("2) IPhone 12 Pro\n");
4   printf("3) IPhone 12 Pro Max Max\n");

    // Get item
5   int item_choice;
6   scanf("%d", &item_choice);



7   printf("Great device, how many?\n");
8   int item_quantity;
9   scanf("%d", &item_quantity);



10  if (item_quantity <= 0) {
11    printf("You should buy at least one Iphone!\n");
12    return -1;
    }



13  int insurance = 1200;
14  if (item_choice == 3){
15    if (1500 < (INT_MAX - insurance) / item_quantity) {
16       int price = 1500*item_quantity + insurance;
```

```
17          if (price == 0) {
18            printf("You solved the problem\n");
19            printf("The Iphone Max Max is yours\n");
20            return 1;
            }
21      printf("You have to pay %d\n", price);
        }
      }
22    else {
23      if (item_quantity > 3) {
24        printf("You can buy maximum 3\n");
25        return -1;
        }
26          if (1000 < (INT_MAX / item_quantity) {
27              int price = 1000 * item_quantity;
28              printf("You have to pay %d\n", price);
            }
      }
29    return 0;
  }
```

CÓDIGO 3.2: integer_overflow_fixed.c code fixed

3.2.1. Flow Graph

The flow graph corresponding to the fixed version of *integer_overflow.c* can be seen in the above diagram.

1
2
3
4
5
6
7
8
9
10
If -> true
11
12
If -> False
13
18
17
16
True
15
If -> True
14
19
If -> False
False
Else
22
20
21
24
If -> True
23
27
If -> False
25
26
True
28
False
30
29

3.2.2. Gen-Kill-In-Out Table

The *Gen-Kill-In-Out* table displaying the different states of the previous graph is showed below.

In this case, as at the lines 15 and 26 the variable *item_quantity* is added to the Kill column, in the following lines the variable *price* is not tainted. Therefore, the IoF vulnerability is fixed.

Fixed integer_overflow.c

State	Gen	Kill	In	Out
1	{}	{}	{}	{}
2	{}	{}	{}	{}
3	{}	{}	{}	{}
4	{}	{}	{}	{}
5	{}	{}	{}	{}
6	{item_choice}	{}	{}	{item_choice}
7	{}	{}	{item_choice}	{item_choice}
8	{}	{}	{item_choice}	{item_choice}
9	{item_quantity}	{}	{item_choice}	{item_choice, item_quantity}
10	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
11	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
12	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
13	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
14	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
15	{}	{item_quantity}	{item_choice, item_quantity}	{item_choice}
16	{[item_quantity -> T] price}	{[item_quantity -> F] price}	{item_choice}	{item_choice}
17	{}	{}	{item_choice}	{item_choice}
18	{}	{}	{item_choice}	{item_choice}
19	{}	{}	{item_choice}	{item_choice}
20	{}	{}	{item_choice}	{item_choice}
21	{}	{}	{item_choice}	{item_choice}
22	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
23	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
24	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
25	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}
26	{}	{item_quantity}	{item_choice, item_quantity}	{item_choice}
27	{[item_quantity -> T] price}	{[item_quantity -> F] price}	{item_choice}	{item_choice}
28	{}	{}	{item_choice}	{item_choice}
29	{}	{}	{item_choice}	{item_choice}
30	{}	{}	{item_choice, item_quantity}	{item_choice, item_quantity}

Figure 3.2: Gen-Kill-In-Out Table belonging to integer_overflow.c fixed.