TDA602 Lab1 - ToCToU

Olof Magnusson, Michalis Kaili,

Group 5

1 Part 1: Exploit your program

1.1 What is the shared resource? Who is sharing it?

The shared resource in this program is the files `wallet.txt` and `pocket.txt` since they are read and writable by instances of the ShoppingCart.

1.2 What is the root of the problem?

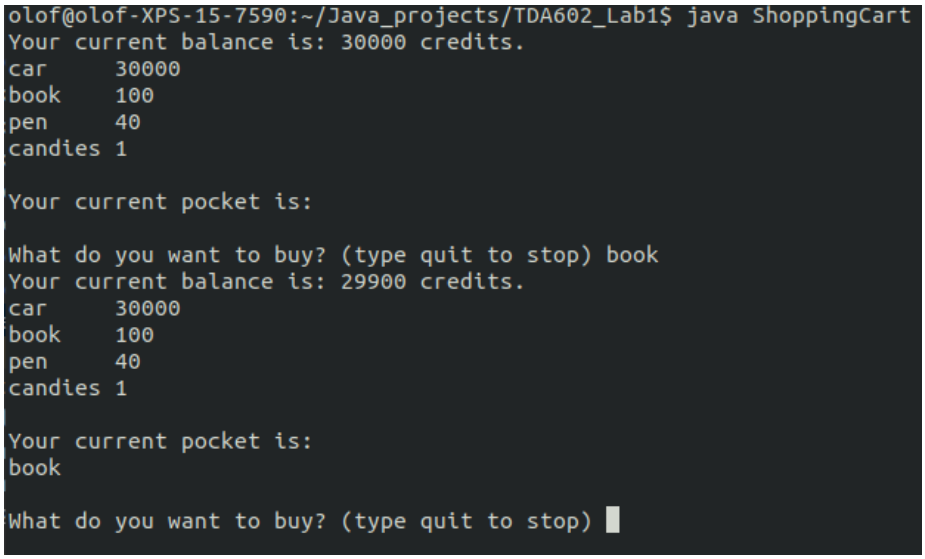
The root of the problem is execution of concurrent programs where at least one of the threads writes to the variable in the shared memory before another thread has confirmed the balance. We race to assign to balance because of the interleaving.

1.3 Explain in detail how you can attack this system

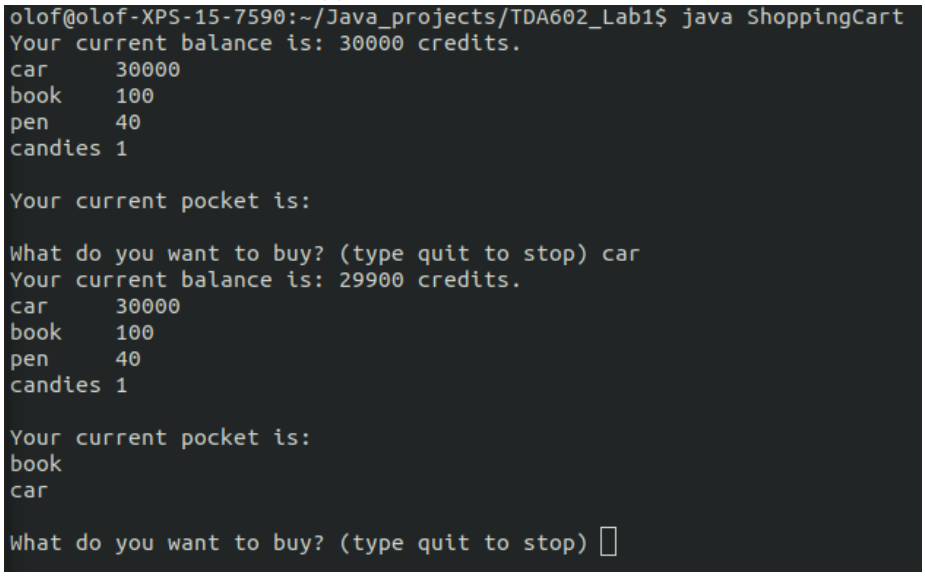
The program in question does not have any thread safety with concurrent access to a resource. This implies that if we have multiple requests from the backend, the result will depend on the interleaving whereby an attacker could exploit this non-determinism and get a product by paying less than its value.

1.4 Provide the program output and result, explaining the interleaving to achieve them.

The result of running two instances of the program in parallel can be show in Figure a and b whereby we have the possibility to buy a car by paying less than its value. As there is none synchronization event between the two threads, the both processes can perform local computation and assign to global balance because of the interleaving.



(a) Buying a book in the shop



(b) Buying a car with insufficient founds by exploiting the interleaving between the threads

2 Part 2: Fix the API

Java does not emphasises any language-level guarantees when dealing with concurrent programs and there could exist more possible data races in the program. Besides `withdraw`, another method that has some challenges is `addProduct` which takes a product and adds it to the pocket in an unsafe manner. This method can be exploited by the interleaving even though we raised an exception for

those events shown in Figure 1. No logic checks either that the file has changed before closing. We'll discuss them more closely now by fixing the API and implementing countermeasures together with a discussion.

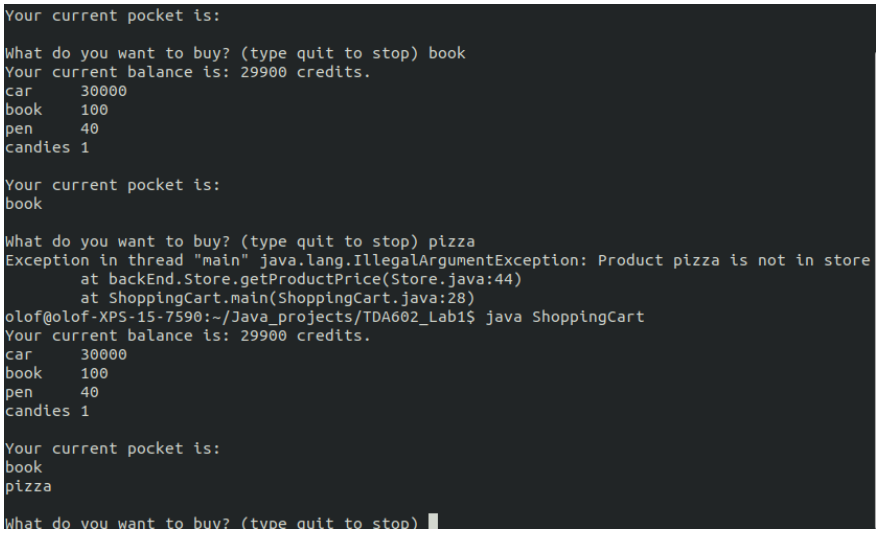


Figure 1: Illustrating the possibility to buy non-products in the program

Parallelism is desirable for performance and computation when dealing with processors that enable multi-cores. However, minimizing synchronization between different threads could have side effects by the source of non-determinism. A sequential computation is more desirable with shared variables that isolate the processes to avoid the timing-dependent data races discussed in the previous section. Using lock objects provide soundness and essentially allows one thread to invoke the lock, perform the computations, and release whereby the second thread can grab the lock. Because locks are objects with states lock must be released after execution independent of the result. Using try and finally statement is therefore desirable since an attacker could force the program into a deadlock. This approach assumes that the code written by the developer is correct since sequential computations with schedulers still can provide errors due to bad synchronization. Another thing to take into consideration is using exceptions can have a side effect if an attacker could observe and understand information about the errors in the stack trace.

Another attack scenario that we find possible is when we are opening and closing files. A memory-efficient operation is to recheck the value between locking and releasing the variable to detect any changes. In a more practical application, the difference in computation speed could be measured to locate if the data is stored in cache or main memory. However, we believe that a good understanding of the balance between performance and security is needed when implementing language-based security mechanisms.

```
olof@olof-XPS-15-7590:~/Java_projects/TDA602/Lab1$ java ShoppingCart
Your current balance is: 30000 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:

What do you want to buy? (type quit to stop) book
Your new balance is: 29900
Your current balance is: 29900 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:
book

What do you want to buy? (type quit to stop)
```

(a) Buying a book

```
Your current balance is: 29900 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:
book

What do you want to buy? (type quit to stop) car
Exception in thread "main" java.lang.Exception: Insufficent founds
        at backEnd.Wallet.safeWithdraw(Wallet.java:51)
        at ShoppingCart.main(ShoppingCart.java:27)
olof@olof-XPS-15-7590:~/Java_projects/TDA602/Lab1$ java ShoppingCart
Your current balance is: 29900 credits.
car     30000
book    100
pen     40
candies 1

Your current pocket is:
book

What do you want to buy? (type quit to stop)
```

(b) Trying to buy a car again where an exception was raised. Here we use synchronization between the threads to deal with the previous problem of interleaving

Trying to buy a pizza:

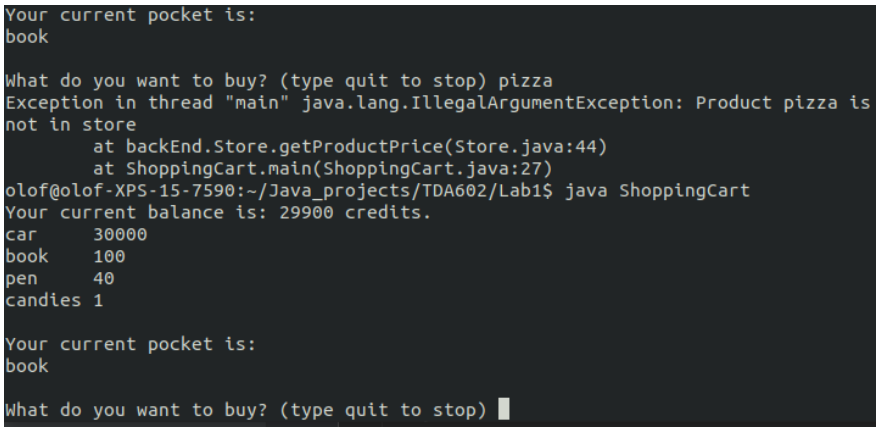


Figure 2: Trying to buy a pizza that is not in the store