Lab I – Understanding the TOCTOU Vulnerability

CPS 499-02/592-02

Software/Language Based Security

Fall 2020

Dr. Phu Phung

Evan Krimpenfort

Task I: Exploit the Program

Part a: Prove your exploitation:

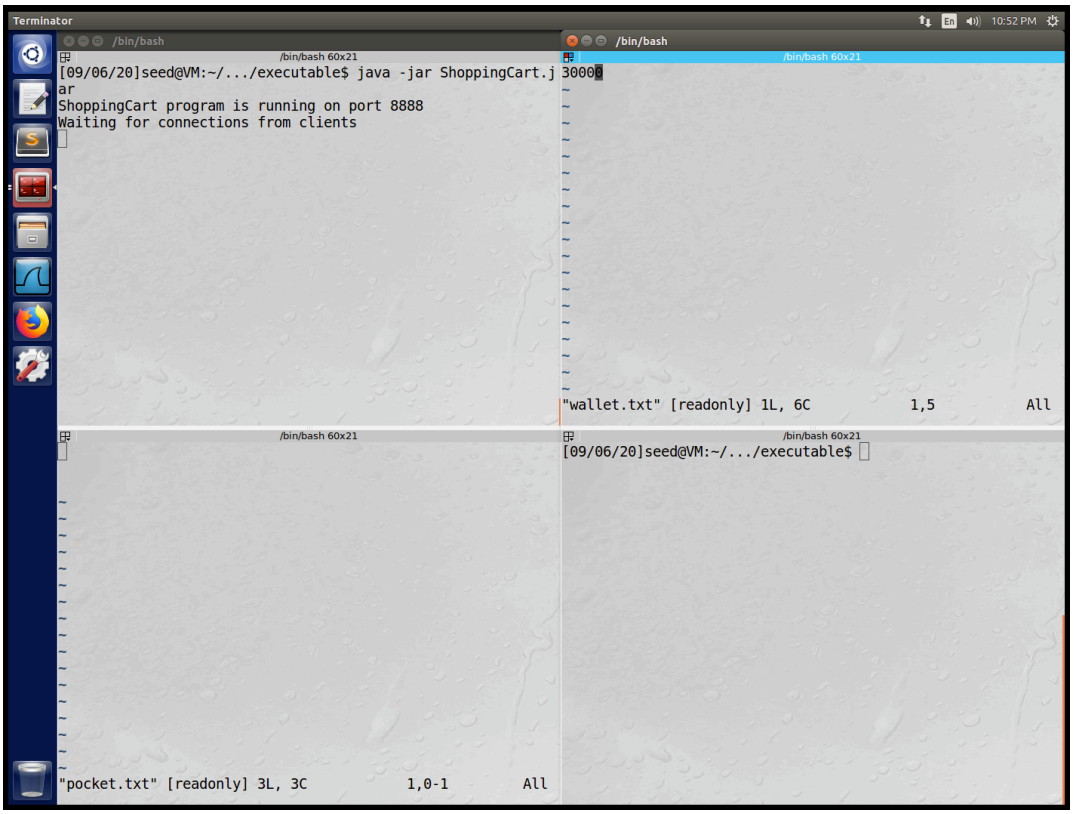

Figure 1: Wallet.txt and Pocket.txt shown before the vuln is exploited

Here, in figure 1, we can see that the *pocket.txt* is empty in the lower left-hand corner. We can also see that *wallet.txt* is 30000 in the upper right-hand corner. With this in place, we are going to pull off a three-car heist.

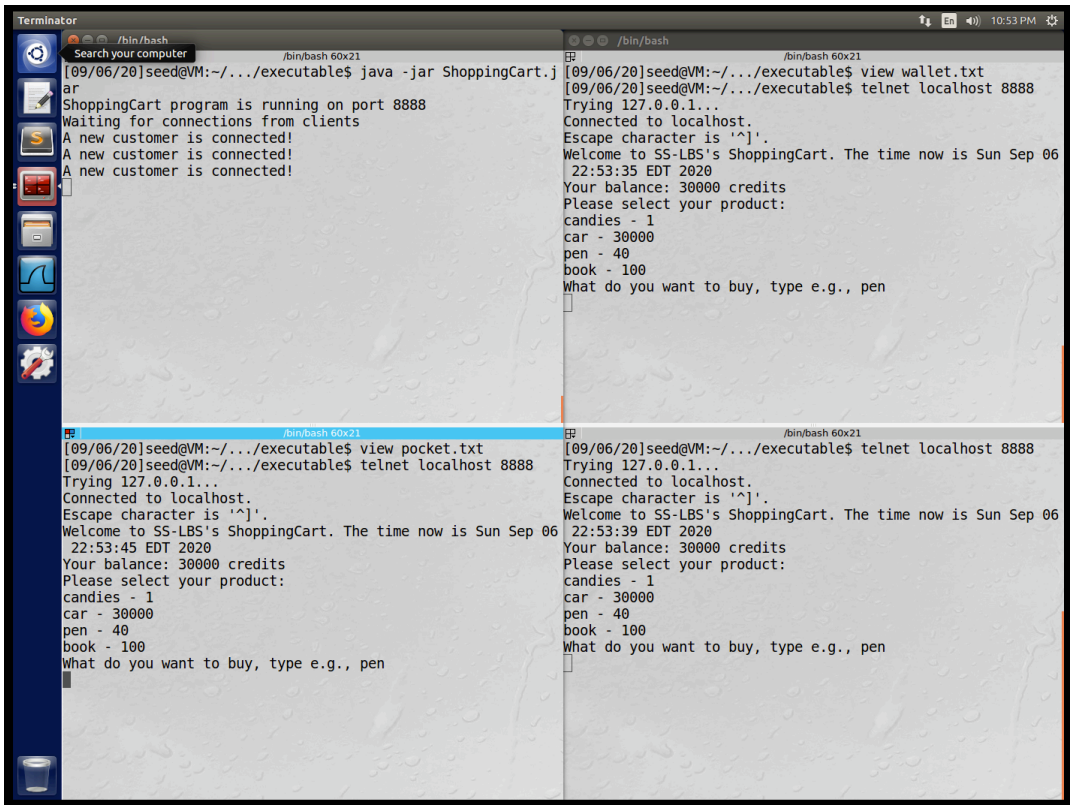

Figure 2: Connection of three buyers is shown

Here, in figure 2, I have set up three buyers (telnet connections to the host 8888). Our goal is to buy three cars with just the 30000 we have (which is just one car). We can see that in figure 3 below us.

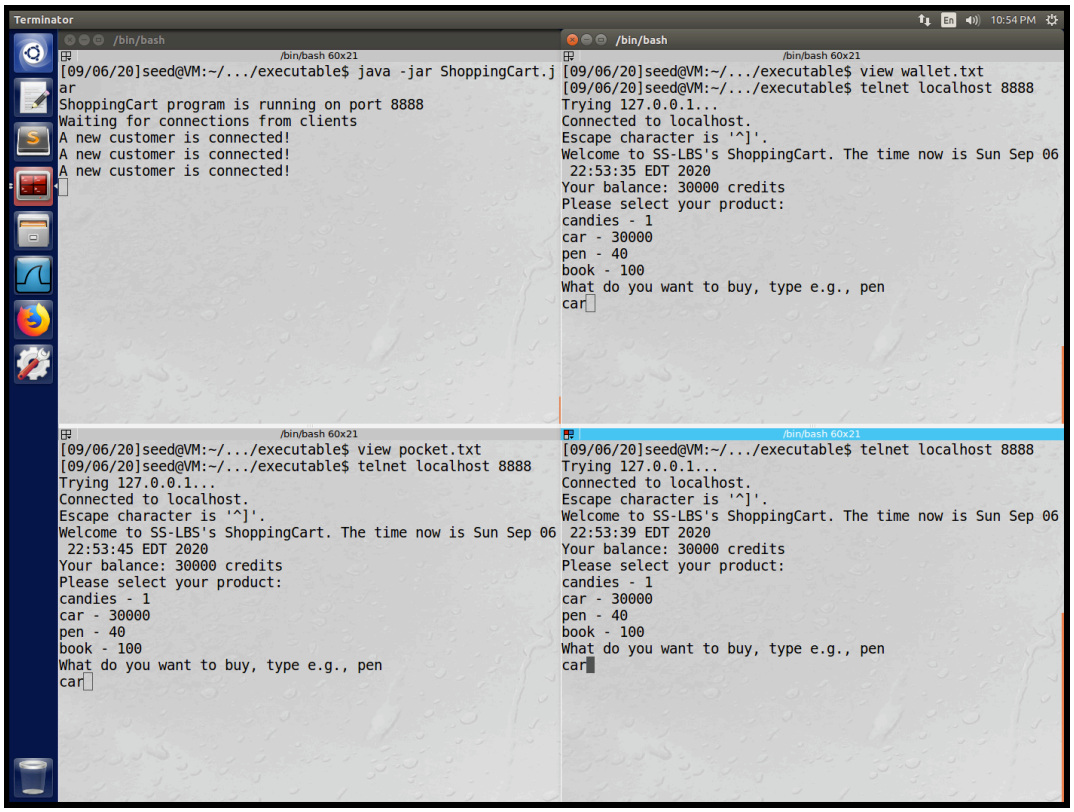

Figure 3: Setup each buyer with three cars

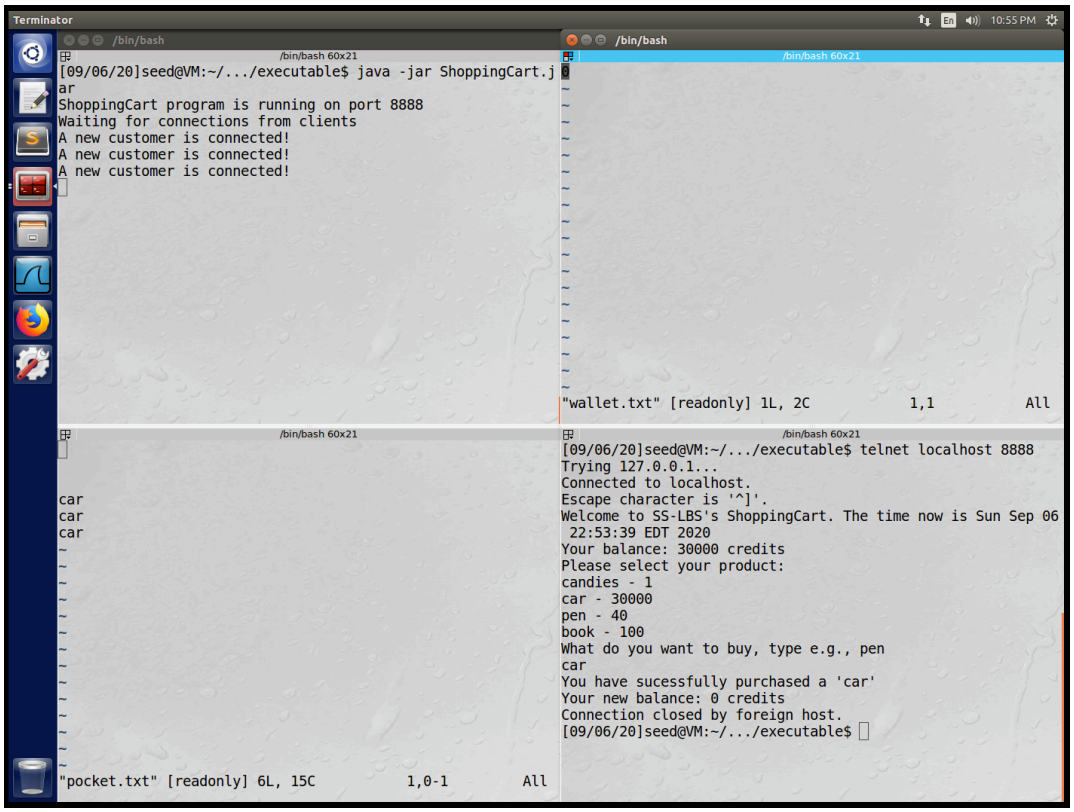

Figure 4: Three cars have been purchased with just 30000 credits

Success! We were able to show in figure 4 that we bought three cars with only an allowance for one car. After each car was bought, each client cut their connection with the host. That can be seen in the bottom right-hand corner of figure 4.

Part b: Explain…

I: How did you attack the system?

I attacked the system by opening up multiple client connections with the *ShoppingCart.jar* service before buying anything (you can see that by looking at the jump between figure 2 and figure 3). Each client connection starts out with 30000 credits to use. This amount allows us to buy just one car. But, the vulnerability is seen when one connection buys a car and the other two connections weren't updated from the other buyer's purchase. Thus, we still have 30000 credits in two other client connections. That allows the purchase of two more cars. And, as you can see in figure 4, we successfully got three cars with just 30000 credits.

II: Why does this attack happen (Include in detail)?

This attack happens because of a data race between all three of the clients. Normally, if you kept it to only one buyer at a time, no race would ever occur because…

-Person-	-Action-	-Item-	-Amount-	-Total-
Buyer 1:	**Retrieves**	**~~~**	**30000**	**30000**
Buyer 1:	**Spends**	**book**	**-100**	**29900**
Buyer 2:	**Retrieves**	**~~~**	**29900**	**29900**
Buyer 2:	**Spends**	**pen**	**-40**	**29860**

And so on. But, if you put buyers (client connections) in unison, then this is what the data race looks like…

-Person-	-Action-	-Item-	-Amount-	-Total-
Buyer 1:	**Retrieves**	**~~~**	**30000**	**30000**
Buyer 2:	**Retrieves**	**~~~**	**30000**	**30000**
Buyer 3:	**Retrieves**	**~~~**	**30000**	**30000**
Buyer 2:	**Spends**	**car**	**-30000**	**0**

***notice how since two different buyers retrieved 30000 and that amount is never overwritten for them. That's the key here – Data was written to that matters to the other buyers but they won't get to see that.*

-Person-	-Action-	-Item-	-Amount-	-Total-
Buyer 1:	**Spends**	**car**	**-30000**	**0**
Buyer 3:	**Spends**	**car**	**-30000**	**0**

Each buyer started out with 30000 credits and, as each person bought something, that amount was never updated because they didn't re-retrieve it. Thus, three cars were bought.

Task II: Source Code and Security Analysis

Part a: Source Code Modification:

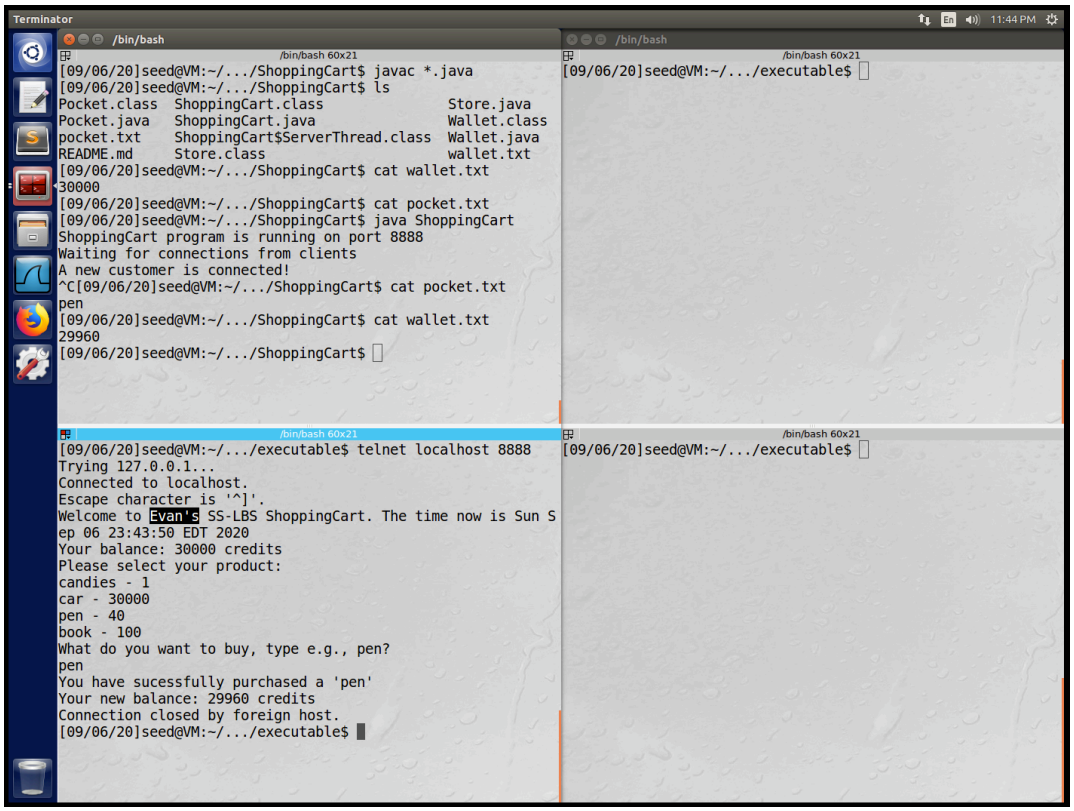

Figure 5: Evan's SS-LBS Shopping cart is running – a pen was bought

Part b: Which lines of code caused the program to be exploited?

The lines where the program becomes vulnerable would be the extraction of data in *ShoppingCart.java* (roughly line 55 – *int balance = Wallet.getBalance();*) and the saving of the new data later in the same file when the client buys something (roughly line 73 – *Wallet.setBalance(balance-price);*). This is vulnerable because it does it on a thread per thread basis.

Part c: Suggest solutions to fix the issue. Provide technical detail.

The exploit could be fixed either one of two ways. The first way could be by doing a version of a mutex, where, the buyers would have to follow something similar to a line that we see every single day in our own lives. If one client connects, and another client connects, and a third client connects, the first one would have a balance loaded and the other two would be waiting for the current buyer to be done. That way the clients can follow an order and the program would eradicate the race war with the private data *Wallet*. Read *Balance*, write *Balance*, repeat. Only one person has access to the wallet at that time. The other way you could do it is where the

server updates its clients with new information after a client closes their socket. Thus, forcing all of the threads to have the correct volatile data (the *Wallet*). This would allow multiple clients to work much faster than the last solution, but may be more difficult to implement.

Part d: Explain the functionalities of the following API's and list any issues:

I: Wallet.getBalance()

This function sets the file pointer to 0 and reads the first line of the "wallet.txt" file. This line that is read is then parsed into a integer and is returned as that type.

II: Store.getPrice(String product)

With the string *product* that comes into the function, it is then put through nested if's to return the proper charge to the buyer. The items for sale are candies, a car, a pen, and a book. If the client does not type in an item listed out from the server, a price of 0 is returned and the balance does not go down.

III: Wallet.setBalance(int newBalance)

This function wipes the file by setting the length to 0. It takes the *newBalance* and converts that variable to a string so that the private *RandomAccessFile* can write those bytes to the "wallet.txt" file.

IV: Pocket.addProduct(String Product)

The function *addProduct* adds a successfully purchased item to the end of the "pocket.txt" file. The first line in the subroutine sets the cursor to the end of the file and writes the *Product* to the end adding a new line. **ISSUE:** when the client purchases a product that is not listed from the server (example, coat), the item purchased is still placed into the "pocket.txt."