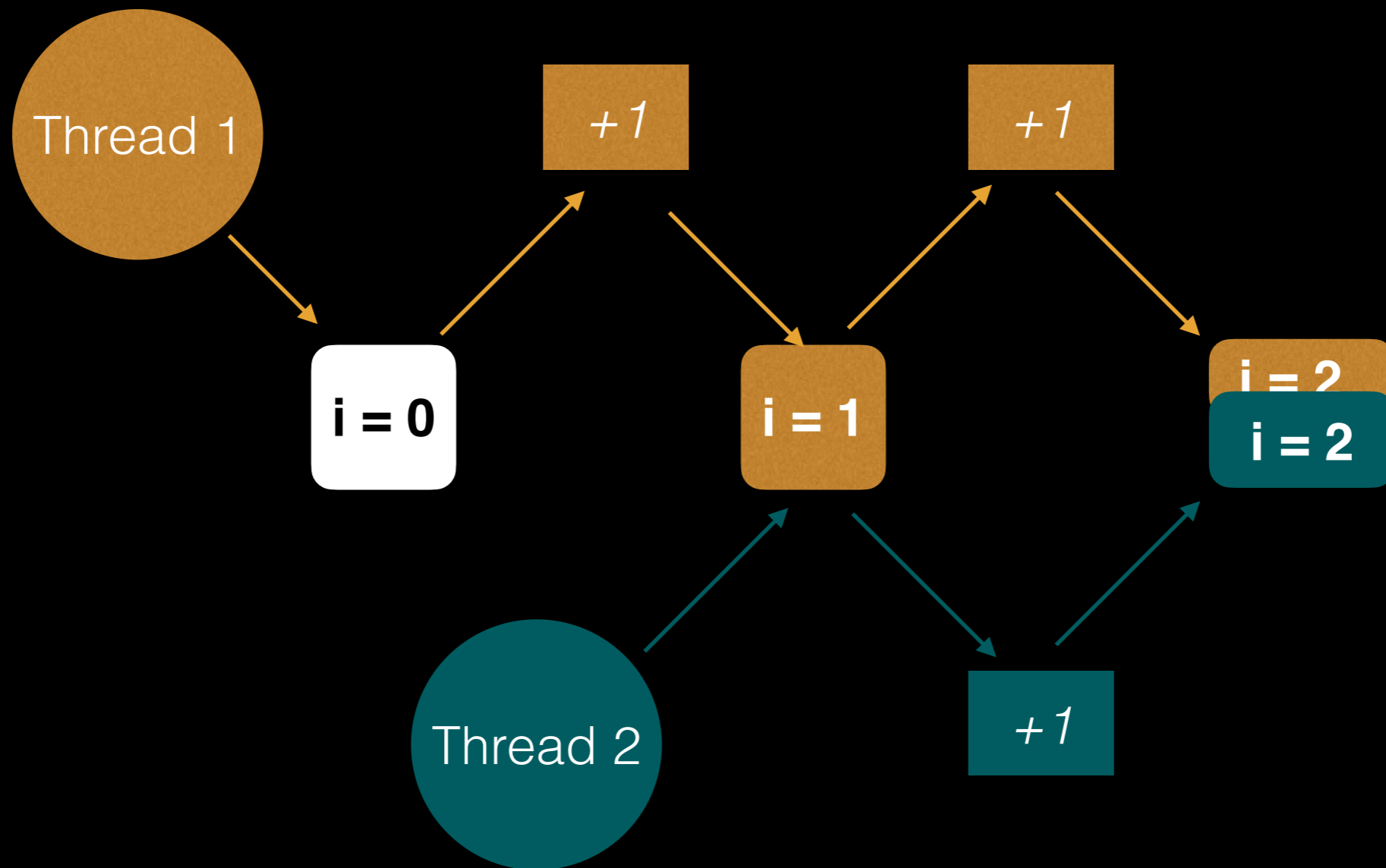


# Race Condition

# WTF is Race Condition?

- It's a logical vulnerability
- It's really easy to miss in a source code
- Can be anywhere, in web apps too
- Can be a critical vulnerability in some cases
- Easy to find and to reproduce

# How does it work?



# WEB and Race Condition:

Temp Files

# Race Condition in WEB #1

1. Receive file from user
2. Write file to temp dir
3. Read and check file from temp dir
4. Delete file, if check function return False
5. Print results

# Race Condition in WEB #1

1. Receive file from user

**2. Write file to temp dir**



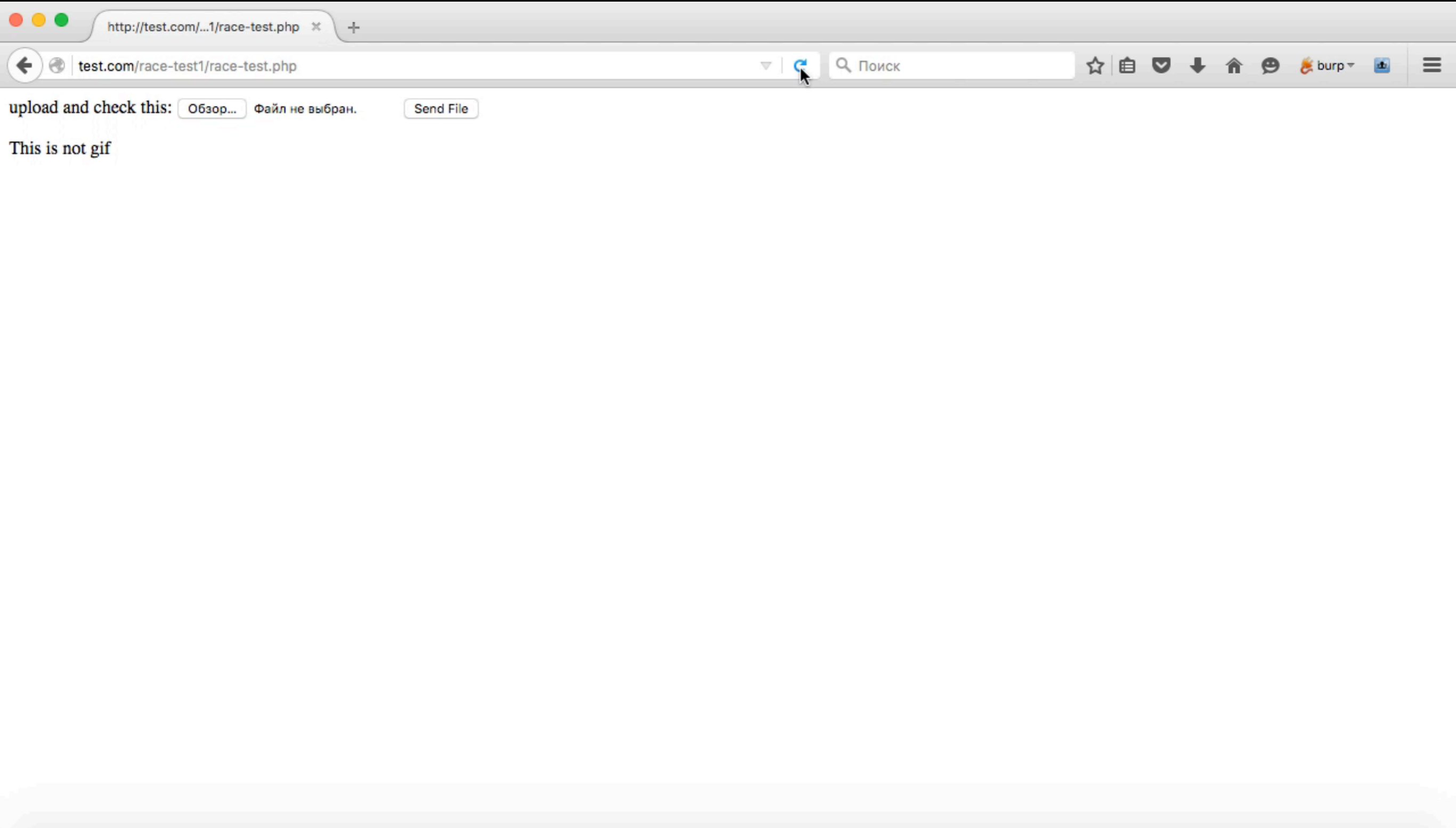
Any our file is on server's FS!

**3. Read and check file from temp dir**

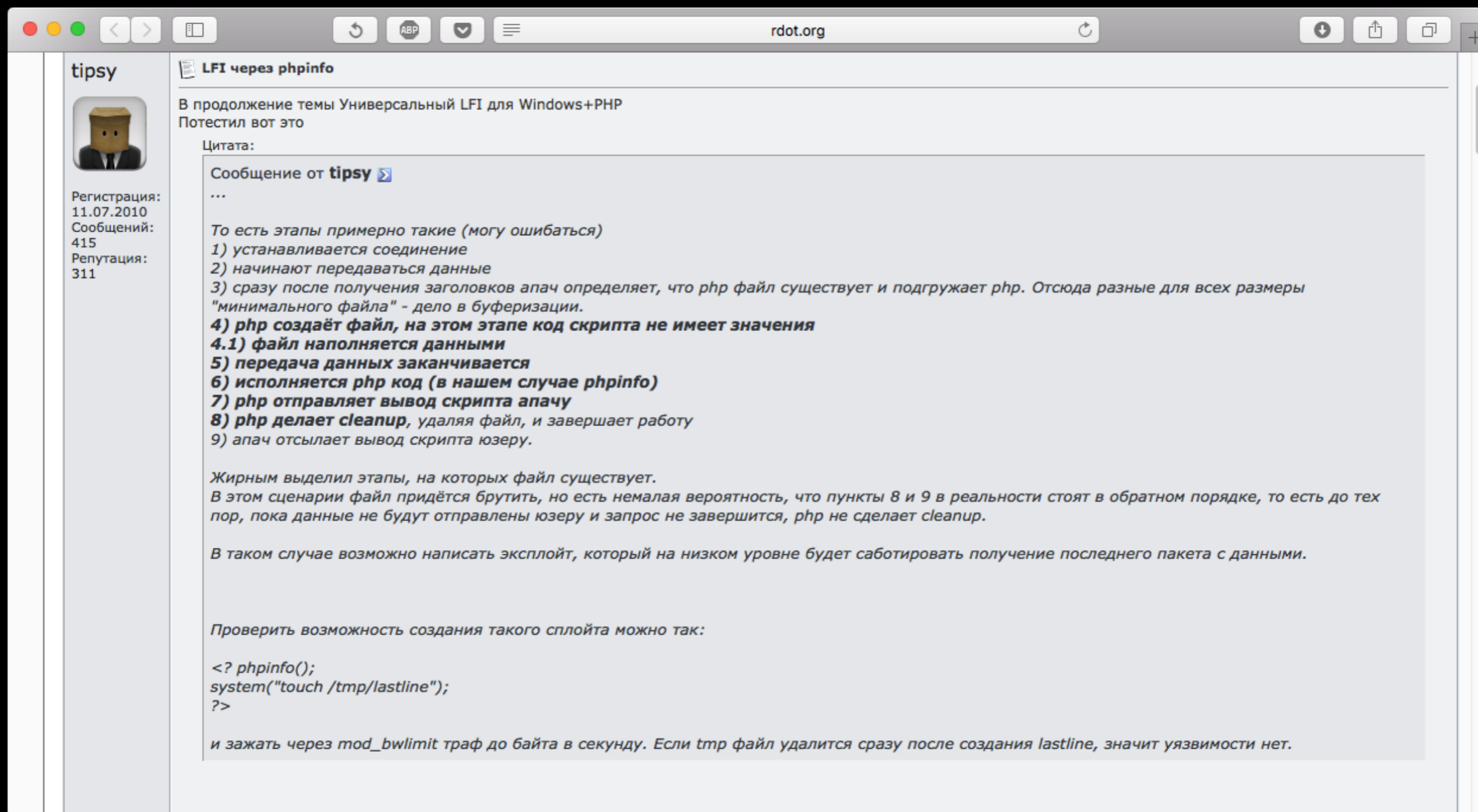
4. Delete file, if check function return False

5. Print results

# Race Condition in WEB #1



# Example from real life #1



The screenshot shows a forum post on the website rdot.org. The post is titled "LFI через phpinfo" and is a reply to a thread about "Универсальный LFI для Windows+PHP". The user "tipsy" has posted a detailed explanation of the LFI process, including a list of steps and a PHP code snippet for a proof-of-concept exploit.

tipsy

Регистрация: 11.07.2010  
Сообщений: 415  
Репутация: 311

**LFI через phpinfo**

В продолжение темы Универсальный LFI для Windows+PHP  
Потестил вот это

Цитата:

Сообщение от tipsy

...

*То есть этапы примерно такие (могу ошибаться)*

- 1) устанавливается соединение
- 2) начинают передаваться данные
- 3) сразу после получения заголовков апач определяет, что php файл существует и подгружает php. Отсюда разные для всех размеры "минимального файла" - дело в буферизации.
- 4) **php создаёт файл, на этом этапе код скрипта не имеет значения**
  - 4.1) файл наполняется данными
- 5) передача данных заканчивается
- 6) **исполняется php код (в нашем случае phpinfo)**
- 7) **php отправляет вывод скрипта апачу**
- 8) **php делает cleanup**, удаляя файл, и завершает работу
- 9) апач отправляет вывод скрипта юзеру.

*Жирным выделил этапы, на которых файл существует.  
В этом сценарии файл придётся брутить, но есть немалая вероятность, что пункты 8 и 9 в реальности стоят в обратном порядке, то есть до тех пор, пока данные не будут отправлены юзеру и запрос не завершится, php не сделает cleanup.*

*В таком случае возможно написать эксплойт, который на низком уровне будет саботировать получение последнего пакета с данными.*

*Проверить возможность создания такого сплойта можно так:*

```
<? phpinfo();  
system("touch /tmp/lastline");  
?>
```

*и зажать через mod\_bwlimit траф до байта в секунду. Если tmp файл удалится сразу после создания lastline, значит уязвимости нет.*



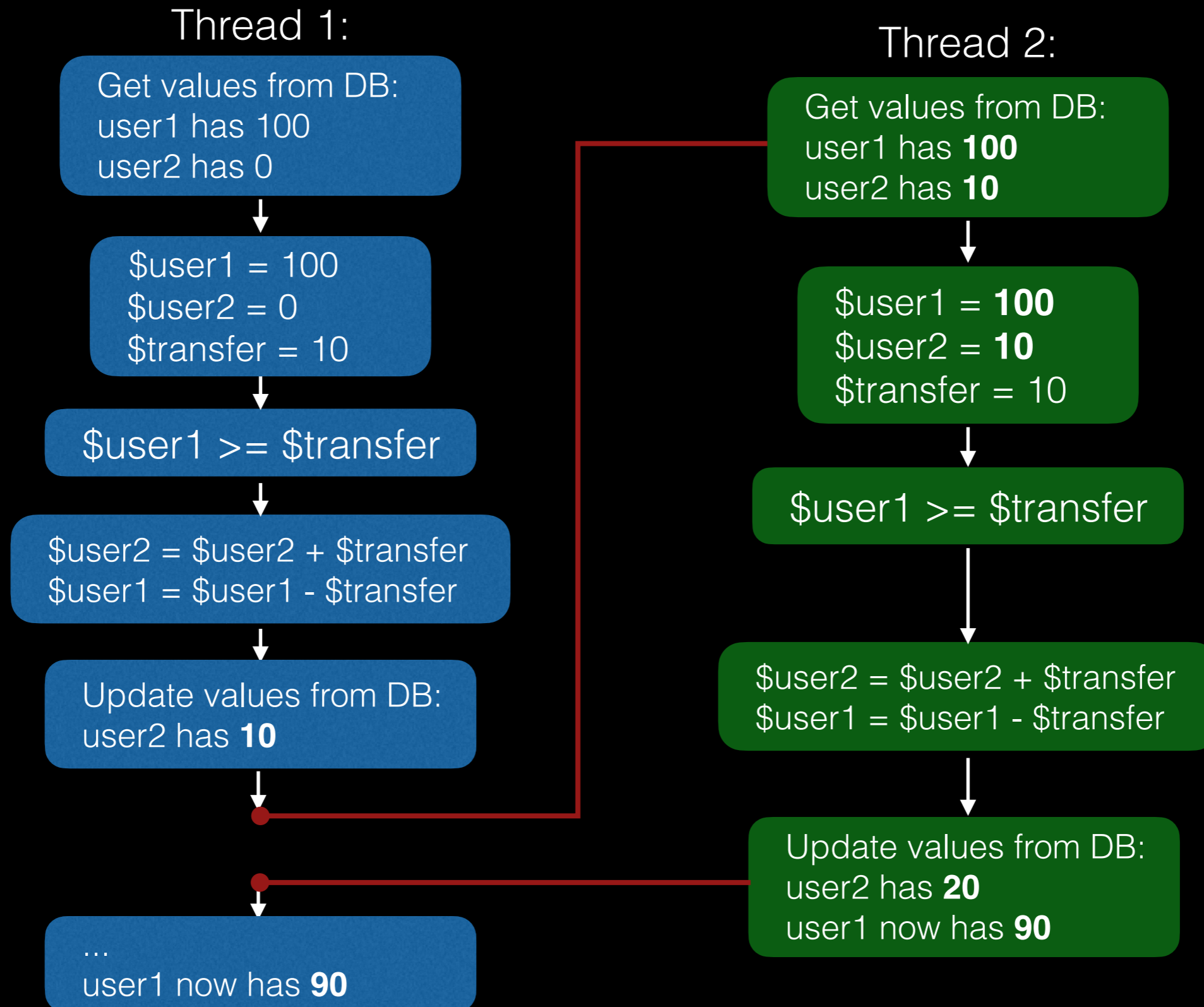
# WEB and Race Condition:

Money Transfer

# Race Condition in WEB #2

1. Get user1 balance
2. Get user2 balance
3. Check, if user has available money to transfer
4. Calculate new balances for accounts
5. Update balance at DB

# Race Condition in WEB #2



# Race Condition in WEB #2

Thread 1:

Get values from DB:  
user1 has 100  
user2 has 0

$\$user1 = 100$   
 $\$user2 = 0$   
 $\$transfer = 10$

$\$user1 \geq \$transfer$

$\$user2 = \$user2 + \$transfer$   
 $\$user1 = \$user1 - \$transfer$

Update values from DB:  
user2 has **10**

...  
user1 now has **90**

Thread 2:

Get values from DB:  
user1 has **100**  
user2 has **10**

$\$user1 = 100$   
 $\$user2 = 10$   
 $\$transfer = 10$

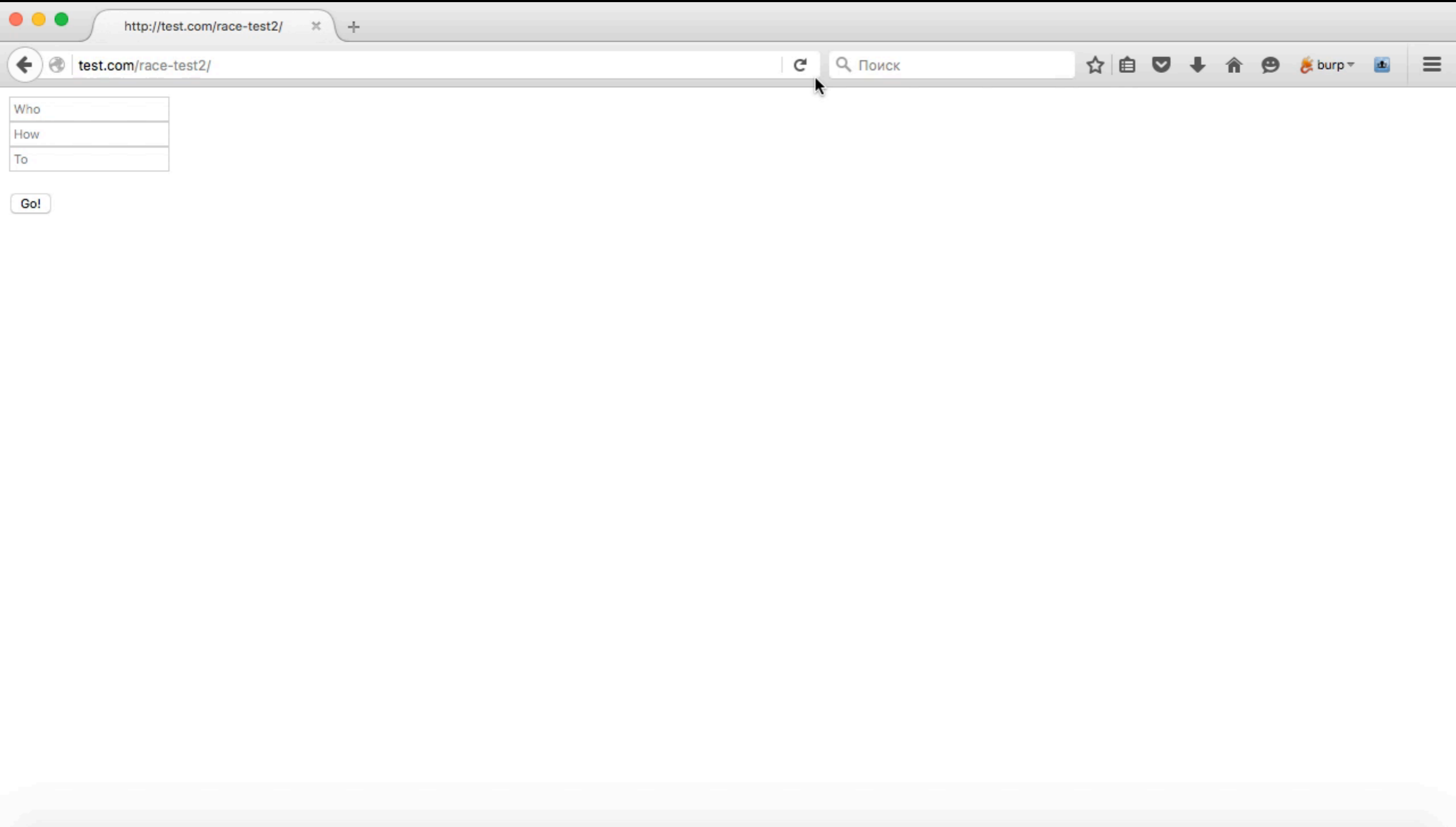
$\$user1 \geq \$transfer$

$\$user2 = \$user2 + \$transfer$   
 $\$user1 = \$user1 - \$transfer$

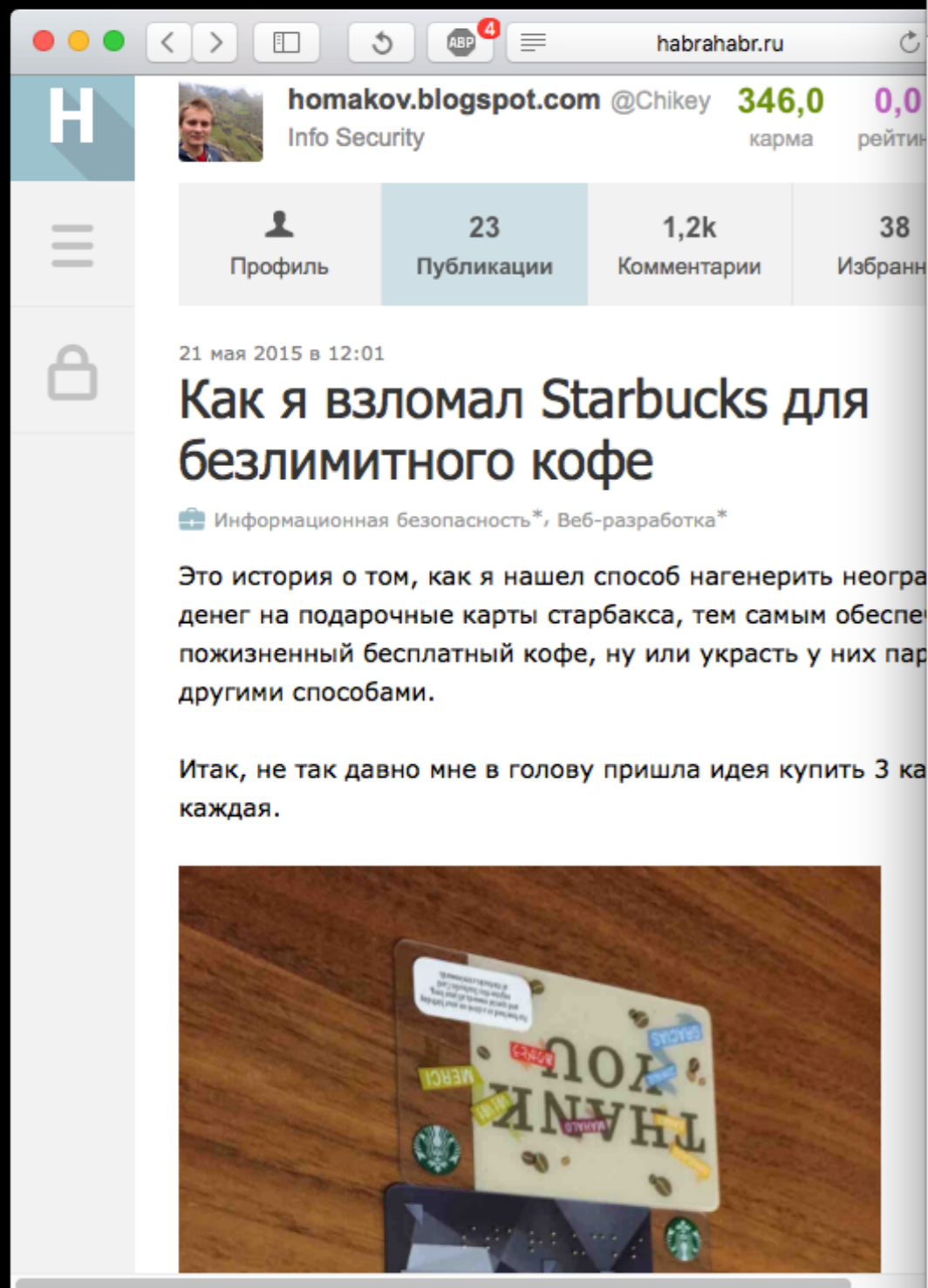
Update values from DB:  
user2 has **20**  
user1 now has **90**

Result:  
user1 - **90**  
user2 - **20**

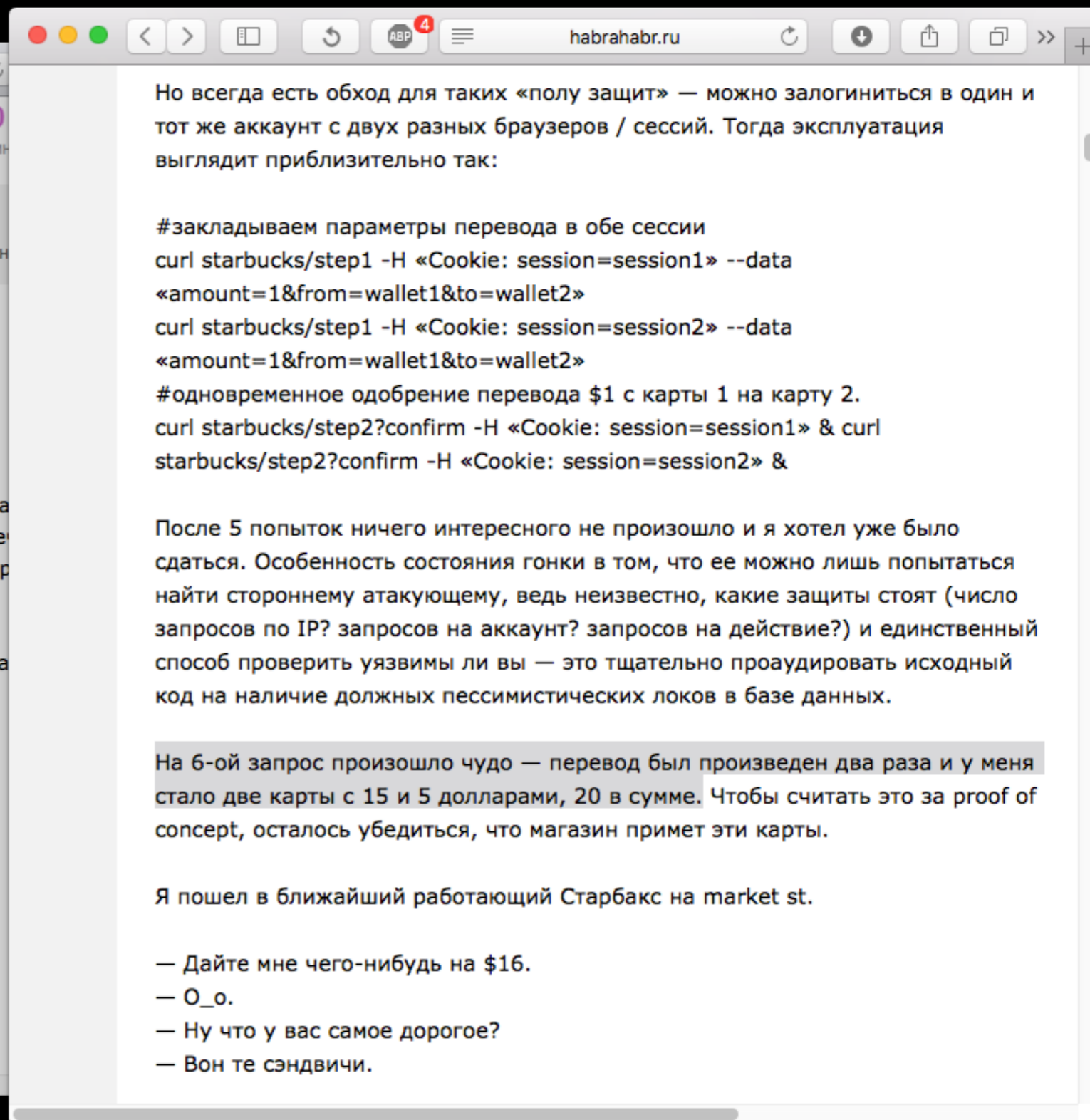
# Race Condition in WEB #2



# Example from real life #2



The screenshot shows a Habr article by user homakov.blogspot.com (@Chikey) with 346,0 karma and 0,0 rating. The article is titled "Как я взломал Starbucks для безлимитного кофе" and is categorized under "Информационная безопасность\*" and "Веб-разработка\*". The article text begins with "Это история о том, как я нашел способ нагенерить неограниченно денег на подарочные карты старбакса..." and includes a photo of a Starbucks gift card with a "THANK YOU" sticker.



The screenshot shows a browser window with the URL habrahabr.ru. The article text is as follows:

Но всегда есть обход для таких «полу защит» — можно залогиниться в один и тот же аккаунт с двух разных браузеров / сессий. Тогда эксплуатация выглядит приблизительно так:

```
#закладываем параметры перевода в обе сессии
curl starbucks/step1 -H «Cookie: session=session1» --data «amount=1&from=wallet1&to=wallet2»
curl starbucks/step1 -H «Cookie: session=session2» --data «amount=1&from=wallet1&to=wallet2»
#одновременное одобрение перевода $1 с карты 1 на карту 2.
curl starbucks/step2?confirm -H «Cookie: session=session1» & curl starbucks/step2?confirm -H «Cookie: session=session2» &
```

После 5 попыток ничего интересного не произошло и я хотел уже было сдаться. Особенность состояния гонки в том, что ее можно лишь попытаться найти стороннему атакующему, ведь неизвестно, какие защиты стоят (число запросов по IP? запросов на аккаунт? запросов на действие?) и единственный способ проверить уязвимы ли вы — это тщательно проаудировать исходный код на наличие должных пессимистических локов в базе данных.

На 6-ой запрос произошло чудо — перевод был произведен два раза и у меня стало две карты с 15 и 5 долларами, 20 в сумме. Чтобы считать это за proof of concept, осталось убедиться, что магазин примет эти карты.

Я пошел в ближайший работающий Старбакс на market st.

- Дайте мне чего-нибудь на \$16.
- О\_о.
- Ну что у вас самое дорогое?
- Вон те сэндвичи.

# Conclusion

- It's not a bug, it is a **security vulnerability!**
- Make your multithreading application safe
- Lock or sync vars, files, etc to avoid it
- **Never forget about Race Condition!**