**Chapter 11**



**Attacking Application Logic:**

**What is “logic” in a web application?**

Every web application (like a banking site, shopping website, or college portal) works based on logic. Logic means rules and decisions that tell the application *what to do* in different situations.

Web applications rely heavily on logic, which converts human requirements into small executable steps. Logic flaws are hard to detect, often missed by automated tools, and overlooked compared to common vulnerabilities like SQL injection. Due to their uniqueness and subtlety, logic flaws are highly valuable targets for attackers.

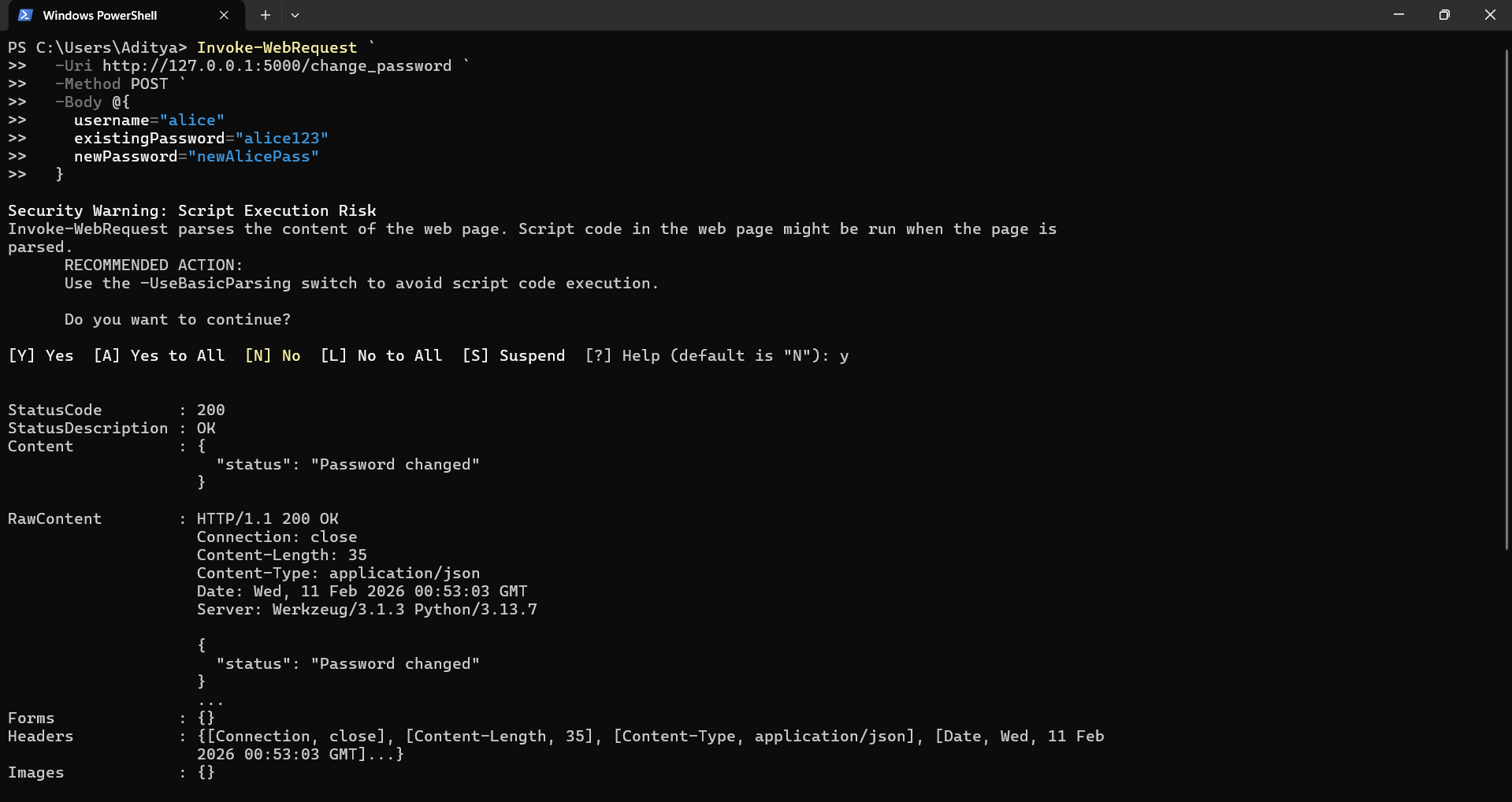
**The Nature of Logic Flaws:**

Logic flaws are errors in application reasoning caused by faulty or incomplete assumptions made by developers. They have no fixed patterns, making them difficult for automated tools and standard testing to detect. Due to their diversity and subtlety, logic flaws remain a long-term and valuable target for attackers.

**Real-World Logic Flaws:**

**Example1: Fooling a Password Change Function**

Step1: (Goal: Normal password change)

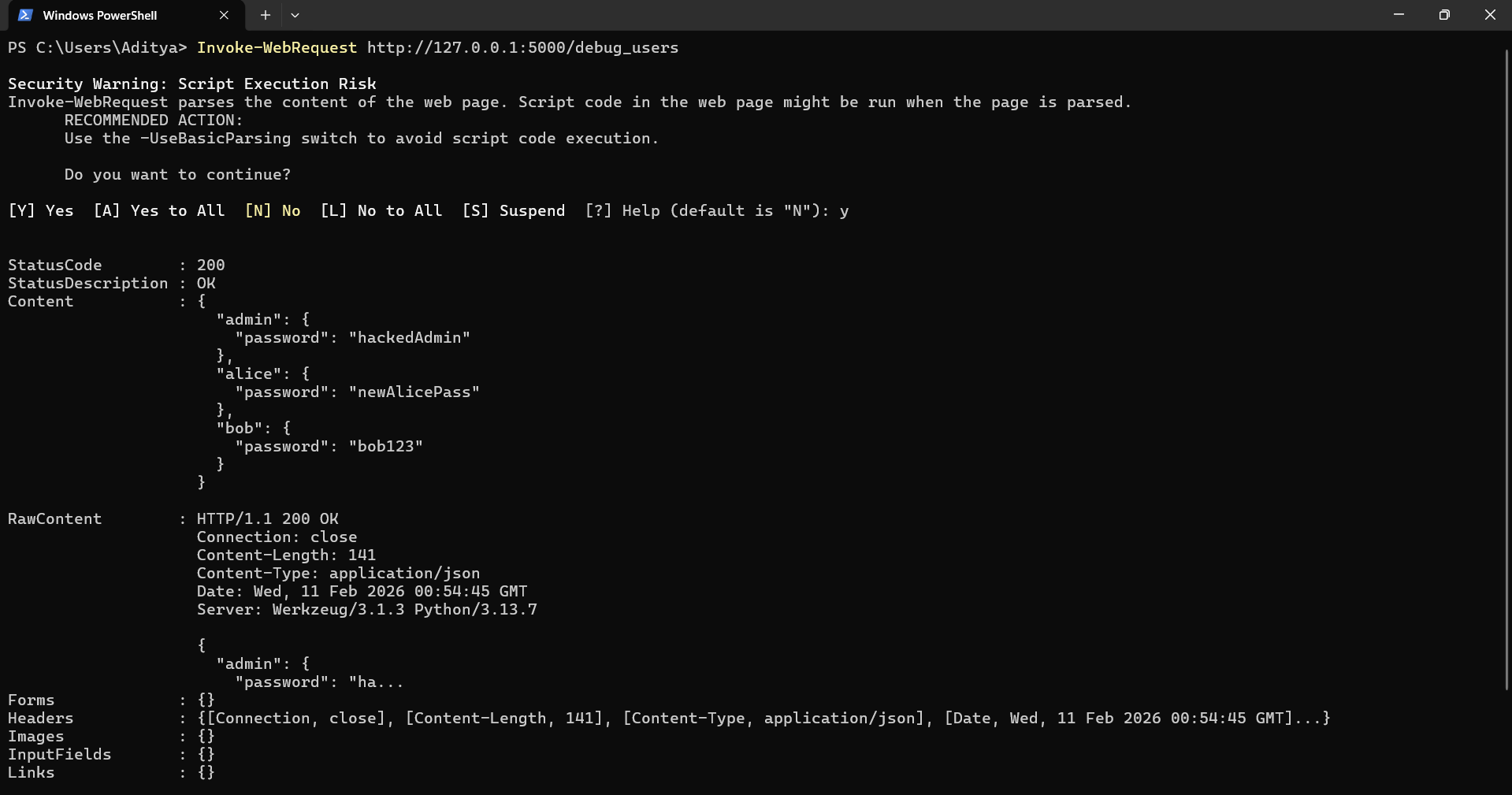


Step2: (Goal: The logic flaw attack)



This exploits the logic flaw. No existingPassword parameter at all.

Step3: (Goal: Verify if the attack worked)



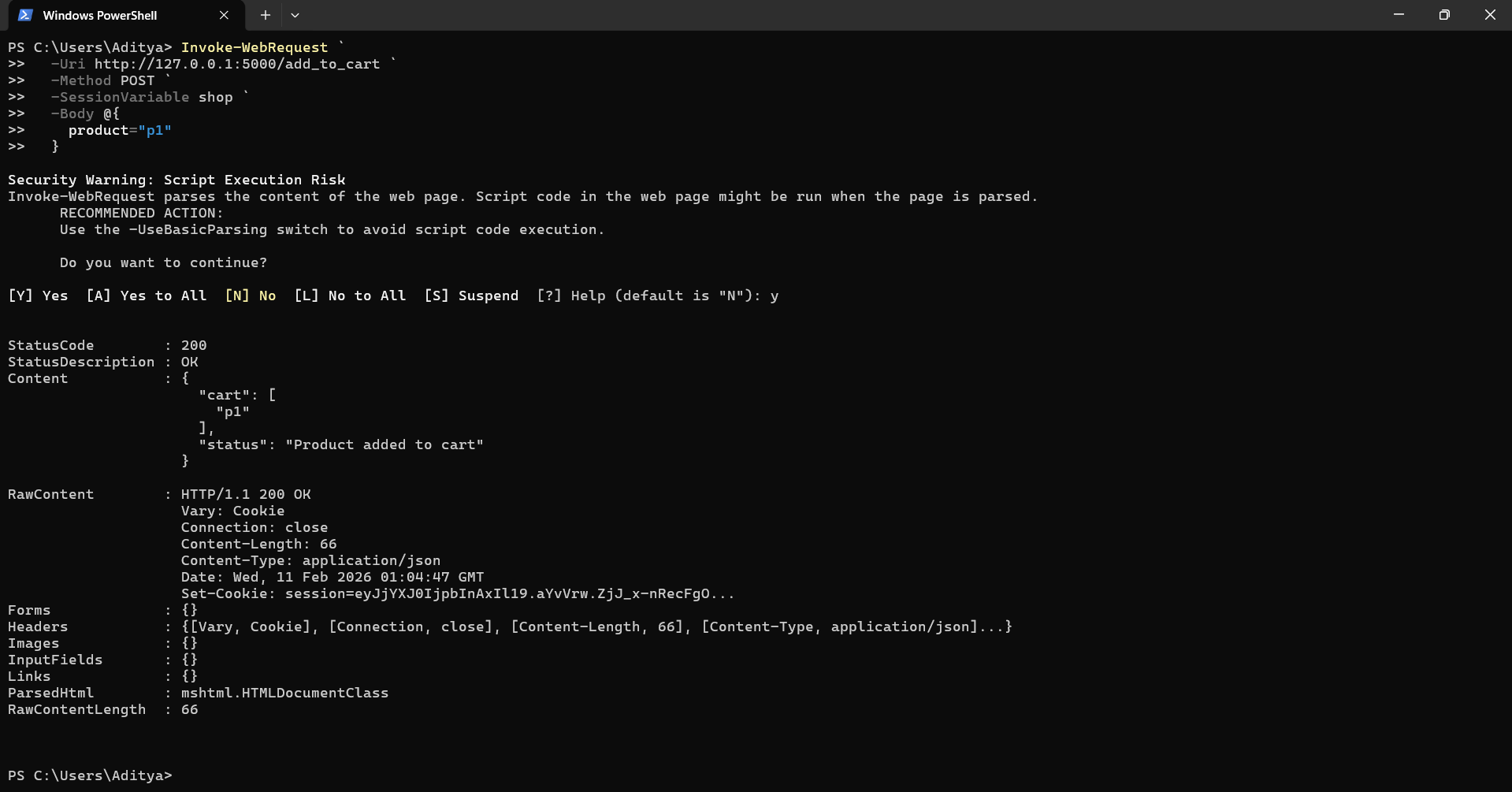
In case it is secured then the following output would have came:



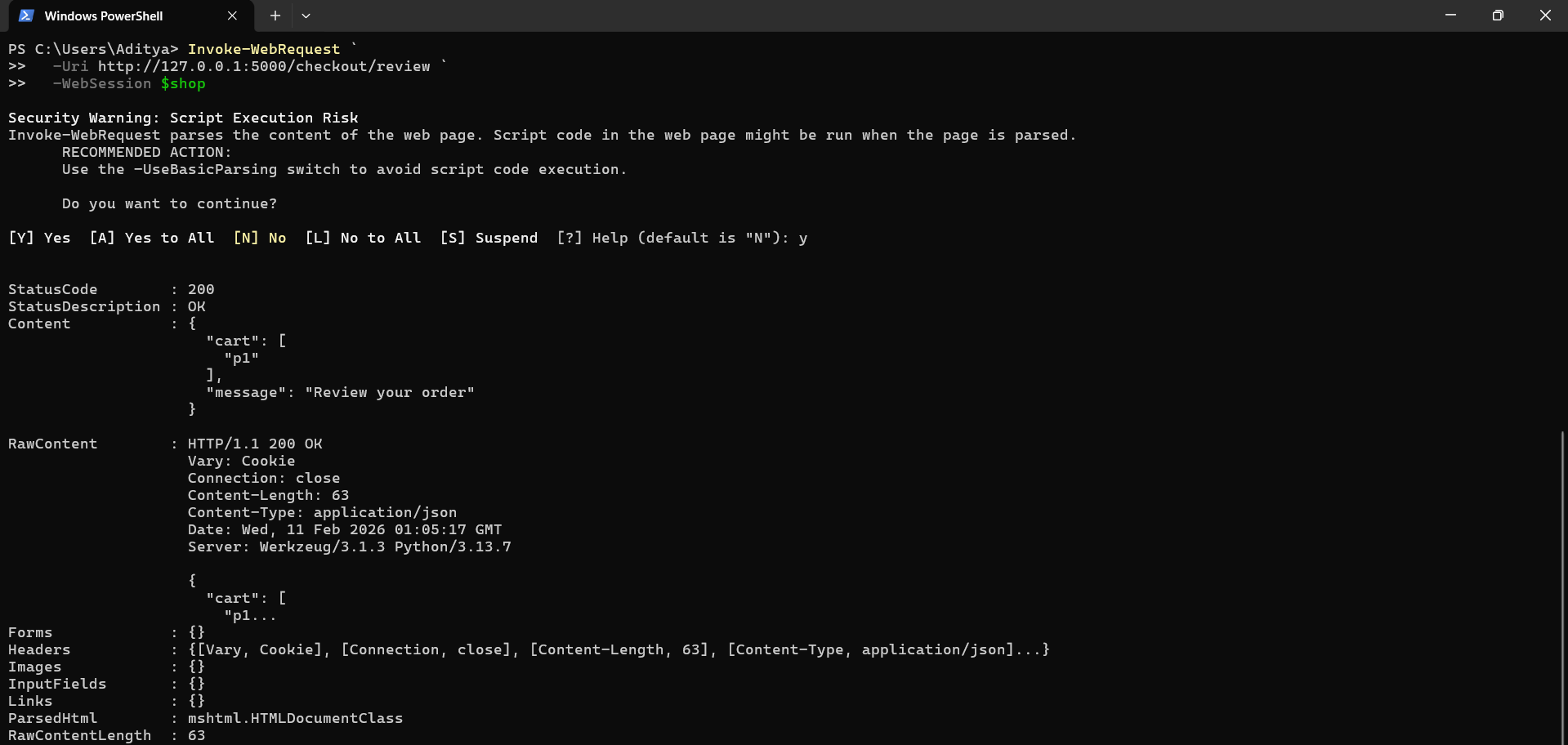
**Example2: Proceeding to Checkout**

Step1: (Goal: Check the normal behaviour)

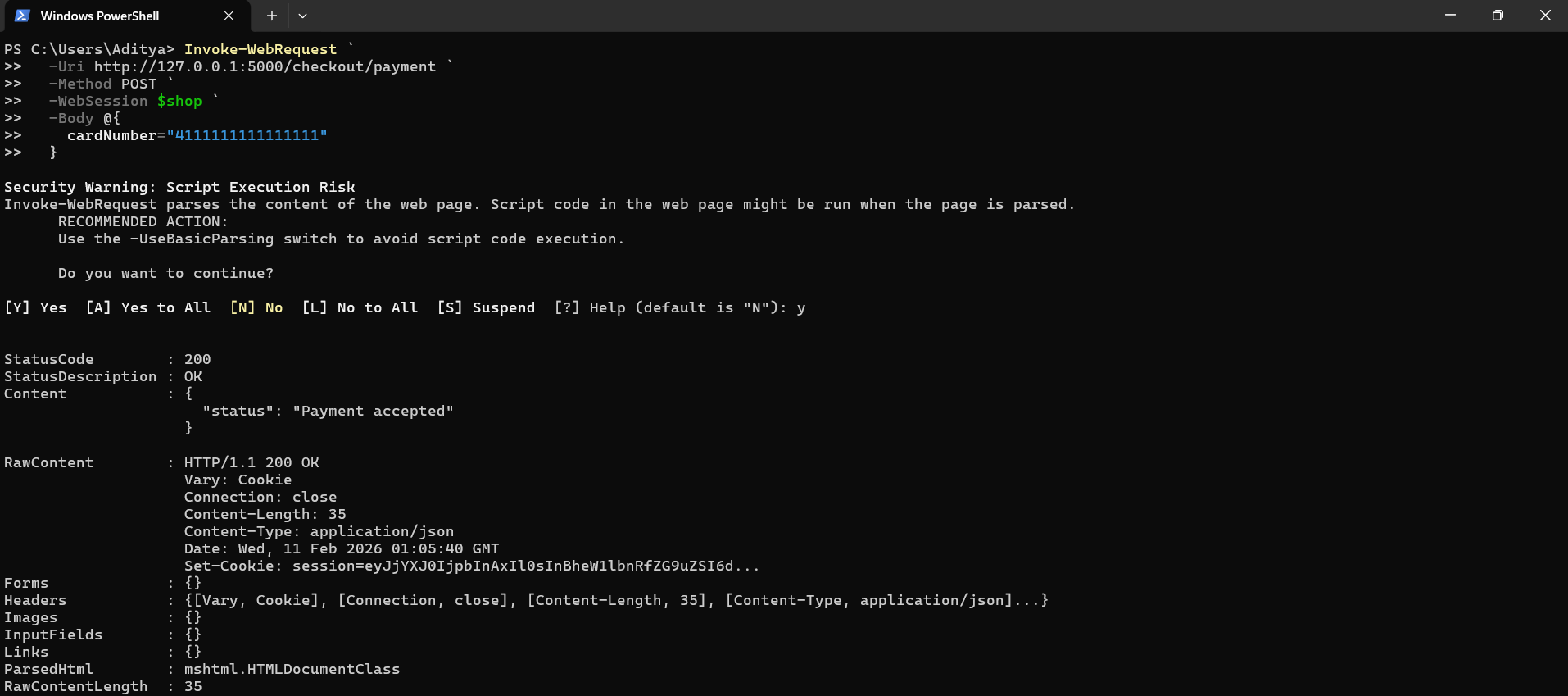
Add item to cart:



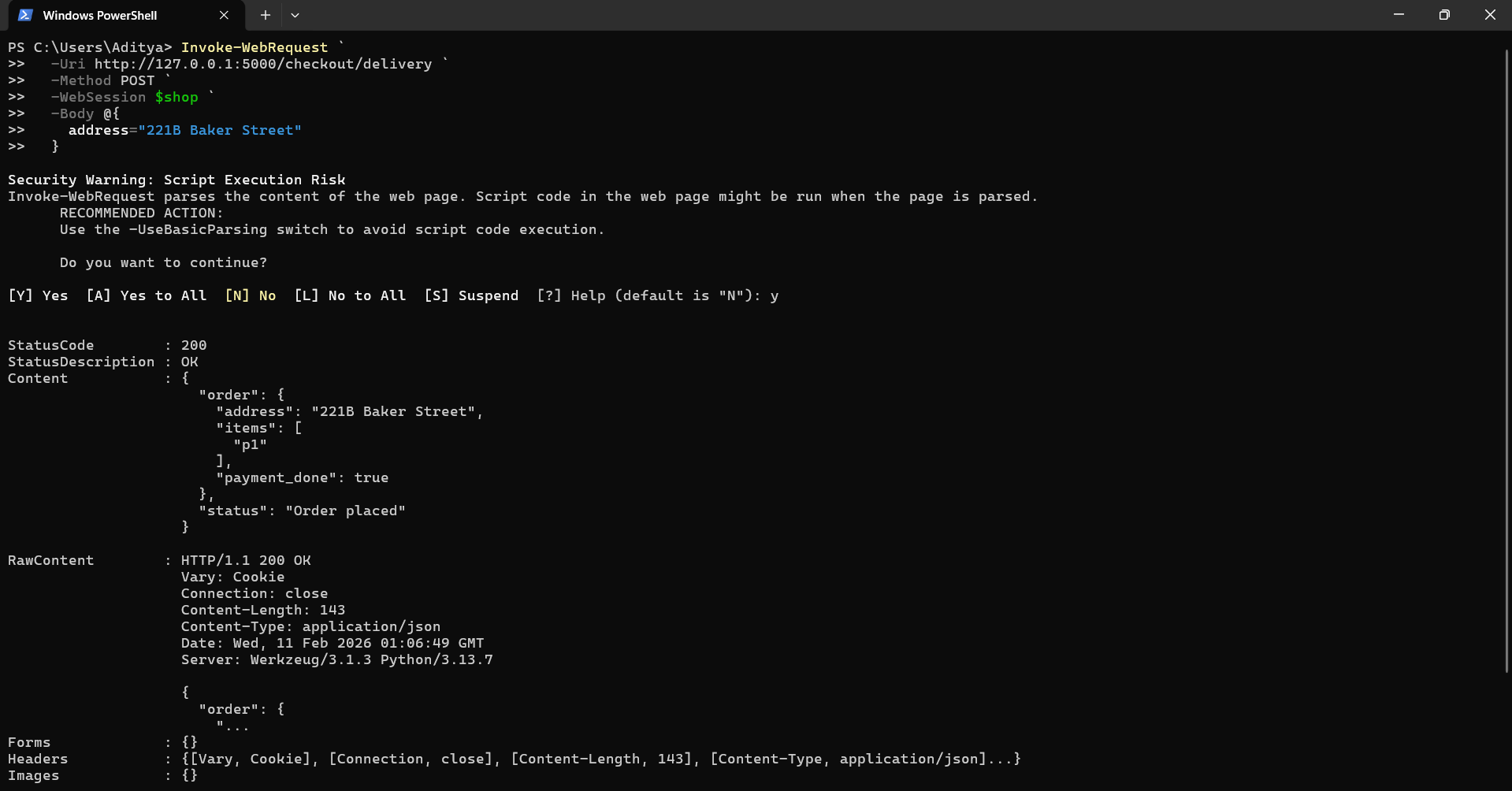
Review cart:



Payment (legitimate):

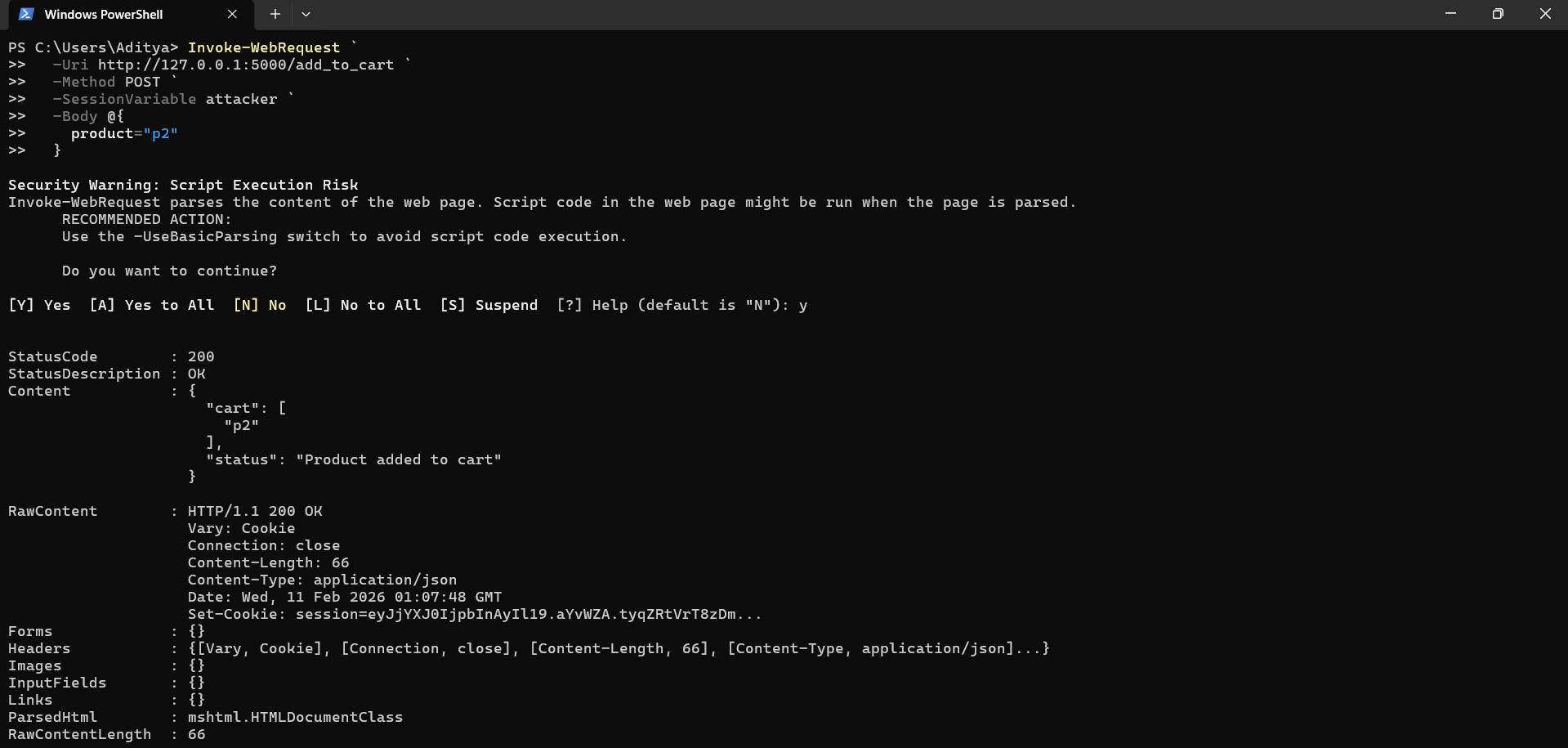


Delivery (legitimate):

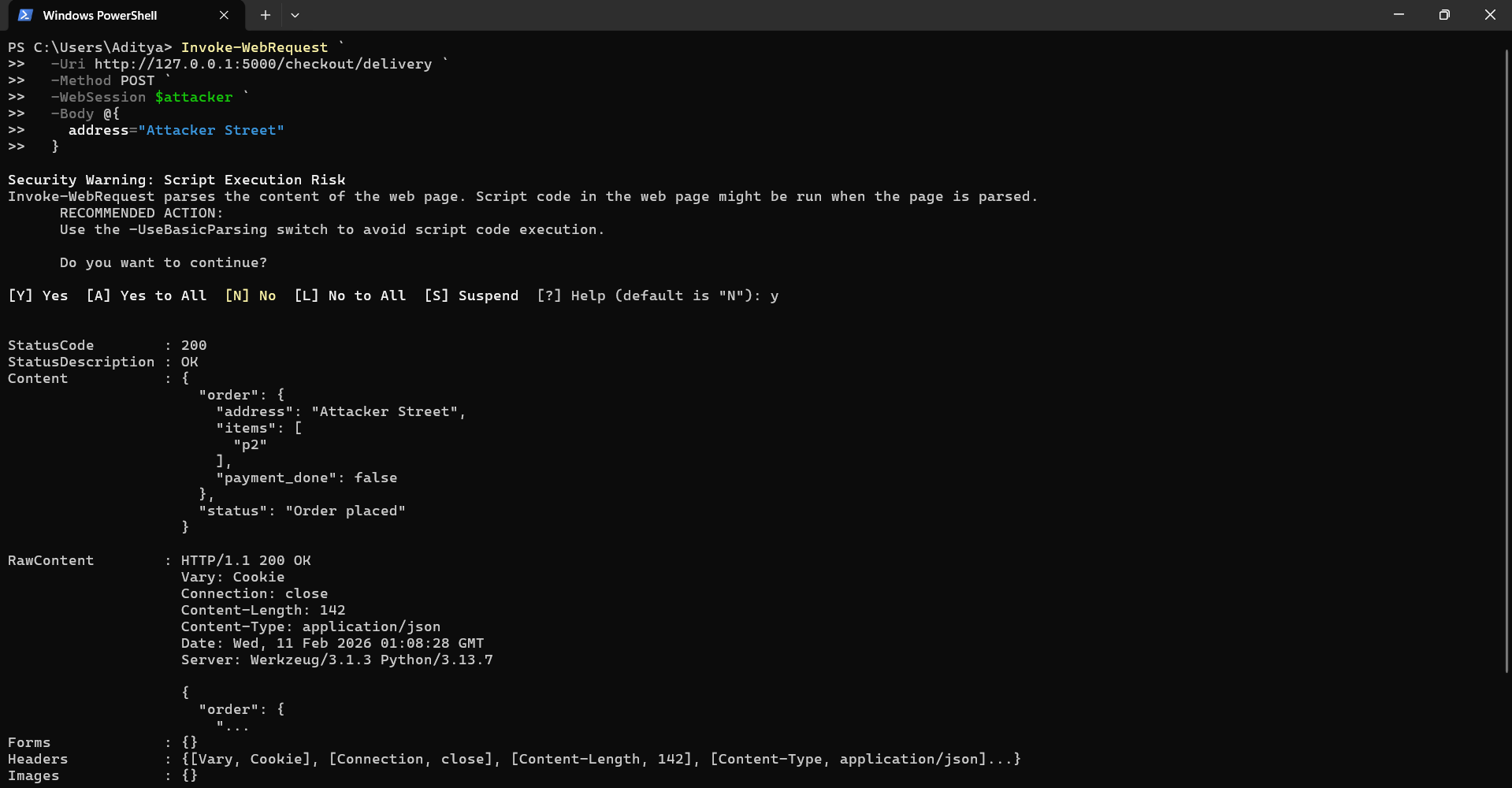


Step2: (Goal: Forced browsing)

Add item to cart (same as normal):

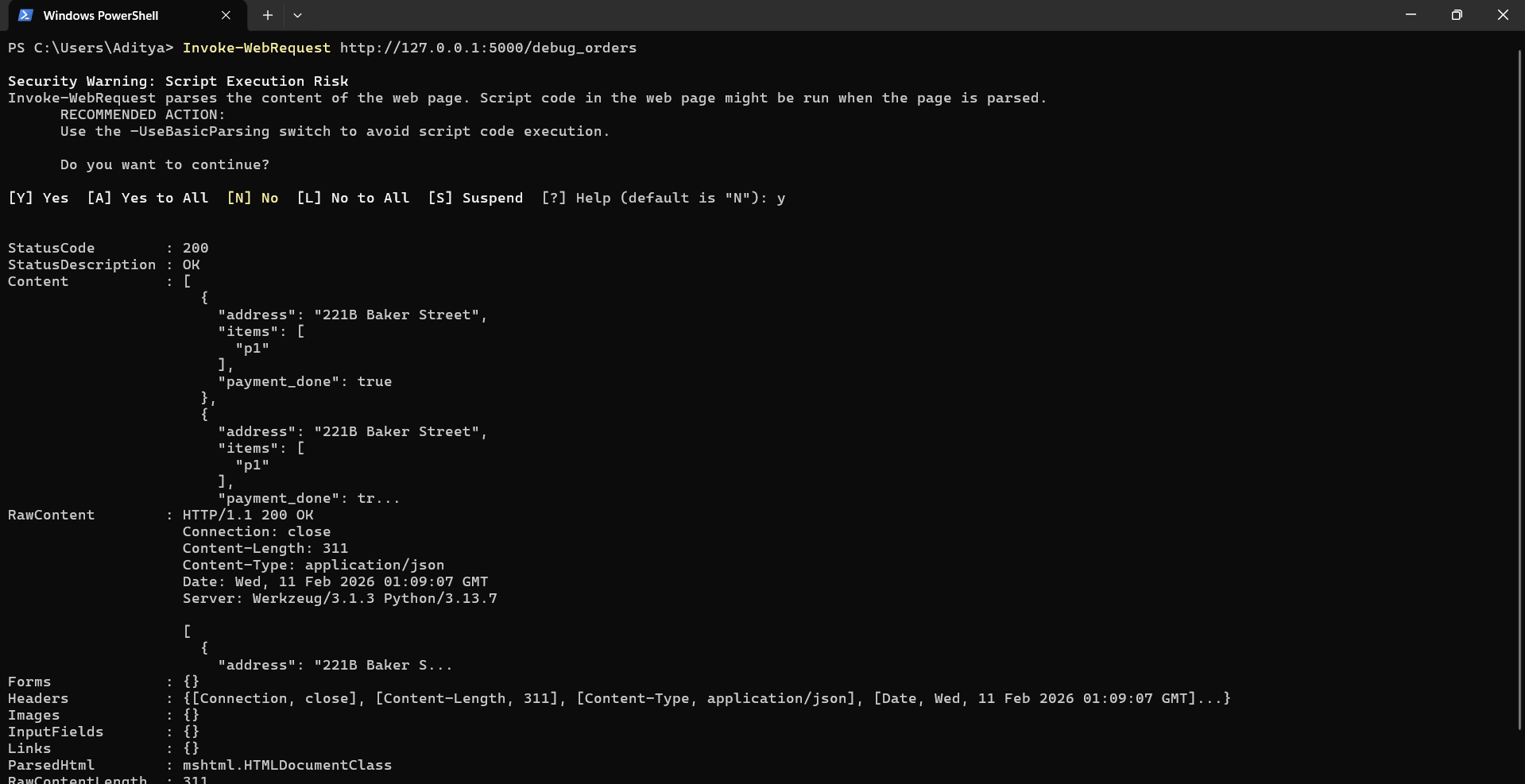


Jump straight to delivery:



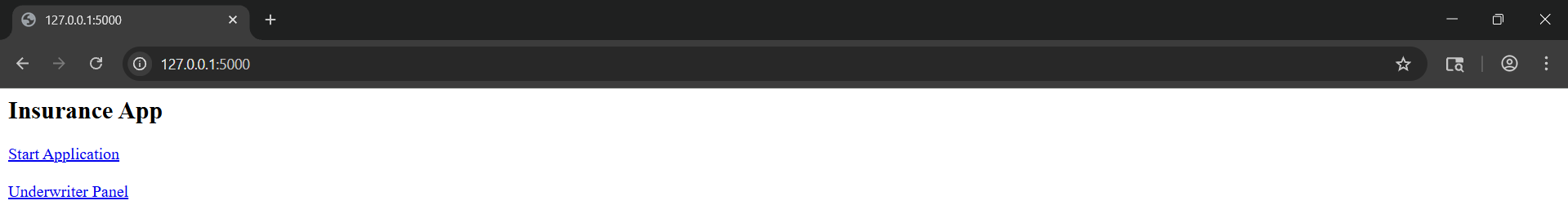
Clearly, Order placed WITHOUT payment.

Verify the Exploit:

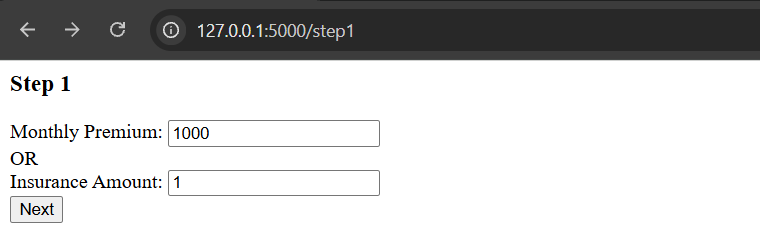


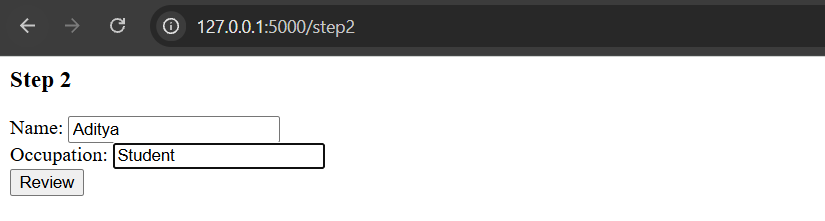
**Example3: Rolling Your Own Insurance**

Step1: Open the Burp Suite, and in the browser open the website as shown below:

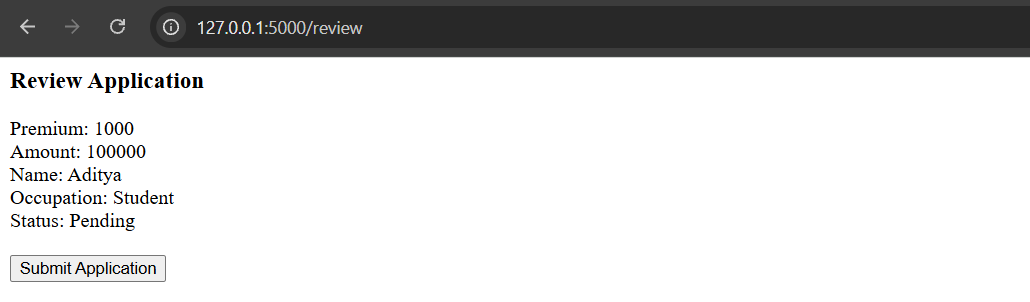


Step2: Enter the amount in normal way, and proceed normally:



Then, 

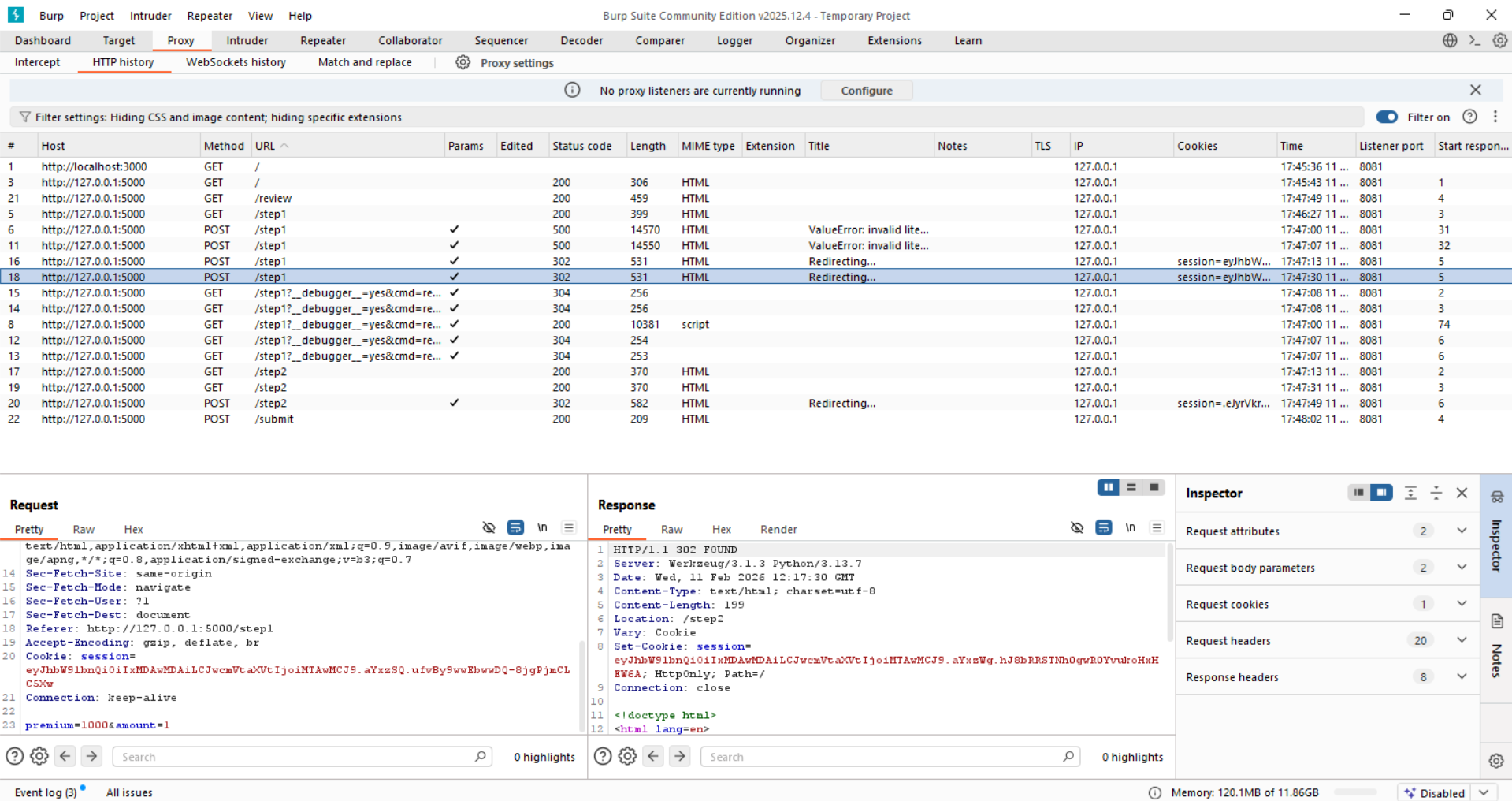
Then,



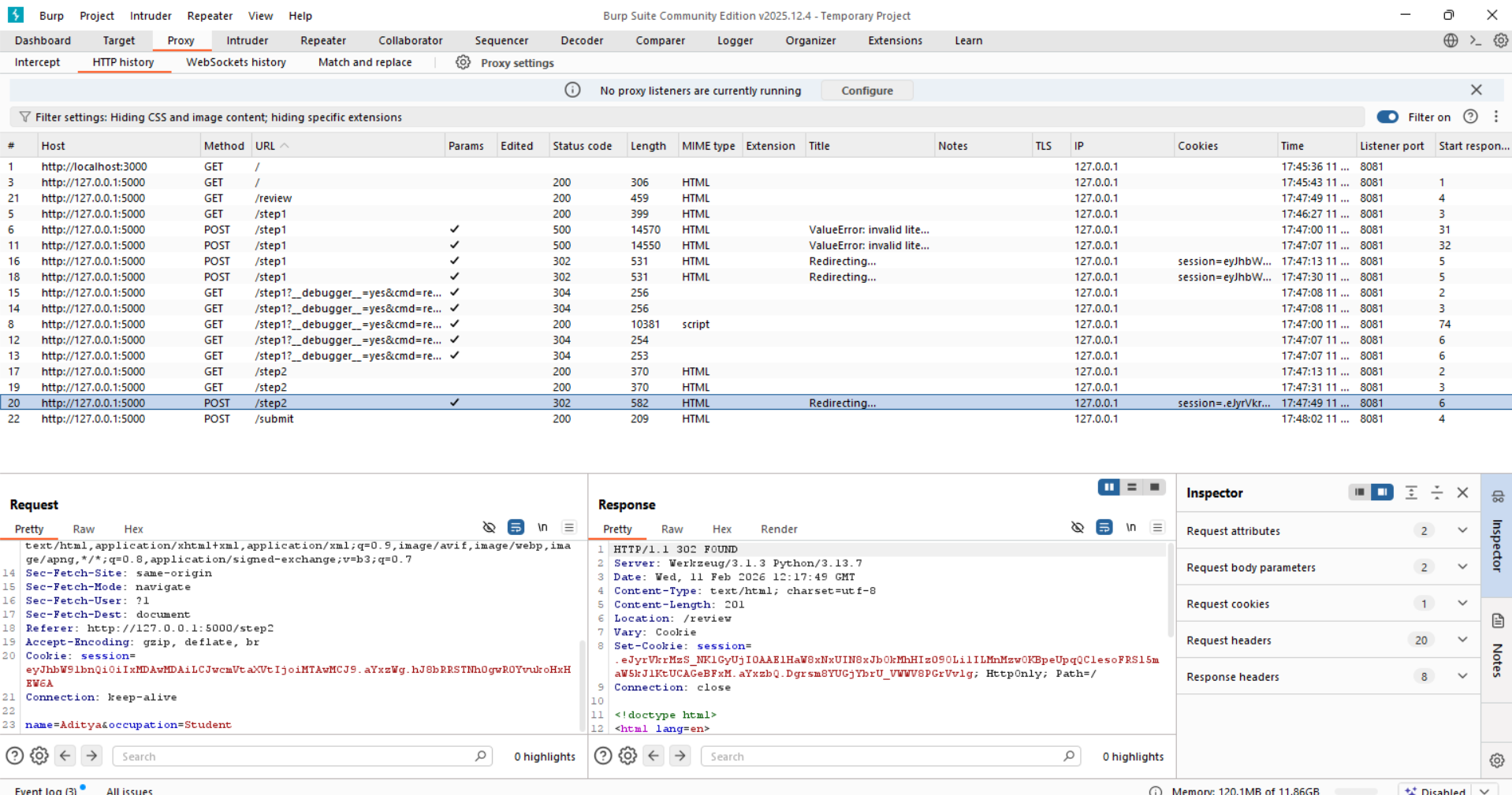
Then,



Observe the burp:

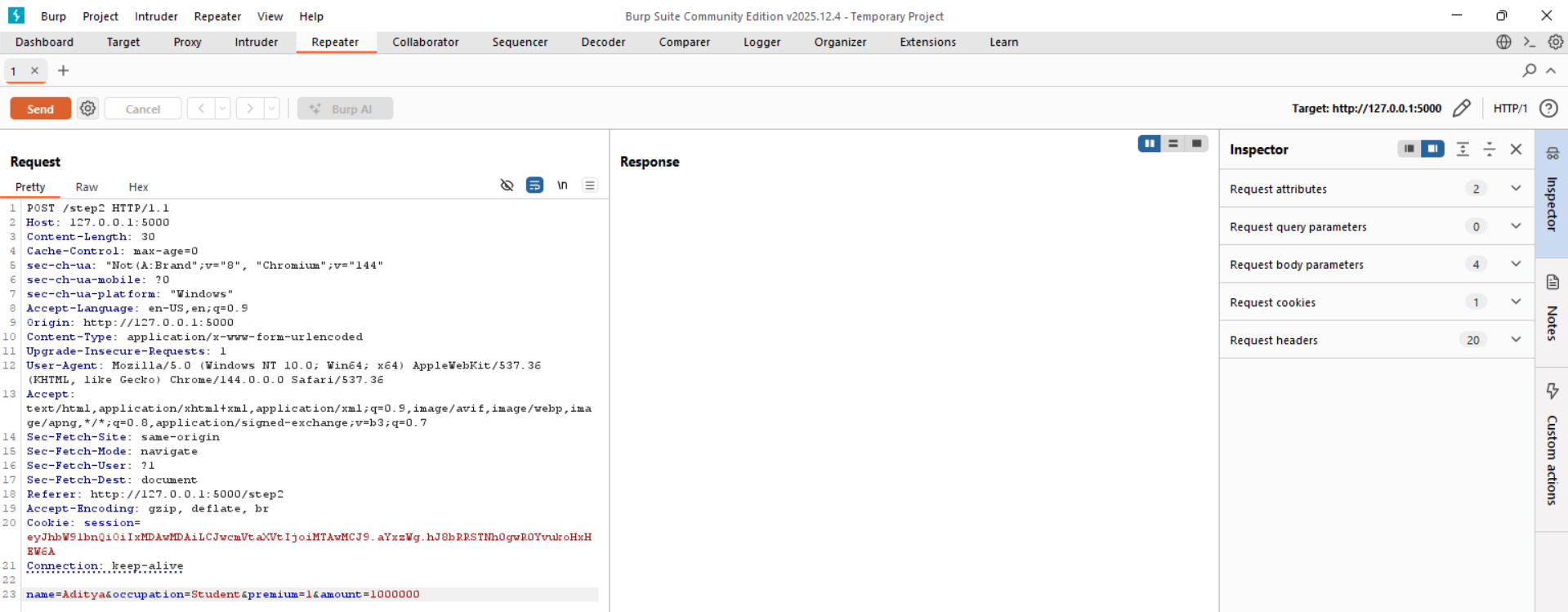


See the /step1. And the /step2:



Step3: (Goal: Change Price After Step 1)

For this, forward the request to repeater:



**Example4: Breaking the Bank**

Browser:



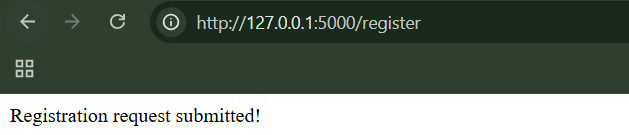
Step1: (Goal: Login as normal user). Alice:alice123



Now, click on “Register Another Account” and register the other account:



Following confirmation comes:

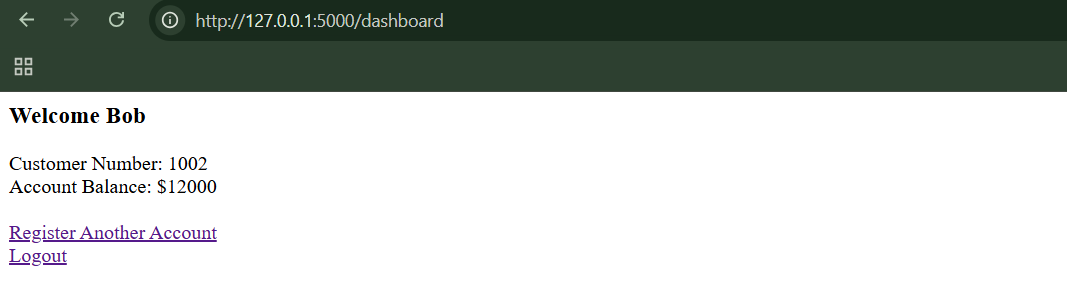


The app did: session["customer"] = create\_customer\_object(data, cust\_number)

It OVERWROTE our authenticated identity.

Step2: (Goal: Over writing the registration)

In the new tab, go to the ‘/dashboard’ :



We logged in as Alice… But now we're inside Bob’s account.

Why This Is Dangerous? The app reused the same object: session["customer"]

For:

* Authentication
* Registration
* Identity storage

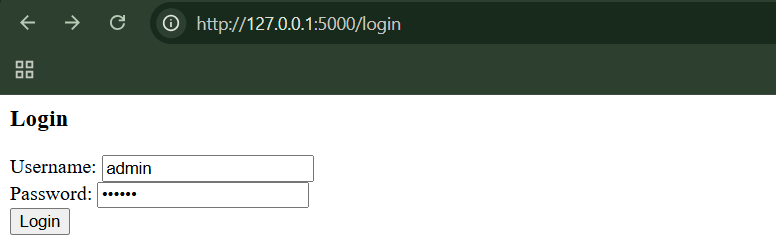
Registration overwrote it.

**Example5: Erasing an Audit Trail**

Browser:



Step1: (Goal: Login as the normal user)

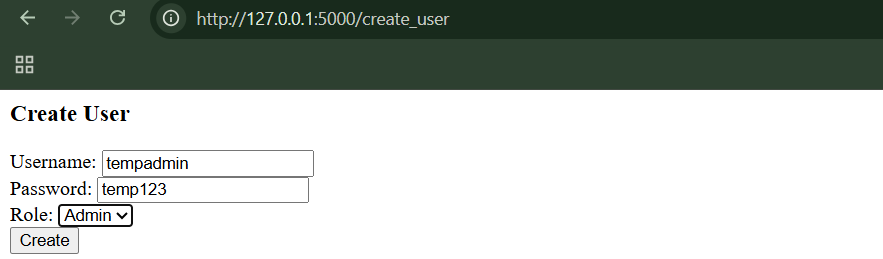


Following dashboard shall appear:  


Step2: (Goal: To exploit) Create a fake admin. Go to the “create user” link:



Fill it like this, and click on “Create” button, and then logout:

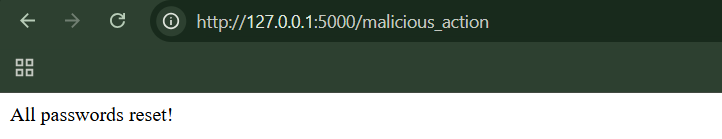


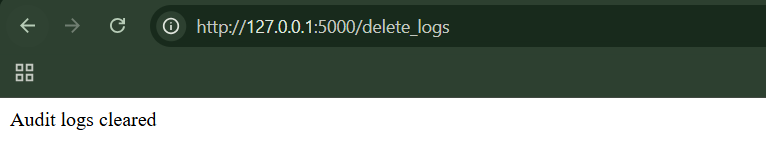
Now, login as the tempadmin:



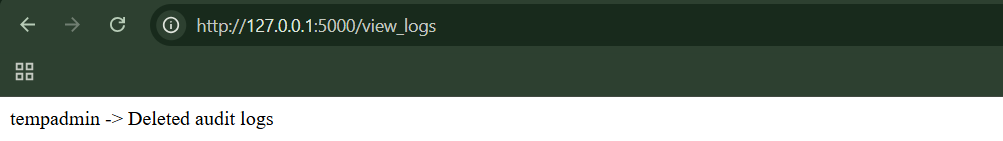
Step3: (Goal: Malicious activity)

Click on : Reset all passwords. Now every user password = hacked



Click on: Delete Audit Logs. 

Now, Click on View logs:



Everything else is gone.

What Happened?

The system assumed: If someone deletes logs, the deletion itself will be logged.

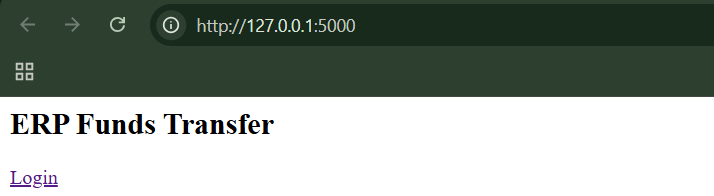
But:

1. You created a second admin.
2. Used it for attack.
3. Deleted logs.
4. Only entry left points to fake account.

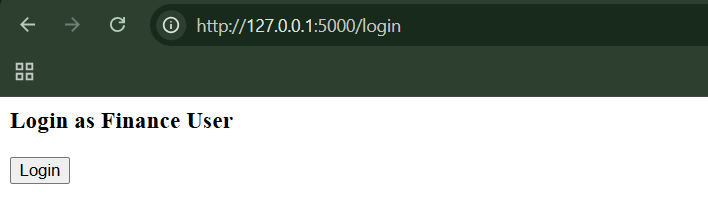
There is no evidence linking original admin. Perfect crime.

**Example6: Beating a Business Limit**

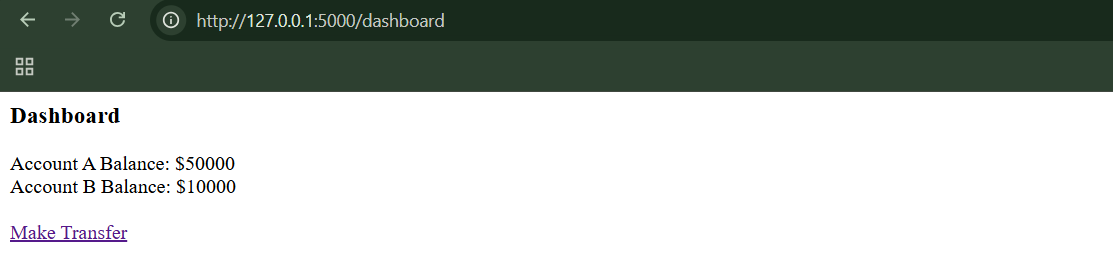
Browser:



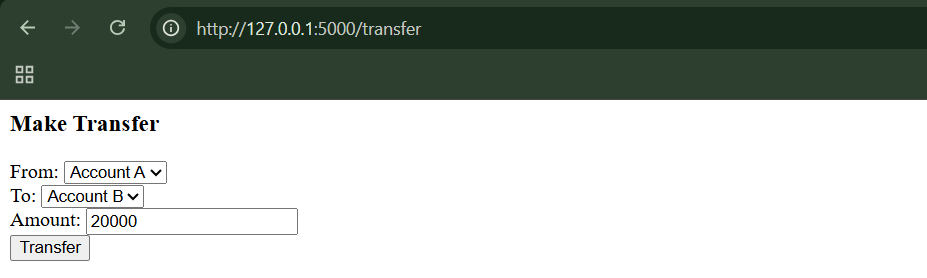
When clicked on “Login”:



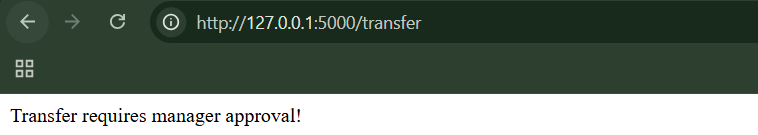
Again,



Step1: (Goal: Normal test)

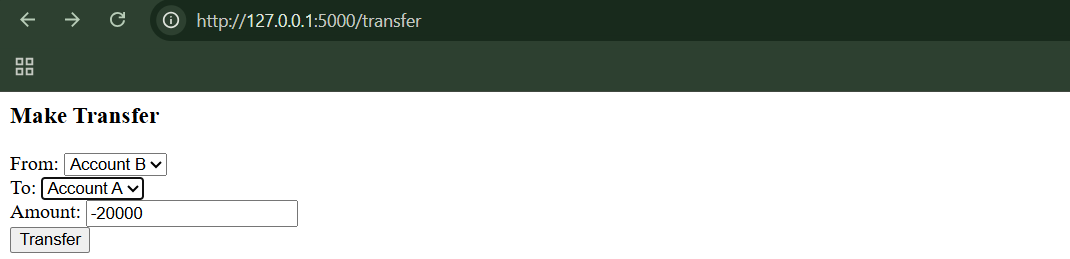


When clicked on “Transfer” button:

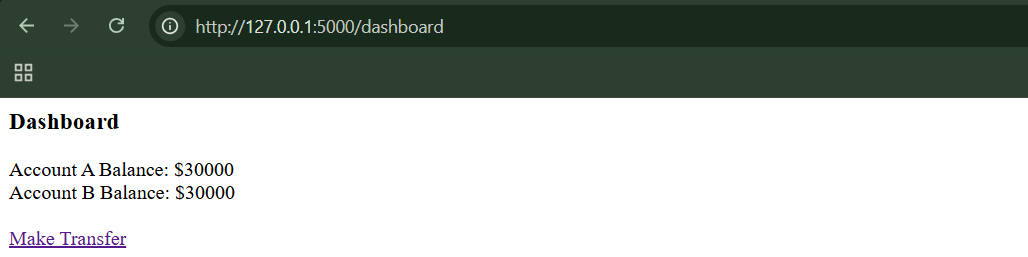
****

Clearly, Protection works for positive numbers.

Step2: (Goal: bypassing)



When did:



Transfer 20,000 from A → B. WITHOUT approval.

Balances become:

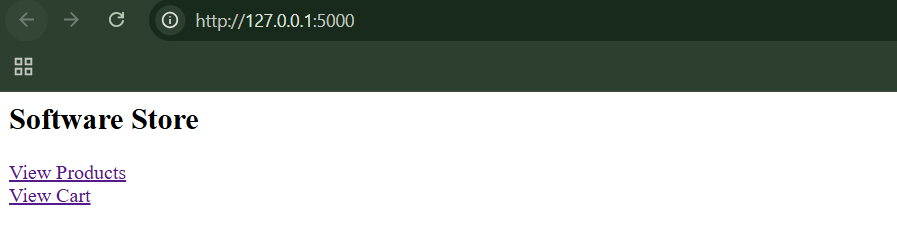
Account A: 30000

Account B: 30000

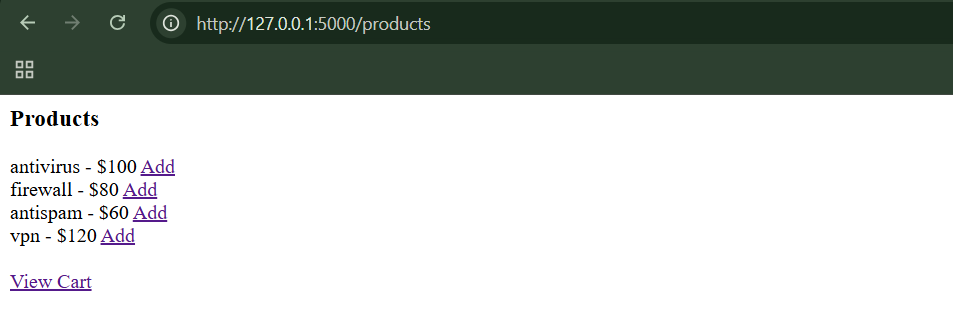
We bypassed anti-fraud protection.

**Example7: Cheating on Bulk Discounts**

Browser:



Step1: (Goal: Normal behaviour)



Add the first three, then visit the View Cart:



Clearly, Discount applied.

Step2: (Goal: To cheat)

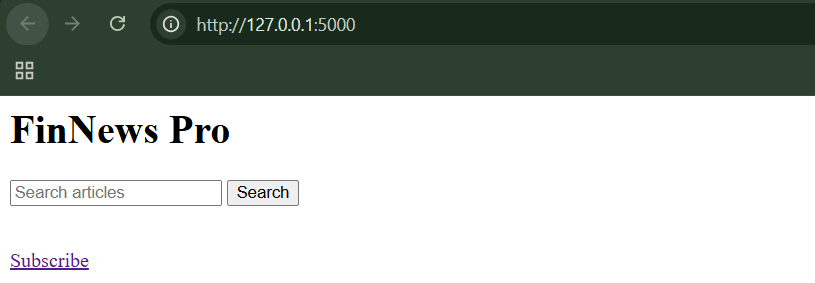
Remove any two of them from the cart:



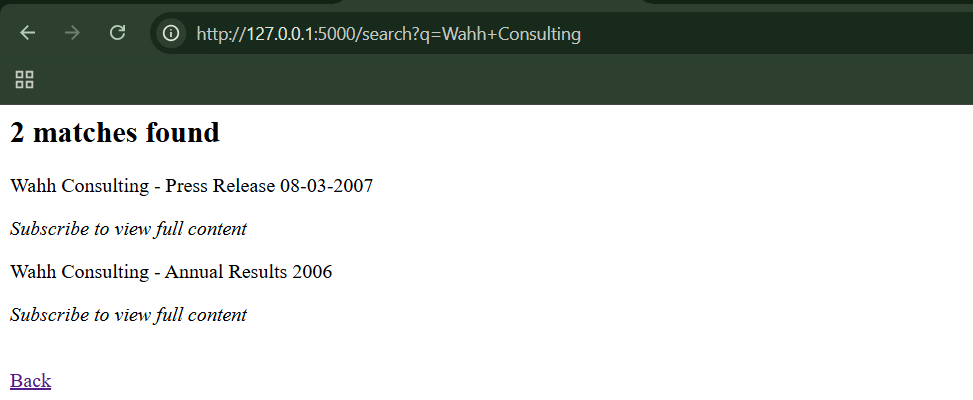
Original price was $80. We still have 25% discount. System did NOT recalculate.

**Example8: Abusing a Search Function**

Browser:

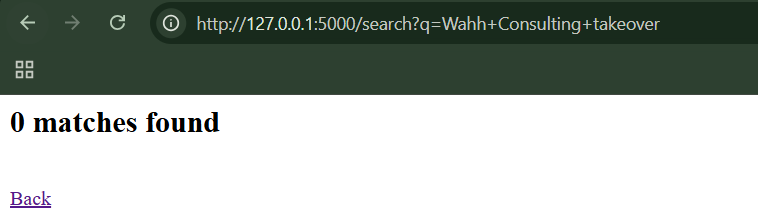


Step1: (Goal: Broad Search) Search: Wahh Consulting



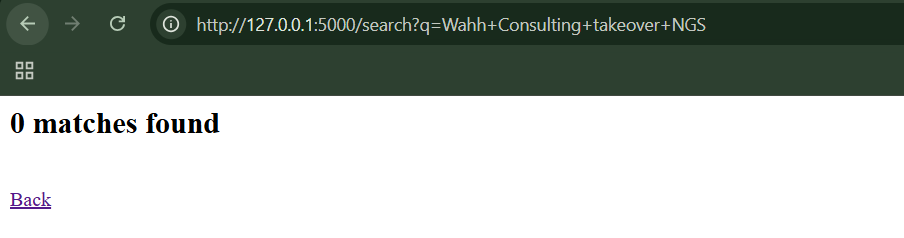
We are NOT subscribed. We see titles but no content.

Step2: (Goal: Try Narrowing) Search: Wahh Consulting takeover



We just learned: One article contains the word “takeover”.

Step3: (Goal: Refine further) Search: Wahh Consulting takeover NGS



Step4: (Goal: Test Different Outcomes)

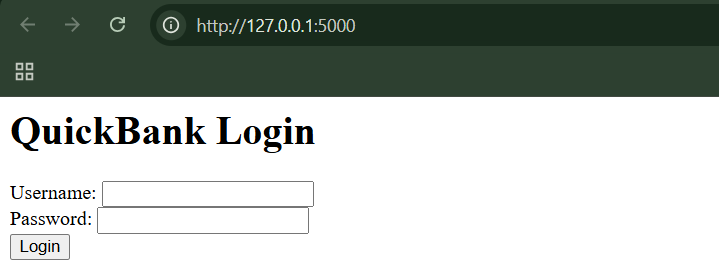
Search: Wahh Consulting takeover cancelled -> 0 matches

Search: Wahh Consulting takeover completed -> 0 matches

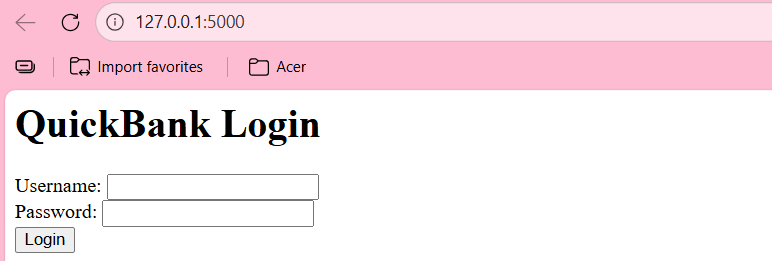
Without subscribing, we just reconstructed: Wahh Consulting completed takeover of NGS, is not present.

**Example9: Snarfing Debug Messages**

Browser1:



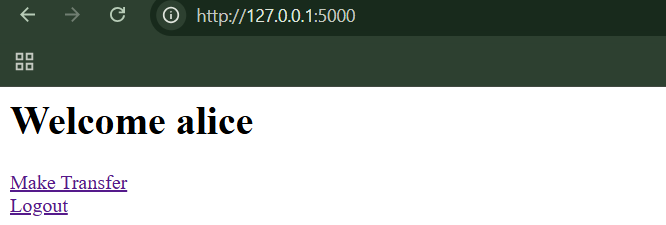
Browser2:



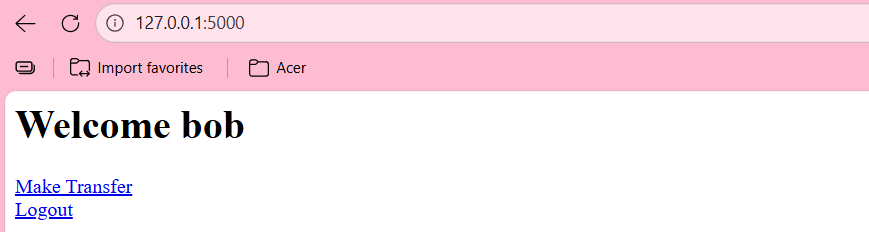
Step1: (Goal: Login)

* Browser 1 → Login as alice
* Browser 2 → Login as bob

Browser1:



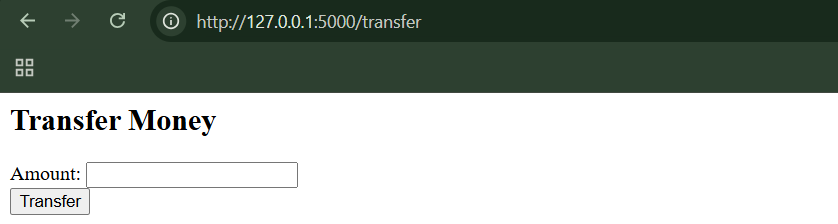
Browser 2:



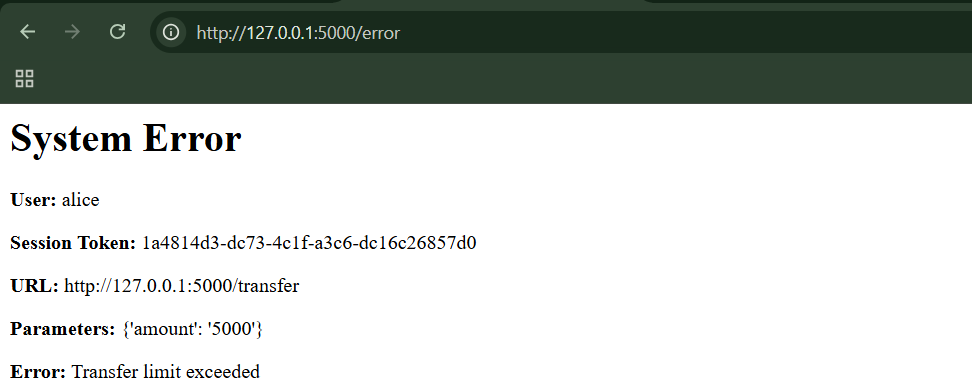
Step2: (Goal: Alice Triggers an Error)

In Browser 1 (Alice):

Go to: Transfer Money



Try to transfer the money ’5000’.



This triggers the artificial bug.

We get redirected to: /error

You’ll see:

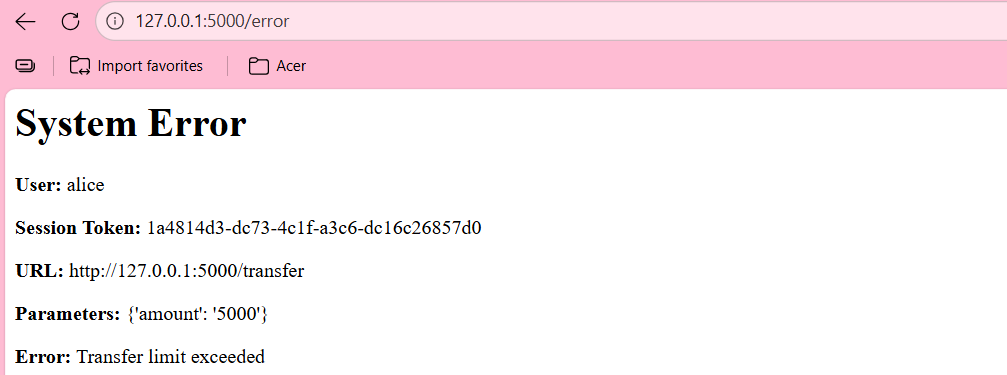
* Alice’s username
* Alice’s session token
* Alice’s parameters
* Error message

So far, this seems normal.

Step3: (Goal: Bob Steals Alice’s Debug Info)

In Browser 2 (Bob): Without triggering any error, manually go to:

<http://127.0.0.1:5000/error>



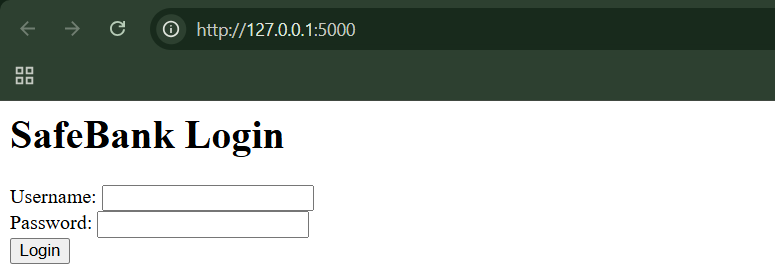
Bob now sees:

* Alice’s username
* Alice’s session token
* Alice’s transfer details

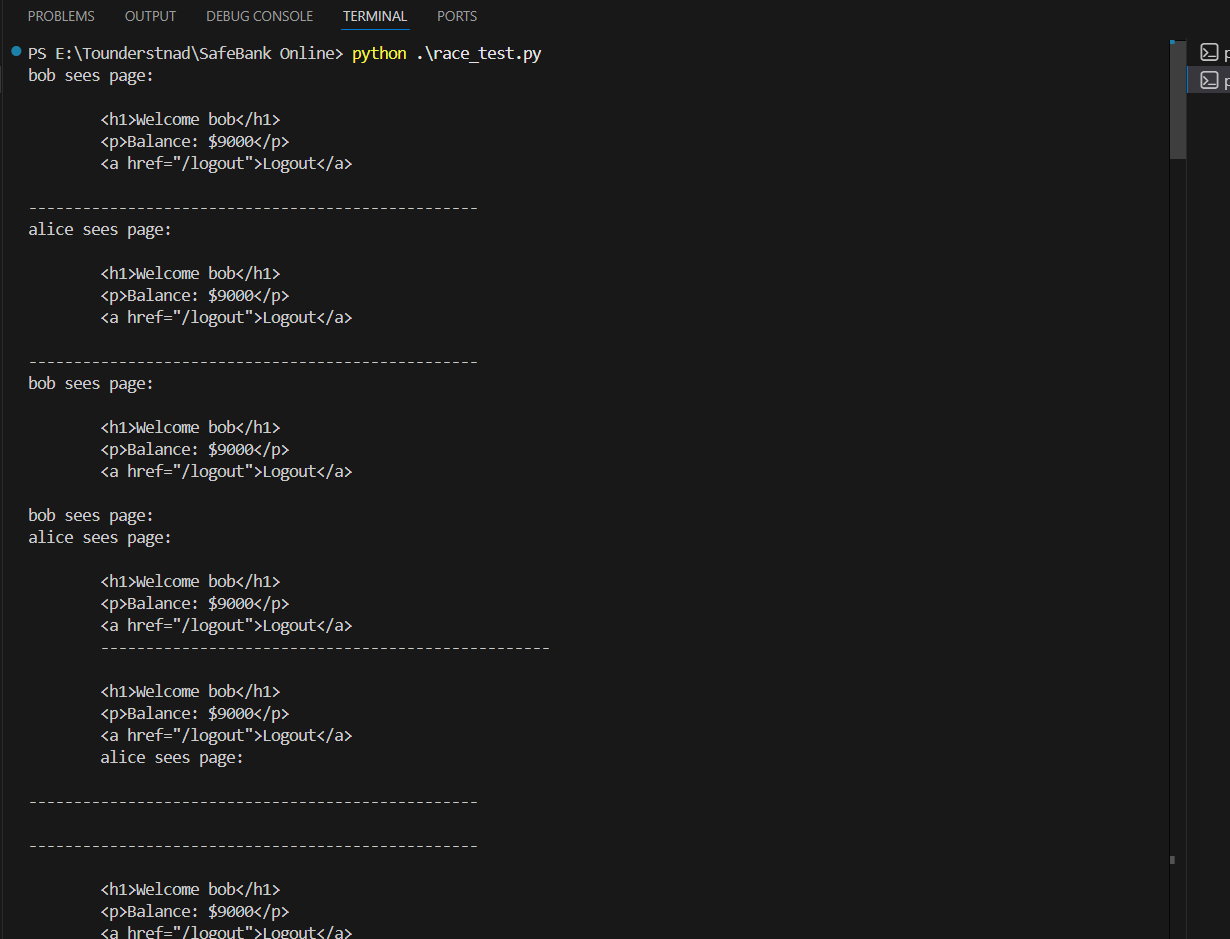
This is cross-user data leakage.

**Example10: Racing against the Login**

Browser:



Race condition:



**Avoiding Logic Flaws:**

Logic flaws are prevented by:

* Clear documentation
* Strict session-based identity
* No shared mutable state
* Careful state transitions
* Defensive design reviews
* Lateral thinking during code review

--The End--