Dry Ice ’n Co — CTF Write-up(incomplete solve)

Challenge Summary

This challenge presents a small Spring Boot e-commerce web application for buying dry ice products — and a special secret product named `"flag"`.

The goal is to unlock the flag by purchasing it.

Application Overview

- Language & stack: Java, Spring Boot, Thymeleaf.

- Products: Stored in memory, with name, description, price, and stock.

- Cart & balance: Per session (HttpSession).

- Coupon: One valid coupon `SMILEICE` for 20% off.

- Admin panel: Can add new products.

Key Code Details

`/admin/add-product`

java

```
if ((user.admin = true) && user != null && name != "flag")
```

Uses assignment = instead of comparison == → grants admin to anyone who hits the endpoint.

`/purchase`

```
if (cart.canAfford()) {
boolean allInStock = ...
if (allInStock) {
reduce stock...
if (items.size() == 1 && items.get(0).getName().equals("flag")) {
boughtFlag = true;
}
cart.purchase();
}
}
```

Only unlocks the flag if the cart has exactly one item named `"flag"` and you can afford it.

Vulnerabilities Discovered

ID Description Impact

1. Assignment in `if`	`(user.admin = true)` instead of comparison	Anyone can become admin
2. Bad reference comparison	`name != "flag"` uses reference not content	Makes blocking duplicates weak
3. Insecure stock control	No locks; global in-memory stock	Race condition: possible oversell
4. Cart logic flaw	Purchase works with empty cart	Harmless bug
5. Session stickiness	Load balancer can break session consistency	Causes unexpected 500 errors

What I Tried

Attempt	Idea	Result
Privilege escalation	POST to `/admin/add-product` → assignment bug makes me admin	Worked
Overwrite real `"flag"`	Add a new product with same name `"flag"` but cheaper price	Added, but server always picks the first `"flag"` in list
Homoglyphs & encoding	Tried `flag%00`, Unicode, double-encoding	Displays in cart but `.equals("flag")` passes only for exact match
Negative price	Create product with negative price to profit	Blocked by `Math.abs`
Intercepted purchase	Pause `/purchase` request, modify cart, then resume	Server uses session cart snapshot, so forging body doesn't change logic
Repeated purchase	Replay `/purchase` multiple times	Does nothing: cart is emptied on first purchase
Parallel race condition	Race multiple sessions to oversell stock	Possible for cheap products but no impact on flag due to price

How I Used Burp Suite

I used Burp Suite in two main ways:

1. **Intercepting and analyzing requests**
2. Opened the shop in a browser with Burp Proxy running.
3. Used Intercept to capture:
4. Adding products (`/add`)
5. Removing products (`/remove/{index}`)
6. Applying coupon codes (`/apply-coupon`)
7. Purchasing (`/purchase`)
8. Checked POST bodies and cookies (`JSESSIONID`).
9. **Crafting custom POST requests**
10. Sent custom requests in Repeater:
11. POST to `/admin/add-product` as a non-admin to exploit the `user.admin = true` bug.
12. Added a fake `"flag"` with low price.
13. Added homoglyph names and tested Unicode.
14. Intercepted the purchase request and modified the cart in parallel.
15. Confirmed that forging the request body does not change what the server uses: only the session-side cart matters.

Why Creating a Cheap `"flag"` Fails

When adding to cart:

```
availableProducts.stream()

.filter(p -> p.getName().equals(productName))

.findFirst()
```

This means the app always picks the first product in the list with name `"flag"`. So the original `"flag"` with price $1,000,000 is always used, and my fake cheap `"flag"` is ignored.

Final Analysis of Purchase Logic

1. Cart must have exactly one item named `"flag"`
2. Must pass `canAfford()`

3. `.equals("flag")` blocks homoglyph or encoded name tricks.
4. `Math.abs` neutralizes negative totals.
5. So:
6. Cannot underpay.
7. Cannot trick stock drain.
8. Cannot unlock flag with a fake product.

Valid Exploits

Exploit	Status	Notes
Become admin	Worked	By POSTing to `/admin/add-product`
Create arbitrary products	Worked	Any name except `"flag"`
Overwrite flag	Fails	Only first match is used
Unlock flag	Fails	Cannot bypass `.equals("flag")` and cost check
Race condition	Works	Only oversells normal stock, not flag

Key Takeaways

1. Privilege escalation: Simple but effective bug (assignment instead of comparison).
2. String comparison bug: Causes weak duplicate check but does not affect the secure `.equals` flag condition.
3. Business logic race: Classic concurrency flaw for normal items.
4. Purchase bypass: Empty cart purchase possible but harmless.
5. Burp Suite: Crucial for verifying session handling, cookie reuse, and server trust model.

Conclusion

Despite multiple flaws, the flag remains secure due to a robust final check:

1. Must pay the full price ($1,000,000)

2. Must have exactly one true `"flag"` product in the cart
3. Must pass `.equals("flag")`

Found no feasible way to bypass the price or inject a cheap substitute because of list order and correct comparison.