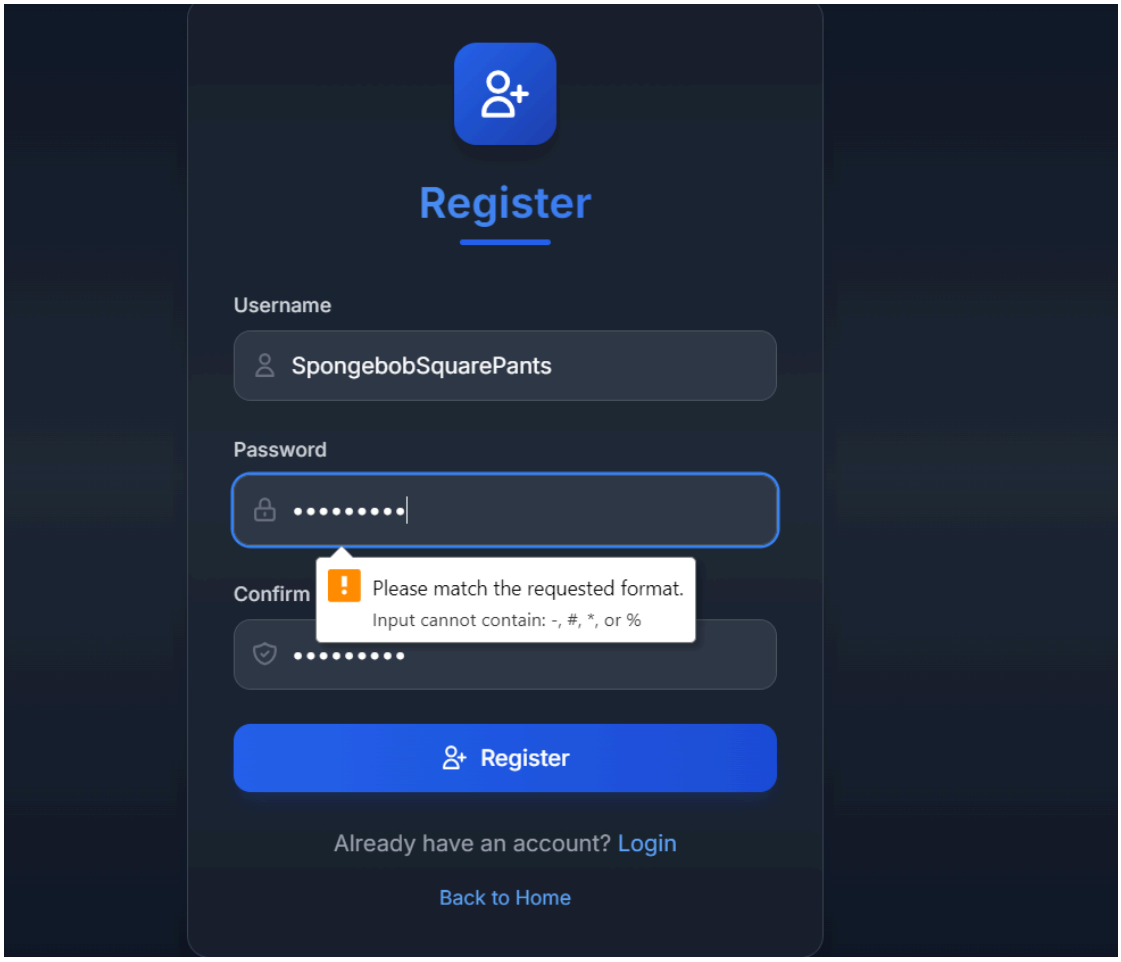
Admin Account Registration Attack:

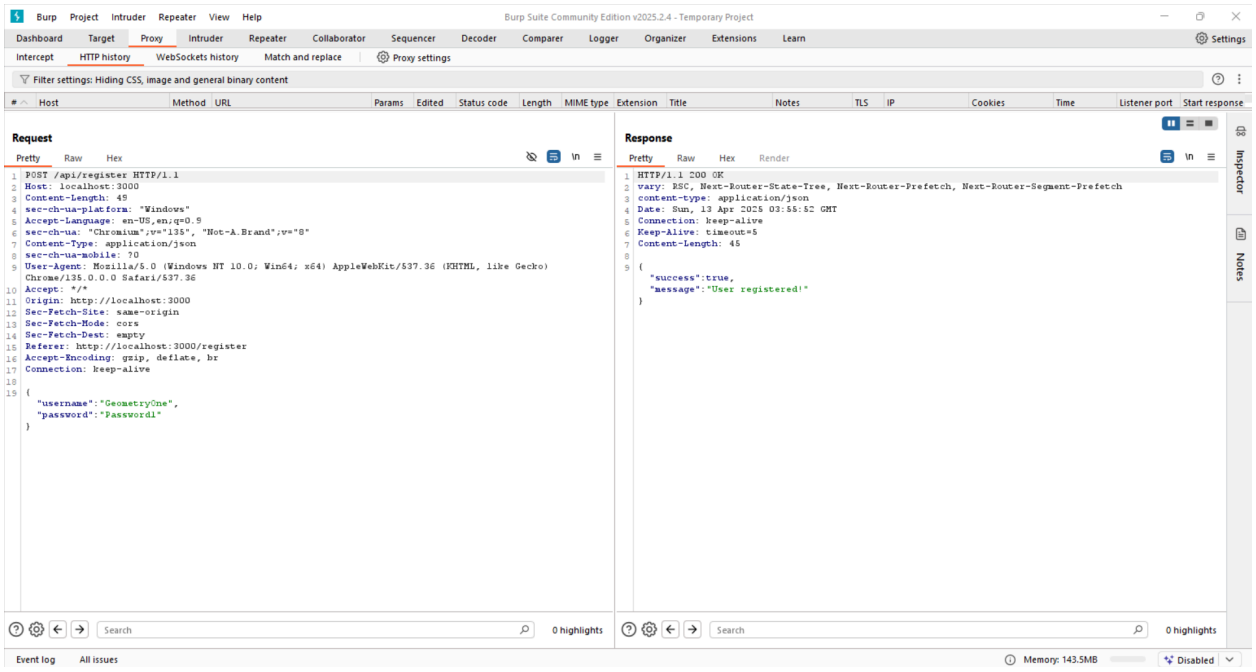
Challenge Creator: Ajay Ariaran
Writeup Author/Solver: Daniyal Abbas Lilani
Date: April 15, 2025

Objective: To exploit the registration page of this website for the purpose of registering a new account under the “admin’’ role
The page currently only registers new accounts under the “user” role

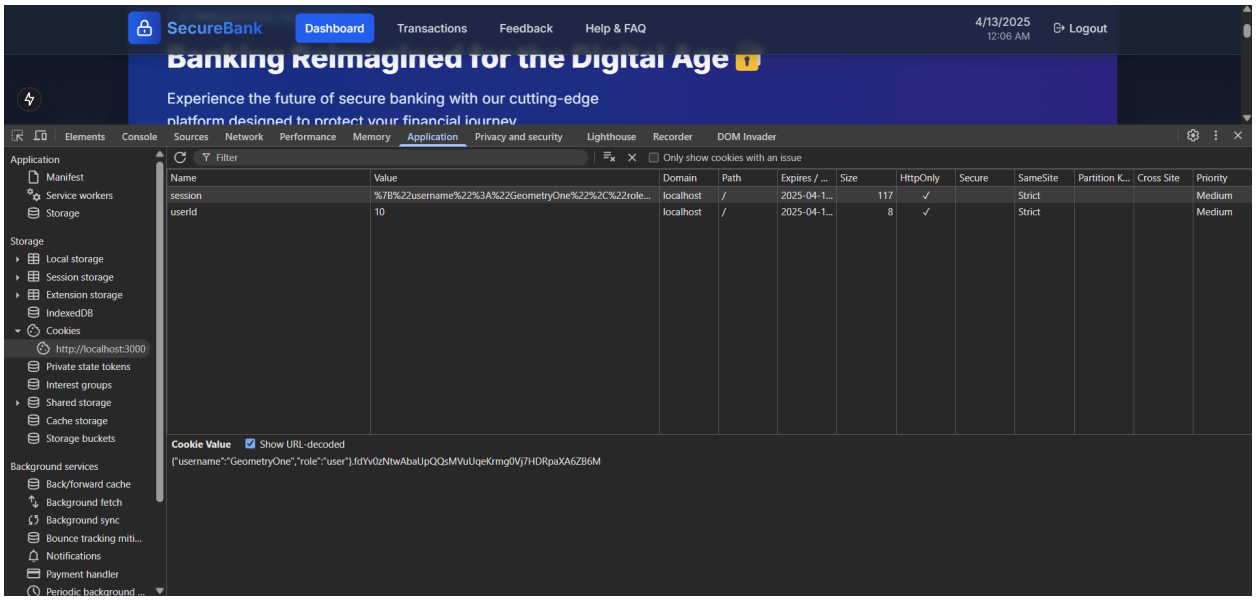
First let's open the page to figure out if doing injection within the field is possible, and what filtering might exist



- ➔ It seems this page has some filtration functionality
- ➔ With the input not being able to contain -, #, *, or %, performing an injection through the user interface itself would prove to be difficult
- ➔ Perhaps this filtration is only on the frontend
- ➔ Let's use Burp Suite to test this further

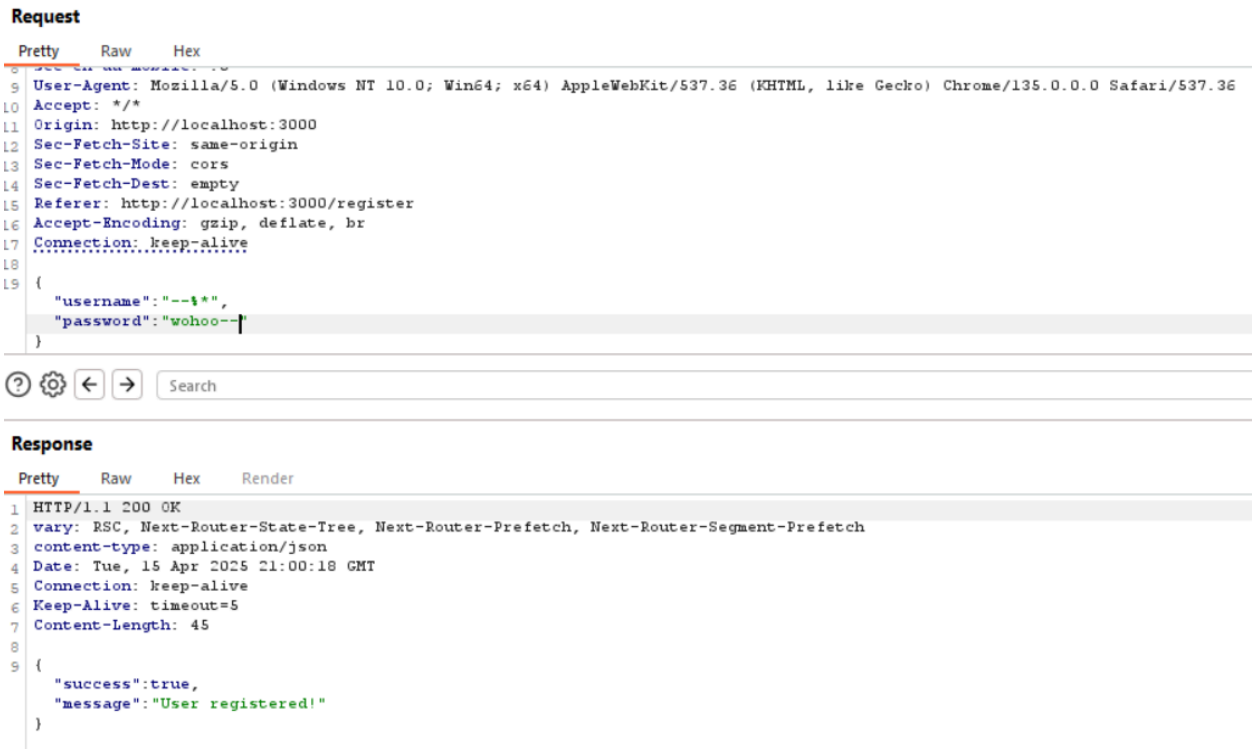


- ➔ Here I have registered an account without any of the illegal characters the for the purpose of catching that request
- ➔ Before we attempt to alter the request in any manner, let us explore beyond the login page to see if we can collect any useful information



- ➔ As seen in class, analyzing cookies are imperative to understanding how session tokens are typically stored
- ➔ Here we can see that the information of this “user” account is displayed in the following order: username then role.

➔ This kind of information is important for crafting my first SQL injection. First, I need to check if I can use the illegal characters or If I need encoding.



➔ Okay, no encoding needed

➔ Since this is a registration page, I can assume that the backend code looks something like

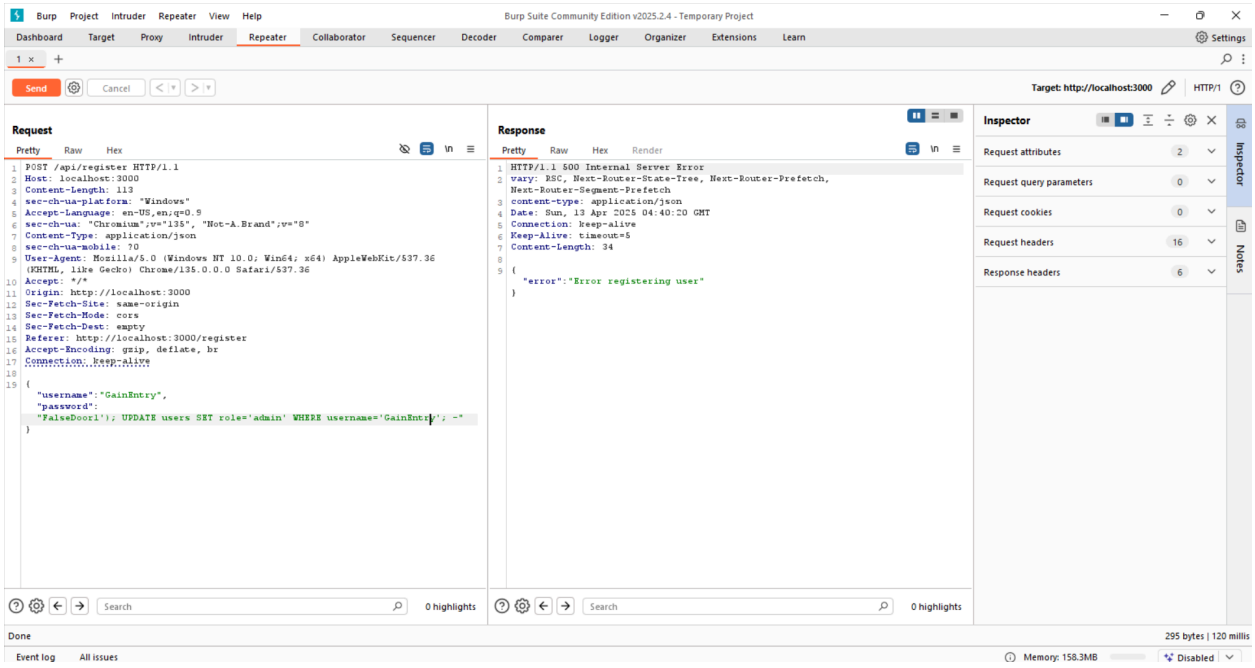
```
`INSERT INTO Users (username, password, role) VALUES ('${username}', '${password}', 'user')`;
```

➔ Based on this information we've seen I'll use the following attack first:
'); UPDATE users SET role='admin' WHERE username='Hello01'; – in the password section

➔ The reasoning here is as follows:

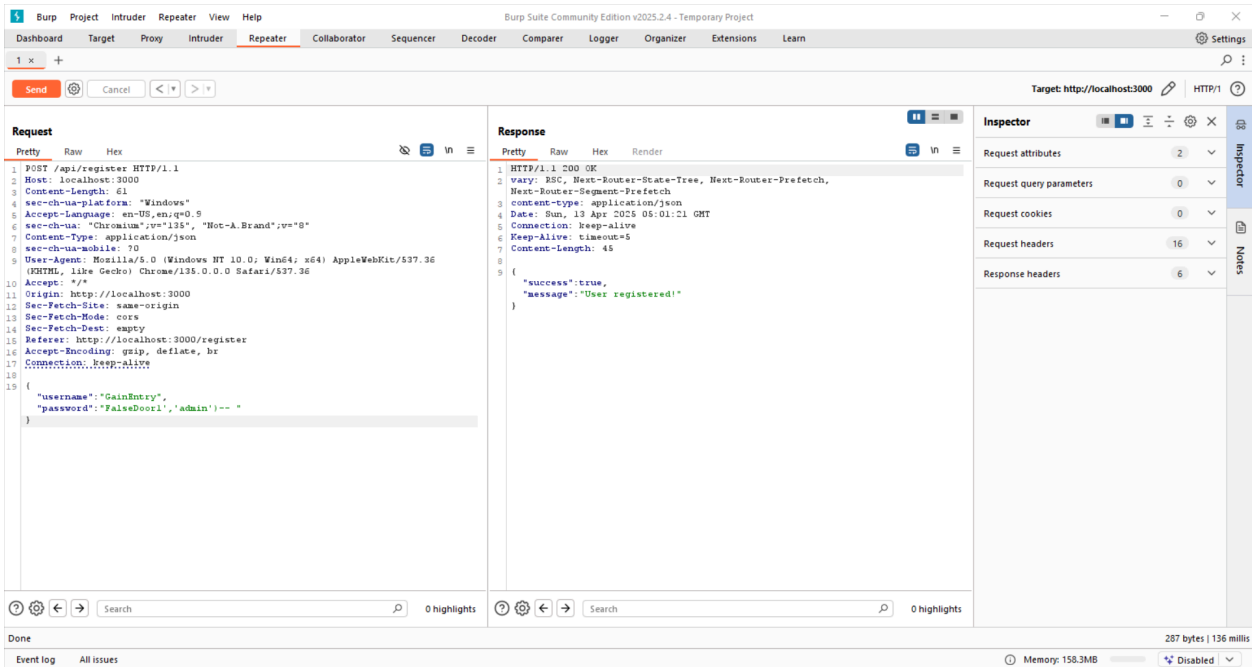
- ◆ I want to try a simple approach first where I essentially "break out" of the intended INSERT and also run an UPDATE in the same query
- ◆ With this injection I essentially want to close off the password string and force a second statement which would be responsible for setting the role to admin

➔ Let's try that here below by tweaking the following request we caught earlier:



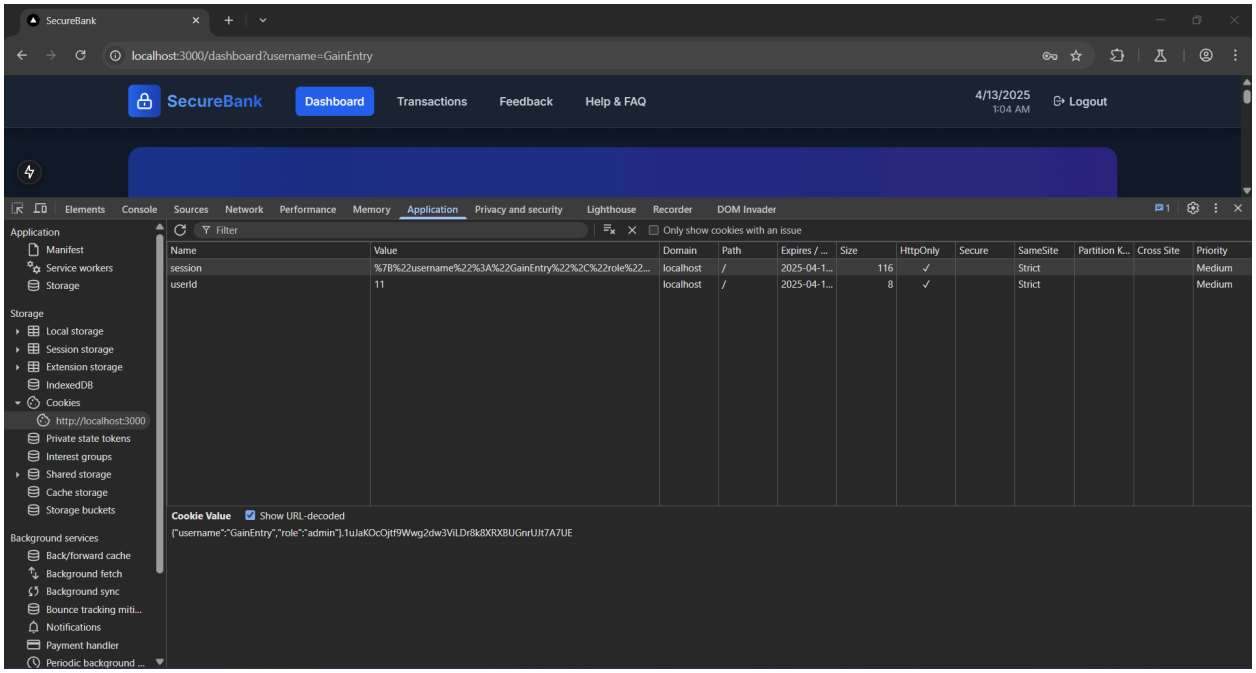
- ➔ Interesting, this injection was unsuccessful but I believe I know the reason
- ➔ Some background information I already know about this challenge is that the database uses SQLite specifically
- ➔ What's clear to me is that the injection did in fact break out of the password string and add another statement but I think SQLite is facing an error while attempting to run both statements
- ➔ **This is because most of these libraries default to single-statement execution only and will output an error if a semicolon is inside user input**
- ➔ **Upon further research, I came to find out that this happens because semicolons are used to separate multiple SQL statements, and allowing them would make it easier for attackers to chain malicious queries.**
- ➔ Now that I know this, I can tweak the way my injection is performed and try another method, as it is clear I cannot use 2 statements in one request.
- ➔ What I'll do here is turn the single INSERT itself into one that inserts the admin for the role
- ➔ I'll do that by breaking out of the password field, appending a value for role and comment out the rest of the information
- ➔ My new injection is as follows:

FalseDoor1','admin')--



➔ As you can see, this injection was successful this time around

➔ To confirm whether this injection truly worked, let us take a look at the cookies once we login with these new credentials



➔ Based on the analysis of the cookies, it appears that I have successfully performed a valid attack on the registration page – one that registers an account as an admin instead of a user

Therefore the challenge is now complete

Improving security

- Use parameterized queries instead of appending user input to the POST request directly
- Add filtration on the server side
- Add a rule in the database that only one (or as few as necessary) admins can exist