Challenge: PawForce

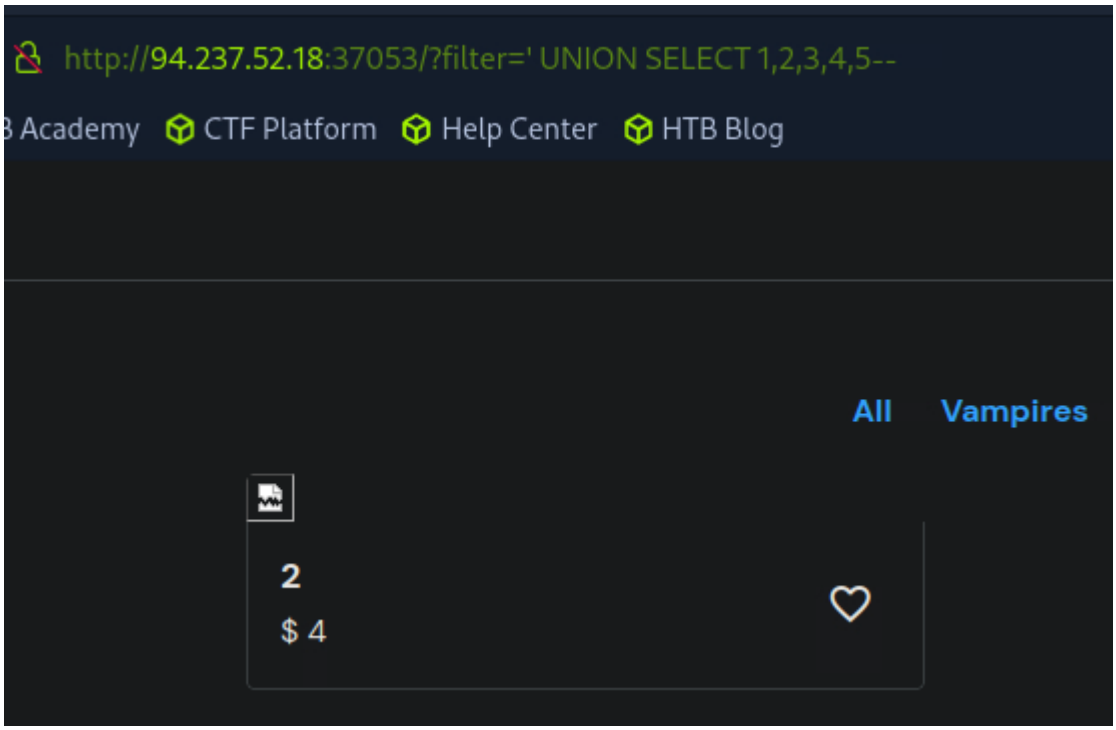
Dive into 'Super Cats': where feline fans meet. But beware, not all purrs are safe. Can you outsmart the system with a clever query?

Answer:HTB{SQL1t3_1nj3ct10n_c4n_b3_4nywh3r3_m4n!!_91df6bae57348feb59001efd112d681d}

Procedure:

So first, with the little description when it mentioned clever query I could tell that this was going to be an SQL injection challenge. However, when entering into the target I didn't observed any textfield to try the injection. Nonetheless, when clicking the links I could observe that in the url we got an injection on the filter parameter. For example, http://94.237.52.18:37053/?filter=Vampires.

I knew that this was the way to go. I started right away trying the UNION clause, and I try this: http://94.237.52.18:37053/?filter=' UNION SELECT 1,2,3,4,5--



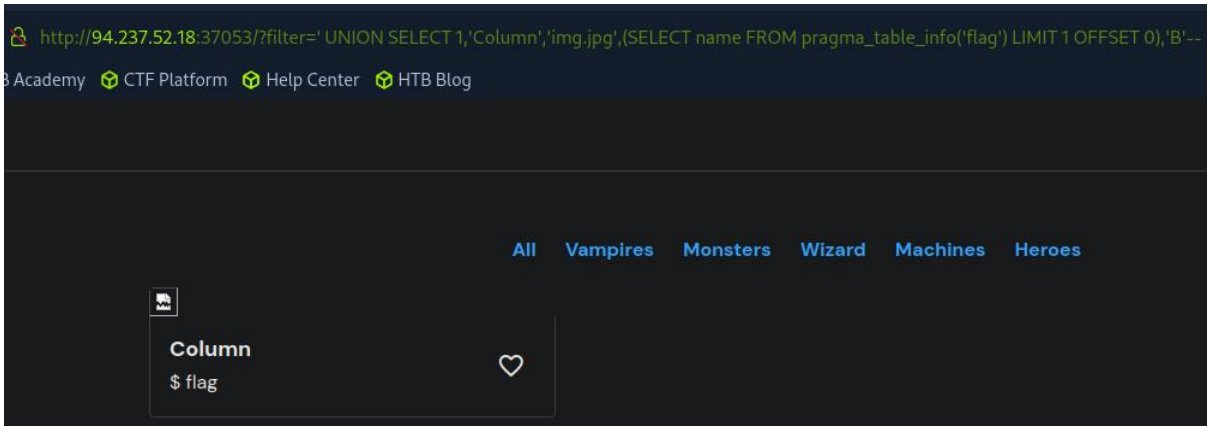
As you can observe on the image, the values for 2 and 4 are shown, on the title of the image and on the price tag. Meaning that these are the ones that are shown on the page.

Afterwards, I need to know which tables are present on the database, when analyzing the files that could be downloaded from HTB, I found a file called databases and found out there is a table called paws and flag. Like is shown in the image:

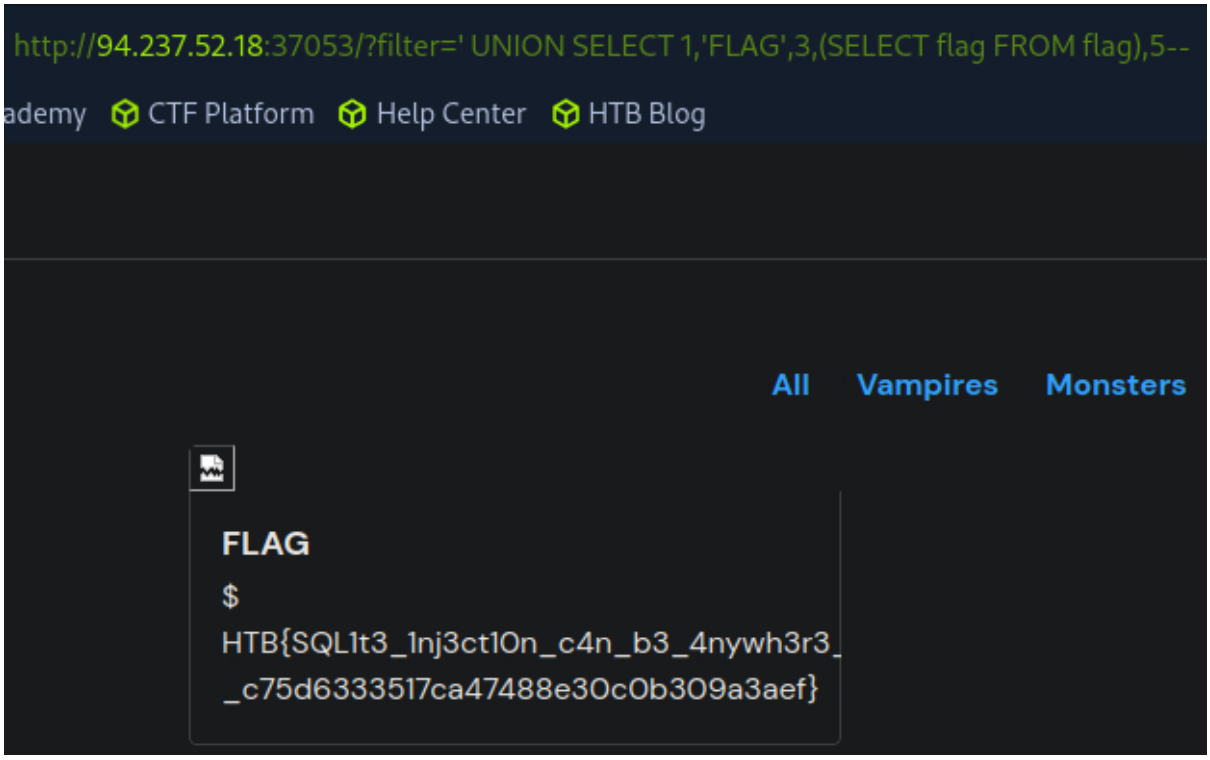
```
class Database {
        constructor(db_file) {
                this.db_file = db_file;
                this.db = undefined;
        }

        async connect() {
                this.db = await sqlite.open(this.db_file);
        }

        async migrate() {
                return this.db.exec(`
            DROP TABLE IF EXISTS paws;
            DROP TABLE IF EXISTS flag;


            CREATE TABLE paws(
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name VARCHAR(255) NOT NULL,
                image VARCHAR(255) NOT NULL,
                rent VARCHAR(255) NOT NULL,
                                paw_type VARCHAR(255) NOT NULL
            );

                        CREATE TABLE flag(
                                flag VARCHAR(255) NOT NULL
                        );

                        INSERT INTO flag(flag) VALUES('${flag}');

                        INSERT INTO paws(name, image, rent, paw_type) VALUES (
                                'Thunderpurr',
                                'asguard.webp',
                                '$ 10/h',
                                'Heroes'
                        );
```

With this, I also find out that it was using sqlite. Then I needed to found out the column name, and I asked chatgpt how could I do that for sqllite and it send me this injection:

http://94.237.52.18:37053/?filter=%27%20UNION%20SELECT%201,%27Column%27,%27img.jpg%27,(SELECT%20name%20FROM%20pragma_table_info(%27flag%27)%20LIMIT%201%20OFFSET%200),%27B%27--



I found out that the column is flag inside the table flag. So knowing this I could search for the flag using this injection in the second and fourth place of the union clause: http://94.237.52.18:37053/?filter=%27%20UNION%20SELECT%201,%27FLAG%27,3,(SELECT%20flag%20FROM%20flag),5--



However, I didn't let me copy and paste the flag, I tried to copy the flag into HTB but I had many errors, so I enter into inspecting the page and look for the flag.

This is the flag:
HTB{SQL1t3_1nj3ct10n_c4n_b3_4nywh3r3_m4n!!_91df6bae57348feb59001efd112d681d}

Risk Mitigation Strategy: User input should always be treated as untrusted and must undergo strict validation before being used in any SQL query. In this challenge, since the filter parameter is supposed to match predefined categories such as Heroes, Monsters, Wizard, etc., the application should only accept input that exactly matches one of these valid options. This can be achieved through whitelisting, where user input is compared against a fixed list of acceptable values. Any input that does not match should be rejected before reaching the database layer. Additionally, input sanitization can be applied to escape or remove potentially dangerous characters, although strong validation combined with prepared statements remains the most effective defense. Implementing proper input validation greatly reduces the attack surface by limiting how user-controlled data can influence query logic.