Burp Sutie Intruder

Sniper: Default and most commonly used. Used for effective single-position attacks such as password - brute force or fuzzing for API endpoints. In a sniper attack we provide a set of payloads which can be a wordlist or range of numbers, and intruder inserts each payload in a defined position in the request.

Let's refer to our example template from before:

Example Positions

```
POST /support/login/ HTTP/1.1
Host: MACHINE_IP
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0) Gecko/20100101 Firefox/80.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 37
Origin: http://MACHINE_IP
Connection: close
Referer: http://MACHINE_IP/support/login/
Upgrade-Insecure-Requests: 1

username=§pentester§&password=§Expl01ted§
```

In this example, we have two positions defined for the `username` and `password` body parameters. In a Sniper attack, Intruder takes each payload from the payload set and substitutes it into each defined position in turn.

Assuming we have a wordlist with three words: `burp`, `suite`, and `intruder`, Intruder would generate six requests:

Request Number	Request Body
1	`username=burp&password=Expl01ted`
2	`username=suite&password=Expl01ted`
3	`username=intruder&password=Expl01ted`
4	`username=pentester&password=burp`
5	`username=pentester&password=suite`
6	`username=pentester&password=intruder`

Observe how Intruder starts with the first position (`username`) and substitutes each payload into it, then moves to the second position (`password`) and performs the same substitution with the payloads. The total number of requests made by Intruder Sniper can be calculated as `requests = numberOfWords * numberOfPositions`.

The Sniper attack type is beneficial when we want to perform tests with single-position attacks, utilizing different payloads for each position. It allows for precise testing and analysis of different payload variations.

Answer the questions below

If you were using Sniper to fuzz three parameters in a request with a wordlist containing 100 words, how many requests would Burp Suite need to send to complete the attack?

300 ✓ Correct Answer

How many sets of payloads will Sniper accept for conducting an attack?

1 ✓ Correct Answer

Battering-Ram: Differs from sniper in that it places the same payload into every position simultaneously rather than substituting each payload into each position in turn.

Let's refer back to our previous example template:

Example Positions

```
POST /support/login/ HTTP/1.1
    Host: MACHINE_IP
    User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0) Gecko/20100101 Firefox/80.0
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
    Accept-Language: en-US,en;q=0.5
    Accept-Encoding: gzip, deflate
    Content-Type: application/x-www-form-urlencoded
    Content-Length: 37
    Origin: http://MACHINE_IP
    Connection: close
    Referer: http://MACHINE_IP/support/login/
    Upgrade-Insecure-Requests: 1

    username=§pentester§&password=§Expl01ted§
```

Using the Battering Ram attack type with the same wordlist from before (`burp` , `suite` , and `intruder`), Intruder would generate three requests:

Request Number	Request Body
1	`username=burp&password=burp`
2	`username=suite&password=suite`
3	`username=intruder&password=intruder`

As shown in the table, each payload from the wordlist is inserted into every position for each request made. In a Battering Ram attack, the same payload is thrown at every defined position simultaneously, providing a brute-force-like approach to testing.

The Battering Ram attack type is useful when we want to test the same payload against multiple positions at once without the need for sequential substitution.

In the upcoming tasks, we will explore further configurations and settings related to Intruder's Battering Ram attack type and examine its applications in different scenarios.

Answer the questions below

As a hypothetical question: You need to perform a Battering ram Intruder attack on the example request above.

If you have a wordlist with two words in it (`admin` and `Guest`) and the positions in the request template look like this:
`username=§pentester§&password=§Expl01ted§`

What would the body parameters of the *first* request that Burp Suite sends be?

username=admin&password=admin

✓ Correct Answer

Hint

Pitchfork: Similar to having multiple sniper attacks running simultaneously. While sniper only uses 1 payload, pitchfork utilises one payload set per position (up to a max of 20) and itereates through them al simultaneously.
Eg two worklists:

1. **First wordlist contains usernames joel, harriet and alex**
2. **Second wordlist contains passwords jo3l, Emma1815 and sk1ll.**

We can use these two lists to perform a Pitchfork attack on the login form. Each request made during the attack would look like this:

Request Number	Request Body
1	`username=joel&password=J03l`
2	`username=harriet&password=Emma1815`
3	`username=alex&password=Sk1ll`

As shown in the table, Pitchfork takes the first item from each list and substitutes them into the request, one per position. It then repeats this process for the next request by taking the second item from each list and substituting it into the template. Intruder continues this iteration until one or all of the lists run out of items. It's important to note that Intruder stops testing as soon as one of the lists is complete. Therefore, in Pitchfork attacks, it is ideal for the payload sets to have the same length. If the lengths of the payload sets differ, Intruder will only make requests until the shorter list is exhausted, and the remaining items in the longer list will not be tested.

The Pitchfork attack type is especially useful when conducting credential-stuffing attacks or when multiple positions require separate payload sets. It allows for simultaneous testing of multiple positions with different payloads.

In the upcoming tasks, we will explore further configurations and settings related to Intruder's Pitchfork attack type and explore its applications in different scenarios, including credential-stuffing attacks.

Answer the questions below

What is the maximum number of payload sets we can load into Intruder in Pitchfork mode?

20 ✓ Correct Answer

Cluster Bomb: Allows us to choose multiple payload sets, one per positon. Up to a max of 20 of course. Unlike pitchfork where all the payload sets are tested simultaneously, cluster bomb iterates through each payload set individually, ensuring that every possible combination of payload is tested.

To illustrate the Cluster bomb attack type, let's use the same wordlists as before:

- Usernames: `joel`, `harriet`, and `alex`.
- Passwords: `J03l`, `Emma1815`, and `Sk1ll`.

In this example, let's assume that we don't know which password belongs to which user. We have three users and three passwords, but the mappings are unknown. In this case, we can use a Cluster bomb attack to try every combination of values. The request table for our username and password positions would look like this:

Request Number	Request Body
1	`username=joel&password=J03l`
2	`username=harriet&password=J03l`
3	`username=alex&password=J03l`
4	`username=joel&password=Emma1815`
5	`username=harriet&password=Emma1815`
6	`username=alex&password=Emma1815`
7	`username=joel&password=Sk1ll`
8	`username=harriet&password=Sk1ll`
9	`username=alex&password=Sk1ll`

As shown in the table, the Cluster bomb attack type iterates through every combination of the provided payload sets. It tests every possibility by substituting each value from each payload set into the corresponding position in the request.

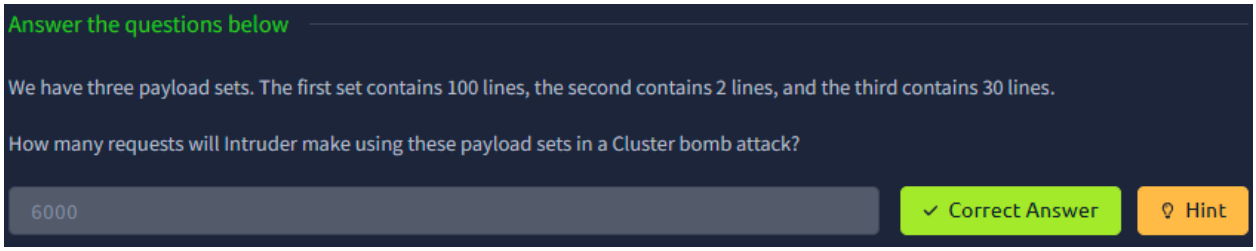
Cluster bomb attacks can generate a significant amount of traffic as it tests every combination. The number of requests made by a Cluster bomb attack can be calculated by multiplying the number of lines in each payload set together. It's important to be cautious when using this attack type, especially when dealing with large payload sets. Additionally, when using Burp Community and its Intruder rate-limiting, the execution of a Cluster bomb attack with a moderately sized payload set can take a significantly longer time.

The Cluster bomb attack type is particularly useful for credential brute-forcing scenarios where the mapping between usernames and passwords is unknown.

In the upcoming tasks, we will explore further configurations and settings related to Intruder's Cluster bomb attack type and examine its applications in different scenarios.

100 (Set 1) × 2 (Set 2) × 30 (Set 3) = 6,000 requests

✅ Answer: 6000



Practical:

We have the IP address of: http://10.10.250.250/support/login/ and upon viewing the source code, no protective measures have been implemented.

Given the absence of protective measures we have multiple options to exploit in this form, including cluster bomb attack for brute forcing credentials, but we do have an easier approach.

Approximately 3 months ago Bastio Hosting fell victim to a cyber attack compromising 3 employee usernames, email addresses and plain text passwords. While the affected employees were instructed to change their passwords promptly there is a possibility that someone disregarded this advice.

As we possess a list of known usernames, each accompanied by a corresponding password, we can leverage a credential stuffing attack instead of a straight forward brute force attack. This method provides an advantageous and significantly quicker attack, especially when utilising the rate limited edition we have on the community burp suite.

In order to access the leaked credentials we must visit: `wget http://10.10.250.250:9999/Credentials/BastionHostingCreds.zip`

```
root@ip-10-10-62-144:~# wget http://10.10.250.250:9999/Credentials/BastionHostin
gCreds.zip
--2025-07-23 16:13:06--  http://10.10.250.250:9999/Credentials/BastionHostingCre
ds.zip
Connecting to 10.10.250.250:9999... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3505 (3.4K) [application/zip]
Saving to: 'BastionHostingCreds.zip'

BastionHostingCreds 100%[===================>]   3.42K  --.-KB/s    in 0s

2025-07-23 16:13:06 (114 MB/s) - 'BastionHostingCreds.zip' saved [3505/3505]

root@ip-10-10-62-144:~#
```

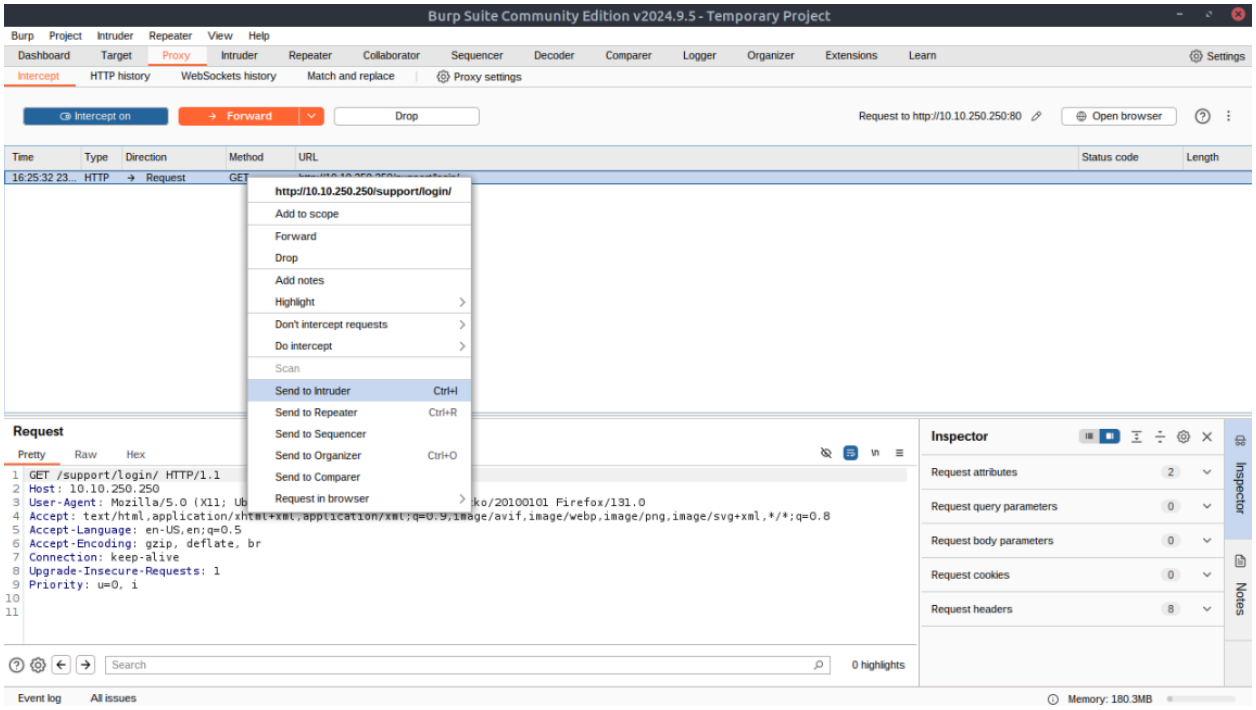
Here we can see it was successfully installed.

```
root@ip-10-10-62-144:~# unzip BastionHostingCreds.zip
```

```
Archive:  BastionHostingCreds.zip
  inflating: combined.txt
  inflating: emails.txt
  inflating: passwords.txt
  inflating: usernames.txt
```

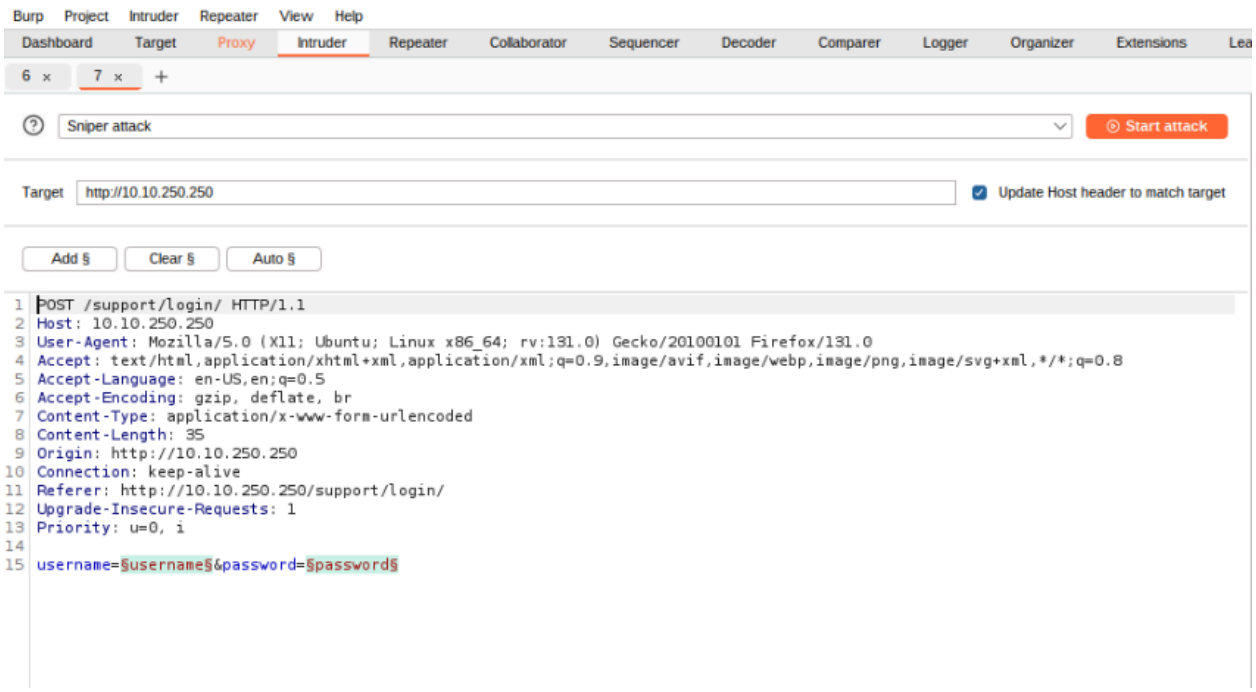
I created sub folders individually, and went through extracting the correct information so I could use it in the intruder attacks.

I then made sure FoxyProxy was enabled and Burp suite were running to capture the proxy request.

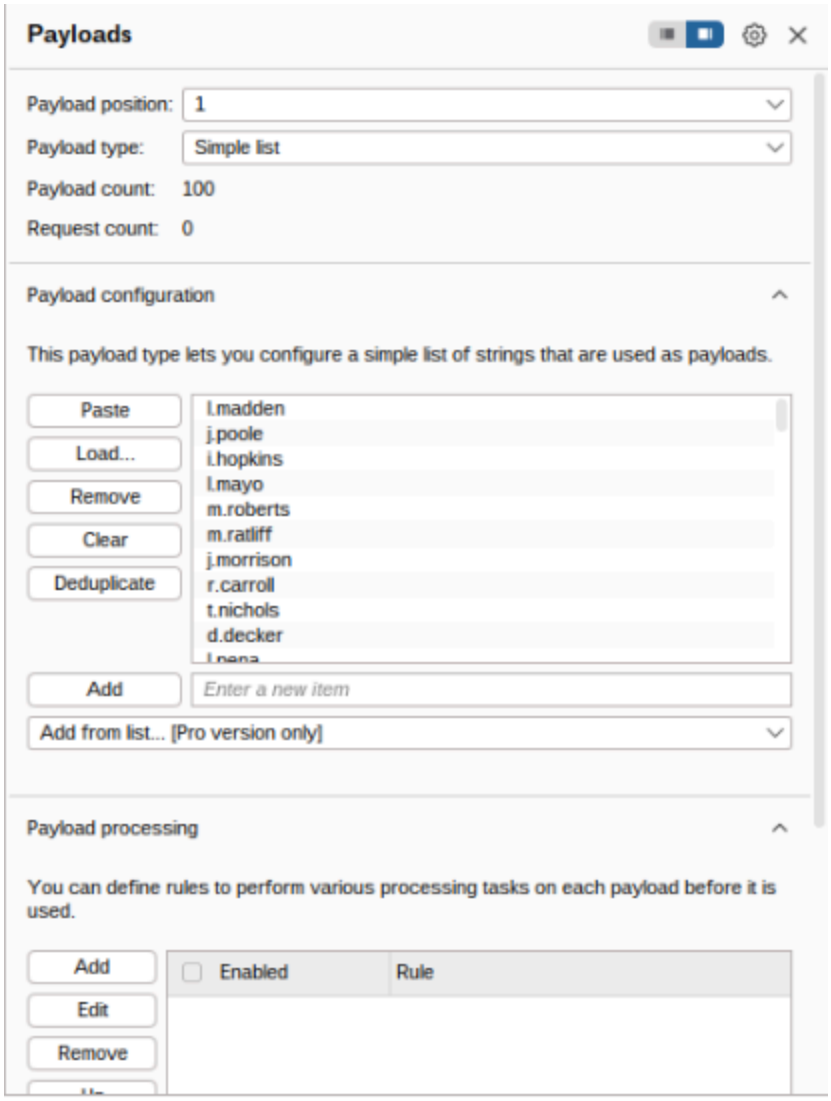


I then sent the GET request to the intruder. I then configure the Intruder tab as appropriate and select the “Pitchfork” attack.

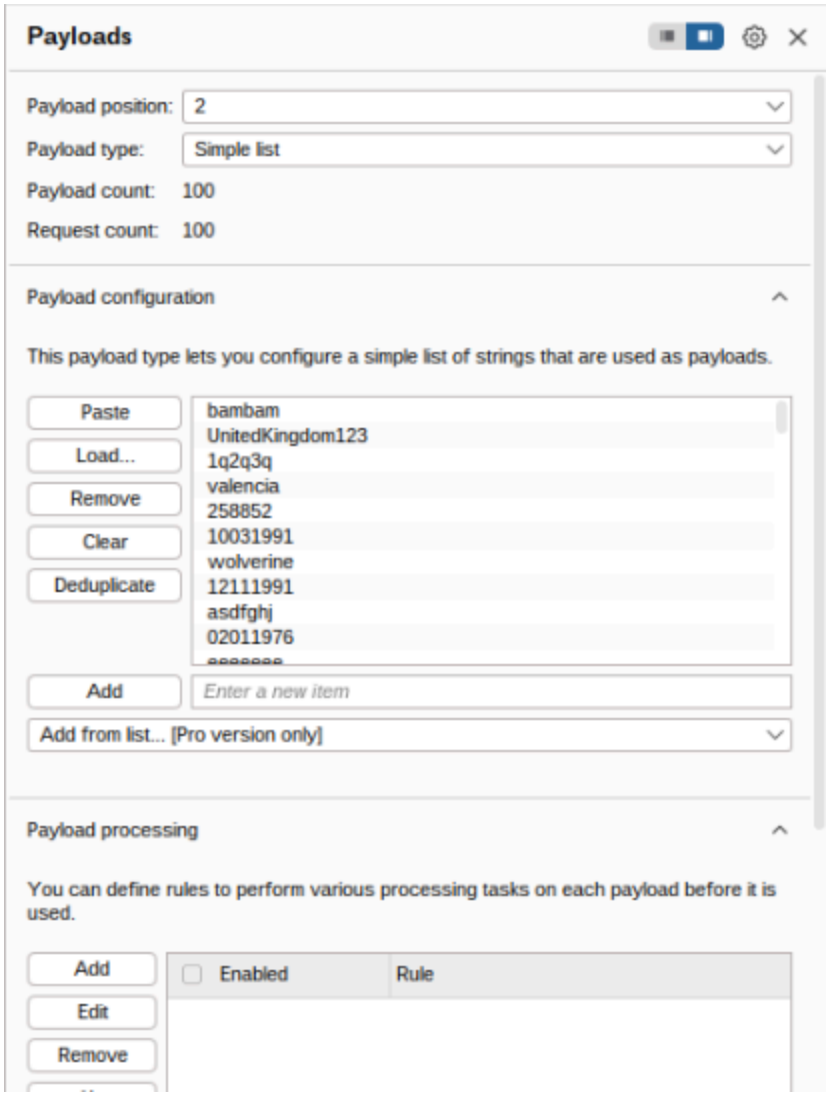
But, we need a POST request instead, so we send a random set of info into the username and password fields and enter.



Selecting “Auto” will automatically highlight the username and password field as shown.

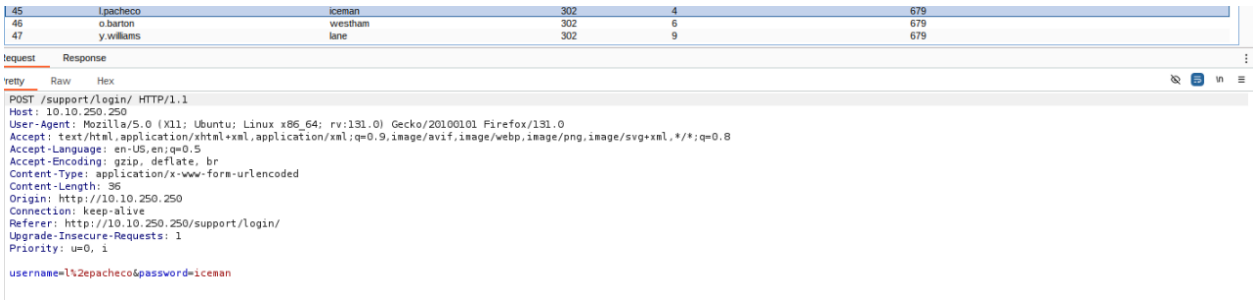


I then select payload position 1 “username.txt” which loads 100 payload.

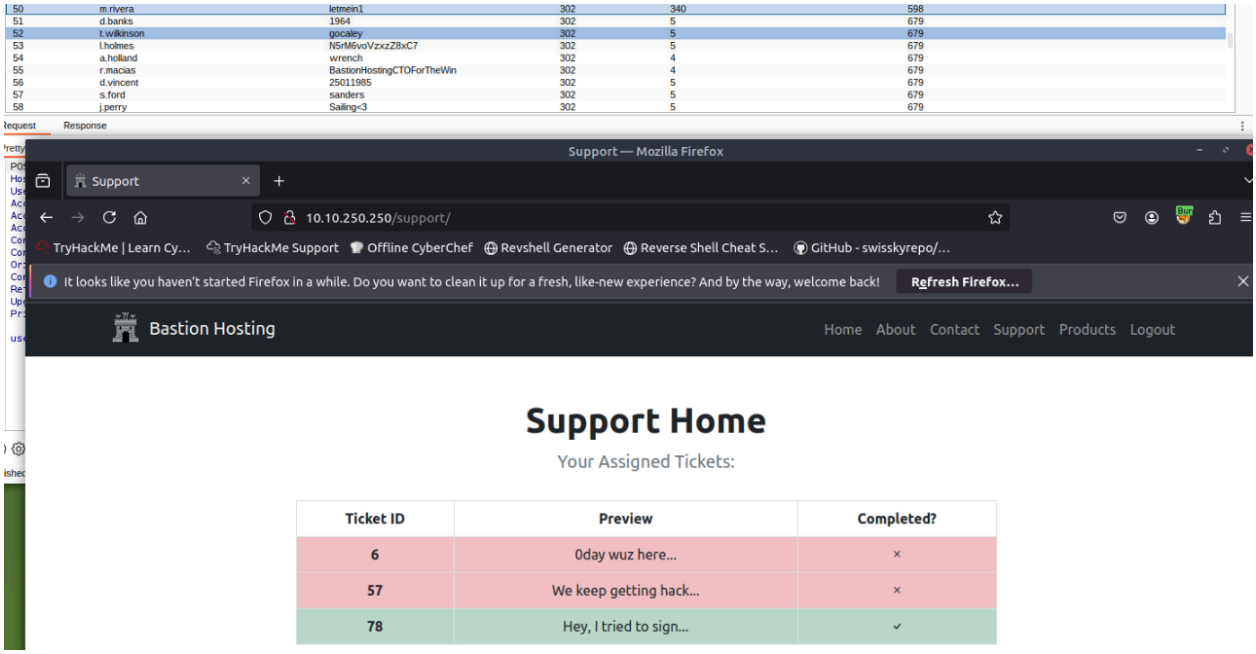


I then select payload positon 2 “password.txt” which loads 100 payloads.

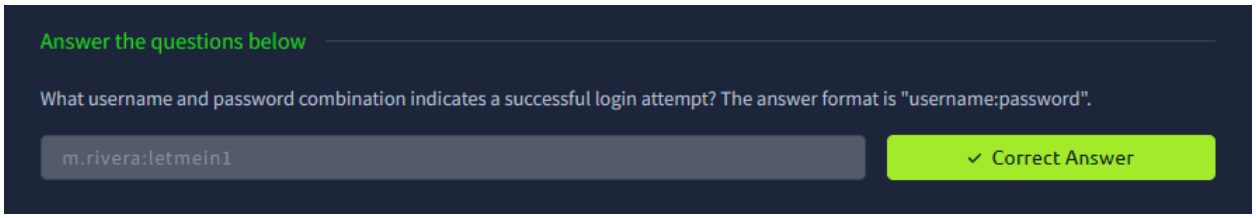
Once these are both configured I am ready to launch the attack!



So we are looking for the shortest response time that indicates a successful login attempt. So we can now try and see if it works, logging in using their credentials. Did not work…



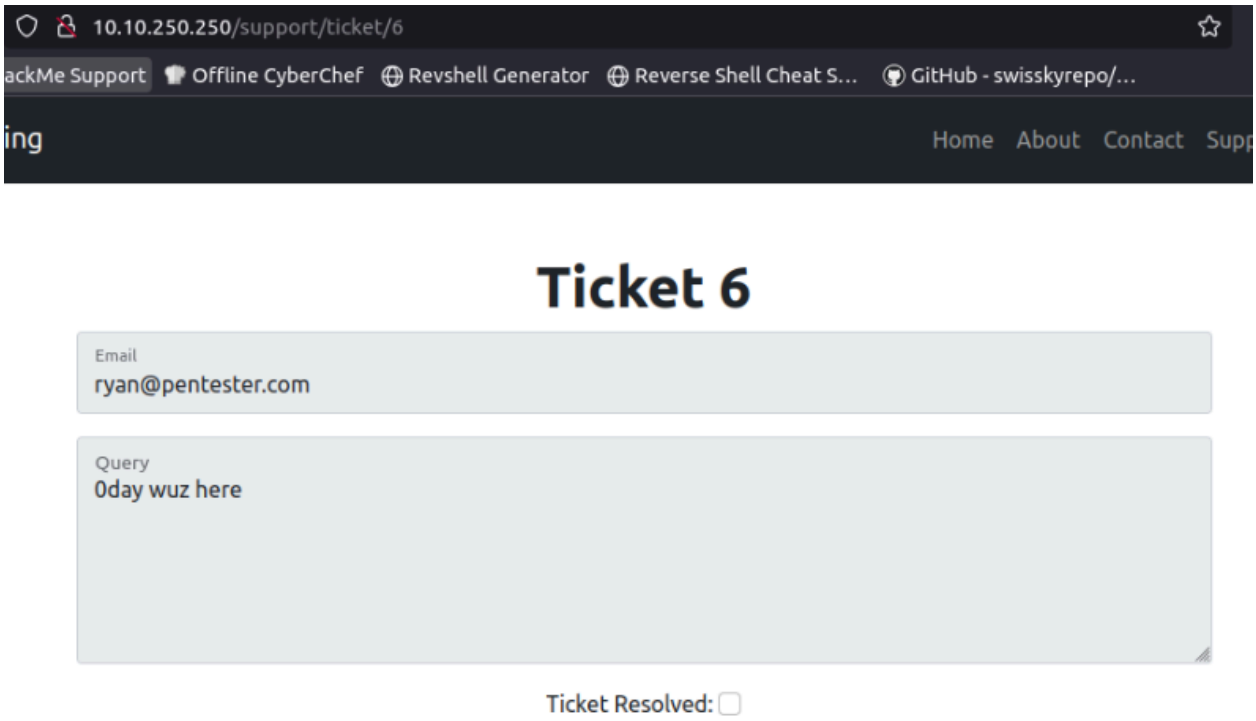
Or maybe not… The longest response time seemed to work instead?



Trial and error! We did it!!

THM Challenge:
So having gained access to the support system we now have the opportunity to explore its functionalities and see what actions we can perform.
Upon accessing the home interface I am presented with a table displaying various tickets. Clicking on any row redirects us to a page where we can view the incomplete ticket. By examining the URL structure we observe that the pages are numbered in the following format:

```
http://10.10.250.250/support/ticket/NUMBER
```

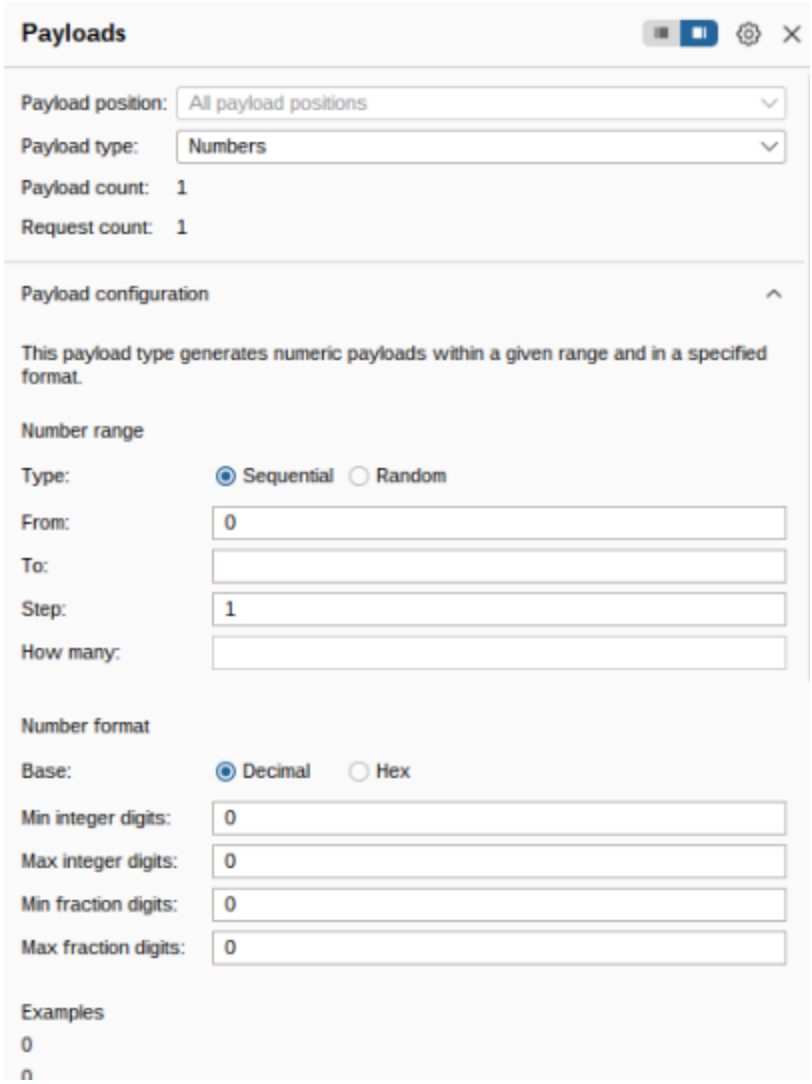


After some wild guessing! Ticket 6 is available. But there could be more and to reduce the time spent doing them all individually let's set up a sniper attack.

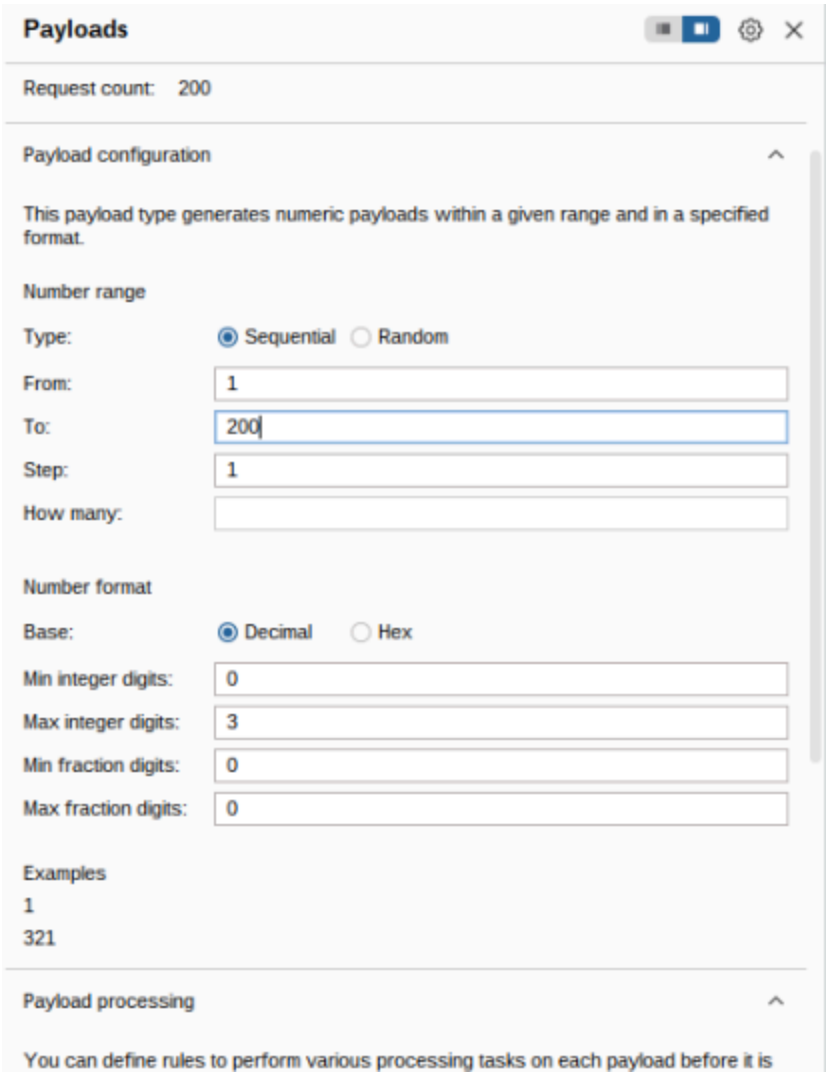
`GET /support/ticket/6 HTTP/1.1` this is what is shown in the intruder window after sending the GET request.

`GET /support/ticket/§6§ HTTP/1.1`

I change it to select the number within the section symbol: §



I will select a numbers file form the number 1 - 200.



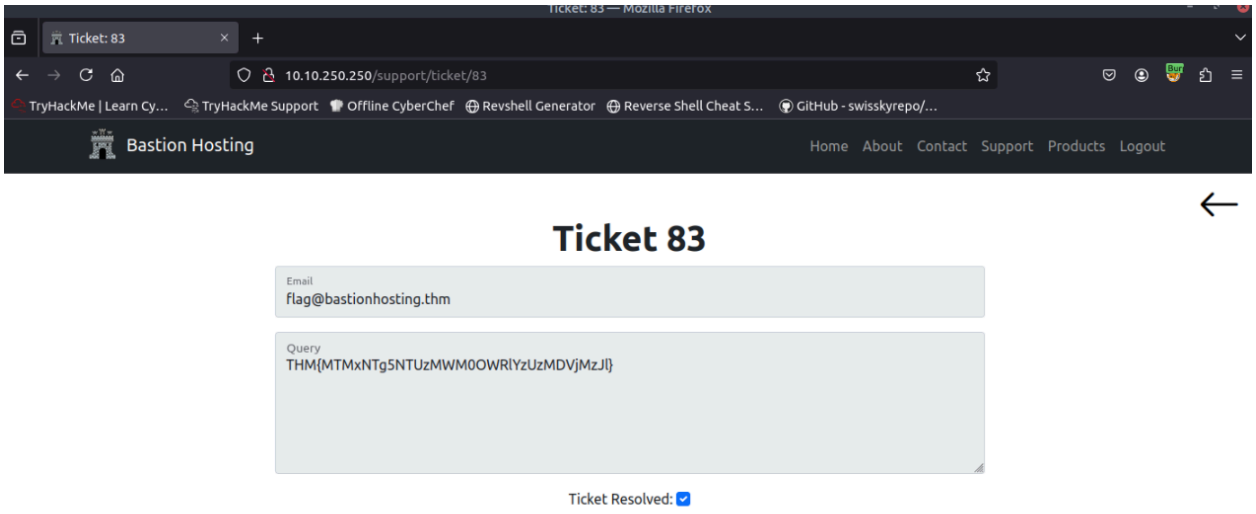
Going 1 step at a time I assume just means going 1 at a time.. So let's see what happens.

So already seeing the progress we have some with the status code of 200 which is good! And 404 means it doesn't exist.. Not so good. But that's okay!

0		200	4
1	1	404	7
2	2	404	4
3	3	404	5
4	4	404	4
5	5	404	5
6	6	200	4
7	7	404	4
8	8	404	4
9	9	404	5
10	10	404	4

See already we picked up on an error of doing it manually I didn't think of putting 0, which means we missed it, and 6 we know was correct because that was a baseline for us, the first one we discovered doing it manually.

So let’s make a table of our findings:
0 = OK
6 = OK
47 = OK
57 = OK
78 = OK
83 = OK
By the looks of it, those are all the results we got back, 6 positive results. Now all we need to do, turn intercept off and manually type those values in to find the flag. This is a good time to mention, doing this manually we would have probably been done by now.. But let’s say that there was 100s or 1000s of different IDs, doing it manually would take time, and like me, I could overlook some values very easily, so this negates human error but also identified it in a list you can directly refer from.

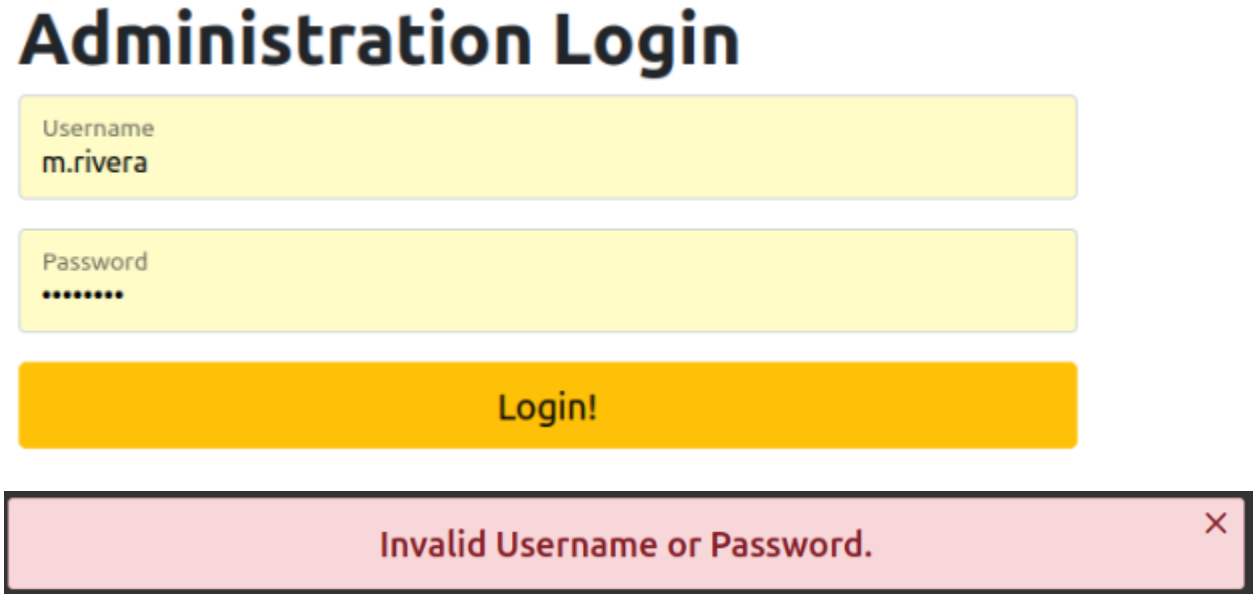


There we go, in ticket 83 is my sweet, sweet flag.
THM{MTMxNTg5NTUzMWM0OWRlYzUzMDVjMzJl}

Another Challenge:

In this extra-mile exercise, we will tackle a more challenging variant of the credential-stuffing attack we previously performed. However, this time, additional measures have been implemented to make brute-forcing more difficult. If you are comfortable using Burp Macros, you can attempt this challenge without the instructions below. Otherwise, let's proceed with the step-by-step approach.

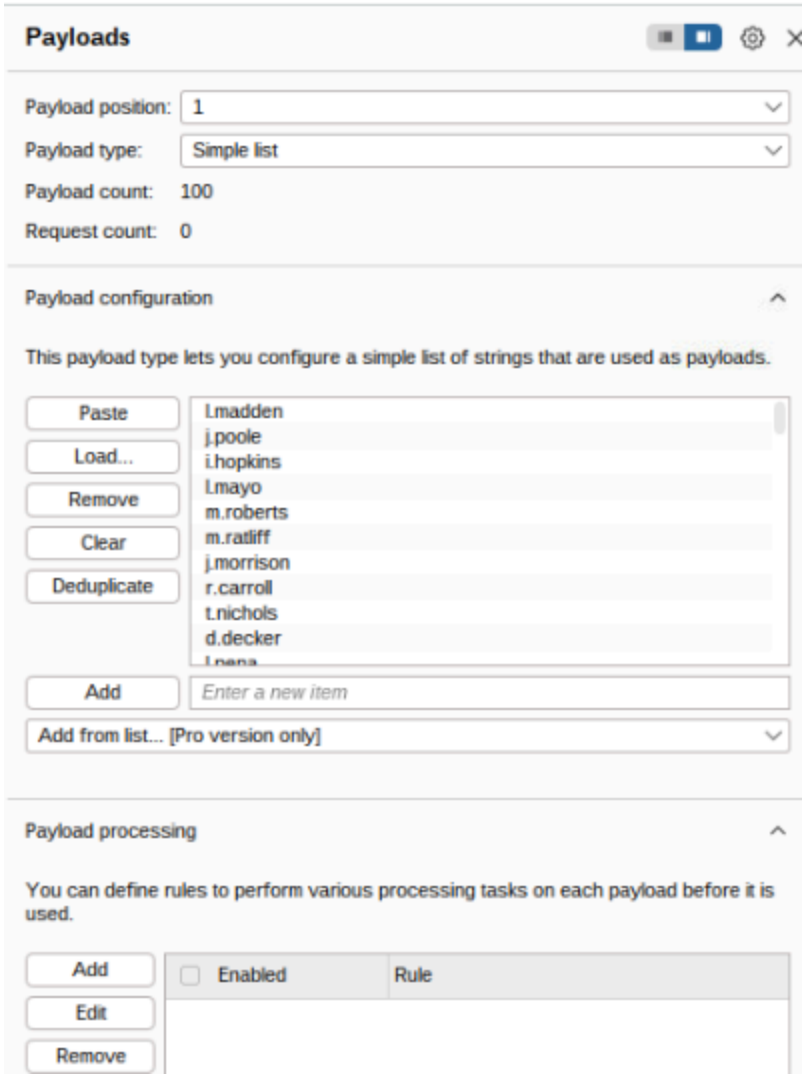
So we need to navigate to http://10.10.250.250/admin/login/



Worth a try seeing as it was just sitting there! (This was from our previous attempts).
First I am going to turn intercept on and get a grab of this page, and because it is now a post request

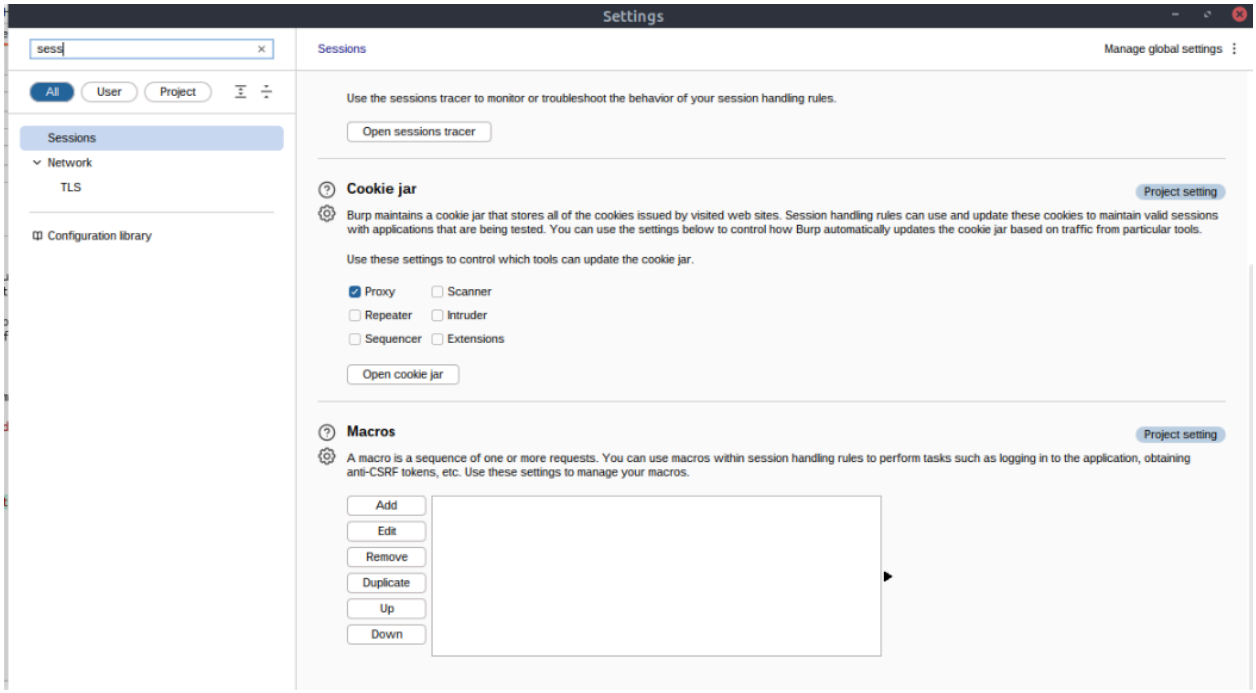
Because I tried to login already to no success but this allows me to manipulate it. So right click and send to the intruder!

I have configured everything and now we will load the same username and password list from before to test out the credentials again and try find someone with some significance to get past the admin authentication.

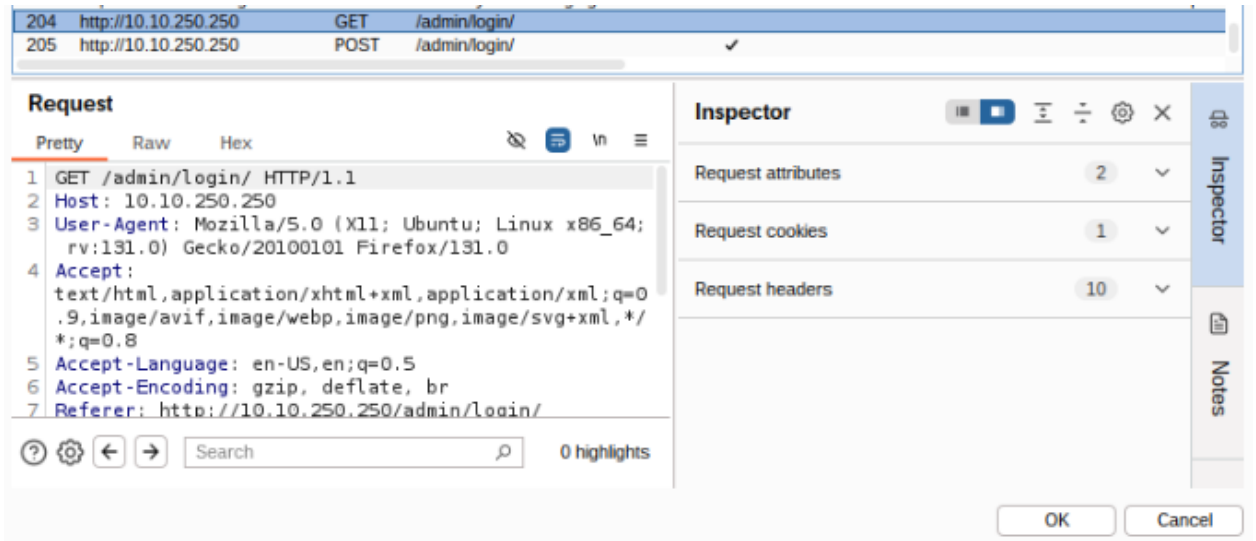


Same process as last time just loading the username.txt file and password.txt file on separate payload positions and then I will starts the attack.

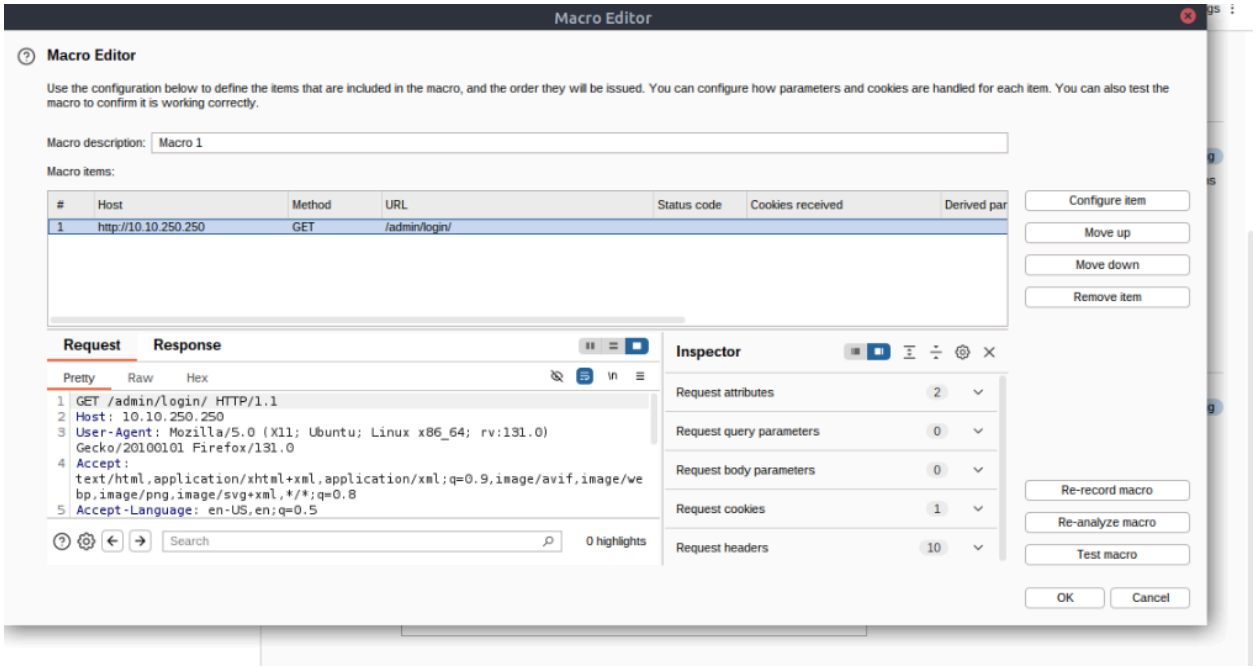
But we now need to find a way to to grab the changing login token and session cookie. Recursive grep won't work here due to the direct response. So we will need to build a "Macro". This will allow us to perform the same set of actions repeatedly. In this case we simply want to send a GET request to /admin/login/.


Settings
sess
All
User
Project
Sessions
Network
TLS
Configuration library
Sessions
Manage global settings
Use the sessions tracer to monitor or troubleshoot the behavior of your session handling rules.
Open sessions tracer
Cookie jar
Project setting
Burp maintains a cookie jar that stores all of the cookies issued by visited web sites. Session handling rules can use and update these cookies to maintain valid sessions with applications that are being tested. You can use the settings below to control how Burp automatically updates the cookie jar based on traffic from particular tools.
Use these settings to control which tools can update the cookie jar.
Proxy
Scanner
Repeater
Intruder
Sequencer
Extensions
Open cookie jar
Macros
Project setting
A macro is a sequence of one or more requests. You can use macros within session handling rules to perform tasks such as logging in to the application, obtaining anti-CSRF tokens, etc. Use these settings to manage your macros.
Add
Edit
Remove
Duplicate
Up
Down

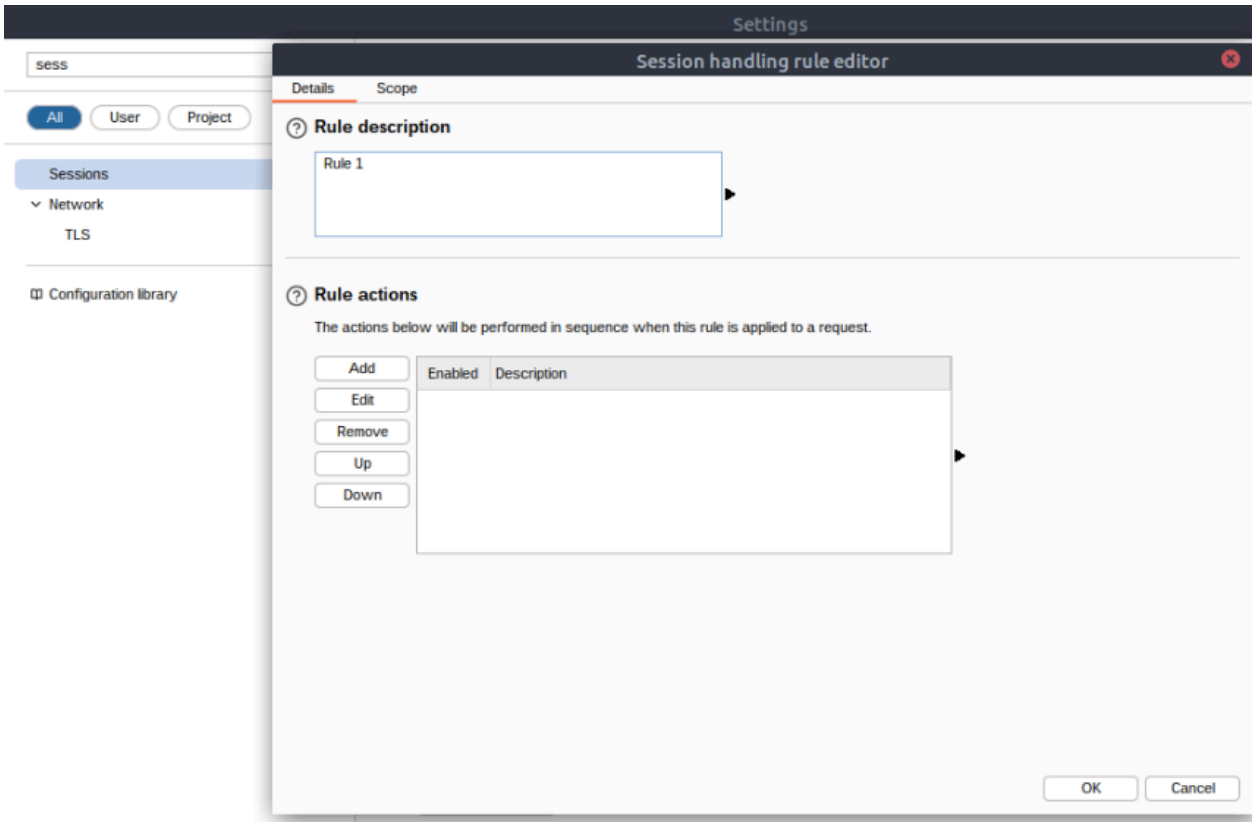
“Add” -


204 http://10.10.250.250 GET /admin/login/
205 http://10.10.250.250 POST /admin/login/
Request
Pretty
Raw
Hex
1 GET /admin/login/ HTTP/1.1
2 Host: 10.10.250.250
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:131.0) Gecko/20100101 Firefox/131.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://10.10.250.250/admin/login/
Search
0 highlights
Inspector
Request attributes 2
Request cookies 1
Request headers 10
Inspector
Notes
OK
Cancel

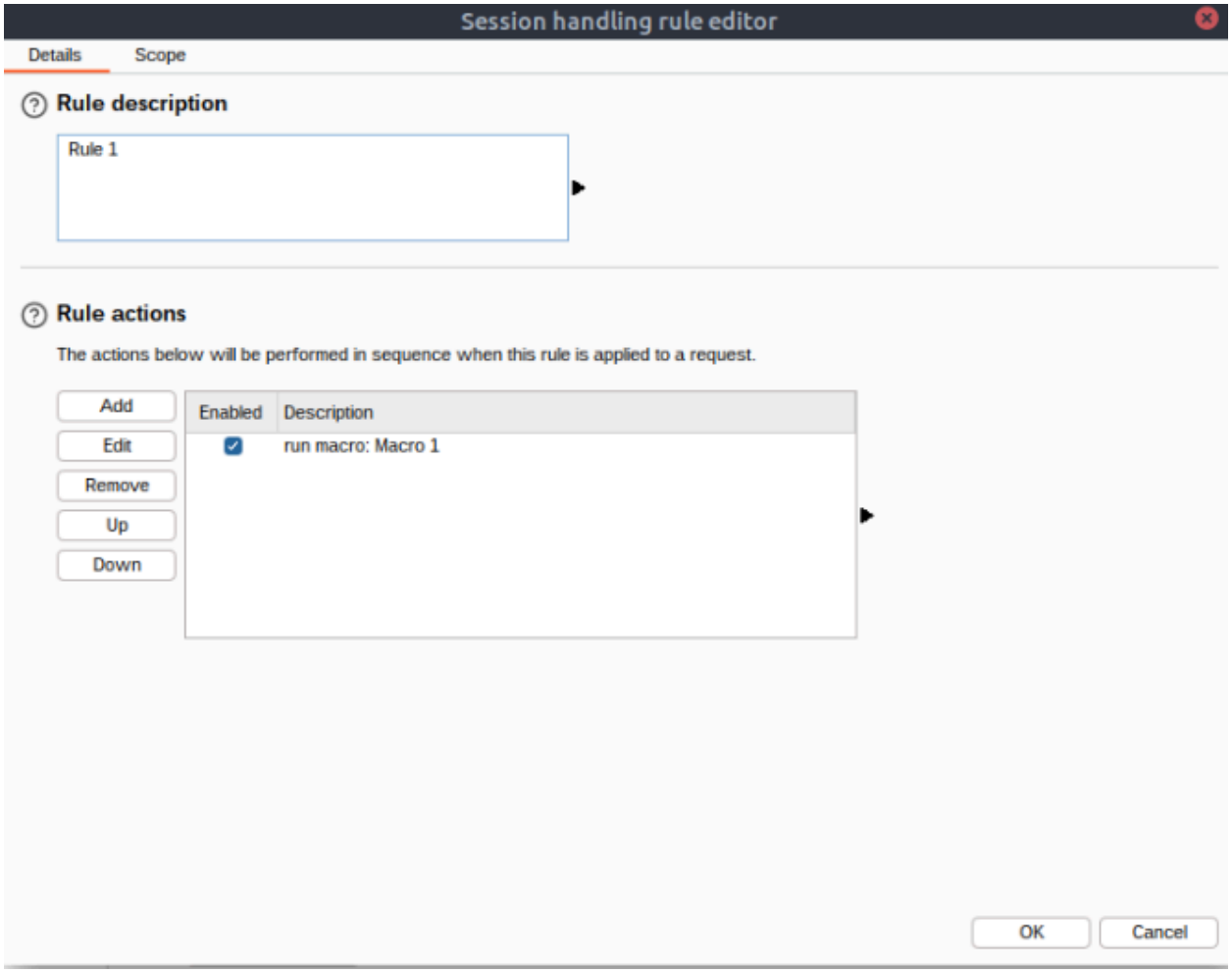
“Ok”



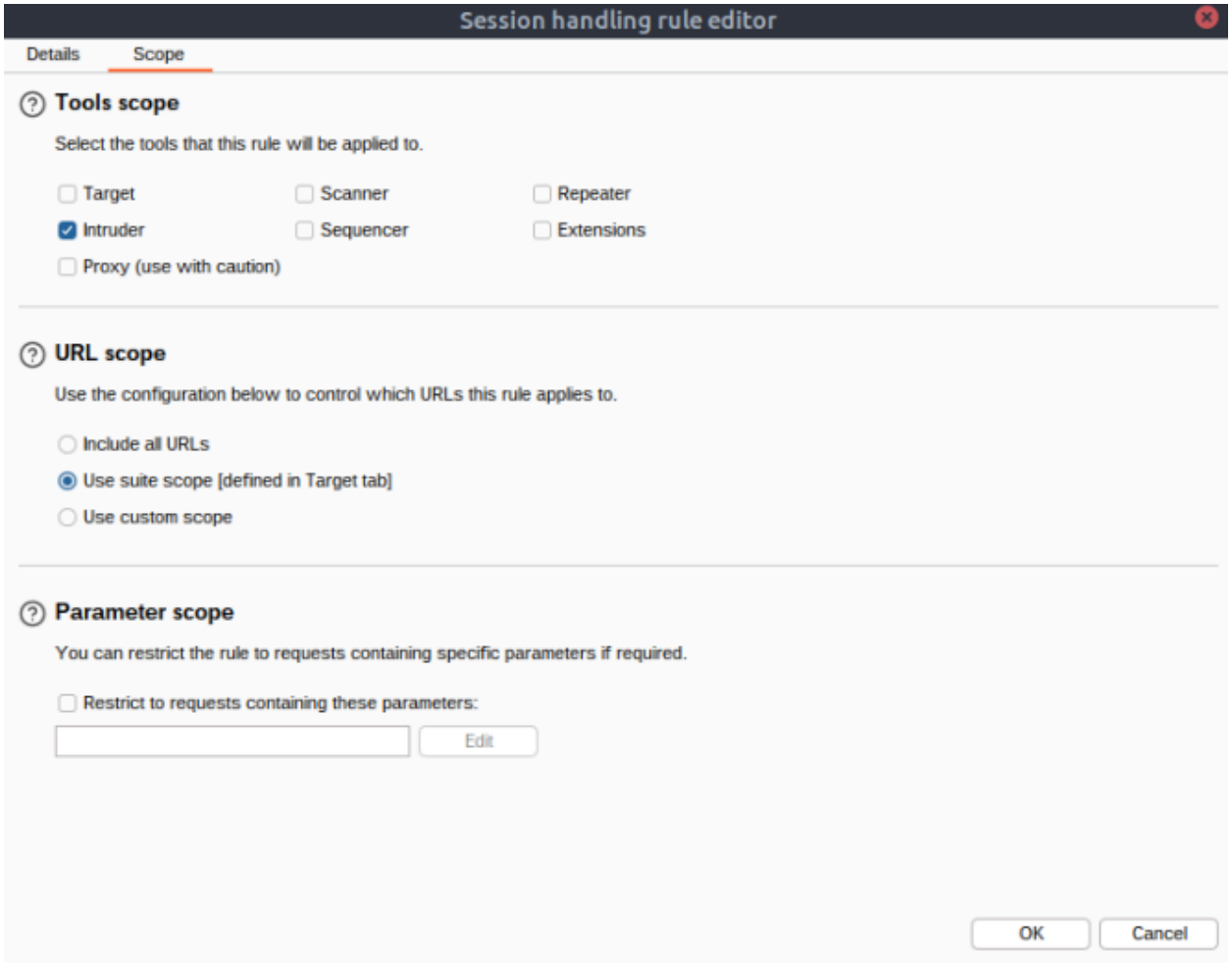
“Ok”



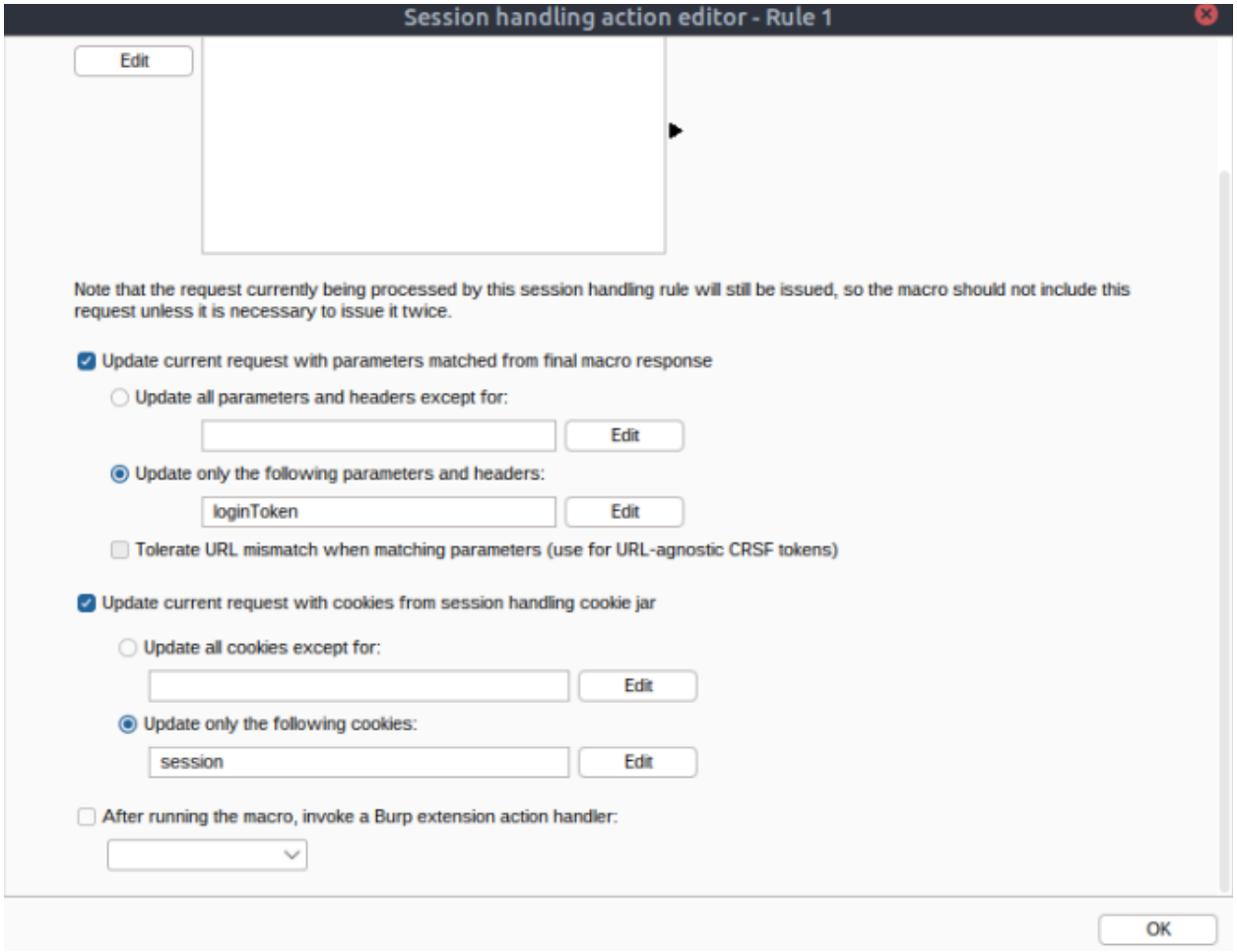
”Add” New rule - Details and scope.



Run macro: Macro 1.

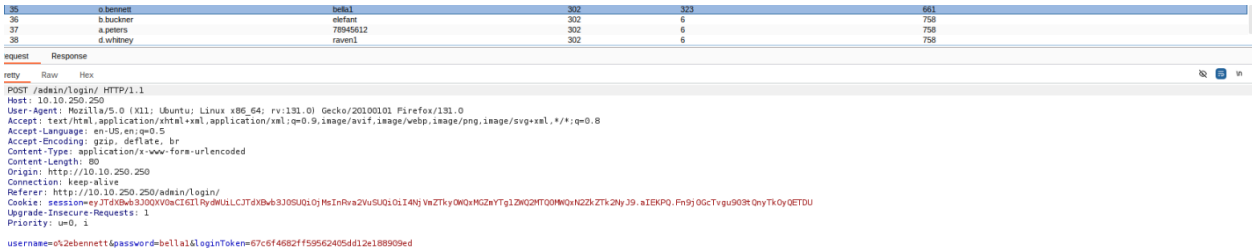


Configure scope settings.



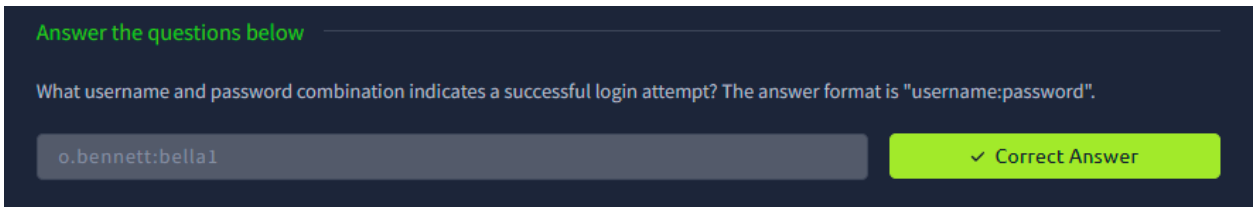
Update Macro settings to update only the following parameters and headers “loginToken” and “sessions”..

Selecting okay and ! Ready! Now we run it as usual. Getting 302 responses from the server.



Boom! We found one!
Username: o.bennett
Password: bella1
Let’s now see if we can use this to login.

We did it!



Conclusion:
This was certainly more advanced than the repeater, using Macros and really piecing bits of information together using wordlists can really generate some sensitive information. I love how using these techniques we can bypass security features such as cookies and login tokens. It just goes to show how effective tools and applications can be when you know how to configure them. This was very enjoyable, to see how different attacks can be performed, and how to overcome issues should they arise.