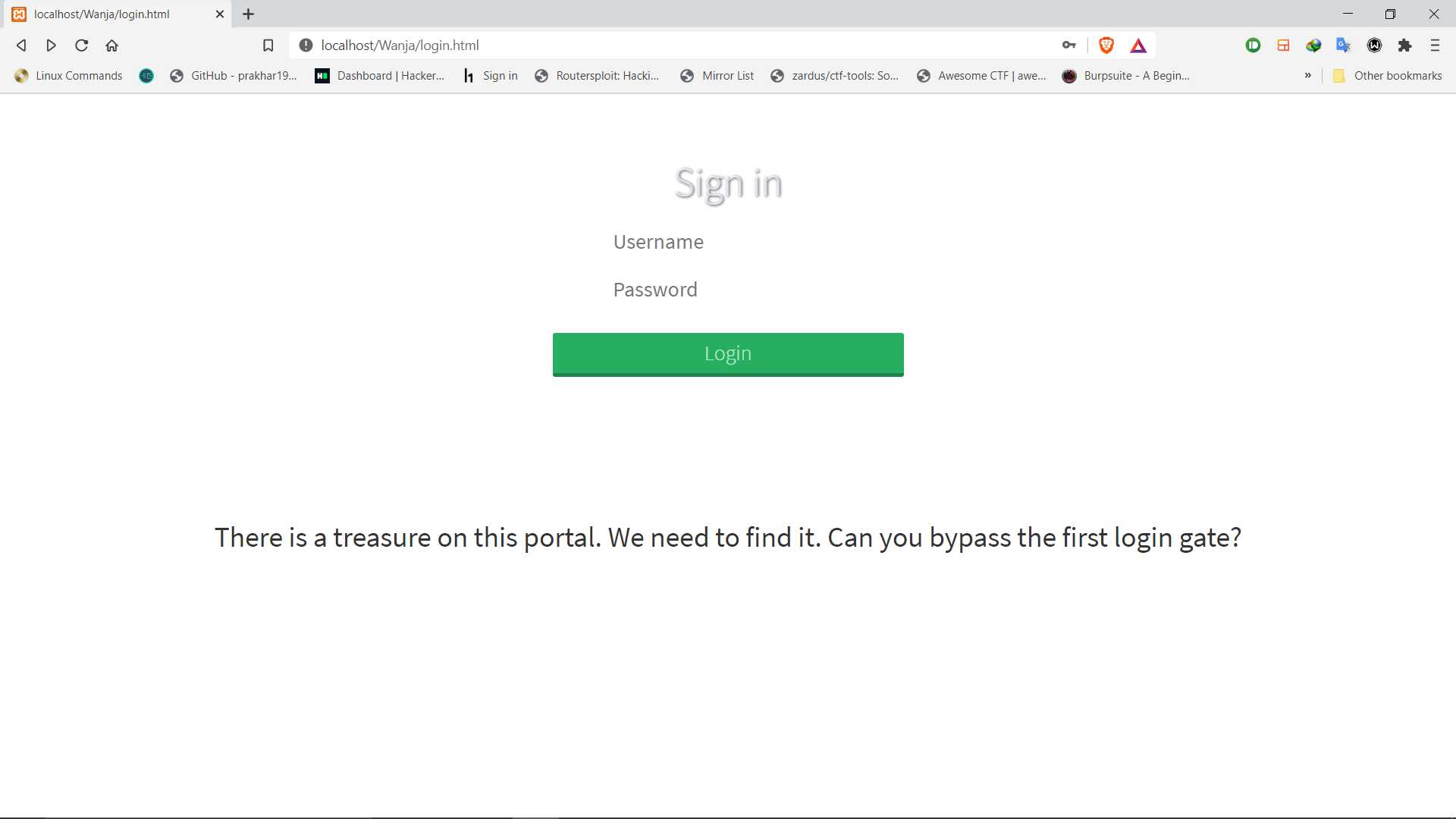
**WINJA CTF SCREENING ASSIGNMENT**

**CTF Challenge Write-up**

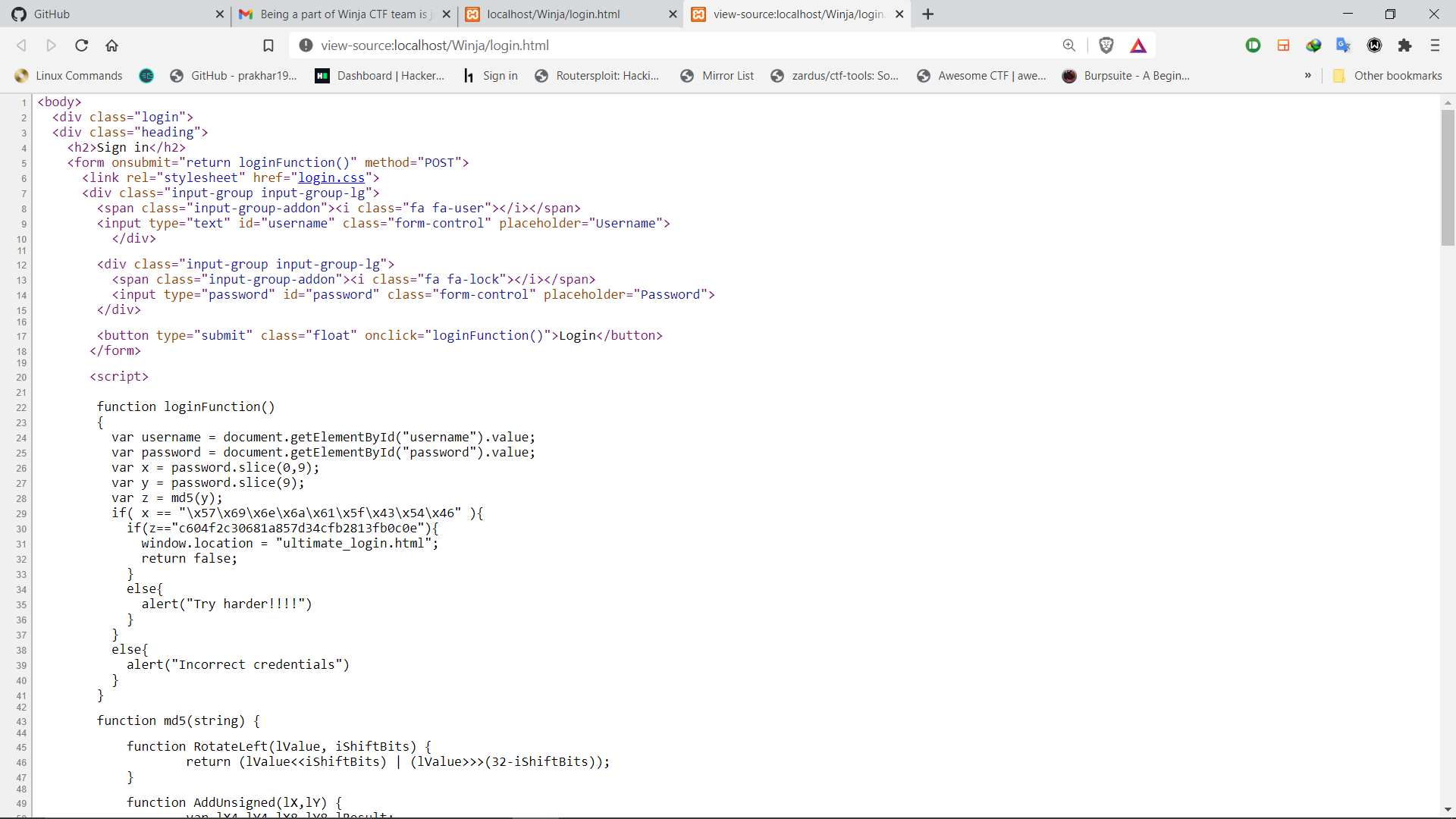
**ANEESH DUA**

**(**[workstuff.dua@gmail.com](mailto:workstuff.dua@gmail.com)**)**

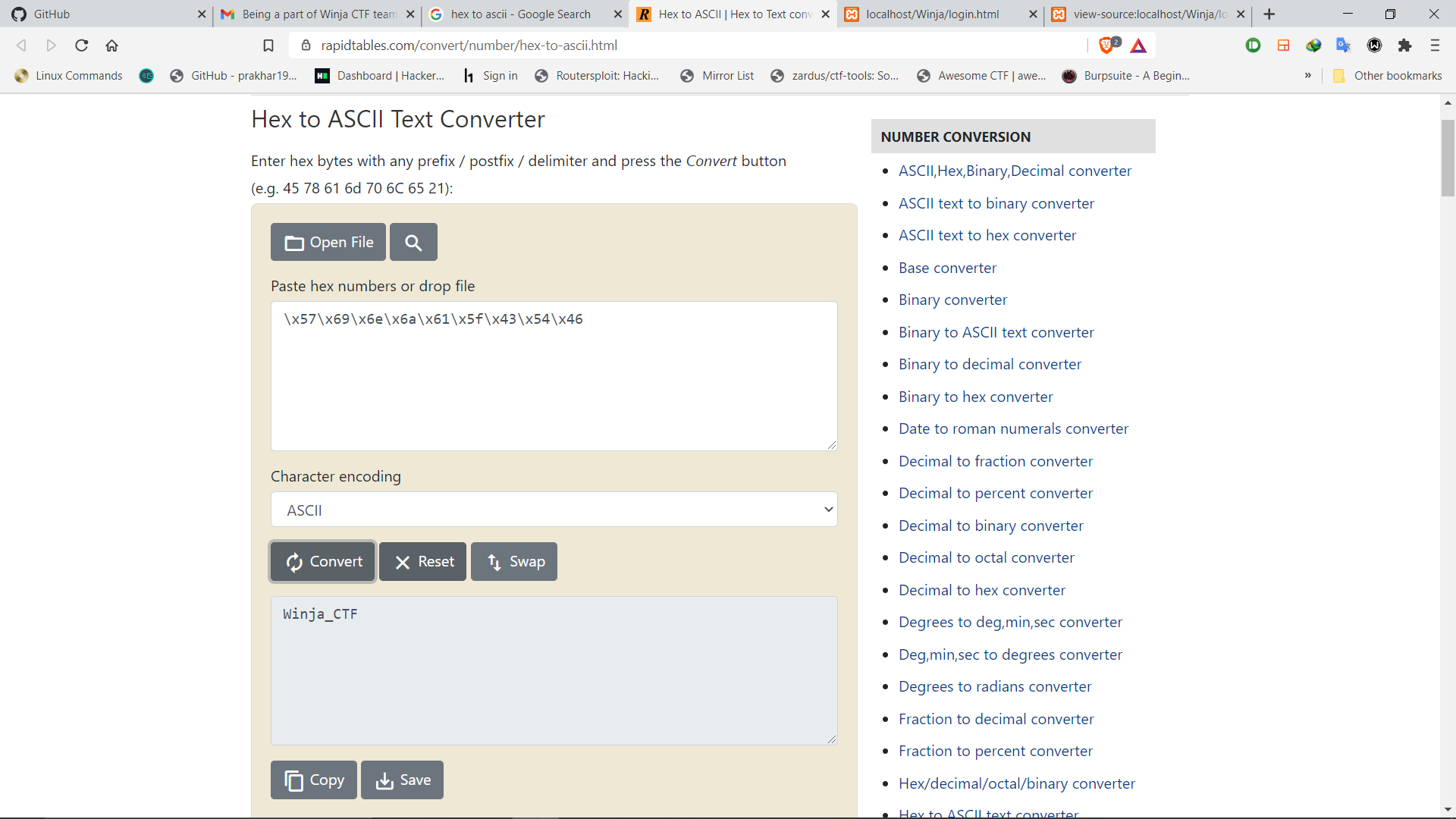
**Level 1: HardCoded Password Leak - Login Bypass**

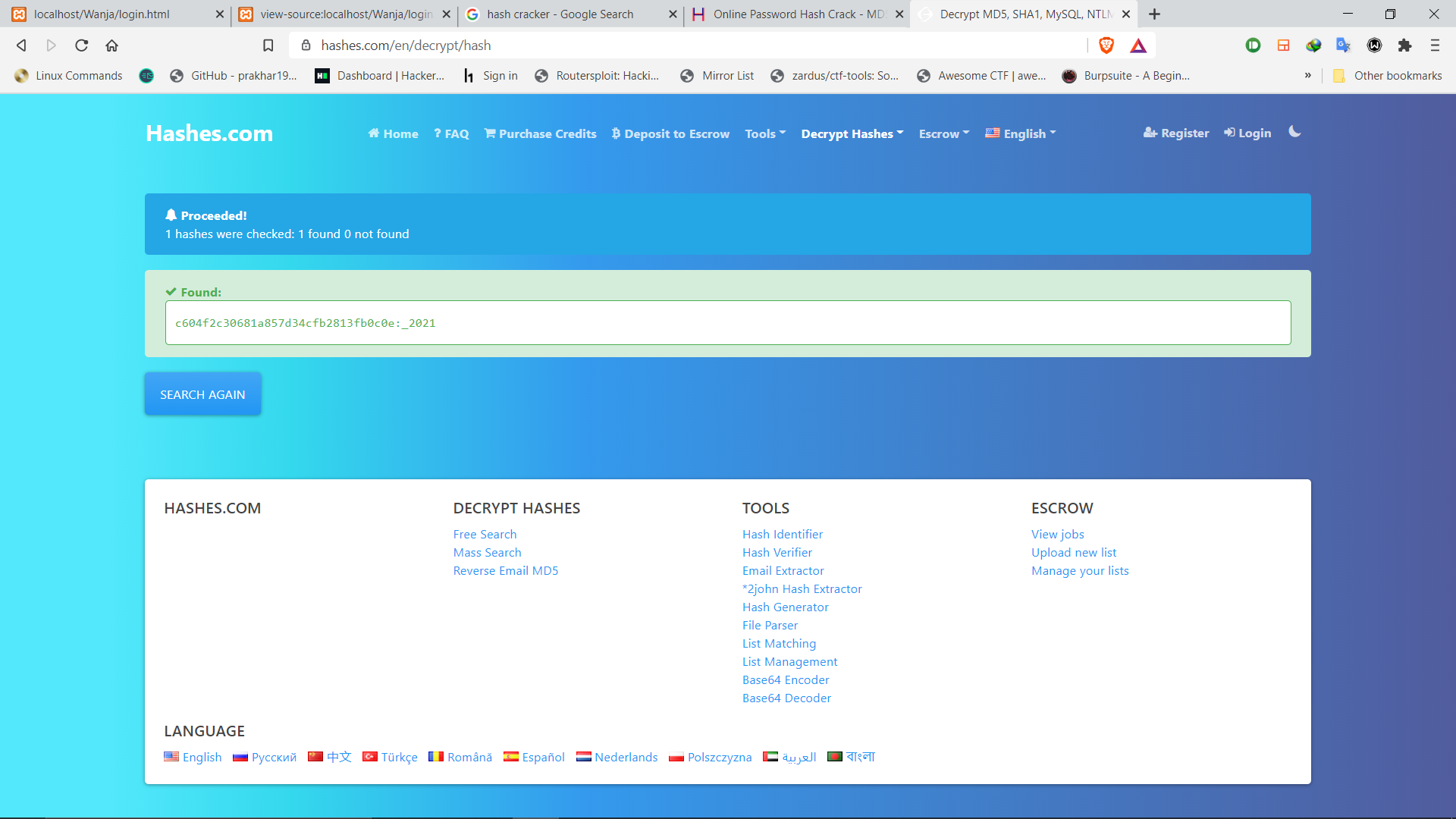


1. On visiting the challenge URL, we are presented with a login page. On inspecting the code using Ctrl+U or by simply right clicking and Select View Page Source, we find the logInFunction() inside the *script* tags of the page. This function validates the username and password credentials.



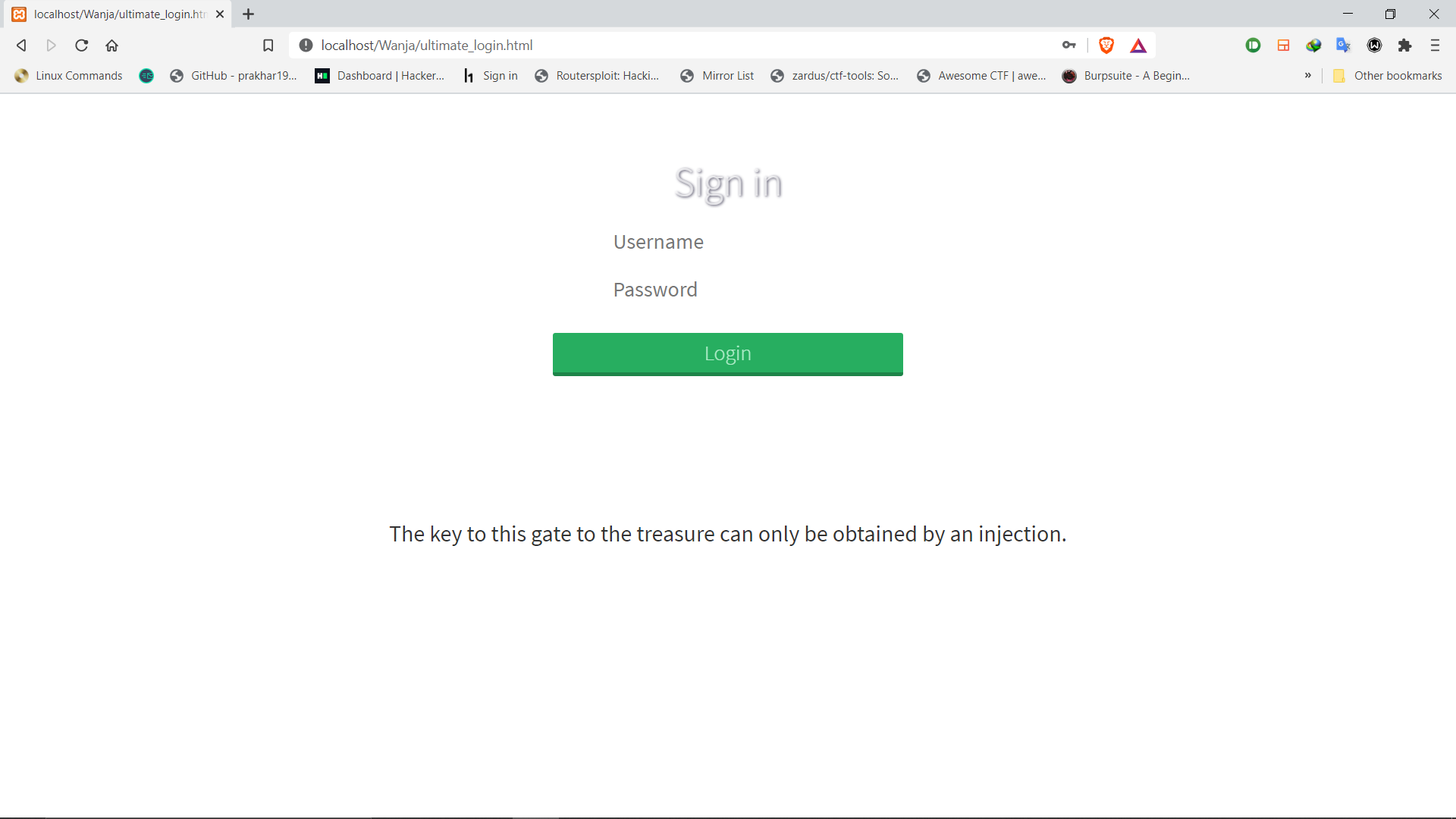
1. Assessing the code workflow, we understand that we **DO NOT** require a username for logging in as the logInFunction() does not process the username value. As you can see this javascript function is taking the input from the login form and the password is being sliced in 2 parts. The first part is being stored in x variable and getting compared with the hex value. Where as the another half of the password is stored in y variable and being compared with the MD5 hash of the string. Let’s decode both of them.





1. We have successfully got the password, now we can login by combining both of them. The password is: Winja\_CTF\_2021.

**Level 2: SQL Injection**



1. We are redirected to the “ULTIMATE” login portal and we are required to bypass it. On inspecting the code same as before we are unable to find anything in the code. Trying default username-password combinations like admin-admin, root-toor etc also does not work.
2. However, we are given a hint related to injection. This directs us to the common SQL Injection vulnerability. Hence we try injecting the username field with the string ' OR 1 = 1 LIMIT 1 -- ' and any random string as password. This injection works when the username input is not sanitised. For example- For the SQL query

SELECT \* FROM users WHERE username = '$username' AND password = '$password'

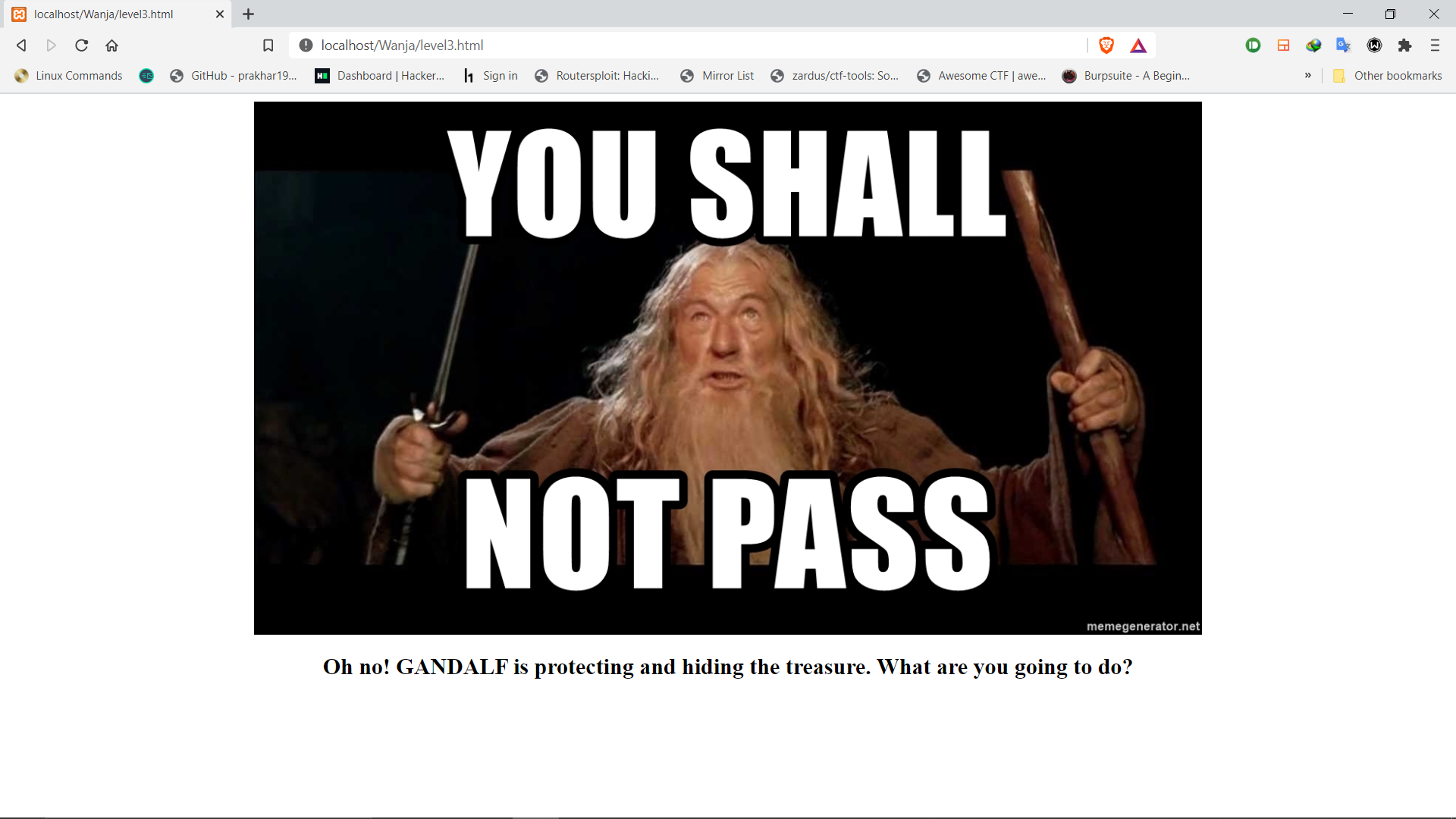
The injected string makes it –

SELECT \* FROM users WHERE username = ' OR 1 = 1 LIMIT 1 -- '  AND password = abc

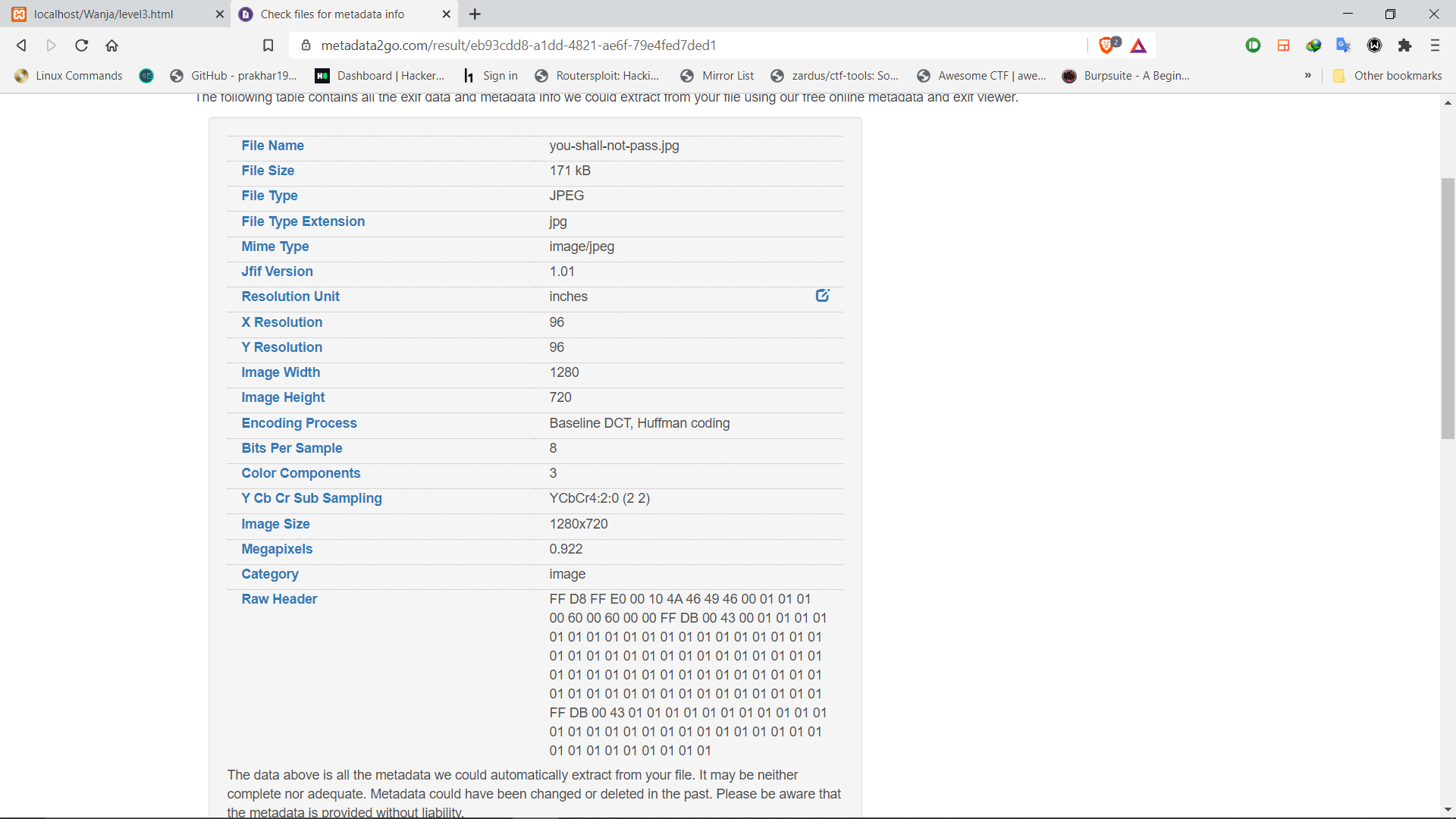
Thus, resulting in a True return for the query as 1=1 is always true. Thus, we bypass this level also.

**Level 3: Steganography**

1. We are redirected to an html page with a picture and a hint that Gandalf is hiding the treasure.

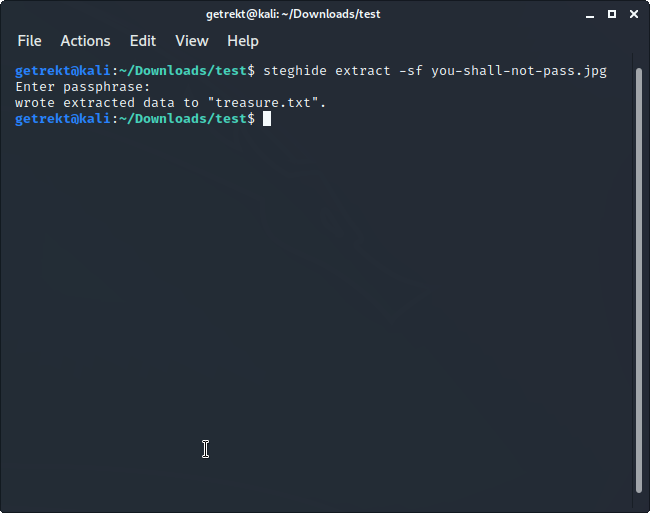


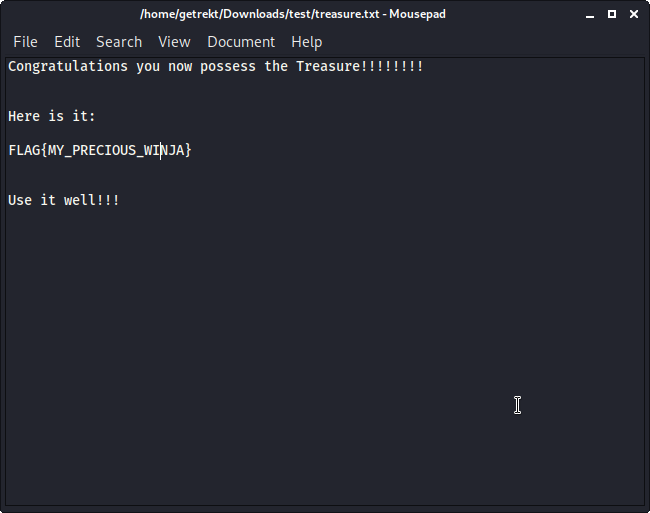
1. On initial inspection of the website source code as before we can conclude that we have only the image and the hint to go on.
2. We download the image and inspect its meta-data. But still we are unable to find anything.



1. Focusing on the hint, we are told that Gandalf is “hiding” the treasure. Thus, we try extracting data using reverse steganography using the popular tool [steghide](http://steghide.sourceforge.net/documentation/manpage.php).

We use the command *steghide extract -s you-shall-not-pass.jpg*. When asked for the passphrase, we use the biggest hint in the level – Gandalf.





1. Voila! All challenges completed and the precious flag is given to us in the treasure.txt file. Hope Gandalf learns that he shouldn’t use his own name for the secret passphrase!