Minor Project

-By Nikhil Nettar

**Overview**

Modern websites use databases to store and manage information. We want to better understand the framework that makes the connection between a webpage and a database possible. And will try to exploit the known vulnerability to get access into the database and find out more information.

**Objectives**

We will learn about one of the worst known vulnerabilities, SQL injection. This exercise will help us to explore web-application security.

**Target URL**

<http://lab.hackerinside.xyz>

**Finding out the vulnerability**

## 

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behaviour.

In some situations, an attacker can escalate an SQL injection attack to compromise the underlying server or other back-end infrastructure, or perform a denial-of-service attack.

We can see that the target URL is http website which uses the post method to retrieve data. Therefore we can exploit this by using a proxy which in this case is Burp suite.

Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work seamlessly together to support the entire testing process, from initial mapping and analysis of an application’s attack surface, through to finding and exploiting security vulnerabilities.

Burp Suite contains an intercepting proxy that lets the user see and modify the contents of requests and responses while they are in transit. It also lets the user send the request/response under monitoring to another relevant tool in Burp Suite, removing the burden of copy-paste. The proxy server can be adjusted to run on a specific loop-back ip and a port. The proxy can also be configured to filter out specific types of request-response pairs.

**Exploiting the vulnerability**

We first configure the database properly using the following commands

Sudo –i // To gain root access

cd// Change Directory

Git clone // To copy the required config files to required directory

Mv ls vim and cp //To manage the files in directory

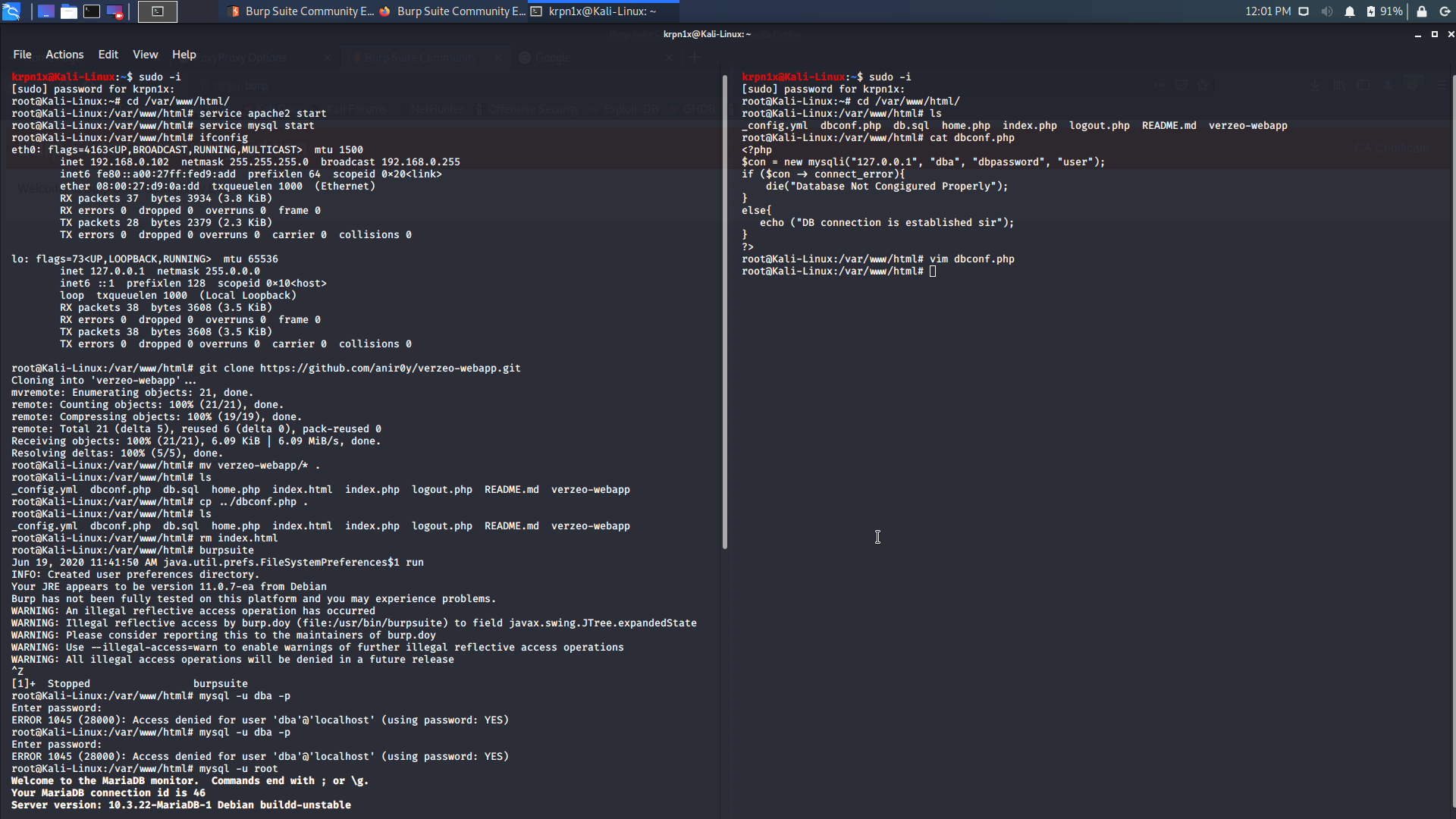
Service apache2server start Service MySQL start// to start the server

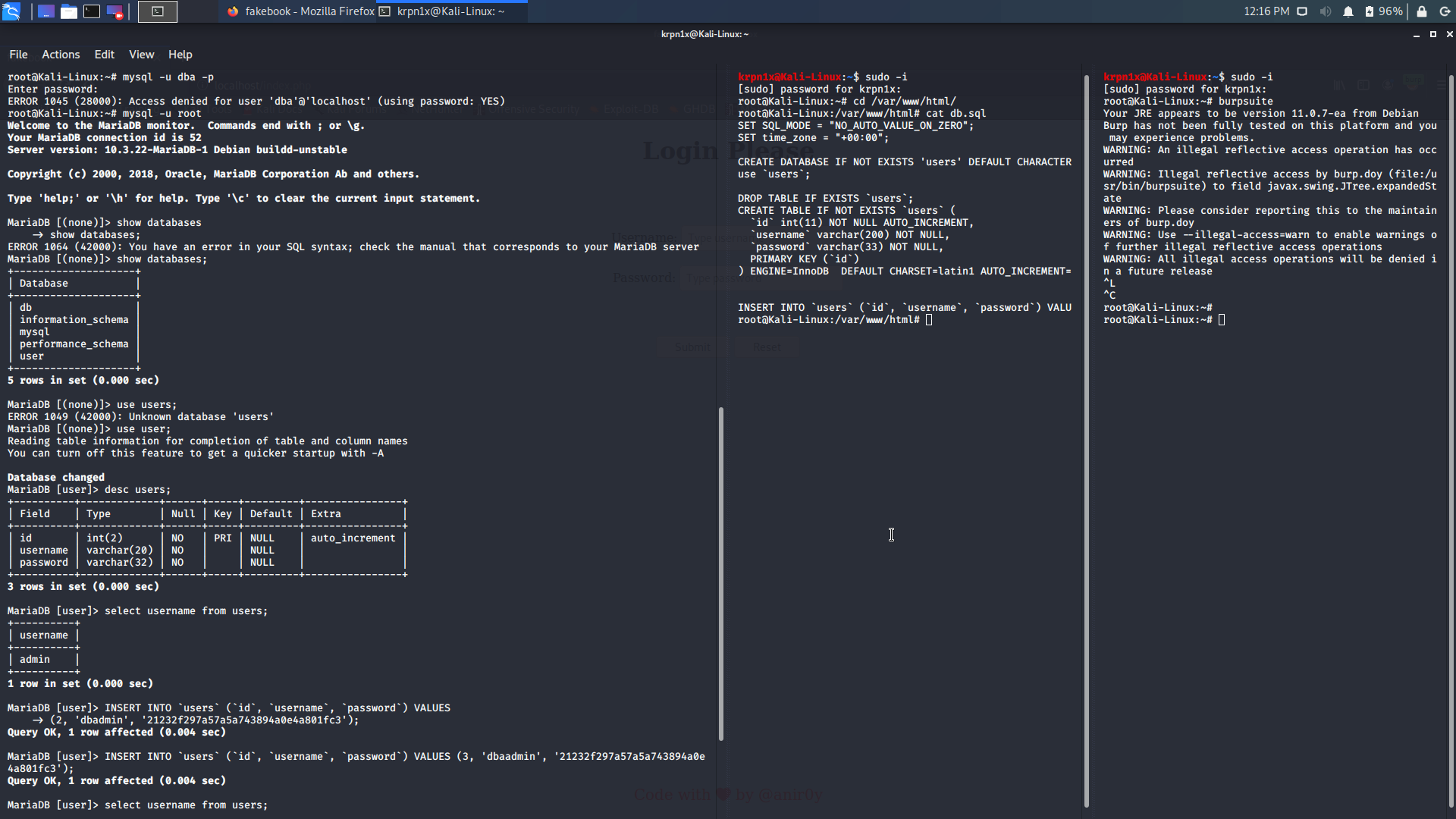
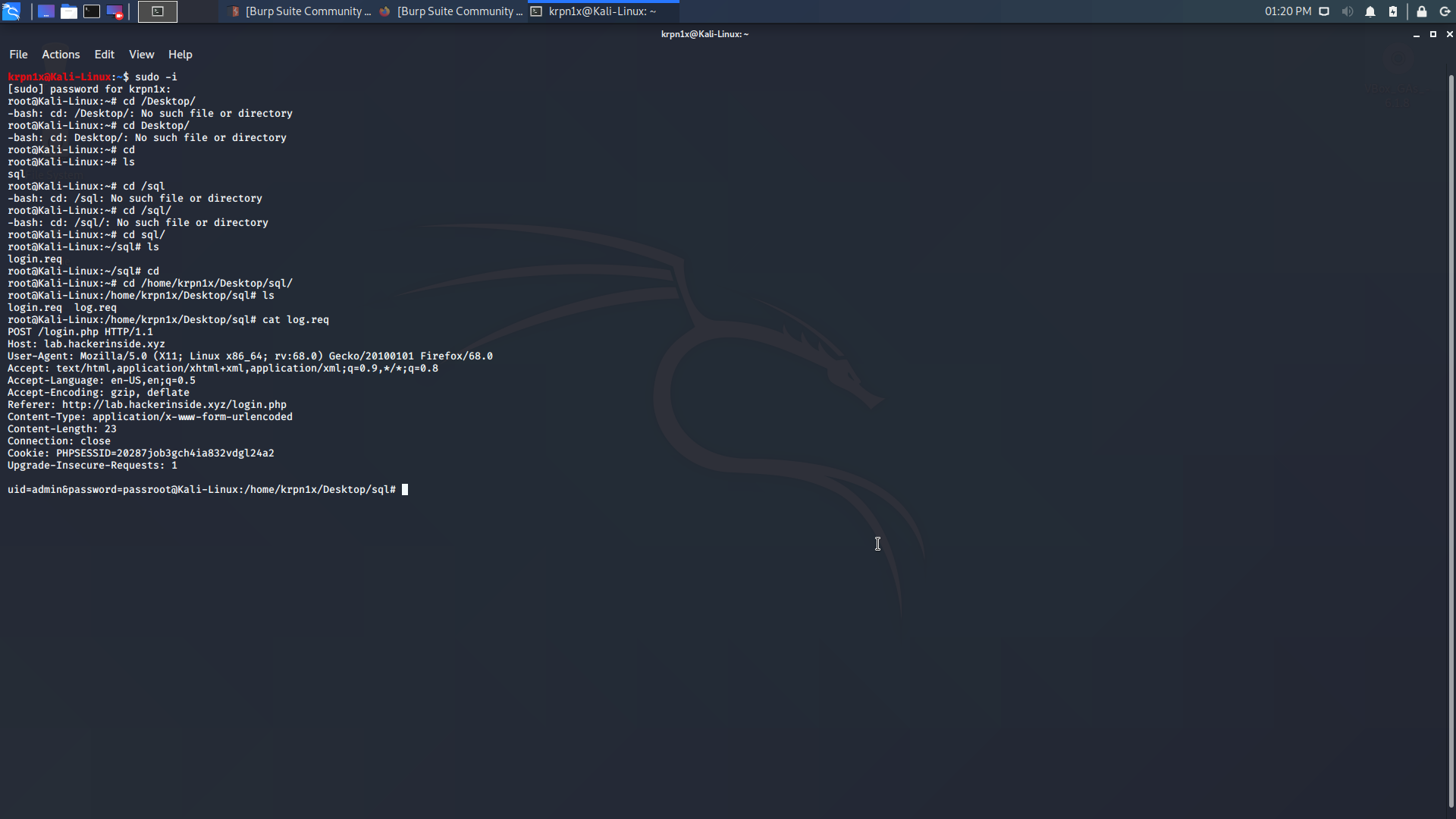
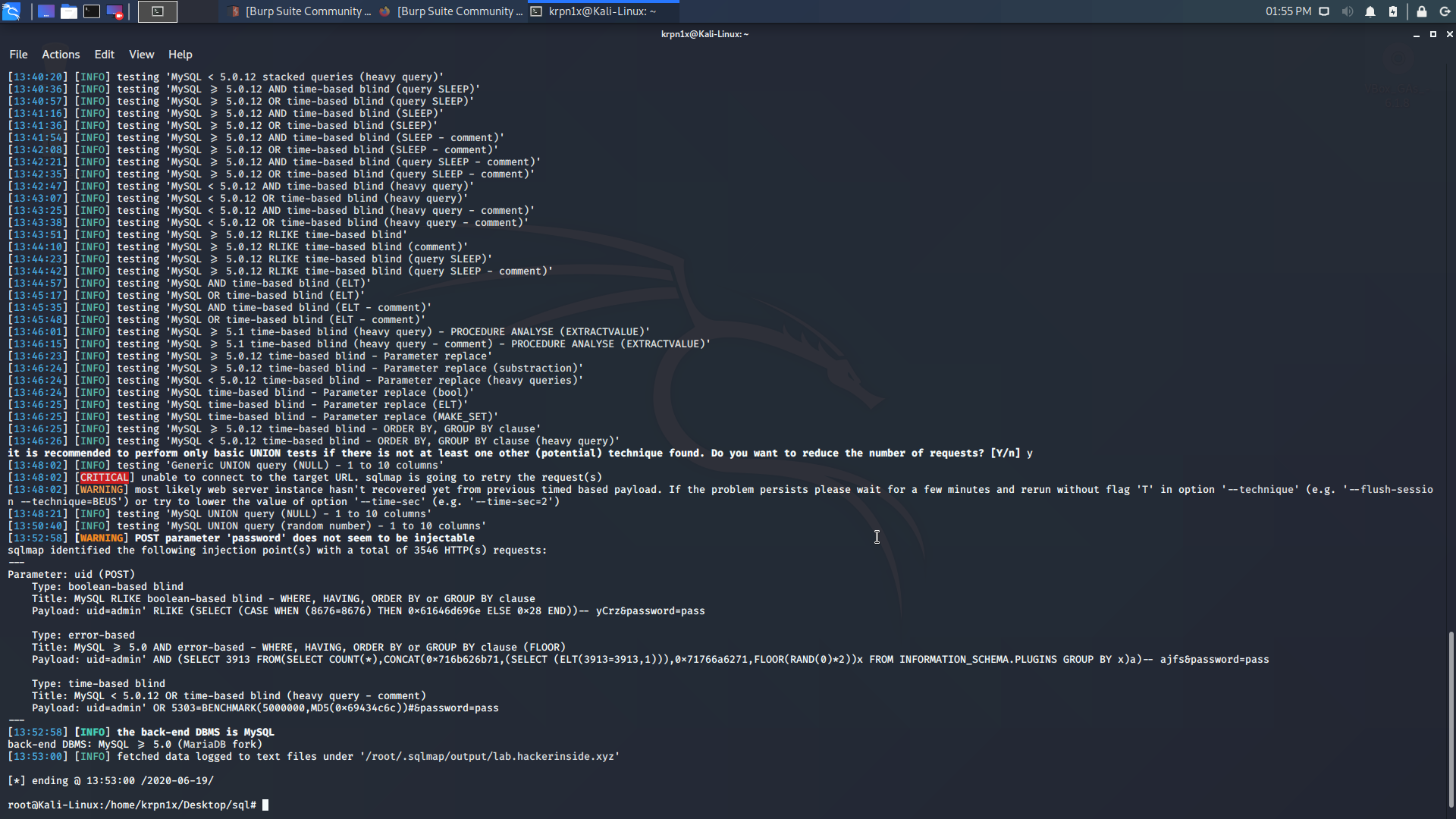
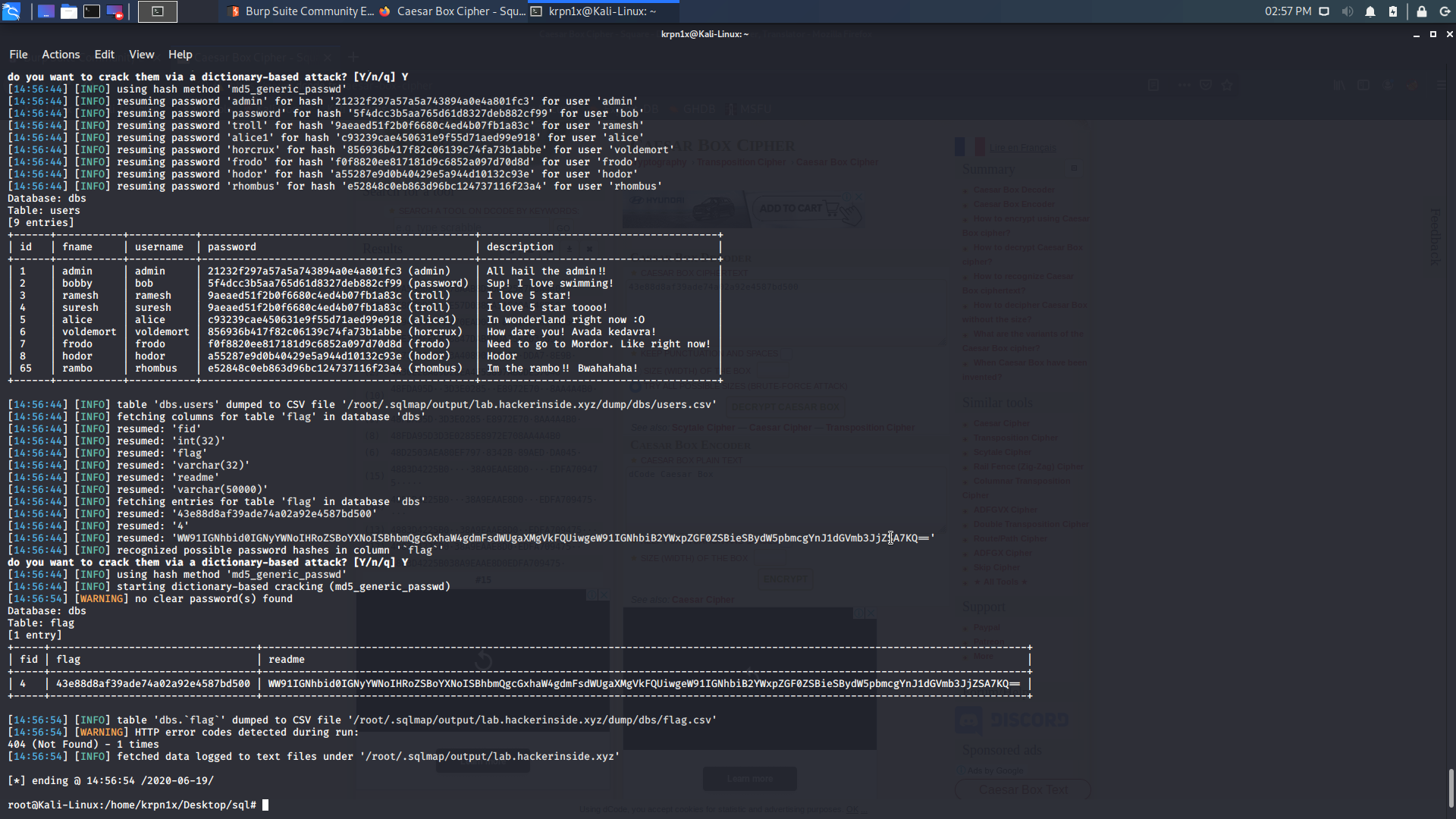
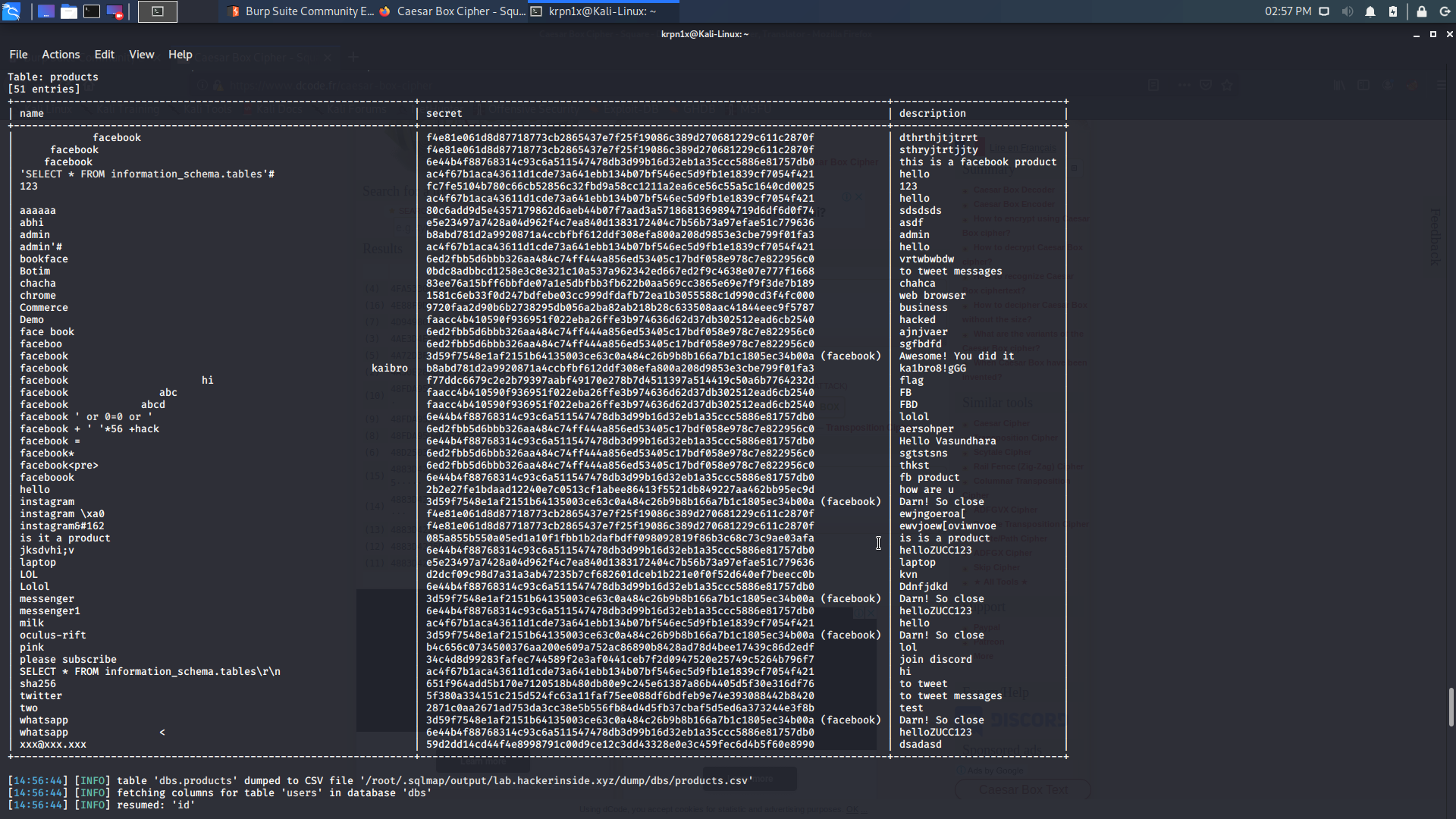
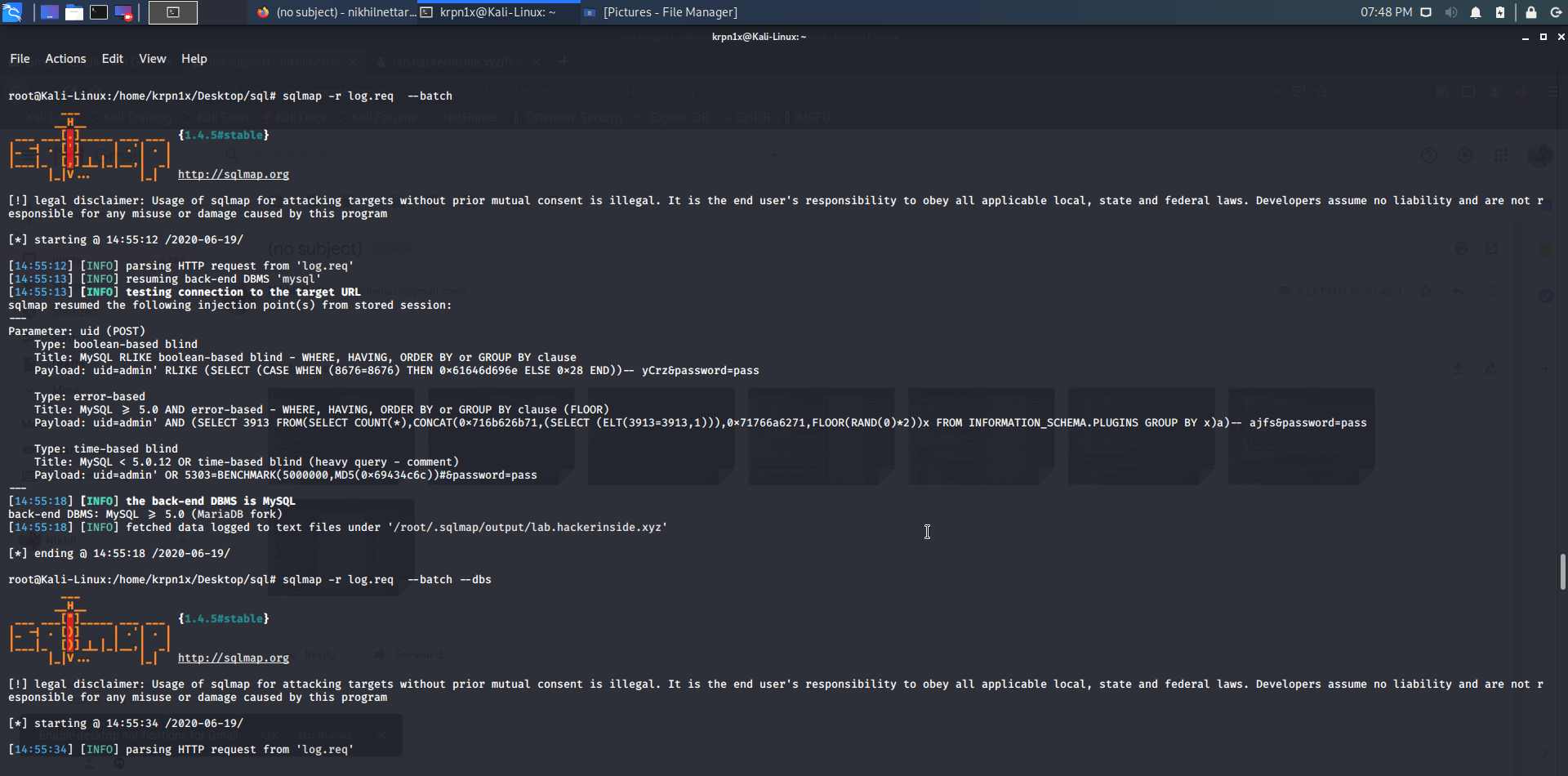
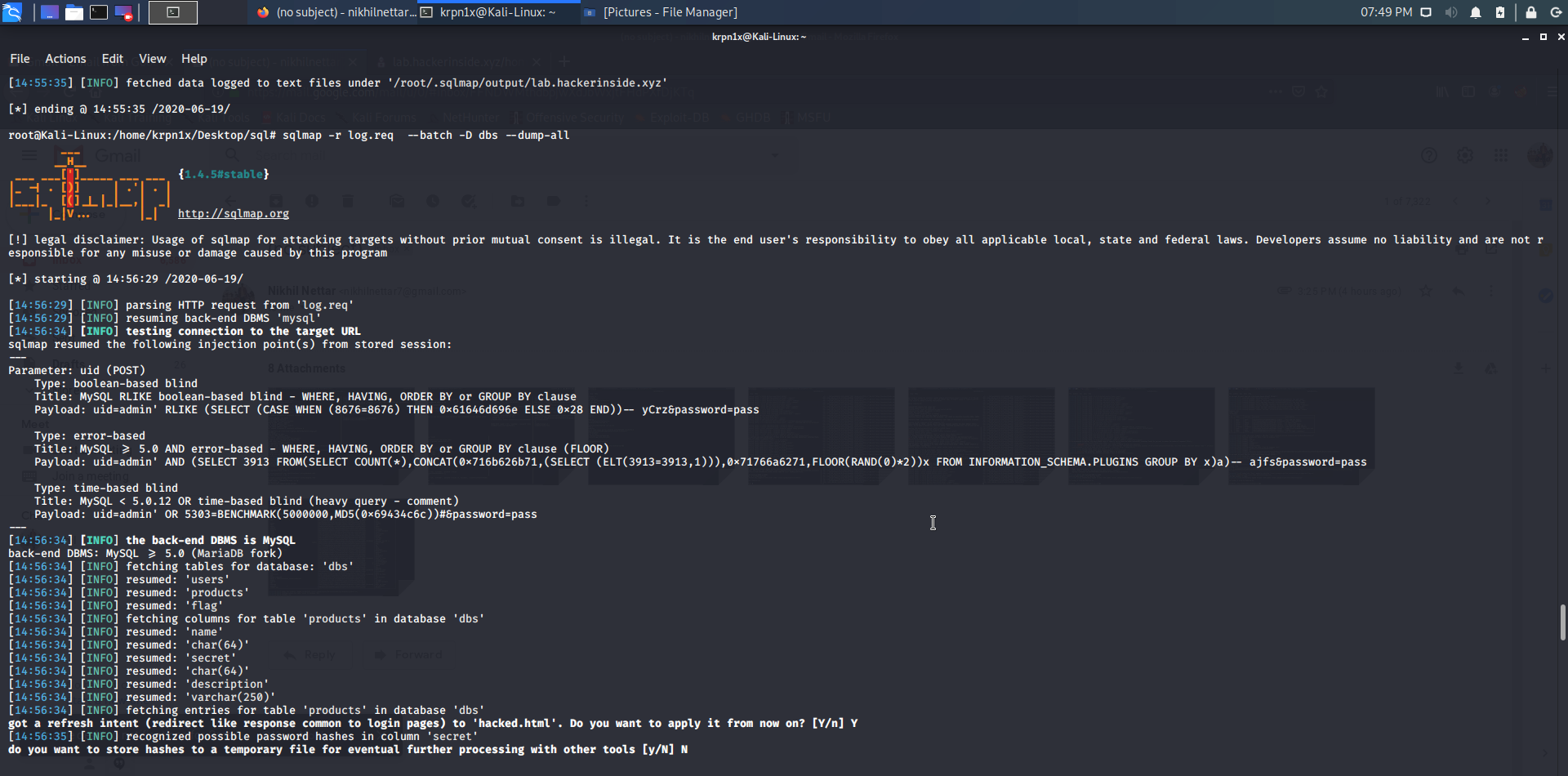
MySQL –u // to manage server database. Once the server database has been set up properly we can configure burpsuite by using the command burpsuite. The burpsuite has to be setup properly in the firebox browser. Once that is done we can open the target URL and then open burpsuite and go to proxy window and switch on intercept. Once this is done we can go to the website and enter any credentials. Now the POST request will be captured in burpsuite. This request is copied to a file named log.req. We use this file as an argument to the sqlmap command.

sqlmap is an open source penetration testing tool that automates the process of detecting and exploiting SQL injection flaws and taking over of database servers.

When we use the sqlmap command using the appropriate options as shown in screenshots we can get access to the data within website database and if we search the tables we can get the required flag.

**Snapshots**





**Captured Request**

The HTTP POST request captured by burpsuite will be on this form

POST /login.php HTTP/1.1

Host: lab.hackerinside.xyz

User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:68.0) Gecko/20100101 Firefox/68.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://lab.hackerinside.xyz/login.php

Content-Type: application/x-www-form-urlencoded

Content-Length: 23

Connection: close

Cookie: PHPSESSID=20287job3gch4ia832vdgl24a2

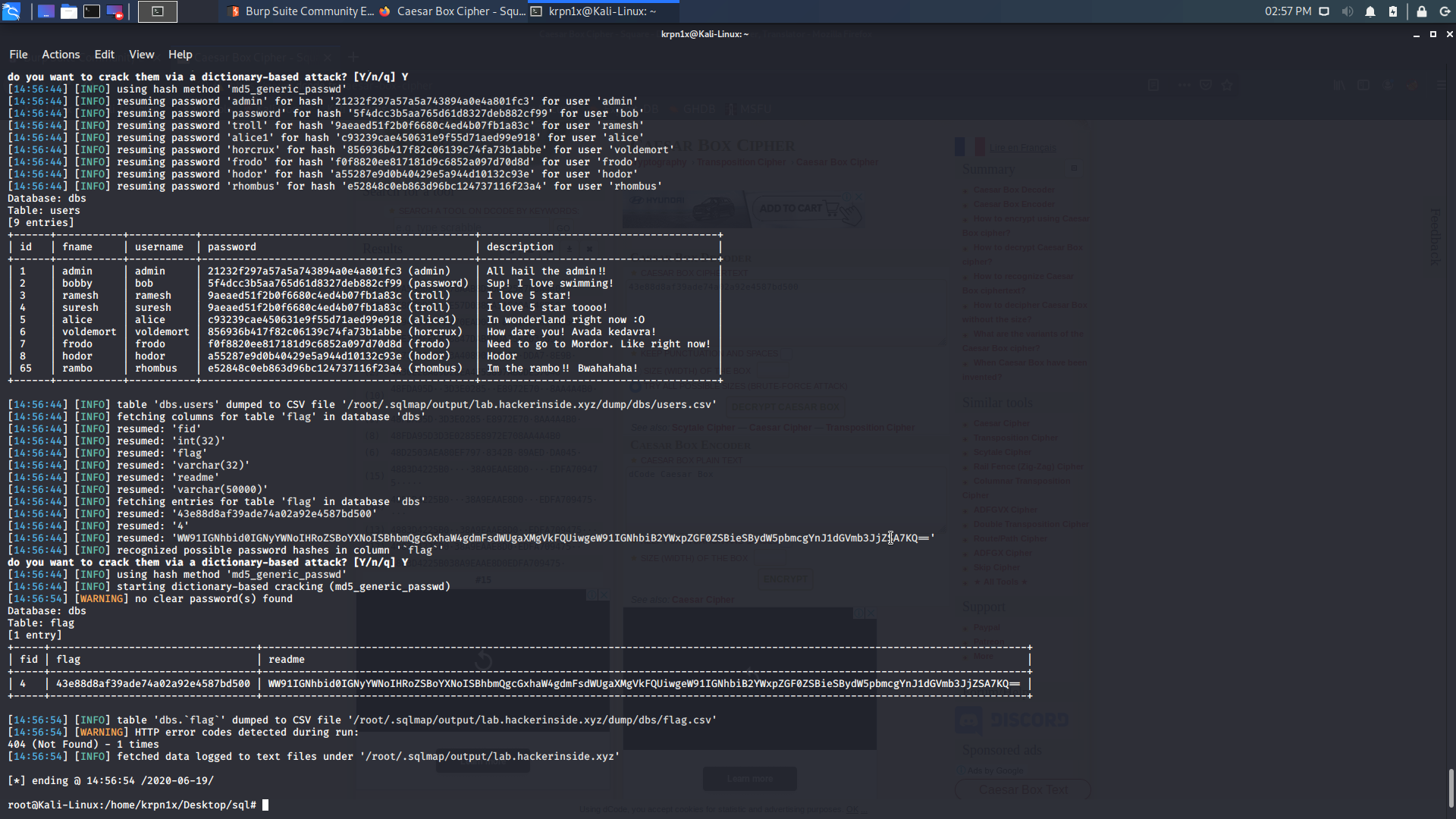
Upgrade-Insecure-Requests: 1

uid=admin&password=pass

**Capturing the flag**

We can see from the screenshot that the captured flag is 43e88d8af39ade74a02a92e4587bd500

We get this by searching the retrieved data for the table named flag in the various databases using sqlmap with the various options.

****

**How to prevent this attack**

[SQL Injection](https://owasp.org/www-community/attacks/SQL_Injection) attacks are unfortunately very common, and this is due to two factors:

1. The significant prevalence of SQL Injection vulnerabilities, and
2. The attractiveness of the target (i.e., the database typically contains all the interesting/critical data for your application).

SQL Injection flaws are introduced when software developers create dynamic database queries that include user supplied input. To avoid SQL injection flaws is simple. Developers need to either: a) stop writing dynamic queries; and/or b) prevent user supplied input which contains malicious SQL from affecting the logic of the executed query.

Most instances of SQL injection can be prevented by using parameterized queries (also known as prepared statements) instead of string concatenation within the query.