A

Project on

SQL Injection

By Ashish Gajjela

Table of Contents

INTRODUCTION

SQL injection is a code injection technique used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

1.1 What is SQL Injection (SQL)?

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.

1.2 What is the impact of a successful SQL injection attack?

A successful SQL injection attack can result in unauthorized access to sensitive data, such as passwords, credit card details, or personal user information. Many high-profile data breaches in recent years have been the result of SQL injection attacks, leading to reputational damage and regulatory fines. In some cases, an attacker can obtain a persistent backdoor into an organization's systems, leading to a long-term compromise that can go unnoticed for an extended period.

How to prevent SQL injection

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

The following code is vulnerable to SQL injection because the user input is concatenated directly into the query:

```
String query = "SELECT * FROM products WHERE category = '"+ input + "'";
```

```
Statement statement = connection.createStatement();
```

```
ResultSet resultSet = statement.executeQuery(query);
```

This code can be easily rewritten in a way that prevents the user input from interfering with the query structure:

```
PreparedStatement statement = connection.prepareStatement("SELECT * FROM products WHERE category = ?");
```

```
statement.setString(1, input);
```

```
ResultSet resultSet = statement.executeQuery();
```

Parameterized queries can be used for any situation where untrusted input appears as data within the query, including the WHERE clause and values in an INSERT or UPDATE statement. They can't be used to handle untrusted input in other parts of the query, such as table or column names, or the ORDER BY clause. Application functionality that places untrusted data into those parts of the query will need to take a different approach, such as white-listing permitted input values, or using different logic to deliver the required behavior.

For a parameterized query to be effective in preventing SQL injection, the string that is used in the query must always be a hard-coded constant, and must never contain any variable data from any origin. Do not be tempted to decide case-by-case whether an item of data is trusted, and continue using string concatenation within the query for cases that are considered safe. It is all too easy to make mistakes about the possible origin of data, or for changes in other code to violate assumptions about what data is tainted.

How does a SQL injection work?

Developed in the early 70s, SQL (short for structured query language) is one of the oldest programming languages still in use today for managing online databases. These databases contain things like prices and inventory levels for online shopping sites. When a user needs to access database information, SQL is used to access and present that data to the user. But these databases can also contain more sensitive and valuable data like usernames and passwords, credit card information, and social security numbers. This is where SQL injections come into play.

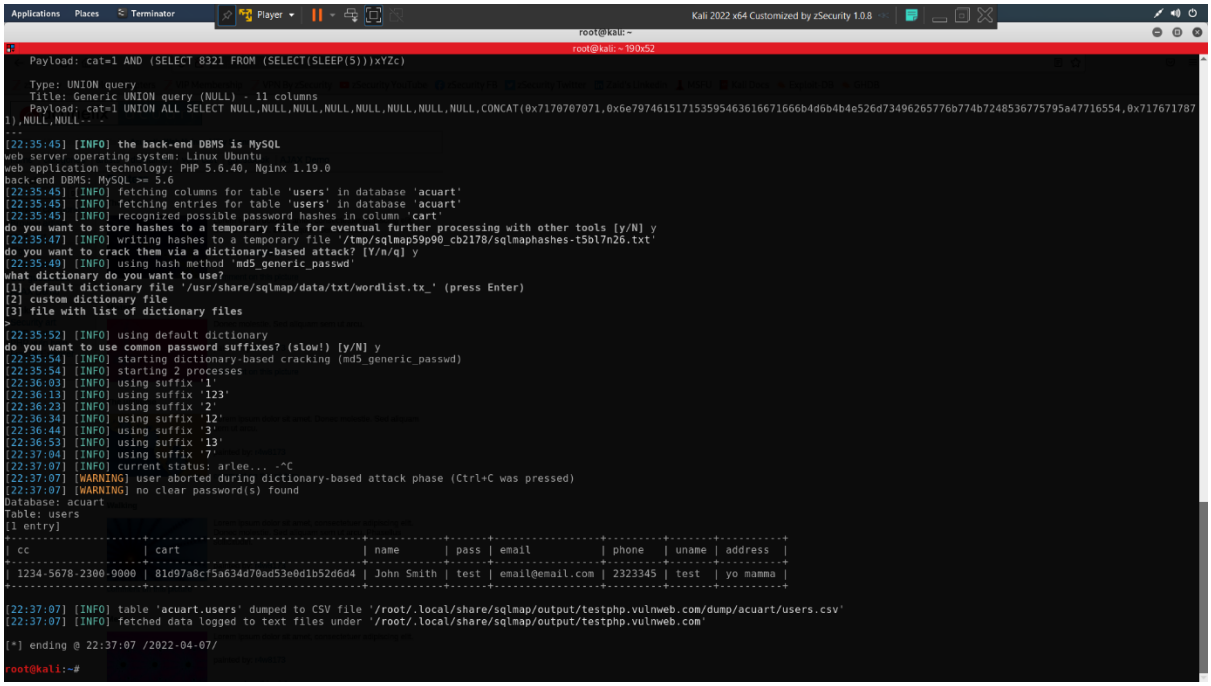
Put simply, a SQL injection is when criminal hackers enter malicious commands into web forms, like the search field, login field, or URL, of an unsecure website to gain unauthorized access to sensitive and valuable data.

Here's an example. Imagine going to your favourite online clothing site. You're shopping for socks and you're looking at a Technicolor world of colourful socks, all available with a click of your mouse. The wonders of technology! Every sock you see exists in a database on some server somewhere. When you find a sock you like and click on that sock, you're sending a request to the sock database, and the shopping site responds with the information on the sock you clicked. Now imagine your favourite online shopping website is constructed in a slipshod manner, rife with exploitable SQL vulnerabilities. A cybercriminal can manipulate database queries in such a way that a request for information about a pair of socks returns the credit card number for some unfortunate customer. By repeating this process over and over again, a cybercriminal can plumb the depths of the database and steal sensitive information on every customer that's ever shopped at your favourite online clothing site—including you. Taking the thought experiment even further, imagine you're the owner of this clothing site. You've got a huge data breach on your hands.

One SQLI attack can net cybercriminals personal information, emails, logins, credit card numbers, and social security numbers for millions of consumers. Cybercriminals can then turnaround and sell this personal info on the gloomiest corners of the dark web, to be used for all kinds of illegal purposes. Stolen emails can be used for phishing and malspam attacks. Malspam attacks, in turn, can be used to infect victims with all kinds of destructive malware like ransomware, adware, cryptojackers, and Trojans (e.g., Emotet), to name a few. Stolen phone numbers for Android and iOS mobile devices can be targeted with robocalls and text message spam.

Tasks

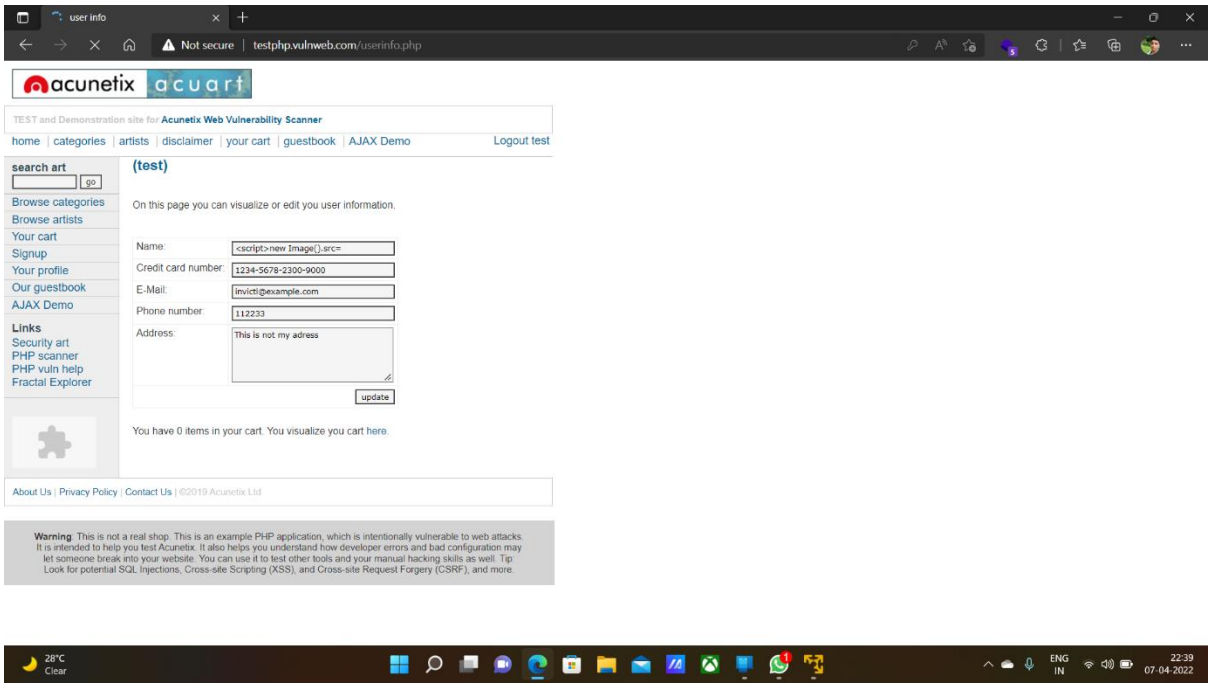
4.1 SQL Injection using sqlmap:



One way of gaining access to the admin panel is using a tool called sqlmap.

Sqlmap is used to gain entry to a database management system which are not secured.

After gaining access to the DBMS, we can use the login present in the table to gain entry to a system or web application.



4.2 Manual SQL Injection:

We can also gain access to a website if we comment some part of the SQL statement. But to comment the specific part of the SQL statement we first need to have an idea how a SQL statement works.

Generally, in a login page after clicking on submit button a SQL query will be generated. And with the successful execution of this query, we will get access to the web-application. The result of this should be “1”, then only we will get access to the web-application.

The query will generally be in the form

“select * from users where username = ‘uname’ and password = ‘pass’;”

In the above query uname is the variable in which the user provided username will be stored and pass is the variable in which the user provided password will be stored. Users is the name of the table in which we will store the usernames and passwords of the users. If the uname is present in the table then it will return 1 and if it is not present then it will return 0. Similarly, if we have pass in the table, it will return 1 or else it will return 0. In between uname and pass we are using “and” operator, which means that it will be true and will return the value if both the values are 1.

In manual SQL Injection we have 2 cases. They are:

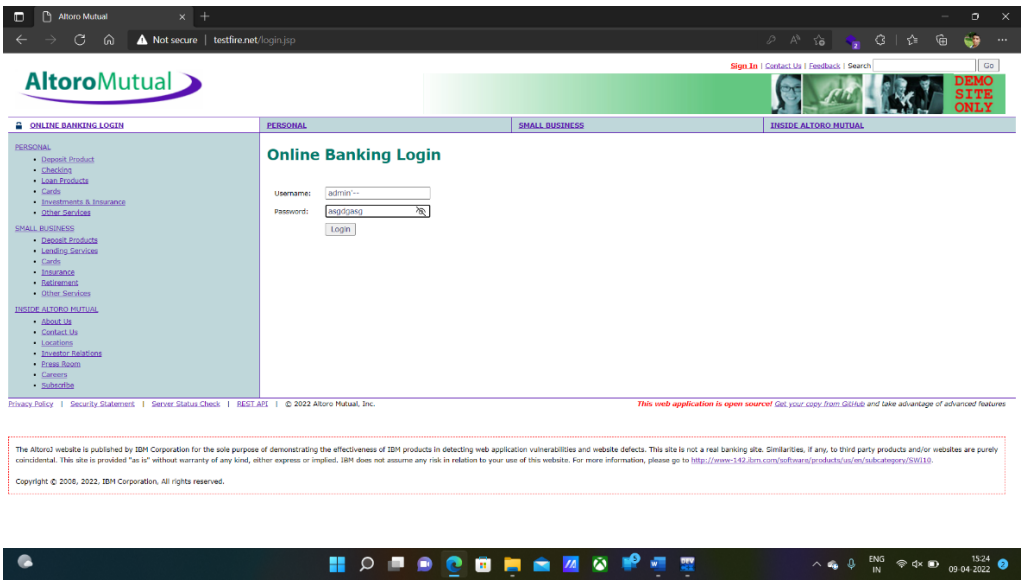
1. If we know username
2. If we don’t know both username and password

Case 1:

In this case we know the username but not the password. The website I used for Manual SQL Injection is http://testfire.net

In this website I know there is a user with the username “admin”. Now I want to access the account of this particular user but I don’t know his password. So now my task is to comment the part of the statement after username.

I know the username as “admin” so I will type the username as “admin’--” and password as any random data.

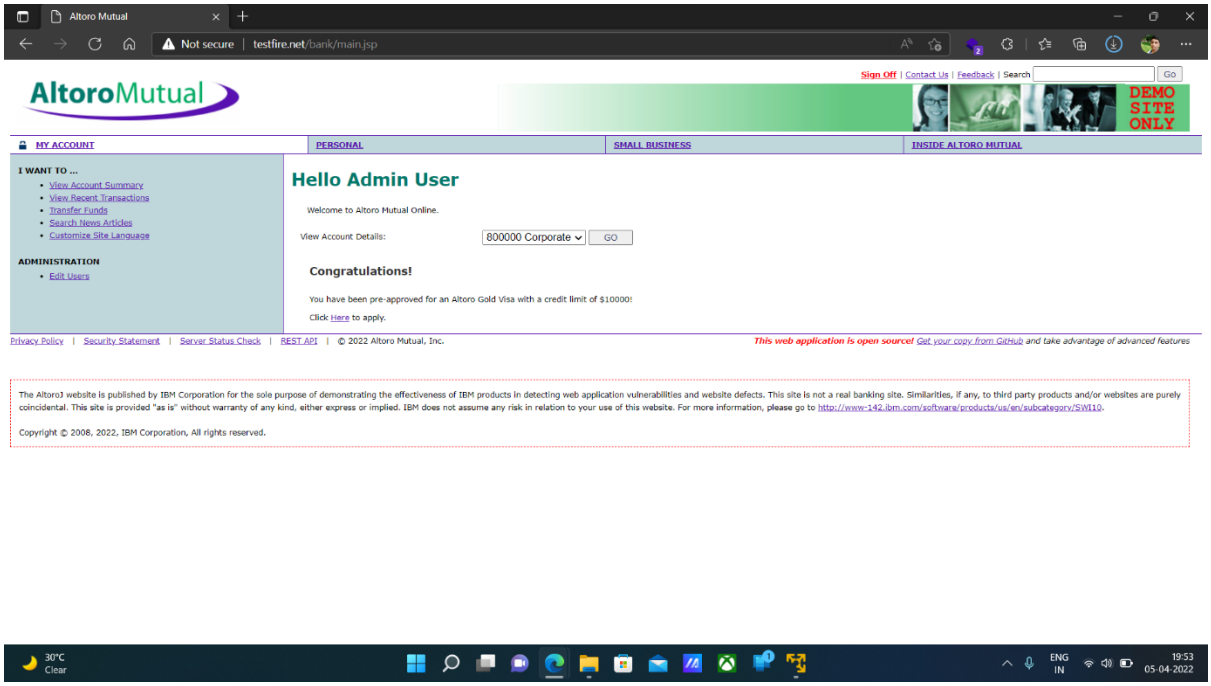


Now the with the given data the SQL query will become as

“select * from users where username = ‘admin’—’and password = ‘asgdgasg’”

But since we are using username as admin’--, and in SQL “--” will start the comment, so the SQL query after admin’ will be considered as a comment. The compiler will think that we are only asking for the user with username as admin and will not check the password as we have commented it.

So, if there is any user with the username as admin then it will give us access to it.



Case 2:

In this case we don’t know both the username and password. So, we will have to get the result of the SQL query as 1 to gain access to the web-application.

So, as we know the SQL query will be in the form of

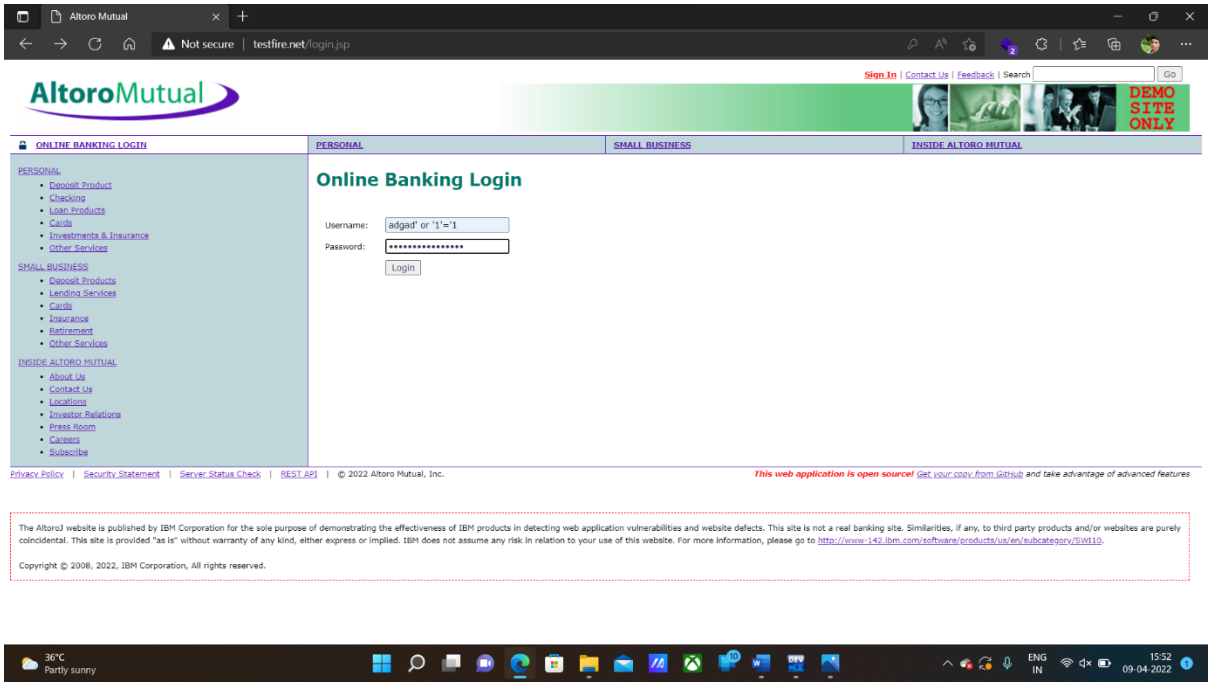
“select * from users where username = ‘uname’ and password = ‘pass’”

So now we should get the result as 1 and 1 so that we can gain access to the web-application.

Now let’s use the “OR” operator to gain access to the web-application.

Type some random data in the username field. After that we have to use or and type a conditional statement whose truth value will be true. And copy the same and paste it in the password field.

Here I am using adgad' or '1'='1 as both username and password.



After clicking the login button, the SQL statement generated will be

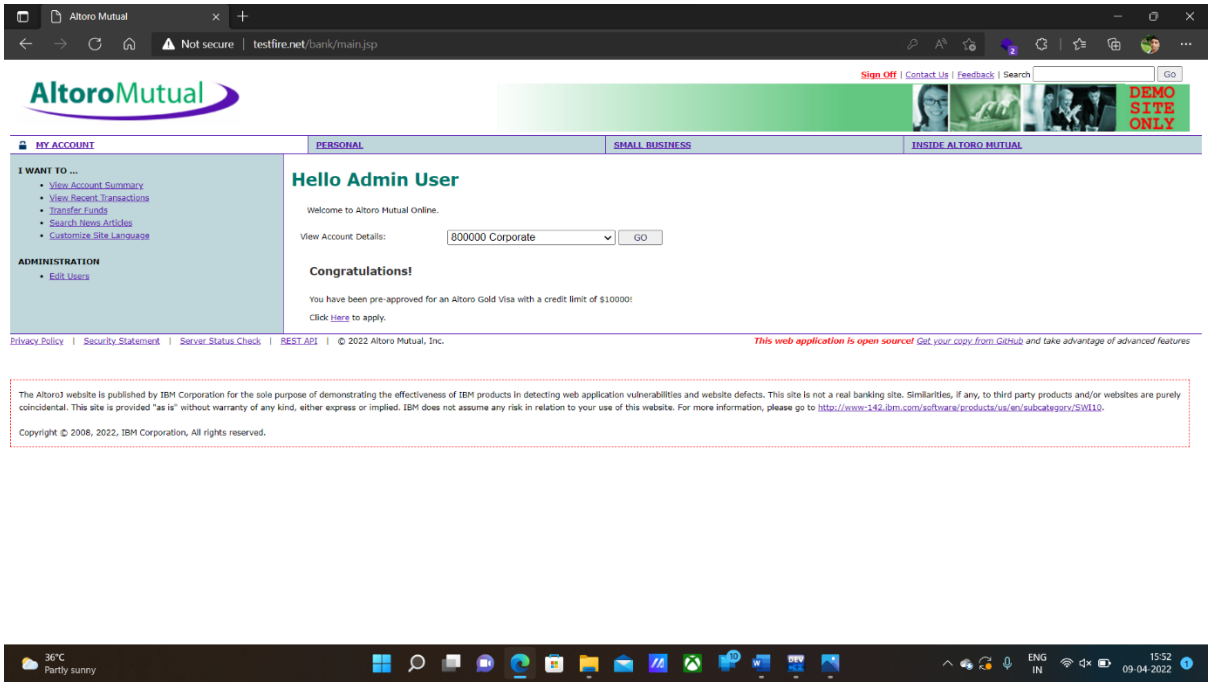
“select * from users where username = ‘adgad' or '1'='1’ and password = ‘adgad' or '1'='1’”

In the both the field the compiler will first check with the first one is present in the table or not. If yes then it will return 1 and go with the second conditional statement. If the conditional statement is true, it will return 1 or else 0.

For the above case the SQL query will return all the statement “(0 or 1) and (0 or 1)”

As we know the compiler will follow BODMAS rule so first it will check the statement present in the brackets and the statement will become as “1 and 1”

As the result of the statement is 1, we will get permission to enter the web-application.

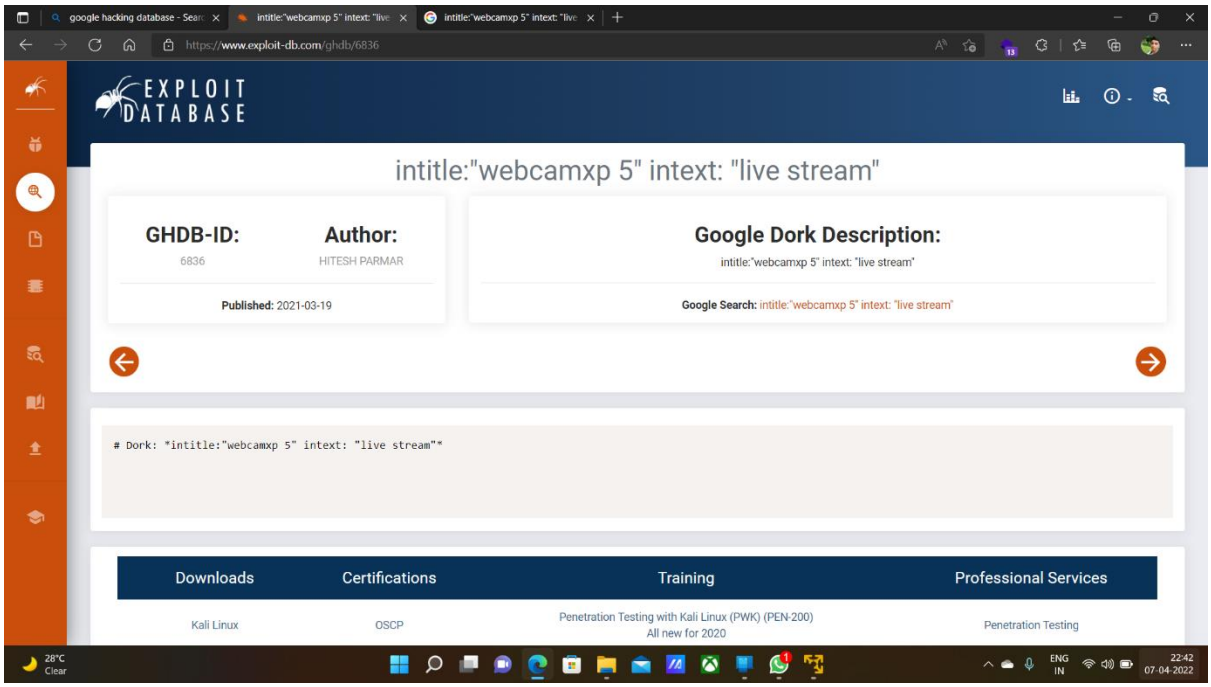


4.3 Live Cameras using google hacking database

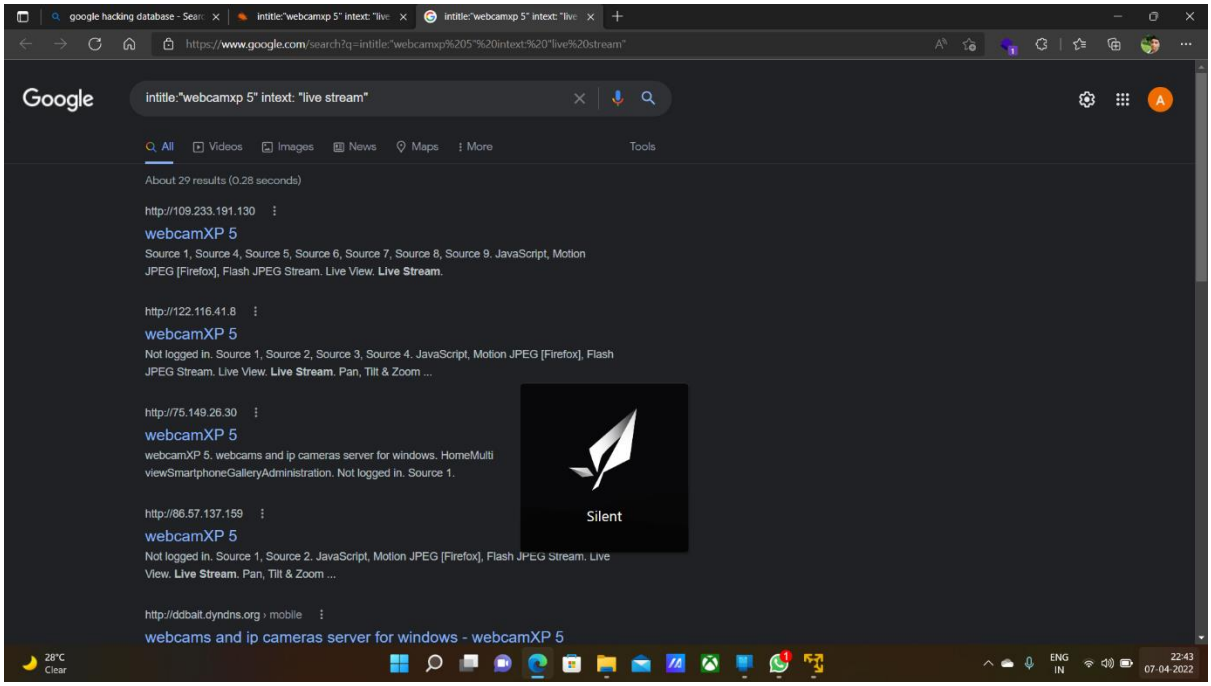
The **Google Hacking Database** (GHDB) is a categorized index of Internet search engine queries designed to uncover interesting, and usually sensitive, information made publicly available on the Internet.

To get the related dorks to see live cameras we need to search for live cameras.

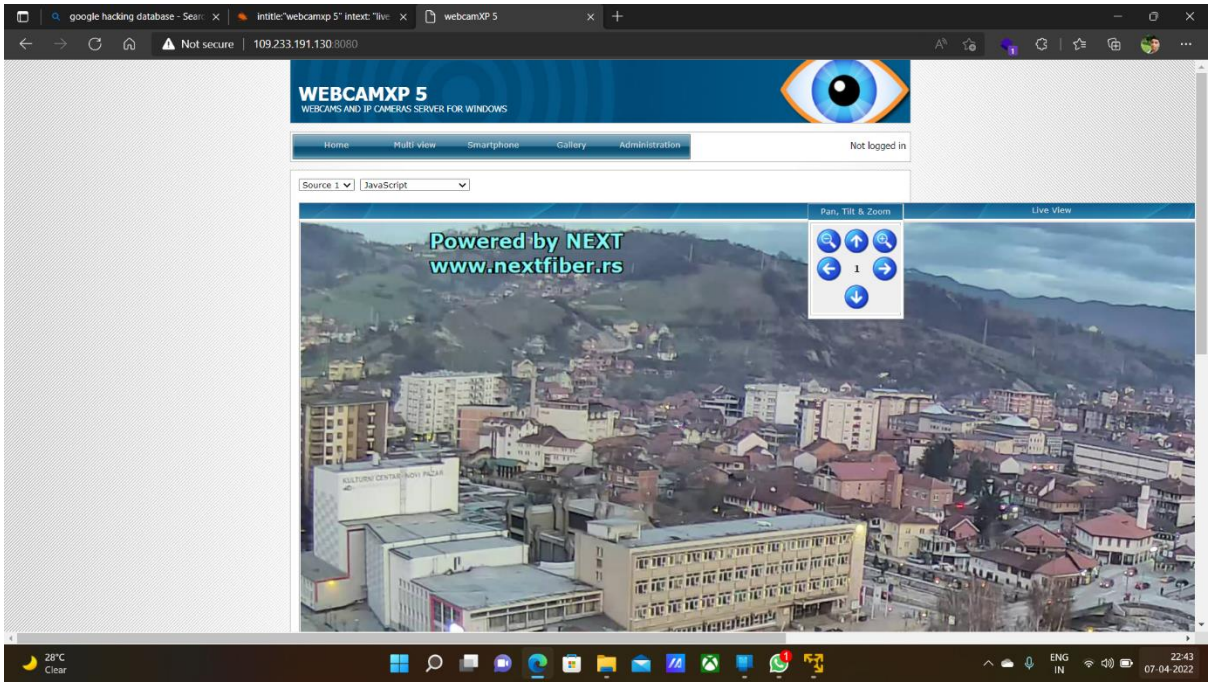
The GHDB-ID of the dork I'm using is 6836 and it was published on 19-03-2021. The author of this dork is Hitesh Parmar.

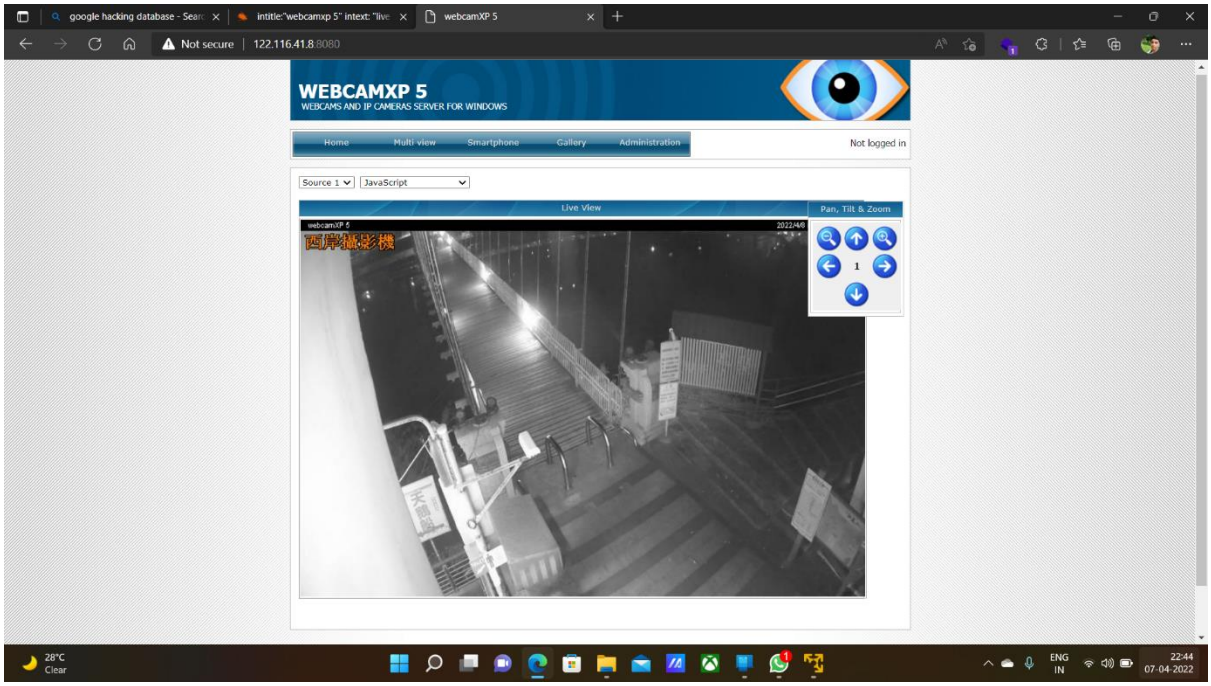


After finding a suitable dork we need to copy paste the dork in any search engine. After searching we will get all the links related to the particular dork.



Most of the links in the page will show the result we required.




google hacking database - Sear
intitle:"webcamxp 5" intext: "live
webcamXP 5
Not secure | 122.116.41.8:8080
WEBCAMXP 5
WEBCAMS AND IP CAMERAS SERVER FOR WINDOWS
Home
Multi view
Smartphone
Gallery
Administration
Not logged in
Source 1
JavaScript
Live View
Pan, Tilt & Zoom
webcamXP 5
2022/4/6
西岸攝影機
1
28°C
Clear
ENG
IN
22:44
07-04-2022