SQLi for Beginners

Penetration Testing Technique

Cameron Wade

Overview

SQL Injection (SQLi for Short) attacks are a method of attack that takes advantage of poor web development. Web applications are prone to SQL Injection attacks when web developers fail to consider user input validation, when asking a user for any kind of input. No validation of the user's input occurs, so the user can submit a SQL Query as their input and query the backend databases to extract information from them.

Background

The internet has made it possible for its users to access countless numbers of hosted webpages that provide a wide range of services like media outlets, news sources, web applications, and more! It takes both front-end and back-end developers to make sure that a web page performs and appears the way that it does, when a user visits them. Web pages will store, retrieve, and interact with information stored in backend SQL databases. Web Development is a crucial part of preventing SQL Injection attacks from occurring.

Methodology

1. Preparation

Before SQL Injection occurs, the threat actor must find a parameter vulnerable for exploitation. There are various tools, like SQLMap, that can be used to test a web page for vulnerable parameters. If there are fields where users can provide their own input, it is likely that the web page is interacting with a back-end database to store the input or retrieve information that the user requested.

The "SQL-To-Shell" Vulnerable VM, found on VulnHub, is a great resource for beginners who are looking to learn SQL Injection attacks. This will be the Virtual Machine used in this example.

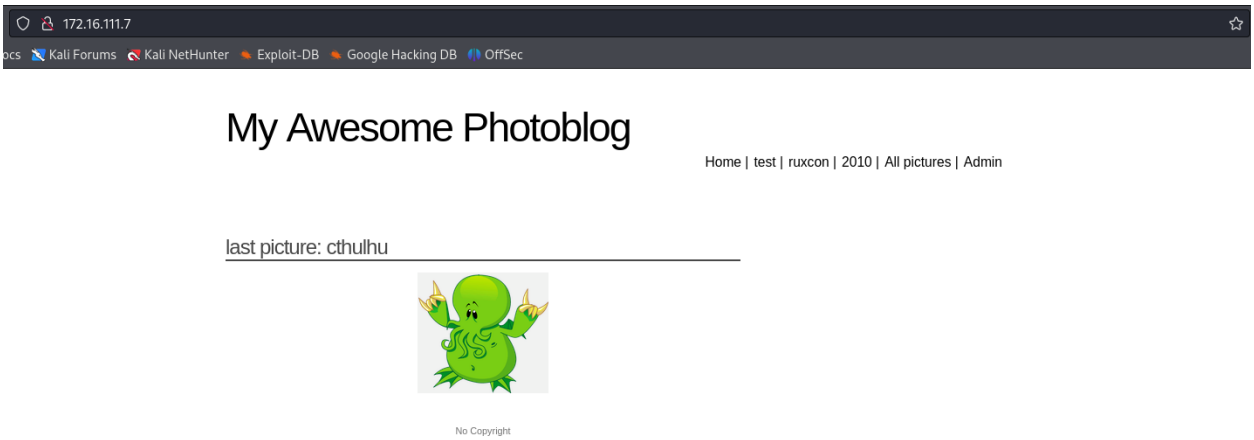
2. Method Execution Steps

Step 1: Determining a Vulnerability

When the Vulnerable Virtual Machine has been powered on and you have determined that the attacking machine can communicate with it, it is now time to determine what vulnerabilities exist on the target machine. This can be done using a simple “nmap -sV [Target Host IP]” command to scan the target host for open ports and services/service versions that are operating on those ports.

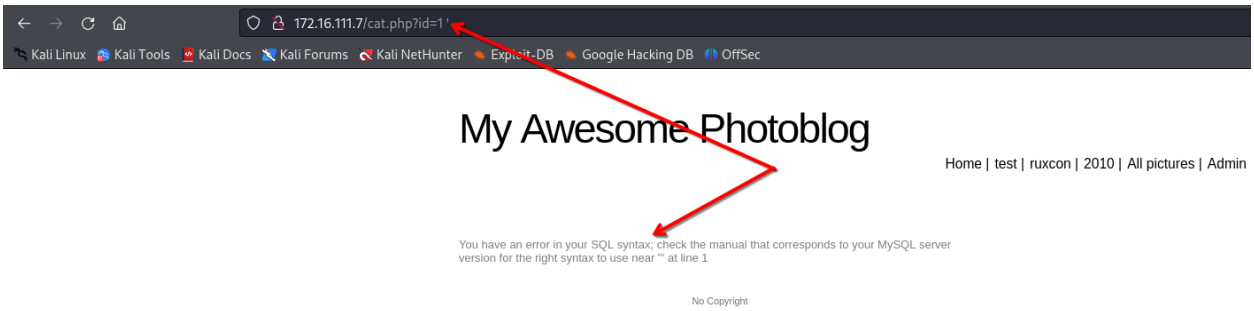
```
┌──(kali㉿kali)-[~]
└─$ nmap -sV 172.16.111.7
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-06 11:44 MST
Nmap scan report for 172.16.111.7
Host is up (0.00042s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 5.5p1 Debian 6+squeeze2 (protocol 2.0)
80/tcp open  http    Apache httpd 2.2.16 ((Debian))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 19.36 seconds
```

An HTTP web service was found to be operating on exposed port 80. Access to this service can be tested by opening a web browser and navigating to the target IP and specifying the desired port. http://172.16.111.7:80. Since it is an HTTP service, the attacker must be sure to not utilize https:// in the URL.

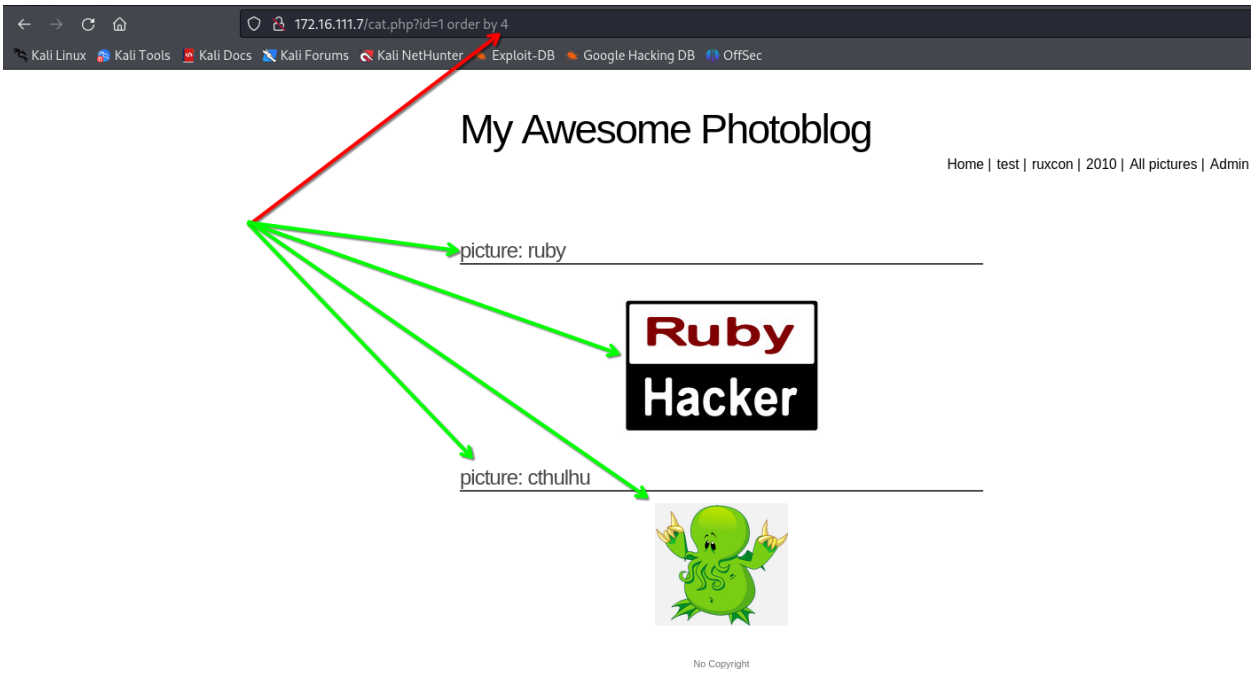


The webpage was determined to be accessible, but it doesn’t look like the webpage is utilizing a parameter that can be used for a SQL Injection attack. There are additional tabs that should be checked for vulnerable parameters.

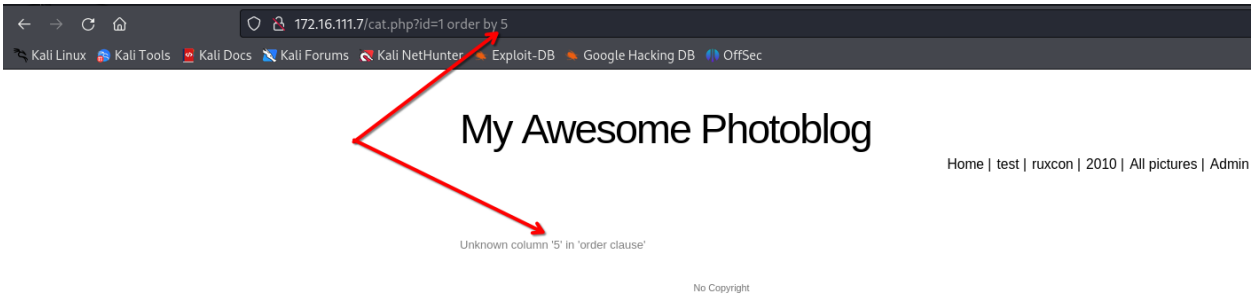
When navigating to the" test" tab of the site, there is an exposed parameter in the URL of the page that indicates the page is interacting and pulling information from a SQL database and displaying anything with an id of 1 (as indicated by the id=1 portion of the URL). To test if this "id=1" parameter is vulnerable to exploitation, a common tactic that bad actors will use is placing a space and an apostrophe at the end of the URL to see if a SQL syntax error is displayed on the page



As noticed in the screenshot displayed above, there is a SQL syntax error that is displayed on the page, when the space and the apostrophe were added to the end of the URL. This is a good sign that the parameter is vulnerable to exploitation. Another common tactic employed by threat actors to determine how many data fields from the database are present on the page is to use an "ORDER BY" clause, utilized in the same way as the last method. This clause is used to sort the fetched data in ascending order.



The screenshot above shows that an ”ORDER BY 4” clause was used and successfully displayed information. On the screenshot above, there are four pieces of data that were retrieved when that query was sent to the database. The retrieved data was then displayed on the page. Since there are up to four pieces of data that were displayed this means that, in theory, ”ORDER BY” clauses for numbers one through three should also correctly display retrieved data.

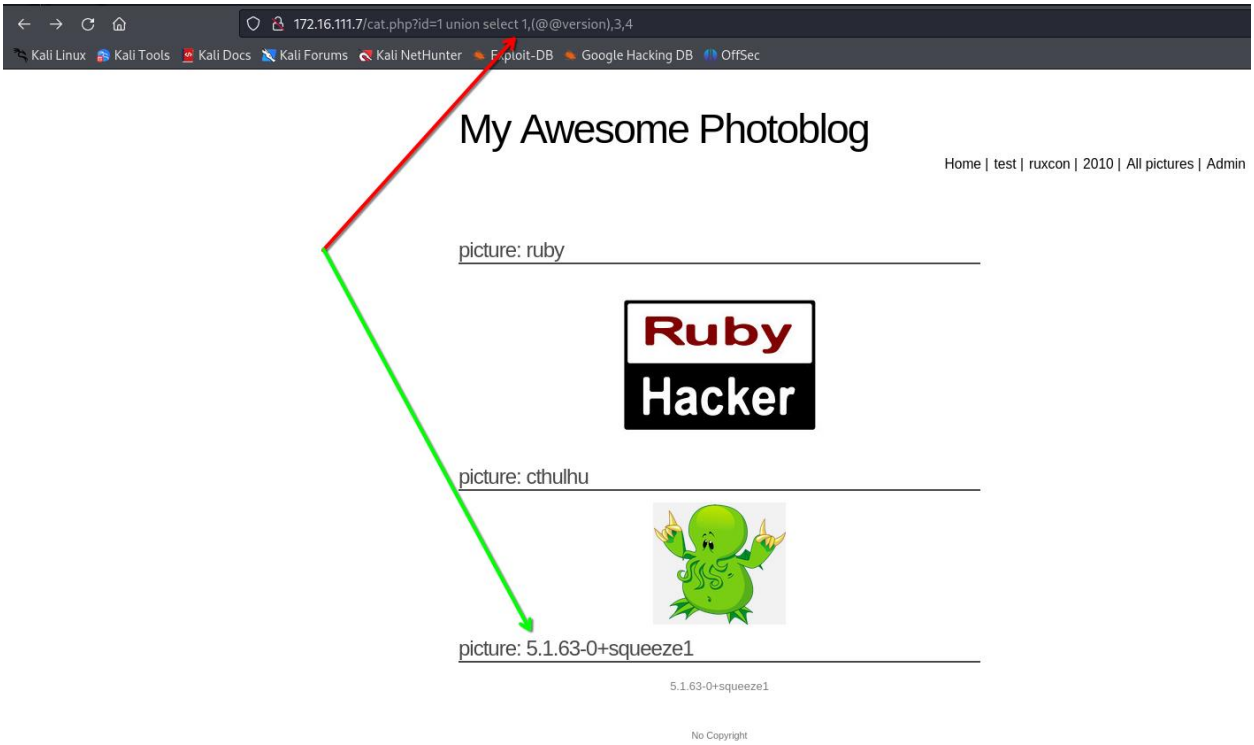


Displayed above is a screenshot of an ”ORDER BY” clause that responded in error the the query that was sent. The meaning that can be derived from this error is that there are no more than four pieces of retrieved data from the database that can potentially be modified to display different information, using SQLi exploitation.

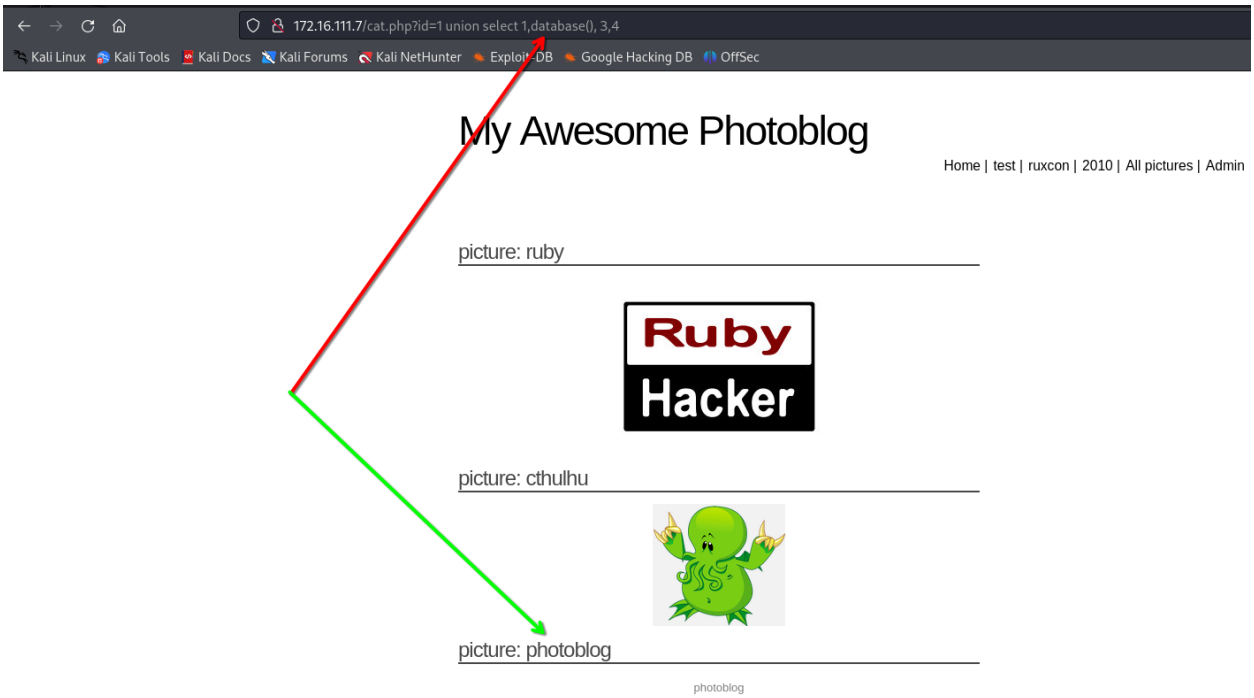
Step 2: Examining Threat Landscape

Now that a vulnerable parameter has been confirmed, it’s time to determine what is available for a threat actor to access. When attempting SQLi attacks, it’s helpful to have a basic understanding of SQL syntax to craft queries that will display information that will

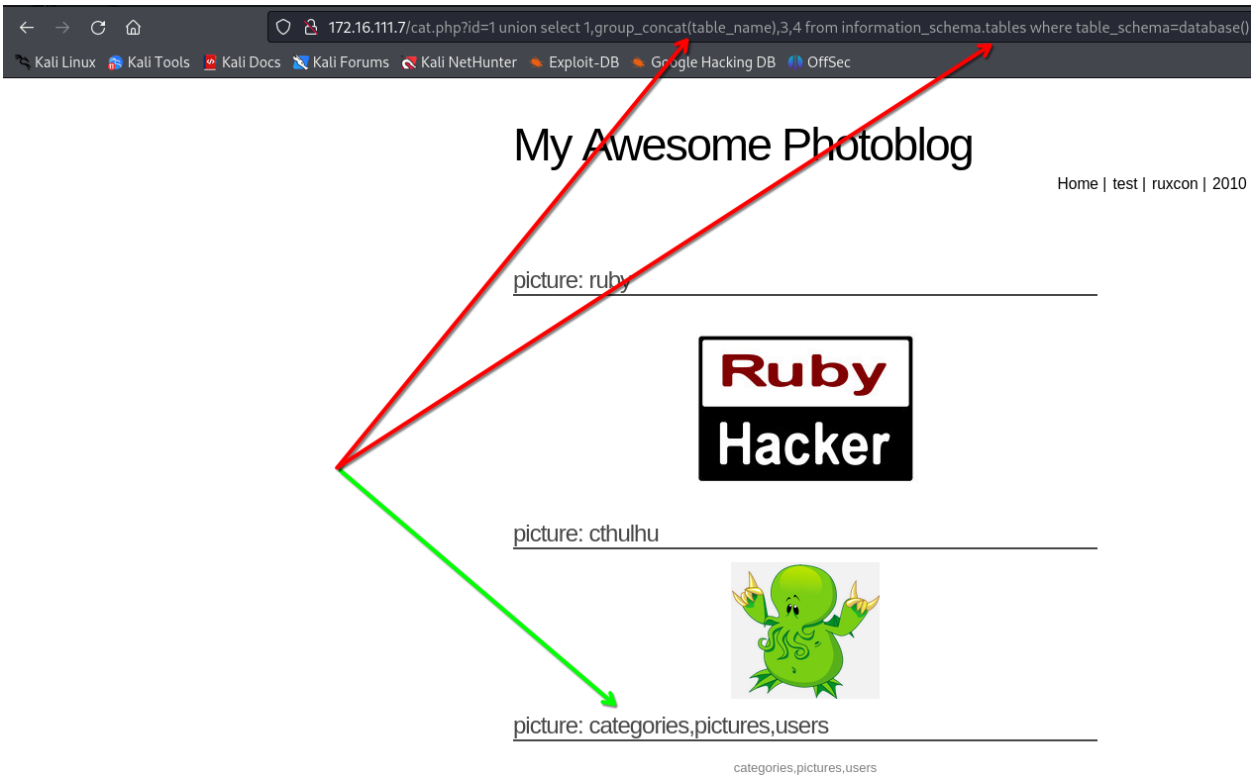
help drive an attack. For example, we know there are four available columns of information we are working with. When we are crafting exploitation statements, we must utilize the “UNION SELECT” syntax to join multiple selected sets of information into one response.



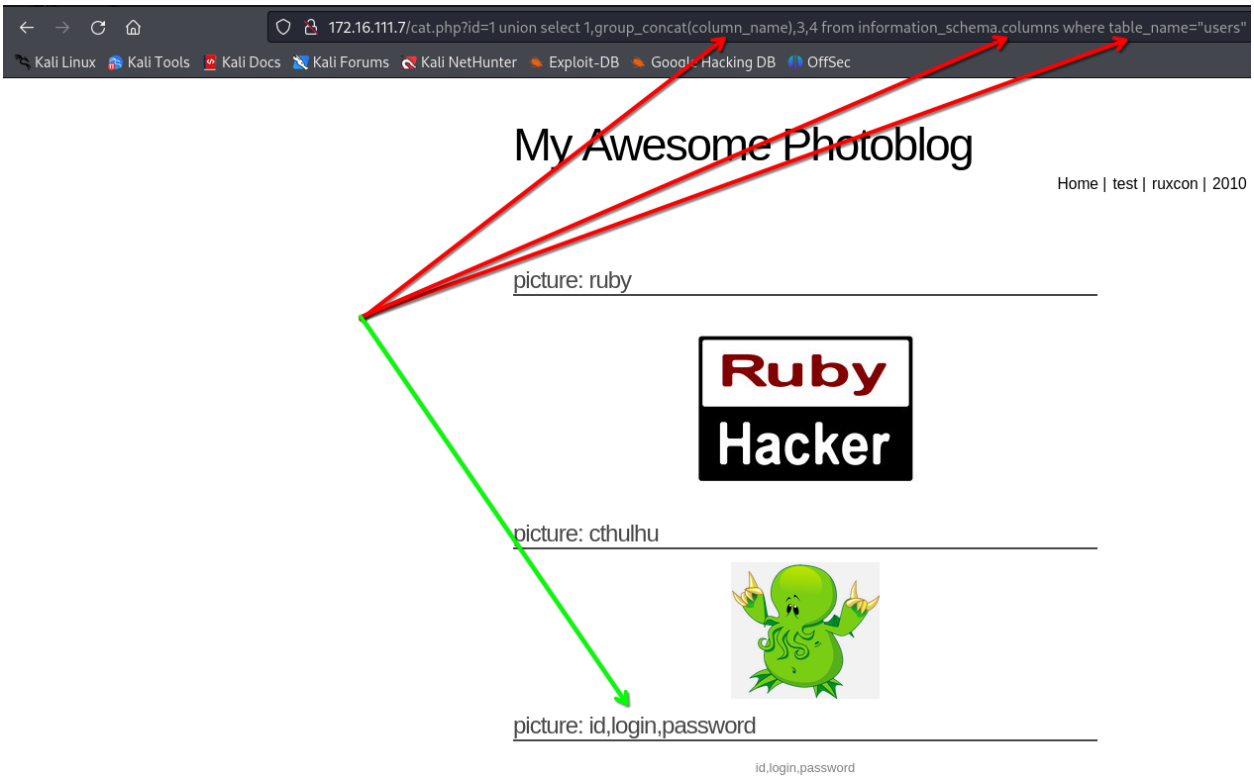
Noted by the red arrow in the screenshot above, you can see the use of the ”UNION SELECT” statement to combine all four sets of information into one response. Take notice that the ”(@@version)” statement is placed in the second position, replacing the ”2” that would be in the basic ”UNION SELECT” statement. Noted by the green arrow in the above screenshot is the field that was modified by the modification of the ”UNION SELECT” statement. We now know that the ”2” controls the field on the page that is marked by the green arrow. (The ”@@version” command is used to display the version and the type of database that is being interacted with)



The ”database()” command was used in the above screenshot. This command is used to determine the active database that is being interacted with. Present in every MySQL instance is an information_schema database that stores all the information and metadata about the tables and columns within each database. That information_schema database can be queried to determine the available table within the photoblog database.



There are a few things to break down from the query in the above screenshot. The first being the ”group_concat(table_name)” statement that was used. This was used to concatenate multiple outputs from the ”table_name” command into one output. Because the information_schema database contains information about tables from all databases, it’s important that we only select to view the information from the database that we are currently working with. This can be done by also using the ”where table_schema=database()” to make sure that the query only fetches and displays tables that are from the current database.



A similar command as the previous one can be used to view the available columns in one of the tables found in the last column. To do this, the previous query needs to be modified in a few ways to specify key pieces of information. Firstly, the set of data we are looking to retrieve this time is the ”column_name” which will be used to replace the previous ”table_name” value. The next thing that needs modified is the table from the information_schema database we are querying. For this query, the information we are searching for lies in the columns table. Lastly, since we don’t want all the available columns in all databases, we need to specify the ”table_name” we are looking to search within. Above, we can see there are three available columns to potentially pull information from... id, login, and password.

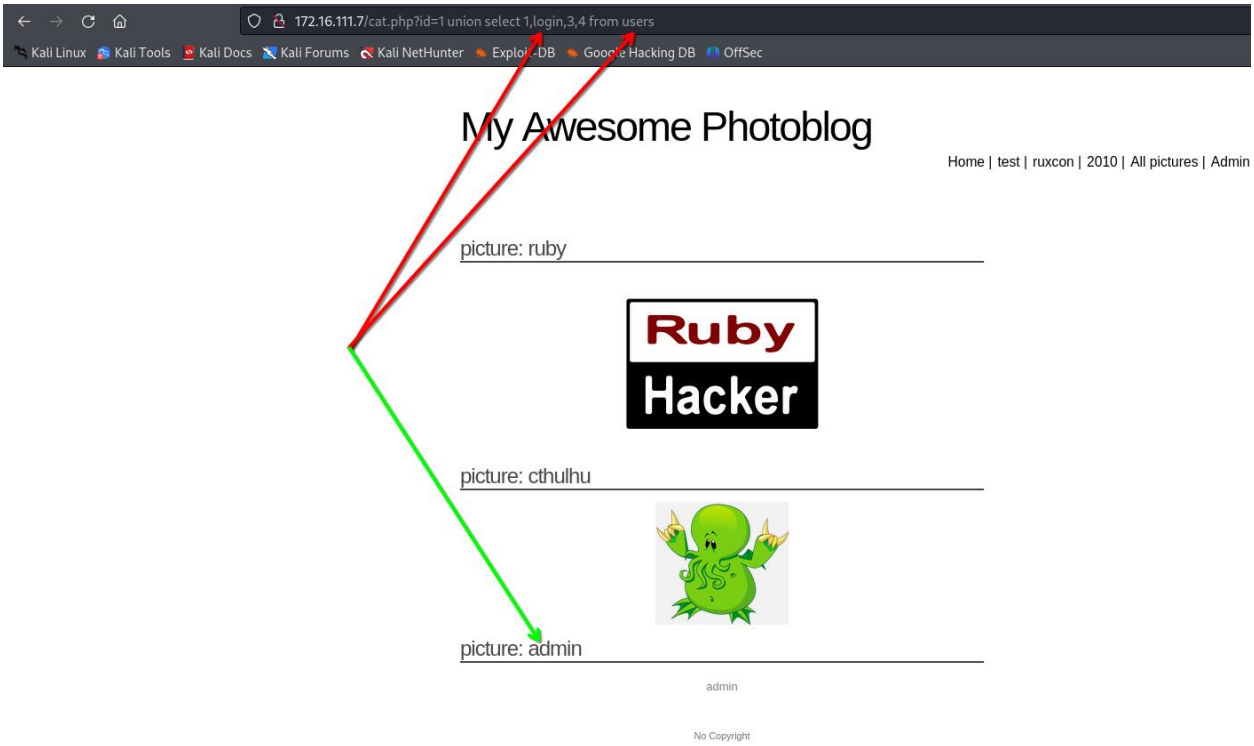
We now have a decent understanding of the available threat landscape, and we know what can potentially be exploited in order to retrieve information that we wouldn’t normally have access to. This also allows us to make more informed decisions on where we think sensitive information is contained.

Step 3: Vulnerability Exploitation

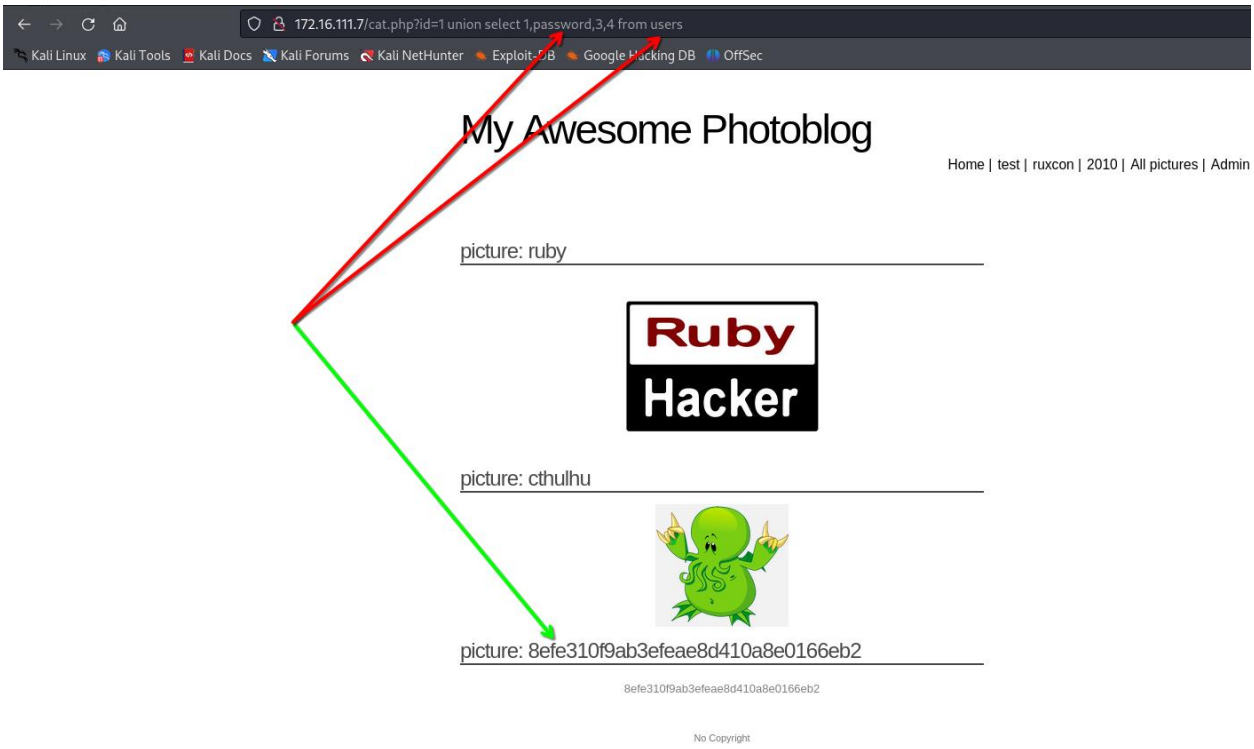
Now that the threat landscape has been examined and the attacker is now informed on potentially available targets, the attacker can proceed with creating a malicious query that

will reach the backend database and return information that would normally be accessible to normal users.

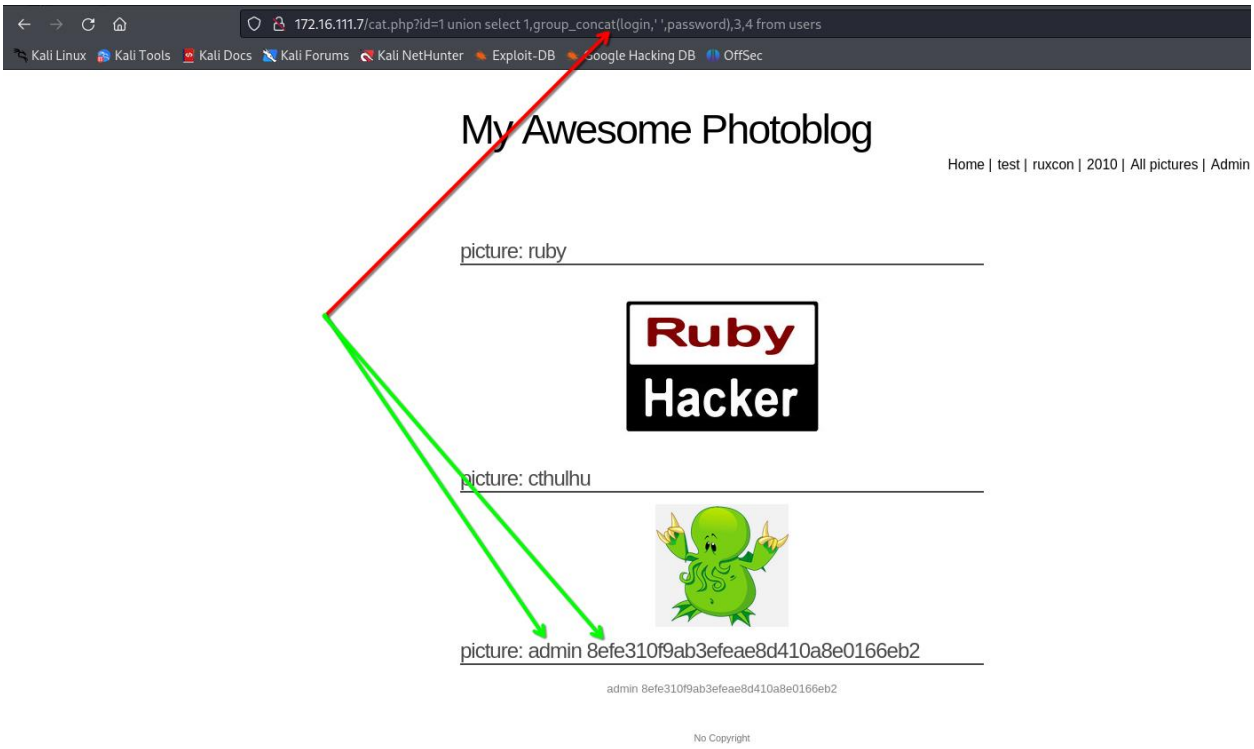
During Threat Landscape Examination, the attacker was able to determine that there were two columns in the “users” table of the currently active “photoblog” database that may contain sensitive information. These two columns are the “login” and “password” column which may contain cleartext information that may give an attacker the necessary information to gain access to an administrator account.



Pictured in the screenshot above, the attacker was able to craft a malicious query that the web server sent to the backend database and retrieved information that looks to be the login username of an administrator user. Noted by the red arrows, the attacker had to specify the name of the column they want to retrieve data from in the ”UNION SELECT” statement. Also indicated by the red arrow, another important factor is the attacker specified which table they wanted to pull the column information from. This is important because if the attacker is attempting to pull information from a column in a specific table and fails to specify what table the column resides in, the web server will not know where to look for the column.



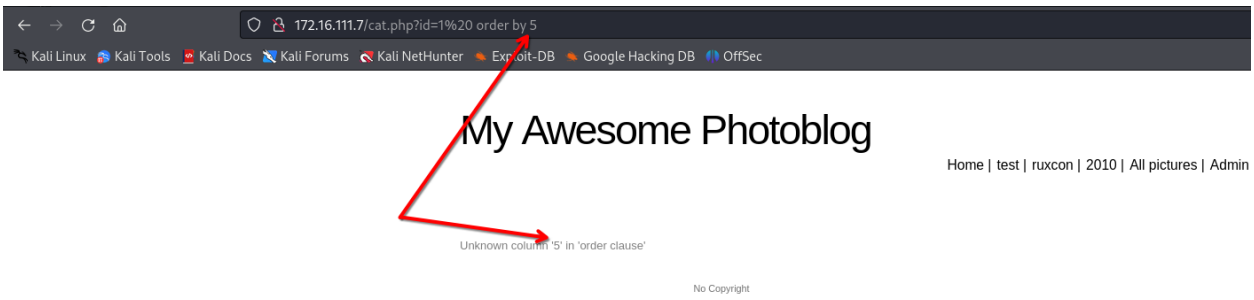
Pictured in the screenshot above, the attacker used an almost identical approach to query the backend database for user passwords. The previous malicious query was slightly modified to, instead, query the ”password” column of the ”users” table but the attacker properly specified the name of the table that the column resides in.



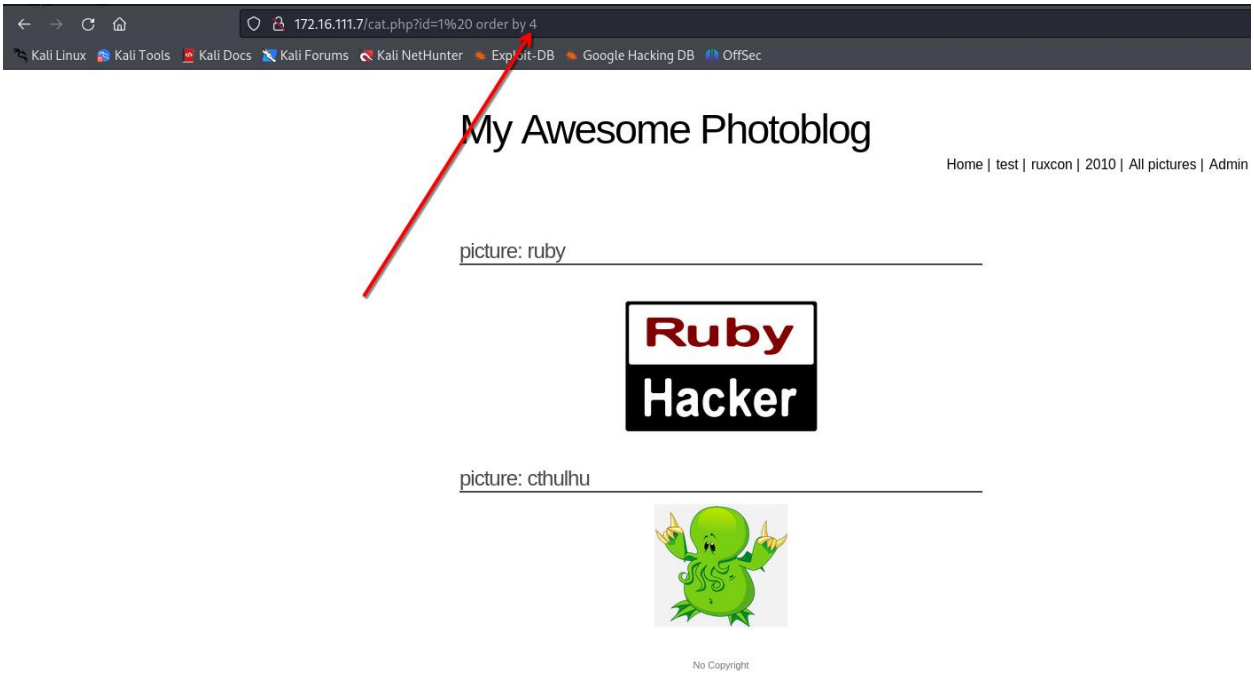
The attacker was able to modify the previous request and use a ”group_concat” function to combine the data sets from both the ”login” and ”password” columns into one unified output. The attacker still had to pay careful attention to specifying the name of the table that the columns live in. As seen in the screenshot below marked by the green arrows, the data sets were unified into one clean output.

3. Validation

When performing SQL Injection attacks, it is imperative to pay attention to and listen to any syntax errors that you receive while testing for vulnerable parameters or queries. This will help drive the vulnerability analysis process. For example, when using “order by 5” in earlier examples, the syntax error that was received was “unknown column ‘5’ i ‘order clause’” meaning that there wasn’t a column 5 being displayed, indicating that the value of displayed columns is less than 5.



This can help the attacker determine how many modifiable columns there are being displayed on the webpage by decrementing the value of ‘5’ by one until the attacker receives valid data instead of a syntax error



Notice above, the attacker only had to decrement the value of five by one to receive a valid response that didn’t result in an error. The attacker can now determine that there are four total interactable columns vulnerable to exploitation.

Best Practices

Performing SQLi attacks relies heavily on the attackers understanding of SQL interactions in terms of how to analyze a web URL and determine the type of SQL query that is being utilized to retrieve the information and how to utilize SQL syntax to create and modify these queries. This means that a good starting place for beginners looking to perform these kinds of attacks is familiarize yourself with basic SQL syntax as well as how web servers query backend database management systems to retrieve and display information.

The vulnerable virtual machine used as an example in this document is a beginner’s playground for those looking to break into the SQLi space or looking to understand more about this type of attack. This means that real-world scenarios and other VMs may require different approaches to successfully perform a SQLi attack. This can be due to various elements like varied security from organization to organization, different database

versions, different database structure, etc. These approaches may require a more advanced understanding of these concepts.

Conclusion

In today's ever-changing world of technology and the internet, information of all kinds is stored in various locations on the internet. Bad actors utilize many different attack types and take advantage of flaws and vulnerabilities to obtain access to systems or information that they normally wouldn't have access to.

SQL Injection (SQLi for Short) attacks are a type of attack that take advantage of a flaw in web development. This flaw stems from web developers who fail to consider input validation, when requesting input from a user for a request. Because the input validation is not occurring, the attacker can pass malicious input through the request that may result in access to systems or information that they would normally not have access to.

When attempting a SQLi attack, it is critical for the attacker to have a basic and fundamental understanding of SQL syntax and the ability to analyze a URL or a webpage to determine what kind of SQL query is being made. These two abilities allow the attacker to analyze a webpage for vulnerable parameters and queries and modify web requests with malicious SQL queries to gain access to information that normally wouldn't be accessible.

References

https://pentesterlab.com/exercises/from_sqli_to_shell/course

Revision History

05/30/2024 - Overview, Background

06/06/2024 - Preparation, "Determining a Vulnerability"

06/08/2024 - "Examining Threat Landscape", Added Reference

06/09/2024 - "Vulnerability Exploitation", Validation, Best Practices, Conclusion