



DoS ATTACKS USING SQL WILDCARDS

Ferruh Mavituna

www.portcullis-security.com

This paper discusses abusing Microsoft SQL Query wildcards to consume CPU in database servers. This can be achieved using only the search field present in most common web applications¹.

If an application has the following properties then it is highly possibly vulnerable to wildcard attacks:

- 1- An SQL Server Backend;
- 2- More than 300 records in the database and around 500 bytes of data per row;
- 3- An application level search feature.

As you might notice I have just described 90% of Microsoft SQL Server based CMSs, blogs, CRMs and e-commerce web applications. Other databases could be vulnerable depending on how the applications implement search functionalities although common implementation of the search functionality in SQL Server back-end applications is vulnerable.

SEARCH QUERIES

The SQL Server supports the following wildcards in LIKE queries². "%", "[", "[^]" and "_"

If you were to enter 'foo' into the search box, the resulting SQL query might be:

```
SELECT * FROM Article WHERE Content LIKE '%foo%'
```

In a decent database with 1-100000 records the query above will take less than **a second**. The following query, in the very same database, will take about **6 seconds** with only **2600 records**.³

¹ This attack does not only affect web applications, also can occur in any other application with a back-end database and search functionality.

² [http://msdn2.microsoft.com/en-us/library/aa933232\(SQL.80\).aspx](http://msdn2.microsoft.com/en-us/library/aa933232(SQL.80).aspx)

```
SELECT TOP 10 * FROM Article
WHERE Content LIKE
'%_[^!_%/a?F%D)_(F%)_%([({}){()})£$&N%_)*£()*$*R"_)][%](%[x])%a][\$*"£$-9]_%'
```

So, if an attacker wanted to tie up the CPU for 6 seconds they would enter the following to the search box:

```
_[^!_%/a?F%D)_(F%)_%([({}){()})£$&N%_)*£()*$*R"_)][%](%[x])%a][\$*"£$-9]_
```

Furthermore, if multiple search terms are supplied (separated with spaces), the application will often separate them with spaces, e.g. If we search for 'foo bar', the resulting SQL query will be:

```
SELECT * FROM Article WHERE Content LIKE '%foo%' OR Content LIKE '%bar%'
```

This leads to an even more efficient attack if we supply several wildcard attacks separated with spaces. This will cause a SQL Query similar to the following to be run in the database:

```
SELECT TOP 10 * FROM Article
WHERE Content LIKE
'%_[^!_%/a?F%D)_(F%)_%([({}){()})£$&N%_)*£()*$*R"_)][%](%[x])%a][\$*"£$-9]_%'
OR Content LIKE
'%_[X_%/a?F%D)_(F%)_%([({}){()})£$&N%_)*£()*$*R"_)][%](%[x])%a][\$*"£$-9]_%'
OR Content LIKE
'%_[()_%/a?F%D)_(F%)_%([({}){()})£$&N%_)*£()*$*R"_)][%](%[x])%a][\$*"£$-9]_%'
```

This takes **36 seconds** in the same database and consumes **100% of CPU** during the execution. As seen in this example the affects of extra OR statements are linear. This problem allows an attacker to consume database resources and the database connection pool within 1-2 minutes with a humble Internet connection such as 0.5-1.0 Mbit/s., maybe even with a dial-up connection.

The query time is also strongly correlated with number of records searched and the length of each field searched.

TESTING FOR SQL WILDCARD DOS

Testing is quite simple; just craft a query which will not return a result⁴ and includes several wildcards. Send this data through the search feature of the application. If the application takes more time than a usual search, it is vulnerable. More details about crafting a wildcard attack can be found under "[Crafting Search Keywords](#)" below.

During testing I used the "Web Application Stress Tool⁵" to carry out attacks. Any similar tool can be employed for this purpose.

CONNECTION POOLING

Depending on the connection pooling settings of the application and the time taken for attack query to execute, an attacker might be able to consume all connections in the connection pool, which will cause database queries to fail for legitimate users.

³ These benchmarks based on a test hardware, timing might differ in another systems. Benchmarks timing generally refers to CPU Time.

⁴ To be able to force the database server to scan whole index,

⁵ <http://www.microsoft.com/downloads/details.aspx?FamilyID=e2c0585a-062a-439e-a67d-75a89aa36495&displaylang=en>

Both of the test applications listed in here are forum applications, which does not mean that other applications are safe. These were just easy to setup and test also they are prone to get bigger in data level which makes them more vulnerable to wildcard attacks. Besides these, I did several tests against bespoke web applications. Results were similar.

YET ANOTHER FORUM

[Yet Another Forum](#) is an open source ASP.NET application with an SQL Server backend.

- After 2000 records one search query takes more than 30 seconds to execute, which leads to timeout database connection.
- An example setup of YAF stopped responding within 2 minutes with a connection speed of 1.5Mb.

WEB WIZ FORUMS

- Web Wiz Forums is a classical ASP application with an SQL Server backend,
- After 500 records following search keywords causes to a query that takes more than 30 seconds to execute: `_[r/a]_%_ab_ (r/b)_ (r-d)_`
- Web Wiz generates OR statements for every keyword and uses “space” as separator,
- Web Wiz has several DoS protections in the search feature but none of them can stop someone who can craft a query which takes 30 seconds. These DoS protections are designed for queries which might take 1-2 seconds,
- An example setup of Web Wiz Forums stopped responding within 2 minutes with a connections speed of 1.5Mb.

OTHER DBMSs

Even though other database servers are not included in this paper they are vulnerable to this attack with different search operators but SQL Server is the only major database system which comes with extra wildcards in the LIKE queries.

The following search operators are vulnerable to this problem:⁸

- PostgreSQL RegExp Match (~⁹)
- SQL Server Full Text Search Functionality (CONTAINS¹⁰)
- MS Access LIKE Queries (LIKE¹¹)
- MySQL REGEXP (REGEXP¹²)

⁸ Other major DBMSs should have similar functionalities but it's out of the scope of this paper

⁹ <http://www.postgresql.org/docs/current/static/functions-matching.html>

¹⁰ <http://msdn2.microsoft.com/en-us/library/ms142538.aspx>

¹¹ <http://office.microsoft.com/en-us/access/HP051881851033.aspx>

¹² <http://dev.mysql.com/doc/refman/5.0/en/string-comparison-functions.html>

INCREASING THE IMPACT

Some applications allow users to choose different options for the advanced search functionality. This will generally help an attacker to increase the impact.

- If an application allows you to choose the fields to search in:
 - If it searches them with **OR** operator then **select all fields**;
 - If it's using **AND** operator to do searches then select the largest field such as text data type instead of a varchar type and do not use spaces to avoid several **AND** conditions in the query.
- Choose OR conditions where possible. This is generally corresponds to the "Any of These Keywords" option in the web applications;
- Try to force the application to join tables as much as possible (*although in rare instances joins can decrease query time*);
- Generally applications will try to join conditions with **OR** operators by default, try using spaces between entered keywords as I have done in the Web Wiz Forums like : `_(x/a)_%_fm_`
`_(x/b) _ (x-d) _`

DDoS (DISTRIBUTED DENIAL OF SERVICE) ATTACKS

If an application accepts search queries over GET requests¹³ then an attacker can use a "Web Spamming Tool"¹⁴ to auto-register to open forums, send forum posts and blog comments with the URL of this search or can send this search URL as the "src" value of an image tag. In this way attacker can use other websites visitors to attack the targeted application.

Obviously it is possible for an attacker to perform XSS attacks in some popular websites and then by producing several "Iframe" or "image" tags, to attack the application. In this situation, applications that use POST requests are still vulnerable since an attacker can force victims to carry out POST requests over an XSS vulnerability.

APPLICATION LEVEL DOS

These described attacks generally will affect database servers not the application itself but from an attacker's point of view there are some possibilities for performing application level DoS by exploiting search features in the application¹⁵. This is mainly about causing the application to process so many records and consume CPU in the application server.

In wildcard attacks it is generally a good idea to not return any records since that means you have done the largest search in the index but when it comes to DoS attacks against the application itself we should try to return as many as records that we can. This will cause the application to process those records and depending on the application level algorithm and the SQL query, this might result in a big

¹³ Application shouldn't have a CSRF protection either

¹⁴ XRumer is an example of Web Spamming Tools -
<http://www.botmaster.net/movies/XFull.htm>

¹⁵ Logical problems or different Application Level DoS Attacks like filling up the database, locking accounts, password reset spam are out of the scope of this document.

impact. If the application actually tries to return 100.000 records because of a buggy paging SQL query, this will consume a lot of CPU in the application and database layers.

One real world example of this can be seen in Subsonic ORM system¹⁶, where if an attacker can send “-1” as the requested data page number, the application will try to return all results in the dataset.

Paging issues are not only common in search pages, they are also quite common in other listing pages where query conditions cannot be controlled by the attacker. Supplying large numbers, negative numbers and unexpected input can force an application to return all of the results in one response.

SQL INJECTION AND DOS ATTACKS

I have talked about DoS in the application without SQL Injection but if an application is vulnerable to SQL Injection DoS attacks are quite trivial. Different DBMSs provide different functions to achieve this. An infinite loop with some expensive functions¹⁷ will consume the CPU as soon as the connection times out. Some DBMSs also provide specific functions such as BENCHMARK in MySQL which allows executing a function repeatedly.

Using SQL Injection the possibilities of DoS are endless, for example an attacker can simply DROP all tables in the database. Depending on the privileges of an attacker, they can even shutdown the system or database server.

PROTECTION

It is important to realise that wildcard attacks are not the result of a flaw at the database level, it is the application that is at fault. The application should be responsible for filtering wildcards. The subsections below detail some mitigation strategies which can be implemented at the application level to protect against wildcard attacks.

LIMITING SEARCH

If the application does not require this sort of advanced search, all wildcards should be escaped or filtered.

Alternatively, the application should restrict access to the advanced search functionality by implementing CAPTCHA.

In order to maintain usability, an application could implement different solutions such as only allowing registered users to access the search functionality. Each user should be given a quota of searches per day or a quota of CPU time to prevent registered users from mounting wildcard attacks. Obviously registration would need to implement a CAPTCHA solution, to prevent automated registration.

Since longer queries generally take longer to process a limit on the number of characters or words in a query could be enforced.

¹⁶

<http://www.codeplex.com/subsonic/WorkItem/View.aspx?WorkItemId=16112>

¹⁷ Such as SHA1() in MySQL

WHITE LISTING

Where possible the application should have a white list of accepted characters. After white listing wildcard characters should be escaped unless wildcard queries are required.

CAPTCHA SECURITY

This paper refers to CAPTCHA¹⁸ as a security mechanism. However, weak CAPTCHA implementations can be a problem, for example:

- CAPTCHAs that actually be solved by a computer;
- Poor implementation of CAPTCHAs at the application level – i.e. Where the CAPTCHA solution is stored in an image filename or ALT attribute.

In some conditions implementing CAPTCHA or similar functionality might not be enough. In this paper I showed some examples where a query takes about **40 seconds**. This is not acceptable by any means. In this situation even a manual attack can cause DoS using this sort of queries.

LIMITING SQL QUERY EXECUTION TIME

To improve security, the database connection timeout could be reduced, or the execution time of for search queries could be limited. However, limiting execution time for search queries might not be an easy task in some web application frameworks or languages.

PREVENTING APPLICATION LEVEL DoS

Limiting the number of records returned by database queries using LIMIT or TOP or similar keywords will help to prevent application level DoS attacks.

Typically each record returned to the application will need to be processed before being sent to the user – *e.g. converted to presentation markup (often HTML)*. Processing 1 million records for example would most likely cause a significant load on the application layer.

The application should carry out strict checking over user input, especially in the record paging functionality. Such as input should be numeric, should be positive and should not be more than the defined maximum number of records.

CSRF PROTECTIONS

CSRF¹⁹ (*Cross-site Request Forgery*) protections are essential to prevent the types of DDoS attacks discussed above. This is particularly important for pages which are CPU-intensive to generate.

¹⁸ Completely Automated Public Turing test to tell Computers and Humans Apart - <http://en.wikipedia.org/wiki/Captcha>

¹⁹ http://en.wikipedia.org/wiki/Cross-site_request_forgery

DOCUMENT HISTORY

- 20/04/2008 – Idea & Experimentation
- 25/04/2008 – Initial Write-up and Peer Reviews
- 28/04/2008 - Revising the paper based on Peer Reviews
- 02/05/2008 –Improvements in several sections based on new fuzz results and analyses,
- 08/05/2008 – Improvements and proof read,
- 09/05/2008 – Ready to public Release

CREDITS & THANKS

Thanks to Mark Lowe of Portcullis Computer Security for peer review, proof reading and all of his help during the research. Thanks to Bedirhan Urgan of WTG (www.webguvenligi.org) and Sumit Siddarth of Portcullis Computer Security for peer reviews.