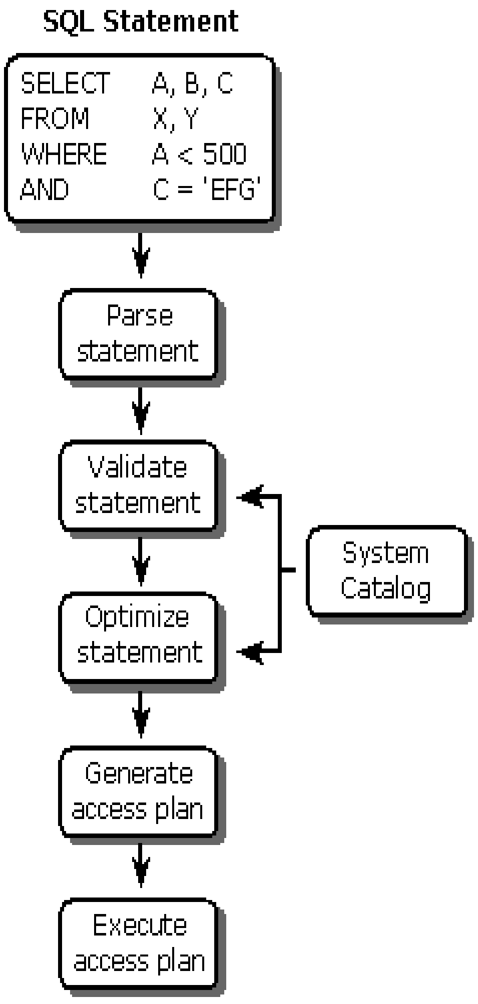
SQL INJECTION ATTACKS

By Zelinski Radu, Technical University of Moldova

Where someone is building a Web application, often he need to use databases to store information, or to manage user accounts. And also very often, developers does not think that the security of their application depends of how they interact the database with the script. Any database use SQL language to manage the data stored in it. And an specific attack that can use this is so-called "SQL Injection Attack". SQL injection is a attack technique which is used to pass SQL commands through a Web application directly to the database by taking advantage of insecure code's non-validated input values. Its source is the incorrect escaping of dynamically-generated string literals embedded in SQL statements. It is in fact an instance of more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.

SQL is a keyword based language. Each statement begins with a keyword, such as SELECT, INSERT, or other. The basic structure of SQL is the query. A query consists of a single or more than one statement. The statement of SQL consists of clauses.

We can represent a SQL statement, and how it is implemented this way:



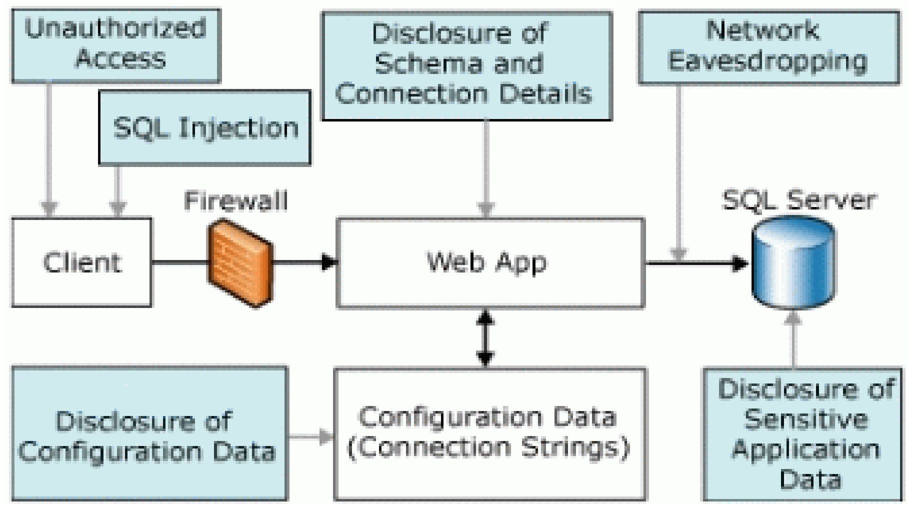
By using multiple statement queries, we can force the database to execute the statements one after other, in a more complex query.

We already said that SQL Injection in Web applications works using the dynamically-generated SQL queries. What is a dynamic query and how it works? Most Web applications that use databases do a specific task. But a lot of applications must build and process a variety of SQL statements at run time. Such statements can, and probably will, change from execution to execution. They are called dynamic SQL statements. Unlike static SQL statements, dynamic SQL statements are not embedded in our source code. Instead, they are stored in character strings input build by the program at run time.

For our applications, we are not often using the most complex forms of dynamic SQL, but the common method of building SQL queries from user's input. The basic SQL injection attack uses the multiple statement method to insert his SQL command into the general query string.

SQL injections can be very dangerous for the integrity of our Web application. Here are the most common dangers for our databases, caused by SQL injections:

- for this attack, only 80 port is needed;
- any random visitor can access the database;
- the attacker can change the information stored in our database;
- the attacker can delete the information from database;
- it is possible to full control the database.


Unauthorized Access
SQL Injection
Disclosure of Schema and Connection Details
Network Eavesdropping
Firewall
SQL Server
Client
Web App
Disclosure of Configuration Data
Configuration Data (Connection Strings)
Disclosure of Sensitive Application Data

The above picture represent how SQL injection works.

There are a lot of methods to protect ourselves from these attacks. We will present an overview of classic methods of protection against SQL injections.

The simplest way is to constrain input by validating it for type, length, format and range. Any input provided by a visitor can be harmful. It is better to use filter routines to sanitize the code by adding escape characters to characters that have special meanings to SQL. But these types of filter routines can be bypassed by an attacker that uses ASCII hex characters. So they should be used as just another part of our defense strategy.

Because the possibility of SQL injections is caused by dynamically-generated SQL queries, we can avoid the injections if we never use this kind of SQL in our code. That means we have to use parametrized queries and stored procedures, so the injections cannot occur against our application.

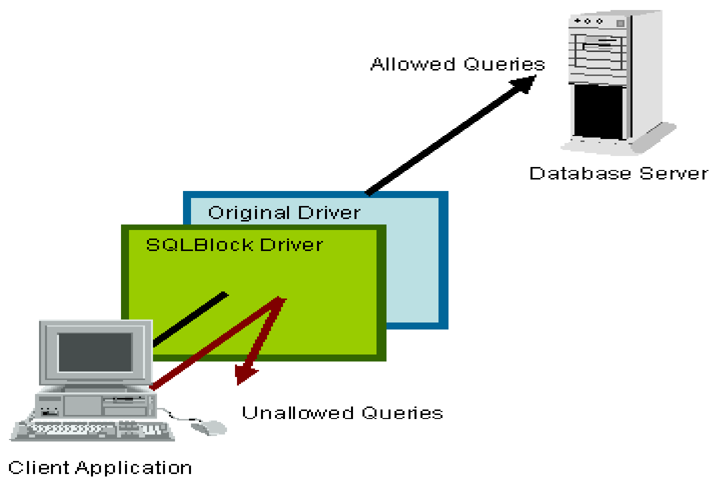
Stored Procedures add an extra layer of abstraction in to the design of a software system. This means that, so long as the interface on the stored procedure stays the same, the underlying table structure can change with no noticeable consequence to the application that is using the database. This layer of abstraction also helps put up an extra barrier to potential attackers. If access to the data in SQL server is only ever permitted via stored procedures, then permission does not need to be explicitly set on any of the tables. Therefore, none of the tables should ever need to be exposed directly to outside applications. For an outside application to read or modify the database, it must go through stored procedures. Even though some stored procedures, if used incorrectly, could potentially damage the database, anything that can reduce the attack surface is beneficial. Stored procedures can be written to validate any input that is sent to them to ensure the integrity of the data beyond the simple constraints otherwise available on the tables. Parameters can be checked for valid ranges. Information can be cross checked with data in other tables. For example, consider a database that has the user details for a website, this includes the user name and password. It is important that an attacker is unable to get a list of passwords or even one password. The stored procedures are designed so that a password can be passed in, but it will never put a password in any result set. The stored procedures for registering and authenticating a user for the website might be:

RegisterUser;
VerifyCredentials;
ChangePassword.

RegisterUser takes the user name and password as parameters (possibly along with other information that is necessary for registering on the website) and returns the UserID. VerifyCredentials would be used for logging into the site by accepting the user name and the password. If there is a match the UserID is returned, if not then a NULL value. ChangePassword would take the UserID, the old password and the new password. If the UserID and the password match, the password can be changed. A value that indicates success or failure is returned.

All SQL servers support such a concept as parametrized queries. This is where the SQL command uses a parameter instead of injecting the values directly into the command. The particular second-order attack would not have been possible if parametrized queries had been used. While many second-order attacks can be prevented by using parameters, they can only be used in places where a parameter is permitted in the SQL statement.

Another method of prevention can be use of ODBC/JDBC proxy drivers. It works as an ordinary ODBC/JDBC data source and monitor every SQL statements being executed. If the client application tries to execute any un-allowed SQL statements, the driver will block the execution and will send alert to administrator. The blocking operation looks like this:



Another biggest problem of almost all Web applications is that, for connecting to the database, the script uses DB administrator's account, so the injected statements runs on the database with administrator's privileges. In order to prevent the injections performed by usual visitors, it can be a good idea to run every query to database under a different user, with strictly-defined privileges. That means, that for an usual authentification, the only needed privilege of the user who runs the query is to select the data from the database, not to insert or to delete. If we have a registration form, the only needed operation is to insert the data. The problem is that still, any visitor can make changes to the database, and even he will not be able to attack the most important data, the DB administrator will be forced to verify the data often.

It is possible to avoid SQL injections by evaluating the syntax or the semantic sense of the query, and not only to block the characters, or the operations a user can do. That means, before executing the query, a script, which we will call evaluator, will analyze the query, and if it can be executed, will return "true", and then execute, but if the evaluator will return "false", the query will not be executed.

Before starting analyze how the evaluator will work, we have to understand the notions we will use. First of all, it will be impossible for evaluator to work, if we will not be able to formalize the SQL query. In formal sciences of logic and mathematics, together with the allied branches of computer science and information theory, a formal system is a formal grammar used for modeling purposes. Formalization is the act of creating a formal system, in an attempt to capture the essentials features of a real-world or conceptual system in formal language. The discipline designed for discussing formal systems is called metamathematics. The language described by a formal grammar is called an object language. In our case, the object language is SQL.

Formal systems consists of the following:

- a finite set of symbols which can be used for constructing formulae;
- a grammar, which is a way of constructing well-formed formulae out of the symbols, such that is possible to find a decision procedure for deciding whether a formula is a well-formed formula (WFF) or not;
- a set of axiom or axiom schemata : each axiom has to be WFF;
- a set of inference rules;
- a set of proofs. This set includes all the axioms, plus all WFF which can be derived from previous-derived proofs by means of rules of inference.

It is not the issue of this work to analyze how formal system works, so, after the preview on foundations of formal systems, we will show how this principle works as protection against SQL injections.

There are two ways of modeling formal structure of SQL : by formalizing the syntax of the language, of by formalizing the semantic structure. In first case we are working with basic structures of the language, such as keywords, separators, in the second case – with syntactic blocks, composed of keywords, separators. Because the syntactic level is more simple, we will start with it.

A SQL query is a phrase, beginning with a keyword, such as SELECT, INSERT, DELETE, and ending with the separator character ";". A query can be single or multiple statement. SQL injection is working by inserting in the user's input a statement, so the original single statement query becomes a multiple statement one.

The following structures are used to build a SQL query:

keywords

separators, such as (,) , ' , ;
identifiers (databases, tables, fields names),
and, in the case of applications, input values, entered by users.
The standard SQL single statement query looks like this:

keyword ... ('input value');

In the case of SQL injections, the attacker include in input value a statement, which is being executed by the SQL interpreter. The statement looks like this:

keyword ... ('input value'; attacking statement');

The query becomes a multiple statement one, where the instructions are executed one after other.

If we will formalize the SQL query, so the multiple statement ones to be excluded, the problem of injections will be solved.

For this, we have to create a template, which will be used by the evaluator to check the syntax of the query. The template will look like this:

<u>TEMPLATE</u>

keyword
...
('
input value
')
;

The "..." structure will contain the identifiers provided by the script, so the attacker cannot change them. This structure will be defined as a block, starting after the keyword, and beginning before "('"; The evaluator will always consider this block as one.

The formalized phrase will be used as a template for evaluation. Let's consider that the user is trying to login, and he is perform a valid username. The script will compare the username performed by user with the values stored in the database. Because the username is valid, the result will be the correct authentification. How the evaluator will works? The example above will be evaluated in this way:

INPUT	TEMPLATE	X&Y
keyword	keyword	true
...	...	true
('	('	true
value	value	true
')	')	true
;	;	true

If the user will try to insert in his input a statement, the evaluation will look this way:

INPUT	TEMPLATE	X&Y
keyword	keyword	true
...	...	true

('	('	true
value	value	true
'; statement	')	false
')	;	false
;		false

How the evaluator will decide if the whole phrase can be executed or not by the SQL interpreter? There are three methods of evaluating the query:

- iteration of "&" logic function;
- logic analyze;
- neural interpretation.

The first method looks will take the X&Y values, and will iterate the function like this:

(X&Y)1 & (X&Y)2 & ... & (X&Y)n = "true", or "false".

If all the X&Y values are "true", as in the first case, the returned value will be "true", and the query will be executed. If at least one value is "false", the query will not be executed.

Logic analyze can be implemented with so-called expert systems. The idea is that, if our logic analyzer will find at least one "false" value, the query will be considered as "false", and will not be executed.

With neural interpretation, the evaluator will use the structure called "neuron" to analyze the syntax of the query. A neuron is a structure, with a n number of inputs, and one output. The input values can be or "true", or "false", and the output, by passing the input values through its activation function, will return also or "true", or "false".

Activation function is the function that's analyzing the inputs, for returning the logic value. Most commonly used activation function is so-called "sigmoid function", that looks this way:

y = 1 / 1 + exp(-ax), where "x" is the input vector, and "a" is a constant value.

Input vector x looks like this:

x = w1x1 + w2x2 + ... +wnxn, where xn is the value of an input, and wn is synaptic weight of an input. xn is "true" or "false".

The output y value depends of input values and synaptic weights. The advantage of neural evaluation is that code can be used not only for evaluating SQL queries, but also for a lot of other operations, only by changing the values of weights, which is called "learning process".

The algorithm for all types of SQL statements, all depends of formalization degree. For semantic evaluation, the formalization needs to be at a higher level than for syntax evaluation.

Such method of preventing SQL injection attacks has the advantage that is universal, and can be implemented in all scripting languages. Another advantage is that we do not have to specify for every input form the values that can be interpreted. This method is assuring a secure dynamically generated SQL query, which is more comfortable to use that stored procedures. The evaluation algorithm is based on the logic algebra, and the final code will be very small, which is an advantage for programmers. All other methods cannot be used effectively all alone, that's for, by using classic methods of protection, the coder was forced to use a complex of different measures. SQL injection attacks are very dangerous for Web application's security, and this issue cannot be ignored by Web developers. With this method, the injection is no more a danger for our applications.