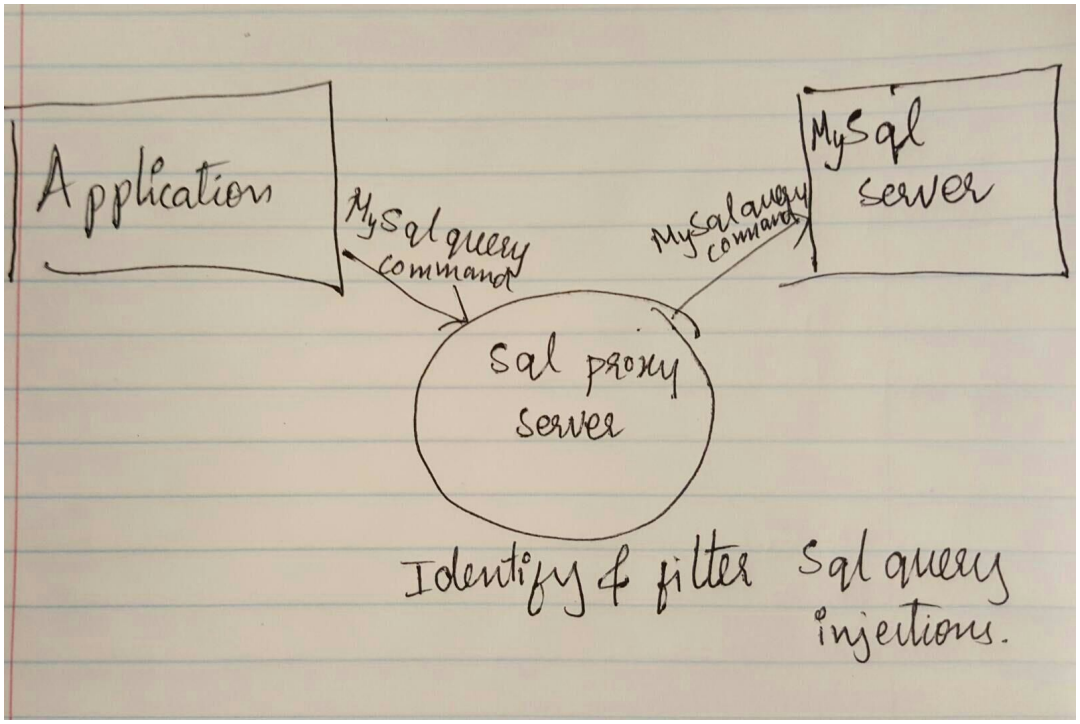
SQL Injection Prevention

Describe the Problem you are solving.

SQL injections occur when the user input is not sanitized. This attack can jeopardize the confidential information or modify the server databases. These kind od attacks can be avoided if the developer sanitizes the user input. But in many cases the they are not validated before it is presented to the MySQL server.
Our project is an attempt to detect SQL injections and prevent attacks on the MySQL database.

Describe the approach to implement.

Interaction Between Application, SQL Proxy Server Library and SQL Server.



- All the traffic to the MySQL server is intercepted by the SQL Proxy Server Library.
- SQL Proxy receives the SQL queries from the User application.
  - If there is no SQL injection, then the query is forwarded to the MySQL server
  - If there is SQL injection, then it is detected. The query is not forwarded to the MySQL server. Instead the warning packet is sent to the application.

- All the packet exchange between the Application, SQL proxy server and MySQL server adhere to SQL Client-Server protocol. (https://dev.mysql.com/doc/internals/en/client-server-protocol.html)

Advantages of using SQL-Proxy Server Library instead of Client-side Library

- **Programming Language Agnostic:** Our library is not dependent on the programming language the application uses because it intercepts the SQL packets.
- **Ease of use:** Developer need not make any changes to his application other than the MySQL port number.
- **Reducing MITM attack:** In the architecture the SQLproxy server is close to the MySQL server, so the MITM attack is reduced and all the queries are filtered before reaching the MySQL server.

Logic on how non-sanitized SQL queries are detected

We look for the cases that were taught in the course and in OWASP.org. Generally, our logic for all the cases depend on how SQLParser(with professor's permission we are using SQLParser library) tokenizes the input string.

1. **TAUTOLOGY (Always true and Always false conditions):** We parse the condition part of where clause in select statement. Each keyword/value is labelled with either unknown, true, false, string or int. We then created a ruleset that represents the logical and conditional rules that is applied on the condition part. Based on the rules, new label for the tokens get changed each step. If at the end, the result comes to true or false, then SQL injection is detected and an error packet is sent to client without communication with server. Here the checking of tautology is limited to logical and comparison conditions. Example: Select id from table where id = 1 or 1=1 which results in unk=int or int=int. In the next iteration it becomes unk or true which results in true. A Sql injection detected.
2. **Multiple Commands:** We parse all the tokens from the parser and check if there is a token match for semi-colon (;). If semi-colon is present and is not inside the user string, then the error SQL packet is sent to the application.
3. **Comment at the end:** We parse final token and check for the presence of the comment(-- or /*). If the comment is present, then the error SQL packet is sent to the application.
4. **Union, Except and Intersect commands:** We parse all the tokens from the parser and check if there is a token match for keywords UNION, EXCEPT AND INTERSECT(;). If they are present and are not inside the user string, then the error SQL packet is sent to the application.

Describe what a developer or security expert must do to use your approach

In order to integrate our security library to any of the project. Developer or security expert must run the jar file by providing the hostname as ip address of SQL Database, remote port as port number of SQL Database and local port as command line argument. This will start a proxy server will be intercept all the sql query. The developer should change the port number and ip address of sql server to proxy server.

Please refer to readme file for instructions for how to run the project.

Describe the limitations of your tool: What types of applications can it be applied to?

- Dependent on the Database: We chose to provide library to MySQL because that is one of the predominantly used database. We intercept the SQL packets that adhere to MySQL protocol so it is not possible to support other databases.
- Adds Overhead: Adding proxy server always adds overhead. Packet interception and processing to filter for SQL-injections increases processing time.

Describe the future work that you can see which would improve your tool.

- Our project is not Database agnostic. This project can be extended to support multiple databases like PostgreSQL, MySQL or MangoDB.
- At present, we are not handling second order injection.