Netzwerkforensik: Erkennung von SQL-Injections

Computerforensik

Stefan Braun

23. Mai 2016

Technische Hochschule Ingolstadt

Inhaltsverzeichnis

1 SEITE 3
SQL-Injections im Jahr 2016

1.1 Verhinderung von SQL-Injections *3*
1.2 Alternativen *4*

2 SEITE 6
Arten von SQL-Injections

2.1 Tautologie-basierte Injection *6*
2.2 UNION based Injection *7*
2.3 Statement Injection *7*
2.4 Error based SQL Injection *7*
2.5 Time based SQL injection *8*

3 SEITE 9
Versuchsbeschreibung

3.1 Grundlegende Überlegungen *9*
3.2 Verwendete Werkzeuge *9*
3.3 Versuchsablauf *9*
3.4 Automatisierung der SQL-Injections mit sqlmap *10*
3.5 Verwundbare Beispielapplikation *11*
3.6 Apache und ModSecurity *11*
3.7 MySQL Enterprise: Datenbank und Firewall *12*

4 SEITE 13
Ergebnisse

SQL-Injections im Jahr 2016

1

Alle drei Jahre veröffentlicht das *Open Web Application Security Project* – kurz OWASP – eine Liste der derzeit als am kritischsten eingestuften Sicherheitsrisiken in Webapplikationen. Und auch in der derzeit aktuellsten Fassung der Liste aus dem Jahr 2013 findet sich die Kategorie „Injections“ auf Platz Eins wieder.

Derzeit werden Daten für die kommende OWASP Top Ten 2016 gesammelt.

	Kategorie
1	Injection
2	Broken Authentication
3	Cross-Site-Scripting

Tabelle 1.1: Die ersten drei Kategorien der aktuellen OWASP Top Ten aus dem Jahr 2013, nach `www.owasp.org`

Derartige Angriffe basieren darauf, dass Benutzereingaben ungeprüft in Abfragen an LDAP-Dienste und vor allem SQL-Datenbanken als Parameter eingefügt werden. Entsprechend geformte Eingaben können somit die grundlegende Struktur der Anfrage manipulieren. Diese Manipulation kann Verlust der Informationsvertraulichkeit oder der Datenintegrität zur Folge haben, unter Umständen kann ein Angreifer Vollzugriff auf die zugrundeliegende Serverstruktur erhalten. Die vorliegende Arbeit konzentriert sich hierbei insbesondere auf gefährdete SQL-Anfragen.

1.1 Verhinderung von SQL-Injections

„Don't trust user input.“

Es stellt sich folglich die Frage, wie derartige Angriffe verhindert werden können. Die übliche Vorgehensweise stellt hierbei die Überprüfung der vom Client übergebenen Parameter dar. Etwa könnte unter PHP ein Parameter, für den nur Ganzzahlen vorgesehen sind, per Konvertierung durch `intval()` abgesichert werden. Bei beliebigen Zeichenketten escaped die Funktion `mysql_real_escape_string()` bestimmte Zeichen, die einen Ausbruch aus der Abfrage erlauben können. Sicherer sind allerdings sogenannte *Prepared Statements*, die die Anfrage und die zugehörigen Parameter getrennt voneinander übertragen und dadurch Injections verhindern.

Diese PHP-Funktion ersetzt beispielsweise ' durch \'. Dadurch wird es erschwert, das aktuelle String-Literal im SQL-Statement zu beenden und zusätzliche Befehle anzufügen.

Wenn also die Verhinderung von SQL-Injections eine triviale Angelegenheit ist, weshalb bestimmen auch heutzutage Nachrichten über aktuelle, derartige

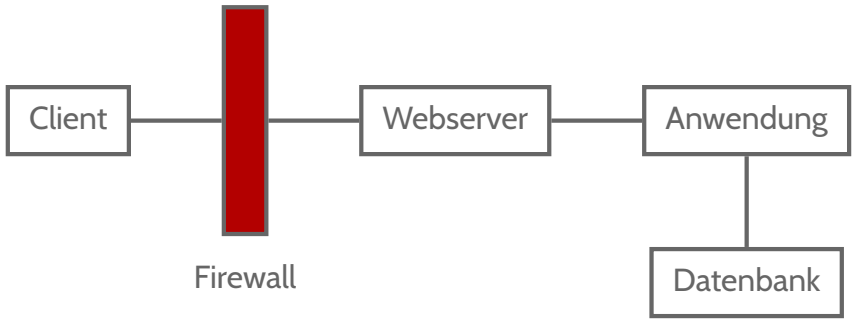

Abbildung 1.1: Zugriffe auf eine Webserverapplikation passieren üblicherweise zuerst eine Firewall und werden anschließend von einem Webserver – etwa *Apache* – zur Applikation weitergeleitet. Diese Applikation kann anschließend auf den Datenbankserver zugreifen.

Angriffe die Fachpresse? Die Gründe hierfür sind vielfältig. Möglicherweise ist veraltete Software im Einsatz oder dem Entwickler mangelt es schlicht an Vorwissen im Bereich der IT-Sicherheit. Ebenfalls vorstellbar ist Software, die nicht mehr geändert werden kann – etwa weil der Aufwand zu groß wäre, keine Entwickler vorhanden sind, oder aber der zugehörige Sourcecode nicht zur Verfügung steht. Außerdem können Queries konstruiert werden, die ein fachliches Problem zwar auf einfache Weise lösen, andererseits jedoch die Verwendung eines parametrisierten Prepared Statements unmöglich machen.

Ein weiteres Beispiel könnte zugekaufte Fremdsoftware darstellen, die im eigenen Netzwerk betrieben wird.

```
1 $query  = ""
2     ."SELECT                          "
3     ."    $choosenText AS myText,     "
4     ."    name                        "
5     ."FROM                            "
6     ."    report                      "
7     ."ORDER BY                        "
8     ."    name $sorting               ";
9 mysqli_query($connection, $query);
```

Listing 1.1: In diesem PHP-Code wird mit der Variable `choosenText` eine Spalte und mit `sorting` eine Sortierreihenfolge ausgewählt. In beiden Fällen können keine Parameter für Prepared Statements verwendet werden.

1.2 Alternativen

In all diesen Fällen muss die gefährdete Applikation also auf andere Art und Weise abgesichert werden. Ein gängiger Ansatz zur Realisierung einer solchen Schutzmaßnahme stellt eine vorgeschaltete Softwarekomponente dar, welche auf Basis von Filterregeln einzelne Requests verwirft oder modifiziert. Hierzu soll zuerst ein übliches Schema einer Client-Server-Architektur skizziert werden.

In aktuellen Webserverarchitekturen können weitere Komponenten enthalten sein, die an dieser Stelle jedoch vernachlässigt und abstrahiert werden sollen.

Ein Beispiel hierfür stellen etwa *Load Balancer* zur Lastverteilung auf mehrere Server dar.

In dem abstrakten Schema eines Requests aus Abbildung 1.1 bieten sich zwei

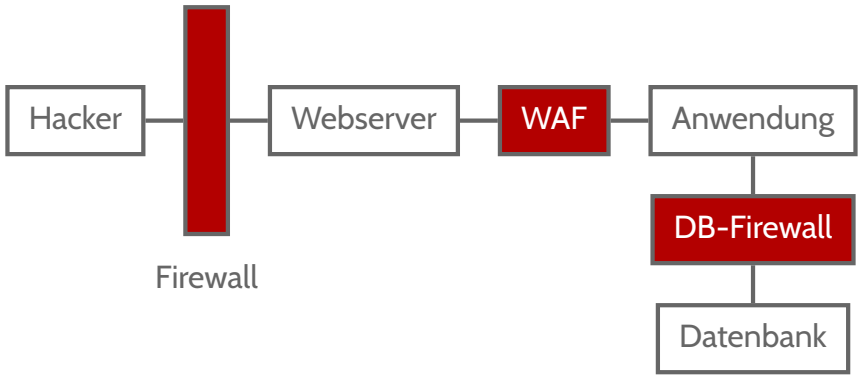

Abbildung 1.2: In die Abbildung 1.1 wurden an den entsprechenden Stellen Schutzmechanismen eingefügt. Möglichkeiten hierfür sind eine *Web-Application Firewall* – kurz WAF – und eine *Datenbank-Firewall*. Es stellt sich die Frage, wie effektiv die jeweiligen Maßnahmen SQL-Injections mitigieren können.

Stellen an, an welchen einzelne Parameter der Anfrage auf ihre Gefährlichkeit in Bezug auf SQL-Injections hin untersucht werden können. Analysiert man die beispielsweise die GET und POST Parameter eines Requests noch bevor sie beim Webserver ankommt, spricht man von einer *Web Application Firewall*. Alternativ können auch die Zugriffe auf den Datenbankserver selbst untersucht werden – und zwar von einer vorgeschalteten *Datenbank-Firewall*. Es stellt sich die Frage, inwiefern die beiden Ansätze in Bezug auf ihre Effektivität miteinander vergleichbar sind – und ob sie einen wirksamen Schutz vor SQL-Injections bieten können.

Sowohl Web Application Firewall als auch Datenbank-Firewall stellen *Intrusion Detection* Systeme dar.

2 Arten von SQL-Injections

SQL-Injections lassen sich nach unterschiedlichen Kriterien klassifizieren. Dies geschieht in Hinblick auf die Art und Weise wie ein Angreifer die verwundbare SQL-Anfrage entdeckt, wie er den Schadcode einfügt und schließlich Daten auslesen kann. In diesem Kapitel soll ein grober Überblick über verschiedene Ansätze von SQL-Injections vermittelt werden.

2.1 Tautologie-basierte Injection

Die einfachste Variante einer SQL-Injection sorgt dafür, dass eine logische Überprüfung im `WHERE`-Teil der Abfrage immer zu `true` evaluiert und somit alle betroffenen Zeilen zurückgegeben werden. Das Paradebeispiel hierfür ist eine Abfrage zur Authentisierung eines Benutzers.

```
$query  = ""
    ."SELECT                        "
    ."    username                  "
    ."FROM                          "
    ."    users                     "
    ."WHERE                         "
    ."    username = '$username'    "
    ."AND                           "
    ."    password = '$password'    ";

$result = mysqli_query($connection, $query);

if(mysqli_num_rows($result) != 0){
    $_SESSION['user_logged_in'] = $username;
}
```

Listing 2.1: Eine einfache Anmeldungslogik: Wird in der Datenbank ein Nutzer mit dem übergebenen Nutzernamen und Passwort gefunden, wird eine Sessionvariable gesetzt.

Wenn ein Angreifer die Logindaten jeweils auf `' OR '' = '` setzt, wird jeweils überprüft, ob ein Leerstring identisch zu einem Leerstring ist. Da dieser Vergleich immer `true` ergibt, werden alle in der Tabelle `users` enthaltenen Zeilen zurückgegeben. Der Benutzer wird eingeloggt.

2.2 UNION based Injection

Etwas komplexer als die im vorherigen Abschnitt vorgestellte Methode sind Anfragen, welche die Menge der zurückgegebenen Zeilen eines `SELECT` Statements durch ein angefügtes `UNION` erweitern. Wichtig ist hierbei, dass die Anzahl der Spalten der mit `UNION` angefügten Query identisch mit der Spaltenanzahl der ursprünglichen `SELECT` Abfrage sein muss. Führt der restliche Teil der originalen Abfrage zu Problemen, kann er gegebenenfalls auskommentiert werden. Hierzu muss an das Ende des injizierten `UNION` Teils per `--` ein Kommentar eingeleitet werden.

2.3 Statement Injection

Statement Injection funktioniert ähnlich wie die `UNION` basierte Variante, jedoch wird hier eine komplette, zusätzliche SQL-Abfrage eingefügt. Hierzu wird zuerst die aktuelle Anfrage valide vervollständigt und anschließend per Semikolon beendet. Nun kann ein eigenständiges SQL-Statement angehängt werden – beispielsweise `DROP DATABASE wordpress;`. Folgender SQL-Code kann wie im vorherigen Abschnitt erläutert einfach auskommentiert werden.

Damit derartige Angriffe unter PHP funktionieren, muss statt der Funktion `mysqli_query()` die Variante `mysqli_multi_query()` verwendet werden. Andernfalls ist die Verwendung konkatenierter SQL-Statements nicht möglich.

Angemerkt: In einem `xkcd` Webcomic unter `https://xkcd.com/327/` löscht eine Mutter Daten der Schule: Sie hatte ihren Sohn „`Robert'); DROP TABLE Students; --`“ getauft – Ein klassisches Beispiel für eine Statement Injection.

2.4 Error based SQL Injection

Wenn eine Anfrage zwar anfällig für SQL-Injections ist, die zugehörige Webseite allerdings keine Daten direkt ausgibt, können möglicherweise dennoch Daten ausgelesen werden. In manchen Webanwendungen wird bei einem Fehler in einer Datenbankabfrage die entsprechende Fehlermeldung ausgegeben. Kann diese Ausgabe durch die Injection provoziert und geändert werden, so ist eine *Error based SQL Injection* möglich.

```
1 $sql = "update `".DC_MV_CAL."` set"
2     . " `exdate`='" . esc_sql($exdate) . "' "
3     . "where `id`=" . $id;
4
5 if ($wpdb->query($sql)=== FALSE){
6     $ret['IsSuccess'] = false;
7     $ret['Msg'] = $wpdb->last_error;
8 }
```

Listing 2.2: Ein Auszug aus dem Wordpress-Plugin `cp multi view calendar`[a]. Die Variable `id` wird nicht überprüft und ermöglicht so SQL-Injections. Im Fehlerfall wird die Meldung in Zeile 7 in eine lokale Variable geschrieben und später ausgegeben.

[a] `https://github.com/wp-plugins/cp-multi-view-calendar`

Übergibt man als `id` etwa `EXTRACTVALUE(0x0a,(SELECT USERNAME())`, so

wird der Inhalt von Listing 2.3 ausgegeben. Die Funktion `EXTRACTVALUE` erwartet als zweiten Parameter eine gültige XPath-Anweisung. Der Benutzername, der per `USERNAME()` in einer Subquery ausgelesen wird, ist kein gültiges Argument – und wird daher in der Fehlermeldung mit ausgegeben.

```
{
    "IsSuccess":false,
    "Msg":"XPATH syntax error: '@localhost'"
}
```

Listing 2.3: Ausgabe der *Error based* SQL-Injection. In der Fehlermeldung ist das Resultat der Subquery zu sehen, in diesem Fall der Rückgabewert der Funktion `USERNAME()`.

2.5 Time based SQL injection

Ändert sich an der Ausgabe der Seite trotz erfolgreich ausgeführter SQL-Injection nichts, so ist es dennoch möglich, Daten auszulesen. Hierfür werden SQL Funktionen wie `SLEEP()` oder `BENCHMARK()` verwendet, die die Ausführungsdauer einer Query erhöhen können. Verbindet man dies mit einer `IF()` Bedingung und misst die Dauer des gesamten Requests, so erlaubt dies erneut Rückschlüsse auf Datenbankinhalte.

```
(
    SELECT
        IF(
            SUBSTRING(user.Password,1,1) = CHAR(12)
            ,SLEEP(5)
            ,2
        )
    FROM
        mysql.user
    LIMIT
        1
)
```

Listing 2.4: Diese Query vergleicht ein einzelnes Zeichen einer Zeichenkette mit einem bestimmten ASCII-Code. Liefert der Vergleich `true`, so wird fünf Sekunden gewartet.

3 Versuchsbeschreibung

Folgende Abschnitte sollen aufzeigen, wie im Zuge dieser Seminararbeit die Tauglichkeit verschiedener automatisierter Angriffs- und Verteidigungsmechanismen rund um SQL-Injections getestet wurde.

3.1 Grundlegende Überlegungen

Wenn die Bewertung einer derartigen Abwehrmaßnahme zum Thema wird, kommen zwei einfache Metriken in Frage: die Anzahl der *false positives* und die der *true negatives*. Wie viele HTTP-Anfragen führen zwar zu SQL-Injections, werden jedoch nicht erkannt – und auf der anderen Seite: Wie viele Anfragen werden verworfen, obwohl sie eigentlich ungefährlich sind?

Je eines der Beiden zu 100% zu erfüllen ist einfach: *false positives* werden verhindert, indem die Firewall alle Anfragen blockiert, *true negatives* treten nicht auf, wenn keine einzige Überprüfung stattfindet oder die Abwehrmaßnahme komplett deaktiviert wird. Da beide Extreme üblicherweise unerwünscht sind, gilt es ein passendes Mittelmaß zu finden.

Es sei zudem darauf hingewiesen, dass die Nichterkennung einer SQL-Injection größeren Schaden verursachen kann als das versehentliche Verwerfen einer normalen Anfrage.

Um ein Beispiel zu geben: Einem harmlosen Benutzer, der sich `"DROP DATABASE"` nennen möchte, eine Fehlermeldung zu präsentieren, ist der Wirtschaftlichkeit der Website meist weniger abträglich als das Löschen der kompletten Datenbank.

3.2 Verwendete Werkzeuge

Bezugnehmend auf die Abbildung 1.2 wird jeweils eine Web Application Firewall und eine Database Firewall getestet. Damit diese Tests möglich sind, muss zum einen die grundlegende Infrastruktur aufgebaut werden: Die Basis bildet ein Apache 2.4 Webserver auf einem Linux-Serversystem. Damit Angriffe möglich sind, muss eine für SQL-Injections verwundbare Applikation eingerichtet werden. Mögliche Schwachstellen werden schließlich automatisiert mit dem Tool *sqlmap* gesucht und getestet.

Zum Einsatz kommt ein 64-bit Ubuntu Server. Siehe auch `http://www.ubuntu.com/download/server`

3.3 Versuchsablauf

Schwachstellen, die mit sqlmap entdeckt wurden, werden als `Request` Objekte in einem Pythonskript zusammengefasst. Anschließend wird je eine der beiden Abwehrmaßnahmen aktiviert. Das Pythonskript führt die Requests der Reihe nach aus und überprüft ob die SQL-Injection erfolgreich abgefangen wurde.

Abschließend wird eine Statistik der erfolgreichen und der abgewehrten SQL-Injections ausgegeben.

3.4 Automatisierung der SQL-Injections mit sqlmap

Zum Auffinden verwundbarer Request-Parameter wurde das Python basierte Werkzeug *sqlmap* ausgewählt. Es bietet verschiedene Kommandozeilenoptionen an, deren Verwendung an dieser Stelle kurz erläutert wird. Das frei verfügbare Tool kann von `http://sqlmap.org/` aus installiert werden.

Vergleiche hierzu `https://github.com/sqlmapproject/sqlmap/wiki/Usage`

```
> python sqlmap.py -u "http://www.example.com/app.php?id=1"
```

Mit dem Parameter `-u` wird die anzugreifende URL angegeben. Bereits dieses Argument genügt für eine erste Analyse: Standardmäßig wird ein `GET`-Request verwendet und dabei versucht, alle in der URL enthaltenen Parameter anzugreifen. Hierbei setzt sqlmap verschiedene mögliche Payloads für die angegeben Parameter ein und überprüft anhand der Serverantwort, ob die SQL-Injection erfolgreich war.

```
> python sqlmap.py -u "http://www.example.com/app.php"
↪ --data={username=&password=}
```

sqlmap ist ebenfalls in der Lage, `POST`-Requests durchzuführen. Hierzu werden die Parameter mit dem Argument `--data` angegeben.

```
sqlmap identified the following injection point(s) with a total
↪ of 307 HTTP(s) requests:
---
Parameter: username (POST)
    Type: boolean-based blind
    Title: OR boolean-based blind - WHERE or HAVING clause (MySQL
↪ comment) (NOT)
    Payload: uname=") OR NOT 7407=7407#&passwd=

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 OR time-based blind (comment)
    Payload: uname=") OR SLEEP(5)#&passwd=
```

Findet sqlmap verwundbare Parameter, so werden diese zusammen mit dem genutzten Payload ausgegeben. Im obigen Beispiel war der Parameter `username` angreifbar, sqlmap liefert hierzu eine time-based und eine boolean-based SQL-Injection.

```
> python sqlmap.py --level=5 -u "http://www.example.com/app.php"
↪ --data={username=&password=}
```

Gelingt es sqlmap nicht, eine SQL-Injection zu finden, so kann mit `--level` die Anzahl der getesteten SQL-Injection-Varianten erhöhen – wobei 1 die niedrigste Stufe und 5 jene Stufe mit der größten Zahl an Anfragen darstellt. Neben der Anzahl der möglichen Varianten pro Parameter werden mit höheren Stufen auch Cookies und HTTP-Header getestet. Die größere Anzahl an Anfragen hat allerdings auch eine längere Ausführungsdauer des Befehls zur Folge.

```
> python sqlmap.py  -o --dbms=MySQL -u
↪  "http://www.example.com/app.php?id"
```

Gerade hierfür kann die Verwendung der Argumente `-o` und `--dbms` sinnvoll sein. Mit `-o` werden allgemeine optimierende Optionen aktiviert – etwa Multithreading – welche die Ausführungsdauer reduzieren können. Ist das verwendete Datenbankmanagementsystem bekannt, so kann auch mit der Angabe von `--dbms` verhindert werden, dass SQL-Injections, die für andere DBMS spezifischen Code enthalten, durchgeführt wird.

Alternativ kann auch mit `--threads` die Anzahl der zu verwendenden Threads explizit angegeben werden. Der Parameter `-o` setzt standardmäßig drei Threads ein.

3.5 Verwundbare Beispielapplikation

Um die Möglichkeiten sowohl von sqlmap als auch der Firewall-Anwendungen auszuloten zu können, wird eine verwundbare Beispielapplikation benötigt. Verwendet wird an dieser Stelle *SQLI-labs* . Diese beinhaltet 53 einfache Formulare auf Basis von PHP 5 und ist als Lernplattform für SQL-Injections gedacht. Die einzelnen Formulare greifen auf GET-Parameter wie `id` oder `sort` zu, auch Loginfelder mit Benutzername und Passwort sind gegeben.

Vergleiche `https://github.com/Audi-1/sqli-labs`

Von diesen 53 Tests konnte sqlmap bei 32 mögliche SQL-Injections finden, jeweils zwischen einer und vier verschiedenen. Diese summieren sich zu 90 verschiedenen GET- und POST-Anfragen auf, die die Grundlage für den Versuch darstellen. Die gefundenen SQL-Injections lassen sich verschiedenen Kategorien zu ordnen, unter anderem Time-based und Error-based Injections waren vertreten.

3.6 Apache und ModSecurity

Die eingesetzte WAF-Lösung *ModSecurity* ist ein Webserverplugin, welches für nginx, IIS und Apache verfügbar ist. Es basiert darauf, eingehende Serveranfragen anhand von Filterregeln auf potentielle Angriffe hin zu untersuchen.

Matchen eine oder mehrere dieser Regeln, so kann die Anfrage abgelehnt oder geloggt werden .

Die Filterregeln sind nicht Bestandteil von ModSecurity und müssen zusätzlich installiert und aktiviert werden. Neben dem kostenpflichtigen Angebot der Firma Trustwave – des Herstellers von ModSecurity – können auch frei verfügbare Regelsätze des OWASP verwendet werden. Neben der Filterung von SQL-Injections werden auch XSS-Attacken und Schwachstellenscanner abgewehrt.

Entsprechend ist die Verhaltensweise ein *Intrusion Detection System* oder ein *Intrusion Prevention System*.

Die Regelsätze von Trustwave können für jährliche 495 Dollar lizenziert werden und beinhalten die OWASP Regeln. OWASP stellt seine Regeln unter `https://github.com/SpiderLabs/owasp-modsecurity-crs` zur Verfügung.

3.7 MySQL Enterprise: Datenbank und Firewall

Als Datenbanksystem wird MySQL eingesetzt. In der kostenpflichtigen Enterprise-Variante bietet dieses DBMS zusätzlich eine Datenbankfirewall an. Wie diese Firewallmechanik funktioniert, wird im Ablaufdiagramm in Abbildung 3.1 dargestellt.

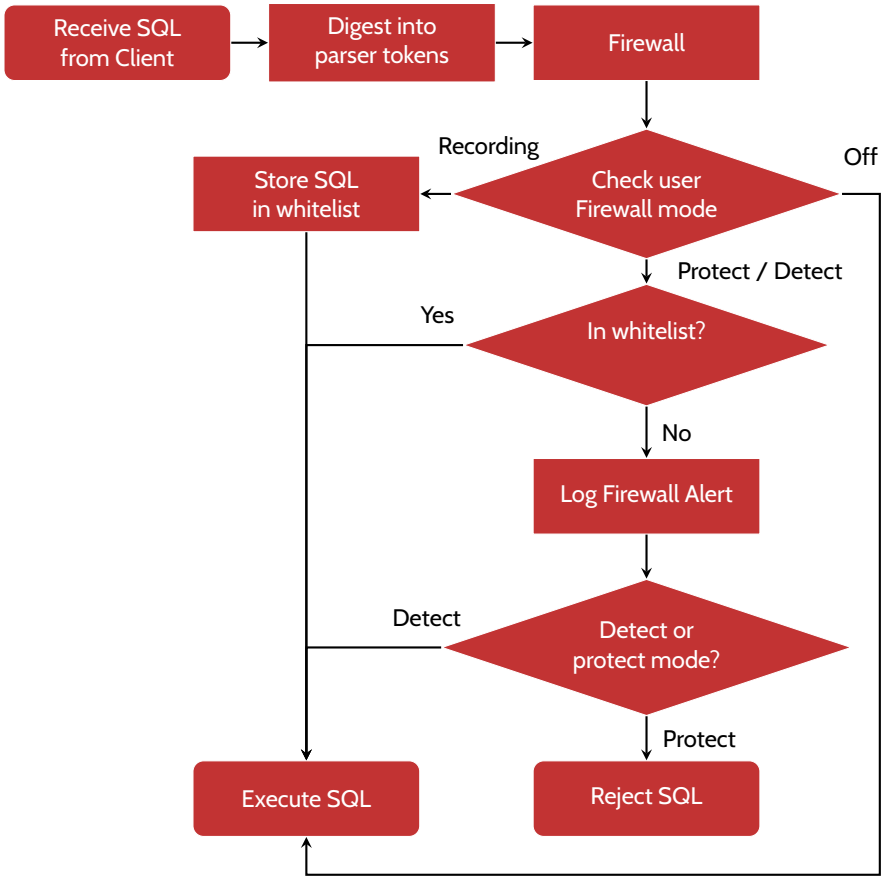

Abbildung 3.1: Diese Grafik skizziert die Funktionsweise der MySQL Datenbankfirewall.[a] Eingehende SQL-Anfragen werden geparst und in Tokens zerlegt. Pro Datenbankbenutzer existiert nun eine Whitelist mit zulässigen Abfragen: Ist der Lernmodus aktiv, so werden alle neuen Anfragen in die Whitelist mit aufgenommen. Wird die Firewall aktiviert, so werden neue Anfragen je nach Einstellung nur geloggt oder auch abgelehnt. Entsprechend kann die Datenbankfirewall wieder als IDS oder IDP konfiguriert werden.

[a] nach `http://dev.mysql.com/doc/refman/5.6/en/firewall.html`

Ergebnisse

4

90 von 90,
keine 100