# Netzwerkforensik: Erkennung von SQL-Injections

Computerforensik

Stefan Braun

23. Mai 2016

# 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

# 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

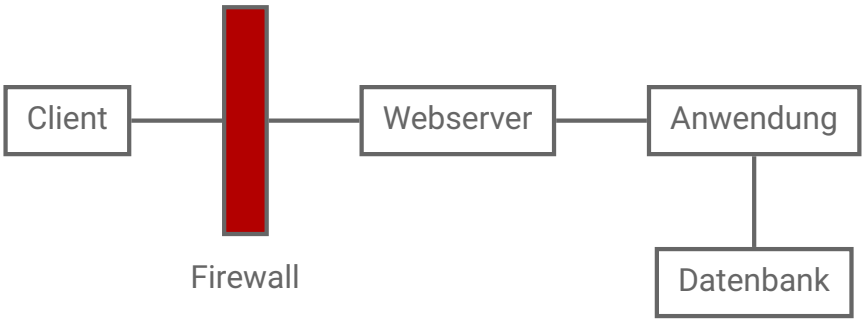


**Abbildung 1.1:** Zugriffe auf eine Webserverapplikation passieren üblicherweise zuerst eine Firewall und werden anschließend von einem Webserver – etwa *Apache* – zur Applikation weitergeleitet. Diese Applikation kann anschließend auf den Datenbankserver zugreifen.

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.

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

Außerdem können Queries konstruiert werden, die ein fachliches Problem zwar auf einfache Weise lösen, andererseits jedoch die Verwendung eines parametrisierten Prepared Statements unmöglich machen.

```
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.

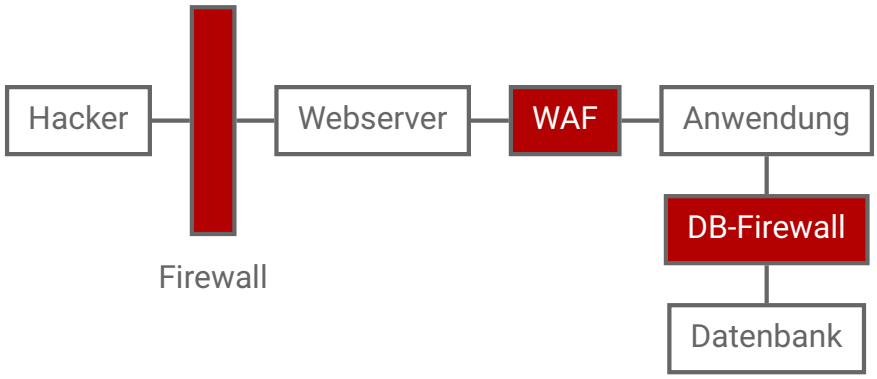


**Abbildung 1.2:** In die Abbildung 1.1 wurden an den entsprechenden Stellen Schutzmechanismen eingefügt. Möglichkeiten hierfür sind eine *Web-Application Firewall* – kurz WAF – und eine *Datenbank-Firewall*. Es stellt sich die Frage, wie effektiv die jeweiligen Maßnahmen SQL-Injections mitigieren können.

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