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
Versuchsbeschreibung

1 SQL-Injections im Jahr 2016

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

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

zugehörigen Parameter getrennt voneinander übertragen und dadurch Injections verhindern.

Wenn also die Verhinderung von SQL-Injections eine triviale Angelegenheit ist, weshalb bestimmen auch heutzutage Nachrichten über aktuelle, derartige 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 $sorting               ";
9 mssql_query($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 vorgeschaltene Softwarekomponenta 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 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,

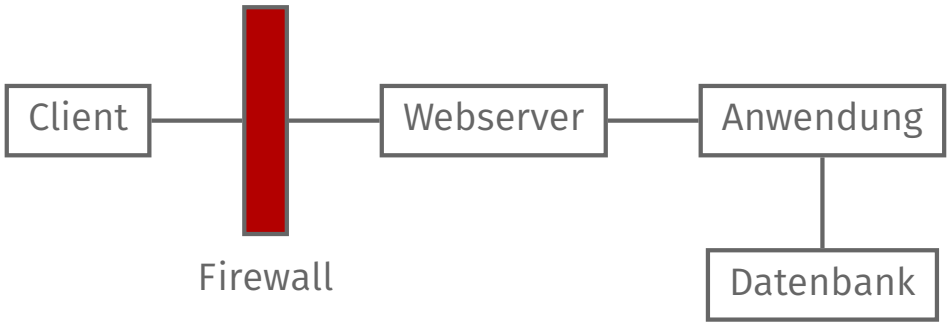

Abbildung 1.1
Zugriffe auf eine Webserverapplikation passieren üblicherweise zuerst eine Firewall und werden anschließend von einem Webserver – etwa *Apache* – zur Applikation weitergeleitet. Diese Applikation kann anschließend auf den Datenbankserver zugreifen.

spricht man von einer *Web Application Firewall*. Alternativ können auch die Zugriffe auf den Datenbankserver selbst untersucht werden – und zwar von einer vorgeschaltenen *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=Ïnjections bieten können.

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

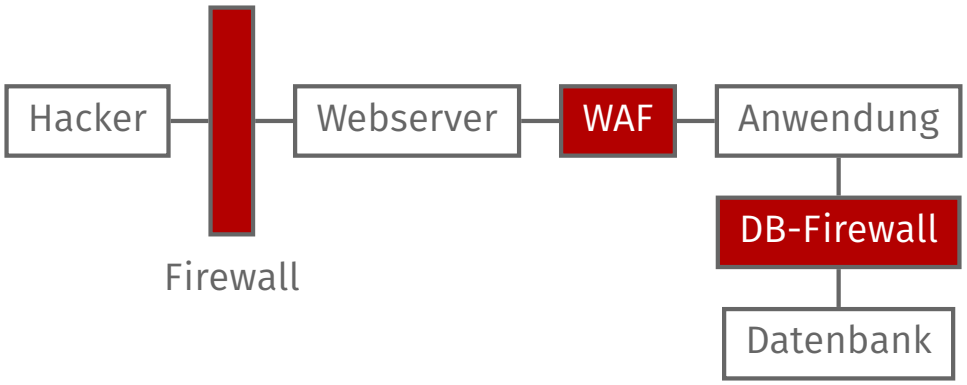

Abbildung 1.2
In die Abbildung 1.1 wurden an den entsprechenden Stellen Schutzmechanismen eingefügt. Möglichkeiten hierfür sind eine *Web-Application Firewall* – kurz WAF – und eine *Datenbank-Firewall.* Es stellt sich die Frage, wie effektiv die jeweiligen Maßnahmen SQL-Injections mitigieren können.

2 Versuchsbeschreibung