Angriffe auf Datenhaltung - „SQL Injection“

Daniel Schosser
Betreuer: Nadine Herold
Seminar Angriffe auf Datenhaltung WS1112
Lehrstuhl Netzarchitekturen und Netzdienste, Lehrstuhl Betriebssysteme und Systemarchitektur
Fakultät für Informatik, Technische Universität München
Email: schosser@in.tum.de

KURZFASSUNG

SQL ist die am weitesten verbreitete Sprache zur Definition von Datenstrukturen und zur Abfrage der darauf basierenden Datenstände in relationalen Datenbanken. Durch die große Anzahl an Einsatzmöglichkeiten in Webseiten, Firmendatenbanken und in Desktopanwendungen gerieten SQL-Datenbanken mit der Zeit immer mehr in das Visier der Hacker. In den letzten Jahren stieg die Verbreitung von SQL-Datenbanken exponentiell an, da Smartphone-Apps SQLite-Datenbanken zum speichern interner Daten und Konfigurationen nutzen. Durch SQL Injection ist es Angreifern möglich unberechtigt auf den Inhalt von Datenbanken zuzugreifen oder diese zu manipulieren.

Diese Arbeit liefert einen Überblick über die Gefahren durch SQL Injections und beschreibt eine Vielzahl von Schutzmechanismen, welche dazu dienen Angriffe dieser Art zu verhindern.

Schlüsselworte

Angriffsmöglichkeiten, Datenbank, Gegenmaßnahmen, Hacking, SQL Injection, Schutzmechanismen, Sicherheit

1. EINLEITUNG

Das Internet besteht aktuell aus über 250 Millionen Webseiten und jährlich kommen mehr als 20 Millionen neue hinzu. [24] Angefangen bei Webshops über Onlinebanking und Computerspiele, werden im Internet fast überall auf SQL basierende Datenbanksysteme zur Verwaltung der anfallenden Daten verwendet. Bei beinahe allen modernen Smartphones läuft eine SQLite Datenbank im Hintergrund mit, um installierte Apps, SMS, Kontakte uvm. zu verwalten. Jede strukturierte Firma nutzt eine Datenbank um Mitarbeiter, Kunden, Produkte und viele weitere Dinge zu verwalten.

Aus der Sicht eines Angreifers bieten sich nahezu unbegrenzt viele Möglichkeiten und Ziele. Wie aber kann man sich gegen solche Angriffe schützen? Wie kann man Datenbanken nutzen ohne fürchten zu müssen, das Unbefugte auf diese zugreifen können? In einer kurzen Einführung im ersten Kapitel werden die Grundlagen von SQL Injections und SQL dargelegt. Das zweite Kapitel behandelt die verschiedenen Möglichkeiten, welche einem Angreifer durch eine SQL Injection zur Verfügung stehen und im letzten Kapitel werden Schutzmechanismen zur Vermeidung von SQL Injections behandelt.

2. EINFÜHRUNG

2.1 Was ist SQL?

SQL steht für „Structured Query Language“ und basiert auf der Sprache SEQUEL, die in den 1970ern von Edgar F. Codd bei IBM entwickelt wurde. [28] Bei SQL handelt es sich um eine Datenbanksprache zur Kommunikation zwischen einer Anwendung und einer relationalen Datenbank. 1986 wurde SQL unter dem Namen SQL1 erstmals durch das „American National Standards Institute“ (ANSI) als Standard deklariert und ein Jahr später ebenfalls durch die „Internationale Organisation für Normung“ (ISO). Inzwischen ist die achte Version von SQL als Standard anerkannt worden.

Jahr [Quelle]	Name	Organisation
1986 [28]	SQL1	ANSI
1987 [13]	SQL1	ISO
1992 [14]	SQL2	ISO
1999 [15]	SQL:1999	ISO
2003 [16]	SQL:2003	ISO
2006 [28]	SQL:2006	ISO
2008 [17]	SQL:2008	ISO
2011 [18]	SQL:2011	ISO

Tabelle 1: Übersicht aller SQL-Standards

SQL basiert auf Statements, die an die englische Umgangssprache angelehnt sind und welche im allgemeinen in drei Kategorien einzuordnen sind:

1. **Data Control Language (DCL)**: Unter DCL versteht man den Bereich einer Datenbanksprache welcher für das Hinzufügen, Entfernen oder Verändern von Rechten zuständig ist. [2]
2. **Data Manipulation Language (DML)**: Dieser Bereich umfasst alle Statements, die Lese- und Schreibzugriffe auf Einträge in Datenbanken ausführen. Das bedeutet Statements, die Daten verändern, hinzufügen oder löschen. [12]
3. **Data Definition Language (DDL)**: DDL beinhaltet alle Befehle, die Veränderungen an der Datenbankstruktur auslösen. Darin enthalten sind Statements, die neue Tabellen hinzufügen oder löschen, sowie Statements, die vorhandene Tabellen verändern, indem beispielsweise neue Spalten hinzugefügt werden. [11]

Um diverse Funktionen in SQL ausführen zu können, werden spezielle Metazeichen verwendet. Wenn der SQL-Interpreter

auf ein solches Zeichen stößt, wird das Selbige nicht als String behandelt, sondern löst eine Funktion auf dem Interpreter aus. Für eine SQL Injection sind Metazeichen von essentieller Bedeutung, wobei dies die wichtigsten Metazeichen sind:

*	Alle Spalten einer Tabelle werden ausgewählt
%	Wichtig für Substringsuche in Texten
'	Zeigt den Anfang und das Ende eines Parameters an
- -	Markiert die restliche Zeile als Kommentar
#	Markiert alle folgenden Zeichen als Kommentar

Tabelle 2: Metazeichen in SQL

2.2 Was ist eine SQL Injection und wie läuft sie ab?

Bei einer SQL Injection verschafft sich ein Angreifer unerlaubt Zugriff auf die Datenbank einer Anwendung. Dabei nutzt er eine Sicherheitslücke, die der Entwickler offen gelassen hat. Der Begriff Sicherheitslücke umfasst hierbei alle Möglichkeiten, bei denen ein Nutzer Daten ändern oder hinzufügen kann, die ohne - oder mit mangelhafter - Überprüfung in ein SQL-Statement eingefügt werden. [27] Den größten Teil dieser Lücken machen Formulare auf Internetseiten und GET-Parameter aus. Bei GET-Paramtern handelt es sich um Zeichenketten, die an die URL einer Webseite angehängt werden.

```
http://www.example.com/?name=foo&email=bar
```

Ähnlich wie bei Formularen sind diese Daten direkt vom Nutzer veränderbar. Dies ermöglicht dem Angreifer eigene Datenbankbefehle an die Datenbank zu senden und die dort liegenden Daten zu manipulieren oder auszulesen. Das vom Entwickler genutzte SQL-Statement, welches der Angreifer zum Übermitteln seiner eigenen Befehle nutzt, wird im folgenden als Trägerstatement bezeichnet.

Um eine gültige SQL-Abfrage einfügen zu können, muss der Angreifer ausreichend Kenntnisse über die Benennung der Datenbanktabellen und -spalten haben. Diese Kenntnisse können beispielsweise durch „Trial & Error“ erlangt werden. Alternativ sind viele Anwendungen und Server so konfiguriert, dass sie aussagekräftige Fehlermeldungen zurückgeben, falls die Abfrage ungültige Spalten- oder Tabellennamen enthält. Allgemein beginnt man eine SQL Injection, indem ein Statement an den entsprechenden Stellen eines Formulars oder GET-Parameters platziert wird. Sollte das Statement bis zur Datenbank vordringen können, resultiert dies in einer SQL-Fehlermeldung. [27] Wenn dieser Fall eintritt weiß der Angreifer, dass an dieser Stelle die Nutzereingabe nicht richtig validiert wird. Anhand der ausgegebenen Fehlermeldungen lassen sich viele Rückschlüsse auf die Datenbankstruktur erlangen, welche dem Angreifer helfen, die richtigen Benennungen zu finden.

Angenommen, es handelt sich um den Anmeldebereich einer Onlinebanking-Seite mit einem Microsoft SQL Server. [29] Der Angreifer hat drei Felder: Loginname, Passwort und PIN. Er tippt in das PIN-Feld den folgenden SQL-Befehl ein:

```
1 convert(int,(SELECT top 1
2  name FROM sysobjects WHERE xtype='u'))
```

Auflistung 1: Angriff beim Onlinebanking

Durch die Anwendung wird das fertige SQL-Statement zusammengesetzt und sieht so aus:

```
1 SELECT accounts FROM users WHERE login=''
2 AND pass=''
3 AND pin='convert(int,(SELECT top 1 name
4          FROM sysobjects WHERE xtype='u'))'
```

Auflistung 2: Onlinebanking Statement

Bei Microsoft SQL-Servern heißt die Metatabelle sysobjects und beinhaltet Informationen über alle anderen Tabellen, Spalten, usw.. Die Bedingung „xtype='u'“ sorgt dafür, dass die erste vom Benutzer angelegte Tabelle ausgelesen wird. In diesem Beispiel kann der Server mit folgender Antwort reagieren:

```
Microsoft OLE DB Provider for SQL Server (0x80040E07)
Error converting nvarchar value 'CreditCards'
to a column of data type int.
```

Aus der Fehlermeldung lässt sich leicht ablesen, dass es eine Tabelle mit dem Namen 'CreditCards' gibt. Durch mehrfaches Ausführen des Queries mit verschiedenen Zahlen, können beliebig viele Tabellennamen ausgelesen werden.

Erstmals erwähnt wurde ein solcher Angriff im Dezember 1998. In der 54. Ausgabe der Zeitschrift „Phrack“ [25] schreibt ein Mitglied der Community unter dem Pseudonym „rain forest puppy“ (rfp) unter anderem über Angriffe, welche Sicherheitslücken in Internet Datencentern und ASP Anwendungen ausnutzen, um auf eine dahinter stehende Datenbank (SQL Server 6.5) zuzugreifen. ASP ist ein Format zur Entwicklung von Webseiten, welches von Microsoft entwickelt wurde. Zu diesem Zeitpunkt existierte der Fachbegriff der SQL Injection noch nicht. Der dort beschriebene Vorgang entspricht allerdings sehr genau dem, was man heute unter einer SQL Injection versteht. [20] Im Verlauf der folgenden Monate veröffentlichte „rfp“ weitere Artikel [21] die zeigten, wie er aufgrund unachtsamer Entwickler Zugriff auf Datenbanken erlangen konnte.

3. ANGRIFFSMÖGLICHKEITEN

3.1 Wo kann ein Angriff ansetzen?

Der Hauptangriffspunkt für eine SQL Injection ist die Schnittstelle zwischen Endanwender und der Datenbank. Das kann beispielsweise eine Webseite, eine Desktopanwendung oder eine App auf dem Smartphone sein.

Wie in Abbildung 1 zu sehen ist, kann der Endanwender nicht direkt auf die Datenbank zugreifen. Der Angreifer kann nur mit der Anwendung kommunizieren, welche dann als Mittelsmann zwischen ihm und der Datenbank fungiert. Um eine erfolgreiche SQL-Injection umzusetzen, muss der Angreifer eine Lücke finden, mit welcher er die Datenbankanfragen der Anwendung verändern kann.

Abbildung 1: Kommunikationswege zwischen Datenbank und Endanwender

Je nachdem was für eine Datenbank hinter der Anwendung steht, wie professionell das Produkt entwickelt wurde und wie sehr der Entwickler auf die Sicherheit geachtet hat, kann eine SQL Injection sehr einfach oder schwer sein. Hierbei ist vor allem die verwendete Datenbanksoftware und deren Version ausschlaggebend, da auch die Hersteller von Datenbanksoftware immer mehr auf Sicherheit bedacht sind und so viele Angriffsmöglichkeiten, die in früheren Versionen erfolgreich waren, von vornherein unterbinden. Aber auch die Möglichkeit von OpenSource-Software erleichtert es den Angreifern Zugriff auf die Datenbank zu erlangen. Wenn Quellcode für alle zugänglich ist, haben es Angreifer viel leichter eventuelle Sicherheitslücken zu finden. [5] [10]

Wenn der Quellcode nicht zugänglich ist, gibt es andere Wege die Lücken in Anwendungen zu finden. So stehen zahlreiche Hilfsmittel, die große Teile der aufwendigen Suche nach Lücken automatisch abdecken kostenlos zur Verfügung. Unter anderem die Programme SQL-PowerInjector [19] und sqlmap [1] bieten einen großen Umfang an Funktionen, um die eigene Anwendung auf Sicherheitslücken zu testen. Wie so oft lassen sich aber eben diese Programme auch für den Angriff auf andere Webseiten verwenden.

3.2 Beispiel einer SQL Injection anhand eines Loginformulars

Die am weitesten verbreitete Art von Formularen trifft man bei der Anmeldung auf Webseiten oder bei Anwendungen an. Es besteht in der einfachsten Form aus zwei Feldern und einem Button. Während für einen Nutzer die Formulare fast immer identisch aussehen, macht die Implementierung den Unterschied. Gerade in einfachen Anwendungen läuft der Login meistens wie folgt ab:

1. Der Benutzer gibt seine Daten in das Formular ein und schickt diese ab
2. Die Daten werden in ein SQL-Statement eingefügt und an die Datenbank weitergeleitet
3. Die Datenbank schickt die angeforderten Nutzerdaten, bei denen Benutzernamen und Passwort mit den Formulareingaben übereinstimmen an die Anwendung zurück

Diese drei Schritte werden in jedem Loginvorgang in allen Anwendungen vorkommen, jedoch sollte ein sicherer Login weit mehr als nur diese drei Schritte umfassen. Angenommen, das SQL-Statement sieht wie folgt aus:

```
1 SELECT *
2 FROM users
3 WHERE name='$name'
4   AND password='$password';--
```

Auflistung 3: Beispielstatement eines Logins

Bei dem SQL-Query aus Auflistung 3 ist es möglich, sich ohne Kenntnis des zugehörigen Passworts mit jedem beliebigen registrierten Benutzer anzumelden, falls die Nutzereingabe nicht überprüft wird. Wenn die Anwendung eine Mitgliederliste anzeigt, vereinfacht das die Suche nach einem Nutzernamen erheblich, andernfalls kann der Angreifer solange Namen raten, bis er einen in der Datenbank vorhandenen Namen trifft. Beispielsweise hat der Angreifer herausgefunden, dass es einen Nutzer mit dem Namen „John“ gibt. Alles was der Angreifer für den Login noch machen muss ist folgendes als Benutzername einzugeben:

```
1 John';--
```

Auflistung 4: SQL-Injection über den Benutzernamen

Wenn die so eingegebenen Daten ohne Validierungstests in das Statement eingefügt werden, wird durch die Abfolge von Metazeichen in der Nutzereingabe der Teil des ursprünglichen Queries auskommentiert, der für das Passwort verantwortlich war. Das heißt, es wird lediglich überprüft ob ein Nutzer mit dem Namen „John“ in der Datenbank existiert. Das Passwort ist für den Anmeldevorgang nicht mehr relevant.

```
1 SELECT *
2 FROM users
3 WHERE name='John';--
```

Auflistung 5: Statement ohne Passwortabfrage

Der Login ohne Passwort ist aber auch möglich, wenn man die Nutzung von „- -“ in den Feldern verbietet. In diesem Fall kann man sich unter anderem die Vorrangregeln der booleschen Operatoren zunutze machen. Da AND Vorrang vor OR hat, kann der Angreifer den Login umgehen, indem er ein leeres Passwort eingibt und den Benutzernamen so wählt:

```
1 John' OR 'a'='b
```

Auflistung 6: SQL-Injection über den Benutzernamen

In das Trägerstatement eingesetzt ergibt sich folgender Query:

```
1 SELECT *
2 FROM users
3 WHERE name='John' OR 'a'='b'
4   AND password='';--
```

Auflistung 7: Zusammengesetzter Query

Da der Buchstabe 'a' nicht der selbe wie 'b' ist, ergibt sich „FALSE AND password=' '“. Im Normalfall gibt es auch keine leeren Passwörter, was dazu führt, dass der gesamte Ausdruck nach dem OR zu FALSE wird. Dadurch bleibt als auszuwertender Ausdruck nur noch „WHERE name='John'“ übrig.

Wenn ein Angreifer erst einmal eine Sicherheitslücke gefunden hat, welche ihm Zugriff auf die Datenbank gewährt, sind seine Möglichkeiten nahezu unbegrenzt. Für die folgenden Beispiele wird immer das Trägerstatement aus Auflistung 3 verwendet.

3.3 Daten hinzufügen

Der Angreifer kann beliebige Daten in die Datenbank einfügen. Alleine hierfür gibt es schon zahlreiche Anwendungsfälle:

- Man kann einen neuen Benutzer in die entsprechende Tabelle einfügen und sich dadurch Zugang zu gesperrten Bereichen verschaffen.
- Ebenfalls ist es möglich eigene Einträge in fremde Webseiten einbetten. Wenn durch die eingefügten Einträge verdeutlicht wird, wie unsicher die Anwendung ist, kann ein solches Vorgehen zu Rufschädigung führen. Diese Methode wird vor allem bei großen Unternehmen angewandt, bei denen man als Kunde davon ausgeht, dass die eigenen Daten dort sicher aufgehoben wären. Unter anderem die Gema [7], Nokia [8] und Sony [6] [9] sind Opfer solcher Attacken geworden. Alternativ können die eingefügten Artikel als Werbung für eigene Produkte dienen oder andere Vorteile für den Angreifer bieten.
- Durch die Möglichkeit Quellcode in fremde Webseiten einzubauen, ist auch die Nutzung für Cross-Site Scripting (XSS) gegeben. Um dauerhaft Daten abzugreifen oder Popups zu öffnen, reicht es häufig bereits wenige Zeilen Javascript in fremden Anwendungen zu platzieren.

Eigene Daten können in die fremde Datenbank eingetragen werden, indem man im Trägerstatement das Passwort folgendermaßen setzt:

```
1 abc';
2 INSERT INTO tabelle
3     (spalte1, spalte2, spalte3) VALUES
4     ('foo', 'bar', 'foo@bar.com')
```

Auflistung 8: Passwort mit INSERT-Statement

Die Benutzerabfrage mit Name und Passwort wird höchstwarscheinlich fehlschlagen. Dem Angreifer ist dies allerdings gleichgültig. Er möchte lediglich erreichen, dass die zweite - von ihm angefügte - Anfrage unverändert an die Datenbank gesendet und ausgeführt wird.

3.4 Daten verändern

Neben dem Hinzufügen von Einträgen kann ein Angreifer auch vorhandene Einträge verändern. So ist es beispielsweise möglich den eigenen Benutzer als Administrator zu markieren, wenn die Datenbank den Unterschied zwischen normalen Nutzern und Administratoren durch ein Flag in der Datenbank behandelt. Der Angreifer könnte sich in einer Datenbank, in der Passwörter unverschlüsselt gespeichert werden durch Ändern des Passworts Zugang zu jedem Nutzerprofil verschaffen.

Falls Passwörter verschlüsselt gespeichert werden, ist es dennoch möglich sich Zugang zu jedem beliebigen Profil zu verschaffen. Es reicht meistens aus, die Emailadresse eines Nutzers in der Datenbank durch die Eigene zu ersetzen, um dann die „Passwort vergessen“-Funktion zu nutzen, wodurch einem ein neues Passwort zugeschickt wird.

Um die Emailadresse eines beliebigen Nutzers zu verändern, ist es aussreichend als Passwort folgendes einzugeben:

```
1 abc';
2 UPDATE users
3     SET email='foo@bar.com'
4     WHERE name='John'
```

Auflistung 9: UPDATE-Statement im Passwortfeld

3.5 Daten löschen

Wenn ein Nutzer auf einer „schwarzen Liste“ gespeichert wurde, da er beispielsweise unangenehm aufgefallen ist, so könnte er über eine SQL Injection den entsprechenden Eintrag aus der Datenbank wieder entfernen.

Um einen bestimmten Eintrag aus der Datenbank zu löschen, kann der Angreifer das Passwort beispielsweise so setzen:

```
1 abc';
2 DELETE FROM tabelle WHERE spalte='wert';--
```

Auflistung 10: Passwort mit DELETE-Statement

Es ist aber auch möglich die gesamten Datenbankstruktur zu verändern oder ganze Tabellen zu löschen wie im Beispiel von „Little Bobby Tables“ [22]:

```
1 Robert'); DROP TABLE Students;--
```

Auflistung 11: Little Bobby Tables

Neben dem Löschen von Einträgen kann ein Angreifer in den schlimmsten Fällen auch direkt auf den Server der Datenbank zugreifen. Über den folgenden SQL-Befehl wäre es möglich den Datenbankserver herunterfahren, sofern der aktuelle Datenbanknutzer root-Rechte besitzt:

```
1 GO EXEC cmdshell('shutdown /s'); --
```

Auflistung 12: Shell-Befehl Shutdown bei Microsoft SQL-Servern

3.6 Daten sammeln

Alle bisher genannten Methoden würden früher oder später einem aufmerksamen Beobachter auffallen, da sichtbar Änderungen an der Datenbank getätigt wurden. Der Angreifer könnte aber auch - ohne Änderungen in der Datenbank vorzunehmen - Daten auslesen. Dies ist die unauffälligste Möglichkeit für eine SQL Injection. Es ist nur schwer möglich solche Angriffe aufzudecken, bevor der Angreifer den Besitzer der Datenbank mit den gesammelten Informationen konfrontiert.

Das Sammeln von Daten wird über den UNION-Befehl der SQL-Syntax ausgeführt und ermöglicht es, mehrere SELECT-Abfragen zu einer Ausgabe zu kombinieren. Dies ist nötig, da eine Anwendung die beispielsweise Bücher auflistet, welche nach einem Suchkriterium aus der Datenbank ausgelesen werden, eine Fehlermeldung zurückgeben würde, wenn die Spalten der Antwort auf einmal „Passwort“ und „Emailadresse“ anstatt „Buchtitel“ und „Erscheinungsjahr“ heißen würden. Bei der Nutzung des UNION-Befehls ist zu beachten, dass alle SELECT-Anfragen die selbe Anzahl an Spalten haben müssen. Wenn das Trägerstatement vier Spalten aus einer Tabelle abfrägt, muss das vom Angreifer angehängte Statement ebenfalls vier Spalten abfragen. Falls der Angreifer keine vier Spalten der neuen Tabelle kennt oder es nur weniger als vier Spalten gibt, müssen die fehlenden Stellen mit „null“ aufgefüllt werden.

```
1 SELECT title, autor, price, year
2 FROM books
3 WHERE title='$title';--
```

Auflistung 13: Trägerstatement einer Buchsuche

In dem Beispiel aus Listing 13, welches eigentlich „Buchtitel“, „Autor“, „Preis“ und „Erscheinungsjahr“ durch den Query abfrägt, setzt der Angreifer den Buchtitel z.B. folgendermaßen:

```
1 UNION SELECT name, passwort, email, null
2       FROM users;--
```

Auflistung 14: UNION Beispiel

Die Antwort der Datenbank würde dann zuerst alle Buchinformationen sammeln, die auf die übergebenen Kriterien zutreffen und danach die gesamte Benutzertabelle „users“ auslesen. Dabei werden alle „Namen“ in die Spalte „Buchtitel“, alle „Passwörter“ in die Spalte „Autor“, und das alle „Emailadressen“ in die „Preise“-Spalte geschrieben. Die Spalte „Erscheinungsjahr“ wäre für die Benutzer leer.

Buchtitel	Autor	Preis	Jahr
Buch #1	Autor #1	25€	2011
Buch #2	Autor #2	30€	2011
⋮	⋮	⋮	⋮
Benutzer #1	Passwort #1	Email #1	
Benutzer #2	Passwort #2	Email #2	
⋮	⋮	⋮	⋮

Tabelle 3: Ausgabe einer UNION-Abfrage

4. URSACHEN & LÖSUNGEN

All diese Angriffe sind nur möglich, weil der Entwickler versäumt hat, an den entsprechenden Stellen die Eingaben der Nutzer ausreichend überprüfen zu lassen, bevor diese an die Datenbank gesendet werden. Dabei gibt es eine ganze Reihe von Möglichkeiten solche Angriffe zu erschweren, beziehungsweise zu verhindern. Im Folgenden werden die wichtigsten Methoden vorgestellt.

4.1 Eingaben validieren

Da SQL Injections hauptsächlich möglich sind, weil Nutzer eigene SQL-Statements in Formularfeldern eingeben und abschicken können, ist es naheliegend eben diese Felder auf deren Inhalt zu überprüfen. So ist es möglich über „Reguläre Ausdrücke“ nur spezielle Zeichen in Feldern zu erlauben. Beispielweise wäre es folgerichtig in Feldern, in denen man das Alter oder eine Postleitzahl eingeben muss ausschließlich Zahlen zu zulassen. Ebenso sind in einem Feld für ein Datum Buchstaben fehl am Platz. Ein Feld für das Alter sollte außerdem beachten, ob ein vierstelliges oder noch höheres Alter Sinn macht. Folglich reicht es hier, das Feld auf maximal drei Zeichen zu beschränken. In einem Feld, welches nur für drei Zeichen zulässig ist, wird es sehr schwierig ein SQL-Statement unterzubringen. Wenn die Daten an den Server übermittelt wurden, können weiterhin alle Werte über Funktionen auf Korrektheit überprüft werden. Vom korrektem Format einer Emailadresse bis hin zu einem gültigen Datum lassen sich für alle vom Nutzer übermittelten Werte Möglichkeiten finden, diese zu validieren.

Eine einfache Variante wäre, für jede Art von Formularfeld eine Funktion zu schreiben. Diese Funktion nimmt den Wert aus dem Formular und überprüft ihn auf Korrektheit. Hierbei wird zwischen Validatoren und Filtern unterschieden. Während Validatoren lediglich „true“ bzw. „false“ zurückgeben und somit die Formularauswertung abbrechen können, löschen Filter alle nicht erlaubten Zeichen aus der Nutzereingabe und können so die Nutzereingaben verändern. Filter sind allerdings nicht für alle Formularfelder zu empfehlen, da beispielsweise von Filtern veränderte Passwörter dem Nutzer später die Anmeldung verweigern könnten.

4.2 Escapen

Da jedes SQL-Statement mit einem Schlüsselwort anfängt, klingt es verlockend eben diese zu suchen und unschädlich zu machen. Allerdings reicht es nicht, alle Wörter wie SELECT, UPDATE, DELETE, INSERT, ... zu entfernen. Das wäre unproduktiv, wenn es sich um eine Anwendung handelt, die über SQL-Statements informiert, diese Wörter aber nicht speichern kann. Daher beschränkt man sich im allgemeinen darauf die Metazeichen zu suchen und zu beseitigen. Diese Zeichen zu löschen würde wieder zu Veränderungen des urspünglichen Sinns führen. Wenn beispielsweise ein Nutzer in ein Profilfeld schreibt: „Mir geht’s gut. Ich bin zu 100% fit.“ Durch das Entfernen der Metazeichen „’“ und „%“ würde der Sinn der Nutzereingabe verändert werden. Um die Zeichen trotzdem von ihrem SQL-Sonderstatus zu befreien, gibt es die Möglichkeit die Zeichen zu „escapen“.

Escapen ist eine der wirkungsvollsten Maßnahmen gegen SQL Injections und verhindert beinahe alle aktuell bekannten Angriffsmöglichkeiten. Bei diesem Vorgang werden alle Metazeichen in SQL durch einen Backslash markiert, wenn

sie in einer Nutzereingabe vorkommen. Dies sorgt dafür, dass der SQL-Interpreter die Zeichen wie einen normalen Text behandelt und nicht als Metazeichen einstuft. Aus: „Mir geht's gut. Ich bin zu 100% fit.“, wird durch Escapen: „Mit geht\'s gut. Ich bin zu 100\% fit.“ Wenn dieser String - ohne vorher escaped zu werden - in ein Statement eingebaut wird, so liefert der SQL-Interpreter Fehlermeldungen aufgrund einer falschen Syntax. Durch das Apostroph wird die Variable beendet, und alle nachfolgenden Zeichen müssten aus valider SQL-Syntax bestehen. Da dies hier nicht der Fall ist wird die Anfrage mit einem Fehler beendet. Wenn nur das Apostroph markiert wird, würde das %-Zeichen den Fehler generieren. Da Prozentzeichen nur in LIKE-Abfragen erlaubt sind, muss ebenfalls dieses hier markiert werden, damit die Anfrage erfolgreich beendet werden kann. Die meisten aktuellen Programmiersprachen bieten bereits Funktionen an, um Metazeichen zu escapen:

```
1 mysql_real_escape_string($variable);
```

Auflistung 15: PHP-Funktion um Metazeichen in einer Variable zu escapen

4.3 Prepared Statements zur Trennung von Abfrage und Daten

Eine weitere Möglichkeit, die beinahe alle SQL Injection Angriffe verhindert, ist die Nutzung von Prepared Statements. Die auch als „Stored Procedures“ bekannte Funktion besteht aus mindestens zwei Teilen. Im ersten Teil wird das SQL-Statement in einer Variable gespeichert.

```
1 $std = "SELECT * FROM table
2         WHERE param ='__param__'"
```

Auflistung 16: Prepared Statement

Die Stellen, die später durch die Nutzerdaten gefüllt werden, sind mit Platzhaltern versehen. Je nach Programmiersprache können sie folgende Formen annehmen: „:param:“, „__param__“ oder „@param“. Aufgrund der strikten Trennung von Statement und Nutzerdaten ist es möglich, die Daten ausreichend zu überprüfen bevor diese überhaupt mit dem SQL-Query in Berührung kommen. Wenn die Daten alle vom Entwickler eingebauten Validierungstests bestanden haben, werden sie im zweiten Teil in die Stored Procedure eingefügt.

```
1 $std->addParam('__param__', $param);
```

Auflistung 17: Parameter zu einem Prepared Statement hinzufügen

Ähnlich wie für das Escapen gibt es auch hier zahlreiche Schnittstellen, welche Prepared Statements unterstützen. Eine Übersicht zu den Sprachen C++, Java, Perl und PHP zeigt Tabelle 4.

Sprache	Schnittstelle [Quelle]
C++	MySQL Connector [3]
Java	PreparedStatement-Klasse [4]
Perl	DBI-Interface [26]
PHP	Stored Procedures [23]

Tabelle 4: Verbreitete Programmiersprachen und ihre Schnittstellen zu Prepared Statements

4.4 Bewertung und Auswirkungen der Methoden

Jede Methode für sich kann SQL-Injections komplett verhindern, wenn sie korrekt eingesetzt wird. Den Unterschied macht hierbei der damit verbundene Aufwand. Bei der „Eingaben validieren“-Methode (Kapitel 4.1) ist der Aufwand vergleichsweise groß, da alle Eventualitäten beachtet werden müssen. Es muss für jeden Formulartyp überprüft werden, ob die Daten korrekt sind, ob die Daten keine unerlaubten Zeichen enthalten, usw.. Wenn eine mögliche Variante oder ein Formularfeld vergessen wurde, bringen alle anderen validierten Felder nichts. Das „Escapen“ (Kapitel 4.2) ist hier schon viel bequemer. Es muss lediglich jeder Datensatz einmal durch eine Funktion geschickt werden, die alle Sonderzeichen von SQL maskiert und somit unbrauchbar für Angriffe macht. Aktuell ist kein Angriff bekannt, der nicht durch „Escapen“ verhindert werden kann. In den meisten Fällen ist letztendlich eine Kombination aus den beiden genannten Methoden nötig, um alle erwünschten Funktionen zu garantieren. Somit werden alle Felder auf korrekten Inhalt geprüft und dennoch ist kein Angriff möglich. Eine Kombination der beiden Möglichkeiten sind die „Prepared-Statements“ (Kapitel 4.3) die vor allem durch Objektorientierte Programmierung in den letzten Jahren immer mehr an Bedeutung gewinnen. Hier ist bereits vorgesehen, dass die Daten erst validiert werden und dann im zweiten Schritt in ein vorgefertigtes Statement eingebunden werden.

4.5 Rechteverwaltung

Wer Wert auf eine noch höhere Sicherheitsstufe legt, kann die verschiedenen Statements welche die Anwendung an die Datenbank verschickt über unterschiedliche Datenbanknutzer abwickeln lassen. So können beispielsweise alle Funktionen, die ausschließlich Leserechte benötigen (z.B. Loginabfrage, Artikel auslesen, ...) über einen Datenbanknutzer ausgeführt werden, welcher nur Leserechte besitzt. Dadurch ist es für einen Angreifer nicht mehr möglich über diese Funktionen neue Daten hinzuzufügen oder vorhandene Einträge zu bearbeiten. Allerdings kann auch durch die Nutzung mehrerer Datenbanknutzer nicht verhindert werden, dass ein Angreifer heimlich Daten sammelt. Deutlich erschweren könnte man dies, indem die einzelnen Datenbanknutzer nur Zugang auf einige Tabellen haben. Wichtige Tabellen wie z.B die Benutzertabelle oder eine mit Kreditkarteninformationen wären von anderen Nutzern nicht abrufbar. Solch ein Konzept umzusetzen ist - alleine aus Übersichtsgründen - in den meisten Fällen aber komplizierter zu implementieren als die anderen bisher genannten Methoden.

5. ZUSAMMENFASSUNG

Die Gefahr einer SQL Injection spielt eine große Rolle in allen Anwendungen, die auf relationalen Datenbanken basieren. Wenn ein Angreifer erst einmal die Möglichkeit bekommt, durch eine Sicherheitslücke wie ein Formular oder GET-Parameter eigenen SQL-Code in die Datenbank einzuschleusen, sind seinen Möglichkeiten so gut wie keine Grenzen mehr gesetzt. Der Angreifer kann Daten hinzufügen, neue Tabellen erstellen oder bereits Vorhandene löschen.

Er kann auch bestehende Daten verändern oder unbemerkt sammeln.

Um sich vor solchen Angriffen zu schützen, ist es zwingend notwendig, die vom Nutzer eingegebenen Daten zu überprüfen, bevor diese von der Anwendung zur Datenbank weitergeleitet werden. Zur Validierung der Daten stehen zahlreiche Hilfsmittel, wie zum Beispiel „Prepared Statements“ oder das „Escapen“ von Metazeichen zur Verfügung. Wenn eine Anwendung diese beiden Mechanismen konsequent einsetzt ist es für einen Angreifer beinahe unmöglich, Zugriff auf die Datenbank zu erlangen. Zusätzlich können die Eingaben des Nutzers noch stärker beschränkt werden, indem man feste Vorgaben gibt, welche Zeichen - oder Zeichenlängen - in einzelnen Feldern erlaubt sind.

6. LITERATUR

[1] Bernardo Damele A. G. SQLmap. http://sqlmap.sourceforge.net/, 2011. [Aufgerufen am 17.12.2011].

[2] T. Database. SQL Data Control Language. www.info.teradata.com/edownload.cfm?itemid=091270003, 2009. [Aufgerufen am 20.12.2011].

[3] Dev.mysql.com. C++ Prepared Statement. http://dev.mysql.com/doc/refman/5.1/en/connector-cpp-examples-prepared-statements.html, 2011. [Aufgerufen am 18.12.2011].

[4] Docs.oracle.com. Java Prepared Statement. http://docs.oracle.com/javase/6/docs/api/java/sql/PreparedStatement.html, 2011. [Aufgerufen am 18.12.2011].

[5] Heise.de. Typo3-Update schließt zahlreiche kritische Lücken. http://heise.de/-837788, Oct. 2009. [Aufgerufen am 17.12.2011].

[6] Heise.de. Angriffe auf Sony gehen weiter. http://heise.de/-1250054, May 2011. [Aufgerufen am 17.12.2011].

[7] Heise.de. Gema offenbar gleich mehrfach gehackt. http://heise.de/-1328737, Aug. 2011. [Aufgerufen am 17.12.2011].

[8] Heise.de. Hacker späht Nutzerdaten in Nokias Entwicklerforum aus. http://heise.de/-1332704, Aug. 2011. [Aufgerufen am 17.12.2011].

[9] Heise.de. Hacktivisten knacken Datenbank von Sony Pictures. http://heise.de/-1254485, June 2011. [Aufgerufen am 17.12.2011].

[10] Heise.de. Kritische Lücke in verbreitetem Shopsystem xt:commerce. http://heise.de/-1193394, Feb. 2011. [Aufgerufen am 17.12.2011].

[11] IBM. Data Definition Language. http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=%2Fsqlp%2Frbafysqltech.htm, 2011. [Aufgerufen am 20.12.2011].

[12] IBM. Data Manipulation Language. http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=%2Fsqlp%2Frbafydml.htm, 2011. [Aufgerufen am 20.12.2011].

[13] Iso.org. ISO 9075:1987 - Information processing systems – Database language – SQL. http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16661, 1987. [Aufgerufen am 20.12.2011].

[14] Iso.org. ISO/IEC 9075:1992 - Information technology – Database languages – SQL. http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=16663, 1992. [Aufgerufen am 20.12.2011].

[15] Iso.org. ISO/IEC 9075-1:1999 - Information technology – Database languages – SQL. http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=26196, 1999. [Aufgerufen am 20.12.2011].

[16] Iso.org. ISO/IEC 9075-1:2003 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=34132, 2003. [Aufgerufen am 20.12.2011].

[17] Iso.org. ISO/IEC 9075-1:2008 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=45498, 2008. [Aufgerufen am 20.12.2011].

[18] Iso.org. ISO/IEC 9075-1:2011 - Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=53681, 2011. [Aufgerufen am 20.12.2011].

[19] F. Larouche. SQL-PowerInjector. http://www.sqlpowerinjector.com/, 2006. [Aufgerufen am 17.12.2011].

[20] D. Litchfield. Data-mining with SQL Injection and Inference. Technical report, NGSSoftware, 2005.

[21] R. F. P. Matthew Astley. NT ODBC Remote Compromise. http://www.securiteam.com/windowsntfocus/2AVQ5QAQKM.html, 1999. [Aufgerufen am 20.12.2011].

[22] R. Munroe. Exploits of a Mom. http://xkcd.com/327/, 2008. [Aufgerufen am 19.12.2011].

[23] Php.net. PHP Stored Procedures. http://www.php.net/manual/de/pdo.prepared-statements.php, 2011. [Aufgerufen am 18.12.2011].

[24] Pingdom. Internet 2010 in numbers. http://royal.pingdom.com/2011/01/12/internet-2010-in-numbers/, Jan. 2011. [Aufgerufen am 17.12.2011].

[25] R. F. Puppy. Phrack.org. http://www.phrack.org/, 1998. [Aufgerufen am 19.12.2011].

[26] Search.cpan.org. Perl Database independent interface. http://search.cpan.org/$\sim$timb/DBI-1.616/DBI.pm, 2011. [Aufgerufen am 18.12.2011].

[27] K. Spett. Blind SQL Injection. Technical report, SPI Dynamics, 2003. [Aufgerufen am 19.12.2011].

[28] Wikipedia.org. Wikipedia SQL. http://en.wikipedia.org/wiki/SQL, 2011. [Aufgerufen am 20.12.2011].

[29] A. O. William G.J. Halfond, Jeremy Viegas. A Classification of SQL Injection Attacks and Countermeasures. Technical report, Georgia Institute of Technology, 2006.