Handout «SQL-Injection»

SQL-Injection ist eine weit verbreitete Angriffsmethode um an Daten zu gelangen, oder um Daten zu zerstören. Es ist einfach zu erlernen und wird darum so oft angewendet. Es gibt aber auch genauso einfache Methoden, um diese Angriffe abzuwehren. In diesem Handout, erstellt von Sasha Keller, als Zusatz zur Präsentation vom 21.11.2018, wird «SQL-Injection» beschrieben und wie man sich schützen kann vor solchen Angriffen. Zusätzlich sind in diesem Dokument noch Code-Beispiele aufgeführt, wie solch ein Angriff aussieht und wie man sich mit paar Zeilen Code davor schützen kann.

Was ist eine «SQL-Injection»

Wie oben erwähnt, ist «SQL-Injection» sehr beliebt. Es ist eine einfache Angriffsmethode, um an Daten zu gelangen. Aber wie funktioniert eine SQL-Injection?

Wie der Name schon sagt, versucht man bei SQL-Statements eine kleine «Spritze» zu injizieren, damit das SQL-Statement so ausgeführt wird, dass alle möglichen Daten ausgegeben werden. Beliebte Ziele sind dabei Userinputs, wie zum Beispiel Anmeldeformulare. Die Anmeldung wird im Hintergrund oft über eine SQL-Abfrage überprüft und diese Abfrage will man nun so modifizieren um an Daten zu gelangen.

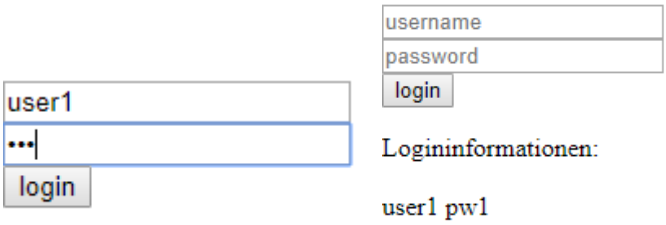
Dies erfordert nur minimalen Aufwand und kleines Knowhow im Bereich SQL. Es gibt auch zahlreiche Webseiten, die zeigen, wie man eine SQL-Injection am besten durchführt mit möglichen Userinputs.

Schutzmassnahmen

Um sich vor SQL-Injections zu schützen, muss man nicht viel Aufwand betreiben. Um SQL-Injections zu verhindern, muss man lediglich dafür sorgen, dass die Userinputs nicht als SQL-Codes gelesen werden, sondern als String, Integer etc. behandelt werden. Dafür gibt es für zum Beispiel PHP vorgefertigte Funktionen, die man schnell implementieren kann. So einfach wie der Angriff ist, so einfach kann man sich auch dagegen schützen. Leider gibt es viele Seiten, die solch eine Schutzmassnahme nicht mit implementiert haben. Dies sorgt immer wieder für Datendiebstahl, sowie Datenverlust.

Codebeispiele

Hier ist ein einfaches Login-Fenster mit Username und Passwort und wenn man auf «login» klickt, werden die Login-Informationen angezeigt.

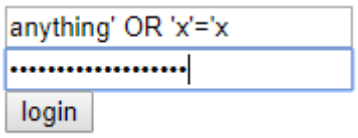


Der Angriff:

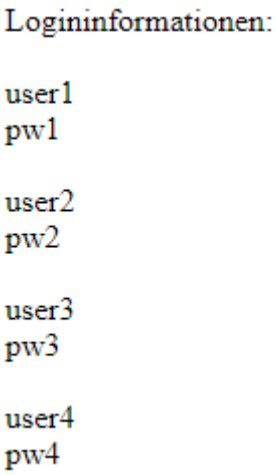
Als Angreifer, weiss man in etwa, wie das SQL-Statement aussieht, in diesem Fall folgendermassen:

SELECT name, passwort FROM user where passwort = '".$pw."' and name = '".$user."'

$pw und $user sind die Werte vom Userinput. Wenn man nun SQL-Code in die Userinputs einfügt, kann man zum Beispiel den Select so modifizieren, dass immer «true» ist und alle Werte somit ausgegeben werden:



Beim Usernamen und Passwort wurde der gleiche Wert angegeben. Das SQL-Statement lautet nun so, dass es alle Werte zurückgibt, bei dem das Passwort und der Username gleich «anything» ist, oder es soll alle zurückgeben wo «x=x». Da «x=x» immer «true» ist, werden dadurch alle Daten zurückgegeben.



Man kann auch weitere Funktionen hinten anfügen wie «DROP TABLE …;» um ganze Tabellen etc. zu löschen.

Die Schutzmassnahme

Bei «Prepared Statements» ist es das Ziel, alle Userinputs so zu deklarieren, dass beim Ausführen des SQL-Codes klar bestimmt wird, ob es sich um einen String, Int, etc. handelt. Somit wird keine Eingabe als SQL-Code behandelt, sondern als normale Variable.

Im ersten Schritt muss man das SQL-Statement vorbereiten:

$stmt = $mysqli->prepare("SELECT name, passwort FROM user where passwort = ? and name = ?");

Die «?» werden im zweiten Schritt ersetzt durch die Userinputs. In diesem Schritt wird auch festgelegt um welchen Datentyp es sich handelt.

$stmt->bind_param("ss", $pw, $user);

s -> String

i -> Integer

d -> Double

b -> Blob

Für jeden Parameter wird der Typ angegeben.

Dies war alles um sich vor SQL-Injections zu schützen. Anschliessend muss man es nur noch ausführen, allenfalls Daten ausgeben und das Statement wieder schliessen. Hier noch ein komplettes Beispiel:

```php
$n = "name";
$p = "passwort";
$stmt = $mysqli->prepare("SELECT name, passwort FROM user where passwort = ? and name = ?");
$stmt->bind_param("ss", $pw, $user);
$stmt->execute();
$stmt->bind_result($n, $p);
while ($stmt->fetch()) {
    printf("%s %s\n", $n, $p);
}
$stmt->close();
```

Wenn man einen «SELECT» macht, muss man am Anfang die Spaltennamen in einer Variable festlegen und nach dem Execute-Befehl, noch «bind_result» mit den Spaltennamen ausführen. Somit kann man die Daten des ausgeführten Statements ausgeben. Bei «INSERT» etc. muss man nur bis zur Execute-Funktion gehen, dann ist das SQL schon fertig ausgeführt.

Somit können keine Daten mehr ausgelesen oder zerstört werden über SQL-Injection.