

SQL Injection – Funktionsweise und Gegenmaßnahmen

EUROSEC GmbH Chiffriertechnik & Sicherheit

Tel: 06173 / 60850, www.eurosec.com

© EUROSEC GmbH Chiffriertechnik & Sicherheit, 2005

Problematik

- SQL-Injection ist eine Angriffstechnik auf Anwendungen, die Benutzereingaben in eine SQL-Datenbank schreiben
- Benutzereingabe kann unerwünschte SQL-Befehle enthalten, die ungeprüft zur Ausführung gelangen
- Mögliche Risiken
 - Zugriff ohne Authentisierung (z.B. wenn die Datenbank für die Benutzerauthentisierung verwendet wird)
 - Unbefugte Datenänderungen (auch wenn ursprünglich nur lesende Zugriffe verwendet wurden)
 - Denial-of-Service (wenn Daten in der Datenbank gelöscht werden)

Beispiel - Authentisierung

- Aufbau einer Beispielanfrage:
 - query = "SELECT * FROM client WHERE username=" + *userName* + " AND ' password=" + *userPwd* + "';"
- Resultierende Anfrage mit gültigen Daten:
 - Benutzername: *Testuser*, Kennwort: *Secret*
 - *SELECT * FROM client WHERE username='Testuser' AND password='Secret';*
- Resultierende Anfrage mit Angriffsdaten:
 - Benutzername: *asd*, Kennwort: *' OR 1=1--*
 - *SELECT * FROM client WHERE username='asd' AND password = ' ' OR 1=1 --;*

Weitere Beispiele

- Unautorisierte Operationen auf der Datenbank ohne Anmeldung möglich:
 - Benutzername: `' ; DROP TABLE Clients--`
 - Benutzer-ID: `123; shutdown--`
- Sogar Ausführung eigener Anfragen ist möglich:
 - Company-ID: `' UNION ALL SELECT CCNumber FROM Customers WHERE '='`
 - query = `"SELECT CompanyAddress FROM Companies WHERE CompanyName = '" + company-ID + "'";"`
 - `SELECT CompanyAddress FROM Companies WHERE CompanyName = '' UNION ALL SELECT CCNumber FROM Customers WHERE''=''`
- Voraussetzung für erfolgreiche Angriffe:
 - Benutzereingaben werden nicht geprüft/gefiltert

Tests

- Jedes Eingabefeld muss auf eventuelle Schwächen getestet werden
 - '
 - *BadValue'*
 - *'BadValue*
 - *' OR '*
 - *' OR*
 - ;
 - ...
 - Dieselben Werte aber URL-kodiert (für die Übertragung mittels HTTP-GET):
 - z.B.: ;%20-- oder %27%3b%2d%2d
- Jeder Fehler (Server-intern oder direkt mit allen Fehlerinformationen ausgegeben) wird für Angreifer ein Indiz sein, dass eine SQL-Injection-Schwachstelle vorliegen kann

Anfällige SQL-Abfragen

- Alle SQL-Abfragen können vom Angreifer modifiziert werden
- "SELECT ... FROM ... WHERE ...=" + *userInput* + ";"
- INSERT INTO TableName VALUES ('Wert1', 'Wert2')
 - "INSERT INTO TableName VALUES (' " + *var1* + " ', ' " + *var2* + " ');"
 - Wert1: ' + (*SELECT TOP 1 FieldName FROM TableName*) + '
 - Wert2: *lakdsjfladsfj*
 - Ergebnis wird in der ersten Spalte angezeigt
- INSERT-Abfragen produzieren u.U. viele Einträge in der Datenbank und können so entdeckt werden

Lösungen - Applikationsebene

- Validierung der Benutzereingaben
 - Filterung: nur erlaubte Zeichen in Eingabefeldern zulassen (ohne weiteres für Benutzernamen möglich)
 - Regular Expression: `s/[^0-9a-zA-Z]//g`
 - Problematisch z.B. bei Kennwortfeldern oder Texteingabefeldern (Bemerkungen o.Ä.), da Sonderzeichen erlaubt sein sollten
 - Generell: Filter spezifisch wie nur möglich
- Single Quote ' bei allen Benutzereingaben in SQL-Anfragen verwenden, auch bei numerischen Angaben
- Fehlermeldungen mit wenig Informationen (am besten nur eine generierte Fehler-ID)
- Längeneinschränkungen für Parameter
 - Je kleiner das Eingabefeld, um so schwieriger SQL-Injection
 - Feldlänge muss serverseitig beschränkt sein

Lösungen - Datenbank

- Benutzerberechtigungen in der Datenbank stark einschränken
 - Rollenbasierte Benutzer (Lesen, Schreiben, Daten-abhängig)
- Strenge Typisierung in der Datenbank
 - Numerische Werte wie Sitzungs-IDs oder Postleitzahlen als ganze Zahlen und nicht als Strings usw.
- DB-Funktionen, Stored Procedures können statt der in der Anwendung zusammengestellten Anfragen definiert und benutzt werden
 - Auf diese Weise bessere Verarbeitung der Anfragen und kleinere Manipulationsfreiheit für Angreifer

Lösung – Prepared Statements

- Grundidee:
 - Keine konkatenierten Strings als SQL-Statements
 - SQL-Statement mit Platzhaltern für Anfragewerte, die später übergeben werden
 - Dadurch strenge Kontrolle, dass tatsächlich nur eine Abfrage gestellt wird. Eine Erweiterung/Änderung der definierten Abfrage nicht möglich
- Java bietet eine Implementierung für Prepared Statements an:
 - Klassen *Statement* und *PreparedStatement*
 - *PreparedStatement* ist performanter, da die Abfrage vorkompiliert wird

Prepared Statement - Beispiel

- Initialisierung:
Connection con;
...
PreparedStatement updateTable1 = con.prepareStatement("UPDATE
table1 SET field1 = ? WHERE field2 LIKE ?");
- Werteübergabe:
updateTable1.setInt(1, 75);
updateTable1.setString(2, "Test");
- Ausführung:
updateTable1.executeUpdate();

Fazit

- Um SQL-Injection zu vermeiden, sind Vorkehrungen auf mehreren Ebenen notwendig:
 - In der Anwendung
 - Auch in der Datenbank
- Benutzereingaben müssen immer überprüft werden
 - Filterung ist absolut notwendig

Anhang

Warum dieser Vortrag von uns?

- Unsere Erfahrung:

- mehrere Personenjahre in Forschungsprojekten zur sicheren Softwareentwicklung; derzeit gemeinsam mit Partnern wie SAP, Commerzbank, Universitäten, ...
- zahlreiche Schwachstellenanalysen für Softwarehersteller, nebst intensiver Feedbackzyklen mit den Entwicklern
- Erstellung von Anforderungs- und Designspezifikationen in mehreren großen Entwicklungsprojekten
- Erstellung von Guidelines zur sicheren Softwareentwicklung, mit Schwerpunkten Banking & Finance, sowie Webapplikationen
- Reverse Engineering und Gutachten von Sicherheitsfunktionen und Kryptomechanismen
- Implementierung von Sicherheitsfunktionen im Auftrag

Abschlussbemerkung

- die vorliegende Dokumentation wurde von EUROSEC erstellt im Rahmen des secologic Forschungsprojekts, Laufzeit 2005 und 2006, nähere Informationen unter www.secologic.org
- wir bedanken uns beim Bundesministerium für Wirtschaft für die Förderung dieses Projektes
- Anregungen und Feedback sind jederzeit willkommen, ebenso Anfragen zu Sicherheitsaspekten, die hier nicht behandelt werden konnten.

Copyright Hinweis

- Diese Folien wurden von EUROSEC erstellt und dienen der Durchführung von Schulungen oder Seminaren zum Thema Sichere Anwendungsentwicklung, mit Fokus Webapplikationen.
- Wir haben diese Folien veröffentlicht, um die Entwicklung besserer Softwareprodukte zu unterstützen.
- Die Folien dürfen gerne von Ihnen für eigene Zwecke im eigenen Unternehmen verwendet werden, unter Beibehaltung eines Herkunftshinweises auf EUROSEC.
- Eine kommerzielle Verwertung, insbesondere durch Schulungs- oder Beratungsunternehmen, wie beispielsweise Verkauf an Dritte oder ähnliches ist jedoch nicht gestattet.