SQL Injektions Attacken bedrohen das Web. Wie funktionieren sie, und wie kann man sie verhindern?

Simon Kokerbeck

Universität Paderborn koker@upb.de

Die schnelle Entwicklung des Internets in den letzten Jahren hat das Leben vieler Menschen beeinflusst. Nicht nur hat sich die Zahl der Internetnutzer stetig erhöht - im Jahr 2007 nutzten laut einer Studie des Statistischen Bundesamtes 68% der deutschen Bevölkerung das Internet regelmäßig [1]. - auch die Art der Anwendungen hat sich in der Vergangenheit verändert. Waren vor einiger Zeit noch statische Internetseiten ohne weitergehende Funktionalität die Regel, so bietet das Internet heute die Möglichkeit interaktiv nach Informationen zu suchen, zu kommunizieren oder komplexe Anwendungen zu nutzen. Diese Entwicklung bringt aber nicht nur Positives mit sich. Durch die Übertragung vertraulicher Informationen oder Zugangsdaten entstehen Sicherheitsrisiken, die bei der Entwicklung von Onlineanwendungen bedacht werden müssen. In dieser Arbeit wird die Sicherheitsbedrohung der sogenannten SQL Injektions Attacken beschrieben und geeignete Maßnahmen aufgezeigt, die es ermöglichen diese Sicherheitslücke möglichst vollständig zu schließen. Im Anschluss wird ein Verfahren - AMNESIA (Analysis and Monitoring for Neutralizing SQL-Injection Attacks) - vorgestellt, welches in Untersuchungen als besonders zuverlässig und wirksam bewertet wurde.

1 Einleitung

Eine einfache Webanwendung besteht typischerweise aus einer HTML Seite, auf der der Benutzer Eingaben in einem Formular tätigen kann. Diese Eingaben werden von einem Programm (z.B. PHP, ASP.NET, CGI oder Java Servlet), welches direkt auf dem Server ausgeführt wird, verarbeitet [2, S.1-2]. Um Daten aus einer Datenbank zu lesen, werden von dem Programm SQL Aufrufe generiert und an die Datenbank gesendet. Die Daten werden dann von der Datenbank an das Programm übergeben und dort für den Benutzer aufgearbeitet. Ein weitere Möglichkeit wäre es aber, auch Änderungen an den Daten in der Datenbank vorzunehmen oder neue Daten hinzuzufügen.

Im Falle von SQL-Injektions Attacken (SQLIAs) wird die Schnittstelle zwischen dem Benutzer und dem serverseitigen Programm dazu benutzt, durch bestimmte Benutzereingaben die Generierung der SQL Anfragen so zu beeinflussen, dass nicht vorgesehene Daten aus der Datenbank gelesen werden oder fälschlicherweise Daten verändert werden. Im Extremfall kann es einem Angreifer dadurch sogar möglich sein, die Kontrolle über die Datenbank zu gewinnen und vorhandene, möglicherweise vertrauliche Informationen einzusehen und zu manipulieren [3, S.1].
In dem zweiten Kapitel werden die Arten von SQLIAs aufgezählt und jeweils an einem Beispiel praktisch dargestellt. Das dritte Kapitel zeigt Maßnahmen auf, mit denen SQLIAs vermieden werden können, und welche Vor- bzw. Nachteile diese Maßnahmen mit sich bringen. Ein konkretes Verfahren, welches als besonders positiv bewertet wurde (AMNESIA), wird im vierten Kapitel genauer beleuchtet und seine Funktionsweise beispielhaft beschrieben. In dem fünften Kapitel wird die Effektivität von AMNESIA und weiterer Verfahren vorgestellt und verglichen.

2 SQL Injektions Attacken

Es gibt eine Vielzahl von verschiedenen Arten von SQLIAs. Je nachdem, welche Arten von SQLIAs von dem anzugreifenden System zugelassen werden und der Intention, die der Angreifer mit der SQLIA verfolgt, können unterschiedliche Arten von SQLIAs Anwendung finden. In diesem Kapitel wird zunächst eine Beispielanwendung vorgestellt, anhand derer die Arten der SQLIAs beispielhaft gezeigt werden. Im Anschluß werden die Typen der SQLIAs beschrieben. Diese Typen stellen dabei immer nur die Oberkategorien der SQLIA dar. Es gibt jeweils viele Variationen der SQLIAs und auch Kombinationen sind denkbar [3, S.3].

2.1 Beispielanwendung

Das folgende Java Servlet stellt eine kleine Bespielanwendung dar, wie sie in vielen Webanwendungen denkbar ist. Der Benutzer wird auf einer HTML Seite aufgefordert, seine Zugangsdaten (Benutzernamen und Passwort) in zwei dafür vorgesehene Formularfelder einzugeben. Diese Daten werden dann per HTTP GET oder POST an den Server übermittelt und von dem Servlet verarbeitet. Dabei generiert das Servlet eine SQL Anfrage, die aus einer Datenbank (möglicherweise vertrauliche) Informationen ausliest und dem Benutzer präsentiert.

```
public class Show extends HttpServlet {
...
1. public ResultSet getUserInfo(String login, String password, int pin) {
2. Connection conn = DriverManager.getConnection("MyDB");
3. Statement stmt = conn.createStatement();
4. String queryString = "";
5. queryString = "SELECT info FROM userTable WHERE ";
6. if ((! login.equals("")) && (! password.equals(""))) {
7. queryString += "login='" + login + "' AND pass='" + password + "' AND pin=" + pin;
}
8. else {
9. queryString+="login='guest'";
}
10. ResultSet tempSet = stmt.execute(queryString);
11. return tempSet;
}
...
}
```

Gibt der Benutzer beispielsweise seine Zugangsdaten `mustermann`, Passwort `myPass` und Pin `12345` ein, so lautet die generierte SQL Anfrage:

```
SELECT info FROM userTable WHERE login='mustermann' AND pass='myPass'
AND pin=12345
```

Falls ein entsprechender Datensatz in der Datenbank existiert, wird dieser abgerufen und angezeigt.

2.2 Arten von SQL-Injektions Attacken

Tautologien

Mithilfe von Tautologien kann eine SQL Abfrage so modifiziert werden, dass ein Auslesen von Daten auch dann möglich ist, wenn falsche Zugangsdaten eingegeben werden. Diese Art der SQLIA dient dem Umgehen von Zugangsmechanismen und dem unberechtigten Auslesen von Daten aus einer Datenbank.
Gibt der Angreifer in dem Feld „Benutzername“ folgendes ein: „'OR 1=1 - -“ und lässt die übrigen Felder frei, dann lautet die generierte SQL Anfrage:

```
SELECT info FROM userTable WHERE login='' OR 1=1 --' AND pass=''
AND pin=
```

Da das OR Konstrukt immer wahr ist, und die Eingabe hinter den Kommentar Zeichen (- -) ignoriert wird, werden dem Angreifer sämtliche Einträge der Spalte info aus dem userTable angezeigt [3, S.3].

Illegale oder logisch nicht korrekte Anfragen

Bei dieser SQLIA wird eine Anfrage durch Eingabe bestimmter Werte in die Formularfelder bewusst so verfälscht, dass die Datenbank kein korrektes Ergebnis zurückgeben kann. Stattdessen wird dem Angreifer eine Fehlermeldung präsentiert, die nützliche Informationen über die Struktur der Datenbank enthalten kann. Das folgende Beispiel setzt dabei voraus, dass es sich bei der Datenbank um einen Microsoft SQL Server handelt. Ähnliche Varianten sind aber auch für andere Datenbanksysteme denkbar.
Gibt der Benutzer im Feld „pin“ folgenden Text ein: „convert(int, (select top 1 name from sysobjects where xtype='u'))“ und lässt die beiden übrigen Felder leer lautet die SQL Anfrage:

```
SELECT info FROM userTable WHERE login='' AND pass='' AND pin=convert(int,
(select top 1 name from sysobjects where xtype='u'))
```

Hierbei versucht die Datenbank den Namen des ersten tables in einen integer Wert umzuwandeln. Falls es sich bei dem Namen nicht um eine Zahl handelt, erscheint eine Fehlermeldung, die wertvolle Informationen für den Angreifer enthalten kann: „Microsoft OLE DB Provider for SQLServer (0x80040E07) Error converting nvarchar value 'CreditCards' to a column of data type int.“. Der Angreifer sieht zum einen, dass es sich bei der Datenbank um einen Microsoft SQL Server handelt, zum anderen sieht er, dass der erste benutzerdefinierte Table „CreditCards“ heißt. Auf gleiche Weise ließen sich weitere Informationen der Datenbank auslesen [3, S.3-4].

Union Query

Diese SQLIA macht sich die Möglichkeit der Vereinigung von Suchergebnissen mittels SQL Keyword `UNION` zu Nutze. Gibt der Angreifer im Feld „Benutzername“ folgendes ein: „'UNION SELECT * FROM CreditCards - -“, dann lautet die generierte SQL Anfrage:

```
SELECT info FROM userTable WHERE login='' UNION SELECT * FROM CreditCards
--' AND pass='' AND pin=
```

Der erste Teil der Anfrage wird kein Ergebnis zurückliefern, da `login` leer ist. Jedoch werden sämtliche Daten des Tables „CreditCards“ angezeigt [3, S.4].

Piggy-Backed Queries

Voraussetzung für diese SQLIA ist es, dass das Datenbanksystem aneinandergereihte SQL Anfragen zulässt. Sind die möglich, kann der Angreifer durch Eingaben in folgender Form „'; DROP TABLE userTable - -“ jede beliebige

SQL Anfrage auf der Datenbank ausführen.

```
SELECT info FROM userTable WHERE login=’’; DROP TABLE userTable
--’ AND pass=’’ AND pin=
```

In diesem Fall würde die erste SQL Anfrage kein Ergebnis liefern, die zweite Anfrage würde den Table userTable löschen. Jedoch wäre auch jede beliebige SQL Anfrage möglich, der Angreifer könnte also beliebige Informationen aus der Datenbank auslesen oder verändern bzw. löschen [3, S.4].

Stored Procedures

Bei den Stored Procedures handelt es sich um Funktionen, die in dem Datenbanksystem gespeichert sind und die zum Beispiel Abfragen auf der Datenbank ausführen können. Dadurch lässt es sich realisieren, dass keine SQL Anfragen an die Datenbank gesendet werden brauchen, sondern lediglich die Prozedur mit den nötigen Variablen auf dem Datenbanksystem aufgerufen wird. Angenommen, das in 2.1 vorgestellte Servlet generiert in den Zeilen fünf bis sieben nicht die SQL Anfrage, sondern ruft folgende Stored Procedure auf der Datenbank auf.

```
CREATE PROCEDURE DBO.isAuthenticated
  @userName varchar2, @pass varchar2, @pin int
AS
  EXEC("SELECT info FROM userTable WHERE login=’" +@userName+
  "’ AND pass=’" +@pass+ "’ AND pin=" +@pin);
GO
```

Genau wie bei den Piggy-Backed Queries könnte der Benutzer in einem der Felder folgenden Text injizieren: „’; SHUTDOWN;- -“. Die SQL Anfrage, die die Stored Procedure generieren würde, würde wie folgt lauten.

```
SELECT info FROM userTable WHERE login=’’; SHUTDOWN; --’ AND pass=’’
AND pin=
```

In diesem Fall hätte das Ausführen der SQL Anfrage das Herunterfahren des Datenbanksystems zur Folge. Es sind jedoch auch beliebige SQL Befehle denkbar. Auch Stored Procedures sind also genauso anfällig gegen die bekannten SQLIAs [3, S.4].

Inferenz

Ziel dieser SQLIAs ist es, ähnlich wie bei den illegalen oder logisch nicht korrekten Anfragen, Informationen über die Struktur der Datenbank zu erlangen. Wenn ein Datenbanksystem bei inkorrekten SQL Anfragen keine aussagekräftigen Fehlermeldungen ausgibt, lassen sich durch eine geschickte Wahl der SQL Injektionen trotzdem Informationen gewinnnen.

Blind injection

Bei der blind injection werden Anfragen an die Datenbank gestellt und das jeweilige Verhalten des Servlets beobachtet. Angenommen es werden folgende zwei Werte in das Feld „Benutzername“ eingetragen und jeweils die Anfrage abgeschickt: „legalUser’ AND 1=0 - -“ und „legalUser’ AND 1=1 - -“. Die resultierenden SQL Anfragen sehen wie folg aus:

```
SELECT info FROM userTable WHERE login='legalUser' AND 1=0 --'
AND pass='' AND pin=
```

```
SELECT info FROM userTable WHERE login='legalUser' AND 1=1 --'
AND pass='' AND pin=
```

Der Benutzername „legalUser“ muss hierfür jeweils ein existierender Benutzername sein. Falls sowohl bei der ersten, wie auch bei der zweiten Anfrage eine Login-Fehlermeldung erscheint, so ist in diesem Servlet das Feld für den Benutzernamen gegen SQLIAs abgesichert, da die Fehlermeldung auch bei der zweiten Anfrage auftaucht, obwohl bei dieser die WHERE Klausel immer wahr ist. Der injizierte Teil der Anfrage (AND 1=1) hat also nicht zu einer positiven Benutzererkennung in dem Servlet geführt. Wenn jedoch die zweite Anfrage anders als die erste Anfrage keine Login-Fehlermeldung ausgibt, ist das Feld für den Benutzernamen für die Ausführung von SQLIAs geeignet [3, S.5].

Timing Attack

Diese Art der Inferenz macht sich das SQL Keyword `WAITFOR` zu Nutze. Es werden Anfragen injiziert die anghängig davon, ob die WHERE Klausel wahr oder falsch ist, die Ausführung der Anfrage um eine bestimmte Zeit verzögern. Wenn zum Beispiel der Angreifer in dem Feld „Benutzername“ folgenden Text injiziert „legalUser’ AND ASCII(SUBSTRING((select top 1 name from sysobjects),1,1)) > X WAITFOR 5 - -“, dann wird das Ergebnis des Aufrufes um fünf Sekunden verzögert, wenn der ASCII-Wert des ersten Buchstaben des ersten Tables größer als ein beliebiger Wert X ist. So lässt sich Stück für Stück der Name eines Tables rekonstruieren [3, S.5].

Alternate Encodings

Die Möglichkeit der Alternate Encodings kann für Angreifer in dem Fall zum tragen kommen, wenn das Servlet soweit gegen SQLIAs abgesichert ist, dass beispielsweise SQL Keywords nicht in den Eingabefeldern eingetragen werden dürfen oder der eingegebene Text auf bestimmte Textbausteine wie das Kommentarzeichen hin überprüft wird (- -). Angenommen das Servlet erlaubt keine SQL Keywords in der Eingabe, dann kann der Angreifer seine SQLIAs anders

codieren. Wenn zum Beispiel der Angreifer folgenden Text in das Feld „Benutzername“ eingibt „legalUser’; exec(char(0x73687574646f776e) - -“, dann lautet die resultierende SQL Anfrage:

```
SELECT info FROM userTable WHERE login='legalUser'; exec(char(0x73687574646f776e)
--' AND pass='' AND pin=
```

Durch die Benutzung der char() Funktion wird die hexadezimal codierte Zeichenfolge 0x73687574646f776e in das SQL Statement `SHUTDOWN` decodiert und die Ausführung führt zum Herunterfahren des Datenbanksystems [3, S.5-6].

3 Maßnahmen zur Vermeidung von SQL-Injektions Attacken

Dieses Kapitel gibt zunächst einen Überblick über die manuellen Verfahren zur Vermeidung von SQLIAs. In einem zweiten Abschnitt werden Techniken aufgezeigt, die eine Webanwendung größtenteils automatisch gegen SQLIAs absichern sollen.

3.1 Manuelle Programmiermethoden

Der Programmierer selbst hat bei der Entwicklung einer Webanwendung die Möglichkeit die Gefahr von SQLIAs möglichst gering zu halten. Zunächst sollte darauf geachtet werden, dass der Typ der eingegebenen Daten überprüft wird. So können beispielsweise Daten aus einem Eingabefeld für numerische Werte nur dann erlaubt werden, wenn diese keine Zeichen außer Ziffern enthalten. Des Weiteren können die generierten Suchanfragen auf bestimmte Textbausteine, wie das Kommentarzeichen oder SQL Keywords, untersucht werden. Auf diese Weise kann eine Vielzahl der gängigen SQLIAs abgewendet werden. Eine dritte sinnvolle Technik besteht darin, die Eingabedaten so zu codieren, dass SQL Keywords o.ä. von dem Datenbanksystem nicht als solche interpretiert werden [3, S.6].

Der Nachteil dieser manuellen Methoden ist jedoch die Anfälligkeit gegen menschliche Fehler. Erfahrungsgemäß werden Angriffspunkte für SQLIAs trotz defensiver Programmierung vorhanden sein oder durch das Finden neuer SQLIAs auftreten [3, S.6].

3.2 Automatisierte Verfahren

Dieser Abschnitt listet die wichtigsten automatisierten Verfahren zur Vermeidung von SQLIAs auf und beschreibt ihre Funktionsweise.

Combined Static and Dynamic Analysis

AMNESIA

AMNESIA durchsucht in der statischen Phase den Quelltext der Webanwendung nach Positionen an denen SQL Anfragen generiert werden. An diesen Stellen wird jeweils ein Modell der möglichen SQL Anfrage erstellt. In der dynamischen Phase werden alle generierten SQL Anfragen gegen das entsprechende Modell geprüft und erst nach positiver Überprüfung an das Datenbanksystem weitergeleitet [3, S.6-7]. Dieses Verfahren wird in Kapitel 4 noch detailierter beschrieben.

SQLCheck / SQLGuard

Diese Verfahren arbeiten ähnlich wie AMNESIA. Auch hier werden in der statischen Überprüfung des Quelltextes Modelle der möglichen SQL Anfragen erzeugt und die dynamisch generierten SQL Anfragen gegen diese Modelle geprüft. Beide Verfahren erstellen Modelle in Form von Grammatiken, die als Sprache nur legale SQL Anfragen akzeptieren. Der Unterschied zwischen SQLGuard und SQLCheck besteht darin, dass bei letzterem das Modell von dem Entwickler spezifiziert wird [3, S.7].

New Query Development Paradigms

SQL DOM / Safe Query Objects

Die beiden Implementierungen SQL DOM und Safe Query Objects basieren auf der Technik, dass die SQL Anfragen nicht durch das Aneinanderknüpfen von einzelnen Strings geschieht, sondern die Anfrage an die Datenbank über ein integriertes API erfolgt [3, S.7]. Dieses Verfahren ist rein dynamisch und funktioniert ähnlich wie die manuelle defensive Programmierung jedoch als vorgefertigtes Paket.

Instruction Set Randomization

SQLrand

SQLrand versieht SQL Keywords mit einem zufällig generierten Integerwert. Dieser Wert müsste einem Angreifer bekannt sein, um eine erfolgreiche SQLIA durchzuführen. Eine SQL Anfrage

```
SELECT info FROM userTable WHERE login='legalUser' AND pass='passwort'
AND pin=12345
```

wird unter der Annahme, dass der zufällige Schlüssel „123“ lautet, in folgende Anfrage überführt.

```
SELECT123 info FROM123 userTable WHERE123 login='legalUser' AND123
pass='passwort' AND123 pin=12345
```

Eine Komponente, welche zwischen Webanwendung und Datenbank geschaltet ist, überprüft, ob jedes SQL Keyword mit dem Schlüssel versehen ist. Injizierte SQL Anweisungen, die diesen Schlüssel nicht enthalten, werden von der Komponente erkannt [4, S.3-4].

4 AMNESIA - Analysis and Monitoring for Neutralizing SQL-Injection Attacks

In diesem Kapitel wird das Verfahren, das AMNESIA zugrunde liegt, genauer beschrieben und beispielhaft verdeutlicht. AMNESIA wurde im Vergleich mit anderen Implementierungen zur Vermeidung von SQLIAs als positiv bewertet [3, S.6 ff], sowohl bei der Wirksamkeit zur Abwehr von SQLIAs, als auch bei der Anwendung.

4.1 Beschreibung des Verfahrens

Das Verfahren, auf dem AMNESIA beruht, lässt sich in vier wesentliche Schritte unterteilen. (1) Untersuchung des Quelltextes nach „Hotspots“, an denen SQL Anfragen generiert werden. (2) Für jeden dieser Hotspots wird ein Modell in Form eines NFA (nichtdeterministischer endlicher Automat) erstellt, dessen Transitionen mit SQL Keywords, Trennsymbolen und Platzhaltern für Benutzereingaben versehen sind. (3) Der Quelltext an jedem der Hotspots wird so angepasst, dass die SQL Anfrage nicht direkt an das Datenbanksystem weitergeleitet wird, sondern zunächst eine Prüfung der SQL Anfrage durch die dynamische Komponente erfolgt. (4) Die dynamische Komponente teilt die aktuell generierte SQL Anfrage in Einzelstücke auf und prüft, ob dieser Pfad in dem statischen Modell existiert. SQL Anfragen, die das statische Modell verletzen werden abgewiesen, andere werden an das Datenbanksystem weitergeleitet [5, S.4].

4.2 Beispiel

Das folgende Beispiel soll zeigen, wie AMNESIA arbeitet und SQLIAs erfolgreich abwehren kann. Als Programm wird auf die Beispielanwendung aus 2.1 zurückgegriffen, wobei hier auf die dritte Benutzereingabe „pin“ verzichtet wurde.

```
public class Show extends HttpServlet {
...
1. public ResultSet getUserInfo(String login, String password) {
2. Connection conn = DriverManager.getConnection("MyDB");
3. Statement stmt = conn.createStatement();
4. String queryString = "";
5. queryString = "SELECT info FROM userTable WHERE ";
6. if ((! login.equals("")) && (! password.equals(""))) {
7. queryString += "login='" + login + "' AND pass='" + password + "'";
}
8. else {
9. queryString+="login='guest'";
}
10. ResultSet tempSet = stmt.execute(queryString);
11. return tempSet;
}
...
}
```

In dem ersten Schritt erkennt AMNESIA, dass in der Zeilen 10 des Programms eine SQL Anfrage an die Datenbank gestellt wird. Diese Position ist nun als Hotspot erkannt. In dem nächsten Schritt wird untersucht, wie sich eine SQL Anfrage dieses Hotspots zusammensetzen könnte. Hierfür gibt es in dem Beispiel zwei Möglichkeiten. Falls die Variablen `login` UND `password` nicht leer sind, sieht eine Anfrage wie folgt aus:

`SELECT info FROM userTable WHERE login='`β`' AND pass='`β`'`,

wobei β als Platzhalter für eine Benutzereingabe fungiert. In dem Fall, dass mindestens eine der beiden o.g. Variablen leer ist, sieht die Anfrage so aus:

`SELECT info FROM userTable WHERE login='guest'`,

Der resultierende NFA hat also zwei mögliche Pfade (vgl. Abb. 1).

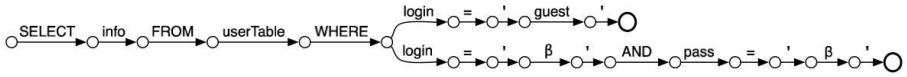

Abbildung 1. Modell der möglichen SQL Anfragen eines Hotspots

Der nächste Schritt besteht darin, den Quelltext des Programms an der Stelle des Hotspots so anzupassen, dass der Anfrage an die Datenbank eine dynamische Prüfung der SQL Anfrage vorgeschaltet ist. Dazu werden die Zeilen 10 und 11 folgendermaßen modifiziert:

```
...
10a. if (monitor.accepts (<hotspot ID>, queryString)){
10b.   ResultSet tempSet = stmt.execute(queryString);
11.  return tempSet;}
...
```

Durch die bedingte Anweisung ist sichergestellt, dass die SQL Anfrage nur dann an die Datenbank gestellt wird, wenn die Methode `monitor.accepts(<hotspot ID>, queryString)` eine erfolgreiche Prüfung der Anfrage durchführt und `true` zurückgibt.

Der Monitor zerlegt die SQL Anfragen zur Überprüfung in einzelne Token, ähnlich wie bei der Erstellung des Modells. Danach wird überprüft, ob die aneinandergereihten Token einem Pfad in dem erzeugten NFA des Hotspots entsprechen (siehe Abb. 2). Die obere Zerlegung (a) stellt hier eine legale Anfrage dar. Sie wird von dem Automaten auf dem unteren Pfad akzeptiert. Die Zerlegung (b) wird hingegen nicht akzeptiert. Der Monitor gibt den Wert `false` zurück und die SQL Anfrage wird verworfen [5, S.5].

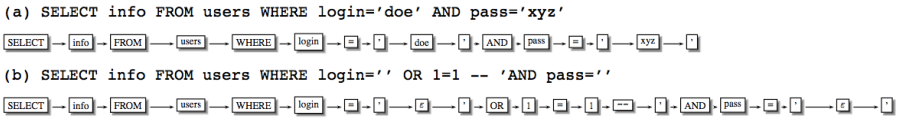

Abbildung 2. In Token zerlegte SQL Anfragen (a): legal (b): illegal

5 Die Qualität verschiedener Verfahren

Anhand verschiedener Kriterien werden die Stärken und Schwächen ausgewählter Verfahren zur Vermeidung von SQLIAs aufgezeigt und zusammengefasst. Unterschieden wird dabei zwischen Techniken, die präventiv die Gefahr von SQLIAs begrenzen, und Techniken, die im laufenden Betrieb der Webanwendung (dynamisch) arbeiten.

Es fällt auf, dass die ausgewählten Verfahren mit Außnahme des Tautologycheckers, welcher explizit nur zum Auffinden von Tautologien geeignet ist, zur Abwehr der meisten Typen von SQLIAs nützlich sind. Problematisch sind aber in fast allen Fällen die Stored Procedures. Bei den Stored Procedures wird von der Web Anwendung keine SQL Anfrage generiert, sondern eine entsprechende, auf dem Datenbanksystem gespeicherte Prozedur ausgeführt (vgl. 2.2). Von den dynamischen Verfahren sind lediglich drei (AMNESIA, SQLCheck, SQLGuard) gegen codierte SQLIAS (Alternate Encodings) wirksam.

Tabelle 1. Vergleich der Wirksamkeit dynamischer Verfahren

Verfahren	Tautol.	Illegal	Union	Piggyb.	Sto.Proc.	Inferenz	Alt.Enc.
AMNESIA	√	√	√	√	x	√	√
CSSE	√	√	√	√	x	√	x
SQLCheck	√	√	√	√	x	√	√
SQLGuard	√	√	√	√	x	√	√
SQLrand	√	x	√	√	x	√	x
Tautology-checker	√	x	x	x	x	x	x
Web App. Hardening	√	√	√	√	x	√	x

Tabelle 2. Vergleich der Wirksamkeit präventiver Verfahren

Verfahren	Tautol.	Illegal	Union	Piggyb.	Sto.Proc.	Inferenz	Alt.Enc.
Java Static Tainting	√	√	√	√	√	√	√
Safe Query Objects	√	√	√	√	x	√	√
SQL DOM	√	√	√	√	x	√	√
WebSSARI	√	√	√	√	√	√	√

Diese Techniken nutzen zum dekodieren der Zeichenfolgen dasselbe Verfahren, welches auch in dem Datenbanksystem selbst zum Tragen kommt. Daher werden codierte SQL Anfragen dekodiert behandelt und wirksam geprüft [3, S.8-9].

Die Verfahren SQLCheck, SQLGuard, SQL DOM und SQLrand erfordern Änderungen des Quelltextes der Webanwendung durch den Entwickler. Dies ist bei den übrigen Verfahren nicht erforderlich. Durch zusätzlichen Programmieraufwand des Entwicklers werden Fehler und daraus resultierende Lücken bei der Vermeidung von SQLIAs wahrscheinlicher. Diese Problematik ist für die Verfahren, die keine Änderungen am Quelltext erfordern bzw. diese automatisch durchführen (z.B. AMNESIA), geringer [3, S.9].

5.1 Wirksamkeit von AMNESIA

Am Beispiel AMNESIA lässt sich die Wirksamkeit und die Zuverlässigkeit bei der Vermeidung von SQLIAS sehr gut zeigen. Halfond und Orso haben hierfür an verschiedenen Fallbeispielen getestet, wie viele SQLIAs sich durch die Nutzung von AMNESIA verhindern lassen. Verschiedene kommerzielle Webanwendungen, deren Quelltext zur Verfügung stand, wurden dabei durch zahlreiche SQLIAs verschiedenster Typen attackiert. Gezählt wurden die Anzahl der SQLIAs, welche die „ungeschützte“ Webanwendung nicht beeinflussen konnten (ungefährlich), die Anzahl der SQLIAs, welche die Webanwendung erfolgreich attackieren konnten (gefährlich), und die Anzahl der SQLIAs,

welche durch die Nutzung von AMNESIA erfolgreich abgewehrt werden konnten.

Tabelle 3. Effektivität von AMNESIA

Webanwendung	ungefährlich	gefährlich	davon durch AMNESIA abgewehrt
Checkers	1195	248	248
Office Talk	598	160	160
Employee Directory	413	280	280
Bookstore	1028	182	182
Events	875	260	260
Classifieds	823	200	200
Portal	880	140	140

In allen Fällen wurde durch die Nutzung von AMNESIA eine einhundertprozentige Erfolgsquote bei der Abwehr von SQLIAs erzielt. Ein sehr gutes Ergebnis, welches für AMNESIA spricht und zeigt, dass die Gefahr von SQLIAs durchaus wirksam bekämpft werden kann [5, S.8-9].

6 Fazit

Die Gefahr, die von SQLIAs ausgeht, ist in unserer Zeit, in der Datenhaltung im Internet die Regel ist, erheblich. Nicht nur die Tatsache, dass vertrauliche Daten von dritten eingesehen werden können, sondern auch dass Daten bewusst verändert oder gelöscht werden können, ist sehr bedenklich. Es zeigt sich jedoch auch, dass es nicht nur Forschungsgegenstand ist, SQLIAs abzuwehren, sondern dass bereits wirksame und gut zu bedienende Lösungen existieren, die das Problem der SQLIAs erfolgreich bekämpfen. Durch die Aufklärung und Sensibilisierung der verantwortlichen Entwickler von Webanwendungen können Daten, die online in Datenbanksystemen gespeichert sind, sicher verwahrt und nur den befugten Personen zugänglich gemacht werden.

Literatur

1. Statistisches Bundesamt Deutschland (2007) Pressemitteilung Nr.486 vom 30.11.2007. Fast 70% der Bevölkerung ab zehn Jahren nutzen das Internet
2. Gregory T. Buehrer, et.al. (2005) Using Parse Tree Validation to Prevent SQL Injection Attacks
3. William G.J. Halfond, et.al. (2006) A Classification of SQL Injection Attacks and Countermeasures
4. Stephen W. Boyd, Angelos D. Keromytis (2004) SQLrand: Preventing SQL Injection Attacks
5. William G.J. Halfond, Alessandro Orso (2005) AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks