Ausgabe 04/2020

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

D: 4,90 EUR | CH: 9,80 CHF | A: 5,60 EUR | Benelux: 5,80 EUR

4 191978 304903 04

Java aktuell

iJUG Verbund
www.ijug.eu

Java 14
Einblick in die neuen Features

Testing
Neue Impulse für das effektivere Testen von Software

Testautomatisierung
Sind automatisierte Tests eine Illusion?

NoSQL-Injections und wie sie verhindert werden können

Matthias Altmann, Micromata GmbH

Mit der zunehmenden Beliebtheit von NoSQL-Datenbanken wächst auch die Gefahr, dass sie zum Ziel von Cyberattacken werden. Dieser Artikel beschreibt zunächst die Gefahrenlage, um dann praktische Tipps zur Abwehr von NoSQL-Injections zu geben.

Die Speicherung von Daten erfolgte bisher in der Regel über die Structured Query Language, kurz SQL. Heute hat sich darüber hinaus mit den sogenannten NoSQL-Datenbanken („Not only SQL") ein alternativer Ansatz etabliert, der zunehmend an Bedeutung gewinnt. Die Website db-engines.com listet 25 Prozent aller derzeit genutzten DB-Systeme als NoSQL-Datenbanken, Tendenz steigend.

Im Falle der klassischen SQL haben wir es traditionell mit eher festen Strukturen zu tun, die angesichts der zunehmenden Digitalisierung nicht ohne Weiteres mitskalieren. Zum einen, weil das Datenvolumen weltweit rapide steigt, zum anderen, weil diese Daten als logische Folge einer immer intensiveren Nutzung digitaler Technologien vielfältiger und diverser werden: Kundendaten oder Social-Media-Daten, Daten aus komplexen Lieferketten oder Webtraffic-Daten, Sensordaten aus Smartphones, Kameras und Automobilen. Jeden Tag bewegen wir einen schier unendlichen Datenstrom – mit dem auch unsere Datenbanken umgehen müssen.

Ein probates Mittel, der Lage Herr zu werden, sind NoSQL-Datenbanken: Sie kommen deshalb gut mit der dynamischen Datenentwicklung zurecht, weil sie wesentlich besser mit schnell veränderbaren Daten umgehen können.

NoSQL – Paradigmenwechsel für die Softwareentwicklung

Dazu brechen sie mit den Paradigmen von relationalen Datenbanken und mit SQL. So erlaubt NoSQL beispielsweise deutlich flexiblere, offenere Datenstrukturen und -abfragen, ist hochskalierbar und erfordert nicht zwingend, dass Daten sofort konsistent zur Verfügung stehen müssen. Das hat nicht nur zur Folge, dass Änderungen in NoSQL-Datenbanken viel schneller und einfacher durchgeführt werden können – sondern auch, dass mehr Code aus der Datenbank in die Programmlogik wandert.

Vieles, was früher der Datenbankadministrator beziehungsweise die Datenbank selbst durchgeführt hat, wandert nun als Aufgabe zum Softwareentwickler.

NoSQL als Herausforderung für die Datensicherheit

Allerdings sind Programmiersprachen und Frameworks oft nicht ausreichend gehärtet, um das Entstehen von Sicherheitslücken gänzlich zu verhindern. Deshalb sollten Softwareentwickler unbedingt wissen, was sie je nach NoSQL-System, um die es im weiteren Verlauf gehen soll, beachten müssen.

Die Agentur für Cybersicherheit gibt jedes Jahr einen Report zur aktuellen Sicherheitslage *[1]* im Netz heraus. Der letzte Stand von Januar 2020 beschreibt noch das Jahr 2018 und zeigt, dass der Angriff auf SQL-Datenbanken mit Abstand der häufigste ist. Da NoSQL-Datenbanken im Wachstum sind, kann sich dieser anhaltende Trend schnell auch auf diesen Datenbanktyp ausweiten. Datenbanken vor unbefugtem Zugriff zu schützen sollte also oberstes Gebot einer professionellen IT-Security sein.

NoSQL-Injections wirksam abwehren

Die Gefahren auf dem Gebiet von NoSQL sind bislang noch kaum untersucht, ausgereifte Tools in dem Bereich fehlen auch hier. Anders als bei SQL gibt es hier auch keinen Standard, an dem sich die einzelnen Abfragetypen orientieren. Dadurch können auf unterschiedliche Weisen Lücken aufgemacht werden, deren Tragweite (Stand heute) noch schwer abzuschätzen ist. Was wir wissen, ist: Das Einschleusen von Späh- oder Schadcode gelingt vor allem dort, wo Eingaben nicht korrekt überprüft werden oder wo Daten durch sogenannte String-Konkatenation mit Code vermischt werden.

Deshalb ist es wichtig, Daten und Code sortenrein voneinander zu trennen. Denn wenn wir keine Ordnung zwischen beiden herstellen, sind auch bei NoSQL prinzipiell ähnliche Attacken möglich wie auf relationale Datenbanken. Diese sogenannten NoSQL-Injections sind verheerend, weil sie die Daten in der Datenbank nicht nur auslesen, sondern auch verändern können. Schlimmer noch: Im Unterschied zu relationalen Datenbanken befindet sich der Angreifer hier oft nicht nur auf der Datenbank selbst, sondern hat auch Zugriff auf den Anwendungscode. Das bedeutet, dass er im Ernstfall die komplette Anwendung umbauen kann.

Sanitizer und Validations für mehr Datenhygiene

Es folgen zwei Beispiele mit MongoDB, um zu veranschaulichen, wie solche Angriffe aussehen können – inklusive Coding-Tipps, wie man das verhindern kann.

Beispielangriff auf Document Store von MongoDB

Der Document Store von MongoDB ist derzeit das beliebteste NoSQL-System *[2]*. Nehmen wir beispielhaft an, Request-Daten wandern hier direkt in eine Abfrage – im in *Listing 1* gezeigten Beispiel mit PHP der Einfachheit halber als GET. Dann sieht eine Standardabfrage zum Beispiel so aus: `http://mydomain.com?user=matthias`.

Problematisch sind nun die Operatoren: Mittels `http://mydomain.com?user[$ne]=matthias` können nun alle anderen Nutzer abgefragt werden. Dadurch wird daraus die in *Listing 2* gezeigte Anfrage. `$ne` ist der Operator für „nicht gleich". Alles, was nicht gleich „matthias" ist, wird für `user` ebenso herangezogen. Ähnliches gilt für `http://mydomain.com?user[$gt]=""`. Hier sind alle Strings größer als der leere String.

Sieht der Code zum Abfragen eines Nutzerzugangs wie in *Listing 3* aus, so können mit `http://mydomain.com?user[$ne]=xyz&pass[$ne]=p` alle Zugänge abgefragt werden, die nicht dem Zugang mit Nutzer „xyz" und nicht dem Passwort „p" entsprechen.

Verwendet der Softwareentwickler in diesem Szenario nun die Menge der Passwort-Rückmeldungen für das Login, hat sich der Angreifer erfolgreich eingeloggt – und kann dies jetzt durch Austauschen des Nutzernamens beliebig vielen weiteren Nutzern ermöglichen.

Beispielangriff mit Java auf MongoDB

Ähnliches gilt auch in der Java-Welt. Die aktuelle Library 3.4 von MongoDB verwendet ein sogenanntes BasicDBObject für den Zugriff *[3]*. Wird stattdessen aber die Eingabe als JSON geparst, so sind dergleichen Angriffe möglich.

Als Beispiel wird hier der Nutzername in der Anmeldemaske in der Variablen `user` hineingereicht. Statt also *Listing 4* auszuführen, geschieht das in *Listing 5* gezeigte Szenario. Dies führt im Ergebnis dazu, dass die Eingabe `matthias', name:{$ne:'matthias'}, password:'mypass` in der Folge alle anderen Nutzer ausgibt, wenn `matthias/mypass` die Credentials sind.

```
$query = array(“user” = > $_GET[‘user’]);
$result = $collection.find($query);
```

Listing 1

```
$query = array(”user” => array(“$ne”: matthias));
```

Listing 2

```
$query = array(“user” = > $_GET[‘user’], “pass” => $_GET[‘pass’]);
$result = $collection.find($query);
```

Listing 3

```
BasicDBObject dbQuery = new BasicDBObject("user", user);
DBCursor result = characters.find(dbQuery);
```

Listing 4

```
String stringQuery = "{ 'user' : '" + user + "'}";
DBObject query = (DBObject) JSON.parse(stringQuery);
DBCursor result = characters.find(query);
```

Listing 5

Validations und Sanitizer gegen NoSQL-Injections

Je nach Framework und Programmiersprache sind die Methoden gegen die Angriffe unterschiedlich, Folgendes sollten wir aber prinzipiell beherzigen.

Validation

Die erste Regel lautet: Eingaben validieren! Wir wissen ja zum Glück einiges über den Nutzer – zum Beispiel, dass er ein String sein muss, oder auch, dass eventuell nur bestimmte Zeichen oder Längen verwendet werden dürfen. Also prüfen wir erst mal genau das.

In PHP können wir dafür `is_string` verwenden – oder, sofern wir bestimmte Klassen wie E-Mails, URLs oder IPs identifizieren wollen, auch `filter_var`. Bei Java bieten sich Apache Commons hierfür an.

Sanitizing

Die zweite Regel lautet: Sanitizing! Im ersten Schritt haben wir geprüft, ob die Bedingungen, die wir bei der Eingabe erwarten, tatsächlich stimmen. In diesem zweiten Schritt geht es darum, alles rauszuwerfen, was nicht reinpasst.

- In PHP kann diese Eingaben-Desinfektion ebenfalls mit `filter_var` erledigt werden. Hierzu bietet die Sprache verschiedene Filterregeln an, die man mitgeben kann, zum Beispiel `FILTER_SANITIZE_STRING`.
- In JavaScript kann für diesen Zweck die Library `mongo-sanitize` [4] eingesetzt werden.
- In Java empfiehlt sich dafür meistens Apache Commons – hier zum Beispiel `normalize`- und `strip`-Methoden anwenden.

In anderen Programmiersprachen und Frameworks können diese beiden Maßnahmen anders aussehen oder bereits durch das Framework oder die Sprache erledigt werden. Im Zweifelsfall sollten Softwareentwickler dies nachschlagen.

Tools zur Behebung von NoSQL-Lücken

Um NoSQL-Injections zu verhindern, kommt auch dem Pentesting eine entscheidende Rolle zu.

Ausgereifte Tools zum Aufspüren von NoSQL-Lücken müssen allerdings noch entwickelt werden. Versuche wie NoSQLMap *[5]* haben zurzeit noch großes Potenzial nach oben, insbesondere im Bereich der Webschnittstelle. SQLMap kann bisher gar kein NoSQL integrieren *[6]*. Bis es so weit ist, bleibt auch hier nur das manuelle Testen.

Zusammenfassung

In Zeiten wachsender Popularität von NoSQL-Datenbanken und der noch immer großen Beliebtheit von Datenbankangriffen ist es wichtig, unsere IT-Systeme so weit wie möglich abzudichten. Hierfür ist bis auf Weiteres den Empfehlungen der jeweiligen Datenbank Folge zu leisten, sofern solche vorhanden sind.

Falls nicht, empfiehlt es sich, Eingaben zu validieren und zu säubern. Wer von uns will schließlich für den nächsten großen Daten-Leak verantwortlich sein, bei dem große Datenmengen illegal abgeschöpft werden konnten?

Quellen

[1] *https://www.enisa.europa.eu/publications/enisa-threat-landscape-report-2018*
[2] Stand 5.2.2020 *https://db-engines.com/en/ranking*
[3] *https://mongodb.github.io/mongo-java-driver/3.4/javadoc/com/mongodb/BasicDBObject.html*
[4] *https://www.npmjs.com/package/mongo-sanitize*
[5] *https://github.com/codingo/NoSQLMap*
[6] *https://github.com/sqlmapproject/sqlmap/issues/1674*
[7] *https://www.meetup.com/de-DE/IT-Security-Kassel/*
[8] *https://www.heise-devsec.de/2019/lecture.php?id=8488&source*
[9] *https://www.micromata.de/referenzen/publikationen/*
[10] *https://secf00tprint.github.io/blog/about.html*

Matthias Altmann
Micromata GmbH
m.altmann@micromata.de

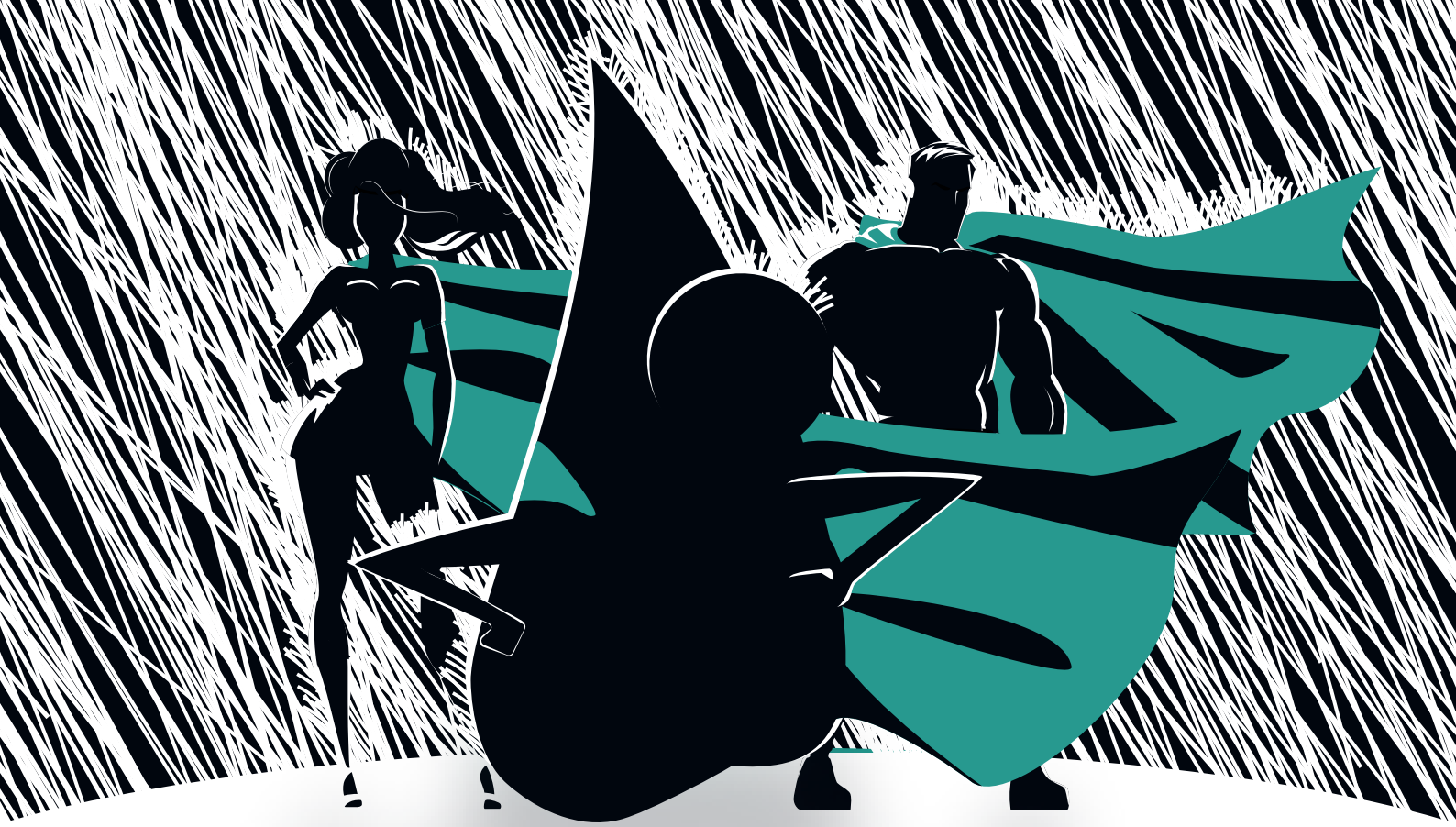
Matthias Altmann ist Softwareentwickler und IT-Security-Experte bei der Micromata GmbH, wo er gemeinsam mit seinen Kollegen den Bereich IT-Sicherheit betreut und fortentwickelt. Er ist außerdem Mitbegründer und Organisator des IT-Security-Meetups Kassel [7] und teilt sein Know-how darüber hinaus auf Fachkonferenzen [8], in Fachbeiträgen [9] und gelegentlich auch auf seinem Blog [10].

Java aktuell

Mehr Informationen zum Magazin und Abo unter:

https://www.ijug.eu/de/java-aktuell

iJUG Verbund
www.ijug.eu



IHR SEID UNSERE JavaLand HEROES!

Zahlreiche Teilnehmer, Referenten und Sponsoren haben entschieden, die JavaLand in der Krise mit einem finanziellen Beitrag zu unterstützen.

Wir sind überwältigt von eurer Großzügigkeit.
Schön, dass wir auf eine starke Community zählen können!

Bisher konnten wir dadurch über 46.000 Euro sammeln.
Insgesamt müssen aufgrund des Veranstaltungsausfalls mehr als 200.000 Euro gedeckt werden.

JavaLand

Besucht unsere Hall of Fame: