NoSQL-Injection Angriffe verhindern

In diesem Abschnitt werden verschiedene Methoden erläutert, wie NoSQL-Injection Angriffe verhindert werden können. Dabei werden verschiedene Methoden erläutert und spezifische Beispiele für eine MongoDB Datenbank in PHP-Umgebung gegeben.

## Methoden:

1. **Search and Replace**: Durch das Überprüfen der Benutzereingabe kann verhindert werden, dass Datenbankoperatoren (wie z.B. $where) ausgeführt werden. Dafür kann die Benutzereingabe auf nicht gewünschte Zeichen wie „$“ oder andere Sonderzeichen geprüft werden, um diese dann aus der Eingabe zu entfernen. In einer PHP-Umgebung wäre dies mit der Funktion preg\_replace möglich [1].
2. **Type Casting**: Beim Type Casting wird die Eingabe des Benutzers in den gewünschten Datentyp umgewandelt. Bei der Eingabe von LogIn-Daten wäre dies bspw. eine Umwandlung der Benutzereingabe in den Datentyp String. In einer PHP-Umgebung wäre dies mit der Funktion type-juggling möglich [2][6].
3. **Prepared Statements**: Benutzereingaben sollten getrennt von der Datenbankabfrage verwendet werden, um zu verhindern das die Benutzereingabe von der Datenbank als Teil der Abfrage interpretiert wird. Mögliche Datenbankoperatoren aus der Benutzereingabe werden so nicht in der Datenbankabfrage ausgeführt [3].
4. **Least-Privilige-Prinzip**: Benutzeraccounts sollten die geringstmöglichen Zugriffsrechte zur Datenbank haben. Durch eine sorgfältige Verteilung von Zugriffsrechten, kann bei einem erfolgreichen Angriff der mögliche Schaden reduziert werden, da Angreifer durch die niedrige Zugriffsebene auf weniger Daten Zugriff haben [3][4].
5. **Whitelist**/**Blacklist**: Ähnlich zur Search and Replace Methode kann auch eine Whitelist/Blacklist mit zulässigen/unzulässigen Zeichen erstellt werden, um Angriffe zu verhindern. Bei einer Whitelist werden die Benutzereingaben auf zulässige Zeichen überprüft. Nur Eingaben mit zulässigen Zeichen werden erkannt und im Anschluss ausgeführt. Das Prinzip der Blacklist funktioniert analog dazu umgekehrt.
6. **Aktuelle Software-Versionen**: Beim Implementieren einer Anwendung sollte darauf geachtet werden die neuesten stabilen Versionen der verwendeten Architektur zu verwenden, um interne Sicherheitslücken in der Architektur zu vermeiden.
7. **Pentesting/Security Audits**: Eine weitere Möglichkeit ist das Nutzen von Security Audits. Diese Tools überprüfen die Einstellungen der Datenbank auf bekannte Fehler und geben Empfehlungen wie man seine Datenbank sicherer gestalten kann. Außerdem können einige dieser Tools auch Pentesting durchführen, um Schwachstellen in der Architektur zu finden. Für MongoDB wäre das bspw. MongoAudit [10].
8. **Libraries/Packages**: Es gibt auch bereits Libaries oder Packages, die mehrere der vorher genannten Methoden umsetzen. Für MongoDB wären das bspw. mongo-sanitize oder mongoose, welche die Benutzereingaben überprüfen und Type Casting oder Search and Replace beinhalten [7][8][9][14].
9. **Secure Deployment**: Nicht jeder sollte direkten Zugriff auf die Datenbank haben. Nutzer sollten bspw. keinen direkten Zugriff auf die Datenbank haben, sondern nur über den Anwendungsserver. Beim Nutzen von APIs für den Zugriff auf Datenbanken sollte drauf geachtet werden, dass diese ausreichend geschützt sind. Bspw. sollte Input im JSON-Format nur als Content Type akzeptiert werden um Angriffe über die API zu verhindern. Gerade beim Nutzen von Drittentwickler-APIs sollte beachtet werden, dass diese APIs ausreichen gesichert sind [11].
10. **Monitoring/Attack Detection**: Nach dem Deployment der Anwendung kann man weiterhin die Aktivitäten beobachten, um Angriffe auf die Datenbank zu erkennen. Eine Möglichkeit sind Web Application Firewalls (WAF), welche den Datenfluss über HTTP-Anfragen beobachten und Angriffe erkennen können. Eine weitere Möglichkeit ist die Aktivität der Datenbank zu beobachten. Dafür gibt es direkte Möglichkeiten wie den MongoDB Cloud Manager für MongoDB [12]. Außerdem gibt es für viele weitere Monitoring Angebote von Drittanbietern seine Dantebank zu beobachten [11][13]. Durch das beobachten der Anfragen auf die Datenbank können dann ungewöhnliche oder bösartige Anfragen erkannt werden.

## Beispiele für MongoDB auf PHP-Server

1. LogIn Website ohne jegliche Schutzvorkehrungen
2. Login Website mit Funktion die Search and Replace (Methode 1). Search and Replace kann in PHP mit folgender Funktion umgesetzt werden: „*preg\_replace("/[^a-zA-Z0-9\_-]/", "", $input)“* [1]
3. Login Website mit Funktion die Type Casting umsetzt (Methode 2). Type Casting kann mit dem gewünschten Datentyp vor der Variable umgesetzt werden: „*(string)$input“* [2]
4. Paketbeispiele für MongoDB: z.B. Mongoose [8], mongo-sanitize [7]
   1. Für die PHP-Umgebung im Beispiel wurde die mongo-sanitize PHP Version genutzt [14]. Diese wird über den Dependency Manager composer installiert und kann dann in PHP genutzt werden [15].
5. Pentesting & Security Audits Beispiel mit Umsetzung von Mongoaudit [5][10]

Quellen

[1] <https://www.php.net/manual/en/function.preg-replace.php>

[2] <https://www.php.net/manual/en/language.types.type-juggling.php>

[3] <https://infosecwriteups.com/nosql-injection-8732c2140576>

[4] <https://www.cyberark.com/de/what-is/least-privilege/>

[5] <https://geekflare.com/de/nosql-security-scan/>

[6] <https://owasp.org/www-pdf-archive/GOD16-NOSQL.pdf>

[7] <https://www.npmjs.com/package/mongo-sanitize>

[8] <https://mongoosejs.com/>

[9] <https://appsecco.com/blog/hacking-apps-using-nosql-injection>

[10] <https://github.com/stampery/mongoaudit>

[11] Ron, Aviv & Shulman-Peleg, Alexandra & Puzanov, Anton. (2016). Analysis and Mitigation of NoSQL Injections. IEEE Security & Privacy. 14. 30-39. 10.1109/MSP.2016.36.

[12] <https://www.mongodb.com/docs/manual/administration/monitoring/>

[13] <https://sematext.com/blog/mongodb-monitoring-tools/>

[14] <https://github.com/Aldin-SXR/mongo-sanitize>

[15] <https://getcomposer.org/download/>