SQL injection vulnerabilities in modern web applications

Luca Marlon Schneider

HTWK Leipzig
luca.schneider@stud.htwk-leipzig.de

09.12.2024

Zusammenfassung

Auch 26 Jahre nach der ersten, von Jeffrey Moss im Phrack-Magazin dokumentierten SQL-Injection-Schwachstelle bleibt diese Technik eine erhebliche Herausforderung für die Sicherheit moderner Webanwendungen. Besonders im Zeitalter datenbankgestützter Systeme entfaltet sich ihr destruktives Potenzial – trotz zahlreicher präventiver Maßnahmen. Es stellt sich die Frage, warum SQL-Injections weiterhin so verbreitet und unvermindert effizient bleiben.

1 Einführung

Trotz kontinuierlicher Bemühungen im Bereich der Websicherheit bleibt SQL“ Injection (SQLi) ein anhaltendes Problem. In der 2021 veröffentlichten ”OWASP Top Ten” Statistik, welche die größten Risiken für Webanwendungen des Jahres empirisch bewertet, belegten Injections den dritten Platz[1]. Diese hohe Platzierung unterstreicht das anhaltende Ausmaß der Bedrohung, welches SQL-Injection auch in modernen Webarchitekturen innewohnt.

Resultate der SQL-Injection-Angriffe fallen dabei teils verheerend aus und reichen von Umgehen lückenhaft implementierter Authentifizierungsmechanismen bis hin zur Exfiltration sensibler, interner Daten. Besonders gefährlich ist hierbei die Kombination aus einer oft vergleichsweise simplen Angriffsmethodik und deren potenziell tiefgreifenden Folgen.

Dieser Gefahr ebnen zahlreiche Aspekte den Weg. Grundsätzlich lassen sich diese jedoch in drei thematische Unterklassen kategorisieren, die in den folgenden Abschnitten detaillierter beleuchtet werden.

2 Architekturbedingte Schwächen von SQL-Systemen

Die wohl ausschlaggebendste Schwachstelle von SQL ist eine grundlegende Anfälligkeit, die dem System von Haus aus eigen ist. Besonders das Fehlen einer eindeutigen Trennung zwischen Daten und ausführbarem Code wird dabei zum Verhängnis. Während diese Trennung einerseits zu den größten Stärken von SQL gehört und seiner Flexibilität dient, schafft sie gleichzeitig Spielraum für Angreifende, eigenen Code in die Abfragen einzuschleusen. Besonders gefährlich wird es bei dynamischen Abfragen, bei denen Benutzereingaben direkt und ungefiltert in die Abfrage integriert werden.

```
SELECT * FROM users WHERE username =
'input';
```

Dieser schädliche, als Payloads bezeichnete Code kann dabei je nach Ziel der Attack unterschiedliche Formen annehmen. Es zieht sich jedoch ein roter Faden durch die für SQL-Injection entworfenen Payloads. Der Angreifende macht sich hierbei die SQL-Syntax zu nutze und das Verfahren der „String Termination“ aus. Durch das Setzen eines einzelnen Apostrophs (‘) beendet er vorzeitig den String, in den die Benutzereingabe gespeichert werden soll. Alles, was auf dieses Zeichen folgt, wird von SQL als ausführbarer Code interpretiert, wodurch dem Angreifende freie Hand gelassen wird. Ein einfaches Payload zum Löschen einer ganzen Tabelle kann folgendermaßen aussehen:

```
SELECT * FROM users WHERE username =
''; DROP TABLE users; --';
```

Sowohl die Art des Feedbacks, das Angreifer für die Auswertung nutzen (z. B. Time-Based, Blind oder Third-Party-Injections), als auch die eingesetzten Payloads variieren je nach Ziel und beabsichtigter Wirkung[SJ14]. Dennoch bleibt der Aufwand der Angriffe meist überschaubar und leicht zu automatisieren.

[1] Open Worldwide Application Security Project (OWASP), ”OWASP Top Ten 2021”, https://owasp.org/www-project-top-ten/

3 Wachsende Komplexität

Im starken Kontrast dazu steht das wachsende Problem immer größerer und komplexerer Webanwendungen, die steigende Anforderungen an Ressourcen und qualifiziertes Fachpersonal stellen – sowohl im Aufbau als auch in der Wartung. Diese zunehmende Komplexität führt zu einem Anstieg potenzieller Sicherheitslücken. Webanwendungen, die ständig weiterentwickelt werden, bringen oft unentdeckte Schwächen mit sich, die durch die komplexe Architektur und die Vielzahl an Integrationen schwer zu identifizieren sind. Lehman's Gesetze der Software-Evolution verdeutlichen [AC16], dass Software mit der Zeit mehr Features und Funktionen integriert, was die Angriffsfläche vergrößert.

4 Legacy-Code als Barriere

Ein häufig übersehener Faktor bei der Anfälligkeit von SQL-Injektionen ist der Bestandscode älterer Webanwendungen, der nach wie vor zentrale Komponenten dieser Systeme ausmacht. Neues Personal, das nicht an der Entwicklung dieses Codes beteiligt war oder nicht ausreichend in dessen Wartung eingeweiht wurde, ist oft mit den spezifischen Sicherheitsanforderungen des alten Codes nicht vertraut.

Zudem führt die Kurzlebigkeit technologischer Trends dazu, dass jüngeres Personal häufig nicht im Umgang mit älteren Technologien geschult wurde. Diese Lücken in der Schulung und das daraus resultierende fehlende Verständnis erschweren die kontinuierliche Gewährleistung von Sicherheit. Laut dem 2020 veröffentlichten Open Source Security and Risk Analysis Report (OSSRA), der vom Synopsys Cybersecurity Research Center (CyRC) erstellt wurde[2], enthalten 82 % aller untersuchten Codebasen Komponenten, die mehr als vier Jahre alt sind. Parallel wurden in 75 % der gemessenen Codebasen potenzielle Schwachstellen identifiziert.

5 Fazit

Um die Sicherheit vor SQL-Injektionen langfristig zu sichern, sollten verschiedene präventive Maßnahmen ergriffen werden. Eine grundlegende Sicherheitsmaßnahme ist die Eingabevalidierung, bei der Benutzereingaben auf korrekte Formate und Werte geprüft werden, um schadhafte Eingaben frühzeitig zu identifizieren. Zudem empfiehlt es sich, Abfragen zu parametrisieren. Dieser Ansatz trennt Benutzereingaben von SQL-Befehlen und schützt so vor Manipulation.

Auch der Einsatz von Object-Relational-Mapping (ORM)-Tools, wie etwa Hibernate[3] oder SQLAlchemy[4], ist eine etablierte Methode, um den direkten Umgang mit SQL zu minimieren und gleichzeitig integrierte Sicherheitsmechanismen zu nutzen.

Abseits des Codes gibt es verschiedene Ansätze, die zur Verbesserung der Sicherheit beitragen können. Ein zentrales Ziel sollte es sein, den Anteil veralteten Codes zu minimieren und gleichzeitig das Onboarding neuer Teammitglieder zu erleichtern, besonders für junge Entwickler. Eine detaillierte und nachvollziehbare Dokumentation stellt hierbei einen oft unterschätzten Vorteil dar. Darüber hinaus kann eine Mentoring-basierte Teamstruktur dabei helfen, die Barriere zwischen erfahrenen Entwicklern und neuen Mitarbeitern zu verringern und den Wissensaustausch zu fördern.[Fag+14]

Literatur

[AC16] Theodoros Amanatidis und Alexander Chatzigeorgiou. "Studying the evolution of PHP web applications". In: *Information and Software Technology* 72 (2016), S. 48–67. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2015.11.009. URL: https://www.sciencedirect.com/science/article/pii/S0950584915002062.

[Fag+14] Fabian Fagerholm u. a. "The role of mentoring and project characteristics for onboarding in open source software projects". In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ESEM '14. Torino, Italy: Association for Computing Machinery, 2014. ISBN: 9781450327749. DOI: 10.1145/2652524.2652540. URL: https://doi.org/10.1145/2652524.2652540.

[SJ14] Chandershekhar Sharma und S.C. Jain. "Analysis and classification of SQL injection vulnerabilities and attacks on web applications". In: *2014 International Conference on Advances in Engineering Technology Research (ICAETR - 2014)*. 2014, S. 1–6. DOI: 10.1109/ICAETR.2014.7012815.

[2] Synopsys Cybersecurity Research Center (CyRC), "2020 Open Source Security and Risk Analysis Report", https://www.blackduck.com/?cmp=pr-sig

[3] Hibernate: populäres Java-ORM-Framework zur Vereinfachung der Interaktion zwischen Java-Objekten und relationalen Datenbanken. https://hibernate.org/

[4] SQLAlchemy: flexibles ORM-Toolkit für Python, das eine objektorientierte Schnittstelle für relationale Datenbanken bietet. https://www.sqlalchemy.org/