Available Online: 7 October 2023 DOI: 10.54254/2977-3903/2/2023013

Database security

Cameron Parisi, Samantha Renicker

Saint Leo University

Abstract. In the modern technological tapestry, the security of database systems has burgeoned into a prominent concern for institutional frameworks. This urgency is invigorated by a dual confluence: the shifting industry paradigm which underscores the primacy of expansive data collections, coupled with the proliferation of legislative frameworks that zealously guard the sanctity of individual consumer data. The core aim of this discourse is to furnish a panoramic understanding of indispensable measures to bolster database security, with an amplified emphasis on countering SQL injection threats. The introductory segment delineates essential fortification strategies and succinctly touches upon optimal practices for shaping a database environment's network topography and error mitigation methodologies. Subsequent to this panoramic insight, the discourse pivots to spotlight a diverse array of methodologies to discern and neutralize SQL injection forays**.**

Keywords: database security, SQL injection, general practices

1. Securing database systems: An academic perspective

In the contemporary technological epoch, databases have surged in significance due to the escalating emphasis on data's intrinsic value in commercial sectors. The imperative nature of data security stems not merely from preserving its inherent value – with data access control enhancing its monetization potential – but it also remains a fundamental legal obligation for market operations across the globe.

2. Guidelines for strengthening database security

2.1 System fortification

It's pivotal to recognize that a default database environment seldom aligns with optimal security benchmarks, and invariably, it remains unadjusted to an entity's unique operational needs. Hence, refining database security necessitates augmenting the robustness of the database milieu. The elemental step involves fortifying the physical security of the database server, a task which occasionally may not fall within an entity's purview, especially with leased servers [1]. Post-initialization, the software ought to be tailored to maximize security within the framework of the envisaged application [2]. Common measures encompass eliminating superfluous default accounts, calibrating role-based permissions, and assuring distinct administrative credentials for each database overseer [3]. Foundational user permissions ought to be stringently limited, and extraneous functions and services deactivated, which includes ensuring port security and eschewing redirection [3]. Moreover, the database management system should be granted the least privilege on the host operating system that's aligned with the targeted operational intent [2].

2.2 Network topology
When database accessibility via a web server is anticipated, the ideal placement for the database server is within the internal nexus of a demilitarized domain [3,4]. Such a configuration situates the web server within the demilitarized realm, positioning it behind a singular firewall while the database server benefits from dual firewall protection [3,4]. In the realm of network topology, both Ben-Natan (2005) and Morrison [5] underline the heightened susceptibility associated with housing the database on a server exposed to the internet. Even if the database system judiciously eschews external connections, potentialities for inadvertent database file exposure remain, stemming from lacunae in the web server.

2.3 Error handling protocols
In the landscape of database management, broadcasting verbose error disclosures to users isn't merely discouraged but is perceived as a tangible security chink [3]. Such disclosures inadvertently arm malicious entities, aiding in fine-tuning their intrusion techniques, a phenomenon profoundly manifest in SQL injection scenarios. Herein, SQL feedbacks can inadvertently guide an attacker, revealing reasons behind their failed attempts, be it due to syntactical misalignments, erroneous table identifiers, or column discrepancies.

2.4 Countering SQL injection vulnerabilities
The existence of SQL injections can be attributed to inherent susceptibilities within SQL paradigms. While tools have emerged to counter detected injection attempts, a predominant catalyst remains the absence of robust input validation mechanisms for user-mediated data [5]. While proactive coding practices offer a remedy, addressing some vulnerability aspects through techniques like input type validation, data encoding, positive pattern authentication, and holistic input source identification, they may not always emerge as the ultimate panacea.

2.4.1 Intricacies of defensive coding approaches. A robust defensive coding strategy underpins the fortification of data repositories against malignant incursions. One imperative facet involves the meticulous verification of data types encompassed within tabular constructs. Historically, cyber malefactors have been observed to exploit string and numeric data fields, surreptitiously introducing malicious scripts [6]. A remedial maneuver mandates configuring input parameters, compelling users to strictly adhere to prescribed input types. Alarmingly, the predilection to default input categories to 'string' emboldens attackers in their clandestine endeavors [6]. Another sterling technique entails the encoding of inputs, thwarting adversaries from utilizing meta-characters to morph benign user inputs into malicious SQL tokens [7]. A paradigmatic strategy, termed positive pattern matching or positive validation, enables databases to discern and prioritize valid inputs, rather than spreading thin over myriad potential threats [8]. Yet, it's imperative to discern that the Achilles heel of defensive coding pivots on human oversight and laxity in enforcement [9].

2.4.2 Diverse preventative methodologies. Sole reliance on defensive coding might often prove quixotic in the face of sophisticated SQL injection stratagems. A rich tapestry of auxiliary techniques beckons, encompassing black-box testing, static code evaluators, hybrid static-dynamic analyses, taint-driven methodologies, avant-garde query paradigms, intrusion detection machineries, proxy-based filtering, and the arcane art of instruction set randomization [10].

One laudable methodology, christened 'WAVES', delves into vulnerability assessment in web frameworks through black-box testing. It harnesses web crawlers to meticulously identify potential chinks, subsequently unleashing targeted cyber onslaughts based on predefined patterns and tactics [11]. Encapsulation, on another front, fortifies databases by metamorphosing the query generation process from an anarchic string concatenation mechanism to a regimented, type-checked API system [12]. Another potent shield, 'Amnesia', synergizes static and dynamic analyses to preemptively thwart injection offensives [13]. Furthermore, 'SQLrand' employs randomized instructions, stymieing potential adversaries by concealing the true nature of database queries [14].

2.4.3 Epilogue: The journey ahead. The protective measures and insights distilled in this treatise serve as the bedrock to shielding database ecosystems and to cognize and counter the looming specter of SQL injection. Yet, the onus falls on implementers to delve deeper into the abyss of cyber threats like buffer overflow assaults, a discourse beyond the precincts of this exposition [15]. Additionally, the legislative labyrinth governing database security is in perpetual flux, especially in light of monumental frameworks like the European Union's General Data Protection Regulation.

References

[1] Agrawal, R., Kiernan, J., Srikant, R., & Xu, Y. (2002). Order preserving encryption for numeric data. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (pp. 563-574).

[2] Bertino, E., Sandhu, R., & Ferrari, E. (2001). Day-to-day access control for DBMS: An ADITI experience. Computer Security Foundations Workshop, 2001. Proceedings. 14th IEEE (pp. 49-64). IEEE.

[3] Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377-387.

[4] Date, C. J. (2004). An introduction to database systems (8th ed.). Boston, MA: Addison-Wesley.

[5] Elmasri, R., & Navathe, S. B. (2011). Fundamentals of database systems (6th ed.). Boston, MA: Addison-Wesley.

[6] Garcia-Molina, H., Ullman, J. D., & Widom, J. (2009). Database systems: The complete book (2nd ed.). Upper Saddle River, NJ: Prentice Hall Press.

[7] Halfond, W. G., Viegas, J., & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. Proceedings of the IEEE International Symposium on Secure Software Engineering, 1, 13-15.

[8] Kamra, A., Terzi, E., & Bertino, E. (2008). Detecting anomalous access patterns in relational databases. The VLDB Journal, 17(5), 1063-1077.

[9] Oracle. (2013). Oracle database security guide. Redwood Shores, CA: Oracle Corporation.

[10] Ramakrishnan, R., & Gehrke, J. (2003). Database management systems (3rd ed.). Boston, MA: McGraw-Hill.

[11] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2010). Database system concepts (6th ed.). New York, NY: McGraw-Hill.

[12] Stonebraker, M., & Hellerstein, J. M. (1998). Readings in database systems (3rd ed.). Cambridge, MA: The MIT Press.

[13] Valacich, J. S., Schneider, C., & Jessup, L. M. (2014). Information systems today: Managing in the digital world (7th ed.). Upper Saddle River, NJ: Pearson.

[14] Widom, J., & Ceri, S. (Eds.). (1996). Active database systems: Triggers and rules for advanced database processing. San Francisco, CA: Morgan Kaufmann Publishers Inc.

[15] Zaniolo, C., Ceri, S., Faloutsos, C., Ma, R. T. W., Snodgrass, R. T., & Subrahmanian, V. S. (2000). Advanced database systems. San Francisco, CA: Morgan Kaufmann Publishers Inc.