Brian Engel

Database Narrative

Artifact Description: The artifact used to demonstrate databases is an assignment from CS-405: Secure Coding on SQL Injection, completed in summer of 2024. This project uses C++ and imports Sqlite3 to create a database to hold records. The original program:

1. Creates a database and populates it with some records.
2. Runs a query that returns all results.
3. Runs a series of queries that should return one record but adds a simulated injection attack to test if it returns more than the requested record.

Justify the inclusion of the artifact: This project is included in my ePortfolio because it represents a fundamental aspect of computer science and demonstrates my understanding and application of key concepts in database management and security principles. The original artifact contained functions that we were not permitted to modify, which made it challenging to ensure SQL injections were fully mitigated. By revisiting and modifying the project, I plan to redesign its layout and incorporate a logging database. This enhancement will not only prevent SQL injection attacks but also enable me to monitor and document any attempts at injection, further showcasing my commitment to secure coding practices.

Improvements and Enhancements: To improve the artifact I was initially going to try and keep all the functions that were not supposed to be altered the way they were, but I quickly realized that no one who wanted to stop any form of SQL injection would have ever set it up that way. I overhauled the project layout to better isolate and manage database interactions, making it easier to implement and maintain security measures. First, I addressed SQL injection vulnerabilities

more effectively by implementing parameterized queries and prepared statements. This approach prevents malicious inputs from altering the intended queries and improves overall database security. Then I added a logging database to track and document SQL injection attempts. This involved:

- Designing a Logging Schema: Created a new table within the database to record all transactions, including timestamps, queries, and the results of each attempt.
- Implementing Logging Functions: Added code to log every query executed and its outcome, enabling monitoring and analysis of potential injection attempts.
- Integrating Logging into Existing Code: Modified the original query functions to include logging, ensuring that all interactions with the database are recorded.

These improvements significantly improve the system's security by having a robust method of preventing SQL injection and adding a logging system to catch attempts of misusing the system.

One other small change that I made was the inclusion of a menu / loop in the main function for testing purposes (insert, get all, get by name, print transaction log, and print failed transactions log).

Course Outcomes: I had originally only considered these improvements to cover the last outcome - Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources – since they demonstrate anticipating and mitigating security vulnerabilities in software design, but it actually does more than that. The second outcome - Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific

audiences and contexts – is being demonstrated with the code review and narrative provided with the artifact. The third outcome - Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution, while managing the trade-offs involved in design choices - is also demonstrated by redesigning the project layout and implementing parameterized queries and prepared statements and using sound database management practices. This also involved integrating a logging database and the tradeoff of speed and overhead vs security, monitoring and documentation.

Reflection on Process: In enhancing this artifact, I realized the critical importance of designing systems with security in mind from the very beginning. While I have extensive experience with SQL and crafting queries, my familiarity with SQLite was quite limited. I found it enjoyable to dive deeper into SQLite, especially given its widespread use in mobile devices—a field that I am particularly interested in. Implementing parameterized queries and prepared statements was a straightforward decision, aligning with best practices for security. However, integrating a logging database presented a more complex challenge. Despite my hesitation about introducing additional overhead to a project, the advantages of having detailed transaction logs for monitoring and security ultimately justified the decision. I carefully considered what information to include in the transaction log, aiming to strike a balance between providing sufficient detail, ensuring readability, and managing memory usage effectively. This process not only improved the security of the system but also enhanced my skills in working with SQLite and understanding its practical applications.

SQLite. (2024). *SQLite*. Retrieved August 4, 2024, from https://www.sqlite.org/