# Database Security

Cameron Parisi

Samantha Renicker

Saint Leo University

Dr. Omar

COM-450

March 4, 2019

Abstract

Database security is an increasing concern for organizations due to an industry shift in focus to the value of large data sets as well as increasing amounts of legislation concerned with the security of consumers personal information. The objective of this report is to provide an overview of the necessary steps to ensure the security of a database environment as well as provide a more focused look into defending against SQL injection. The overview consists of general hardening and briefly notes best practices for a database environment's network architecture and error response. Following this overview, the report makes focuses on various techniques for detecting and preventing SQL injection.

*Keywords*: Database Security; SQL injection; General Practices

# Database Security

Databases are an ever-growing part commercial technology due to the industry's increasing focus on the value of data. Maintaining the security of this data is not only vital to ensuring its value – as controlling access to data allows for monetization of that data –, but it is also a legal requirement to operate in most of the world's markets.

## General Practices for Ensuring Database Security

### System Hardening

When working with any database environment, the default configuration is almost never sufficiently secure, and it will never be optimized for an organization's exact implementation. As such, database security should include hardening the database environment. The database server should be physically secured (Ben-Natan, 2005); though this may not be handled directly by an organization when using rented servers. Once the database is running, software should be configured for the highest allowable security for the desired implementation (Ben-Natan, 2005). In general, this includes removing all unused default accounts, adjusting role permissions, and ensuring that each database administrator has a separate admin account (Ben-Natan, 2005). The default permissions for basic users should be as restricted as possible and unnecessary services and functions should be disabled (Ben-Natan, 2005); this includes closing ports and avoiding port redirection, whenever possible (Ben-Natan, 2005). The database management service being used should also be given as low-privilege access to the operating system as can be used for the desired implementation (Ben-Natan, 2005).

### Network Architecture

In implementations where the database is meant to be accessed through a web server, the database server should be located within the internal network of a demilitarized zone (Ben-Natan, 2005; Stewart, 2014); In this architecture, the web server is located inside the demilitarized zone, putting the web server behind one firewall and the database server behind two firewalls (Ben-Natan, 2005; Stewart, 2014). When determining a network's architecture, both Ben-Natan (2005) and Morrison (2003) note that hosting the database on the same server as an Internet-facing, web server is leads to unacceptable risk because even if the database service correctly ignores remote connections, there is still risk of critical database files leaking from vulnerabilities in the web server.

### Responding to Errors

When implementing a database server, sending users detailed error messages can be a security vulnerability and is considered bad practice (Ben-Natan, 2005). Attackers can use the error messages to refine their attacks; this is most notable for databases when discussing SQL injection. SQL error messages can let an attacker know if an SQL injection failed due to an invalid table name, improper number of columns – usually relevant when using UNION) –, type mismatch, syntax error, etc.

## SQL Injection Detection and Prevention

SQL injections exist because of vulnerabilities that are found within SQL. However, to make it a little more secure, once these injections are detected, there are tools that can be used to try and stop them. One of the main causes of SQL injection occurrences is due to the lack of input validation for input that the user gives the

database (Halfond, Viegas & Orso, n.d.). Defensive coding can be used to resolve some of these issues, and is one of the best methods; however, it's not always the most ideal or effective methods, so there are also other techniques. Using defensive coding can help eliminate some vulnerabilities to a certain extent, some of these techniques include input type checking, encoding of inputs, positive pattern matching, and identification of all input sources.

**Defensive Coding Techniques**

Developers can check the input type of the elements within tables because attackers like to inject codes into strings and numeric values (Halfond et al., n.d.). By checking the input type boxes, coders can force the user to only input letters or only numbers. Attackers like to take advantage of this because input boxes are typically set to 'string', which makes it easier for them to sneak in their commands (Halfond et al., n.d.). Encoding of inputs can be used to prevent attackers from injecting codes into a string parameter by them using meta-characters because this can turn user input into SQL tokens (Halfond et al., n.d.). Developers can stop these tokens from happening by encoding strings to make the database detect all strings and meta-characters as regular character input (Halfond et al., n.d.). Positive pattern matching, also called positive validation, is a good way for developers to allow the database to search for good input that users submit (Halfond et al., n.d.). Positive validation can be much easier for them to look for valid input rather than only focusing on every attack that could be throw at the database. The sources of all input should be checked by developers to make sure there's nothing injected into the sources because attackers may use it as a way to hide commands. The problem with defensive coding is that even though it can be useful in

preventing injections, developers neglect to enforce these techniques and when they are enforced, human error can fall into play (Halfond et al., n.d.).

**Prevention Techniques**

Since defensive coding cannot be heavily relied on all the time to prevent SQL injections, there are some other helpful techniques that can be used either in place of it or along with it. Some of the other prevention methods are black box testing, static code checkers, combined static and dynamic analysis, taint-based approaches, new query development paradigms, intrusion detection systems, proxy filters, and instruction set randomization (Halfond et al., n.d.).

WAVES is a technique that black-box tests web applications and looks for any possible vulnerabilities that attackers could use for injections (Halfond et al., n.d.). The technique uses a web crawler to identify all points in a web application and then builds attacks that target such points based on a specified list of patterns and attack techniques (Halfond et al., n.d.). Once this is completed, WAVES can monitor how long it takes for the application to respond to the threat and then go from there. Other useful techniques developers use are approaches that use encapsulation to hide certain information in a database from users (Halfond et al., n.d.). Encapsulation makes a database safer by changing the query-building process from an unregulated one that uses string concatenation to a systematic one that uses a type-checked API (Halfond et al., n.d.). This allows developers to change programming patterns when making new queries within a database and keeps the attacker guessing. The only problem with using new query development paradigms is that developers would have to learn a completely new query development process in order to achieve the safety of queries. A

combined static and dynamic analysis technique called Amnesia can be used to prevent injection attacks (Halfond et al., n.d.). Amnesia has a static and a dynamic phase; during static, models are created of all the different legitimate queries that could be made (Halfond et al., n.d.). To test the legitimacy of queries, each one is checked against the created models before being put into the database because Amnesia intercepts all queries (Halfond et al., n.d.). Instruction set randomization, also called SQLrand, allows developers to use random instructions when they create queries (Halfond et al., n.d.). When queries are sent to the database, a proxy filter can be used to intercept them to check that randomized instructions were used on that query because the attacker injecting into a query wouldn't know all the other queries were given random instructions, making it easy to pin point injections (Halfond et al., n.d.).

## Going Further

The techniques and guidelines provided by this paper represent a crucial step to securing a database environment as well as understanding and defending against one of the prevailing threats to databases, SQL injection. Still, implementors will need to go even further to ensure security and understand the various threats a database faces, such as buffer overflow attack – topics which lie outside the scope of this report (Ben-Natan, 2005). Additionally, the legal requirements for database security may rise in the future, if legislation similar to the European Union's General Data Protection Regulation continues to pass.

References

Ben-Natan, R. (2005). *Implementing database security and auditing : A guide for dbas, information security administrators and auditors*. Burlington, MA: Elsevier Digital Press. (2005). Retrieved from https://saintleo.idm.oclc.org/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=187259&site=ehost-live&scope=site

Haldond, W., Viegas, J., & Orso, A. (n.d.). *A Classification of SQL Injection Attacks and Countermeasures*. Retrieved from https://www.cc.gatech.edu/fac/Alex.Orso/papers/halfond.viegas.orso.ISSSE06.pdf

Morrison, P. (2003). Database security. *Network Security, 2003*(6), 11-12. https://doi.org/10.1016/S1353-4858(03)00610-X

Stewart, J. M. (2014) *Network Security, Firewalls and VPNs, 2nd Edition*. [BryteWave]. Retrieved from https://shelf.brytewave.com/#/books/9781284047431/