Theodore Laurita

Cyber Security

4/30/18

Ming Chow

Why is SQL Injection Still A Problem?

Abstract

SQL injection attacks are a form of database attack against which it is impossible to defend in a completely foolproof manner. Furthermore they are rather difficult to identify and preempt against as they do not involve gaining remote control of a system and instead make use of the regular function of a web application or infrastructure. This widespread security vulnerability has only grown as web based applications and infrastructure rely more and more heavily on firewalls or intrusion detection and prevention systems. SQL injection has been an issue that has plagued developers for the better part of two decades, and there is a wealth of research about how to prevent SQL injection. The bigger question one must ask as we close the 2010s, and especially as governments and organizations begin to collect massive amounts of sensitive data about our daily lives, is: why is this basic vulnerability still a problem? The OWASP Top 10 Most Critical Web Application Security Risks lists injection attacks as the number one most serious web application vulnerability in 2017 (*OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks* 6). This paper will analyze why this ancient and basic security vulnerability continues to plague our systems in such an endemic manner and enumerate some of the modern looks about how to begin moving away from this issue.

Introduction

The main question when addressing SQL injection in 2018 is not how to defend against it, as such research has been readily available since the early 2000s to anyone with an internet connection. Instead, the question one should ask oneself is: why is SQL injection still a problem. There are myriad theories as to why, including one presented by Veracode employee Sarah Gibson in 2017.

When analyzing the prevalence of SQL injection in applications over the last five years, according to reports from Veracode, Sarah Gibson found a fairly steady rate of this vulnerability in web applications as a whole. She explains that the above metric relates to web applications with at least one instance of SQL injection vulnerability on the first scan. Therefore, as the above metric related only to non-updated code, she came to the conclusion that SQL injection is still such a prevalent vulnerability because developers are human. She states in her 2017 BSidesLV talk that "everyone poops" and, as we know bugs happen we should assume that even modern developers will produce security vulnerabilities in their first iteration web applications (Gibson).

Other theories as to why SQL injection is still around can be found in David Strom's 2016 blog post on Veracode. He blames the widespread assumption that "Web developers tend to think database queries are coming from a trusted source, namely the database server itself" . Furthermore, he blames the simple prevalence of situations in which SQL injection could be an issue. Strom identifies two cases in which faulty development might lead to a disastrous injection vulnerability: "Places that directly enter database parameters into the URL string itself" or "Fill-in forms on web pages that will take this information and pass it along to the database server via the HTTP POST command" (Strom).

Another idea as to why SQL injection continues to top the OWASP Top 10 lists, even in 2017, is the time to market issue. Dave Lewis writes that on multiple occasions in his own experience, a business leader who has his or her bonus structure tied to the delivery of a web application has opted to bypass security in favor of a quicker time to market. Lewis writes that this problem will plague developers and cost web applications their security so long as the requirement for a security review is not a requirement for business processes, as well as an element of corporate culture (Lewis).

A fourth argument as to why SQL injection is still an issue come from Dan Kuykendall from infosecisland.com who enumerates no less than twelve different reasons for modern injection attacks. His reasons tend to center around a chronic dearth of funds for security teams and the "So many vulns, so little time" against which they are tasked with defending. Kuykendall echoes the idea expressed by Sarah Gibson that penetration testers are only human, and that a system for rechecking applications for security issues after development is finished must be implemented. Furthermore, he expresses his concern that security vulnerabilities such as SQL injection can be identified by automated systems, but many companies opt to avoid the time and capital required to vet an automated system and instead assign penetration testers with "rudimentary technology and limited automation" to the job (Kuykendall).

A fifth and final thesis as to why SQL injection is still an issue decades after its introduction and discovery of prevention techniques is posited by Denny Cherry from IT Knowledge Exchange. He claims that SQL injection remains a problem because of the separation of duties many companies employ. He writes, "[I say this] because the software developers that build the applications never have to deal with the cleanup from the SQL injection attack. Many

developers… see SQL injection as a SQL server problem". However, Cherry sees this as wrong, he sees the responsibilities of software developers and database administrators as necessarily interconnected, as the only way to prevent a SQL injection is to protect data at the application layer with good coding practices (Cherry).

To the Community

I chose this topic because I myself am a application layer software developer who focuses on database design and implementation, but more importantly I am a modern consumer and citizen. My professional interest in this topic, and subsequent horror to see SQL injection at #1 on the 2017 OWASP top 10, is completely eclipsed by my desire to feel safe in the security of my information. As a consumer whose personal information has been leaked by attacks on companies in whom I placed a great deal of trust to safeguard information like my social security number and banking information, I have found myself asking the question "Why is SQL injection still a problem?". I believe that in order to have a society in which companies and governments collect sensitive data about hundreds of millions of users, such a disastrous vulnerability as SQL injection positively needs to be mitigated.

The reasons for the continued prevalence of SQL injection in web application servers, even in 2018, are necessary to know as both software developers but also as citizens of a democratic system of society. In order to be informed enough to make decisions about how and whether our data will be collected by companies and government, and whether we can trust such institutions with such data, we must as a population understand why data breaches continue to be a commonplace news headline. Furthermore, as issues such as Facebook's collection and distribution of personal data become problems we must decide upon as a nation, we must

understand the risks involved in copious amounts of personal data being housed by one organization. These issues of data collection and storage are current, and we must educate ourselves about the risks involved if we are to make informed decisions regarding them.

Action Items

If we are to be a society that collects data about our citizens, we must proceed with securing our information against vulnerabilities such as SQL injection. There are certain action items that can be enumerated on how to go about securing our valuables. Sarah Gibson says in her talk that it is unreasonable to expect that developers will write flawless code on the first try given enough education or information about defense against security vulnerabilities. Instead, she recommends giving developers the change to tools and opportunities to fix the flaws they will inevitably introduce into their code. She says in her aforementioned 2017 talk "Sympathy for the Developer": "But we can see that if you're giving them [developers] the ability to fix them [security flaws] if you're giving them the tools to be able to play and learn, they'll fix flaws" (Gibson: 11:33). David Strom echoes these sentiments expressed by Gibson. He writes, "If security is built into the software development lifecycle, automated test can help find instances of SQLi" (Strom). The assumption is the same: developers will make mistakes. Instead both Gibson and Strom posit that we must accommodate for that fact by allowing for security-focused post production analysis of new web applications.

Conclusion

SQL injection was #1 on the OWASP Top 10 Most Critical Web Application Security Risks in 2017. There is a great wealth of research on how to prevent SQL injection in terms of coding practices, instead we must ask ourselves how such an old security vulnerability can still

be so prevalent. There are many opinions on the problem including a talk by Sarah Gibson at BSidesLV 2017 in which she posited the idea that security vulnerabilities, just like any other type of bug, are an inevitability in code created by human beings. David Strom, in his article on Veracode.com blames the tendency of developers to implicitly trust database queries. Dave Lewis on csoonline.com posits that SQL injection is still around due to a corporate culture that values time to market over sound security. Dan Kuykendall identifies twelve different reasons for the survival of the injection vulnerability including the idea presented by other authors that penetration testers are only human, and are often tasked with the impossible given the resources allocated to them. Finally, Denny Cherry of IT Knowledge Exchange sees the separation of web developers and database administrators as cause for the modern day commonality of such a potentially catastrophic vulnerability. The action item almost all the above authors recommend is allowing for the fact web developers will enter security issues into their code and, instead of ignoring that fact, devoting the necessary resources to account for it. The importance of fixing SQL injections revolves around the fact we live in a society where organizations house immense amounts of sensitive data about citizens, and if we are to make informed decisions regarding that fact, we must understand why some of our most excruciating security problems continue to plague us.

Works Cited

Cherry, Denny. “Why Is SQL Injection Still a Problem?” *SQL Server with Mr. Denny*, 27 Apr. 2015,

Gibson, Sarah. “Sympathy for the Developer” *YouTube,* https://www.youtube.com/watch?v=xhnjD6AW_Cg

itknowledgeexchange.techtarget.com/sql-server/why-is-sql-injection-still-a-problem/.

Kuykendall, Dan. “Why SQL Injection Still Plagues Us.” *Infosec Island*, www.infosecisland.com/blogview/23298-Why-SQL-Injection-Still-Plagues-Us-.html.

Lewis, Dave. “Why Does SQL Injection Still Exist?” *CSO Online*, CSO, 31 July 2015, www.csoonline.com/article/2955300/data-breach/why-does-sql-injection-still-exist.html.

OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks. N.p.: n.p., n.d. PDF. https://www.owasp.org/images/7/72/OWASP_Top_10-2017_(en).pdf.pdf

Strom, David. “Why Is SQL Injection Still Around?” *Veracode*, 26 Jan. 2018, www.veracode.com/blog/2016/04/why-sql-injection-still-around.

Deliverable

https://github.com/woak/CyberSecurityFinalProjectExample

This deliverable demonstrates why SQL injection is still a problem as, even using modern frameworks such as NodeJS and PostgreSQL, today's web application servers and databases are still vulnerable. Unfortunately, the impetus to prevent injection attacks falls completely on the shoulders of the developer as described by Sarah Gibson in her 2017 talk. It is impossible, even for modern systems, to prevent SQL injection if the developer fails to defend against such attacks. I've also attached a Veracode static scan of the above codebase. As one can see, on pages 6-7 of the attached report the scan recommends a fix to a SQL injection vulnerability, even giving the file and line number (server.js:36). This, I think, demonstrates beautifully the action items determined by Sarah Gibson in her talk and David Strom in his article. Allowing for post-production analysis of my deliverable application would have shown me, the developer, the massive SQL injection vulnerability I had introduced into my application server, and even gives me recommendations on how I might fix it.