Sarah Gibson
sgibso03

SQLi Forever: An investigation into SQLi incidence and remediation with static scanning.

Abstract:

The Application Security Industry has become proficient in finding and identifying vulnerabilities in software, while at the same time Web Application attacks are now involved in almost 40% of confirmed breaches (1). We're finding the vulnerabilities but we're failing to fix them fast enough with industries averaging a fix rate of less than 60% (2). While there are many products on the market to mitigate these vulnerabilities, there is a push within the industry to fix what we find. The goal of this paper is to further investigate the rate of incidence of SQL injection flaws in applications tested, and the rate of remediation of those flaws.

Introduction:

In an increasingly digital world software is being used to automate, control, and operate most facets of modern living. From banking, to healthcare, to cars, software runs the world around us. This software contains security flaws, which have been at the root of many headlines in the past several years. As the amount of software increases, a steady flow of security vulnerabilities are being reported to the CVE database with 5288, 5186, 7937, and 6488 vulnerabilities being reported yearly from 2012-2015 (3). In previous years it has been shown that the majority of the errors being reported were introduced due to programming errors (4), and reports from software security vendors have provided more insight into the errors that are being introduced. Yearly reports from Veracode and WhiteHat Security have been confirming the relatively high incidence rate of security vulnerabilities and flaws in commercial software for the last 8 years. With certain flaw types appearing in over 50% of applications scanned statically (5).

With security scanning services gaining traction in the marketplace, and scanning more and more applications written in more languages, it is possible to be able to get a consistent and general view into the likelihood and prevalence of software security vulnerabilities in any deployed software. The large sample of applications being tested across industries is one view into the likelihood of software flaws, which could lead to security incidents if shown to be exploitable vulnerabilities.

When discussing findings from static analysis of source code, or executables, those findings are considered flaws or weaknesses within the design and/or implementation of the software. These weaknesses are patterns within the software that could allow for loss of confidentiality, integrity, or availability of the application, should they be exploitable. If a weakness within an application is found to be potentially exploitable, it would then be considered a vulnerability. Static analysis does not generally have the ability to determine if a weakness could lead to a vulnerability, and so findings tend to be higher in number than techniques that exercise the application at runtime. This is demonstrated by the instance rate of SQL injection found in static analysis in respect to the same rate found in dynamic analysis. We see about 1/5 the

number of vulnerabilities in dynamic analysis, as we see flaws found through static analysis with this flaw type (5).

While SQL injection numbers differ between static and dynamic analysis, they do not appear to differ very much over time within a given analysis type. The percent of applications found to have SQL injection through static analysis has been reported around 30% in the Veracode State of Software Security for the last 6 years. A recent report on Sql injection flaws within PHP code snippets posted on Stack Overflow, found that the rate of flaws per questions asked had held steady around 45% since 2008 (6). This would indicate a near constant rate of SQL injection weaknesses introduced in code independent to the availability of prepared statements. While PHP had prepared statements introduced in 2004, the MySQL library that did not support them was not deprecated until 2013 (7). Within the php code snippet data, there does not appear to be a trend downwards either from the beginning of the sample period, or after 2013.

To the community:

As more data becomes available from scanning services and other sources, it is becoming clear that security flaws in software occur even when preventative measures are available. While it may be possible to lower the rate of introduction of flaws over time, just having preventative measures be available does not appear to make a significant difference in the rate the flaws occur. The report on SQL injection flaws in PHP questions on Stack Overflow, is an example of this. The rate of introduction of new flaws does not appear to be affected by changes in the language overtime that allow for protections from the flaw.

Veracode in their yearly State of Software Security report have been reporting on the percentage of applications containing one or more SQL injection flaws for the past six years. The most recent report, released in October, reports that of all applications scanned for the first time during the reporting period, 35% of them contained a SQLi flaw (5). A previous version of the report, which covered scans in 2011, found an incident rate of 32% within just web applications (8). The most recent reported rate is for all applications scanned, not just those flagged as web applications. While there is a difference in sampling between the reports, it is clear that SQL injection incidence in static scanning has remained relatively high, and may have increased.

Reports on Dynamic scanning over the past 6 years have also shown that SQL injection remains a fairly prominent vulnerability, with the WhiteHat yearly report indicating that around 6% of all applications tested in 2014 contained the issue (10). The rate was reported at 7% for 2012 and 11% for 2011 (11). This slow fall in vulnerability detection is in contrast to a fairly steady rate of detection of flaws in static analysis. This may be a reflection of an increase in effectiveness of security tools to mitigate flaws, or a trend for a subset of applications to move away from SQL to non-relational databases. None the less, there remains a fairly steady 6-7% of applications tested dynamically that are found to be vulnerable to SQL injection.

Sarah Gibson
sgibso03

While the rate of detection of this flaw in newly scanned applications remains high, despite the availability of methods and tools that can eliminate the flaw, there is some hope that more secure applications are possible. As part of the research for this paper, there was further investigation into remediation trends for applications as they are scanned with Veracode, both statically and dynamically using Veracode's database of anonymized scanning data.

Applications:

Automated security testing has shown that it is able to consistently find security flaws within the code of a given program, as well as the ability to exploit a certain percentage of these flaws when exercising the run time environment. This investigation is primarily concerned with findings related to SQL injection flaws and vulnerabilities. As previously discussed this flaw type has an accepted remediation that prevents static analysis tools from flagging it, and dynamic analysis from finding it. When prepared statements are used properly the use of tainted data is considered safe. If prepared statements are not used, generally speaking validating or cleansing the tainted data before using it in the query is not enough to guarantee a prevention of SQL injection.

Prepared statements have been available in Java, .NET and PHP throughout the sample period of the Veracode, WhiteHat, and Stack Overflow reports. If adoption of these methods was increasing, it would be reasonable to expect that all three reports would have seen a marked decrease in flaw and vulnerability detection. This is not what we are seeing. The only report that has demonstrated a decrease is the dynamic report from WhiteHat. This report, based on the methodology information provided in the report, is not just for first scans, and may incorporate applications that have scanned already, and fixed their flaws based on that initial scan. This is the other reason why the White Hat report may show a decrease in incidence rate of SQL injection over the years.

First scans of applications show a fairly consistent rate of SQL injection flaws across the past 6 years, however, detailed rescan information does show consistent trend of remediation on a flaw by flaw basis [Figure 1]. While developers are introducing the flaw at a regular rate, once they have tested their application, any given flaw has an almost 40% chance of being fixed by the third scan of that application. WhiteHat has reported a higher rate of fix for their dynamic scans. The rate of fix reported in the 2015 report for SQLi found through their dynamic scanning is 52%. Dynamic scans provide proof of exploitation, where a static flaw may not actually be vulnerable to exploitation and may have compensating controls in place instead of requiring a code fix.

The other graph prepared for this report is remediation by application for static scanning. This is not as encouraging as the flaw by flaw view [Figure 2]. While a given flaw is almost 40% likely to be fixed, the same is not true for applications as a whole. There's an effort to reduce the flaws, but by the third scan we are not seeing a significant effort in eradicating flaws within a given application.

Sarah Gibson
sgibso03

Conclusion:

As application security testing becomes more common, more data on the base prevalence of flaws such as SQL injection becomes available. This data reveals a consistent rate of incidence of these flaws upon first scanning. This consistency can also be found through other data sources, such as mining Stack Overflow questions and running static analysis on them. The consistency across scanning methodologies and time is an indication that security flaws are being introduced at a near constant rate, independent of the tools and techniques for remediating them.

While the rate of introduction does not appear to be changing, based on the data at hand, there does appear to be a marked reduction in flaws once development teams test the applications, and receive the results. This feedback on their application, and detailed information as to where the issue is occurring, are allowing developers to fix a portion of those flaws fairly quickly. Going forward, as more companies become aware of the security of their product, testing for those security flaws is an immediate way to help with addressing those issues. Long term, SQL injection flaws may start to drop in number if developer education starts to regularly include information on how the flaws works and what prevents it.

Sarah Gibson
sgibso03

Figure 1.

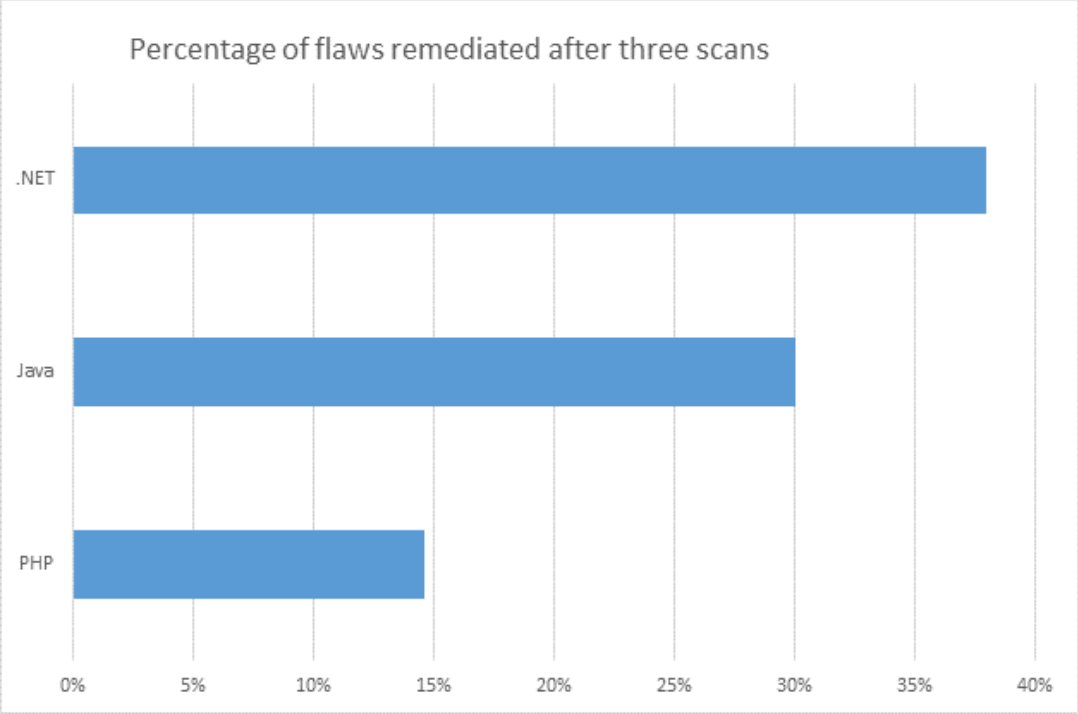
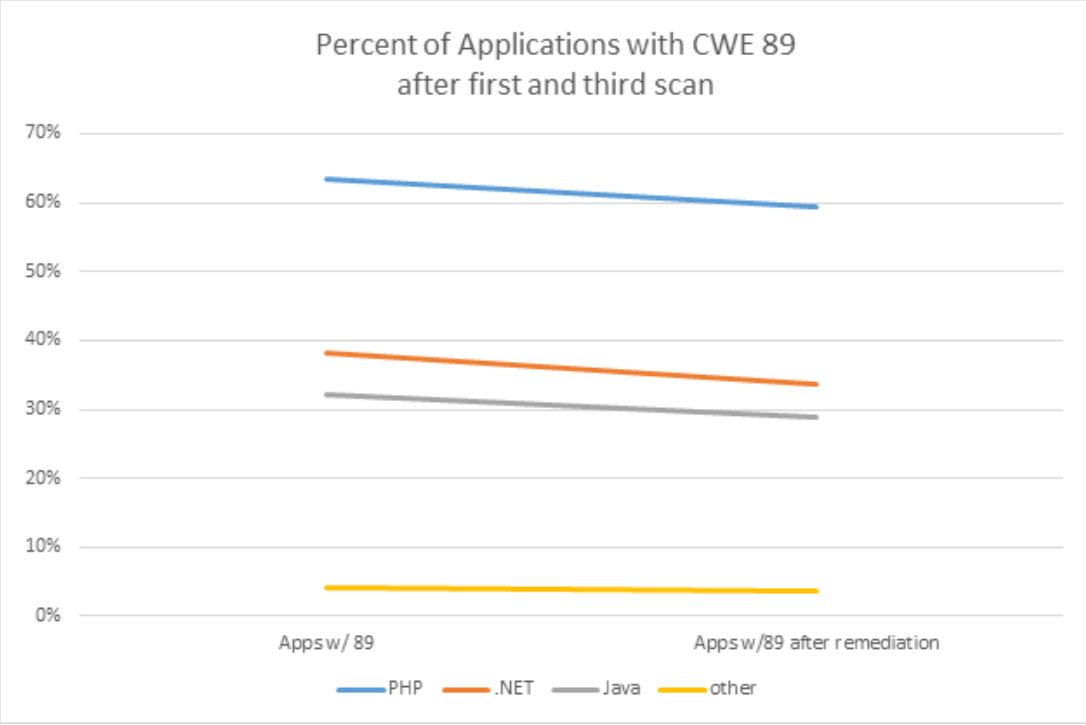


Figure 2.



Sarah Gibson
sgibso03

References:

1. “2015 Data Breach Investigations Report,” Verizon, April 2015, p. 24
2. “2015 State of Software Security,” Veracode, June 2015, p. 8
3. https://web.nvd.nist.gov/view/vuln/statistics
4. Jon Heffley and Pascal Meunier, “Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security?” in Proc. 37th Annual Hawaii International Conference on System Sciences (HICSS-04), 5-8 January 2004, Track 9, Volume 9, IEEE Computer Society, 2004, pp. 1-10, https://doi.org/10.1109/HICSS.2004.1265654.
5. “State of Software Security: Volume 7,” Veracode, October 2016, p. 10
6. https://laurent22.github.io/so-injections/
7. http://php.net/archive/2013.php
8. “State of Software Security: Volume 5,” Veracode, April 2013, p. 30
9. “Website Security Statistics Report 2015,” WhiteHat Security, May 2015, p. 6
10. “Website Security Statistics Report 2013,” WhiteHat Security, May 2013, p. 7