Correlating Automated Static Analysis Alert Density to Reported Vulnerabilities in Sendmail

Michael Gegick and Laurie Williams
Department of Computer Science, North Carolina State University
{mcgegick, lawilli3}@ncsu.edu

Extensive reliability-based research including [2, 8, 10] has shown that software metrics can be used to identify fault- and failure-prone components early in the software process. A fault is an incorrect step, process, or data definition in a computer program [5] whereas a failure is the inability of a software system or component to perform its required function [5]. Our research objective is to identify vulnerability- and attack-prone components as security parallels of reliability-based fault- and failure-prone components. A vulnerability is "an instance of a [fault] in the specification, development, or configuration of software such that its execution can violate the [implicit or explicit] security policy [6]. A vulnerability, like a fault, may lie latent until it is revealed by a tester or by an attacker with malicious intent. An attack occurs when an attacker has a motive or reason to attack and takes advantage of a vulnerability to threaten an asset [4]. From these definitions we propose the following:

- A component is *vulnerability-prone* if it has a high probability that internal verification and validation (V&V) efforts will find security vulnerabilities prior to release [3].
- A component is *attack-prone* if it has a high probability that security vulnerabilities will be exploited after release [3].

Recent research [9, 12] has shown that alerts generated by automated static analyses (ASA) of source code are good indicators of fault-prone components. We define an alert as an issue raised by the ASA tool that may or may not be a true problem in the software. Alert density is the number of alerts per line of code. We are currently investigating if alert density is a good metric for vulnerability- and attack-prone component identification. In general, ASA tools are apt at finding coding problems, but cannot directly find more complex and abstract problems related to the design and operation of a software system. Design and operation problems may require testing or architectural risk analyses for identification. However, the work by Nagappan et al. [9] and Zheng et al. [12] indicate that ASA alerts are good indicators of fault-prone components despite the limitations with ASA tools.

In practice, vulnerabilities are distinguished into three types: implementation bugs, operational vulnerabilities, and design flaws [1, 7, 11]. An implementation bug is a problem at the code level, such as buffer overflows, not checking return codes, and unsecured SQL statements [7]. Operational vulnerabilities are vulnerabilities that arise from the environment in which a software application resides [1]. A design flaw is a vulnerability that is related to the design of the system and can occur even if the program is well-coded [7, 11]. Examples of design flaws include a lack of or incorrect auditing/logging, ordering and timing faults, and improper authentication [7]. An objective and repeatable method does not exist for determining which vulnerabilities belong to a given type.

For a feasibility study, we scanned ten releases of Sendmail[1] (releases 8.12.2-8.12.11) with Fortify Software's Source Code Analyzer (SCA) version 4.0. We used Poisson regression to model the association between number of vulnerabilities reported in the Sendmail RELEASE_NOTES[2] for a given file and the alert density for that file. This Poisson regression is a generalized linear model in which the number of vulnerability reports follows a Poisson distribution with mean which is a linear function of alert density, after log transformation. Twenty-one vulnerabilities or weaknesses that could lead to vulnerabilities had been reported in the Sendmail RELEASE_NOTES. Some of these vulnerabilities were present in multiple releases thus increasing the count of vulnerabilities to 61. We classified 35 of the vulnerabilities as implementation bugs, and 26 as operational vulnerabilities. We did not classify any of the vulnerabilities as design flaws. The files scanned by SCA over the ten releases were aggregated to provide for a larger data set than a data set that only contained the number of files and reported vulnerabilities for a single release of Sendmail. The total number of files scanned by SCA was 996.

Table 1. Initial Sendmail analysis[3].

Reported vulnerability	Slope	p-value	Chi-Square/df Goodness-of-fit measure	Standard error
Implementation bugs	352.7990	<.0001	1.1537	63.9536
Operational vulnerabilities	-207.888	0.4604	1.5773	281.6125
Implementation bugs and operational vulnerabilities	247.8114	<.0001	1.4217	61.0546

Table 1 summarizes our analysis based on the Poisson regression model. The positive slope estimate indicates that a positive association between alert density and reported vulnerabilities was observed. Specifically, a slope of 353 represents the estimated increase in the expected number of vulnerability reports on the log scale per unit increase in the alert density. While the small p-value associated with this estimate provides highly significant evidence of an association, the rather large standard error (64) indicates substantial uncertainty in the magnitude of the effect, based on this preliminary work. The Poisson model appeared to

[1] http://www.sendmail.org
[2] http://www.sendmail.org/ftp/RELEASE_NOTES
[3] Results in Table 1 generated by SAS 9.1.

provide a reasonable fit, as measured by the chi-squared goodness-of-fit test statistic. The expectation of this statistic under a correctly specified model is equal to its associated degrees of freedom (df), so that ratios far from one indicate lack-of-fit. The goodness-of-fit column indicates that the model predicting implementation bugs fits reasonably well, but there is substantial over dispersion in the predictions based on the other two vulnerability report classifications.

There was only one instance where an implementation bug and an operational vulnerability were located in the same file. A larger number of reported vulnerabilities would likely decrease the large value of the standard error.

We have not completed our analysis on correlating alert density to the vulnerabilities that have known exploits which may allow us to determine which files in Sendmail are attack-prone. By our definition of vulnerability-prone components, we have not identified vulnerability-prone components (files) in Sendmail because the vulnerability reports were issued after each release of Sendmail. However, we have shown that alert density of a file can be correlated to vulnerabilities reported in that file. Future analyses on larger software systems will involve gathering vulnerability reports issued before release to determine if ASA alert density can identify vulnerabilities identified by internal V&V efforts. An early knowledge of high-risk security components can afford software engineers to make informed risk management decisions and prioritize redesign, inspection, and testing efforts. Validating the alert density to reported vulnerability correlation on different software may supply evidence that the correlation is similar for other software systems.

Acknowledgment
Thanks to Dr. Jason Osborne (NCSU) for his helpful suggestions with our statistical analysis.

References

[1] M. Dowd, J. McDonald, and J. Schuh, *The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities*. Boston, MA: Addison-Wesley, 2007.
[2] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Transactions on Software Engineering*, vol. 27, pp. 630-650, July.
[3] M. Gegick and L. Williams, "Toward the Use of Static Analysis Alerts for Early Indication of Vulnerability- and Attack-prone Components," *To appear in the Systems Vulnerabilities (SYVUL) Workshop*, July 1-6 2007.
[4] M. Howard and D. LeBlanc, *Writing Secure Code*, 2nd ed. Redmond: Microsoft Corporation, 2003.
[5] IEEE, "ANSI/IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)," Los Alamitos, CA: IEEE Computer Society Press, 1990.
[6] I. Krsul "Software Vulnerability Analysis" PhD in Computer Science, West Lafayette, Purdue University, 1998.
[7] G. McGraw, *Software Security: Building Security In*. Boston: Addison-Wesley, 2006.
[8] J. Munson and T. Khoshgoftaar, "The Detection of Fault-Prone Programs," *IEEE Transactions on Software Engineering*, vol. 18, pp. 423-433,
[9] N. Nagappan and T. Ball, "Static Analysis Tools as Early Indicators of Pre-release Defect Density," in *International Conference on Software Engineering*, St. Louis, MO, 2005, pp. 580-586
[10] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Where the bugs are," in *International Symposium on Software Testing and Analysis*, Boston, Massachusetts, 2004, pp. 86-96
[11] C. Wysopal, L. Nelson, D. Dai Zovi, and E. Dustin, *The Art of Software Security Testing: Identifying Software Security Flaws*. Boston: Addison Wesley, 2006.
[12] J. Zheng, L. Williams, W. Snipes, N. Nagappan, J. Hudepohl, and M. Vouk, "On the Value of Static Analysis Tools for Fault Detection," *IEEE Transactions on Software Engineering*, vol. 32, pp. 240-253, April 2006.