It Doesn't Have to Be Like This: Cybersecurity Vulnerability Trends

D. Richard Kuhn[1], M S Raunak[2], Raghu Kacker[1]
kuhn@nist.gov, raghu.kacker@nist.gov raunak@loyola.edu
[1]National Institute of Standards and Technology [2]Loyola University of Maryland

It often seems that every newly announced major data breach sets a record for the depth and size of impact. Internet users, nearly everyone these days, naturally wonder: Why is this happening, and how much worse can it get? In the inaugural article for this column, published in January 2009, we reviewed trends in vulnerabilities for the previous eight years [2]. Our goal, then as well as now, is to improve the understanding of cybersecurity vulnerabilities so that we can prevent them. One Moore's Law generation later, we followed that article with another review of trends, finding some encouraging results [3]. In this article, we review some of those earlier findings, plus what has happened since then, and prospects for the near future.

Our data source is the US National Vulnerability Database (NVD) [1], which collects nearly all publicly reported vulnerabilities since 1997, using the Common Vulnerabilities and Exposures (CVE) dictionary. It is developed and run by the US National Institute of Standards and Technology, with support from the Department of Homeland Security's National Cyber Security Division. As of 2017, the NVD includes more than 85,000 vulnerabilities, and the collection is expanded daily. With two decades of data, the NVD is an invaluable resource for security analysts.

One of the primary observations from the January 2009 analysis was that the total number of vulnerabilities per year had begun to decline, from a peak of nearly 7,000 in 2006 to about 5,500 in 2008. It appeared that developers and security administrators had begun taking security seriously, including it as a key component in development, and staying up to date on mitigation techniques. Code flaws that were widely used in system exploits in the 1980s and 1990s, such as format string vulnerabilities and race conditions, were appearing in only a dozen or two cases each year, accounting for less than 1% each of the vulnerabilities in thousands of applications. Better development methods and tools had begun to make a difference.

But the 2009 analysis also revealed a trend that we see repeatedly in all aspects of security - new information technology produces new challenges to secure it. During the previous decade, e-commerce and other web-based services had proliferated, producing new challenges for protection and new opportunities for attackers. While buffer overflows and misconfigurations had long been the main sources of weaknesses in systems defenses, SQL injections and cross-site scripting were respectively the #1 and #2 vulnerability types in 2008 (Fig. 1). (Note that the analysis is limited to the distribution of primary vulnerability categories; another 10% - 15% each year are classed as either "other" or "insufficient information".) As we will see later in this article, the trends for these two vulnerability types illustrate an important lesson for managing cybersecurity.

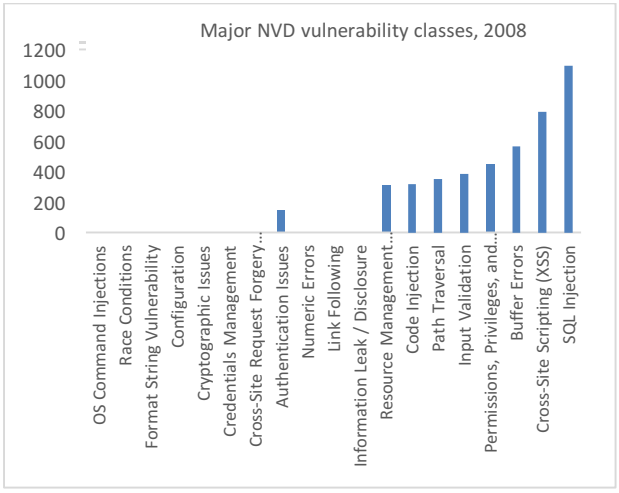

Fig. 1. web related vulnerabilities were common in 2008.

A follow-on review added data from 2009-2010 [3], providing more in-depth analysis, and showing that vulnerabilities continued to decline as they had since 2006. Among the interesting findings from this analysis was that the average difficulty of exploitation began to change in 2006. Prior to this time, nearly all vulnerabilities had been easy to exploit, but after this time, the access complexity of about half of vulnerabilities was either medium or high. This finding suggests that defensive measures in code and system administration were being successfully employed.

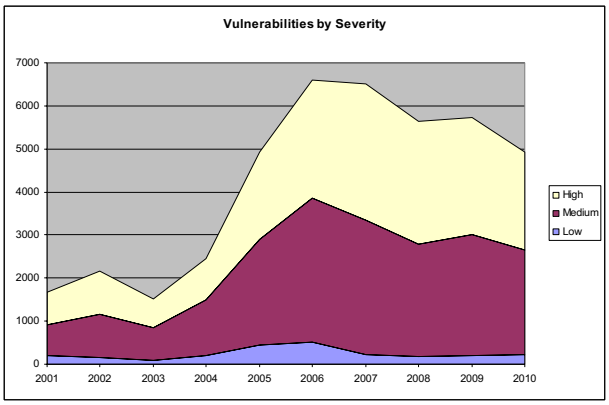

Fig 2. Vulnerabilities declined 2006-2010 but about 96% were medium to high severity.

Among negative findings in that study, it was found that the proportion of high, medium, and low severity vulnerabilities had changed little over the period 2001-2010. That is, serious errors were just as common in 2010 as they had been a decade earlier. Additionally, buffer errors were still one of the major sources of system vulnerabilities, and we reported on a separate analysis that found that roughly 93% of these involved only a single condition (typically failure to check array bounds; a few buffer errors required two or more conditions to be true to exploit). We pointed out that even the most basic of secure programming practices, such as ensuring checks of all input string length, could eliminate a large proportion of these problems.

More recently, we revisited the review of NVD data through 2016 [4], and found that medium to high severity vulnerabilities had declined slightly, from 96% in 2008 to about 90% for 2016 (Fig. 3).

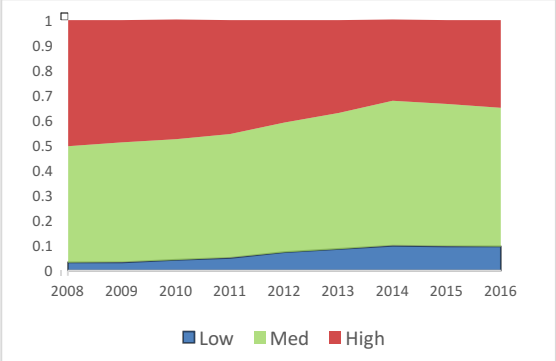

Fig. 3. Vulnerability Severity Trends, 2008-2016

This review also included an additional type of analysis. Not all security-critical errors in software are specifically related to security. For example, buffer overflow errors usually result from failing to check that input is the appropriate size for internal storage, a check that should always be done and may result in ordinary failures that are not necessarily security-relevant. How prevalent are ordinary coding errors like these among the vulnerabilities cataloged in the NVD?

To address this question, we can distinguish at least three types of errors: ordinary coding or implementation errors, administrative and configuration errors, and fundamental design problems:

- Configuration vulnerabilities result from bad configuration files or other administrative errors. One example is missing password checks. Information leaks also frequently result from failing to set up controls, or apply updates.
- Design-related vulnerabilities - which originate in the planning and design of the system, such as selecting an outdated or weak cryptographic algorithm.
- Implementation vulnerabilities are errors in code, such as the buffer overflow example mentioned previously. Cross-site scripting is less obvious, but generally results from missing or inadequate input validation, and other forms of input validation failures are common.

Table II designates Configuration, Design, and Implementation errors as C, D, and I respectively. Note that Table II also indicates whether the different vulnerability types are increasing (↑), decreasing (↓), or approximately unchanged (≈).

As shown in Fig. 3, implementation errors are by far the major source of vulnerabilities, accounting for roughly two-thirds of the total. Note that the number of vulnerabilities is related to the number of applications released, and new applications are released constantly, so it is important to consider the proportion rather than counts of vulnerability types. Remarkably, the proportion of implementation vulnerabilities for 2008 to 2016 is very close to the 64% reported for 1998 to 2003 in another analysis [5]. This is somewhat surprising and discouraging, given that these vulnerabilities result from simple mistakes which should be easy to prevent. However, this finding also suggests the potential for relatively low-cost improvements. Static analysis tools can detect about 20% of CVE-defined errors [8] and formal code inspection has been demonstrated to be highly effective in error reduction [7]. The key point of this analysis is that a very large proportion of security vulnerabilities arise from basic coding errors, which can be prevented and detected with a comprehensive program of static analysis and dynamic test methods.

TABLE II. NVD VULNERABILITY CATEGORIES
(C=CONFIGURATION, D=DESIGN, I=IMPLEMENTATION)

CWE-ID	Description	Type	Trend
CWE-16	Configuration	C	↓
CWE-20	Input Validation	I	↑
CWE-22	Path Traversal	I	↓
CWE-59	Link Following	I	≈
CWE-78	OS Command Injections	I	↑
CWE-79	Cross-Site Scripting (XSS)	I	≈
CWE-89	SQL Injection	I	↓
CWE-94	Code Injection	I	↓
CWE-119	Buffer Errors	I	↑
CWE-134	Format String Vulnerability	I	≈
CWE-189	Numeric Errors	I	↓
CWE-200	Information Leak / Disclosure	C	↑
CWE-255	Credentials Management	D	↑
CWE-264	Permissions, Privileges, Access	D	↑
CWE-287	Authentication Issues	D	≈
CWE-310	Cryptographic Issues	D	↑
CWE-352	Cross-Site Request Forgery	I	≈
CWE-362	Race Conditions	I	↑
CWE-399	Resource Management Errors	I	↓

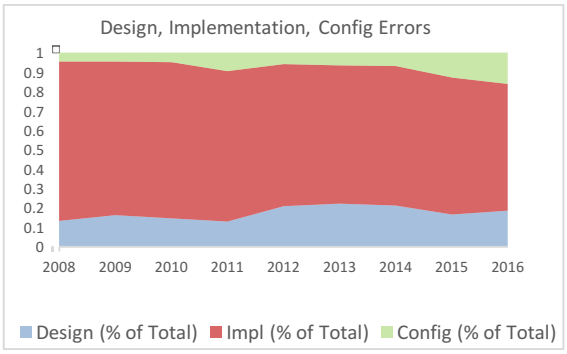

Fig. 3. Vulnerability Class Trends, 2008-2016

As noted previously, SQL injection vulnerabilities were the #1 most common type in 2008. By 2015, vulnerabilities of this type had been dramatically reduced (Fig. 4). Better tools and improved development practices helped prevent this type of implementation error, and can do so for the other types as well. As suggested in the title of this article, we can reduce cybersecurity vulnerabilities, using tools and methods that are readily available but must be applied.

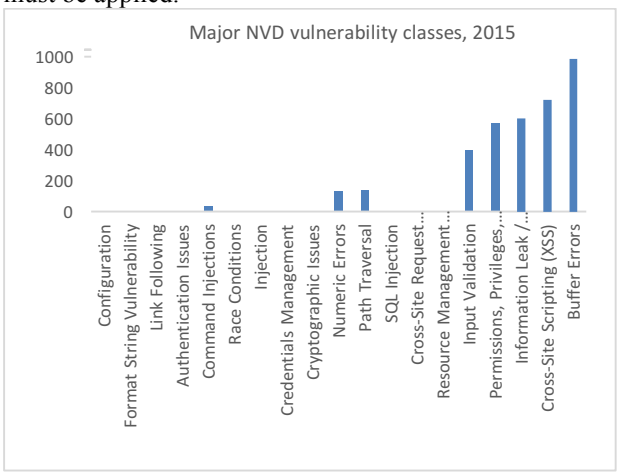

Fig. 4. Distribution of vulnerabilities changed significantly from 2008 - 2015

Products may be identified in this document, but such identification does not imply recommendation by the US National Institute of Standards and Technology or the US Government, nor that the products identified are necessarily the best available for the purpose.

References

[1] National Vulnerability Database, http://nvd.nist.gov 2017

[2] Kuhn, R., Rossman, H., & Liu, S. (2009). Introducing" Insecure IT". *IT professional*, *11*(1), 24-26.

[3] Kuhn, R., & Johnson, C. (2010). Vulnerability trends: measuring progress. *IT professional*, *12*(4), 51-53.

[4] Kuhn, D. R., Raunak, M. S., & Kacker, R. (2017, July). An Analysis of Vulnerability Trends, 2008-2016. In *Software Quality, Reliability and Security Companion (QRS-C), 2017 IEEE International Conference on* (pp. 587-588). IEEE.

[5] Heffley, Jon, and Pascal Meunier. "Can source code auditing software identify common vulnerabilities and be used to evaluate software security?" *System Sciences, 37th Annual Hawaii Intl Conf*, IEEE, 2004.

[6] Okun, Vadim, Aurelien Delaitre, and Paul E. Black. "Report on the static analysis tool exposition (SATE) IV" *NIST Special Publication* 500 (2013): 297.

[7] Jones, C, "Measuring Defect Potentials and Defect Removal Efficiency", *Crosstalk, Journal of Defense Software Engineering* (June 2008).

[8] Medeiros, I., Neves, N. and Correia, M., 2016. Detecting and removing web application vulnerabilities with static analysis and data mining. *IEEE Trans. Reliability*, *65*(1), pp.54-69.

[9] Balachandran, V., 2013, May. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Software Engineering (ICSE), 2013 35th International Conference on* (pp. 931-940). IEEE.