Proceedings of the 51st Hawaii International Conference on System Sciences | 2018

Estimating Software Vulnerability Counts in the Context of Cyber Risk Assessments

Thomas Llansó, thomas.llanso@jhuapl.edu, thllanso@dsu.edu
Martha McNeil, martha.mcneil@jhuapl.edu , martha.mcneil@dsu.edu
Johns Hopkins University Applied Physics Laboratory
Dakota State University

***Abstract*—Stakeholders often conduct cyber risk assessments as a first step towards understanding and managing their risks due to cyber use. Many risk assessment methods in use today include some form of vulnerability analysis. Building on prior research and combining data from several sources, this paper develops and applies a metric to estimate the proportion of latent vulnerabilities to total vulnerabilities in a software system and applies the metric to five scenarios involving software on the scale of operating systems. The findings suggest caution in interpreting the results of cyber risk methodologies that depend on enumerating known software vulnerabilities because the number of unknown vulnerabilities in large-scale software tends to exceed known vulnerabilities.**

***Index Terms*—cyber, risk, vulnerability, estimation, flaw rate, discovery rate**

I. Introduction

Organizations that employ cyber systems to help meet business/mission objectives are often concerned about the degree to which cyber attacks can put those objectives at risk. The motivation for the concern is clear. By nearly any measure, the magnitude of the problem is staggering. As an illustration, Cybersecurity Ventures estimates that cyber crime will cost the world $6 trillion annually by 2021 and that $1 trillion will be spent globally on cybersecurity between 2017 and 2021 [1].

Stakeholders often conduct risk assessments as a first step towards understanding and managing mission risks due to cyber effects, such as cyber attacks. To one degree or another, risk assessment methodologies focus on enumerating cyber-related vulnerabilities as a means to assess cyber risk. NIST defines vulnerability as *a weakness in an information system, system security procedures, internal controls, or implementation that could be exploited by a threat source* [2].

Vulnerabilities can be found in people, processes, and technologies. This paper focuses on vulnerabilities in technology, specifically software on the scale of operating systems (OSs). With major OSs now exceeding 50 million lines of code [3], the issue of vulnerabilities in software remains a key concern, despite raised awareness. As the Hewlett Packard 2015 Cyber Risk Report [4] states:

> *Much has been written to guide software developers on how to integrate secure coding best practices into their daily development work. Despite all of this knowledge, we continue to see old and new vulnerabilities in software that attackers swiftly exploit.*

Software vulnerabilities manifest in many forms, including race conditions, buffer overflows, integer overflows, dangling pointers, poor input validation (e.g., SQL injection, cross-site scripting), information leakage, violation of least privilege and other access control errors, use of weak random numbers in cryptography, protocol errors, and insufficient authentication [5][6]. As cataloged in the National Vulnerability Database [7], prominent recent examples of serious software vulnerabilities include Heartbleed (Common Vulnerabilities and Exposures (CVE) ID CVE-2014-016), Shellshock (CVE-2014-6271), the glibc buffer overflow (CVE-2015-7547), VENOM (CVE-2015-3456), and Microsoft Malware Protection Engine vulnerability (CVE-2017-0290) [7]. According to the website CVE Details [8], there have been 937 vulnerabilities between January and August 2017 to date. Ten percent of those vulnerabilities were in the "critical" (highest severity) category with assigned Common Vulnerability Scoring System (CVSS) scores of 9 or 10 [9].

This paper considers the following question: Given that risk assessment methodologies in use today depend in large part on an understanding of vulnerabilities, and software vulnerabilities in particular, to what degree can we hope to be complete in an enumeration of software vulnerabilities in a target cyber environment?

Our hypothesis is that currently the community can only enumerate a minority of software vulnerabilities present in large software systems. To explore the hypothesis, we develop a metric that estimates the proportion of vulnerabilities that are latent or unknown at any given time in a software system, which we can then use as a means to test our hypothesis. By unknown, we mean unknown to the creators of the software system. Note that we do not require great precision in our metric, as we only seek to establish that unknown software vulnerabilities tend to exceed known vulnerabilities within OS-scale software systems.

The rest of this paper discusses related work, the estimation approach, and an analysis of the approach in the context of OS-scale systems. Conclusions and future work follow.

URL: http://hdl.handle.net/10125/50376
ISBN: 978-0-9981331-1-9

II. Related Work

There are many published cyber risk assessment methodologies in use today, e.g.:

- United States National Institute of Standards and Technology (NIST) Special Publication 800-30: Guide for Conducting Risk Assessments [2],
- Carnegie Mellon Software Engineering Institute Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE) [10],
- MITRE Threat Assessment and Remediation Analysis Methodology (TARA) [11],
- ISACA Control Objectives for Information and Related Technologies (COBIT) [12],
- Johns Hopkins Applied Physics Laboratory Mission Information Risk Analysis (MIRA) [13],
- Air Force Research Laboratory Cyber Risk Assessment in Distributed Information Systems [14].

As an example methodology, the NIST guide presents an approach to risk assessment summarized in Figure 1. As the figure shows, the process involves four major steps. The actual assessment occurs in step two, which has several substeps. The second substep pertains to identifying vulnerabilities and predisposing conditions.

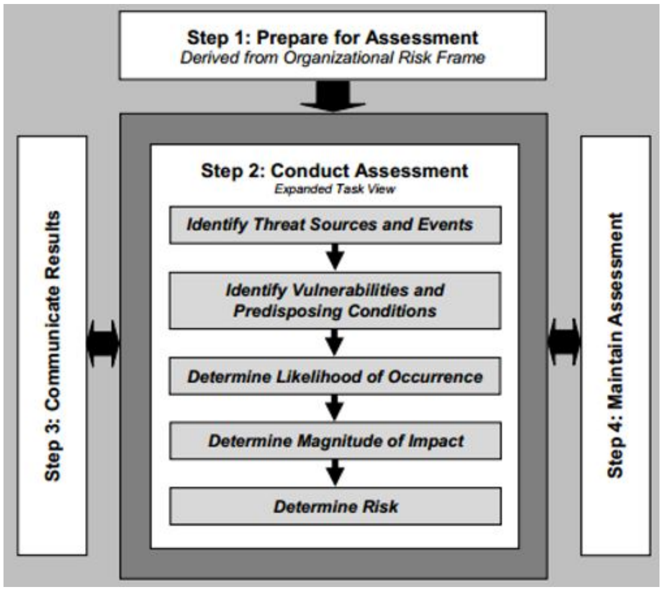

Step 1: Prepare for Assessment
Derived from Organizational Risk Frame
Step 2: Conduct Assessment
Expanded Task View
Identify Threat Sources and Events
Identify Vulnerabilities and Predisposing Conditions
Determine Likelihood of Occurrence
Determine Magnitude of Impact
Determine Risk
Step 3: Communicate Results
Step 4: Maintain Assessment

Fig. 1: NIST 800-30 Process Summary

Another example methodology, OCTAVE, contains phases and steps within phases that pertain to vulnerabilities of different types in Table I. In OCTAVE, vulnerabilities are addressed in phase two, step two.

The study of cyber-related vulnerability prediction is rich and varied. A sampling of prior research related to the topic of this paper follows.

McQueen, et al. [15] estimate the number of zero day vulnerabilities in software. The authors' definition of zero day requires that the vulnerability has already been discovered by some agent but is not yet publicly disclosed. This approach estimates the number of vulnerabilities disclosed to a vendor but not yet made public on any given day as well as the average lifespan of a zero day vulnerability. Jones [16] estimates a related metric, namely the number of vulnerabilities made public but for which no vendor patch is yet available. This paper also estimates unknown vulnerabilities but without constraints related to vendor notification.

Phase	Output
One	Critical Assets Security Requirement for Critical Assets Threats to Critical Assets Current Security Practices
Two	Key Components Current Technology Vulnerabilities
Three	Risks to Critical Assets Risk Measures Protection Strategy Risk Mitigation Plans

TABLE I: OCTAVE Outputs

A number of researchers considered vulnerability discovery modeling. Zhang, Caragea, and Ou [17] make use of data from the National Vulnerability Database (NVD) maintained by NIST [7] applying a data mining approach to try to create a prediction model for time to next vulnerability for several commercial software products.

Vulnerability discovery models, time-based or effort-based, have been researched as a means to predict latent vulnerabilities. Alhazmi and Malaiya [18] quantitatively compare six time-based models of the vulnerability discovery process observing that if vulnerability density can be estimated for a software product from historical data, an estimate of the total number of vulnerabilities, including those not yet discovered, can be computed. Woo, et al. [19] also studied vulnerability discovery models, applying one time-based model and one effort-based model to the Apache and IIS webservers using historical data from 1995 to 2009. Both models demonstrated good fits with the data suggesting that it is feasible to estimate the number and discovery rate of latent vulnerabilities.

Alhazmi, Malaiya, and Ray [20] extend defect density, a common software quality metric, introducing vulnerability density which is calculated as the ratio of vulnerabilities per 1,000 lines of code. They applied vulnerability density and a vulnerability to defect ratio of 1-5% from the literature to arrive at estimates for known and residual (latent) vulnerabilities in several popular operating systems. This paper extends that work but differs from Alhazmi primarily in the method to estimate total flaws and on certain other assumptions discussed later.

Shin and Williams [21] consider whether software fault prediction models can be used for vulnerability prediction using an empirical study of the Mozilla Firefox web browser. The results suggest that fault prediction models based on traditional software quality metrics can be used for vulnerability prediction supporting our work. In the context of the Firefox data, they observed a vulnerability to defect ratio of 15%.

The study of software quality metrics has a history dating back to at least the 1990s. Radjenovic et al. [22] examines 106 papers from seven journals between 1991 and 2011 where software metrics for fault prediction were empirically validated. The source lines of code (SLOC, also abbreviated LOC or KLOC, for thousands of lines of code) metric was deemed an effective software quality metric across various life cycle phases, languages, and sizes. LOC was observed to correlate with number of faults in six studies; however, size does not predispose software to be faulty.

Ozment and Schechter [23] analyze 7.5 years of vulnerability reports for 15 versions of OpenBSD, an operating system known for its emphasis on security, and for which complete source code and change history is readily available. They observed that 62% of the vulnerabilities reported during the study period were foundational (i.e. originated in the first version) and 61% of source code remained unchanged during the 15 versions. Moreover, the lifespan of a vulnerability is at least 2.6 years. They also noted that, while new code resulted in new vulnerabilities, an assertion we adopt, they were unable to demonstrate a significant correlation between the number of new lines of code and the number of new vulnerabilities reported.

III. Estimation Approach

To test our hypothesis that cyber risk assessment approaches that depend on vulnerability enumeration will tend to significantly undercount vulnerabilities in complex software systems, we develop an estimation approach described in this section. Support for the hypothesis would provide motivation to seek risk assessment approaches that do not depend solely on vulnerability enumeration.

Figure 2 places the latent vulnerability metric we develop in this paper into context. S represents the target software system under consideration. We define a number of variables related to S, as summarized in Table II.

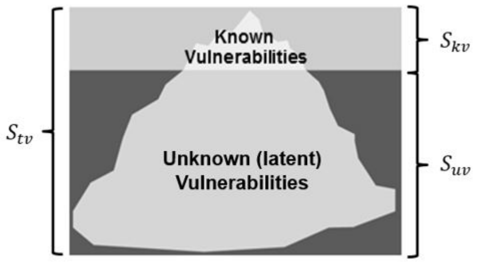


Fig. 2: Known vs. Unknown Vulnerabilities

Variable	Meaning with respect to S
S_{tv}	Total number of vulnerabilities present in S
S_{kv}	Number of known vulnerabilities in S
S_{uv}	Number of unknown vulnerabilities in S
S_{flaws}	Total number of flaws in S
S_{fr}	Rate at which software developers introduce flaws into S
S_{vr}	Fraction of S_{flaws} that represent exploitable vulnerabilities
S_{kloc}	Thousands of lines of code in S

TABLE II: Variables in the Estimation Approach

As the table shows, S_{tv} is the total number of vulnerabilities present in S, S_{kv} is the number of known vulnerabilities in S, and S_{uv} is the number of unknown vulnerabilities in S. S_{tv} is the sum of known and unknown vulnerabilities in S (1).

$$S_{tv} = S_{kv} + S_{uv} \tag{1}$$

While we know of no way to estimate S_{uv} directly, if we can arrive at estimates for S_{tv} and S_{kv}, then a simple rearrangement of (1) yields (2), which is analogous to equation (3) in Alhazmi [20].

$$S_{uv} = S_{tv} - S_{kv} \tag{2}$$

The task then becomes one of estimating S_{tv} and S_{kv}, which we discuss next.

A. Estimating S_{tv}

As has been well documented [24][25], software developers unwittingly introduce flaws or defects into their software as they carry out development. We define a software flaw as an aspect of S that violates the implicit or explicit requirements for that system. Our estimate of S_{tv} is based on the premise that some fraction of flaws in S will be exploitable vulnerabilities. Thus, we must first estimate the flaw rate, S_{fr}. We can estimate S_{fr} using a measure of the software system size, e.g., thousands of source lines of code, S_{kloc}, or function points [26]. We chose to use source lines of code, S_{kloc}, so equation (3) yields the number of flaws in S as a product of the software size and flaw rate.

$$S_{flaws} = S_{fr} \times S_{kloc} \tag{3}$$

While lines of code is not a perfect metric, it serves the purpose relative to the larger question that this paper explores. There is precedent in using lines of code. As Camilo, et al., [27] state:

> *Number of features, source lines of code, and number of pre-release security bugs are, in general, more closely associated with post-release vulnerabilities than any of our non-security bug categories.*

If we can determine S_{vr}, the fraction of S_{flaws} that represent exploitable vulnerabilities, then equation (4) gives an estimate of the total vulnerabilities, S_{tv}.

$$S_{tv} = S_{flaws} \times S_{vr} \tag{4}$$

Combining (2), (3) and (4) above yields (5), an equation for estimating S_{uv}, the unknown vulnerabilities in S.

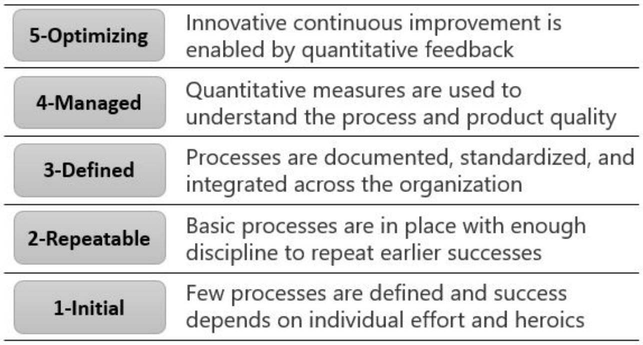
$$S_{uv} = (S_{fr} \times S_{kloc} \times S_{vr}) - S_{kv} \quad (5)$$

As S_{kloc} is known and S_{kv} can per estimated, per the next section, the task now becomes determining reasonable values for the flaw and vulnerability rates of S, namely S_{fr} and S_{vr}.

Research has sought to determine the rate at which software flaws occur. Alhazmi [20] looked across various sources of system-specific defect data available at the time to estimate defect rates. However, such data can be difficult to find and, we suspect, undercounts flaws given the point-in-time nature of such data.

We explore an alternative approach based on the Capability Maturity Model (CMM) [28], which allows us to be more predictive. Jones [25] studied known flaw rates occurring in different types of organizations using the CMM level as the basis of distinguishing organizations (Figure 4).

Organizations can achieve a CMM rating by meeting requirements associated with a given rating level plus all requirements of lower levels. There are five levels: 1-Initial, 2-Repeatable, 3-Defined, 4-Managed, and 5-Optimizing. All organizations begin at level 1. Organizations qualify for level 2 when they employ disciplined processes for software configuration management, quality assurance, project tracking, and requirements management. Level 3 organizations add processes such as peer reviews, software product engineering, and integrated software management. Level 4 adds software quality management and quantitative process management, and, lastly, level 5 adds process and technology change management and defect prevention (Figure 3 [29]).



5-Optimizing	Innovative continuous improvement is enabled by quantitative feedback
4-Managed	Quantitative measures are used to understand the process and product quality
3-Defined	Processes are documented, standardized, and integrated across the organization
2-Repeatable	Basic processes are in place with enough discipline to repeat earlier successes
1-Initial	Few processes are defined and success depends on individual effort and heroics

Fig. 3: Summary of CMM Levels [29]

As Jones discovered, based on an analysis of approximately 9,000 software projects over 15 years, flaw rates in software decrease in a roughly linear fashion as a function of rising CMM levels. For example, developers in an organization at CMM Level 2 can be expected to introduce about 6.2 flaws per KLOC, whereas a CMM Level 4 organization introduces about 2.2 flaws per KLOC. Jones' flaw rate data is based on flaws found, which is a subset of the total flaws actually present. Since we do not know how large the subset is, relative to the whole, we ignore this difference to be conservative in our analysis.

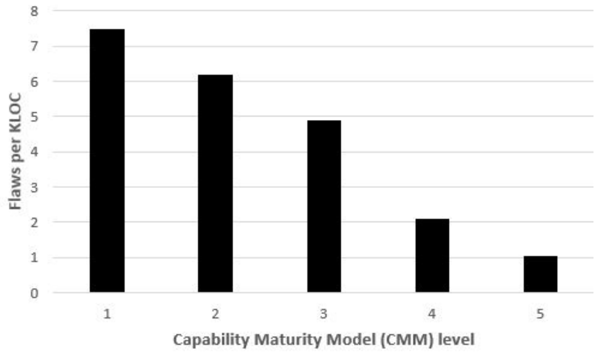


Fig. 4: Software Defects by Organizational Maturity

We can thus estimate the number of flaws; however, it bears repeating that not all flaws are vulnerabilities that attackers can exploit for a security breach. For example, a flaw in a help system that displays the wrong help text is unlikely to be exploitable as a security flaw, whereas a buffer overflow flaw may be an exploitable vulnerability. For our purposes, we need to know the fraction of flaws in a system that actually represent exploitable vulnerabilities. To be conservative, we use a figure of 1% for S_{vr} in this paper, an estimate we base on a discussion by Alhazmi [20].

Table III presents S_{tv} for five operating system flaw rate scenarios. Scenarios 1 and 2 are the Windows NT and Windows 2000 entries from Table 1 of Alhazmi [20]. Scenarios 3, 4, and 5 represent a generic 50,000 KLOC OS-scale software system developed at CMM levels 5, 4, and 3 respectively from Figure 4. On the assumption that most professional software organizations are, in 2017, above CMM level 2, we ignore the CMM level 1 and 2 cases, though some might argue for their inclusion.

Scenario	OS	KLOC	S_{fr}	S_{flaws}	S_{vr}	S_{tv}
1	Win NT	16,000	0.63	10,000	1%	100
2	Win 2K	35,000	1.80	63,000	1%	630
3	Generic	50,000	1.00	50,000	1%	500
4	Generic	50,000	2.20	110,000	1%	1,100
5	Generic	50,000	4.90	245,000	1%	2,450

TABLE III: Total Vulnerability S_{tv} Estimates

For example, in the table Scenario 2 is for Windows 2000, which consists of 35k lines of code. An estimated flaw rate of 1.8 flaws per thousand lines of code implies 63k flaws ($1.8 \times 35k$), which yields an estimate of 630 exploitable vulnerabilities ($63k \times 1\%$).

B. Estimating S_{kv}

One can estimate the known vulnerabilities, S_{kv}, by searching openly available vulnerability repositories for vulnerability data concerning S. One example of such a repository is NVD, mentioned earlier. While one can find major Commercial Off-the-Shelf (COTS) products (e.g., operating systems and applications) in NVD, published vulnerability data on custom-built systems within an enterprise are unlikely to be present in such repositories. Another complicating factor is that such repositories contain known vulnerabilities data up to a point in time. The longer S is in use, the more time the community has had to find vulnerabilities in S, so, in general, S_{kv} can be expected to rise over time and then eventually begin to level off as S is superseded (e.g., the case of Windows XP from Table IV).

Based on the limitations of data from openly available vulnerability repositories as mentioned above, we confine our known vulnerability estimates to major operating systems. Table IV represents data extracted from NVD showing vulnerabilities discovered per year for the Linux Kernel, Red Hat Linux, Mac OS, and four major versions of Windows.

OS	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**
Linux Kernel	63	76	105	125	86	116	189	135	86	218
Red Hat Linux	38	37	17	40	35	57	198	223	234	253
Mac OS	106	94	81	97	74	37	72	151	444	215
Win. XP	34	34	89	98	101	43	87	7		
Win. Vista	31	29	76	86	95	42	95	34	136	125
Win. 7			15	64	102	44	99	36	147	134
Win. Svr 2012						5	51	38	155	156

TABLE IV: Vulnerabilities discovered by OS by year

The median annual discovery count for OS-scale software systems calculated from the values in Table IV is 86. We use this figure rather than a cumulative count as the value for S_{kv} based on the following assumption and hypothesis. While in general there is a lag in patching from the time of vulnerability discovery, we make a simplifying assumption that patching is immediate upon discovery, thus effectively eliminating the corresponding known vulnerability from S. We hypothesize that new vulnerabilities arise about as fast as old vulnerabilities are discovered and patched, justified by the observation that new vulnerabilities stem from the addition of features and the patching process itself. This hypothesis is also justified in part by Camilo, et al. [27]:

> *The strongest indicators of vulnerability are past security-related bugs and new features.*

While eventually vulnerability discovery and introduction rates drop as software interest and use falls off, the hypothesis implies that we never really get ahead. A plot of the median annual discovery rate across OSs appears to support this assertion (Figure 5), as, rather than dropping off, annual discovery rates overall tend to increase, at least for this particular dataset. Note that the plot covers Red Hat Linux, Mac OS, and Windows XP, Vista, and 7, as these OSs had non-zero known vulnerability counts over the years 2009-2016.

IV. Analysis

Applying equation (5) to the data in Table V (an extension of Table III), we estimate S_{uv}. We note that the percentage that S_{uv} represents of S_{tv} (% column) stays well above the 50% mark for all but Scenario 1, which represents the case of Windows NT. At 16,000 KLOC, Windows NT is far smaller in size than modern operating systems.

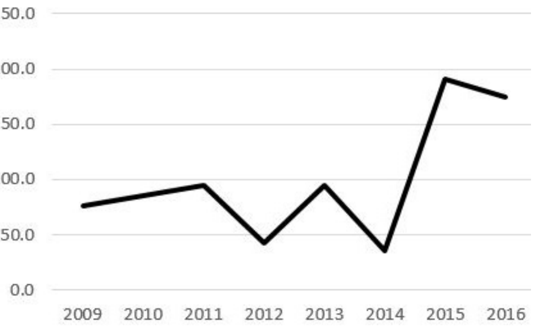


Fig. 5: Median Vulnerabilities Discovered Annually: 2009-2016

Scenario	S_{fr}	S_{kloc}	S_{flaws}	S_{vr}	S_{tv}	S_{kv}	S_{uv}	%
1	0.63	16,000	10,000	1%	100	86	15	15.0%
2	1.80	35,000	63,000	1%	630	86	544	86.3%
3	1.00	50,000	50,000	1%	500	86	414	82.8%
4	2.20	50,000	110,000	1%	1,100	86	1,014	92.2%
5	4.90	50,000	245,000	1%	2,450	86	2,364	96.5%

TABLE V: Unknown Vulnerability (S_{uv}) Results

Figure 6 graphically depicts unknown flaw percentages in each of the five scenarios.

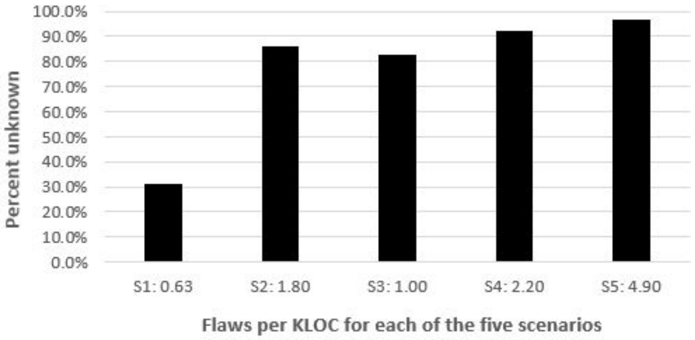


Fig. 6: Percent of total vulnerabilities that are unknown for each scenario

Our hypothesis, that currently the community can only enumerate a minority of software vulnerabilities present in a large software system, was borne out in 4 of the 5 scenarios considered. Pending independent validation of equation (5), as discussed in the next section, this result suggests that risk assessment methods that depend upon enumerating known vulnerabilities will tend to greatly undercount the actual vulnerabilities present. Specifically, if one assesses risk by positing attacks based on known vulnerabilities, as is commonly done, then the risk assessment will only consider a small fraction of all possible attacks and will thus likely underestimate the risk.

V. Conclusion and Future Work

A. Conclusion

This paper explored the efficacy of basing a cyber risk assessment on the enumeration of known vulnerabilities. To examine this question, we developed a metric, described in equation (5), to estimate the total number of unknown or latent vulnerabilities in a large software system. The equation requires an independent means of validation, discussed below in Future Work; however, preliminary analysis of empirical data with this equation suggests that, except in the most optimistic scenario, the majority of vulnerabilities in such systems remain latent at any given time. Indeed, with respect to the research question posed earlier, the authors are concerned that we may never really get ahead in our race to discover all latent vulnerabilities because even as the community discovers and eliminates vulnerabilities, new vulnerabilities are introduced with software feature upgrades and during the patching process itself.

This finding raises concerns about the efficacy of vulnerability-centric cyber risk assessments given the lack of any systematic procedure for finding all vulnerabilities in large-scale software systems. We therefore suggest caution in interpreting results from such assessments because the number of unknown vulnerabilities in large-scale software tends to exceed known vulnerabilities, and we furthermore suggest risk assessment methodologies should consider both known and unknown vulnerabilities, something not commonly done today, though there are exceptions. For example, as part of the risk analysis process, MIRA [30] hypothesizes the existence of latent vulnerabilities and analyzes their potential effects. The findings also support the concept that mission resilience to cyber attack should be nurtured and explored, as prevention-centric mitigation strategies are inherently disadvantaged given that latent vulnerabilities lie in wait for adversaries. In this regard, the NIST Resilience Framework [31] emphasizes not only prevention, but also detection, response, and recovery.

B. Future Work

There are a number of future work possibilities, as given in the list below:

1) Explore ways to independently validate equation (5). There are two assertions embedded in the equation that could be validated, together or separately. In particular, (a) can the number of flaws in a software system can be expressed as a linear function of the size of the software and a rate of introduction? and (b) can the total vulnerabilities can be expressed as a linear function of the flaws and a flaw rate?;
2) Repeat the analysis with updated flaw rate data;
3) Design and carry out experiments to tease out a typical ratio of vulnerabilities introduced vs. vulnerabilities patched across software system updates;
4) Analyze vulnerability data for software beyond operating systems (e.g., commercial or open source software) to examine how flaw and discovery rates vary in other contexts;
5) Expand beyond software to study vulnerability rates related to non-software contexts, including weaknesses in people and processes; and
6) Compare flaw rate predictions from analyses such as that done by Jones [25] to empirical flaw data gathered over the life of a target software system.

Item 1) above is challenging given the lack of any systematic procedure for finding all software vulnerabilities. Thus, an absolute ground truth measurement of S_{uv} is unavailable. However, heuristic approaches could be explored. For example, Chowdhury, and Zulkernine [32] find that software complexity, coupling, and cohesion metrics are correlated to vulnerabilities at a statistically significant level. While their research was focused on web browsers, an investigation of the applicability of the approach to operating system-scale software could be undertaken as one means to generate independent data for comparison to equation (5).

Finally, concerning item 4) above, Table VI summarizes data gathered from NVD [7] for vulnerabilities discovered in selected major application software in 2016. The counts indicate that non-O/S software is an area ripe for investigation for the type of analysis described in this paper.

Application	Vulnerability Count
Adobe Flash Player	249
Adobe Acrobat	224
Google Chrome	159
Mozilla Firefox	133
MS Internet Explorer	122
PHP	107
Wireshark	95

TABLE VI: Sample Application Vulnerabilities Discovered in 2016

References

[1] Cybersecurity Ventures, “Cybersecurity Market Report,” 2016.

[2] National Institute of Standards and Technology, “National Institute of Standards and Technology 800-30: Guide for Conducting Risk Assessments,” Tech. Rep. September, National Institute of Standards and Technology, 2012.

[3] Wikipedia, “Source lines of code.”

[4] Hewlett Packard Enterprise Security Research, “Cyber Risk Report 2015,” 2015.

[5] Apple, “Secure Coding Guide,” 2016.

[6] J. Viega, M. Howard, and D. LeBlanc, *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them.* McGraw-Hill Education, 1st ed., 2009.

[7] NVD, “National Vulnerability Database.”

[8] S. Özkan, “CVE Details,” 2017.

[9] National Institute of Standards and Technology (NIST), “NVD CVSS Support,” 2017.

[10] R. Caralli, J. Stevens, L. Young, and W. Wilson, “Introducing OCTAVE Allegro: Improving the Information Security Risk Assessment Process,” 2007.

[11] J. Wynn, J. Whitmore, G. Upton, L. Spriggs, D. McKinnon, R. McInnis, R. Graubert, and L. Clausen, “Threat Assessment and Remediation Analysis Methodology,” 2012.

[12] ISACA, “COBIT 5 Framework,” 2012.

[13] T. Llanso, P. Hamilton, and M. Silberglitt, “MAAP : Mission Assurance Analytics Platform,” in *IEEE Homeland Security Technologies Conference*, (Boston), pp. 549–555, 2012.

[14] K. Jabbour and J. Poisson, “Cyber Risk Assessment in Distributed Information Systems,” 2016.

[15] M. A. McQueen, T. A. McQueen, W. F. Boyer, and M. R. Chaffin, “Empirical Estimates and Observations of 0Day Vulnerabilities,” in *System Sciences, 2009. HICSS ’09. 42nd Hawaii International Conference on*, pp. 1–12, jan 2009.

[16] J. Jones, “Estimating Software Vulnerabilities,” *IEEE Security & Privacy*, vol. 5, no. undefined, pp. 28–32, 2007.

[17] S. Zhang, D. Caragea, and X. Ou, *An Empirical Study on Using the National Vulnerability Database to Predict Software Vulnerabilities*, pp. 217–231. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[18] O. H. Alhazmi and Y. K. Malaiya, “Application of Vulnerability Discovery Models to Major Operating Systems,” *IEEE Transactions on Reliability*, vol. 57, pp. 14–22, mar 2008.

[19] S.-W. Woo, H. Joh, O. H. Alhazmi, and Y. K. Malaiya, “Modeling vulnerability discovery process in Apache and IIS HTTP servers,” *Computers & Security*, vol. 30, no. 1, pp. 50–62, 2011.

[20] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, “Measuring, analyzing and predicting security vulnerabilities in software systems,” *Computers & Security*, vol. 26, no. 3, pp. 219–228, 2007.

[21] Y. Shin and L. Williams, “Can traditional fault prediction models be used for vulnerability prediction?,” *Empirical Software Engineering*, vol. 18, no. 1, pp. 25–59, 2013.

[22] D. Radjenovic, M. Hericko, R. Torkar, and A. Zivkovic, “Software fault prediction metrics: A systematic literature review,” *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[23] A. Ozment and S. Schechter, “Milk or wine: Does software security improve with age?,” in *Usenix Security*, 2006.

[24] D. Jones and S. Gregor, “The Anatomy of a Design Theory,” *Journal of the Association of Information Systems*, vol. 8, no. 5, 2007.

[25] C. Jones, *Software Assessments, Benchmarks, and Best Practices*. Addison-Wedley Information Technology Series, 2000.

[26] J. E. Matson, B. E. Barrett, and J. M. Mellichamp, “Software development cost estimation using function points,” *IEEE Transactions on Software Engineering*, vol. 20, pp. 275–287, apr 1994.

[27] F. Camilo, A. Meneely, and M. Nagappan, “Do Bugs Foreshadow Vulnerabilities?: A Study of the Chromium Project,” in *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR ’15, (Piscataway, NJ, USA), pp. 269–279, IEEE Press, 2015.

[28] M. Paulk, C. Weber, S. Garcia, M. B. Chrissis, and M. Bush, “Key Practices of the Capability Maturity ModelSM, Version 1.1,” tech. rep., 1993.

[29] K. Hartman, “CMM & Organizational Process Maturity,” 2012.

[30] T. Llanso, G. Tally, M. Silberglitt, and T. Anderson, “Applicability Of Mission-Based Analysis For Assessing Cyber Risk In Critical Infrastructure Systems,” in *International Federation for Information Processing (IFIP) - Critical Infrastructure Protection VII*, vol. VII, ch. 9, pp. 135–148, Springer Berlin Heidelberg New York, 2013 ed., 2013.

[31] National Institute of Standards and Technology, “Framework for Improving Critical Infrastructure Cybersecurity,” tech. rep., 2014.

[32] I. Chowdhury and M. Zulkernine, “Can Complexity, Coupling, and Cohesion Metrics Be Used As Early Indicators of Vulnerabilities?,” in *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC ’10, (New York, NY, USA), pp. 1963–1969, ACM, 2010.