Representativeness in the Benchmark for Vulnerability Analysis Tools (B-VAT)

Kayla N. Afanador
Naval Postgraduate School

Cynthia E. Irvine
Naval Postgraduate School

Abstract

A variety of tools are used to support software vulnerability analysis processes. However, when analysts want to select the optimal tool for a particular use case, or attempt to compare a new tool against others, no standard method is available to do so. In addition, we have determined that although vulnerabilities can be categorized into *vulnerability types*, these types are often disproportionately represented in current datasets. Hence, when comparative analyses of tools based upon these data sets are conducted, the results are distorted and unrealistic. To address this problem, we are building a Benchmark for Vulnerability Analysis Tools (B-VAT).

Representativeness is a key element of a good benchmark. In this paper, we use stratified sampling to identify a representative set of vulnerabilities from over 800 CWE's and 75,000 CVE's. This set becomes the foundation upon which we will build a purpose-based benchmark for vulnerability analysis tools. By using the correlation between current CWE and CVE data, the proposed B-VAT will assess tools using vulnerabilities in the proportions their types occur in the wild.

1 Introduction

The security community relies on tools to support software vulnerability analysis processes. However, there is no benchmark to support the comparison of vulnerability analysis tools. One current comparison approach is to see how well tools perform using test cases.

Hundreds of thousands of publicly available test cases containing known software flaws are aggregated into datasets of vulnerabilities, each with its own structure, test cases, supported languages, and reporting method, e.g., Juliet Test Suite [1]. Databases such as the Software Assurance Reference Dataset (SARD) attempt to inject order by providing a consolidated repository of vulnerability datasets and test cases [2]. Unfortunately, even the SARD, which contains 40 datasets and over 170,000 test cases, is not exhaustive— it excludes datasets such as the Cyber Grand Challenge (CGC) Corpus [3], and OWASP Benchmark Project [4]. We will show that many of these datasets contain an unrealistic representation of weakness types, i.e., Common Weakness Enumeration (CWE) entries, when compared to known vulnerability instances in the wild, i.e., accepted Common Vulnerabilities and Exposures (CVE) entries. Consequently, even if vulnerability analysis tools were assessed using *all* of the SARD test cases, the results would would still not reflect reality. In our effort to create a Benchmark for Vulnerability Analysis Tools (B-VAT), test cases that are statistically representative of the vulnerabilities found in the wild are needed.

The CWE provides a repository of weakness types, and the CVE, a dictionary of vulnerability instances. A **weakness** is a mistake in software or hardware that, under proper conditions, could lead to the introduction of a vulnerability [5, 6]. A **vulnerability** is an occurrence of one or more weaknesses that can be used to, "modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness" [6].

We analyzed four popular datasets of software vulnerabilities: Juliet C/C++, Juliet Java [1], the CGC Corpus [3], and the OWASP Benchmark [4]. Each contains test cases that can be used to assess vulnerability analysis tools. We show that none represent vulnerability types in the proportions those types occur in the wild. By correlating the data from 839 CWE's, and over 75,000 CVE's we identify a representative set of known vulnerabilities that can be used to design a purpose-based benchmark for vulnerability analysis tools.

This paper makes the following contributions:

- We synthesize 839 CWE's with over 75,000 CVE's to determine the relative proportions of vulnerability instances and weakness types in the wild.
- We analyze four popular software vulnerability datasets, and show that none accurately represents vulnerability instances and weakness types as they occur in the wild.
- We perform stratified sampling to determine the appropriate number of test cases for each weakness type. This provides the foundational data upon which B-VAT is being constructed.

We next provide a brief background on benchmarks, and identify the key benchmark property motivating this paper: representativeness. Section 3 explores how representativeness is achieved, and Section 4 offers a conclusion and future work.

2 The Benchmark

There are many types of computer benchmarks. In the 1960's traditional benchmarks were used to compare the speed with which computers accomplished basic data processing functions [7]. In 1965, Joslin proposed an *application benchmark* that used specific applications to emphasize the relative throughput performance of different system configurations [8]. *Specification-based benchmarks* define functions, and include required inputs and expected output [9]. Gustafson defined a *purpose-based benchmark* that, "explicitly and comprehensively measures the ability of a computing system to reach a goal of human interest" [10]. Our benchmark has a clear purpose: to facilitate comparison of vulnerability analysis tools. For this reason, we are designing a purpose-based benchmark.

2.1 Benchmark Characteristics

Researchers have proposed a number of desirable benchmark characteristics [9, 11–13]. We consider the following key characteristics for B-VAT:

Relevant The benchmark problems should be closely connected to reality.

Repeatable The same results should be consistently reproduced when the benchmark is run with the same tool.

Usable The benchmark should be able to be used in multiple operating environments, and run with a variety of tools.

Fair The benchmark should impartially measure each tool.

Verifiable There should be confidence that benchmark results are accurate.

Determining the relevance of benchmark problems involves a number of elements. From a design perspective, relevance involves two dimensions: the breadth of the benchmark's applicability, and the degree to which benchmark problems are relevant in each area [9, 12]. In this paper, we focus on relevance, and specifically on the *representativeness* of the problems in B-VAT. See Section 4 for a description of future work related to B-VAT.

3 A Representative Set

The most important property of a benchmark relates to its problems [8, 10]. Joslin called this the "representativeness" of a benchmark, and Gustafson the benchmark's "problem size." Gustafson proposed a balance between a benchmark's problem size and its usefulness – too many problems in a benchmark raises its cost, while too few reduces its utility [10]. We will refer to benchmark problems as *test cases*.

We define a **representative set** of vulnerabilities as a subset of vulnerabilities instances that adequately represents the larger set of known vulnerability instances and types [14]. We identify a representative set from a repository of over 75,000 accepted CVE's published between 2014 and 2019.

3.1 Vulnerability Instances

Much like the SARD, the CVE provides organization and standardization. The CVE is a dictionary of publicly known vulnerability and exposure *instances* [15]. Each entry describes an instance of a vulnerability, and includes metadata such as a unique identifier (CVE ID), standardized description, and where applicable, a corresponding CWE entry[1]. We cannot predict future vulnerabilities; our work is constrained to the set of "known known" and "known unknown" vulnerabilities [16]. Additionally, we recognize that the CVE is not exhaustive, however, it provides an extensive repository of known vulnerability instances that is suitable for our purposes.

To date, the greatest number, 21,598, of publicly disclosed vulnerabilities and exposures was reported by the CVE in 2018. Over half, 93,056 out of 160,544, of all of vulnerabilities ever reported (excluding 2020) by the CVE were published between 2014 to 2019 [15]. We use these six years of 75,535 community-accepted CVE's as the set from which we identify a representative subset of vulnerability instances.

The CVE provides instances of known vulnerabilities, and the simplest method to identify a representative subset from these data would be to take a random sample of the larger set. However, a simple random sample may result in the misrepresentation of vulnerability *types* [17]. By using the existing correlation between CVE ID's and CWE ID's we can link each CVE ID to one of ten CWE pillars and take a stratified sample of the set.

3.2 Weakness Types

What the CVE provides for vulnerability instances, the CWE provides for weakness *types*. The CWE is a repository of over 1200 hardware and software weaknesses, and provides a common language, identifier, and definition for each weakness type referenced. CWE entries are organized into a number of views to support different objectives. We use view CWE-1000, Research Concepts, that includes a hierarchy of 839 CWE entries. The hierarchy contains one of the following *abstraction types* for each CWE ID [18]:

Pillar Weaknesses that are described in the most abstract fashion (10 CWE's). Pillars include:

1. CWE-284 Improper Access Control
2. CWE-435 Improper Interaction Between Multiple Correctly-Behaving Entities
3. CWE-664 Improper Control of a Resource Through its Lifetime
4. CWE-682 Incorrect Calculation
5. CWE-691 Insufficient Control Flow Management
6. CWE-693 Protection Mechanism Failure

[1] CVE entries published prior to the development of the CWE (2006) do not include a CWE ID

7. CWE-697 Incorrect Comparison
8. CWE-703 Improper Check or Handling of Exceptional Conditions
9. CWE-707 Improper Neutralization
10. CWE-710 Improper Adherence to Coding Standards

Class Abstract weakness, typically independent of any specific language or technology (96 CWE's).

Base A more specific type of weakness (441 CWE's).

Variant A weakness that is described at a very low level of detail, typically limited to a specific language or technology (285 CWE's).

Composite A set of weaknesses that must all be present simultaneously in order to produce an exploitable vulnerability (7 CWE's).

The weakness hierarchy presented by view CWE-1000 can be organized into ten **rooted trees**. A rooted tree is a tree with a single *root* vertex that is distinguished from all others. Each pillar in the hierarchy is a *root* node of a rooted tree.

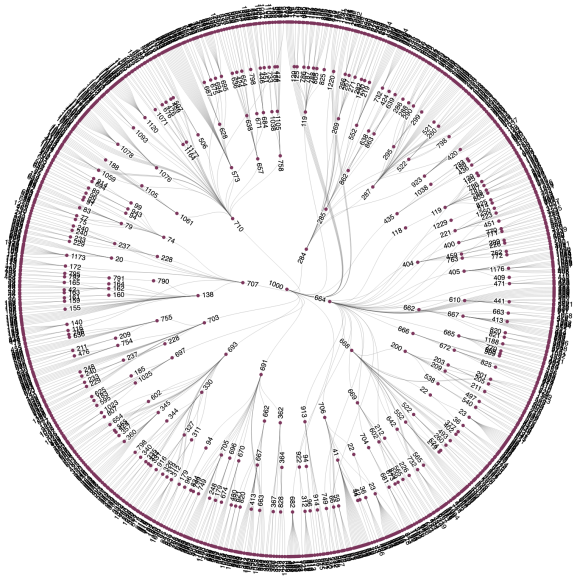

Figure 1: CWE view 1000 as a hierarchical radial dendrogram

Using BeautifulSoup [19], pandas [20], and D3 [21] we crawled over 1000 individual pages on the CWE website to create and visualize the tree data structures for each of the ten CWE Pillars. This approach allows us to view the most up-to-date information on CWE relationships and hierarchies. Then, by using CWE-1000 as the root node of a tree we can create a single rooted tree that includes every CWE in the CWE-1000 view. Figure 1 depicts the ten CWE pillars and their 839 children as a hierarchical radial dendrogram with root node CWE-1000. We use the CWE relationship and hierarchy data to organize the CVE's into ten strata (see Section 3.4).

3.3 Combining Types with Instances

Of the 75,535 community-accepted CVE's, 55,128 have an associated CWE ID. By using the correlations between CVE ID's and CWE ID's, we classify each of these vulnerability instances by their weakness type. Figure 2 shows the sum of known vulnerability instances (CVE IDs), by type (CWE IDs) from 2014-2019. This visualization shows the relative proportions of CWE ID's in the wild. The enclosing circles show the cumulative size of each of the ten CWE pillars (i.e. subtrees), while maintaining relationship and hierarchical data. The exterior circle represents root node, CWE-1000.

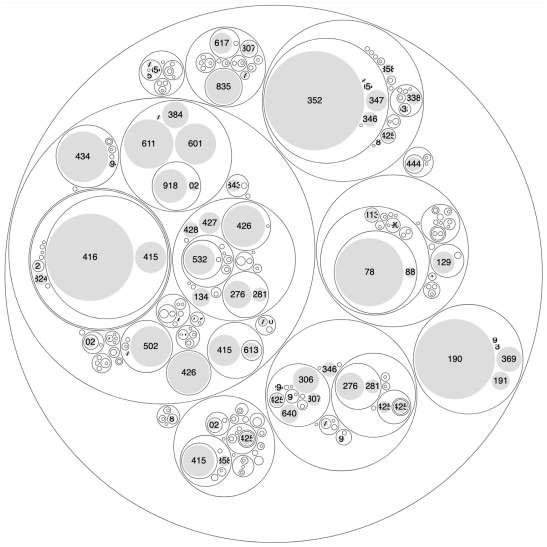

Figure 2: Sum of vulnerability instances (CVE ID) by type (CWE ID) from 2014-2019

We then use the CWE hierarchy to trace each vulnerability instance to a corresponding pillar node. This allows us to represent each CWE pillar as a percentage of the total known vulnerability instances. For example, pillar CWE-664 and its [grand]children represent 45% of community-accepted vulnerabilities (CVE's) published from 2014-2019. We repeat this process using four popular vulnerability datasets: Juliet C/C++ (Jul.(C)), Juliet Java (Jul.(J)), the OWASP Benchmark (OWASP), and the CGC Corpus (CGC).

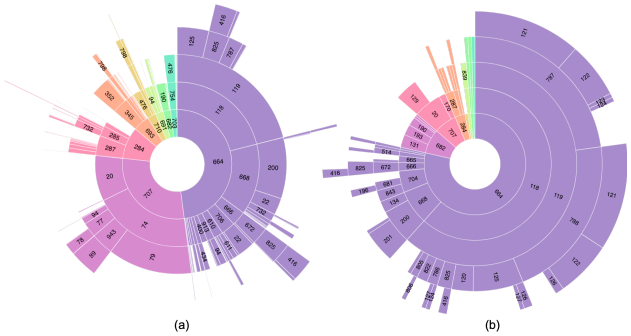

Figure 3: Comparison of sunburst diagrams

Figure 3 also shows the sum of vulnerability instances (CVE ID) by type (CWE ID) from 2014-2019 (a), and includes a

sum of test cases in the CGC Corpus by type (CWE ID) (b) for comparison. These diagrams illustrate the stark contrast between the types of weaknesses in the wild, and those in current vulnerability datasets[2].

Like the CVE, each test case in the reviewed datasets has a corresponding CWE ID that can be traced to a pillar node. Table 1 shows the relative percentages of test cases in each pillar by vulnerability dataset. It shows that none of the vulnerability datasets accurately reflects vulnerabilities as they have occurred in the wild, i.e., CVE's from 2014-2019. By taking a stratified sample of the CVE-CWE data we propose a subset of test cases for B-VAT that proportionately represents vulnerability instances, and weakness types.

Pillar	CVE	CGC	Jul.(C)	Jul.(J)	OWASP
CWE-284	10.66%	3.47%	0.95%	0.89%	0.00%
CWE-435	0.07%	0.00%	0.04%	0.00%	0.00%
CWE-664	45.27%	72.25%	68.48%	25.52%	14.38%
CWE-682	2.53%	9.83%	12.66%	34.06%	0.00%
CWE-691	2.57%	0.00%	0.65%	0.35%	0.00%
CWE-693	4.66%	0.00%	0.47%	1.73%	38.03%
CWE-697	0.03%	1.16%	0.02%	0.12%	0.00%
CWE-703	0.30%	1.16%	0.91%	0.34%	0.00%
CWE-707	32.03%	9.25%	10.40%	33.19%	47.59%
CWE-710	1.87%	0.00%	5.18%	3.43%	0.00%

Table 1: % of test cases in each CWE pillar by dataset

3.4 A Stratified Sample

Stratified sampling is a statistical method that allows subgroups, or *strata* to be proportionately represented [22], thus providing a representative sample of a larger population. Unlike random sampling, which may result in the misrepresentation of vulnerability instances and weakness types, stratified sampling allows us to preserve the relative proportions of each pillar, or strata. Table 2 shows the proportionate stratified sample size of each pillar.

Pillar	Total CVE's	Stratified Sample
CWE-284	5,847	245
CWE-435	40	2
CWE-664	24,957	1,042
CWE-682	1,397	58
CWE-691	1,419	59
CWE-693	2,571	107
CWE-697	15	1
CWE-703	168	7
CWE-707	17,657	737
CWE-710	1,030	43

Table 2: Stratified sample size of each CWE pillar

These sample sizes become the number of test cases in the B-VAT aligned to each CWE pillar. By using the correlation between CWE and CVE entries our B-VAT will represent vulnerability instances and weakness types in the proportions they occur in the wild. In this paper, we used CWE pillars as the strata in our sample, however, it may be prudent to add additional variables before taking a stratified sample. For example, we could also include a severity ranking for each CVE, and force our sample to include vulnerability instances with a high severity [23].

[2]We have similar diagrams for each of the datasets reviewed.

4 Conclusion and Future work

Despite the large number of tools used to support software vulnerability assessments, there is no benchmark that permits evaluation and comparison of those tools. To address this problem, we are developing a Benchmark for Vulnerability Analysis Tools. In this paper, we have discussed a fundamental property of B-VAT: representativeness. We examined four popular datasets of software vulnerability test cases: Juliet C/C++, Juliet Java, the OWASP Benchmark, and the CGC Corpus, and determined that none represent vulnerabilities as they have occurred in the wild from 2014-2019. First, we synthesized the data from 839 CWE's and over 75,000 CVE's, then we used stratified sampling to identify a distribution of weakness types that is representative of known vulnerability instances. This analysis provides a foundation for B-VAT; however, much work remains.

Currently, we are exploring the impact of including additional variables before taking a stratified sample (e.g., relevance of specific CVE's and CWE's, severity rankings, and a weighted component for rare vulnerabilities). After determining the desired number of test cases for B-VAT, we must determine a method to score vulnerability analysis tools. Existing datasets contain over 150,000 test cases; Table 3 shows the collective number of available test cases corresponding to each pillar. We are exploring methods to reuse these test cases in a more representative way, as described in this paper.

Pillar	Required Test Cases	Available Test Cases
CWE-284	245	3,309
CWE-435	2	42
CWE-664	1,042	92,733
CWE-682	58	28,876
CWE-691	59	1,511
CWE-693	107	3,321
CWE-697	1	76
CWE-703	7	1,117
CWE-707	737	34,417
CWE-710	43	7,236

Table 3: Available open-source test cases

This work is ongoing, and we welcome collaboration. All data and code is available upon request.

We wish thank Lyn Whitaker for valuable discussions.

References

[1] T. Boland and P. E. Black, "Juliet 1.1 C/C++ and Java Test Suite," *Computer*, vol. 45, no. 10, pp. 88–90, Oct. 2012.

[2] P. E. Black, "Sard: Thousands of reference programs for software assurance," *J. Cyber Secur. Inf. Syst. Tools Test. Tech. Assur. Softw. Dod Softw. Assur. Community Pract*, vol. 2, no. 5, 2017.

[3] B. Caswell. Cyber grand challenge corpus. [Online]. Available: http://www.lungetech.com/cgc-corpus/

[4] D. Wichers, "OWASP Benchmark," https://owasp.org/www-project-benchmark/.

[5] "CVE - Terminology," https://cve.mitre.org/about/terminology.html#vulnerability.

[6] "CWE - About - CWE Overview," https://cwe.mitre.org/about/index.html.

[7] B. C. Lewis and A. E. Crews, "The Evolution of Benchmarking as a Computer Performance Evaluation Technique," *MIS Quarterly*, vol. 9, no. 1, p. 7, Mar. 1985.

[8] Joslin, E. O., "2.1: Application Benchmarks: The Key to Meaningful Computer Evaluations.pdf," in *ACM 20th National Conference*, EDP Equipment Office, USAF, 1965, pp. 27–37.

[9] J. v. Kistowski, J. A. Arnold, K. Huppler, K.-D. Lange, J. L. Henning, and P. Cao, "How to Build a Benchmark," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering - ICPE '15*. Austin, Texas, USA: ACM Press, 2015, pp. 333–336.

[10] J. Gustafson, "Purpose-Based Benchmarks," *The International Journal of High Performance Computing Applications*, vol. 18, no. 4, pp. 475–487, Nov. 2004.

[11] J. Henning, "SPEC CPU2000: Measuring CPU performance in the New Millennium," *Computer*, vol. 33, no. 7, pp. 28–35, Jul. 2000.

[12] K. Huppler, "The Art of Building a Good Benchmark," in *Performance Evaluation and Benchmarking*, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, R. Nambiar, and M. Poess, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5895, pp. 18–30.

[13] R. H. Saavedra and A. J. Smith, "Analysis of benchmark characteristics and benchmark performance prediction," *ACM Transactions on Computer Systems (TOCS)*, vol. 14, no. 4, pp. 344–384, Nov. 1996.

[14] Feng Pan, Wei Wang, A. K. H. Tung, and Jiong Yang, "Finding representative set from massive data," in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, 2005, pp. 8 pp.–.

[15] MITRE, "CVE - Home," https://cve.mitre.org/about/index.html.

[16] D. Rumsfeld, "Dod news briefing–secretary rumsfeld and gen. myers," *US Department of Defense*, vol. 12, 2002.

[17] H. Taherdoost, "Sampling Methods in Research Methodology; How to Choose a Sampling Technique for Research," *SSRN Electronic Journal*, 2016.

[18] "CWE - A Comparison of the CWE Development and Research Views," https://cwe.mitre.org/documents/views/view-comparison.html.

[19] Richardson, Leonard, "Beautiful Soup Documentation — Beautiful Soup 4.9.0 documentation," https://www.crummy.com/software/BeautifulSoup/bs4/doc/#id19.

[20] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56 – 61.

[21] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 Data-Driven Documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.

[22] Levy, Paul S. and Lemeshow, Stanley, *Sampling of Populations: Methods and Applications*. John Wiley & Sons., 2013.

[23] P. Mell, K. Scarfone, and S. Romanosky, "A complete guide to the common vulnerability scoring system version 2.0," in *Published by FIRST-forum of incident response and security teams*, vol. 1, 2007, p. 23.