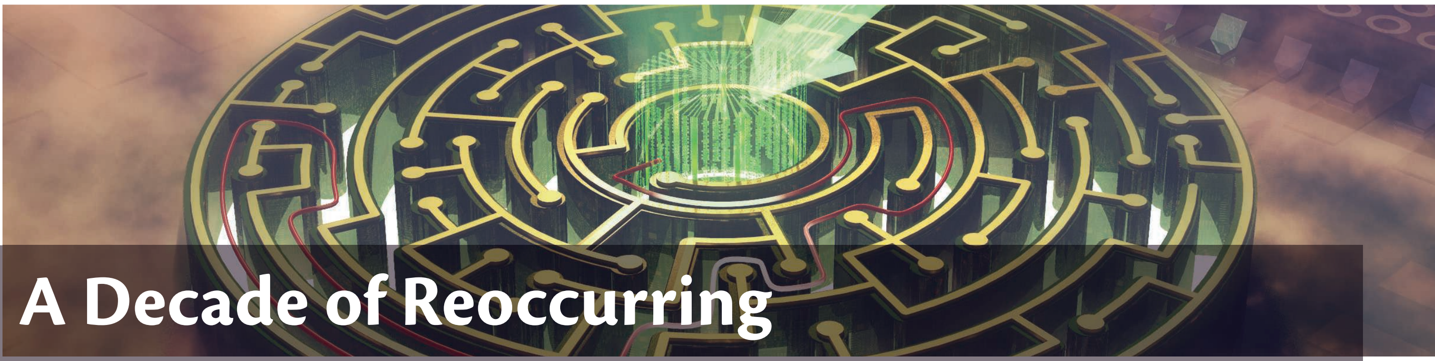

A Decade of Reoccurring Software Weaknesses

Assane Gueye | Carnegie Mellon University Africa
Carlos E.C. Galhardo | National Institute of Metrology, Quality, and Technology
Irena Bojanova and Peter Mell | National Institute of Standards and Technology

The Common Weakness Enumeration community publishes an aggregate metric to calculate the "most dangerous software errors." However, the equation highly biases frequency over exploitability and impact. We provide a metric to mitigate this bias and discuss the most significant weaknesses.

In 2020, there were more than 18,000 documented software vulnerabilities[1] that enable malicious activity. While many have been discovered, they map to a relatively small set of underlying weakness types. We posit that if the most significant of these types can be identified, developers of programming languages, software, and security tools can focus on preventing them and thus, over time, diminish the quantity and severity of newly discovered vulnerabilities. We define a "significant" weakness as one that is both frequently occurring among the set of publicly published vulnerabilities and also results in high severity vulnerabilities (those that are easily exploitable and have high impact). The set of security weakness types upon which we calculate significance comes from the Common Weakness Enumeration (CWE).[2]

In the fall of 2019, the CWE community published an equation to calculate the "top 25 most dangerous software errors (MDSE) among the CWE set.[3] It follows the form of the standard security risk matrix, combining probability/frequency and severity.

Digital Object Identifier 10.1109/MSEC.2021.3082757
Date of current version: 24 June 2021

The MDSE equation claims to combine "the frequency that a CWE is the root cause of a vulnerability with the projected severity"; the equation description implies that both factors are weighed equally (making no mention of any bias). However, we empirically found[4] that the equation highly biases frequency and almost ignores severity in generating top lists of varying sizes. This is due to the equation multiplying calculated frequency and severity values together, though each has very different distributions. Frequency distributions have a power-law-like curve, while severity distributions are more uniform. Our mitigation is to create a revised equation, called the *most significant security weaknesses (MSSW) equation*, which adjusts the frequency distribution using a double log function to better match it to the severity distribution.

Cybersecurity Vulnerabilities

We can define a vulnerability as a weakness in the security of a system that can be exploited.[5] The Common Vulnerabilities and Exposures (CVE) is a large set of publicly disclosed vulnerabilities in widely used software. These vulnerabilities are enumerated with a unique identifier, described, and referenced with external advisories.[6]

Scoring the Severity of Vulnerabilities

The Common Vulnerability Scoring System (CVSS) "provides a way to capture the principal characteristics of a vulnerability and produce a numerical score reflecting its severity."[7] The CVSS base score takes into account the exploitability (how easy it is to use the vulnerability in an attack) and impact (how much damage the vulnerability can cause to an affected component) of a vulnerability apart from any specific environment.

The exploitability score is determined by the following:[16]

- *attack vector*: "the context by which vulnerability exploitation is possible"
- *attack complexity*: "the conditions beyond the attacker's control that must exist in order to exploit the vulnerability"
- *privileges required*: "the level of privileges an attacker must possess before successfully exploiting the vulnerability"
- *user interaction*: a human victim must participate for the vulnerability to be exploited.

The impact score is determined by measuring the impact to the confidentiality, integrity, and availability of the affected system. Also included is a scope metric that "captures whether a vulnerability in one vulnerable component impacts resources in components beyond its security scope."

Weaknesses: Classifying Vulnerabilities

While we define a vulnerability in terms of a weakness, it is hard to define a weakness itself. As different vulnerabilities may be associated with the same weakness type, we could look at a weakness type as a class and a vulnerability as an instance of that class. Although it is uncommon, a single vulnerability could be associated with two or more weaknesses exploited sequentially or in parallel. In that sense, a vulnerability is a set with one or more instances of weaknesses.

The CWE is a "community-developed list of common software security weaknesses."[2] It contains an enumeration, descriptions, and references for 839 software weaknesses that are referred to as CWEs, where each is labeled CWE-*X* with *X* being an integer.

The CWE weaknesses model has four layers of abstraction: pillar, class, base, and variant. There is also the notion of a compound, which associates two or more interacting or co-occurring CWEs.[2] These abstractions reflect to what extent issues are described in terms of five dimensions: behavior, property, technology, language, and resource. Variant weaknesses are at the most specific level of abstraction; they describe at least three dimensions. Base weaknesses are more abstract than variants and more specific than classes; they describe two to three dimensions. Class weaknesses are very abstract; they describe one to two dimensions and are typically not specific about any language or technology. Pillar weaknesses are the highest level of abstraction.

There is a set of taxonomies, called *views*, to help organize the CWEs. Two prominent CWE taxonomies are the Research Concepts (view 1,000) and Development Concepts (view 699). There is also the CWE Weaknesses for Simplified Mapping of Published Vulnerabilities View (view 1,003) that was made to describe the set of CVEs; it contains 127 CWEs.

Binding CVEs, CWEs, and the CVSS

The National Vulnerability Database (NVD)[1] offers a public database that maps all CVE entries to CWEs and CVSS scores. For each CVE, it provides a CVSS score along with the applicable CWE(s) that describe the weakness(es) enabling the vulnerability. The NVD data are the cornerstone of this work, enabling the analysis of the most significant CWEs over the last 10 years.

The MDSE Score

The MDSE equation was designed to balance frequency and severity in ranking the CWEs. The frequency is determined by the number of CVEs that map to a given CWE in the time period of study. The severity is determined by the mean CVSS score for the CVEs mapped to a given CWE. The MDSE score for a CWE is produced by multiplying the normalized frequency by the normalized severity and then multiplying by 100.

Limitation 1: Distribution Differences

The MDSE score appears to equally include both frequency and severity. However, we empirically find that the MDSE equation, in practice, strongly biases frequency over severity.[4] This happens because, even though the equation treats them equally, frequency and severity have very different distributions. This can be illustrated by the analysis of 2019. The frequency distribution has the majority of CWEs at a very low frequency and a few at a very high frequency (somewhat resembling a power-law curve). This can be seen in Figure 1 by looking at how each CWE maps to the *x*-axis (note that most of the yellow dots overlap; there are 102 yellow dots and 20 red triangles). The figure presents the MDSE scores for each CWE and illustrates how (for a top list of size 20) the top-scoring chosen CWEs are exactly the most frequent CWEs.

The severity distribution is more uniform within a limited range. It can be seen in Figure 2 by looking at how the CWEs map to the *x*-axis. This figure shows how the top-MDSE-scoring chosen CWEs do not necessarily map to the CWEs with the highest severity. In fact, only one of the top 10 most severe CWEs made it into the MDSE top 20 list (note that many of the yellow circles lie on top of each other).

Limitation 2: Normalization Error

Figure 2 also reveals that the normalization of the CVSS score does not lead to the expected and desired normalized distribution from 0 to 1. For our data, the range is from .28 to .97. The reason for this is that the mean of the CVSS score for the CVEs that map to a particular CWE has a smaller range than that obtained by the maximum and minimum CVSS scores. This limitation, while of less consequence than the previous, constrains the range of S_i values, thus further lessening the influence that severity has in determining an MDSE score.

The MSSW Score

Our goal is to mitigate the limitations of the MDSE equation. This is done by addressing the distributions and then analyzing them to attempt to balance frequency and severity.

Addressing the MDSE Limitations

To address MDSE limitation 1, we propose a linearization for the normalized frequency. This can be seen in Figure 3. Each value on the *x*-axis represents a particular CWE, ordered from least frequent to most frequent. The lower blue line represents the normalized frequency (i.e., the number of CVEs mapped to a particular CWE). Note the slow increase in frequency up to the 100th CWE, followed by a rapid increase, terminating in an almost vertical line (i.e., a large derivative). This behavior creates big differences among the most frequent CWEs and almost no difference among the infrequent CWEs.

The middle yellow line represents taking the log of the frequency, which helps linearize but still results in an upward curve on the right side. Thus, we apply a double log for further linearization (see the top red line). We note that this approach is not pseudolinear for the most infrequent of CWEs. However, this does not cause problems as our goal is to identify the most significant weakness, and any such CWE must have at least a moderate frequency.

The MSSW equation then multiplies frequency and severity as in the original MDSE equation. However, it multiplies from two distributions with a similar shape for the part of the functions that are of interest. This enables the MSSW equation to more fairly balance frequency and severity in scoring and ranking of a CWE. To address MDSE limitation 2, the MSSW normalizes the severity using the maximum and minimum mean severity values. This gives the distribution a full 0-to-1 range, which is not achieved with the MDSE equation (see Figure 2).

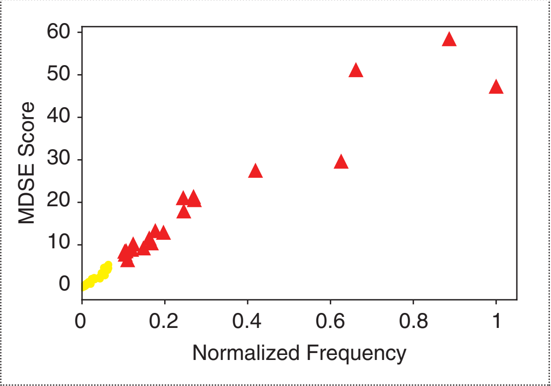

Figure 1. The CWEs chosen (red triangles) and not chosen (yellow circles) for a MDSE top 20 list relative to frequency.

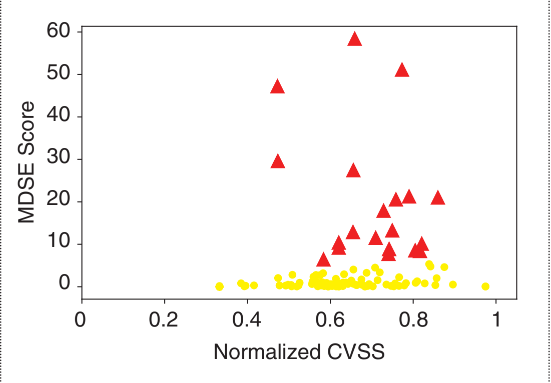

Figure 2. The CWEs chosen (red triangles) and not chosen (yellow circles) for a MDSE top 20 list relative to severity.

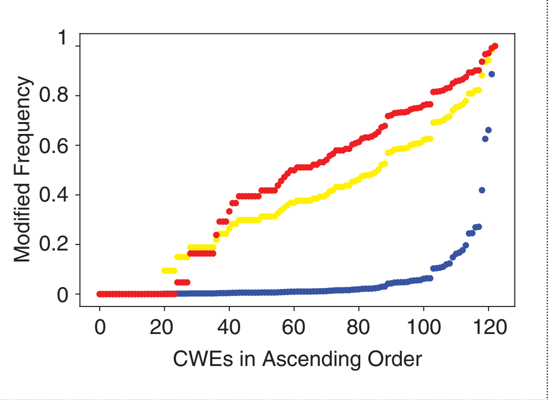

Figure 3. Normalized distributions of frequency (bottom blue line), log of frequency (middle yellow line), and double log of frequency (top red line).

Analyzing the MSSW Equation

We evaluate the effect of the MSSW equation in making the frequency and severity distributions more similar, producing a score with more equal inclusion of both frequency and severity. Figure 4 shows the MSSW scores plotted against the double log frequency scores. Each point represents a CWE. The red triangles indicate the CWEs that were chosen for the MSSW top 20 list. Note how, unlike in the analogous Figure 1 for the MDSE equation, many higher frequency CWEs are not chosen for the top 20 list because their severity is not high enough. Likewise, Figure 5 shows the MSSW scores plotted against the normalized mean CVSS score for each CWE. Note how the range spreads from 0 to 1, unlike in the analogous Figure 2 for the MDSE equation. Also, note how the MSSW equation chooses CWEs for the top 20 list from CWEs with generally higher CVSS scores. However, it excludes many high-severity CWEs because their frequencies are too low.

Figure 6 shows an MDSE risk map for the evaluated CWEs. Each red triangle represents a CWE positioned according to its normalized severity and frequency. In general, CWEs toward the upper right are more significant, and those toward the lower left are less significant. Note how most CWEs are squished very close to the *x*-axis as many have a very small frequency. Also, the range of *x*-values is constrained from .37 to .97 (when the normalization should make it from 0 to 1).

Figure 7 shows the same risk map using our double log frequency and our modified severity. Note how the CWEs are now more uniformly spread over the *y*-axis. Also, the range of *x*-axis values is now from 0 to 1. The MSSW equation combines frequency and severity using the values illustrated in Figure 7. It will now more equally combine them than the MDSE equation values did in Figure 6. Finally, note how Figure 7 locates the most significant CWEs (the top 20 list) at the upper right corner (red triangles). This is the expected behavior of a risk map.

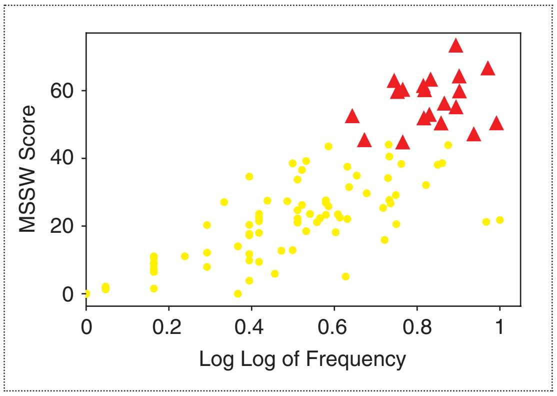

Figure 4. CWEs chosen (red triangles) and not chosen (yellow circles) for a MSSW top 20 list relative to frequency.

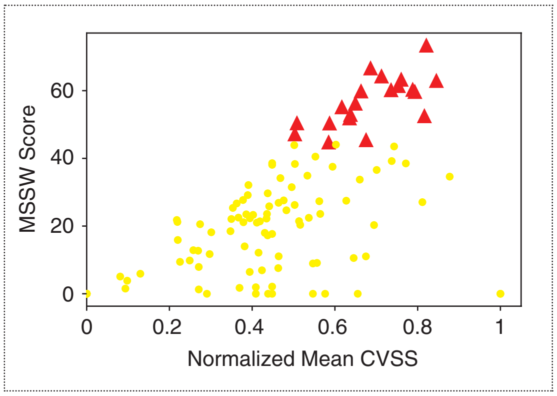

Figure 5. CWEs chosen (red triangles) and not chosen (yellow circles) for a MSSW top 20 list relative to severity.

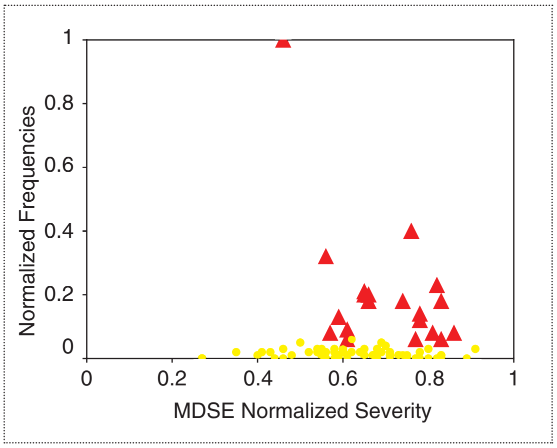

Figure 6. MDSE metric risk map. CWEs chosen (red triangles) and not chosen (yellow circles) for an MDSE top 20 list.

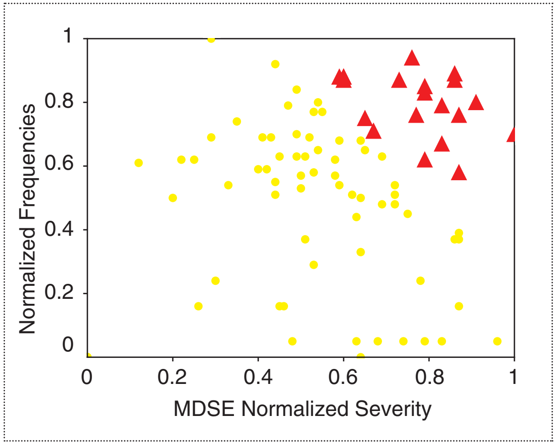

Figure 7. MSSW metric risk map. CWEs chosen (red triangles) and not chosen (yellow circles) for an MSSW top 20 list.

Note that our usage of the double log reflects the nature of the data (observed frequencies) and the desire to fairly balance evaluating frequency and severity in the MDSE equation. If the frequencies had created a different distribution, another "linearization" function would potentially have been better justified. Our main finding is that the MDSE will likely be (and currently is) biased toward one of the parameters (frequency or severity) depending on their relative distributions.

Historical Analysis

We have used our improved scoring equation (MSSW) to perform a historical study of the most significant weaknesses over the 10-year period of 2011–2020. To do so, we collected the 10 CWEs with the highest MSSW value for each year and ranked them in descending order. Our analysis of these lists informs us about the evolution of the software weaknesses landscape so that we can determine if it is changing or static. Our finding is that a similar set of CWEs occupy the top 10 lists each year, and those CWEs can be grouped into an even smaller set of weakness types.

Figure 8(a) shows the top 10 list of CWEs for each year for the base, variant, and compound (BVC) layer. Figure 8(b) shows the same for the pillar and class (PC) layer. Each oval with a number represents a CWE name. The darkness of the oval indicates the number of times that particular CWE has appeared in a top 10 list over the 10-year period. The darker an oval is, the more frequently the corresponding CWE has been in the top 10 lists over the last 10 years. CWEs that appeared less frequently in the lists have a lighter shading.

It can be observed that the figures are rather dark. This indicates that the weaknesses landscape has been dominated by only a few weakness types; this is due to the same CWEs occurring in the top 10 lists each year. Among the 88 possible BVC CWEs, only 19 have appeared in the top 10 lists for the last 10 years (11 of which have appeared at least five times). Similarly, among the 39 possible PC CWEs, only 17 have appeared in the top 10 lists for the same period (nine of which have appeared at least five times). These results show that a minority subset of CWEs has dominated the top 10 lists for the last decade; from this vantage point the software weaknesses landscape is essentially not changing. Instead of seeing a diversity of CWEs entering the top 10 lists, the same kinds of weaknesses reappear year after year.

Two groups of weaknesses dominate the top 10 lists: injection and memory errors. This is illustrated by Figure 9, which shows how the MSSW score in our BVC top 10 lists evolves over the years. The blue line presents the sum of the MSSW score of all CWEs in the BVC top 10 list of each year. The green line shows the sum for memory corruption CWEs, while the red line shows the sum for injection CWEs. The yellow line shows all CWEs that are neither injection nor memory corruption associated. These include CWEs that are related to file management, data authenticity, authentication, and integer arithmetic, which we have put in one group: "other CWEs." The three groups are also shown in Figure 8(a) and (b) by the color of the CWE number inside each oval.

One can observe in Figure 9 a consistent increase in the sum of the MSSW scores of all top 10 BVC CWEs during the last 10 years. This represents a shift where a subset of CWEs increasingly becomes both the most frequent and impactful. Note that this is not due to simply an increase in the number of vulnerabilities discovered because both frequency and impact are normalized within MSSW. One explanation for this trend could simply be that attackers are increasingly leveraging CWEs that give them the greatest influence on targeted systems.

Injection and memory corruption CWEs follow this trend of increasing MSSW scores and they dominate the top 10 lists. Contrasting with Figure 8(a), we can observe that, after 2017, all of the five most dangerous CWEs are related to either injection or memory corruption. After 2019, only two CWEs are outside of those groups in the BVC top 10 lists. This explains the increase of the MSSW score sum for injection and memory corruption CWEs and the decrease of the MSSW score sum for other CWEs.

Injection and Memory Corruption: The Most Dangerous Weaknesses

We now look more closely at these two top CWE groupings. An injection bug happens when an unsanitized input is assembled, added, or inserted in a code fragment or in a command, forming an invalid construct that is not supposed to be executed. In Figure 8(a), injection (red) is represented by CWE-89 (SQL Injection), CWE-78 (OS Command Injection), CWE-94 (Code Injection), CWE-502 (Deserialization of Untrusted Data), CWE-917 (Expression Language Injection), CWE-611 (Improper Restriction of XML External Entity Reference), CWE-22 (Path Traversal), and CWE-79 (Cross-Site Scripting).

The three most dangerous CWEs form the first subgroup in Figure 8(a): CWE-89, CWE-78, and CWE-94. They all appear with very high MSSW scores every year. SQL Injection is by far the most dangerous weakness in our analysis. It is consistently the number one weakness in every top 10 BVC list, with an average MSSW score of 76.6. It contributes to the class CWE-74 (Improper Neutralization of Special Elements in Output Used by a Downstream Component)

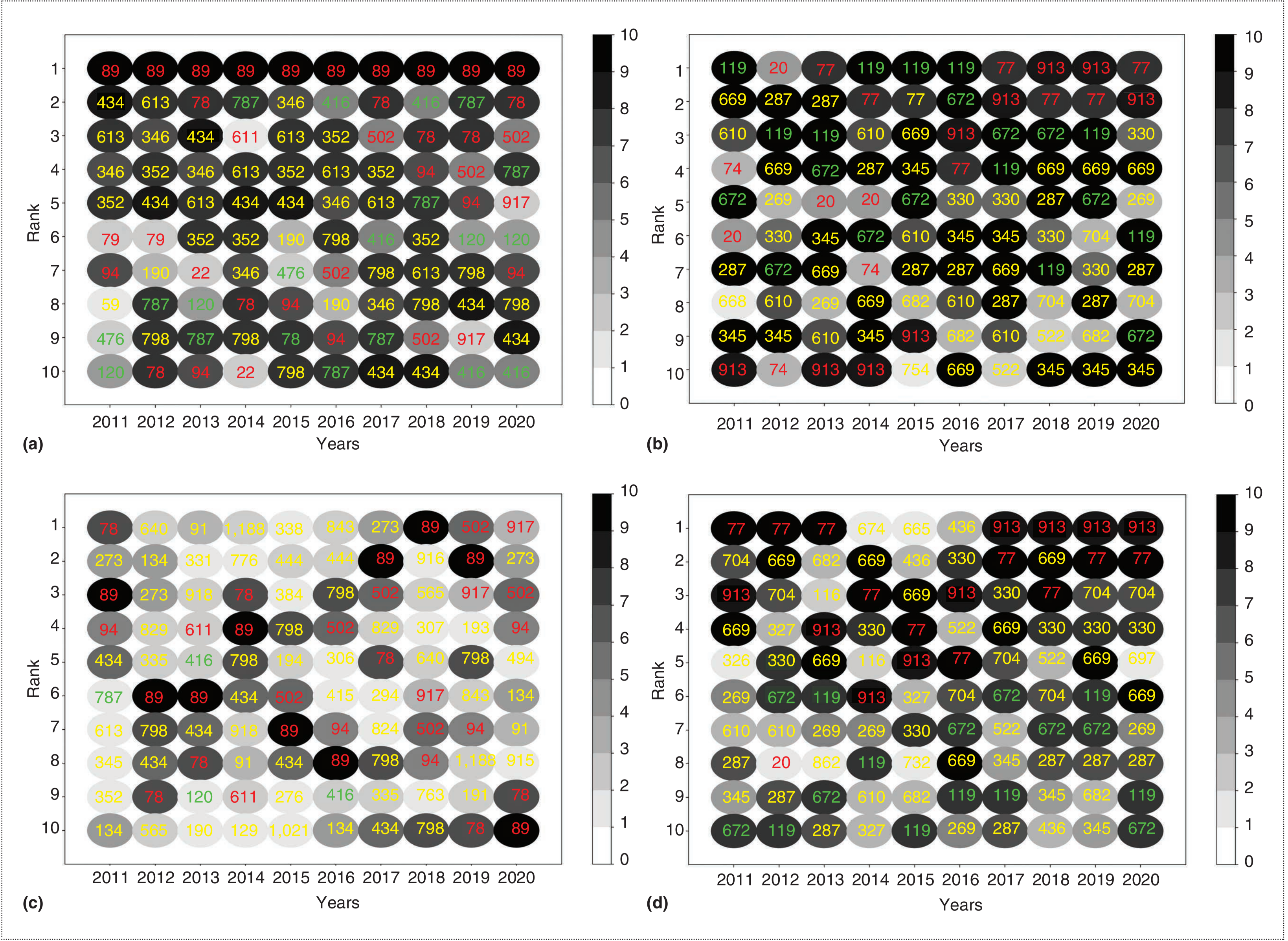

Figure 8. The top 10 CWEs during the last 10 years. The identification is in red for injection CWEs, in green for memory corruption CWEs, and in yellow for all others. The most frequent CWEs are represented by the darkest ovals. (a) and (b) are generated using the MSSW equation; (c) and (d) using the biased MDSE equation. (a) The MSSW top 10 CWEs in the BCW layer. (b) The MSSW top 10 CWEs in the PC layer. (c) The MDSE top 10 CWEs in the BCW layer [compare to Figure 8(a)]. (d) The MDSE top 10 CWEs in the PC layer [compare to Figure 8(b)]. BCW: base, variant, and component; PC: pillar and class.

in Figure 8(b). OS Command Injection is the second most dangerous injection weakness. It appears in every top 10 BVC list except for years 2011 and 2016, with an average MSSW score of 75.18. It is also a contributor to CWE-74 and to CWE-77 [Improper Neutralization of Special Elements used in a Command ('Command Injection') [see Figure 8(b)]]. Code Injection appears in every top 10 BVC list except for years 2012, 2014, and 2017, with an average MSSW score of 66.9. It is also a contributor to CWE-74 and to CWE-913 (Improper Control of Dynamically Managed Code Resources) [see Figure 8(b)]. Interestingly, CWE-74 has a light gray circle in Figure 8(b), while its children base, CWE-89, CWE-78, and CWE-94, have very dark ovals in Figure 8(a). This happens because CWE-74 is also the parent of several CWEs that are either very infrequent or nonsevere.

Deserialization of Untrusted Data (CWE-502) is a new injection weakness. It appears in the BVC top 10 list in all years after 2016 with a high average MSSW score of 72.6. The exploitation of deserialization bugs was leveraged after November 2015, when Foxglove Security published their exploits for the Java deserialization weakness.[8]

A memory corruption bug happens when data stored in memory are unintentionally modified. This could happen during memory allocation, deallocation, or use (read and write data). In Figure 8(a), the memory corruption weaknesses are CWE-787 (Out-of-Bounds Write), CWE-120 (Classic Buffer Overflow), CWE-416 (Use After Free), and CWE-476 (NULL Pointer Dereference). Out-of-Bounds Write is the most dangerous memory corruption weakness. It appears in every top 10 BVC list except for years 2011 and 2015, with an average MSSW score of 70.8. The class CWE-119 is a general memory corruption weakness, which includes use after free and double free. All memory CWEs on the top 10 lists contribute to the class CWE-119, except CWE-476, contributing to CWE-672. Due to its broad scope, CWE-672 is also the parent of CWE-613.

SQL Injection and OS Command Injection weaknesses have a higher average MSSW score than that of any other weakness. The related CVE analysis confirms that the injection CVEs are easier to exploit and have a higher impact. An injection directly leads to arbitrary command, code, or script execution. Once an SQL injection is in place, there is no need of an additional sophisticated attack crafting or the use of glitches in the system. By contrast, it takes considerable extra effort for an attacker to turn a buffer overflow into an arbitrary code execution. The possible damage from an SQL injection is also very high. It may expose vast amounts of structured data, which are generally more valuable than raw data. Well-formed structured data are easy to read, sort, search, and make sense of. An attacker could modify a database—insert data, update data, delete data, execute administration operations, recover file content, and even issue OS commands to the operating system.

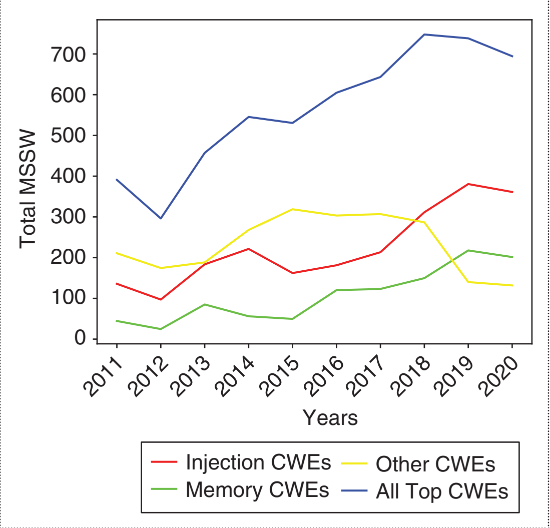

Figure 9. The sum of the MSSW scores of all CWEs in the BVC top 10 list of each year.

Mapping Dependencies

Our historical analysis depends heavily on how the NVD assigns CWEs to particular CVEs, and it is not always possible for this mapping to be done perfectly. The CWE selection is restricted to view CWE-1003. The lack of enough information about a CVE or the lack of a more specific CWE may lead to the CVE described with the closest class CWE or even with a pillar CWE. For example, it makes sense for class CWE-119 to be used for the memory corruption CVE-2019-7098 as there is not much information, neither code nor details about it—it could be any memory use error or a double free. However, there is enough information about the use after free CVE-2019-15554, and it still gets described with class CWE-119, as there is no appropriate base CWE. A close base CWE is CWE-416 (Use After Free), but it does not really reflect memory-safe languages like Rust. It is also possible for a class CWE to be assigned to a CVE even when a specific base CWE is available. For example, the stack buffer overflow write CVE-2019-14363 is assigned to class CWE-119, although there is plenty of information and appropriate bases CWE-121 and CWE-120.

No Ground Truth

The constant need to improve information security has motivated a widespread interest in metrics.[9] As Lord Kelvin's famous quote suggests, "You cannot improve if you cannot measure."

In hard sciences, such as physics and chemistry, a measurement is an experimental procedure that compares a quantity against a well-defined standard. A measurement must be reproducible, allowing the comparison of results over different conditions, such as using different measuring systems. It allows the claim that a measurement result is wrong by showing that it disagrees with other measurements (i.e., it is irreproducible).

In software security, we do not only lack experimental procedures and stable standards. We are in a much earlier stage of science, where we struggle to agree on the quantities to measure. There are members of the software security community who doubt that security can be measured. They argue that software security metrics may be infeasible,[10] difficult to validate, not justified with empirical data, and contain formal treatments adopted from other fields that don't map well to operational security.[11]

We understand that, at least for now, there is no ground truth, and the science of security is still in its early days. However, we posit that acceptable but possibly imperfect metrics must be developed to facilitate security decisions and to evaluate changes in security posture. To this end, there have been substantial efforts to produce security metrics; Verendel[11] surveyed the literature of security metrics published between 1981 and 2008. Specific to software security, there is the Open Web Application Security Project Top 10 for web applications. Also, the CWE project has the Common Weakness Scoring System[12] and the Common Weakness Risk Analysis Framework,[13] which are used together to provide the most important weaknesses tailored to a particular organization.

Given that there is no ground truth upon which to justify how to best combine frequency and severity or to establish the CVSS metric's correctness, it is likely impossible to prove any such metric as maximally effective. We make our "most accurate measurement yet" claim based on the demonstrated limitations in the published MDSE equation and a lack of competing published alternatives. Along with much other work, we believe that our contribution is significant. It points out a severe bias in the CWE MDSE equation that prevents accurate measurements of the most significant software security weaknesses.

Security metrics are a challenging scientific research area because there is often no ground truth, unlike in subjects like physics and chemistry. This may lead one to focus on just taking simple low-level measurements that are inherently defensible; that was the approach taken in the work of Mell and Gueye.[14] However, creating aggregate metrics that compose multiple simple measurements is of practical importance for the field of security. In this work, we did just that, aggregating frequency and severity (i.e., exploitability and impact) into a single metric. Our objective was not for the correlations to necessarily be equal, but to demonstrate that there is a strong correlation for both factors that more evenly balance the inclusion of the top frequency and top severity CWEs.

Using the proposed equation, we explored the software weaknesses landscape. We observed that, in 10 years, the same types of weaknesses have been dominating the threat landscape, and not much has changed. Through the lens of the metrics in this article, we aren't making progress. We believe that the security community needs new approaches. We would prefer not to write this same article 10 years from now, showing that, once again, not much has changed.

It is challenging to catch up with hackers; they need to find only one weak spot, while we (the community) have to defend entire systems. New doors also get opened (e.g., in recent years object deserialization injection). Nevertheless, the results of this study show that either we are incapable of correcting the most common software flaws, or we are focusing on the wrong ones. Although this article is not making a definite conclusion, the comparison with the historical analysis based on the biased MDSE [see Figure 8(c) and (d)] suggests that it is the latter. In either case, it seems to us that there is a need to "stop and think" about the ways we are developing software and/or the methods we use to describe and identify vulnerabilities. A new unambiguous classification of software weaknesses that allows clear structured descriptions of security vulnerabilities would be a first step.[15] That would allow formalization and automatization of weaknesses identification and vulnerabilities mitigation. Operationally, more software development languages and tools need to be developed and/or promoted that automatically prevent or remediate commonly identified software weaknesses. ■

Acknowledgment

This work was partially accomplished under NIST Cooperative Agreement 70NANB19H063 with Prometheus Computing, LLC. The authors would like to thank the NVD staff for their review and consideration of this work.

References

1. "National vulnerability database," NIST, Gaithersburg, MD, 2020. Accessed: Jan. 10, 2020. [Online]. Available: https://nvd.nist.gov
2. "Common weakness enumeration," MITRE, Bedford, MA, 2019. Accessed: Dec. 10, 2019. [Online]. Available: https://cwe.mitre.org

3. "2019 CWE top 25 most dangerous software errors," MITRE, Bedford, MA, 2020. Accessed: Feb. 10, 2020. [Online]. Available: https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html
4. C. C. Galhardo, P. Mell, I. Bojanova, and A. Gueye, "Measurements of the most significant software security weaknesses," in *Proc. Annu. Comput. Security Appl. Conf. (ACSAC)*, 2020, pp. 154–164.
5. R. S. Ross, "Guide for conducting risk assessments," NIST, Gaithersburg, MD, 2012. Accessed: Jan. 10, 2020. [Online]. Available: https://www.nist.gov/publications/guide-conducting-risk-assessments
6. "Common vulnerabilities and exposures," MITRE, Bedford, MA, 1999. Accessed: Feb. 5, 2020. [Online]. Available: https://cve.mitre.org
7. "Common vulnerability scoring system special interest group," FIRST, 2019. Accessed: Dec. 10, 2019. [Online]. Available: https://www.first.org/cvss
8. L. Raghavan, "Lessons learned from the java deserialization bug," Medium, 2016. Accessed: Feb. 22, 2021. [Online]. Available: https://medium.com/paypal-engineering/lessons-learned-from-the-java-deserialization-bug-cb859e9c8d24
9. D. S. Herrmann, *Complete Guide to Security and Privacy Metrics: Measuring Regulatory Compliance, Operational Resilience, and ROI*, 1st ed. Boca Raton, FL: Auerbach Publications, 2007.
10. S. M. Bellovin, "On the brittleness of software and the infeasibility of security metrics," *IEEE Security Privacy*, vol. 4, no. 4, p. 96, July 2006. doi: 10.1109/MSP.2006.101.
11. V. Verendel, "Quantified security is a weak hypothesis: A critical survey of results and assumptions," in *Proc. 2009 Workshop on New Security Paradigms Workshop, ser. NSPW '09*, New York: Association for Computing Machinery, pp. 37–50. doi: 10.1145/1719030.1719036.
12. "Common weakness scoring system (CWSS)," MITRE, Bedford, MA, 2018. Accessed: Apr. 10, 2020. [Online]. Available: https://cwe.mitre.org/cwss/
13. "Common weakness risk analysis framework (CWRAF)," MITRE, Bedford, MA, 2019. Accessed: Apr. 10, 2020. [Online]. Available: https://cwe.mitre.org/cwraf/
14. P. Mell and A. Gueye, "A suite of metrics for calculating the most significant security relevant software flaw types," in *Proc. 2020 Conf. Computers, Software Appl. (COMPSAC)*, Madrid, Spain: IEEE Computer Society Press, pp. 511–516. doi: 10.1109/COMPSAC48688.2020.0-201.
15. "The bugs framework (BF)," NIST, Gaithersburg, MD, 2020. Accessed: May 11, 2020. [Online]. Available: https://samate.nist.gov/BF/
16. "Common vulnerability scoring system version 3.1: Specification document," FIRST, NC, 2019. [Online]. Available: https://www.first.org/cvss/specification-document

Assane Gueye is an assistant teaching professor at Carnegie Mellon University Africa, Kigali, Rwanda, BP 6150, Africa and codirector of the CyLab-Africa Initiative. He also holds a guest researcher position with the National Institute of Standards and Technology. His research interests include cybersecurity, security and resilience of large-scale systems, communication networks, information and communication technologies for development, blockchain and cryptocurrencies, and machine learning. Gueye received a Ph.D. in electrical engineering and computer science from the University of California Berkeley. He is a fellow of the Next Einstein Forum, a fellow of the European Alliance for Innovation, and a member of the Science Advisory Committee of the Future Africa Research Leadership Fellowship. Contact him at assaneg@andrew.cmu.edu.

Carlos E.C. Galhardo is a researcher at the Brazilian National Institute of Metrology, Quality, and Technology, Rio de Janeiro, 25250, Brazil. His research interests include data analysis and mathematical modeling in interdisciplinary applications and models and methods to analyze security weaknesses (program analysis). Galhardo received a Ph.D. in computational physics from Universidade Federal Fluminense. He worked at the National Institute of Standards and Technology as a guest researcher. Contact him at cegalhardo@inmetro.gov.br.

Irena Bojanova is a computer scientist at the National Institute of Standards and Technology, Gaithersburg, Maryland, 20899, USA, where she is the lead of the Bugs Framework. Her research interests are in formal methods, distributed systems, and program analysis for security. Bojanova received a Ph.D. in mathematics/computer science from the Bulgarian Academy of Sciences. She is a Senior Member of IEEE and serves as the editor-in-chief of *IT Professional* magazine and as the editor of the "Education" column in *Computer* magazine. Contact her at irena.bojanova@nist.gov.

Peter Mell is a senior computer scientist in the Computer Security Division at the National Institute of Standards and Technology, Gaithersburg, Maryland, 20899, USA. His research interests include the areas of vulnerability databases, intrusion detection, computer penetration, computer and network security, cryptocurrencies, algorithmic complexity, and graph theory. Mell received an M.S. in computer science from the University of California at Davis. Relevant to this article, Mell invented the U.S. National Vulnerability Database and was a founding editorial board member of the Common Vulnerabilities and Exposures project. Contact him at peter.mell@nist.gov.