

---

# A DECADE OF REOCCURRING SOFTWARE WEAKNESSES

---

THIS IS A PREPRINT OF AN ARTICLE ACCEPTED FOR PUBLICATION IN IEEE SECURITY AND PRIVACY

**Assane Gueye**

Carnegie Mellon University Africa  
assaneg@andrew.cmu.edu

**Carlos E. C. Galhardo**

National Institute of Metrology, Quality, and Technology  
cegalhardo@inmetro.gov.br

**Irena Bojanova**

National Institute of Standards and Technology  
irena.bojanova@nist.gov

**Peter Mell**

National Institute of Standards and Technology  
peter.mell@nist.gov

## ABSTRACT

The Common Weakness Enumeration (CWE) community publishes an aggregate metric to calculate the ‘Most Dangerous Software Errors.’ However, the used equation highly biases frequency over exploitability and impact. We provide a metric to mitigate this bias and discuss the most significant software weaknesses over the last ten years.

## 1 Introduction

In 2020, there were over 18 000 documented software vulnerabilities [1] that enable malicious activity. While many are 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 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 set of CWEs [3]. It follows the form of the standard security risk matrix combining probability/frequency and severity.

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, named MSSW, that adjusts the frequency distribution using a double log function to better match it to the severity distribution.

## 2 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. They are enumerated with a unique identifier, described, and referenced with external advisories [6].

## 2.1 Scoring vulnerabilities severity

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:

- 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’, and
- 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’.

## 2.2 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 Common Weakness Enumeration (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 labelled 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, that 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, typically not specific about any language or technology. Pillar weaknesses are the highest level of abstraction.

There are a set of taxonomies, called views, to help organize the CWEs. Two prominent CWE taxonomies are the ‘Research Concepts’ (view 1000) and ‘Development Concepts’ (view 699). There is also the ‘CWE Weaknesses for Simplified Mapping of Published Vulnerabilities View’ (view 1003) that was made to describe the set of CVEs; it contains 127 CWEs.

## 2.3 Binding CVEs, CWEs, and 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 is the cornerstone of this work, enabling the analysis of the most significant CWEs over the last ten years.

## 3 The MDSE score

The MDSE equation was designed to balance the 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.

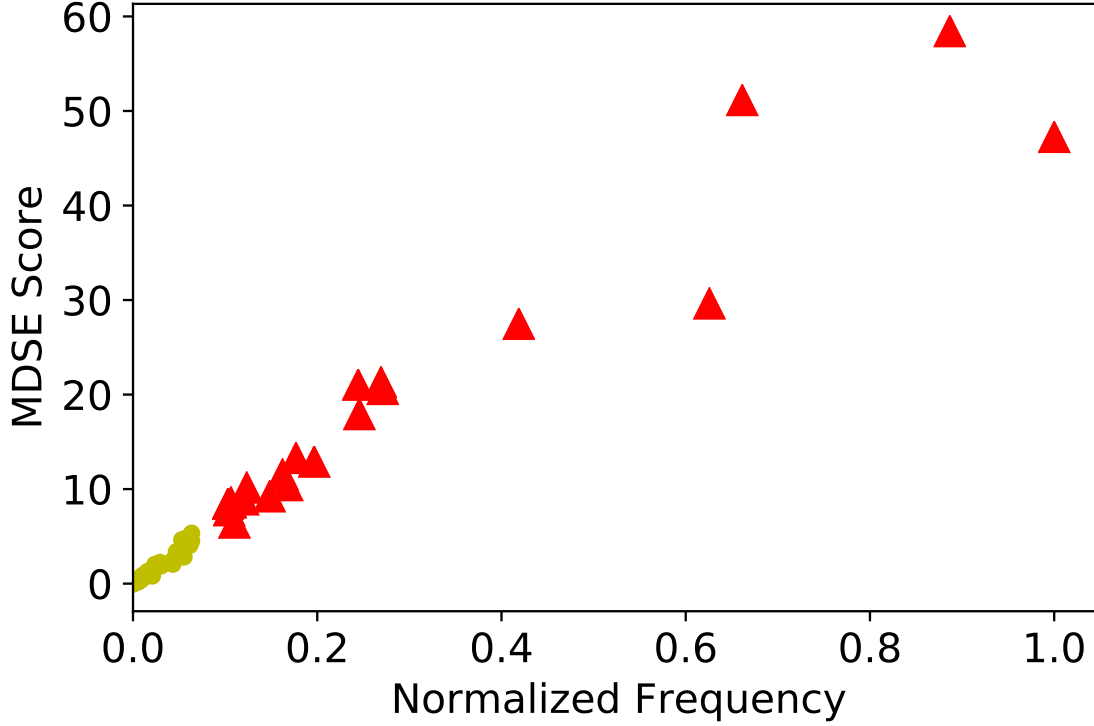


Figure 1: CWEs Chosen (Red Triangles) and Not Chosen (Yellow Circles) for a MDSE Top 20 List Relative to Frequency.

### 3.1 Limitation 1: Distribution Differences

The MDSE score then appears to equally include both frequency and severity. However, we empirically find that the MDSE equation strongly biases frequency over severity [4]. The MDSE equation in practice biases frequency over severity, even though its equation treats them equally, because 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 shows the MDSE scores for each CWE and shows 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 1 of the top 10 most severe CWEs made it into the MDSE top 20 list (note that many of the yellow circles lay on top of each other).

### 3.2 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 the maximum and minimum CVSS score. 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 a MDSE score.

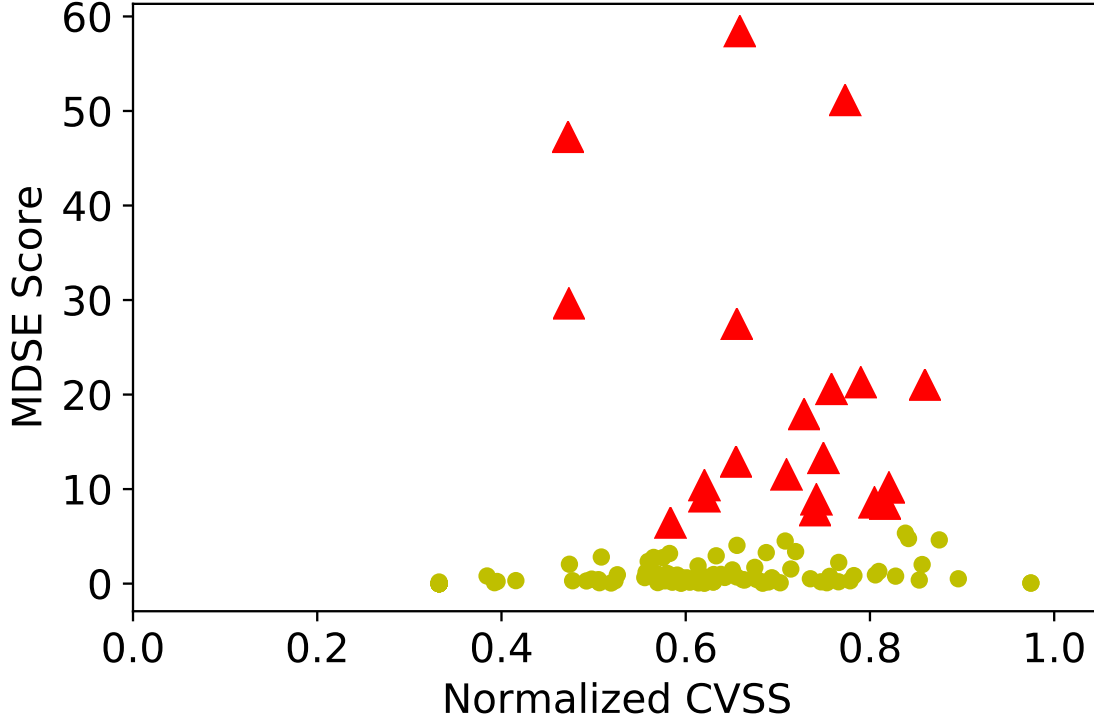


Figure 2: CWEs Chosen (Red Triangles) and Not Chosen (Yellow Circles) for a MDSE Top 20 List Relative to Severity.

## 4 The MSSW score

Our goal is to mitigate the limitations of the MDSE equation.

### 4.1 Addressing the MDSE’s 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., large derivative). This behavior creates large differences between the most frequent CWEs and almost no difference between the infrequent CWEs.

The middle yellow line represents taking the log of the frequency, which helps linearize but still results in an upwards 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 pseudo-linear for the most infrequent of CWEs. However, this does not cause problems as our goal is to identify the most significant 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 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 in the MDSE equation (see Figure 2).

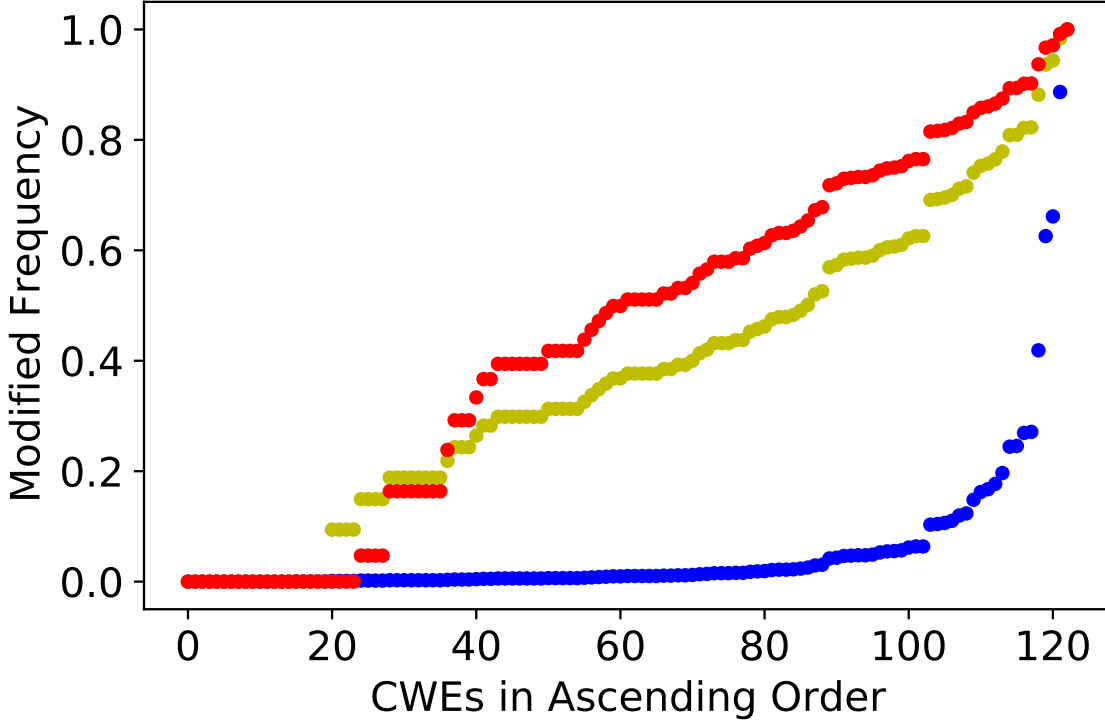


Figure 3: Normalized Distributions of Frequency (bottom blue line), Log of Frequency (middle yellow line), and Double Log of Frequency (top red line).

#### 4.2 Analyzing the MSSW

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 MDSE, many higher frequency CWEs are not chosen for the top 20 list due to their severity not being 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 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 were too low.

Figure 6 shows an MDSE risk map for the evaluated CWEs. Each red dot represents a CWE positioned according to its normalized severity and frequency. In general, CWEs towards the upper right are more significant and those towards 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 shown in Figure 7. It will now more equally combine them than with the MDSE values shown in Figure 6. Finally, note how Figure 7 locates the most significant CWEs (Top 20 List) at the upper right corner (red triangle). This is the expected behavior of a risk map.

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

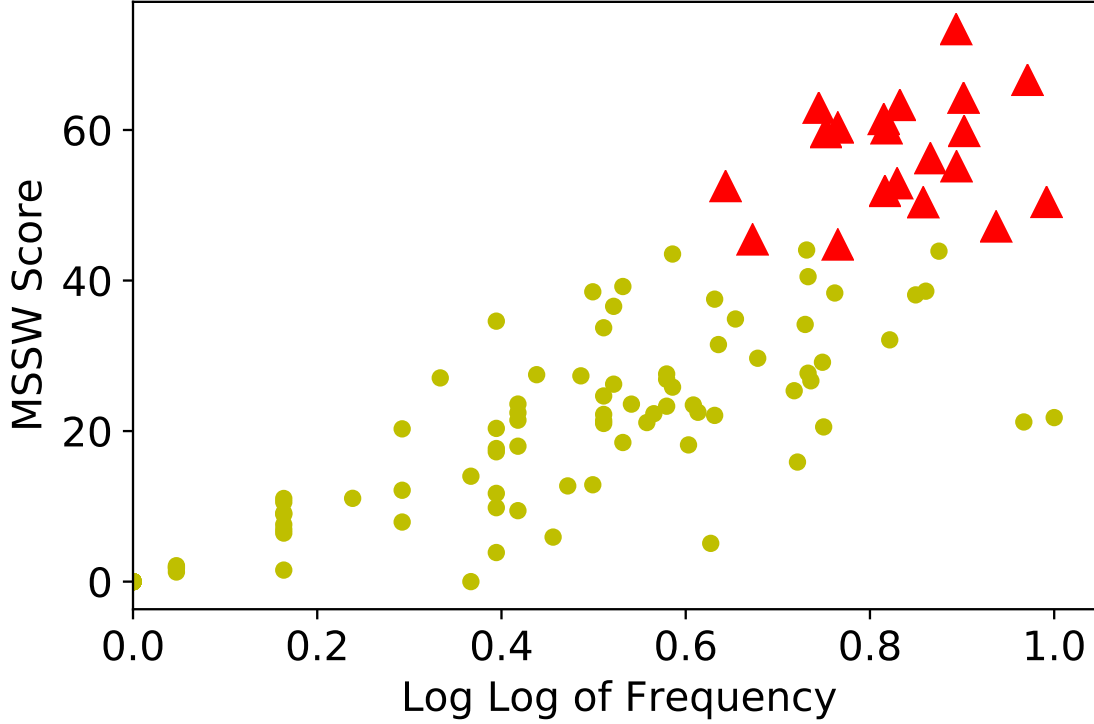


Figure 4: CWEs Chosen (Red Triangles) and Not Chosen (Yellow Circles) for a MSSW Top 20 List Relative to Frequency.

likely be (and currently is) biased toward one of the parameters (frequency or severity) depending on their relative distributions.

## 5 Historical Analysis

We use our improved scoring equation (MSSW) to perform a historical study of the most significant weaknesses over the ten-year period of 2011 to 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 8a shows the Top 10 list of CWEs for each year for the Base, Variant, and Compound (BVC) layer. Figure 8b 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 ten-year period. The darker an oval

is, the more frequent 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; it 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 5 times). Similarly, among the 39 possible PC CWEs, only 17 have appeared in the Top 10 lists for the same period (9 of which have appeared at least 5 times). These results show that a minority subset of CWEs have dominated the

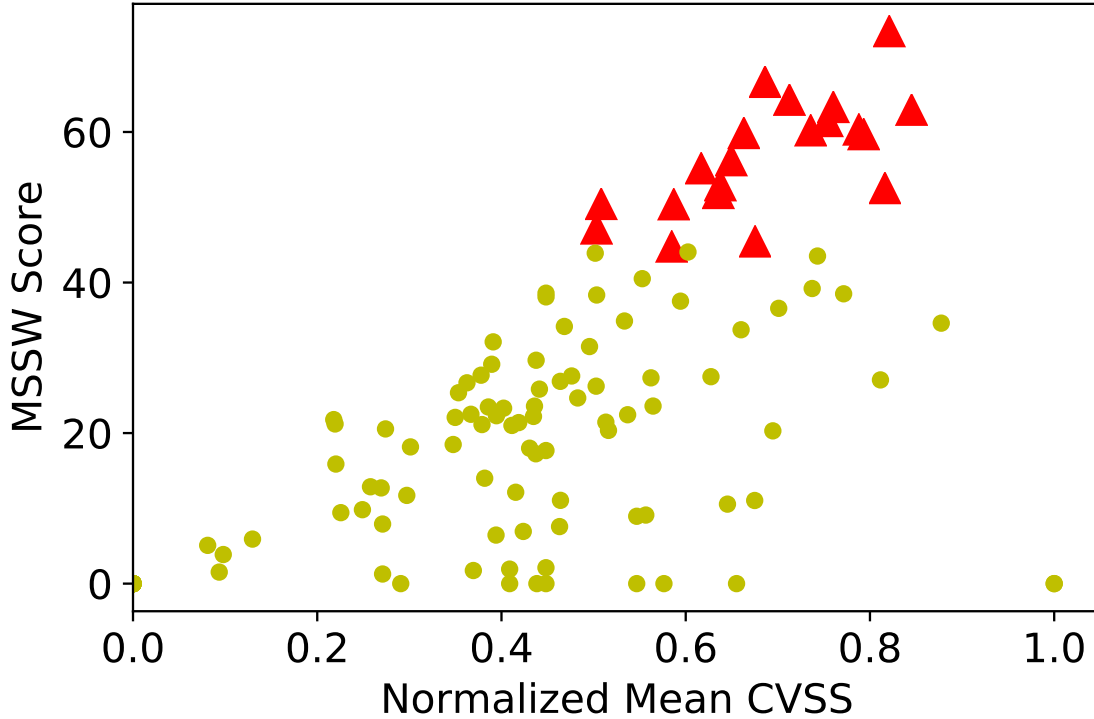


Figure 5: CWEs Chosen (Red Triangles) and Not Chosen (Yellow Circles) for a MSSW Top 20 List Relative to Severity.

Top 10 lists for the last decade; from this vantage point the software weaknesses landscape is practically 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. 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 Figures 8a and 8b 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 ten years. This represents a shift where a subset of CWEs increasingly become 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 8a, we can observe that after 2017, all the five most dangerous CWEs are either injection or memory corruption. After 2019, only 2 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.

### 5.1 Injection and Memory Corruption: The Most Dangerous Weaknesses

We now look more closely at these 2 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

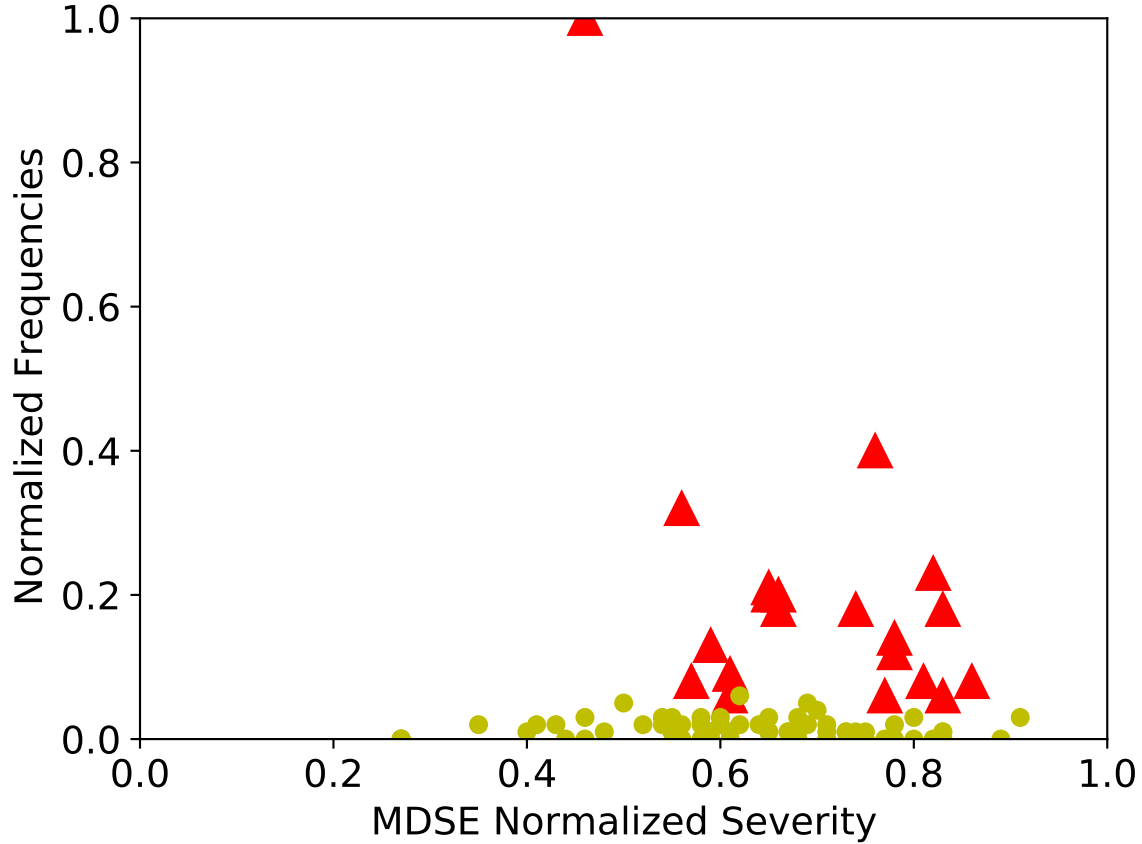


Figure 6: MDSE Metric Risk Map. CWEs Chosen (Red Triangles) and Not Chosen (Yellow Circles) for a MDSE Top 20 List.

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

Deserialization of untrusted data (CWE-502) is a considerably 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].



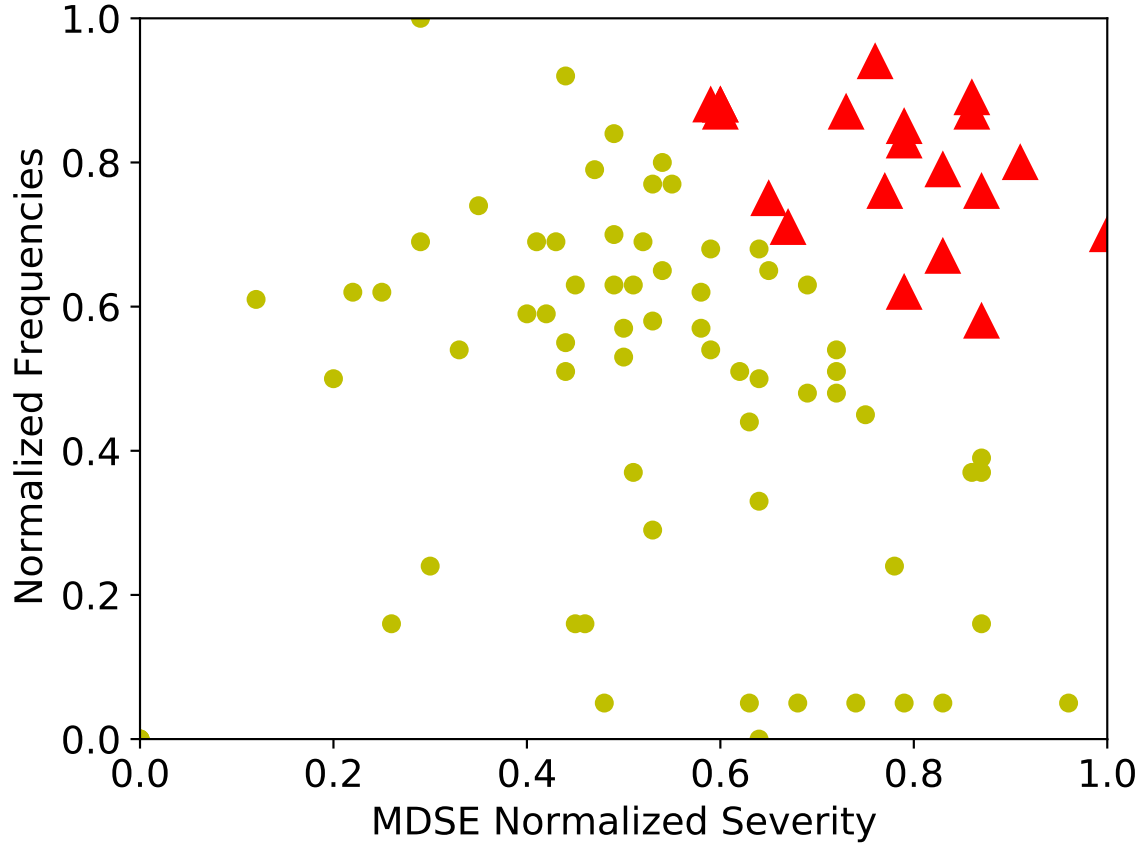


Figure 7: MSSW Metric Risk Map. CWEs Chosen (Red Triangles) and Not Chosen (Yellow Circles) for a MSSW Top 20 List.

A memory corruption bug happens when data stored in memory is unintentionally modified. This could happen during memory allocation, deallocation, and use (read and write data). In Figure 8a, 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 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 weaknesses. The related CVEs 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 a SQL injection is in place, there is no need of an additional sophisticated attack crafting or use of glitches in the system. In contrast, it takes considerable extra effort for an attacker to turn a buffer overflow into an arbitrary code execution. The possible damage from a SQL injection is also very high. It may expose vast amounts of structured data, which is generally more valuable than raw data. Well-formed structured data is easy to read, sort, search, and make sense of. An attacker could modify a database – insert data, update data, delete data, execute admin operations, recover file content, and even issue OS commands to the operating system.

## 5.2 Mapping Dependencies

Our historical analysis heavily depends on how 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

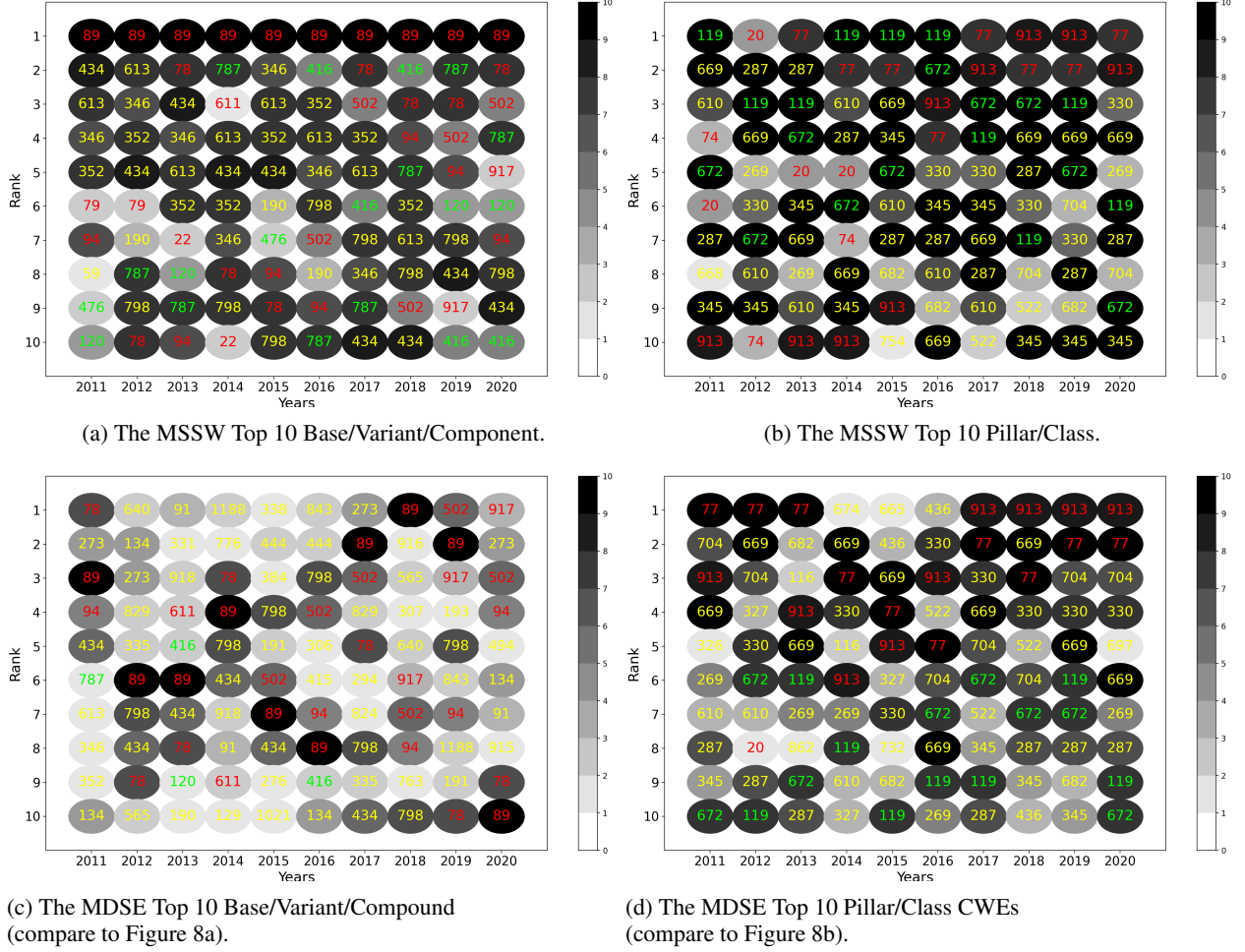


Figure 8: The top 10 CWE during the last 10 years. ID is in red for injection CWEs, in green - for memory corruption CWEs, and in yellow - for all others. The most frequent CWEs are represented by darkest ovals. Top figures are generated using MSSW, bottom figures - using the biased MDSE.

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, no 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 class CWE-119, although there is plenty of information and appropriate bases CWE-121 and CWE-120.

## 6 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 results comparison 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., irreproducible).

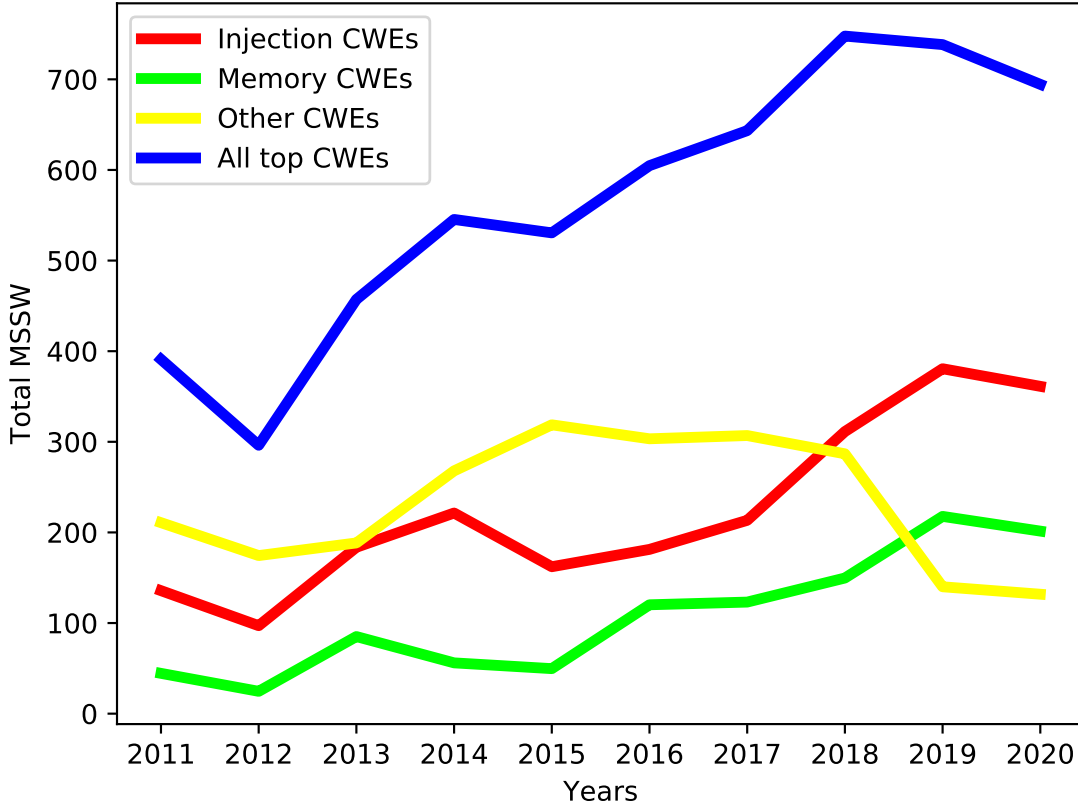


Figure 9: The sum of the MSSW score of all CWEs in the BVC Top 10 list of each year.

In software security, we lack not only 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], be difficult to validate, not be 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 in order to facilitate security decisions and to evaluate changes in security posture. To this end, there have been substantial efforts to produce security metrics; [11] surveys the literature of security metrics published between 1981 and 2008. Specific to software security, there is the OWASP Top 10 for web applications. Also, the CWE project has the Common Weaknesses Scoring System (CWSS) [12] and the Common Weakness Risk Analysis Framework (CWRAF) [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.

## 7 Conclusion

Security metrics are a challenging scientific research area, because there is often no ground truth, unlike 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 [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 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 explore 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 paper we aren't making progress. We believe that the security community needs new approaches. We would prefer not to write this same paper 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 paper is not making a definite conclusion, the comparison with the historical analysis based on the biased MDSE (see Figures 8c and 8d) suggests that it is rather 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 weakness.

## Acknowledgment

This work was partially accomplished under NIST Cooperative Agreement No.70NANB19H063 with Prometheus Computing, LLC. The authors would like to thank the NVD staff for their review and consideration of this work.

## References

- [1] NIST, "National vulnerability database," 2020, accessed: 2020-01-10. [Online]. Available: <https://nvd.nist.gov>
- [2] MITRE, "Common weakness enumeration," 2019, accessed: 2019-12-10. [Online]. Available: <https://cwe.mitre.org>
- [3] MITRE, "2019 cwe top 25 most dangerous software errors," 2020, accessed: 2020-02-01. [Online]. Available: [https://cwe.mitre.org/top25/archive/2019/2019\\_cwe\\_top25.html](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 *Annual Computer Security Applications Conference (ACSAC)*, 2020, pp. 154–164
- [5] Ronald S. Ross, "Guide for conducting risk assessments," 2012, accessed: 2020-01-10. [Online]. Available: <https://www.nist.gov/publications/guide-conducting-risk-assessments>
- [6] MITRE, "Common vulnerabilities and exposures," 1999, accessed: 2020-2-5. [Online]. Available: <https://cve.mitre.org>
- [7] FIRST, "Common vulnerability scoring system special interest group," 2019, accessed: 2019-12-10. [Online]. Available: <https://www.first.org/cvss>
- [8] L. Raghavan, "Lessons learned from the java deserialization bug," 2016, accessed: 2021-02-22. [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", in *1st ed. USA: Auerbach Publications*, 2007.
- [10] S. M. Bellovin, "On the brittleness of software and the infeasibility of security metrics," in *IEEE Security and Privacy*, vol. 4, no. 4, p. 96, Jul. 2006. [Online]. Available: <https://doi.org/10.1109/MSP.2006.101>

- [11] V. Verendel, “Quantified security is a weak hypothesis: A critical survey of results and assumptions,” in *Proceedings of the 2009 Workshop on New Security Paradigms Workshop, ser. NSPW '09*, New York, NY, USA: Association for Computing Machinery, 2009, p. 37–50. [Online]. Available: <https://doi.org/10.1145/1719030.1719036>
- [12] MITRE, “Common weakness scoring system (CWSS),” 2018, accessed: 2020-04-10. [Online]. Available: <https://cwe.mitre.org/cwss/>
- [13] MITRE, “Common weakness risk analysis framework (CWRAF),” 2019, accessed: 2020-04-10. [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 *2020 Conference on Computers, Software and Applications (COMPSAC)*, IEEE, Madrid, Spain: IEEE Computer Society Press, 2020.
- [15] NIST, “The bugs framework (BF),” 2020, accessed:2020-05-11. [Online]. Available: <https://samate.nist.gov/BF/>

**Asane Gueye** is an Assistant Teaching Professor at Carnegie Mellon University Africa (CMU-Africa) and co-Director of the CyLab-Africa Initiative. He also holds a Guest Researcher position with the National Institute for Standards and Technology (NIST). His research interest includes cybersecurity, security and resilience of large-scale systems, communication networks, information and communication technologies for development (ICT4D), blockchain and crypto-currencies, and machine learning. Gueye completed a Ph.D. in EECS from UC Berkeley in March 2011 and a master’s degree in communication systems engineering from Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland. He is a Fellow of the Next Einstein Forum (NEF), Fellow of the European Alliance for Innovation (EAI) and a member of the Science Advisory Committee of the Future Africa Research Leadership Fellowship (FAR-LEAF). Contact him at [assaneg@andrew.cmu.edu](mailto:assaneg@andrew.cmu.edu).

**Carlos E. C. Galhardo** is a researcher at the Brazilian National Institute of Metrology, Quality and Technology, Inmetro. He also worked at NIST as a guest researcher from fall 2019 to fall 2020. He earned his Ph.D. in Computational Physics in 2010 from Universidade Federal Fluminense. His research interests include data analysis and mathematical modeling in interdisciplinary applications. Nowadays, he is working on models and methods to analyze security weaknesses (program analysis). Contact him at [cegalhardo@inmetro.gov.br](mailto:cegalhardo@inmetro.gov.br).

**Irena Bojanova** is a computer scientist at the National Institute of Standards and Technology (NIST), where she is the lead of the Bugs Framework (BF). Her research interests are in formal methods, distributed systems, and program analysis for security. Irena received her 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 the *IT Professional* magazine and as the Editor of the Education column in *IEEE Computer* magazine. Contact her at [irena.bojanova@nist.gov](mailto:irena.bojanova@nist.gov).

**Peter Mell** is a senior computer scientist in the Computer Security Division at the National Institute of Standards and Technology (NIST). His research interests include the areas of vulnerability databases, intrusion detection, computer penetration, computer and network security, cryptocurrencies, algorithmic complexity, and graph theory. Relevant to this paper, Mell invented the United States National Vulnerability Database and was a founding editorial board member of the Common Vulnerabilities and Exposures project. Peter received an MS in computer science from the University of California at Davis in 1998. Contact him at [peter.mell@nist.gov](mailto:peter.mell@nist.gov).