Gotta Catch ’em All: Aggregating CVSS Scores

Ángel Longueira-Romero
Industrial Cybersecurity
Ikerlan Technology Research Centre (BRTA)
Arrasate/Mondragón, Spain
alongueira@ikerlan.es

Jose Luis Flores
Industrial Cybersecurity
Ikerlan Technology Research Centre (BRTA)
Arrasate/Mondragón, Spain
jlflores@ikerlan.es

Rosa Iglesias
Industrial Cybersecurity
Ikerlan Technology Research Centre (BRTA)
Arrasate/Mondragón, Spain
riglesias@ikerlan.es

Iñaki Garitano
Dept. of Electronics and Computing
Mondragon Unibertsitatea
Arrasate/Mondragón, Spain
igaritano@mondragon.edu

***Abstract*—Security metrics are not standardized, but international proposals such as the Common Vulnerability Scoring System (CVSS) for quantifying the severity of known vulnerabilities are widely used. Many CVSS aggregation mechanisms have been proposed in the literature. Nevertheless, factors related to the context of the System Under Test (SUT) are not taken into account in the aggregation process; vulnerabilities that in theory affect the SUT, but are not exploitable in reality. We propose a CVSS aggregation algorithm that integrates information about the functionality disruption of the SUT, exploitation difficulty, existence of exploits, and the context where the SUT operates. The aggregation algorithm was applied to OpenPLC V3, showing that it is capable of filtering out vulnerabilities that cannot be exploited in the real conditions of deployment of the particular system. Finally, because of the nature of the proposed algorithm, the result can be interpreted in the same way as a normal CVSS.**

***Index Terms*—CVSS, security metrics, aggregation, attack graphs, vulnerabilities.**

I. Introduction

System security quantification is not an easy task [1]. There exist both a lack of consensus and standardization around security metrics [2], [3], [4], [5], [6], [7], [8]. For this reason, research efforts keep aiming to unify this field [9].

Among these efforts, the Common Vulnerability Scoring System (CVSS) is a widely extended standard for vulnerability quantification [10]. CVSS is a public framework that provides a standardized method for assigning quantitative values to security vulnerabilities according to their severity. A CVSS score is a decimal number in the range [0, 10][1] [11].

The CVSS is aimed to quantify the severity of vulnerabilities in individual and specific software items, however the majority of systems are actually a composition of simpler isolated items with different interdependencies. This situation highlights one of the biggest problems related to security quantification [12], the difficulty to really measure the global security state of a composite system. To do so, it would be necessary to aggregate each individual CVSS value into a global one in a consistent and coherent way.

The official CVSS documentation does not propose any kind of aggregation mechanism, and nowadays, there is no standardized method [13]. In addition to this, previous research works do not usually integrate contextual or interdependency information about the vulnerabilities to update the CVSS. This means that aspects such as whether affected functionalities, the environment of deployment, or the existence of exploits are usually neglected.

Context is a critical aspect to integrate in the aggregation process. This can be illustrated using a device implementing multiple functionalities as an example. To perform those functionalities, usually it will contain assets that implement those functionalities. But depending on the context where the device is deployed, some of its functionalities might not be needed. So the assets implementing unused functionalities would be disabled, and therefore, their vulnerabilities could not be exploited. It can also be the case that the asset implementing a functionality is simply inaccessible, so it could not also be exploited.

This research proposes a novel aggregation algorithm for a set of CVSS values[2]. This approach is based on the Extended Dependency Graphs (EDGs) proposed by Longueira-Romero *et al.* [14]. Because EDGs are capable of modeling dependencies, this algorithm can also be applied to computer networks. Our proposal is capable of selecting the most relevant CVSS to be aggregated, taking into account four different context-related properties of the System Under Test (SUT):

1) Functionality disruption.
2) Exploitation difficulty.
3) Existence of exploits, and their development state.
4) Context of deployment.

This approach increases the granularity of the CVSS base, environment and temporal metrics, where not every possible

[1]The latest version at the time this paper was written is version 3.1.

[2]The Python code implementing the aggregation algorithm is available at GitHub https://github.com/aaalongueira/CVSS_Aggregation.

value in the scale $[0, 10]$ is achievable, or the result of changing the value of a submetric has almost no effect on the final CVSS [13], [15]. Moreover, our proposal is capable of detecting which branch in the EDG is contributing the most (more critical) to the final score.

This paper is organized as follows: We review existing aggregation methods in Section II. Our proposal is explained in Section III, and tested in a use case in Section IV. Finally, Section V contains the conclusions and future work of this research.

II. Related Work

Nowadays, there is no widely-accepted method to aggregate CVSS values for software composition. All of them can be classified into one of the following categories [16], [17]: (1) Arithmetic Aggregation, (2) Attack Graph-based Aggregation, and (3) Bayesian Network-based Aggregation.

A. Arithmetic Aggregation

This method uses arithmetic operations to aggregate the values [18], [19], [20], [21]. Common examples of this approach are taking the maximum of the CVSS values, their arithmetic mean, or a combination of them. For example, Heyman *et al.* [18], proposed an algorithm to aggregate CVSS values in dependency graph that is based on taking the maximum value in each case, according to certain conditions.

Although their simplicity makes them suitable for initial approximations, their results can be biased in two ways:

1) **Exploitable by quantity:** When a system poses several vulnerabilities that by their own are not critical and cannot be exploited, they can sum up to an aggregated value of a high impact vulnerability (overfitting). This can happen when multiple simple mechanisms are combined as the aggregation algorithm.
2) **Exploitable by criticality:** When there exist a critical vulnerability, the whole system will be usually classified as critical. Nevertheless, that vulnerability might not be exploitable, nor being affecting the functionality of the system. This is specially common when using the maximum as the aggregation algorithm.

B. Attack Graph-based Aggregation

This approach models the relationships between vulnerabilities using attack graphs, converting CVSS scores into probabilities [22], [23], [24], [25], [26], [27]. In this way, both the CVSS value and the place of the vulnerability in the whole graph are taken into account.

Cheng *et al.* in [16] proposed a graph-based aggregation method that uses the underlying metrics of CVSS, where the dependency relationships between vulnerabilities are usually visible. As the center of the aggregation algorithm, they use the product of the CVSS used as probabilities, also known as the join probability of both vulnerability.

The main drawback with these approaches is that the relationship between individual vulnerabilities cannot be obtained straightforwardly from existing databases. This means that establishing a relation between two vulnerabilities implies that they can be chained during an attack, which is not always obvious. Moreover, factors such as exploitability of the vulnerabilities, or existing exploits are not taken into account.

C. Bayesian Network-based Aggregation

Going a step further, these methods integrate the conditional relationship between vulnerabilities, modeling them using Bayesian networks [28], [29], [30]. Poolsappasit *et al.* [29] proposed a CVSS aggregation framework using Bayesian networks. They used the Bayesian probability factorization formula as the aggregation mechanism:

$$p(x) = \prod_{i=0} p(x_v | x_{pa(v)})$$

Bayesian network-based approaches have to deal with establishing the relationships between the vulnerabilities, but also with the calculation of conditional probabilities, that have to be usually estimated. As the previous ones, these techniques do not integrate information about how functionality if affected by existing vulnerabilities, or the possibility to actually exploit them.

III. Proposed Approach for Metric Aggregation

In this paper, we propose a CVSS aggregation algorithm inspired by the risk propagation formula [31] described in MAGERIT [32], [33]. First, we describe the corrections factors involved in our proposal. Then, the aggregation formula is introduced. Finally, the algorithm and the interpretation of the results is explained in detail.

A. Correction Factors

The proposed aggregation algorithm integrates correction factors to adapt the formula described in MAGERIT. These correction factors apply individually for each CVSS, except for the average and summarized factors. Correction factors are summarized in Table I.

1) *Functionality factor (ρ):* This correction factor represents whether any functionality of the systems is affected by its vulnerabilities. It is represented by a binary value, being 0 when no functionality is affected, and 1 when any of them is affected. For example, a cryptographic library with a vulnerability in SHA1. If the SUT does not make use of SHA1 in any way, the vulnerability would not be exploitable, and could be removed from the analysis ($\rho = 0$).
2) *Deepness Factor (β):* This factor represents the difficulty of chained exploitation of each vulnerability. It is represented by a value between $[0, 1]$ inversely proportional to the amount of assets to compromise in order to exploit vulnerability. Vulnerabilities close to the entry point will account more for the final aggregation, whereas those that are far away will account less. In this approach, linear interpolation is proposed to calculate the weight of each layer, because of its simplicity. Nevertheless, different interpolations could be used according to the

TABLE I: Correction factors proposed for adapting the Bayesian sum proposed in MAGERIT.

CORRECTION FACTOR	DESCRIPTION	AUTOMAtED
Functionality factor (ρ)	Binary value indicating whether a vulnerability affects or not the functionality of the SUT.	■
Deepness factor (β)	Value between $[0, 1]$ proportional to the position of the affected asset in the EDG of the SUT.	■
Context factor (γ)	Binary value indicating vulnerability exploitability in the real and particular conditions of the SUT.	□
Exploit factor (μ)	Existence of a public exploit, proportional to its state of development: Not defined ($\mu = 0.5$), Theoretical ($\mu = 1.25$), Proof-Of-Concept ($\mu = 1.5$), Functional ($\mu = 1.75$), and Automated ($\mu = 2$).	■
Summarized factor (λ)	This factor summarizes the effect of all the above ones, $\lambda = \rho\beta\gamma\mu\sigma$.	■
Average factor (σ)	Function that adjust the value of the sum to avoid its rapid evolution to 10.	■

criticality of the system. Fig. 1 shows the corresponding β for a four-layer system.

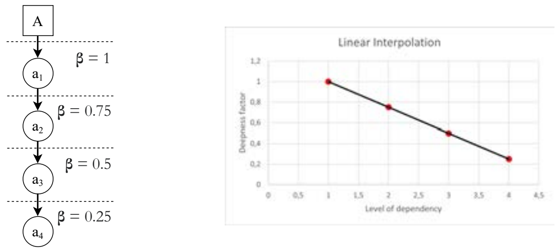

Fig. 1: Calculation of the deepness factor for a four-layer of dependency example.

3) *Context factor (γ):* This factor considers whether the exploitation of a vulnerability is actually possible in the real scenario where the system is deployed. It is represented by a binary value, where 0 indicated that it is not possible, and 1, that it is possible. It is calculated comparing the attack vector of the CVSS with the real conditions where the device is deployed. For example, this can happen when a vulnerability with a high CVSS score needs physical access to be exploited, but in reality the device is physically isolated. To reflect this, the CVSS should be updated, lowering the resulting value [16]. This factor aims to complement the existing submetrics in the temporal and the environment metrics of the CVSS. Both the temporal and the environmental scores lack of an "isolated" value for the attack vector.
4) *Exploit factor (μ):* This factor accounts for the existence of a public exploit for a given vulnerability, being proportional to its state of development. The temporal score of the CVSS already implements this feature, but the CVSS values are not updated in practice [15]. Moreover, taking into account the temporal score has almost no effect as opposed to using the raw initial base score. This means that a CVSS just considering the base score is higher than a CVSS considering an exploit code maturity of "functional exploit exists". To solve this issue, we introduce the following values for the exploit factor: Not defined ($\mu = 0.5$), Theoretical ($\mu = 1.25$), Proof-Of-Concept ($\mu = 1.5$), Functional ($\mu = 1.75$), and Automated ($\mu = 2$). These values are equivalent to the scale defined in the CVSS Specification Document [10].
5) *Summarized factor (λ):* The λ factor accounts for the effect of all the factors above:

$$\lambda = \rho\beta\gamma\mu \tag{1}$$

6) *Average factor (σ):* This factor defines the behavior of the aggregation function. It can be chosen as needed (*e.g.*, the arithmetic or harmonic mean), but taking into account all the values to be added.

B. Aggregation Formula

The aggregation function is defined as:

$$\Gamma(\overrightarrow{V}) = 10 - \frac{1}{\sigma} f(\overrightarrow{V}) \tag{2}$$

Where $\overrightarrow{V}$ is a vector $(cvss_0, cvss_1, \ldots, cvss_n)$ with all the corrected CVSS values to be added, $cvss$, being n the last value to be added. $f(\overrightarrow{V}) = a_n$ is defined as the following recursive function:

$$a_n = 10\left[1 - \left(1 - \frac{\lambda_{a_{n-1}}}{10} a_{n-1}\right) \cdot \left(1 - \frac{\lambda_{cvss_n}}{10} cvss_n\right)\right] \tag{3}$$

Where the base case is defined as:

$$a_0 = \lambda_{cvss_0} cvss_0 \tag{4}$$

C. Algorithm

The proposed aggregation algorithm is divided into the following steps (see Fig. 2):

1) Calculation of the correction factors for each CVSS,
2) Calculation of the summarized factor for each CVSS,
3) Calculation of the corrected CVSS values,
4) Calculation of the average correction function, and
5) Aggregation.

Notice that the dependency graph of the SUT, the vulnerabilities associated to each element of the dependency graph, and their CVSS value are needed.

1) Correction factors for each CVSS: The first step obtains the values of each correction factor for each CVSS:

1) **Functionality factor (ρ):** This factor is obtained using the description provided in the corresponding CVE of each CVSS. The description provides enough information to decide whether the functionality of the system is affected.
2) **Context factor (γ):** This factor is obtained by comparing the value of the Attack Vector (AV) submetric of the CVSS, with the real environment of deployment of the SUT.

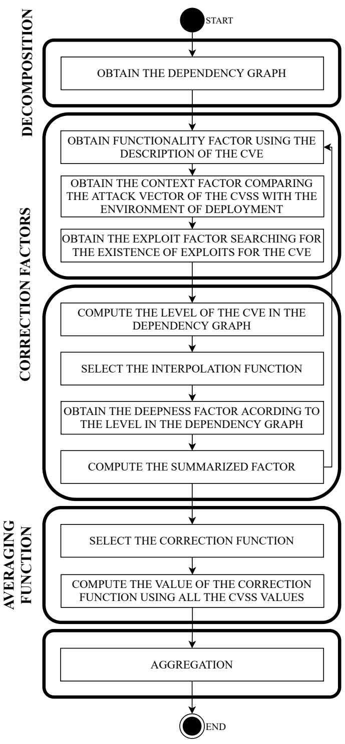
START
DECOMPOSITION
OBTAIN THE DEPENDENCY GRAPH
CORRECTION FACTORS
OBTAIN FUNCTIONALITY FACTOR USING THE DESCRIPTION OF THE CVE
OBTAIN THE CONTEXT FACTOR COMPARING THE ATTACK VECTOR OF THE CVSS WITH THE ENVIRONMENT OF DEPLOYMENT
OBTAIN THE EXPLOIT FACTOR SEARCHING FOR THE EXISTENCE OF EXPLOITS FOR THE CVE
COMPUTE THE LEVEL OF THE CVE IN THE DEPENDENCY GRAPH
SELECT THE INTERPOLATION FUNCTION
OBTAIN THE DEEPNESS FACTOR ACORDING TO THE LEVEL IN THE DEPENDENCY GRAPH
COMPUTE THE SUMMARIZED FACTOR
AVERAGING FUNCTION
SELECT THE CORRECTION FUNCTION
COMPUTE THE VALUE OF THE CORRECTION FUNCTION USING ALL THE CVSS VALUES
AGGREGATION
END

Fig. 2: Flowchart showing the main steps of the aggregation algorithm for each CVSS.

3) **Exploit factor (μ):** To obtain this factor, public databases have to be queried to find any potential exploit for each vulnerability.
4) **Deepness factor (β):** For any given CVSS, its value is obtained according to the deepness in the exploit chain for the SUT [14].

2) Summarized factor for each CVSS: The summarized factor, λ, is obtained by multiplying all the corrections factors obtained in the previous step, following Equation 1.

3) Corrected CVSS values: The corrected CVSS values are obtained by multiplying each CVSS by its corresponding summarized factor, λ. At this point, it is necessary to check for overflows, because the exploitation factor generated corrected CVSS values higher than 10. Values higher than 10 are set to 10 at this stage.

4) Correction function: At this point, it is necessary to choose an averaging function. Choosing one function over the other will cause the aggregation result to grow slower or faster toward 10 in each addition. In this case, and for the sake of clarity, we chose the arithmetic mean, but any other kind of mean (*e.g., harmonic mean*) could be used according to each scenario.

5) Aggregation: Finally, the aggregated value is computed using Equation 2.

D. Interpretation of the result

The advantage of this method is that the result can be interpreted in the same way that a normal CVSS would be interpreted. This is because of the correction factors in Equation 2, that only let the algorithm return high values when vulnerabilities with high CVSS values are exploitable in reality (λ is close to 1). This mechanism ensures that multiple aggregated low CVSS values do not result in a critical score just because there are a large number of them.

IV. Use Case

To test the potential of our proposal, we analyzed Version 3 of OpenPLC project, obtaining a CVSS aggregated value for its vulnerabilities using the proposed algorithm.

OpenPLC is the first functional open source Programmable Logic Controller (PLC), both in software and hardware [34]. It was mainly created for research purposes, because it provides its entire source code [35], [36]. The current version of the project is OpenPLC V3 [37].

A. Use Case Scenario

For this use case, we are going to make the next assumptions:

- The system executing OpenPLC V3 is deployed in an isolated network.
- The system running OpenPLC V3 is physically isolated.
- The attacker is an insider without access to the systems.
- The reference point for the deepness factor will be the `webserver.py` in Fig. 3.

B. Structure of OpenPLC

The first step was to obtain the inner structure of OpenPLC V3 using the Extended Dependency Graph (EDG) proposed in [14]. To simplify the obtained graph, we only represented the shortest path to each node, so the worst case scenario (more accessible from the outside) is considered. The result is shown in Fig. 3.

C. Calculation of the Correcting Factors

OpenPLC V3 has five vulnerabilities: two vulnerabilities affecting `libgcc_s`, and three vulnerabilities affecting `libc`. Table II shows each vulnerability in more detail.

From these data, it is possible to obtain all corrections factors for each vulnerability, as follows (Table II summarizes the results):

1) Functionality Factor (ρ)r: This factor is obtained from the analysis of the description of each CVE. From these data, we have to decide whether the functionality of OpenPLC V3 is affected ("1") or not ("0").

2) Deepness Factor (β)b: By taking a look at Fig. 3, it can be seen that the maximum deepness level is four. So the possible values for the deepness factor are the ones shown in Fig. 1. More precisely, vulnerabilities CVE-2019-15847 and CVE-2018-12886 have a deepness factor of 0.25, because they are at level four. By contrast, vulnerabilities CVE-2017-18269, CVE-2018-11236, and CVE-2018-11237 have a deepness factor of 0.5, because they are at level three.

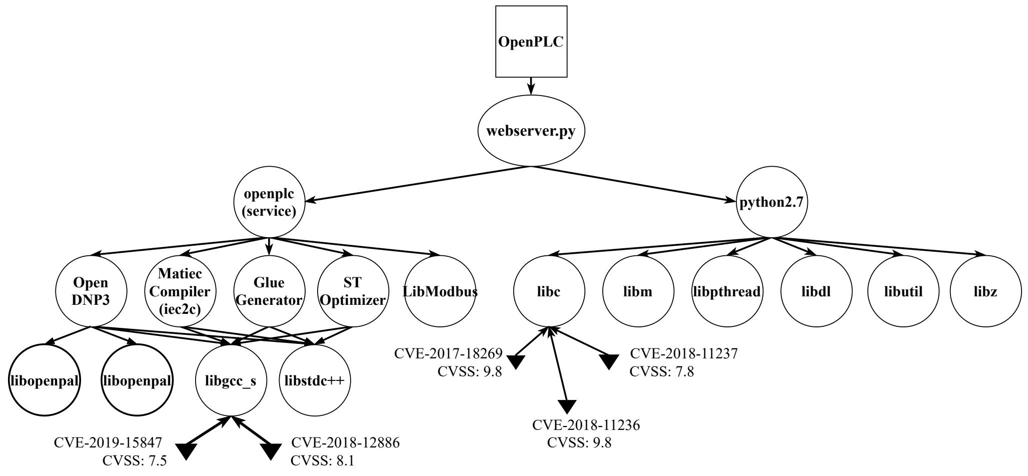

Fig. 3: Extended Dependency Graph of OpenPLC V3. Circles represent individual assets, black triangles are the vulnerabilities associated to each asset, and the square represent the entry point to the system, or root node of dependency.

TABLE II: Vulnerabilities present in OpenPLC V3. For each one, the CVSS is shown, together with their associated Attack Vector (AV), and their correction factors.

CVE	CVSS	Attack Vector	Functionality (ρ)	Deepness (β)	Context (γ)	Exploit (μ)	Summarized (λ)	Corrected CVSS
CVE-2017-18269	9.8	Network	1	0.5	1	1.25	0.625	6.125
CVE-2018-11236	9.8	Network	0	0.5	1	0	0	0
CVE-2018-11237	7.8	Local	1	0.5	0	1.25	0	0
CVE-2018-12886	8.1	Network	1	0.25	1	1.25	0.313	2.530
CVE-2019-15847	7.5	Network	1	0.25	1	1.25	0.313	2.344

3) Context Factor (γ)*g:* From the initial assumptions, insiders can only exploit the existing vulnerabilities from the local network. This means that every vulnerability that has an attack vector of “network” (N) can be exploited, thus CVE-2017-18269, CVE-2018-11236, CVE-2018-12886, and CVE-2019-15847 are exploitable by the attacker. Vulnerabilities whose attack vector is “local” (L) cannot be exploited, because physical access is needed. Therefore, CVE-2018-11237 cannot be exploited.

4) Exploit Factor (μ)*m:* Public databases have to be queried to find existing exploits for each vulnerability. According to their state of development, a different value is assigned.

5) Summarized Factor (λ)*l:* The summarized factor for each vulnerability is obtained as the product of the previous factors, as shown in Equation 1. At this step, by taking a look at the resulting values of λ, it is possible to know which CVSS will contribute to the final aggregation and in which percentage ($\lambda > 0$), and which ones will not contribute at all ($\lambda = 0$).

6) Average Factor (σ)*s:* Finally, we obtained the average factor by calculating the arithmetic mean of all the initial CVSS values: $\sigma = 8.6$.

D. Aggregation

The previous step before the aggregation is obtaining the corrected CVSS value for each initial CVSS. This is done by multiplying each CVSS by their corresponding summarized value (λ). The corrected values are shown in Table II.

Finally, the aggregation is performed using the corrected CVSS values. The aggregation is an iterative process that takes the first two values to be added, and adds them using Equation 2. Then, this result is added to the third value to be added, and so on, until there are no more values.

For OpenPLC V3, this process returns a final aggregated value of 9.1. Without the correction factors, the result would be 10. Nevertheless, taking into account features such as the exploitability of the vulnerabilities, the context of the SUT, or its functionalities, we can select the most important CVSS values to be aggregated. With such process, the total amount of CVSS values to be added is simplified. This also helps to simplify potential attack paths.

This result was obtained aggregating three of the five CVSS values present in OpenPLC V3. The associated CVSS for CVE-2018-11236 and CVE-2018-11237 were not taking into account for the aggregation, because they do not affect to any functionality of the system, Moreover, CVE-2018-11237 cannot be exploited in the conditions described in the use case.

CVE-2017-18269 (with an associated CVSS of 9.8) is the vulnerability with the highest value for λ. Therefore, it is going to contribute the most to the final aggregated value. CVE-2018-12886 and CVE-2019-15847 follow with a CVSS of 8.1 and 7.5 respectively. As it is shown, the selected vulnerabilities have a high CVSS, so it is expected that the aggregated value would be also high. This is reflected in the obtained result of 9.1.

Finally, it is worth highlighting that the final result is lower than the highest CVSS value present in OpenPLC V3. This difference is due to the effect of the correction factors: as the CVE-2017-18269 is further away from the entry point of the system (in layer 3), its real CVSS value in lower.

V. Conclusions and Future Work

In this research work, we proposed a new aggregation algorithm for CVSS values. The proposed approach integrates correction factors to select the most relevant CVSS values to be added based on contextual information. For each vulnerability, we check for:

1) Functionality disruption.
2) Exploitation difficulty.
3) Existence of exploits, and their development state.
4) Context of deployment.

We assigned a different correction factor to each one of the previous properties to further ponder the initial CVSS value and adjust it to the real context where the system is operating.

The proposed aggregation algorithm was applied to OpenPLC V3 in a use case. Two of the existing vulnerabilities were filtered out by the algorithm, as they cannot be exploited in the described context of OpenPLC V3. The rest of the vulnerabilities were aggregated, and the result (9.1) was indeed lower than the highest CVSS present in the system (9.8). This shows that the CVSS for each vulnerability was correctly adjusted to the real context of deployment of OpenPLC V3.

As future work, we plan to perform the aggregation at the submetric level of the CVSS, instead of using the base metric value, giving more granular values for each factor.

Acknowledgements

Iñaki Garitano is a member of the Intelligent Systems for Industrial Systems research group at Mondragon Unibertsitatea (IT1676-22), supported by the Department of Education, Universities and Research of the Basque Government. This work was partially supported by the *Ayudas Cervera para Centros Tecnológicos* grant of the Spanish Center for the Development of Industrial Technology (CDTI) under the project EGIDA (CER-20191012), and by the Basque Country Government under the ELKARTEK program, project REMEDY - Real Time Control And Embedded Security (KK-2021/00091).

References

[1] S. Pfleeger and R. Cunningham, "Why measuring security is hard," *IEEE Security Privacy*, July 2010.

[2] S. M. Bellovin, "On the brittleness of software and the infeasibility of security metrics," *IEEE Security & Privacy*, vol. 4, no. 4, pp. 96–96, 2006.

[3] A. Atzeni and A. Lioy, "Why to adopt a security metric? a brief survey," *Advances in Information Security*, vol. 23, pp. 1 – 12, 2006.

[4] 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*, NSPW '09, (New York, NY, USA), pp. 37–50, ACM, 2009.

[5] S. Stolfo, S. M. Bellovin, and D. Evans, "Measuring security," *IEEE Security Privacy*, May 2011.

[6] W. H. Sanders, "Quantitative security metrics: Unattainable holy grail or a vital breakthrough within our reach?," *IEEE Security Privacy*, vol. 12, no. 2, pp. 67–69, 2014.

[7] M. Rudolph and R. Schwarz, "A critical survey of security indicator approaches," in *2012 Seventh International Conference on Availability, Reliability and Security*, pp. 291–300, Aug 2012.

[8] S. Sentilles, E. Papatheocharous, and F. Ciccozzi, "What do we know about software security evaluation? a preliminary study," in *QuA-SoQ@APSEC*, 2018.

[9] D. G. Ángel Longueira-Romero, Rosa Iglesias and I. Garitano, "How to Quantify the Security Level of Embedded Systems? A Taxonomy of Security Metrics," in *Proceedings of the 2020 18th IEEE International Conference on Industrial Informatics (INDIN)*, 2020.

[10] FIRST - global Forum of Incident Response and Security Teams, "Common Vulnerability Scoring System (CVSS)." https://www.first.org/cvss/v3-1/, 2021-02-03.

[11] National Institute for Standards and Technology (NIST), "National Vulnerability Database NVD — Vulnerabilities." https://nvd.nist.gov/vuln/search, 2021-02-03.

[12] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," vol. 49, dec 2016.

[13] H. Howland, "Cvss: Ubiquitous and broken," *Digital Threats*, sep 2021.

[14] A. Longueira-Romero, R. Iglesias, J. L. Flores, and I. Garitano, "A novel model for vulnerability analysis through enhanced directed graphs and quantitative metrics," *Sensors*, vol. 22, no. 6, 2022.

[15] J. Spring, E. Hatleback, A. Householder, A. Manion, and D. Shick, "Time to change the cvss?," *IEEE Security & Privacy*, vol. 19, no. 2, pp. 74–78, 2021.

[16] P. Cheng, L. Wang, S. Jajodia, and A. Singhal, "Aggregating cvss base scores for semantics-rich network security metrics," in *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pp. 31–40, 2012.

[17] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, "Dag-based attack and defense modeling: Don't miss the forest for the attack trees," *Computer Science Review*, vol. 13-14, pp. 1–38, 2014.

[18] T. Heyman, R. Scandariato, C. Huygens, and W. Joosen, "Using security patterns to combine security metrics," in *2008 Third International Conference on Availability, Reliability and Security*, pp. 1156–1163, 2008.

[19] Z. Song, Y. Wang, P. Zong, Z. Ren, and D. Qi, "An empirical study of comparison of code metric aggregation methods–on embedded software," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pp. 114–119, 2019.

[20] M. Walkowski, M. Krakowiak, M. Jaroszewski, J. Oko, and S. Sujecki, "Automatic cvss-based vulnerability prioritization and response with context information," in *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pp. 1–6, 2021.

[21] C. Fruhwirth and T. Mannisto, "Improving cvss-based vulnerability prioritization and response with context information," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 535–544, 2009.

[22] L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," in *Data and Applications Security XXII* (V. Atluri, ed.), (Berlin, Heidelberg), pp. 283–296, Springer Berlin Heidelberg, 2008.

[23] S. Zhang, X. Ou, A. Singhal, and J. Homer, "An empirical study of a vulnerability metric aggregation method," in *Mission Assurance and Critical Infrastructure Protection*, 2011 World Congress in Computer Science, 2011-08-18 00:08:00 2011.

[24] N. Idika and B. Bhargava, "Extending attack graph-based security metrics and aggregating their application," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 75–85, 2012.

[25] M. Zhang, L. Wang, S. Jajodia, and A. Singhal, "Network attack surface: Lifting the concept of attack surface to the network level for evaluating networks' resilience against zero-day attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 310–324, 2021.

[26] J. Homer, S. Zhang, X. Ou, D. Schmidt, Y. Du, S. R. Rajagopalan, and A. Singhal, "Aggregating vulnerability metrics in enterprise networks using attack graphs," *Journal of Computer Security*, vol. 21, no. 4, pp. 561–597, 2013.

[27] L. Gallon and J.-J. Bascou, "Cvss attack graphs," in *2011 Seventh International Conference on Signal Image Technology Internet-Based Systems*, pp. 24–31, 2011.

[28] M. Frigault, L. Wang, A. Singhal, and S. Jajodia, "Measuring network security using dynamic bayesian network," in *Proceedings of the 4th ACM Workshop on Quality of Protection*, QoP '08, (New York, NY, USA), p. 23–30, Association for Computing Machinery, 2008.

[29] N. Poolsappasit, R. Dewri, and I. Ray, “Dynamic security risk management using bayesian attack graphs,” *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 1, pp. 61–74, 2012.

[30] P. Xie, J. H. Li, X. Ou, P. Liu, and R. Levy, “Using bayesian networks for cyber security analysis,” in *2010 IEEE/IFIP International Conference on Dependable Systems Networks (DSN)*, pp. 211–220, 2010.

[31] M. A. Amutio, J. Candau, and J. A. Mañas, “MAGERIT V3.0. Methodology for Information Systems Risk Analysis and Management. Book III - Technical Guide,” National Standard, Ministry of Finance and Public Administration, Madrid, Spain, 2012.

[32] M. A. Amutio, J. Candau, and J. A. Mañas, “MAGERIT V3.0. Methodology for Information Systems Risk Analysis and Management. Book I - The Method,” National Standard, Ministry of Finance and Public Administration, Madrid, Spain, 2014.

[33] A. Syalim, Y. Hori, and K. Sakurai, “Comparison of risk analysis methods: Mehari, magerit, nist800-30 and microsoft’s security management guide,” in *2009 International Conference on Availability, Reliability and Security*, pp. 726–731, 2009.

[34] Thiago Alves, “OpenPLC Project.” https://www.openplcproject.com/.

[35] T. R. Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, “Openplc: An open source alternative to automation,” in *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, pp. 585–589, 2014.

[36] T. Alves and T. Morris, “Openplc: An iec 61,131–3 compliant open source industrial controller for cyber security research,” *Computers & Security*, vol. 78, pp. 364–379, 2018.

[37] Thiago Alves, “OpenPLC V3.” https://github.com/thiagoralves/OpenPLC_v3, 2021-05-15.