Multi-objective prioritization for data center vulnerability remediation

Felipe Colombelli*†§, Vítor Kehl Matter*§, Bruno Iochins Grisci*†§, Leomar Lima‡, Karine Heinen‡, Marcio Borges‡, Sandro José Rigo*, Jorge Luis Victória Barbosa*, Rodrigo da Rosa Righi*, Cristiano André da Costa*, and Gabriel de Oliveira Ramos*¶

* Graduate Program in Applied Computing, Universidade do Vale do Rio dos Sinos, São Leopoldo, RS, Brazil

† Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, RS, Brazil

‡ Dell Technologies Brazil, Eldorado do Sul, RS, Brazil

§ These authors contributed equally to this work.

¶ gdoramos@unisinos.br

***Abstract*—Nowadays, one of the most relevant challenges of a data center is to keep its information secure. To avoid data leaks and other security problems, data centers have to manage vulnerabilities, including determining the higher-risk vulnerabilities to prioritize. However, the current literature is scarce in the proposal of intelligent methods for the complex problem of vulnerabilities prioritization. Depending on the adopted metrics, the priority could shift, compromising simple sorting-based approaches and impairing the utilization of conflicting risk assessment metrics. Unlike the related work, this study proposes a multi-objective method that uses user-chosen vulnerabilities assessment metrics to output a complete list of these vulnerabilities ranked by their risk and overall impact in the context of an organization. The method includes a multi-objective large-scale optimization problem representation, a novel population initialization scheme, an expressive fitness function, a post-optimization process, and a custom way to select the best solution among the non-dominated ones. The dataset used in the experiments contains anonymized real-world information about database vulnerabilities obtained from a private organization. The experiments' results indicated that the proposed method can reduce the number of vulnerabilities needed to reach an organization's predefined security targets compared to the baselines simulating a security team's analysis. Multi-objective optimization achieved on average a 48,17% reduction in the vulnerabilities needed to reach the organization's target values compared to the baselines.**

***Index Terms*—vulnerabilities, multi-objective optimization, database security, risk prioritization**

I. Introduction

Vulnerabilities are flaws in systems, processes, and strategies that result in risks [6]. These vulnerabilities can be found in data centers, which have to deal with these flaws as soon as they are identified because it is critical to avoid data leaks or problems regarding dependencies that can threaten organizations. Usually, specialists are employed to handle the vulnerabilities patching, but time and costs might compromise this process, so it is necessary to identify the priority in which vulnerabilities must be patched first. Le et al. [14] indicated that dealing with vulnerabilities is a growing area and that it can assist in avoiding critical situations for organizations.

Accepted for publication at IEEE CEC 2022.

According to a survey report [21] conducted in 2019, the leading cause for the most severe data breaches is unpatched vulnerabilities, with 39% of companies scanning vulnerabilities at most once per month. Moreover, approximately half of them do not apply patches in two weeks or less. The average cost of a data breach was $3.86 million in 2020, and proper incident response planning resulted in a $2 million reduction in costs [10].

It is relevant to mention that the patching process occurs after the scanning process and, differently from the scanning that has several specialized frameworks to identify and rank vulnerabilities, the patching process usually depends on the manual evaluation of security teams to define which vulnerability will be patched first [14]. However, considering an organization's dynamic environment, the security teams must rely on more than one metric, which is a costly and time-consuming process to identify the best patching solution. Multiple metrics must be considered so that the algorithm can recognize the changes of context or shifts in the organization's needs [19]. Therefore, a promising approach to handle vulnerabilities prioritization is using multi-objective optimization (MOO), which considers multiple metrics at once [7].

Methods capable of managing complex vulnerabilities prioritization problems require adopting specific priority metrics. However, the vulnerabilities prioritization can have considerable shifts depending on which metric is employed. So, multiple metrics must be considered in complex scenarios like the vulnerabilities remediation to define a prioritization. Nonetheless, simple sorting algorithms applied to conflicting risk assessment metrics can disagree with the prioritization. MOO handles these kinds of disagreements approaches well, justifying its use in this regard. In Section II, these gaps are expanded, and the MOO importance becomes even more evident. Viduto et al. [22] and Farris et al. [5] presented a similar interest in applying MOO to data security, but their works have different methods to evaluate and prioritize which vulnerabilities should be patched. The related work do not provide a complete prioritization rank because they select specific sets of vulnerabilities to be patched. Section V will discuss and expand the differences between this work and the

related work.

This paper proposes a vulnerability prioritization method that allows organizations to decide the most relevant metrics for the evaluation. The method considers predefined metrics and organizations' objectives' targets. These metrics are composed of specific data attributes acquired from an organization to calculate the risk prioritization. Objectives targets are the values that must be reached in each predefined metric.

In this work, the distinct metrics combination is treated as a multi-objective problem [9, 18], in which ranking each metric is an objective, so MOO is applied to generate the solutions (the candidate ranks). The advantage of using a multi-objective approach is the combination and customization of each metric's importance, creating a more intelligent and flexible analysis. The method's inputs are the dataset of servers and vulnerabilities and the metrics that the algorithm should evaluate. The input data must be provided by organizations because the method itself does not obtain the vulnerabilities from data centers. Afterward, our method's strategies for population initialization and fitness (explained in the following sections) can be used. After the optimization, the post-optimization strategy further improves the prioritization results. According to the user's predefined preferences, the method output is a list of vulnerabilities sorted in descending order from the highest severity risk prioritization to the lowest. This work does not contemplate the scanning, classification, or the afterward vulnerabilities patching process.

The experiments showed that the baselines simulating a security specialist analysis do not achieve the most optimized patching scenario. These baselines were simulated using the method explained in Section III-B. Our method expressed a considerable reduction in the vulnerabilities quantity needed to reach an organization's predefined reduction targets. To the best of our knowledge, this is the first work that proposes a complete vulnerabilities prioritization method using MOO. This paper presents the following contributions:

- The proposal of a multi-objective, large-scale optimization problem representation for evaluating risk prioritization of data center vulnerabilities;
- A novel relevant population initialization based on the Borda count rank aggregation method;
- An expressive fitness function that uses the Kendall Tau rank correlation metric for measuring how close to perfect a solution is, considering a particular objective;
- A post-optimization process for correcting clearly out of order elements from the final solutions;
- A custom strategy to select the best solution among the non-dominated ones through metrics that are commonly reported by specialists.

The remainder of this document has the following organization. Section II describes the risk assessment and the MOO approaches. Section III defines the proposed method. Section IV presents the experiments and results. Section V describes the related work, and Section VI presents the conclusions.

II. Background

Ranking the vulnerabilities and servers according to specific objectives to receive patches smartly is recommended to guarantee that critical issues are identified and dealt with rapidly and that the vulnerability remediation lifecycle management is appropriately handled [14]. Patching time and costs can thus be reduced using an algorithm that finds the best ranking solutions. The vulnerability remediation lifecycle starts by scanning the data servers to detect vulnerabilities. After that, the security team performs the classification of these vulnerabilities, which uses predefined metrics (e.g., risk and age) and the organization's guidelines to categorize the severity level of the vulnerability [6]. Afterward, a risk score is given to each vulnerability which enables their prioritization and ranking, allowing the security team to select which solution, a set of vulnerabilities ranked by specific metrics, should be patched first.

MOO is a promising strategy that can consider the vulnerability assessment and the different predefined metrics, considering that each of them is represented as an objective to be optimized simultaneously. There is no single solution in multi-objective problems capable of optimizing all objectives [9, 18]. Thus, this means that there are several *optimal* solutions, because no other solution is better than them considering all the objectives. For these solutions, it is impossible to improve one of the objectives without worsening one or more of the other objectives.

Jacobs et al. [11] argued that the security risk cannot be reduced to a single value regarding data security applications. For instance, if there are two metrics available, one for risk severity and the other measuring the probability of a risk being exploited, one option is to create a new single metric using scalarization of the two original metrics. While this strategy may appear to be a better measurement of risk, only applying mathematical operations to combine metrics should be avoided as it can lead to misinterpretation [11]. When multiple metrics offer orthogonal information about the risk, their values should be considered simultaneously and individually [11].

Simply aggregating the metrics may become a semi-manual process that prevents the stakeholders from evaluating well-informed trade-offs and offers less transparent decisions [9]. Moreover, these strategies fail to account for the changes in a dynamic environment [9]. On the other hand, dealing with risk prioritization using the multi-objective perspective may bring several advantages, even though this topic has been neglected in the literature [1]. By considering each metric associated with risk as an objective to be optimized, multi-objective algorithms can find a set of optimal risk prioritization ranks, and the users can choose a solution that satisfies their current needs. This decision can account for distinct needs for each user and be effectively updated.

The techniques presented in our method are well suited for optimizing the solutions and achieving the best results for the chosen metrics. The following aspects will be discussed in the next section: how the problem is represented; how

the first population is generated to initialize the optimization process; how to compute each solution's quality using a fitness function; which multi-objective algorithms are being used; and, finally, how a post-optimization method is used to fine-tune the final set of non-dominated solutions obtained.

III. Proposed Method

A MOO strategy is employed to tackle the vulnerabilities prioritization problem. The implemented method generates possible prioritization ranks for the vulnerabilities through an evolutionary algorithm. The main goal is to allow users to make well-informed decisions regarding which server and vulnerabilities should receive attention first.

This problem-solving strategy allows treating the distinct vulnerability assessment metrics as objectives in the MOO. Instead of relying solely on fixed metrics that are proprietary to third parties and do not consider the user's infrastructure organization, the multi-objective method allows the combination and customization of each metric's importance. Furthermore, the algorithm enables a versatile parametrization of the method. Therefore, this allows the user to acquire the vulnerabilities prioritization solutions for other objectives' values without executing another risk prioritization process, so tests with different target values can be easily validated.

After choosing the objectives, *i.e.*, the metrics used in the optimization process, the user can execute the prioritization method. These metrics must be numeric or converted to a numeric format to order the vulnerabilities according to that metric. Then, the method generates an initial custom population and utilizes an expressive fitness function to evolve this population by employing an evolutionary algorithm. After the optimization, a post-optimization process is performed to fine-tune the solutions. The user can then define their preferences for the objectives scalarization and target oriented search, which will find the best single solution. If these user preferences change over time, only the scalarization and target-oriented search processes are re-executed, which are very computationally efficient. Unlike the optimization with *a priori* scalarization, this method prevents the need for a new execution of the costly optimization process.

A. Problem representation

As mentioned above, the method aims to provide prioritization rank lists with solutions containing the vulnerabilities that the algorithm should patch depending on the user's preference. Thus, a possible solution for this problem is a list of rankings, *i.e.*, positions in the rank of vulnerabilities, where the ones with the lowest rankings are at the top of the rank and should be patched first. This way, the optimization process must rank each vulnerability, focusing on better contemplating the user's concerns regarding the patching urgency of each vulnerability as measured by the adopted metrics (*e.g.*, Common Vulnerability Scoring System (CVSS)[1] score, number of affected services, and others).

[1]https://www.first.org/cvss/

Additionally, the solutions can assign the same ranking for more than one vulnerability, which means these vulnerabilities are tied and have the same priority. During the optimization process, ties are represented as the minimum ranks' value [16] that would have been assigned to all the tied values (also referred to as *competition* ranking). One example of this behavior is a rank with three vulnerabilities $\{V1, V2, V3\}$; if the vulnerability $V1$ is tied with $V3$ and has a higher priority than $V2$, the solution rank would look like $[1, 3, 1]$. Thus, this means that $V1$ and $V3$ have the same priority ranking of 1 and that the third position is occupied by $V2$. With competition ranking, the vulnerabilities with the same score do not have an arbitrary order; instead, they are tied. A mathematical formulation for this problem is presented by Grisci et al. [7].

B. Population initialization

The MOO can use an initial personalized population to acquire the best solutions faster. Therefore, to achieve this goal, the algorithm received specific additions: (i) the best individuals according to each isolated objective (Section III-C); (ii) individuals generated by rank aggregation; and (iii) some random individuals to explore a broader area of the Pareto front and diversify the final set of non-dominated solutions.

The rank aggregation individuals (baselines) are obtained by applying the Borda count method [16] over the best ranks according to each objective, using different weights for each rank. The Borda count method gives a weight to each rank and performs a weighted sum for each vulnerability ranking, adding up their positions in each rank, multiplied by their respective rank weights. This method and other rank aggregation strategies are used in several fields, with a highlight for the analysis of biological data [3, 8, 15].

The random individuals are obtained by randomly choosing a ranking for each position in the solution vector, *i.e.*, for each vulnerability. The rankings are lower bounded by the top position of the solution rank and upper bounded by the last possible position in this rank. Additionally, as explained in the last subsection, ties are allowed. Thus, the ranking sampling for each position in the solution vector is performed with replacement, which means the same random ranking could be assigned for more than one vulnerability.

C. Fitness function

A fitness function is employed to assess the quality of a solution. Since the problem is an optimization problem with more than one objective, this function must describe separately how good the solution is for each objective. The best possible solution for a single objective is a rank ordered by its defined metric. Additionally, the rank built from sorting a single objective results in too many ties. This rank has its ties broken by using the other objectives sorted by their metrics, resulting in multiple best ranks for each objective depending on each metric's priority.

To better understand this functionality, consider the metrics $\{M1, M2, M3\}$ represented by the objectives $\{O1, O2, O3\}$,

respectively. The two best ranks for $O1$ are: (I) the rank ordered by $M1$, ties broken by $M2$ and then ties broken by $M3$; and (II) the rank ordered by $M1$, ties broken by $M3$ and then ties broken by $M2$. For example, in the case of rank (I), the first tie-breaking is employed because a number of vulnerabilities could have the same value of $M1$, and the second tie-breaking is used because they could also have the same $M1$ and $M2$ values. However, there is no guarantee that other vulnerabilities do not possess the same M1, M2, and M3 values. Such cases may occur in the algorithm given the characteristics of the current problem domain, where millions of vulnerabilities are present, and the range of possible values for each metric is limited.

After computing the two best possible ranks for each objective (in the case with three objectives), it is possible to evaluate the quality of any feasible solution. The evaluation process must compute how close the solution is to each objective's two best possible ranks and return the smallest distance. This rank distance is calculated by the Kendall Tau [13] rank correlation equation, which results in a number within the $[-1, 1]$ range. The more correlated the two ranks are, the greater their Kendall Tau value and, consequently, the closer the ranks (meaning that they agree more). A Kendall Tau value of 1 depicts a total agreement between ranks, and a -1 represents total disagreement. It can be defined by Equations 1 and 2, in which $\tau_1(i)$ and $\tau_2(i)$ are the rankings of element i in the lists τ_1 and τ_2, and P is the set of unsorted pairs of distinct elements in τ_1 and τ_2.

$$K(\tau_1, \tau_2) = \sum_{\{i,j\} \in P} \bar{K}_{i,j}(\tau_1, \tau_2) \bar{K}_{i,j}(\tau_1, \tau_2) \tag{1}$$

$$\bar{K}_{i,j}(\tau_1, \tau_2) = \begin{cases} 0 & \text{if } i \text{ and } j \text{ are in the same order in } \tau_1 \text{ and } \tau_2 \\ 1 & \text{if } i \text{ and } j \text{ are in the reverse order in } \tau_1 \text{ and } \tau_2 \end{cases} \tag{2}$$

The Kendall Tau concept is mapped to the notion of *distance* between two ranks by multiplying the resulted correlation by -1, and, thus, the objective becomes minimizing this distance, the same as maximizing the correlation. In the case with three objectives, the final three values composing the fitness vector correspond to the solution's rank distance to the three objectives being optimized. Each fitness value is calculated by taking the minimum distance between the solution rank and the two best possible ranks for a particular objective.

D. Multi-objective optimization

Two MOO algorithms were selected for comparison in the optimization process, the Non-dominated Sorting Genetic Algorithm (NSGA-II) [4] and the Adaptive Geometry Estimation based Multi-Objective Evolutionary Algorithm (AGE-MOEA) [17]. After the experiments, the algorithm with the best optimization results was chosen for our method. NSGA-II is a genetic algorithm with special mating and survival selection widely used in MOO challenges. In this algorithm, the individuals are selected front-wise, and then the front is split based on a crowding distance (the Manhattan Distance in the objective space) between solutions. This is necessary because not all individuals in the front can be kept in the next generation. The extreme points in the front should always be preserved in the next population. NSGA-II also uses binary tournament mating selection to improve the selection pressure.

AGE-MOEA, one of the most recent algorithms in the MOO field, is an adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization. It estimates the geometry of the generated front and adapts the diversity and proximity metrics accordingly [17]. The main characteristic of the AGE-MOEA is that the non-dominated fronts are sorted using a non-dominated sorting procedure, then the first front obtained is used for normalization of the objective space and estimation of Pareto front geometry. One of the most significant distinctive characteristics of AGE-MOEA is that it estimates the *p* parameter of a Minkowski p-norm to compute a survival score that combines distance from the neighbors and proximity to the ideal point. The algorithm also uses a binary tournament mating selection to increase selection pressure. The first evaluation compares each individual using the rank; afterward, a second evaluation compares the computed score representing proximity and spread.

E. Post-optimization

Slight changes can still improve the solutions generated from the optimizer. One option is to apply slight perturbations to the solutions for obtaining better results. In this case, because the solutions are ranks of vulnerabilities, a sorting algorithm that searches for inversions in the rank and fixes them can be used. An inversion occurs when a vulnerability A is placed in the rank after vulnerability B, but, considering the metrics being optimized, A is better in at least one objective while also being better or equal to B in all the remaining objectives. If this happens, the described sorting procedure changes the rank order to put A in front of B.

This inversions search was implemented based on a bubble sort algorithm due to the lack of the transitivity property in the rank order of the objective tuples. Consider a scenario where there are four objectives being optimized in which there are three elements A, B and C with the metrics values of $[2, 2, 1, 0]$, $[3, 1, 1, 0]$ and $[2, 2, 0, 0]$, respectively. Suppose a solution established the order $\{A, B, C\}$. In this case, the elements $\{A, B\}$ and $\{B, C\}$ do not have a precise order, so they must be kept in this ordering, but the element C should be placed before A. A classic $O(n \log n)$ sorting algorithm could compare A with B and decide that they are ordered (so it must be an $A < B$ relation); compare A with C and observe that $C < A$; and finally it could deduce (without actually comparing) that $C < B$, outputting $\{C, A, B\}$ as the final order, which would violate the original solution's established order of $B < C$. The bubble sort algorithm does not have this problem since it always compares two elements next to each other, performing a swap between them if necessary.

F. Objectives scalarization

In this step, the method receives, from the user, importance weights between 0.0 and 1.0 for each objective. The weights

reflect the importance the user ascribes to each objective, in which 1.0 means that the objective is very significant and 0.0 means that it is irrelevant. The method computes a weighted sum of the objectives' fitness multiplied by their importance weights to obtain a single optimality value representing the quality of a solution given the user preferences. This step is recomputed each time the user changes a weight.

Such post-optimization scalarization process (also known as *a posteriori* scalarization) is currently advised by the scientific literature in decision support scenarios where it may be undesirable, infeasible, or even impossible to adopt a single scalar value for describing the quality of a particular solution during the optimization phase [19]. For instance, consider an organization's complex and dynamic environment in which there could be thousands or even millions of vulnerabilities in a particular set of assets. In this case, it may not be possible to repeat a single-objective optimization process with *a priori* scalarization whenever there is a change in the preferences over the optimized metrics. Additionally, in the multi-objective *a posteriori* scalarization approach, the user can more easily investigate the outcomes a different set of weights (preferences) may provide, which further justifies its usage instead of the single-objective optimization approach.

G. Target-oriented search

A practical evaluation of a solution (vulnerability rank) is to consider the *target-oriented search*, which measures how many vulnerability patches, following the rank order, are needed to reach a specified target value. This value represents the specific score goal an organization desires to attain for the considered environments. For instance, the initial scanned system's mean vulnerability risk can have a severity score that varies between 0 and 1000 (higher values indicate higher vulnerability severity), so, as an example, an organization can have a vulnerability mean risk score of around 600. If the organization wishes to lower this risk score mean value to 400 (in this case, this value represents the target value for particular metric), it must patch some vulnerabilities. The target-oriented search verifies each solution in the Pareto front, searching for the one that reaches the specified targets in the fewest possible vulnerabilities patches. To make this process computationally efficient, the resulting patching impacts of every vulnerability patch are previously computed for all solutions. This computation removes the vulnerabilities from the rank while recalculating new means for the considered metrics.

Different metrics besides the vulnerability risk can be considered in the multi-objective scenario, so the goal is to minimize simultaneously all targets corresponding to other objectives. Through these organization-defined targets and a set of considered solutions, this step computes, for each solution, how many vulnerabilities have to be patched to achieve the chosen targets. The *best solution* reaches the targets in the least amount of patches possible. Thus, the target-oriented search step is an extra functionality provided after the optimization process that complements the user's preferences scalarization, tailoring their needs to actual reportable patching impact results. A user-defined parameter called *number of solutions* is used to couple the objectives scalarization with the target-oriented search for assisting in this process. The *number of solutions* specifies how many solutions are to be considered in the target-oriented search after applying the objectives scalarization.

IV. Experiments and Results

This section presents a comparison between optimization algorithms, population initialization (PI) and post-optimization (PO) strategies to improve the vulnerabilities prioritization. The experiments intend to demonstrate that MOO can reach better improvements in the organization's predefined reduction targets with fewer vulnerabilities being patched than using standard prioritization strategies, achieving the organization's needs more efficiently.

A. Evaluation Methodology

The algorithms described in this work were coded in Python 3 and ran on an Intel(R) Xeon(R) E5-2620 v4 processor, with 32 cores and 2.10GHz, bundled with 128G of RAM. An organization has provided a real-world dataset extracted from one of its environments to test the method. This data contains $64,409$ vulnerabilities distributed throughout $3,583$ assets. Because of the dataset's size and the problem's representation, the solution vector ended up composed of $6,419$ decision variables, which puts it in the large-scale optimization problems category [12].

The real-world data used in the experiments come from lists of vulnerabilities found in a large business organization data center, which obtained the dataset regarding servers and vulnerabilities information from professional scanning softwares. As inputs for the method, this dataset is filtered to select the specific attributes that will be used as the following objectives:

- **Vulnerability risk**: a score that considers the vulnerability severity and exploitability.
- **Vulnerability age**: the measure of time elapsed since the discovery of the vulnerability in the data center.
- **Application importance**: the measure of how critical, confidential, or relevant the data, application, or asset impacted by the vulnerability is for the organization.

The vulnerabilities dataframe was preprocessed to remove every duplicate row, *i.e.*, condense all the vulnerabilities with the same optimized metrics values into a single agglutinated vulnerability. After the optimization process, when showing the prioritization rank for the final user, the single agglutinated vulnerabilities are expanded again into multiple ones, receiving the same ranking as assigned by a particular solution to that agglutinated vulnerability. This process reduced the number of unique entries from $64,409$ to $6,419$, which drastically decreased the number of decision variables being optimized. Nonetheless, the problem is still considered a large-scale optimization problem [12].

In the absence of a widespread benchmark for this area, the authors proposed the baseline used in the validation.

Therefore, the six best individuals (according to each isolated objective) and one random individual (choice based on the population initialization explained in Section III) were selected to represent the baseline.

The NSGA-II and AGE-MOEA algorithms' versions were obtained from pymoo (Multi-objective Optimization in Python library)[2] [2], which models the population size, the number of offsprings, an initial custom population, crossover, mutation, among others. For running the experiments, the method's parametrization followed the recommendations of Kazimipour et al. [12] for large-scale optimization problems, so it utilized the population size of 250 individuals. The chosen number of generations (300) is the same as tested in [17], and the remaining optimization parameters are the defaults offered by pymoo. Additionally, for the experiments considering the custom initial population, 10 random individuals were defined along with all possible combinations of the weights $\{0, 0.25, 0.5, 0.75, 1\}$ for the Borda count solutions, except those where all the three weights were equal (in this case just the Borda count weight tuple $\{1, 1, 1\}$ was considered) and the ones where there were two zeros (since just one objective would be considered). The following metrics were used to verify which experiments achieved the best results in reducing the number of vulnerabilities needed:

- **Hypervolume**: A metric widely used for assessing multi-objective optimization algorithms. It is a metric that considers the points' proximity to the Pareto front, its spread, and diversity. The elements with higher values represent the best results.
- **Vulnerabilities to reach the reduction targets (VRRT)**: A custom metric that considers the organization's goals. This metric considers the reduction of all the objectives (mean) predefined by the organization. In Table I, the reduction targets 1%, 5%, 10%, 15%, and 30% were considered according to the organization's needs. The values presented in the table's cells are the number of vulnerabilities needed to reach the determined reduction target. The lower the better.

The baseline experiments considered all possible compositions for sorting the objectives (described in Section III). The baseline "M1-M2-M3" is equivalent to patching vulnerabilities according to the sorted list of vulnerabilities by objective M1, breaking ties with objective M2 and then M3. Their evaluations considered only the patching impacts because each of these baselines is a single solution, and the hypervolume metric assesses the quality of a solutions' set. The metrics for the comparison between NSGA-II and AGE-MOEA were evaluated considering the mean and the standard deviation of 15 different executions, in which seeds were used to ensure reproducible results.

B. Main Results

The evaluation of the experiments demonstrated that the proposed method achieved better results than the baselines. The comparison between the baselines and the optimization algorithm's variants is shown in Table I. Comparing the resulting hypervolumes, it becomes clear that using the AGE-MOEA algorithm improves the results obtained by NSGA-II. However, the main contribution of the proposed method for the problem domain is depicted by the VRRT results. It is possible to see that the baselines have a hard time conciliating the conflicting metrics, having to patch significantly more vulnerabilities than the proposed method, even considering all its variations. This metrics disagreement behavior is also presented by Fig. 1, which shows how each metric's mean is affected as the vulnerabilities are patched following a specified solution's order. While the AGE-MOEA solution somewhat reduces the three metrics as more vulnerabilities are patched, the best baseline, considering the VRRT of 5%, can only deal with the reduction of one metric, evidencing its inefficacy for the complex problem of multi-criteria vulnerability prioritization. Compared to the baselines, the MOO achieved on average a 48,17% reduction in the vulnerabilities patches needed to reach the organization's target values.

C. Ablation Study

The population initialization and the post-optimization strategies are among the main contributions of this paper. Therefore, they were analyzed together and separately in Table I to ensure their significance in the results. It is possible to see, by analyzing the hypervolume results for AGE-MOEA, that the custom PI significantly improved the method, achieving the best results of the table. Besides, the addition of the PI component greatly impacted the results of AGE-MOEA for all the collected VRRTs. However, for NSGA-II, that was not the case since the method performed better without the PI component for all tested VRRTs, excluding the 10% one.

The PO influence on the results was very mild and without statistical significance. This was somewhat expected since it only fixes clearly out-of-order vulnerabilities from solutions that were already optimized. This way, the PO component can achieve the best results possible if there is room for more computations, but it is not crucial for achieving similar quality results. On the other hand, the PI was very important for achieving high-quality solutions with AGE-MOE and, since it is very computationally efficient, its usage is highly recommended.

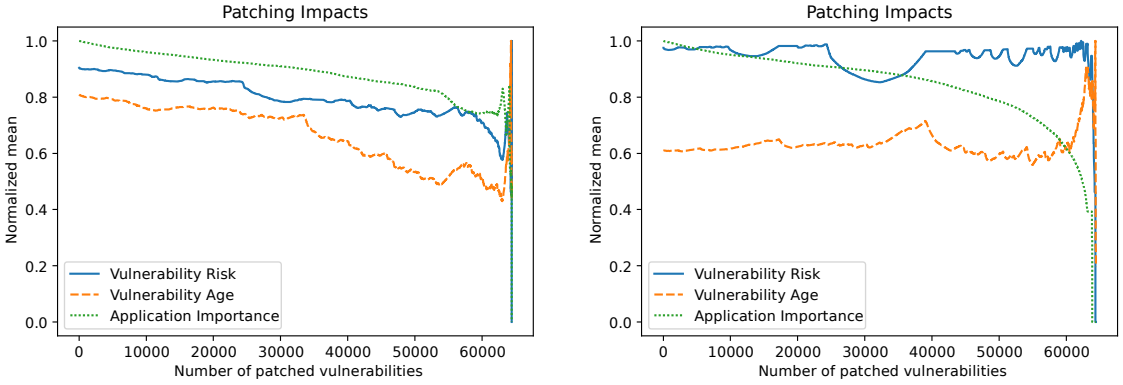
V. Related Work

The vulnerability assessment area has related work that contemplate methods for computing vulnerabilities risk [1, 20, 23]. Although these works do not use MOO, they were considered theoretical bases for interpreting how to evaluate the vulnerabilities risk. The MOO methods are not commonly used for evaluating vulnerability risk prioritization; however, certain works can contribute to this area [7]. Viduto et al. [22] used MOO to choose which countermeasures should be adopted considering the risk assessment cost, while Farris et al. [5] suggested a framework that evaluates which vulnerabilities should be patched or not.

[2]https://pymoo.org/algorithms/moo/age.html#nb-agemoea

TABLE I: Comparison between the baselines and MOO experiments considering an evaluation of the $64,409$ vulnerabilities using the Hypervolume and different percentages of VRRT. Highlights in bold indicate the best results and highlights in gray, the results with no significant statistical difference ($p < 0.01$) considering a Kruskal Wallis test with Bonferroni correction.

Experiments		Hypervolume	Vulnerabilities to reach the reduction targets (VRRT)				
			1%	5%	10%	15%	30%
Baseline	**M1-M2-M3**	–	59520	64409	64409	64409	64409
	M1-M3-M2	–	64340	64340	64340	64350	64350
	M2-M1-M3	–	30520	53440	64409	64409	64409
	M2-M3-M1	–	30420	62580	64350	64350	64409
	M3-M1-M2	–	44940	48170	64340	64350	64350
	M3-M2-M1	–	27840	63890	64360	64370	64380
	random	–	59017.87 ± 4911.09	64186.93 ± 309.17	64382.0 ± 77.34	64391.33 ± 53.37	64409.0 ± 0.0
NSGA-II	**standard**	4.88 ± 0.02	2251.33 ± 112.56	18302.0 ± 1278.77	36427.33 ± 1184.29	56423.33 ± 1892.16	64021.33 ± 206.6
	PI	4.88 ± 0.02	2300.0 ± 149.95	18442.0 ± 1013.13	35846.67 ± 1412.74	57981.33 ± 2419.94	64070.0 ± 93.73
	PO	4.88 ± 0.02	2250.0 ± 112.69	18300.67 ± 1277.54	36424.0 ± 1184.27	56419.33 ± 1889.1	64024.0 ± 202.72
	PI + PO	4.89 ± 0.02	2272.0 ± 126.78	18424.0 ± 1008.21	35847.33 ± 1416.89	57980.67 ± 2413.54	64068.67 ± 93.87
AGE-MOEA	**standard**	5.01 ± 0.01	2171.33 ± 149.33	17815.33 ± 1099.77	37945.33 ± 2034.79	57770.67 ± 1384.62	64201.33 ± 68.65
	PI	**5.38 ± 0.01**	**2066.67 ± 87.97**	**14994.67 ± 1065.98**	**32874.0 ± 1284.79**	46604.67 ± 2281.88	62451.33 ± 912.81
	PO	5.01 ± 0.01	2170.0 ± 149.57	17796.0 ± 1088.78	37945.33 ± 2034.79	57770.67 ± 1384.62	64198.67 ± 68.54
	PI + PO	**5.38 ± 0.01**	2067.33 ± 89.08	14995.33 ± 1065.54	**32874.0 ± 1284.79**	**46604.0 ± 2280.46**	**62448.67 ± 911.83**



(a) AGE-MOEA (PI + PO)

(b) Baseline (M3-M1-M2)

Fig. 1: Patching impacts on each adopted metric's mean. The values of the metrics were normalized to enable comparison among them. The plot on the left shows the best solution of the first AGE-MOEA (PI + PO) experiment, considering the VRRT of 5%. The plot on the right shows the best baseline, considering the same VRRT.

Viduto et al. [22] suggested a countermeasures prioritization approach using Multi-objective Tabu Search (MOTS), which was used to minimize the total investment cost and risk of a vector of vulnerabilities. Therefore, using this algorithm, a Pareto front can be built considering all solutions visited and the ones removed for being dominated. The algorithm's output is a set of countermeasures obtained by evaluating the cost and risk trade-offs.

Farris et al. [5] proposed a vulnerability prioritization framework that integrates output data from a vulnerability scanner tool with user-defined preferences, considering multiple possible prioritization scenarios and recommending the best set of vulnerabilities to be patched for that particular context in a period. The estimated personal hours to remediate vulnerabilities were collected by expert interviews with security engineers among the optimization metrics. While this is an important metric, it depends heavily on the security experts, and it could be impossible to enable such assessment for large organizations with possibly millions of vulnerabilities. The framework's output is an array representation that contains which vulnerabilities should be patched and not the patching prioritization rank; however, in this case, the algorithm established in advance an order based on a utility score. Both papers presented multi-objective problems; nonetheless, their solutions outputs are not related to ranking all the vulnerabilities risk prioritization, so they were used only as theoretical guidelines for MOO.

The vulnerability prioritization ranking is an innovation of this paper. Thus, considering the flexibility achieved from using it, the method can evaluate more scenarios since it does not need to use a previously specified combination of metrics. The main difference between this paper's method and the related work is that, while the other works have established framework-defined metrics, our method can use different combinations of user-defined metrics.

VI. Conclusion

This paper proposed a method that handles the vulnerabilities prioritization problem using a multi-objective approach. The experiments and results show that our method reduces

the number of vulnerabilities that must be patched. One of the main contributions perceived in the results is that our method can prioritize more significant vulnerabilities to reach the organization's predefined reduction targets. A comparison between the baselines, optimizers, and strategies was presented to determine the impact of our method. The hypervolume and the VRRT metrics were used to validate which algorithm's configuration would achieve the best results. The proposed method using the AGE-MOEA optimizer, the population initialization, and the post-optimization strategy achieved the best patching results. These improvements have real-world implications, which can assist the security teams in identifying the most critical vulnerabilities while avoiding costs and time consumption to validate the best solution. It can also help the final users since they will be less exposed to threats and have access to a more dynamic decision-making process.

Despite the efforts to customize the user-defined objectives, our method has the limitation of not dealing with dynamic metrics, in which a simple sorting does not describe the perfect rank efficiently according to that specific metric. In the future, the authors intend to research a solution for the presented limitation, evaluate more multi-objective and many-objective algorithms and investigate other methods for the population initialization and fitness function. The method will be validated with other datasets and application areas to identify adjustments and improvements.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq, Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP (2020/05165-1), and Dell Inc. via the 18th Amendment to the Technical and Scientific Cooperation Agreement No. 01/2017 – Information Technology Innovation Support Law – Brazilian Government.

References

[1] Alexander Beck and Stefan Rass. Using neural networks to aid cvss risk aggregation—an empirically validated approach. *Journal of Innovation in Digital Ecosystems*, 3(2):148–154, 2016.

[2] J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

[3] Felipe Colombelli et al. A hybrid ensemble feature selection design for candidate biomarkers discovery from transcriptome profiles. *arXiv preprint arXiv:2108.00290*, 2021.

[4] Kalyanmoy Deb et al. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[5] Katheryn A Farris et al. Vulcon: A system for vulnerability prioritization, mitigation, and management. *ACM Transactions on Privacy and Security (TOPS)*, 21(4):1–28, 2018.

[6] Park Foreman. *Vulnerability management*. CRC Press, Boca Raton, FL, 2019.

[7] Bruno Grisci et al. Perspectives on risk prioritization of data center vulnerabilities using rank aggregation and multi-objective optimization. *arXiv preprint arXiv:2202.07466*, 2022.

[8] Bruno Iochins Grisci et al. Relevance aggregation for neural networks interpretability and knowledge discovery on tabular data. *Information Sciences*, 559:111–129, 2021.

[9] Conor F Hayes et al. A practical guide to multi-objective reinforcement learning and planning. *arXiv preprint arXiv:2103.09568*, 2021.

[10] IBM. Cost of a data breach report 2020, 2020. URL https://www.ibm.com/security/data-breach.

[11] Jay Jacobs et al. Exploit prediction scoring system (epss). *arXiv preprint arXiv:1908.04856*, 2019.

[12] Borhan Kazimipour et al. Effects of population initialization on differential evolution for large scale optimization. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 2404–2411. IEEE, 2014.

[13] Maurice G Kendall. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.

[14] Triet HM Le et al. A survey on data-driven software vulnerability assessment and prioritization. *arXiv preprint arXiv:2107.08364*, 2021.

[15] Xue Li et al. A comparative study of rank aggregation methods for partial and top ranked lists in genomic applications. *Briefings in bioinformatics*, 20(1):178–189, 2019.

[16] Shili Lin. Rank aggregation methods. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(5):555–570, 2010.

[17] Annibale Panichella. An adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 595–603, 2019.

[18] Gabriel de O. Ramos et al. Toll-based reinforcement learning for efficient equilibria in route choice. *The Knowledge Engineering Review*, 35:e8, 2020.

[19] Diederik M Roijers et al. Multi-objective decision making. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 11(1):1–129, 2017.

[20] Georgios Spanos et al. Wivss: a new methodology for scoring information systems vulnerabilities. In *Proceedings of the 17th panhellenic conference on informatics*, pages 83–90, 2013.

[21] Tripwire. Tripwire 2019 vulnerability management survey, 2019. URL https://www.tripwire.com/state-of-security/wp-content/uploads/sites/3/Tripwire-Dimensional-Research-VM-Survey.pdf.

[22] Valentina Viduto et al. A novel risk assessment and optimisation model for a multi-objective network security countermeasure selection problem. *Decision Support Systems*, 53(3):599–610, 2012.

[23] Jiao Yin et al. Apply transfer learning to cybersecurity:

Predicting exploitability of vulnerabilities by description.
Knowledge-Based Systems, 210:106529, 2020.