Improving Attack Graph-based Self-Protecting Systems: A Computational Pipeline for Accuracy-Scalability Trade-off

Silvia Bonomi[1] [0000−0001−9928−5357], Marco Cuoci[1] [0009−0009−4566−4373], Simone Lenti[1] [0000−0001−8281−3723], and Alessandro Palma[1] [0000−0002−9104−9904]

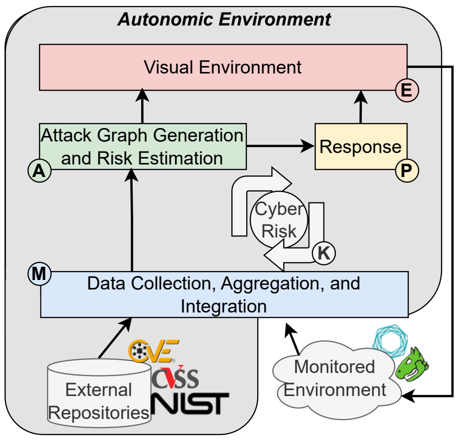
Sapienza University of Rome, Italy
{bonomi, cuoci, lenti, palma}@diag.uniroma1.it

Abstract. Self-protection is a desired property of many modern ICT systems as it enriches them in detecting and reacting to security threats at run-time. Several solutions leveraging vulnerability scanners and attack graphs have recently been proposed to monitor and analyze cyber risks and trigger security adaptations accordingly. They mainly focus on the system design without investigating the potential drawbacks of their components, such as accuracy and scalability. This paper investigates the accuracy of the environment monitoring, the scalability of the security analysis, and their intrinsic relationships. To balance their trade-off, we contribute a computational pipeline that includes vulnerability filtering and aggregation modules that can be used in isolation or combined to tune the monitoring and analysis of Attack Graph-based self-protecting systems. We propose different heuristics for filtering and aggregation, each impacting the accuracy-scalability trade-off at various levels, and we assess their interplay in a real-setting scenario.

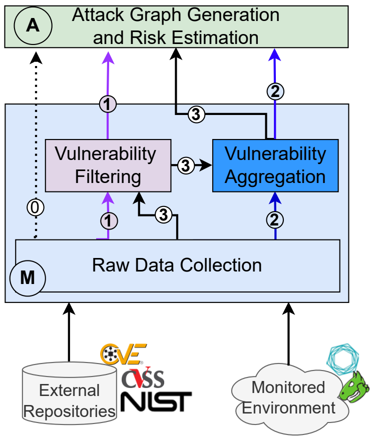
Keywords: Self-Protecting Systems · Attack Graph · Risk Estimation · Risk Accuracy · Scalability.

1 Introduction

Managing security threats and protecting organizations from cyber attacks is becoming complex due to the continuously evolving vulnerability landscape and the always-increasing capabilities of attackers. *Self-protecting systems* have gained a lot of interest as they allow an ICT system to autonomously adapt to an always-evolving environment, thus being responsive, agile, and cost-effective [18]. A self-protecting system has the intrinsic capabilities of (i) monitoring an ICT system, (ii) analyzing its current security level (identifying and quantifying risks), (iii) planning proper countermeasures, and (iv) supporting the enforcement of such a response plan. These capabilities are commonly employed by leveraging the *Monitor-Analyze-Plan-Execute over a shared Knowledge* (MAPE-K) architecture [2,14]. Following this paradigm, several self-protecting systems have been proposed based on the Attack Graph (AG), a graph-based model of the potential attack steps in a network [6,30]. Fig. 1a shows the main building blocks characterizing these solutions. In this type of architecture,



(a) MAPE-K for AG-based self-protection.



(b) Computational pipeline for the DCAI level.

Fig. 1: The MAPE-K architecture for AG-based self-protection systems and the computational pipeline for the *Data Collection, Aggregation, and Integration* level highlighting the four supported workflows.

it is fundamental to guarantee that the cyber risk estimation (i.e., the variable controlling the feedback loop) is as accurate as possible to ensure the effectiveness of self-protection. However, this accuracy requirement contrasts with the performance of the autonomous system. Indeed, AGs are very powerful and potentially accurate models, but they suffer from scalability issues for which it is often necessary to compute an *approximate* version of them [9,24]. This approximation introduces a certain degree of uncertainty in the risk estimation, where accuracy is traded off for scalability. The situation worsens if we consider that state-of-the-art self-protecting systems feed the control loop with input data that are not entirely accurate due to false positives and negatives affecting the monitoring probes. These factors cumulatively worsen the accuracy, affecting the data collection phase first and then the AG and risk estimation.

To address this problem, we analyze the accuracy-scalability trade-off in AG-based self-protecting systems governed by risk estimation, which is generally not considered in existing solutions. We propose a computational pipeline for the *Data Collection, Aggregation, and Integration* level (Fig.1b) whose aim is twofold: (i) improving the accuracy of the risk estimation by enhancing the quality of the data feeding the control loop (and in particular those used as input in the AG generation) by reducing the number of false positives through a semi-automatic validation process and (ii) improve the scalability of the self-protecting system by reducing the size of the input processed by the *AG Generation and Risk Estimation* level. To this aim, we define two modules, namely *Vulnerability Filtering* and *Vulnerability Aggregation*, that can be used either in isolation or in combination, and for each component, we propose algorithms. These algorithms work on the input data for the control loop by *sanitizing and compressing* them, trying to avoid the loss of information (i.e., preventing and quantifying the accuracy loss in risk estimation). Finally, we perform an experimental evaluation to highlight the pipeline's advantages and the contribution of each block to the definition of the accuracy-scalability trade-off.

After describing background concepts (§2) and related work (§3), we present the proposed computational pipeline (§4) and the algorithms for filtering (§4.1) and aggregation (§4.2). We validate the approach (§5) and conclude the paper (§6).

2 Background Notions

Attack Graph Model. An *Attack Graph* (AG) provides a graph-based representation of how an attacker may intrude and compromise network elements through a sequence of vulnerability exploits and reach the attack target. It is modeled considering (i) a network *Reachability Graph* and (ii) a *Vulnerability Inventory*. The former is a directed graph where nodes represent network hosts, and each edge represents direct communication between two hosts; the latter is a collection of vulnerabilities associated with network hosts. Starting from these inputs, an *attack graph computation algorithm* generates the resulting attack graph $AG=(P,Ex)$ as the directed graph where the nodes are levels of privilege that an attacker has on a specific host and an edge represents a possible exploit enabled by a vulnerability [12,37]. The privileges typically considered in AG are *guest*, *user*, and *root* [33,36]. Once AG is modeled, we are interested in the computation of its paths, namely *attack paths*. An Attack Path $AP_{i,j}$ represents the attack steps that an attacker must sequentially take to successfully perform an attack from a *source* node p_i to a *target* node p_j in AG. The most common approach to compute attack paths is the Breadth-First Search (BFS) algorithm, either starting from a set of predefined attack sources [10] or by performing a backward search from the attack targets [11].

Risk Model. Once attack paths have been computed, they are used to estimate the cyber risks of the different attacks on the network. We leverage existing approaches [6] that consider CVSS metrics[1] to estimate the likelihood and impact of an attack path AP and calculate the risk according to its standard definition: $risk(AP)=likelihood(AP)\cdot impact(AP)$. The likelihood is calculated using the CVSS-3.1 exploitability metrics (Attack Vector, Attack Complexity, Privilege Required, and User Interaction), while the impact is determined by CVSS-3.1 impact metrics (more details in [6]). Let us consider the following observations: (i) The accuracy of the risk estimation is impacted by the completeness of the AG, i.e., the (approximated) set of attack paths; (ii) The *completeness* of an AG (in terms of existing attack paths) is affected by the presence of false positives/negatives in the input and the algorithm used to compute AG and its attack paths; (iii) AG algorithms do not scale when the network size and the number of vulnerabilities increases [12,37]. From these observations, it follows that the completeness of the AG can be enhanced by improving the quality of the necessary inputs and considering generation algorithms that can produce a *full attack graph*, that is the AG without any approximation. However, when the system scale

[1] `https://www.first.org/cvss/v3.1/specification-document`

increases, full AGs cannot be computed efficiently, i.e., in a predefined small amount of time); thus, completeness and the accuracy of risk estimation are traded for scalability.

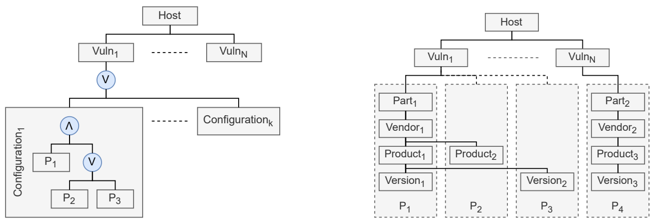
3 Related Work

Self-protecting system. In the literature, different works design self-protecting systems for different goals [27]. They focus on specific domains, as Yuan et al. [34] and English et al. [4], who propose self-protecting architectures to automatically detect and mitigate cyber threats of software (the former) and provide mitigation actions based on event correlation (the latter). Similarly, Liang et al. [22] present a framework to enhance the self-protection of power systems by leveraging blockchain technology. Other works develop self-protection strategies for securing data storage. Strunk et al. [31] minimize the performance costs of system versioning, and Kocher et al. [17] leverage watermarking as a protection strategy. Differently, Li et al. [20] propose a self-adaptation framework using Bayesian games to model cyber attacks, and Girdler et al. [5] propose an IDS to dynamically detect malicious network traffic and adapt the blacklist of malicious devices to block them.

A recent trend employs the Attack Graph (AG) model in self-protecting systems. For example, Gonzalez-Granadillo et al. [6] provides the baseline architecture for dynamic cyber risk analysis leveraging AGs. Zeller et al. [35] employ AG in a self-protecting system to analyze the risks of logic vulnerabilities and apply reconfigurations at runtime. In contrast, Skandylas et al. [30] introduce a formal approach to designing AG-based self-protecting systems. Finally, Kholidy [16] models AG and self-protection for Cyber-Physical systems, while Khakpour et al. [15] use threat modeling to assess the risks of transient configurations during adaptation steps. While these systems are valuable solutions to apply AGs and risk estimation for self-protection, they do not consider the accuracy of risk estimation and the scalability of AG computation.

Attack-graph scalability. A natural direction to improve the performance of AG-based systems is by adopting scalable algorithms, such as shortest paths [32], path pruning [12], or considering distributed approaches [19,26]. Other solutions propose to compute attack paths based on alerts coming from detection systems, such as Nadeem et al. [25] and Moskal et al. [24] who translate alert events to episode sequences used to build a finite automaton of the AG, or employ AI to predict attack paths instead of generating them [21,28]. All these approaches implicitly assume that input data is accurate and thus trade the efficiency for an approximated solution. In addition, they do not assess and quantify the accuracy-scalability trade-off. Thus, it is difficult to estimate the accuracy loss when they are used in self-protecting systems. The approach proposed in this paper takes an orthogonal perspective, focusing on input data used by AG algorithms to improve their quality and reduce their size.

Accuracy of AG input data. State-of-the-art solutions do not consider pre-processing the inputs of the attack graph computation algorithm to improve their quality. To the best of the authors' knowledge, the primary solution for this problem is by leveraging security expert's knowledge to identify false positives from vulnerability



(a) Existing model [3]. (b) Proposed tree model.

Fig. 2: Example of existing and proposed tree models for vulnerability filtering.

scanners by reducing as much as possible the number of interactions [3,8]. It models the dependencies between vulnerabilities and vulnerable platforms in the network and uses a dependency model to drive the interaction with the user. Inspired by this solution, we advance the current state-of-the-art to consider the accuracy of risk estimation in the proposed computational pipeline.

4 Computational Pipeline

Let us consider the architecture of the self-protecting system shown in Fig. 1a. In this paper, we propose a modular computational pipeline for the *Data Collection, Aggregation, and Integration* (DCAI) component aiming at (i) enhancing the accuracy of risk estimation and (ii) improving its overall scalability. The proposed pipeline has two main components: (i) a *Vulnerability Filtering* block and (ii) a *Vulnerability Aggregation* block. They can be orchestrated with the pre-existing sub-components of DCAI to generate different data preparation workflows characterized by varying levels of accuracy. Fig. 1b reports the pipeline overview. The main task of the DCAI component is to produce all data needed in the following steps of the control loop to compute the attack graph, estimate the risk, and finally define countermeasures. Among all those data, we focus on the *Vulnerability Inventory* $\mathcal{VI}$ used to generate the AG and estimate the risk. The following will refer to data extracted with available DCAI solutions as *Raw Data Collection*. The term "raw" refers to the existing data collection implementation that mainly focuses on fusing and integrating data from multiple sources [24,25]. Workflow 0 in Fig. 1b will, thus, represent our analysis baseline (i.e., the current state of the art for AG-based self-protecting systems). Let us note that data produced by the *Raw Data Collection* may be inaccurate due to *false positive*, i.e., vulnerabilities that are not affecting the system or not exploitable.

The *Vulnerability Filtering* component aims to reduce the number of false positives deriving from the raw data collection. Vulnerability scanners automatically identify vulnerabilities in a network by looking for known vulnerabilities and misconfigurations [23]. They could generate such false positives due to, for example, (i) misconfiguration, (ii) misinterpretation of the standard system behavior, (iii) dynamic content of systems' applications, (iv) temporary conditions on the network, and (v) lack of context [1]. False positives in the vulnerability inventory have a double disadvantage. On one side, they reduce the accuracy of the risk estimation as it is performed on a set of vulnerabilities that do not exist. On the other hand, they increase the vulnerability inventory size, thus negatively impacting the scalability of the subsequent AG-based analysis.

The *Vulnerability Aggregation* component aims to further improve the scalability of the overall self-protecting system by reducing the size of the input processed by the attack graph generation and risk estimation module and, in particular, the vulnerabilities in the inventory $\mathcal{VI}$, by aggregating vulnerabilities according to semantic criteria. The core idea is to aggregate a set of vulnerabilities with common features (e.g., that affect the same host and have the same pre and post-conditions) into a *meta-vulnerability* characterized by such features. Let us note that multiple aggregation policies may exist, and we want to maximize the number of vulnerabilities aggregated into a meta-vulnerability while minimizing the potential accuracy loss.

The Vulnerability Filtering and the Vulnerability Aggregation components can be orchestrated to support three different computational workflows as shown in Fig. 1b.

WF1 involves only vulnerability filtering (workflow 1 in Fig. 1b). It can be executed for proactive security analysis with unlimited resources for AG computation.

WF2 involves only vulnerability aggregation (workflow 2 in Fig. 1b). It can be executed for real-time analysis where the computation of AGs must be reduced as much as possible, and the analysis is fully automated without potential delays introduced by vulnerability filtering. This workflow is also suitable when the accuracy of the inventories generated by the raw data collection is particularly high.

WF3 involves both the vulnerability filtering and the vulnerability aggregation components (workflow 3 in Fig. 1b). It represents the most general case that depends on the specific context and is based on the definition of a trade-off between the accuracy and performance of the risk analysis.

4.1 Vulnerability Filtering

The Vulnerability Filtering component proposed in this paper advances the idea presented by Bonomi et al. [3], where a raw vulnerability inventory is improved through user-based validation leveraging a graph-based dependency model that captures the relationships between the vulnerabilities and the vulnerable platforms they affect. Usually, asset management systems are leveraged to retrieve each host's platform configurations automatically [7]. In some cases, this is not possible (e.g., due to non-intrusiveness requirements) and human intervention is necessary: for example, some architectures are unsupported by these systems (e.g., old platforms) or cannot be used (e.g., air-gap networks). In such situations, the user is asked to validate specific platforms' presence which will then be checked against the dependency model to confirm or discard related vulnerabilities. The main challenge is to design an algorithm that raises the smallest number of platform validations to reduce the filtering time. Bonomi et al. [3] leverage the Common Platform Enumeration (CPE)[2] to model the dependencies between platforms and vulnerabilities. They select a platform from a device inventory for a host h, ask the user whether it is present in the host, and apply the outcome to each vulnerability in the vulnerability inventory $\mathcal{VI}$ for host h.

[2] `https://cpe.mitre.org/`

Currently, this approach does not consider optional attributes of a platform (e.g., versioning), and thus may result in an inaccurate analysis as false positive vulnerabilities may persist (e.g., vulnerabilities with non-compatible platform versions). This paper proposes a new tree-based model to track dependencies between vulnerabilities and vulnerable platforms that considers every platform attribute from CPE. More in detail, the vulnerability filtering component will take as input a vulnerability inventory $\mathcal{VI}_{in}$ listing all the hosts in the network and where each host h_i has attached a list of vulnerabilities $\langle u^i_1, u^i_2, \ldots u^i_n\rangle$, and will produce as output a new vulnerability inventory $\mathcal{VI}_{out}$ with the same format that will be the subset of $\mathcal{VI}_{in}$ including only vulnerabilities validated by the user. The key idea behind the new dependency model is to represent the platforms and their attributes in a tree-shaped graph routed on the host identifier. For each host h_i in the input vulnerability inventory $\mathcal{VI}_{in}$, we compute the *vulnerability tree* $\mathcal{T}_i = (V_i, E_i)$, where V_i is partitioned into three subsets: (i) *root nodes* V_r including the host h_i currently under analysis; (ii) *vulnerability nodes* V_u including a node for every vulnerability u listed in the input vulnerability inventory $\mathcal{VI}_{in}$; (iii) *platform nodes* V_{part}, V_{vend}, V_{prod}, V_{vers} including a node respectively for every part, vendor, product and version attribute appearing in at least one platform retrieved starting from a vulnerability in the input vulnerability inventory $\mathcal{VI}_{in}$.

The set of edges E_i is partitioned into the following subsets: (i) $E_{h,u}$ is the set of edges linking a host h to its vulnerability u; (ii) $E_{u,P}$ is the set of edges linking a vulnerability u with its platform P; (iii) E_P is the set of edges derived from a platform P and connecting (through a path) a part node to a vendor node, to a product node, and finally to a version node. Since they form a sub-tree in $\mathcal{T}_i$, we refer to these edges as *platform sub-tree*.

The main difference between the existing model (Fig. 2a) and the proposed one (Fig. 2b) is that the latter highlights commonalities between different platforms, thus being more fine-grained and speeding up the validation of multiple platforms.

Vulnerability filtering is made of the following actions: (i) identify the *best* validation request to be submitted to the human expert; (ii) re-plan based on the received answer. Planning is done by considering a utility function that estimates the individual contribution of each platform to the validation of the pending vulnerabilities and selecting the ones with the highest estimated profit (i.e., the number of validated vulnerabilities with a single request). Let us note that while the worst-case performance derives from asking any platform, we propose a greedy approach that improves the performance (on average) thanks to the dynamic re-ordering of requests based on their expected profit. In particular, we evaluated the following utility functions, namely:

- **RN** (RaNdom filtering): we order platforms with random uniform distribution.
- **SR** (Smart Random filtering): we apply the same process of **RN**; however, platforms that have no impact in the validation process are skipped.
- **VS** (Vulnerability Severity filtering): we order the platforms prioritizing those having the most critical vulnerabilities.
- **PP** (Platform Prioritization filtering): we order the platforms prioritizing the most frequent ones.
- **VP** (Vulnerability Platform filtering): we apply **VS** and consequently **PP** for the platforms pertaining to vulnerabilities of equal severity.

– **PV** (Platform Vulnerability filtering): we apply **PP** and consequently **VS** for the platforms having equal prioritization score.

Let us note that the considered utility functions eventually provide the same output, i.e., the set of false positives to be removed, but they converge with a different speed to the final result as some may be more or less effective in the early filtering requests (see Section 5).

4.2 Vulnerability Aggregation

The core idea of the vulnerability aggregation component is to reduce the size of the vulnerability inventory by aggregating vulnerabilities with shared features retrieved by external repositories (see Fig. 1b). This paper considers a vulnerability u with an *id* and a set of *attributes* $m_1, \cdots, m_n$ quantifiable with numerical values. Without loss of generality, we assume that the attribute values are determined according to predefined scales with the rationale that a higher value corresponds to a higher risk estimation. This information is typically defined with the Common Vulnerabilities and Exposures (CVE)[3], where the attributes correspond to CVSS metrics (see Section 2).

Thus, the input of the vulnerability aggregation component is a vulnerability inventory $\mathcal{VI}$ (either coming directly from the data collection, i.e., workflow 2 in Fig. 1b or after filtering false positives, i.e., workflow 3. The output is an aggregated vulnerability inventory $\mathcal{VI}_a$ that has, for each host, a collection of *meta-vulnerabilities*. A meta-vulnerability has the same set of attributes of vulnerabilities in $\mathcal{VI}$, but the values of such attributes are processed according to an *aggregation strategy*. Let us recall that the aggregation module aims to reduce the size of the vulnerability inventory to improve the scalability of the AG computation. Still, at the same time, it should avoid losing any information that may lead to an accuracy loss for risk estimation. Thus, the technical challenge here is to design aggregation strategies that optimize the scalability-accuracy trade-off of the whole risk estimation process. We propose three main strategies considering different aspects of the risk model:

– **VA** (Vulnerability-based Aggregation): For each host, we aggregate the vulnerabilities that have the same values for *all* their attributes. Two vulnerabilities with the same attribute values are aggregated into a meta-vulnerability with such values. Otherwise, no aggregation is performed. As a consequence, each meta-vulnerability is a representation of the unique combinations of vulnerability attribute values of each host.
– **RA** (Risk-based Aggregation): For each host, we aggregate vulnerabilities with the same values of the attributes used to *evaluate the risk*. The resulting meta-vulnerability has the same value for the attributes used to assess the risk and the maximum value between aggregated vulnerabilities for all the other attributes.
– **HA** (Host-based Aggregation): For each host, we aggregate all the vulnerabilities *independently* from their attributes. The resulting meta-vulnerability has the maximum value of the attributes among the aggregated vulnerabilities. As a consequence, each host has a single meta-vulnerability.

[3] `https://cve.mitre.org/`

The rationale of these aggregation strategies (particularly for RA and HA ones) is that they consider the worst-case scenario. The maximum values of vulnerability attributes lead to higher risk values, according to the rationale of CVSS metrics. Consequently, meta-vulnerabilities tend to overestimate the risk, thus defining an upper bound for the analysis. Stricter aggregation strategies result in more controlled risk overestimation. Thus, VA and RA aggregation strategies represent the risk values with higher accuracy despite generating larger AGs. In contrast, the HA strategy overestimates the risk more than the other. Still, it generates AGs with the same size as the reachability networks, thus avoiding their exponential explosion.

As an example, let us consider a host h with vulnerabilities u_1, u_2, and u_3 in Table 1 and their metrics m_a, m_b, and m_c, where only m_a and m_b are used for risk estimation. The different aggregation strategies generate the following meta-vulnerabilities:
VA: no aggregation is performed because the metric values of the three vulnerabilities are mutually different.
RA: vulnerabilities u_2 and u_3 are aggregated into a meta-vulnerability $u^*_{RA,1}$ with metrics $\{m_a : 0.7, m_b : 0.3, m_c : 0.7\}$, while vulnerability u_1 is not aggregated into $u^*_{RA,1}$ because has a different value of the risk metric m_b.
HA: all the vulnerabilities of host h are aggregated into a meta-vulnerability $u^*_{HA,1}$ with metrics $\{m_a : 0.7, m_b : 0.7, m_c : 0.7\}$.

Table 1: Example aggregation.

	Risk metrics		
	m_a	m_b	m_c
u_1	0.7	0.7	0.3
u_2	0.7	0.3	0.7
u_3	0.7	0.3	0.3

5 Experimental Evaluation

5.1 Setting Configuration

To create the experimental environments, we emulated our department's real network. It comprises around 250 devices arranged in 5 LANs connected through a router in a star topology. We created 8 virtual representations of the network devices, configuring them with platforms and settings mimicking the real ones. The virtual hosts have been scanned for vulnerabilities using commercially available vulnerability scanners (Nessus and OpenVAS). False positives were manually filtered out to obtain the subset of vulnerabilities truly affecting them (i.e., to create a ground truth used to compute our accuracy metrics). Table 2 summarizes the configurations of the hosts and the number of (false and true) platforms and vulnerabilities identified by the scanners. To test the scalability, we considered different LAN networks (i.e., networks with full reachability between hosts) of various sizes and included multiple replicas of the hosts. We regarded as such topology for different motivations: (i) a LAN setting is the typical setup used for the validation of the state-of-the-art AG algorithm [13,29] and (ii) LANs represent a standard infrastructure of more extensive enterprise networks and are currently the parts with the highest impact on scalability. However, additional experiments on different topologies are reported on the open-source repository[4] and do not show significant differences from the reported results. The proposed dataset we made available represents general ICT infrastructures commonly employed and used in AG-based (distributed) approaches [13,37]. We compare the proposed pipeline

[4] `https://github.com/ds-square/self-protecting-ag`

Table 2: Experimental environment. True positives (TP) and false positives (FP) are identified by the scanner and confirmed or discarded by manual validation.

Host	**OS**	**Category**	**LAN Patecipation**	**Platforms**		**Vulnerabilities**	
				TP	FP	TP	FP
H0	Ubuntu Linux 22.10	office	ADMIN, INF, GST, AUTO	2	17	6	1
H1	Debian Linux 10	office	ADMIN, INF, GST, AUTO	3	96	11	5
H2	Ubuntu Linux 18.04.6 LTS	server	DMZ	5	137	12	3
H3	Debian Linux 10	server	DMZ	4	138	6	4
H4	Ubuntu Linux 15.10	development	INF, GST, AUTO	10	116	214	4
H5	Debian Linux 11	development	INF, GST, AUTO	1	15	1	1
H6	Ubuntu Linux 15.04	generic	ADMIN, INF, GST, AUTO	21	407	334	7
H7	Debian Linux 11	generic	ADMIN, INF, GST, AUTO	8	220	11	7

with the current state-of-the-art for AG-based self-protecting systems [6]. We fed the state-of-the-art implementation (SoA in the plots) with the vulnerability inventory manually constructed as ground truth to compare the benefit of our approach to a solution that is not biased from initial false positives (i.e., we considered a good scenario for the state-of-the-art). We conducted the evaluation on a Linux server with an Intel(R) Xeon(R) Gold 6248 CPU 2.50GHz and 256 GB of memory.

5.2 Vulnerability Filtering Evaluation

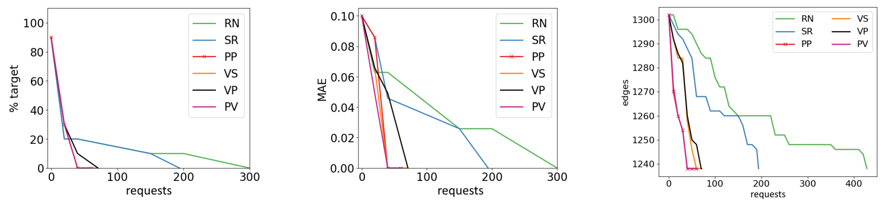
The first step of the pipeline (see Fig. 1b) is vulnerability filtering, whose goal is the improvement of the accuracy of risk estimation by filtering out false positives coming from vulnerability scanners. We examine how different utility functions affect the results of the cyber risk analysis and the AG size in a mesh of the 8 virtual hosts (a configuration resembling a small company with heterogeneous hosts).

As a first result, let us note that in addition to the 1605 vulnerabilities of SoA, the virtual environment contains 91 false positive vulnerabilities detected by the scanners. These vulnerabilities directly impact the risk estimation for eight of the ten attack targets (80%) in the environment with a Mean Absolute Error (MAE) of the risk estimation of 0.1 (i.e., the value of the risk of a target differs from its risk in SoA by 0.1 on average). This result underlines the importance of filtering false positives from vulnerability scans due to their negative impact on cyber risk evaluation. They may impact the risk estimation even more if the risk model does not consider the maximum likelihood for risk calculation (i.e., worst-case scenario).

The percentage of targets affected and the MAE are thus evaluated after each filtering utility function request. Fig. 3a shows the trend of the percentage of targets whose risk analysis is affected by false positives. It is noticeable that VS, VP, PP, and PV functions converge more rapidly than the others. To indicate the time, we experimentally estimated that a request is completed in 1 to 3 seconds by a human expert. In particular, they correctly estimate the risk for all the targets after 71, 69, 60, and 60 requests, respectively. In contrast, the other utility functions need from 200 up to 300 more requests to reach the same objective. This trend is because some functions prioritize the requests using algorithms considering vulnerability severity (VS, PV, and VP) and platform frequency (VP, PV, and PP). In contrast, functions

SR and RN do not apply any order to the sequence of requests, as described in Section 4.1. VS, VP, PP, and PV can, thus, validate more vulnerabilities in parallel, granting better performance.

Result 1: *The best functions have been observed to be PP and PV, with a convergence bound of* 60 *requests, against* 69 *requests of their closest competitor VP.*



(a) Percentage targets. (b) Risk variation. (c) Number of edges.

Fig. 3: Vulnerability filtering analysis in filtered AGs.

To determine the risk variation, we compute each host's MAE of the risk estimation. Fig. 3b shows the MAE after each request of the filtering utility functions. The first relevant aspect to point out is that the false positive vulnerabilities in the network cause an average approximation of the risk of 0.1%. In addition, it can be noticed that VS, PP, and PV functions quickly reduce the variation to zero after 71, 69, and 69 requests, respectively. This rapid convergence is because VS, PP, PV, and VP algorithms are designed to rapidly converge to the ground truth by suitably organizing the requests according to the most common features among the different vulnerabilities, enabling the detection of false positives early in the filtering approach.

Result 2: *The presence of false positives in vulnerability scans may cause an approximation of the cyber risk of* 0.1%*. Four filtering functions (VS, PP, PV, and VP) converge twice as rapidly as the others because they consider aspects common to many vulnerabilities, halving the time necessary to the filtering process.*

The elimination of false positive vulnerabilities has the direct effect of reducing the size of the AG, in addition to improving the accuracy of risk analyses. To measure the AG size reduction, we report the number of edges of the AGs partially filtered with the different filtering functions in Fig. 3c. The filtering functions reduce the number of edges from 1302 to 1238, corresponding to a space reduction of 5% of the AG. It is discernible that the convergence rate to the ground truth is comparable for both the number of edges (Fig. 3c) and the AG risk accuracy (Fig. 3a, Fig. 3b) indicating that the reduction of the AG edges follows the accuracy improvement. However, the two results are not equivalent because the filtering strategies contribute to eliminating the edges due to false positive vulnerabilities even though they do not directly contribute to the risk estimation of a host, i.e., if they do not contribute to the most severe threat for a host.

Result 3: *Filtering out false positive vulnerabilities provides advantages for AG scalability, reducing their size of up to* 5% *of their original size.*

This analysis showed the advantages of the filtering utility functions in the pipeline of self-protection. Without them, the current state-of-the-art solution is 10% less accurate, and the AG has 5% more edges. Among the different functions, PP and PV have the fastest convergence to the ground truth, resulting in a more accurate cyber risk analysis and reduction of the AG size.

5.3 Vulnerability Aggregation Evaluation

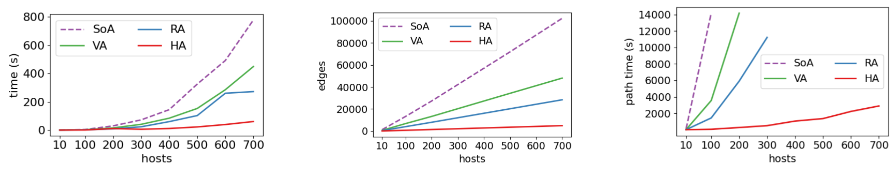
The main goal of vulnerability aggregation is to streamline the vulnerability inventory for generating AGs, enhancing their scalability over time. However, this aggregation can lead to information loss in cyber risk analysis. Thus, it's essential to examine the trade-off between scalability benefits and drawbacks for various aggregation strategies. Table 3 shows the reduction in edges and its impact on risk analysis (number of affected targets and MAE of risk estimation) compared to the state of the art for the 8 hosts mesh. It underlines a great advantage of vulnerability aggregation: the reduction between 68% and 96% of the AG size. Indeed, while the SoA attack graph has 620 nodes and 1238 edges, they are reduced to 196 and 390 for Vulnerability-based Aggregation (VA), 78 and 154 for Risk-based Aggregation (RA), and 27 and 44 for Host-based Aggregation (HA). However, the analysis confirms the side effect that aggregation strategies may have on the accuracy of the cyber risk analysis. The most conservative aggregation (VA) does not impact the estimation of the risk value because it aggregates only vulnerabilities that have the exact attributes; similarly, the impact of the RA strategy is minimal (1% of the hosts, MAE of 0.04) because it aggregates vulnerabilities according to their features used in the risk model. Conversely, the HA strategy has 40% of the cases with more than 0.5 risk variation. This worse accuracy is due to the aggregation being independent of vulnerability attribute values.

Table 3: AG risk accuracy.

Aggregation Strategy	**Affected Targets**	**Risk MAE**	**AG edges reduction**
VA	0%	0	68%
RA	10%	0.04	87%
HA	90%	0.44	96%

Result 4: *The VA strategy does not impact the risk estimation but has the lowest reduction of the AG size. The HA strategy has the biggest reduction of the AG size with a significant risk approximation. The RA strategy represents the accuracy-scalability trade-off with a considerable size reduction and a neglectable risk approximation.*

Considering the trade-off between scalability and accuracy, it is essential to delve deeper into the impact of aggregation strategies on the scalability of AGs. To achieve this objective, we analyze the time and space complexity of AGs generated using aggregated inventories. The scalability evaluation is performed by increasing the number of network hosts up to 700 still using the presented settings to resemble realistic modern ICT networks. The additional hosts were obtained by adding replicas of the hosts to the network. Fig. 4a reports the AG generation time (in seconds) for the different synthetic environments. The generation time includes the time necessary to aggregate the vulnerability inventory. The trends highlight the AG generation speed-up when adopting aggregation strategies, especially for large-size networks. On average, there is a reduction of 8%, 33%, and 78% of the generation time for Vulnerability-based (VA), Risk-based (RA), and Host-based (HA) Aggregation with respect to SoA, (dashed line in Fig. 4a). It is worth noting that the generation time is



(a) AG generation time. (b) AG edges. (c) AP computation.

Fig. 4: Analysis of AG (a) generation, (b) num. edges, and (c) path computation.

considerably increased when there is no aggregation (SoA), for VA and RA strategies, while increases linearly for the HA one. This result is because the HA strategy keeps one vulnerability per host making complexity linear in the network size. This trend is confirmed by the space analysis in Fig. 4b, which summarizes the number of AG edges for each aggregation strategy. The number of edges is reduced by 40%, 55%, and 75% when adopting VA, RA, and HA strategies, respectively.

Result 5: *Both time (Fig. 4a) and space (Fig. 4b) analyses indicate that adopting aggregation strategies provides a significant reduction of the AG structure (from* 40% *to* 75%*) and faster generation (from* 45% *to* 95%*).*

Concerning the accuracy-scalability trade-off of large networks, let us remark that computing the attack paths is computationally expensive when performed in non-aggregated networks, and it is still an open problem in AG [32,37]. Fig. 4c reports the time (in seconds) for the computation of the attack paths for 3 attack targets, randomly chosen, for the different network sizes. Since AGs are used to prevent and respond to cyber-attacks promptly, we assume that when the attack paths computation needs more than 4 hours to be completed, the problem is impractical. When there is no aggregation strategy (SoA, dashed line in Fig. 4c), the only size for which risk analysis can be computed in a practical amount of time is the network with 10 hosts, for which we reported the scalability-accuracy trade-off in the previous subsection. For networks with 100 hosts, the problem is impractical. In contrast, given the high reduction of AG size, it is still possible to compute attack paths for cyber risk analysis when adopting aggregation strategies. In particular, the problem is practical for up to 200 hosts when considering the VA strategy, for which the computation requires almost 4 hours. The RA strategy allows attack path computation of up to 300 hosts with a computation time of almost 3 hours. In contrast, the HA strategy scales up to 700 hosts for whom less than 1 hour is needed to compute attack paths.

Result 6: *Aggregation strategies enable attack path computation for network sizes for which the problem would be impractical. In particular, the HA strategy outperforms all the other strategies, being linear to the network size.*

Given the impossibility of computing the attack paths for SoA, we evaluate the information loss of the aggregation strategies by analyzing the reachability accuracy for all the networks from 10 to 700 hosts. More in detail, given each pair of $\langle p_s, p_t \rangle$ where p_s is an attack source and p_t an attack target in SoA attack graph, let an *s-t path* be an attack path from p_s to p_t. Then, for the AG obtained by applying aggregation strategies, we evaluate the following elements:

- True Positives (TP) are the s-t paths in SoA and the aggregated AG.
- False Positives (FP) are the s-t paths not in SoA but in the aggregated AG.
- False Negatives (FN) are the s-t paths in SoA but not in the aggregated AG.
- True Negatives (TN) are the s-t paths neither in SoA nor aggregated AG.

Fig. 5 reports the confusion matrices of the reachable paths according to the different aggregation strategies. It underlines that the VA strategy has an accuracy of 1, as

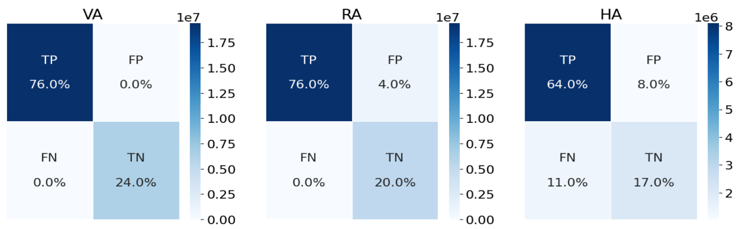


Fig. 5: Confusion matrices for each aggregation strategy.

expected. Indeed, it only aggregates when all the vulnerability features are the same, thus avoiding the generation of meta-vulnerabilities that do not exist in SoA. In contrast, RA and HA strategies have 0.96 and 0.81 accuracy, respectively, due to the presence of meta-vulnerabilities that may alter the original AG topology in which some paths may be wrongly present (at most 8% of the total number of paths) and others missing (11% of the paths only for HA strategy). Interestingly, the RA strategy does not contain false negatives; thus, it is conservative because only additional non-existing paths may cause lower accuracy.

Result 7: *Aggregation strategies may alter the AG topology with meta-vulnerabilities that replace the actual ones. However, the VA strategy does not present any topology variation, while the RA and HA ones miss* 4% *and* 19% *of attack paths.*

This experimental evaluation analyzed the accuracy-scalability trade-off of the proposed pipeline. It showed that false positive vulnerabilities may project lower accuracy in the risk analysis. In contrast, aggregating the vulnerabilities improves the AG scalability, allowing path computation in networks whose size is impractical with existing approaches. On the other hand, a slight degradation of accuracy in the risk evaluation is the price to pay for the improved scalability: it is mitigated by tolerating from 10% to 20% of uncertainty in the risk calculation.

6 Conclusion

This paper presented a novel computational pipeline for the Monitor phase of MAPE-K self-protecting systems based on attack graph models. It comprises two components to (i) filter out false positives from inaccurate vulnerability scanners and (ii) reduce the size of vulnerability inventories to improve the scalability of attack graph computation algorithms. We validated the two components with an experimental evaluation based on a real network. This showed the pipeline's capability to (i) improve risk estimation accuracy and (ii) improve attack graph scalability with traded risk estimation approximation. Finally, we showed how the proposed aggregation strategies enable the computation of attack paths in networks whose size would be intractable with existing solutions. In future work, we are looking to investigate the pipeline's applicability to different risk models in other security domains. Currently, we leverage state-of-the-art risk models, but specific domains, such as automotive and cyber-physical systems, may benefit from different models.

Acknowledgements. This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

1. Alazmi, S., De Leon, D.C.: A systematic literature review on the characteristics and effectiveness of web application vulnerability scanners. IEEE Access **10** (2022)
2. Arcaini, P., Riccobene, E., Scandurra, P.: Modeling and analyzing mape-k feedback loops for self-adaptation. In: SEAMS, Firenze, Italy. pp. 13–23. IEEE/ACM (2015)
3. Bonomi, S., Cuoci, M., Lenti, S.: A semi-automatic approach for enhancing the quality of automatically generated inventories. In: CSR, Venice, Italy. pp. 307–314. IEEE (2023)
4. English, C., Terzis, S., Nixon, P.: Towards self-protecting ubiquitous systems: monitoring trust-based interactions. Personal and Ubiquitous Computing **10**, 50–54 (2006)
5. Girdler, T., Vassilakis, V.G.: Implementing an intrusion detection and prevention system using Software-Defined Networking: Defending against ARP spoofing attacks and Blacklisted MAC Addresses. Computers & Electrical Engineering **90**, 106990 (2021)
6. Gonzalez-Granadillo, G., Dubus, S., Motzek, A., Garcia-Alfaro, J., Alvarez, E., Merialdo, M., Papillon, S., Debar, H.: Dynamic risk management response system to handle cyber threats. Future Generation Computer Systems **83**, 535–552 (2018)
7. Handoko, S., Warnars, H.L.H.S.: Network scanning topology based on inventory using query snmp method. In: Smart Data Intelligence: Proceedings of ICSMDI 2022, pp. 295–306. Springer (2022)
8. Holm, H., Sommestad, T., Almroth, J., Persson, M.: A quantitative evaluation of vulnerability scanning. Information Management & Computer Security **19**(4) (2011)
9. Hu, H., Liu, J., Zhang, Y., Liu, Y., Xu, X., Tan, J.: Attack scenario reconstruction approach using attack graph and alert data mining. Journal of Information Security and Applications **54**, 102522 (2020)
10. Ingols, K., Lippmann, R., Piwowarski, K.: Practical Attack Graph Generation for Network Defense. In: ACSAC. pp. 121–130 (Dec 2006)
11. Jajodia, S., Noel, S., O'berry, B.: Topological analysis of network attack vulnerability. Managing Cyber Threats: Issues, Approaches, and Challenges pp. 247–266 (2005)
12. Kaynar, K.: A taxonomy for attack graph generation and usage in network security. Journal of Information Security and Applications **29**, 27–56 (Aug 2016)
13. Kaynar, K., Sivrikaya, F.: Distributed Attack Graph Generation. IEEE Transactions on Dependable and Secure Computing **13**(5), 519–532 (Sep 2016)
14. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1) (2003)
15. Khakpour, N., Skandylas, C., Nariman, G.S., Weyns, D.: Towards Secure Architecture-Based Adaptations. In: SEAMS. pp. 114–125 (2019)

16. Kholidy, H.A.: Autonomous mitigation of cyber risks in the cyber–physical systems. Future Generation Computer Systems **115**, 171–187 (2021)
17. Kocher, P., Jaffe, J., Jun, B., Laren, C., Lawson, N.: Self-protecting digital content. CRI Content Security Research Initiative, Tech. Rep (2003)
18. Lee, H., Mudgerikar, A., Kundu, A., Li, N., Bertino, E.: An infection-identifying and self-evolving system for iot early defense from multi-step attacks. In: Computer Security – ESORICS 2022. pp. 549–568. Springer Nature Switzerland, Cham (2022)
19. Li, M., Hawrylak, P.J., Hale, J.: Implementing an Attack Graph Generator in CUDA. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 730–738 (May 2020)
20. Li, N., Zhang, M., Li, J., Adepu, S., Kang, E., Jin, Z.: A Game-Theoretical Self-Adaptation Framework for Securing Software-Intensive Systems. ACM Transactions on Autonomous and Adaptive Systems p. 3652949 (2024)
21. Li, T., Jiang, Y., Lin, C., Obaidat, M., Shen, Y., Ma, J.: DeepAG: Attack Graph Construction and Threats Prediction with Bi-directional Deep Learning. IEEE Transactions on Dependable and Secure Computing pp. 1–1 (2022)
22. Liang, G., Weller, S.R., Luo, F., Zhao, J., Dong, Z.Y.: Distributed blockchain-based data protection framework for modern power systems against cyber attacks. IEEE Transactions on Smart Grid **10**(3), 3162–3173 (2019)
23. Makino, Y., Klyuev, V.: Evaluation of web vulnerability scanners. In: IDAACS. vol. 1, pp. 399–402. IEEE (2015)
24. Moskal, S., Yang, S.J., Kuhl, M.E.: Extracting and evaluating similar and unique cyber attack strategies from intrusion alerts. In: ISI. pp. 49–54. IEEE (2018)
25. Nadeem, A., Verwer, S., Moskal, S., Yang, S.J.: Alert-Driven Attack Graph Generation Using S-PDFA. TDSC **19**(2), 731–746 (Mar 2022)
26. Palma, A., Bonomi, S.: A workflow for distributed and resilient attack graph generation. In: DSN-S, Porto, Portugal. pp. 185–187. IEEE (2023)
27. Pekaric, I., Groner, R., Witte, T., Adigun, J.G., Raschke, A., Felderer, M., Tichy, M.: A systematic review on security and safety of self-adaptive systems. Journal of Systems and Software **203**, 111716 (2023)
28. Presekal, A., Stefanov, A., Rajkumar, V.S., Palensky, P.: Attack graph model for cyber-physical power systems using hybrid deep learning. IEEE Transaction on Smart Grid **14**(5), 4007–4020 (2023)
29. Sabur, A., Chowdhary, A., Huang, D., Alshamrani, A.: Toward scalable graph-based security analysis for cloud networks. Computer Networks **206**, 108795 (Apr 2022)
30. Skandylas, C., Khakpour, N.: Design and implementation of self-protecting systems: A formal approach. Future Generation Computer Systems **115**, 421–437 (2021)
31. Strunk, J.D., Goodson, G.R., Scheinholtz, M.L., Soules, C.A., Ganger, G.R.: Self-securing storage: Protecting data in compromised systems. In: OSDI. pp. 165–180 (2000)
32. Sun, W., Li, Q., Wang, P., Hou, J.: Heuristic Network Security Risk Assessment Based on Attack Graph. In: Cloud Computing. pp. 181–194. Springer (2022)
33. Tayouri, D., Baum, N., Shabtai, A., Puzis, R.: A survey of mulval extensions and their attack scenarios coverage. IEEE Access (2023)
34. Yuan, E., Malek, S., Schmerl, B., Garlan, D., Gennari, J.: Architecture-based self-protecting software systems. p. 33–42. QoSA '13, ACM (2013)
35. Zeller, S., Khakpour, N., Weyns, D., Deogun, D.: Self-protection against business logic vulnerabilities. In: SEAMS. pp. 174–180 (2020)
36. Zeng, J., Wu, S., Chen, Y., Zeng, R., Wu, C.: Survey of attack graph analysis methods from the perspective of data and knowledge processing. Security and Communication Networks **2019**, 1–16 (2019)
37. Zenitani, K.: Attack graph analysis: an explanatory guide. Computers & Security (2022)