IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 2025, date of current version Oct 15, 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3622497

Towards Automatic Detection and Mitigation of High-Risk Cybersecurity Vulnerabilities at Networked Systems

JOÃO POLÓNIO[1], JOSÉ MOURA[1,2], RUI NETO MARINHEIRO[1,2]
[1]Instituto Universitário de Lisboa (ISCTE–IUL), 1649-026 Lisbon, Portugal
[2]Instituto de Telecomunições (IT), 1049-001 Lisbon, Portugal

Corresponding authors: jose.moura@iscte-iul.pt, rui.marinheiro@iscte-iul.pt

This work was supported by FCT - Fundação para a Ciência e Tecnologia, I.P. by project reference 2024.07624.IACDC and DOI identifier https://doi.org/10.54499/2024.07624.IACDC. This work was also partially supported by FCT/MECI under UID/50008: Instituto de Telecomunicações.

ABSTRACT The current manuscript investigates a comprehensive security framework designed to proactively detect, classify, prioritize, and mitigate high-risk cybersecurity vulnerabilities in networked systems controlled by software-defined networking (SDN). While available literature explores various approaches, it lacks solutions that aggregate in a logically centralized and automated ways the previous referred capabilities. Orchestrating efficiently all these capabilities is crucial to continuously ensure the reliable operation of high-complexity networked systems. This article integrates in a novel way SDN with the Security Orchestration, Automation, and Response (SOAR) paradigm to automatically identify and address security vulnerabilities in network devices before they can be exploited. The proposed open-source framework leverages standardized risk indicators to rank discovered vulnerabilities and apply the most suitable mitigation strategies to mitigate the vulnerabilities with the highest risk of being explored against the system normal operation. The paper framework enhances the reactive security capabilities offered by legacy network devices such as Firewalls and Intrusion Detection Systems (IDSs). The paper details the design, implementation, and evaluation of the framework, validated through both emulation and hardware-based tests. The results confirm that the solution is effective in identifying and mitigating vulnerabilities across diverse devices. Analyzing the results obtained from scalability tests, as the number of scanned devices exceeds a certain threshold, CPU usage increases significantly, while memory and communication resources remain underutilized. In addition, after identifying high-risk device vulnerabilities, the framework automatically applies mitigation measures, timely protecting the system normal operation. Future work may improve the capabilities of the framework by using artificial intelligence for more efficient device vulnerability discovery, context-aware security risk evaluation, and better-aligned mitigation actions targeting identified high-risk security vulnerabilities.

INDEX TERMS System Vulnerability, Detection, Risk, Mitigation, Software Defined Networks, Automation, Network Security.

I. INTRODUCTION

COMPUTER networks are growing more complex due to the proliferation of data, advanced applications, and the Internet of Things (IoT), all of which have significantly increased both the number and heterogeneity of connected devices. Managing a considerable number of network nodes with distinct functional characteristics is very difficult and prone to mistakes. Many of these nodes often exhibit security vulnerabilities, making them appealing targets for cyberattackers. Typically, attackers have a persistent economic advantage over defenders. While defenders must secure every potential vulnerability, an attacker might need to exploit only a single weakness to achieve their goal. This economic disparity, coupled with the growing difficulty for human defenders to maintain system security amidst constantly evolving cyber threats targeting complex, high-connectivity networked systems, highlights the urgent need for further research into novel solutions. In this context, extensive research into scalable, automated, and proactive approaches is essential for effectively detecting and mitigating system vulnerabilities, thereby

reducing the risk of disruptions to system normal operation, caused by cyberattacks or natural events, which exploit or expose the system security weaknesses [1].

Given the last referred open security issues, we investigate the very common scenario of an organization network infrastructure that is visualized in Fig. 1. The institution aims to keep its networked system as secure as possible from cyberattacks. In this scenario, new devices may be integrated, emerging technologies adopted [2], and existing devices updated. These ongoing system changes can unintentionally introduce new security vulnerabilities, increasing the risk of external threats to undermine the normal operation of the institutional infrastructure. In our opinion, the typical legacy security defense line formed by rigid policy-based devices such as firewalls and intrusion detection systems are not sufficient to keep always secure the organization's network infrastructure and they should be complemented by proactive security solutions running at the internal part of the network infrastructure. These proactive solutions should allow the anticipation of possible occurrences of new security problems, such as vulnerabilities in the network devices. In addition, running the proactive solution at the internal part of the network infrastructure, it allows the first line defense, formed by firewalls and IDSs, to protect the normal operation of that proactive solution.

As already mentioned, the network devices vulnerabilities can expose critical systems to system threats, making crucial the proactive management of those vulnerabilities to guarantee the system normal operation. The primary objective of this research is to correctly coordinate the automated discovery of high-risk security vulnerabilities and the next mitigation of those vulnerabilities. To accomplish this first goal, a resilient and comprehensive architecture was developed, integrating a considerable range of open-source security technologies. Special attention was placed on the seamless orchestration of these tools, thereby facilitating automatic and coordinated responses to discovered security vulnerabilities. This novel architecture relies on the additional security defense provided by legacy security defense solutions such as firewalls and intrusion detection systems (see Fig. 1). In addition, this research has the secondary objective of rigorously evaluating the impact of the new proposal on the network and device performance, ensuring that the detection and mitigation of security vulnerabilities are executed correctly and in a timely manner. The paper main contributions are as follows:

1) The design and implementation of a novel security architecture that extends the capabilities of existing network devices by integrating a proactive vulnerability detection and mitigation layer. This framework enhances system resilience beyond what is offered by traditional security tools, such as, firewalls and IDSs.
2) The development of a customized Security Orchestration, Automation, and Response (SOAR) logic, including a tailored playbook and specifically designed interaction flows. These orchestrated processes enable seamless automation across different components of the SDN-based system, allowing for efficient, scalable, and intelligent responses to detected vulnerabilities.
3) A discussion of the unique contributions of the proposed framework and orchestration logic as a cohesive and novel system, rather than a mere integration of existing tools. The SOAR playbook and interaction flows are central to this innovation, enabling adaptive and context-aware responses that go beyond current approaches in the literature.
4) It highlights that future SOAR-based proposals should consider limitations like restrictive APIs, limited control during automated execution, inefficient debugging, and unclear documentation. It also offers a novel practical example of overcoming these challenges by combining modular and reusable components with agile and flexible strategies to ensure robust, scalable, and adaptable security automation in modern networked environments.
5) It uses a dynamic and virtualized testbed, incorporating Hardware-in-the-Loop (HIL) capabilities, to evaluate the system's performance. The testbed is used to assess system burden and measure the latency from vulnerability detection to host isolation within the network.

The rest of the paper is structured as follows. Section II discusses related literature. Section III presents the design of solution's architecture. Section IV details the implementation of the proposed system. Section V discusses the results obtained from the evaluation of the proposal. Finally, Section VI concludes the paper and establishes upcoming research.

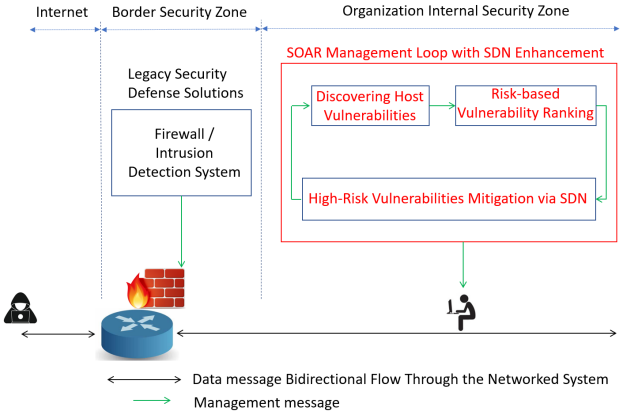

FIGURE 1. Proposed SOAR–SDN management loop, with SDN enhancing SOAR's response phase for host high-risk vulnerability mitigation.

II. LITERATURE REVIEW

A very recent work [1] has comprehensively analyzed the literature for the discovery and mitigation of security vulnerabilities in networking infrastructures controlled by SDN, which are strongly related to the current paper. Most of this revised work supports system security in a reactive way. This means the majority of previous contributions try to successfully detect and mitigate running attacks, which does not guarantee a robust system protection. Exceptionally, some

work have proposed mitigation techniques such as Moving Target Defense (MTD) [3]–[5] or Honeynet [6], which can be seen as proposals toward the proactive defense of networking environments using active cyberdeception techniques against potential attackers. These proposals aim to reduce the likelihood of an attacker successfully exploiting network vulnerabilities. However, there is always a possibility for an attacker may still manage to exploit one of the existing vulnerabilities and carry out a disruptive cyberattack, such as Deny of Service (DoS) or data exfiltration. To effectively prevent these attacks, new security management approaches are needed, such as the one proposed in this paper, which can automatically detect and eliminate high-risk vulnerabilities from network devices.

Regarding the analysis of discovered security vulnerabilities, including those associated with hosts, only a limited number of literature contributions performed security vulnerability assessments using standardized metrics [1], calling for further contributions using Common Vulnerabilities and Exposures (CVE) to identify the vulnerabilities and Common Vulnerability Scoring System (CVSS) to classify their risk. The use of standardized metrics is fundamental for companies seeking to improve their security defenses and actively prioritize the mitigation of high-risk vulnerabilities, before these could be explored. In addition, the adoption of standardized security metrics enhances strategic decision-making by providing deeper insights into emerging threats. It also enables a more efficient dissemination of data associated to security vulnerabilities among the security players of distinct organizations. Another area that remains underexplored in the literature is the integration of active probing tools within SDN environments, relevant for organizations to promptly detect vulnerabilities by actively examining their systems and, as an example, discover open application ports that could be used for non authorized system accesses. After the discovery of these open ports, these should be closed or protected by convenient mitigation techniques. Thus, active probing techniques support the prioritization of mitigation efforts by enabling organizations to swiftly and effectively address the most critical security vulnerabilities within their systems. The active scanning prevents security incidents, because it enables a fast deployment of proactive measures against imminent vulnerabilities exploitation by external attackers.

Further investigation using SOAR is needed for enhancing the automation and orchestration aspects on SDN-based solutions protecting the system security. In fact, the adoption of SOAR has several important benefits. One of the most important is the ability to optimize and automate incident response workflows. With SOAR, incident responses can be executed through automated playbooks and workflows, integrating and orchestrating various tools such as vulnerability scanners, analysis of discovered vulnerabilities, risk classification of vulnerabilities, selection of vulnerabilities to mitigate, and mitigation of high-risk vulnerabilities. The automated and orchestrated response provided by SOAR to timely eliminate high-risk security vulnerabilities in large-scale and complex networks [7] can be further enhanced by the SOAR's flexibility in coordinating artificial intelligence agents using various data learning models [8]–[10].

Another major concern that must be addressed is the security risk posed by IoT devices, which are frequently deployed with insufficient protection, rendering them highly susceptible to cyberattacks. One proposed framework enhances IoT security by leveraging an SDN-based architecture to automatically scan devices for known vulnerabilities before they are granted access to the network [11]. Upon detecting a vulnerability, the system attempts automated remediation; if unsuccessful, it notifies the user and provides recommended mitigation actions. In experimental evaluations, the framework effectively identified and neutralized vulnerabilities—for instance, by isolating compromised devices using firewall rules.

The same evaluation also showed that incorporating additional checks—such as verifying whether a host has already been scanned—introduces only minimal performance overhead. Specifically, the average packet transmission delay increased by just 5.05 ms, and bandwidth usage rose by approximately 0.45% compared to baseline implementations [11]. However, these values reflect internal orchestration overhead rather than the time required to perform actual vulnerability scans. For instance, scanning for weak or default passwords using a custom scanner may take an average latency of 2.12 minutes, while detecting vulnerabilities such as Badlock using tools like Nessus can require up to 7.25 minutes.

An illustrative example of applying automated vulnerability management in real-world environments is the Vulnerability Assessment as a Service (VAaaS) system proposed in [12]. Designed specifically for complex ICT infrastructures—particularly in healthcare, the VAaaS framework addresses the challenges of securing heterogeneous devices distributed across cloud, fog, and extreme edge layers. Its architecture leverage SDN for real-time network monitoring and employs the OpenVAS scanner to assess both newly connected and existing devices. Devices are evaluated against CVSS metrics and subsequently assigned to appropriate VLANs based on their risk profiles. Reported latency results indicate that the scanning duration for generic devices ranged from 13 seconds to 15 minutes, with an average of 441 seconds. In contrast, specialized healthcare devices required significantly more time, averaging 38 minutes and 11 seconds due to their operational complexity.

These examples highlight that while orchestration overhead can be minimal, the overall scan time, impacting latency, remains a critical factor in system responsiveness and scalability. In practice, scanning latency is influenced by multiple parameters, including the number of vulnerability tests performed, the complexity of the detection logic, the computational resources available on the scanner host, and the target device's operating system and service configuration. Additionally, network conditions and scan configurations (e.g., full vs. fast scans) also play a role. Therefore,

the latency should be interpreted in light of these influencing factors, which can vary significantly across different tools and deployment environments.

Complementing orchestration-focused approaches, some research has explored the integration of machine learning for proactive threat detection. For example, the framework introduced in [13] combines vulnerability assessment with a machine learning-based Intrusion Detection System (ML-IDS), enabling real-time monitoring of network flows and predictive detection of attack patterns. Hosts are also segmented into network slices based on their `CVSS v3.0` risk scores, allowing for differentiated handling according to assessed severity.

As summarized in Table 1, most related works address vulnerability detection or mitigation in isolation, often focusing on specific techniques such as MTD, honeynets, or VLAN segregation. However, they generally lack a cohesive orchestration framework that integrates discovery, risk-based prioritization, and mitigation in a unified workflow. Furthermore, to the best of our knowledge, no prior SOAR–SDN proposals provide publicly available implementations that allow for reproducibility or direct benchmarking of orchestration complexity and response times. These gaps highlight the novelty of our contribution, which combines standardized risk scoring (CVSS with QoD) and SDN-based mitigation into an open-source SOAR playbook, enabling both replication and future comparative studies.

III. SYSTEM ARCHITECTURE

This section discusses the system architecture and design concepts, with the objective of addressing the challenges mentioned in Section II. Section III-A discusses the methodology and principles that guided the system's development. Section III-B details each system component, explaining their functions, relationships, and contributions to the overall system operation.

A. METHODOLOGY

The system architecture building blocks and system workflow are visualized in Fig. 2. It begins with the detection phase, when the proposed SOAR-based solution discovers hosts (see 1.2.1 in Fig. 2), using network tools. Then, the SOAR invokes a scanner to inspect host vulnerabilities and classify them via CVE (see 2.2.1). Once the vulnerabilities have been classified, the SOAR framework moves on to the analysis phase, where the severity and risk associated with each vulnerability are assessed by means of CVSS, sorted out, and compared against a decision threshold. This step identifies the vulnerabilities with the higher security risk, which need to be mitigated (see 3.2.1). Based on the last results, the SOAR triggers the mitigation phase of high-risk vulnerabilities (see 4.1), where appropriate mitigation measures are implemented, such as isolating vulnerable devices through VLAN switching or blocking malicious traffic towards those devices. The final solution outcome is to ensure the high-risk vulnerabilities in hosts are successfully removed (see 4.2.1).

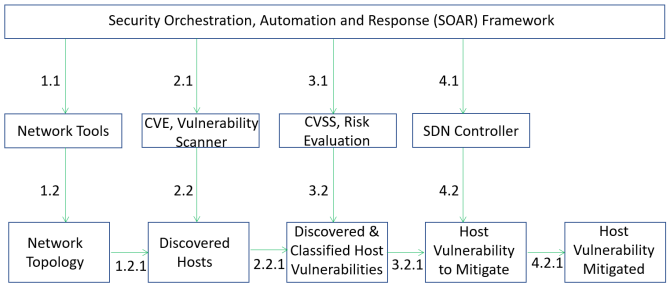

FIGURE 2. Proposal Functional Blocks Orchestrated by SOAR Framework.

Based on the proposed solution methodology, a clear sequence of management steps is required to ensure that each vulnerable device on the network is systematically discovered, classified, analyzed and its associated security risk addressed in a prioritized manner. The correct execution order of all management phases must be orchestrated by a new SOAR playbook. Therefore, in order to develop the proposed system, the following stages were identified:

- Host discovery.
- Detection of host vulnerabilities.
- Generation of a vulnerability report for each host.
- Parse the host vulnerability report and analyze the extracted vulnerability information.
- Classify the risk associated to each host vulnerability.
- Identify the topmost hosts with high-risk vulnerabilities against the system normal operation.
- Mitigate the high-risk vulnerabilities.

Each phase of the system development lifecycle plays a vital role in ensuring robust network security. Device discovery is fundamental, as it provides a comprehensive view of all active components within the network. Identifying vulnerabilities in these devices is crucial for uncovering security flaws before they can be exploited by malicious actors. Generating a detailed vulnerability report ensures consistent and accurate communication of security issues, thereby supporting informed decision-making regarding mitigation strategies. Parsing and analyzing the report is essential for assessing the severity of each vulnerability, enabling the prioritization of responses to the most critical threats. Finally, implementing appropriate mitigation measures is key to effectively reducing the risk of exploitation and strengthening the network's resilience against future attacks.

The effective execution of these development phases followed a set of fundamental design principles outlined in [14]:

- Interoperability.
- Proactivity.
- Adaptability.

Interoperability ensures that the system can seamlessly integrate with diverse tools, platforms, and technologies, thereby enhancing its compatibility and overall operational efficiency. Proactivity empowers the system to anticipate and address potential threats and vulnerabilities before they can

TABLE 1. Related Work Comparison

Paper	Vulnerability Detect	SDN Controller	SOAR	Risk Evaluation	Vulnerability Scan	Mitigation Measure	Resources	Latency
[3]	○	N/A	○	CVSS+Exploitation	○	MTD+Decoy nodes	○	○
[4]	○	ONOS	○	CVSS+Exploitation	○	MTD	●	●
[5]	●	OpenDaylight	○	CVSS+IDS	Nessus	MTD	○	○
[6]	○	POX	○	○	○	Honeynet	●	●
[7]	○	○	○	CVSS+Risk Graph	○	○	○	○
[9]	○	○	●	○	○	VLANs+Honeypots	○	○
[11]	●	POX	○	Custom	Nessus	Firewall	◑	●
[12]	●	ONOS	○	CVSS	OpenVAS	VLANs	○	●
[13]	●	OpenDaylight	○	CVSS	○	Network Slices	○	○
Ours	●	Ryu	●	CVSS	GVM	VLANs	●	●

Legend: ●: topic is covered; ◑: topic partially covered; ○: uncovered topic; N/A: unspecified SDN controller.

be exploited, significantly strengthening network security. Lastly, adaptability preserves the system's flexibility and responsiveness to change, enabling it to evolve in alignment with emerging requirements, technological advancements, and newly identified security challenges.

B. DESIGN

The architecture of the proposed proactive detection and mitigation solution is visualized in Fig. 3. The SOAR platform orchestrates host discovery, vulnerability scanning, risk evaluation, and mitigation through SDN, ensuring an automated and prioritized response to high-risk vulnerabilities. This solution offers a hierarchical design and sequential functionality among architecture components identified by numbered interactions, which follow the discussion made in Section III-A (see Fig. 2). The architecture consists of two primary elements: the SOAR Server and the Security Tools Server. The SOAR Server hosts the SOAR Platform, while the Security Tools Server houses essential tools, including the Device Discovery Module and the Vulnerability Scanner. The separation between the SOAR Server and the Security Tools Server enhances scalability, and flexibility in updates and maintenance on each server, allowing independent changes without negative repercussions on the other server performance. Moreover, it promotes effective task segregation, which is an essential practice to ensure that the centralized control of the SOAR Server remains unaffected by the operational demands of individual tools, thereby reinforcing the system's structural integrity. Additionally, this approach enables the deployment of Security Tools Servers in locations beyond the direct visibility or reach of the SOAR Server, thereby extending the scope and effectiveness of SOAR Server security orchestration capabilities.

There are two additional important architecture components. The Database component, further detailed in Section IV-D, that persistently stores in-memory key–value information about relevant networked system status, namely discovered network devices and found security vulnerabilities. The SDN controller offers an NorthBound API that receives SOAR requests to initiate vulnerability mitigation actions.

As already mentioned, the SOAR platform orchestrates the diverse modules of Fig. 2 such as network tools, vulnerability scanner, risk evaluation, and SDN controller, ensuring that detected vulnerabilities are logged and promptly addressed. This SOAR-based solution enables the fast automatic response to high critical vulnerabilities in a structured and scalable manner, reducing response time significantly compared to manual intervention or non-orchestrated automatic solutions. In the text below, we further detail the following four features orchestrated by SOAR (see Fig. 3): i) device discovery; ii) vulnerability scanner; iii) vulnerability risk evaluation; and iv) vulnerability mitigation.

The SOAR framework, using the interaction numbered as "1.0" of Fig. 3, initiates the sequence of automatic steps to perform the device discovery feature via the security service adapter (see 1.1 of Fig. 3), which in its turn invokes (see 1.1.1) the host discovery function (see 1.2). Each returned result from the discovery function is permanently stored on the database (see 1.2.1) for posterior consultation by other system components interested on that information.

The trigger 2.0 in the SOAR framework starts the sequence of steps to perform the vulnerability scanner feature via the security service adapter (see 2.1), which by its turn invokes (see 2.1.1) the vulnerability scanner function of Security Tools Server over each network host (see 2.2). Each returned result from the vulnerability scanner function is stored on the database (see 2.2.1).

The SOAR framework can initiate (see 3.0) a sequence of automated steps to execute the risk evaluation feature. This process begins by collecting vulnerability data from the database (see 3.1), evaluating the risk associated with each vulnerability, identifying the highest-risk vulnerabilities on each host (see 3.2), and storing the results in the database (see 3.2.1). The severity of the identified vulnerabilities enables the subsequent SOAR capability to make well-informed mitigation decisions based on a risk-prioritized list of host security vulnerabilities.

Through interaction "4.0", the SOAR framework performs the mitigation feature by retrieving high-risk vulnerable devices from the database and triggering (see 4.1) the SDN controller to initiate a sequence of management actions aimed at mitigating the risk associated to each host vulnerability (see 4.2). The results of these mitigation actions are stored in the database (see 4.2.1). Here, the SDN controller plays a central

role. As an example of a possible mitigation action, the SOAR component can instruct the SDN controller to temporarily move a vulnerable host to a separate VLAN. This isolation remains in place until corrective actions are performed on the host to eliminate its vulnerabilities.

IV. IMPLEMENTATION

This section details the process of implementing the proposed system, describing the major elements and their interactions. Section IV-A presents the logic of the system, explaining the server where each component is running and the interaction among the various components of the current proposal, through a system activity diagram. The following sections provide a detailed explanation of each tool or functional module used in the implementation of the proposed solution. The proposal code is publicly available in [15].

A. DATA FLOW AND INTERACTION

The Fig. 4 depicts a comprehensive activity diagram that illustrates the end-to-end process from device discovery, vulnerability scanning, and risk evaluation to the mitigation of high-risk vulnerabilities via SDN. This sequential functional process was deployed as debated in the following text.

The process begins in the SOAR by creating a network scanning ticket that requests a device discovery (1), thus initiating a device discovery task (2) from the SSA. The system then checks for any active devices on the network (3) after sending the request. If no devices are detected, the process waits for a timeout period (4) before returning to the initial device discovery request, restarting the loop. After detection, the status of the device is checked in the database (5). This status refers to the date of the last vulnerability scan carried out on the device, and, based on this parameter, it is decided whether or not the device should be scanned (6). As the SOAR decides the device should be scanned, it then generates a vulnerability scanning ticket for that device (7) and starts the vulnerability scan (8). The scan results after being stored in the database are analyzed (9) to calculate a severity score (10). Based on the calculated score, the SOAR initiates a VLAN change (i.e. Mitigation Measure 1) to relocate the device to a quarantine zone via the SDN controller (11). This mitigation strategy is just one of several possible approaches. Other measures (i.e. Mitigation Measure 2), such as Deep Packet Inspection (DPI) to analyze network traffic for malicious content towards the vulnerable host, the use of firewall rules to block flows involving the vulnerable host, or even MTD to dynamically shift the network configuration and protect the vulnerable host, could also be implemented in the proposed solution. These mitigation alternatives to VLAN isolation can be studied in future work. If the device is considered safe, the process is finished and a new scanning is scheduled in the future (12).

The next Section details how the SOAR was deployed.

B. SOAR PLATFORM

Catalyst [16] is an open-source SOAR platform that automatically alerts about security incidents and deals with them. The platform is adaptable to various processes and workflows, allowing customization through ticket types, conditional custom fields, and playbooks to meet specific requisites. Catalyst enables firms to enhance their operations. It also allows the management of security alerts with high efficiency.

Catalyst version 0.10.3 was chosen because of its support for creating customized automation scripts written in Python. This version has the capability of creating detailed playbooks. It allows the remote execution of processes via an API.

Playbooks are a particularly valuable feature of Catalyst, offering the ability to define automation workflows from scratch using the YAML language, with graphical representation available through the user interface (UI). While the concept of a visual UI is appealing, its current implementation requires further refinement, as constructing complex playbooks remains cumbersome. Although YAML provides considerable flexibility, it lacks support for essential programming constructs such as loops and conditional branching, which are critical for building advanced automation logic. A more intuitive and expressive language could enhance usability and facilitate the creation of sophisticated workflows, including the ability to revisit previous automation steps. As playbook complexity increases, tracking modifications and identifying errors in YAML code become progressively more difficult in the absence of an integrated debugging environment. Furthermore, the documentation available for the Catalyst version used in this study was notably limited, particularly regarding playbook development. This inadequacy compelled developers to rely on example-driven learning and trial-and-error experimentation to understand and utilize the platform's capabilities effectively.

Develop Catalyst automation scripts offers numerous advantages, particularly programming in Python, which provides a wide range of capabilities. In Catalyst, automation scripts are executed within Docker containers, where each script runs in its own container. This approach improves compatibility and ensures process isolation. However, executing each script in a distinct container can create some issues. Specifically, a new container must be instantiated every time a script is run, which increases overhead, latency, and resource consumption.

Catalyst has proven to be a tool with a highly interesting and promising concept. As previously noted, its ability to enhance automation while preserving a traditional ticketing interface for opening and closing tasks demonstrates significant innovation compared to other available open-source tools. However, the software still requires substantial development before it can be considered suitable for organizational use.

During the Catalyst implementation, several challenges and limitations were encountered, which in our opinion should be carefully addressed in future versions:

- The Catalyst API presented several limitations. For instance, it required requests to be made using a complete

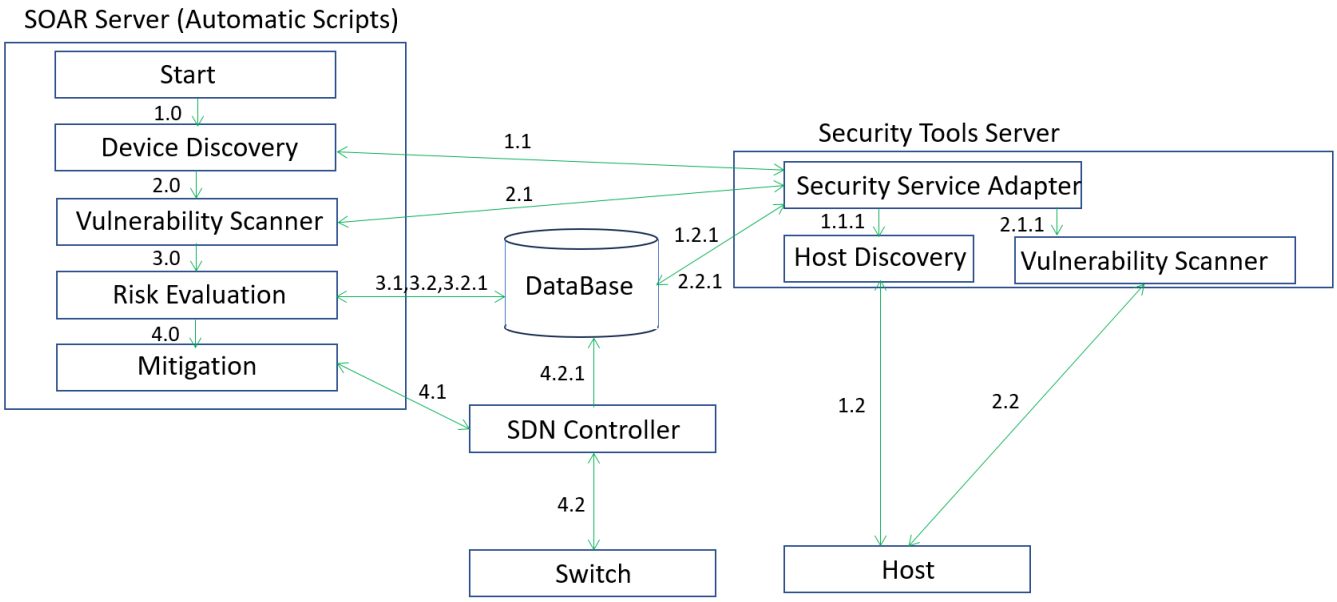

FIGURE 3. **Proposed System Architecture and Functional Components.**

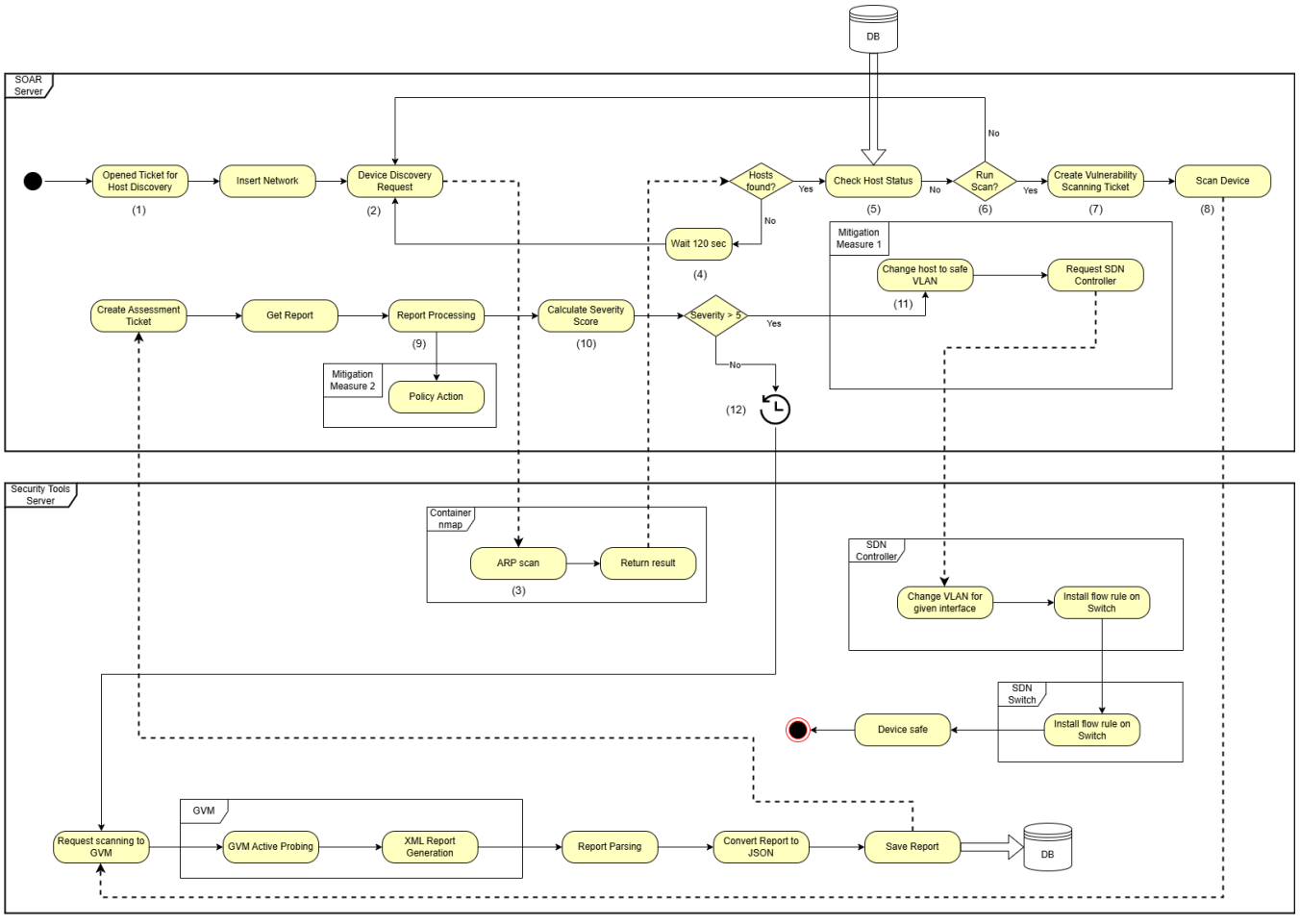

FIGURE 4. **Proposed System Workflow and Interaction Sequence.**

JSON object, with parameters hardcoded directly within it. When creating a ticket via the API, the system did not auto-generate an ID, forcing the user (i.e., the application using the API) to manually generate one. Moreover, there was no built-in function to retrieve the most recently assigned ID, making it necessary to analyze all existing tickets to determine the latest one.

- Another issue with Catalyst was the inability to manually close a task while a playbook was still running. This limitation led to the unnecessary execution of automated scripts, even when further processing was redundant or potentially harmful. This behavior not only increased system load but also posed risks, particularly on the Security Tools Server, where sockets sometimes remained open, requiring a service restart.
- Debugging in Catalyst was notably challenging, primarily because testing a single automation script at the end of a chain required executing the entire playbook. This made the development cycle inefficient and time-consuming.
- The tool's documentation was minimal and often unclear, which significantly hindered the installation, implementation, and overall usability of the system.

To address the limitations encountered with Catalyst, our system was deliberately designed to bypass or mitigate several of these challenges. Specifically, in response to the complexity of YAML-based playbook development and the absence of integrated debugging support, we implemented a set of practical workarounds. One key strategy was the modularization of playbook logic, wherein complex automation tasks were decomposed into smaller, reusable components. This modular approach enhanced maintainability and facilitated iterative testing and development, even in the absence of native debugging tools. By structuring the playbooks in this way, we reduced error-proneness and improved the overall robustness and scalability of the automation logic.

The next Section details the implementation of the Security Service Adapter.

C. SECURITY SERVICE ADAPTER

The Security Service Adapter (SSA) was completely developed in Python. The SSA behaves as an intermediary between the SOAR Server and the security tools, Greenbone Vulnerability Manager (GVM) and Nmap. Through the SSA REST API, it receives instructions from Catalyst SOAR to execute processes related to the GVM, as well as store and read data from a database. FastAPI [17] was selected as the library for the development of the SSA API. This Python library was selected for its simplicity and performance. FastAPI allows SOAR to quickly access the functionalities offered by the SSA, such as running scans and extracting reports. The choice of this technology was reinforced by its use in network automation scenarios, as emphasized in [18]. Although the focus of this research is not to have secure communications, the HTTPS protocol was adopted to improve the security of communications between the SSA and SOAR, thus ensuring the integrity and confidentiality of the data exchanged.

The decision-making logic for handling device vulnerabilities was implemented within SOAR, as it is the most suitable environment for this functionality. This implementation provides users with greater flexibility to tailor vulnerability analysis and mitigation workflows without affecting the overall system operation. Thus, SOAR assumes responsibility for the decision logic, while the SSA focuses on delivering the necessary support functions, promoting a modular and scalable architecture.

The scanning process is initiated when a request is made to scan a specific target, with an IP address provided as input. This triggers the function responsible for launching the scan. In the activity diagram (Fig. 4), this step (8) is represented by the block indicating the execution of the scan task, which continues until the scanner produces a report detailing the identified vulnerabilities.

During the scan, it is essential to monitor its progress to determine when it has completed. In the implementation, this is achieved through a function that periodically checks the scan status using the report identifier. The system queries the current scan status (e.g., "Running", "Stopped", "Done") from the GVM at regular intervals. By default, this interval is set to 2 minutes but can be adjusted via the Device Discovery Automation settings in SOAR to suit the environment's needs.

There are two possible approaches for monitoring: callback-based or polling. A callback mechanism would allow the GVM to notify the system immediately upon scan completion, but it introduces additional implementation complexity. The adopted approach based on periodic polling, it queries about the scan status at configurable intervals. Shorter intervals enable quicker detection of scan completion but increase system load, while longer intervals reduce resource usage at the cost of delayed detection.

Once the scan concludes, the system processes the results by extracting relevant data and storing it in the database for further analysis. This stage begins as soon as the monitoring function detects scan completion. The corresponding function then parses the XML report generated by GVM and converts it into JSON format for analysis by the SOAR platform. As part of this process, the SSA reads the report, extracts all pertinent information, and prepares it for SOAR to make informed decisions. This design allows for customization and alignment of the extracted data with the system's objectives, including specific characteristics of the scanned nodes.

D. DATABASE

Redis [19] was used to manage the work queues and temporary storage of data critical to the system's operation. Redis is an in-memory key-value store, categorized as a NoSQL database, which provides high-performance storage for various data structures such as strings, hashes, lists, sets, and sorted sets. Unlike traditional relational databases, Redis does not employ tables or schemas and does not enforce relational integrity constraints. Its primary advantage is rapid

data access, which makes it particularly suitable for managing queues and transient data required for system responsiveness.

In the Security Service Adapter (SSA), tasks such as device discovery and report parsing are queued in Redis to facilitate asynchronous processing. This setup enables parallel execution of multiple tasks, significantly reducing response times for new requests. Consequently, the SSA can continuously accept and enqueue new scan or operation requests without needing to wait for preceding tasks to complete.

The Redis database itself is deployed within a dedicated virtual machine, which ensures isolation, scalability, and ease of maintenance, while providing flexibility to scale or relocate the database infrastructure if needed.

Next Section discusses how the Device Discovery Module was deployed.

E. DEVICE DISCOVERY MODULE

Nmap (Network Mapper) [20] is an open-source tool used for exploring and evaluating network systems, including network inventory and monitoring the availability of devices or services and security purposes. While it is primarily designed for scanning large networks quickly, it is also effective for targeting individual devices [21]. Nmap is widely recognized for its efficiency and performance, offering fast and comprehensive network scans.

In the deployed system, Nmap (version 7.93) is executed within a Docker container, ensuring an isolated, reproducible, and platform-independent environment for device detection. The integration with the "python-nmap" library facilitates programmatic control of Nmap from Python, enhancing the system's versatility by enabling seamless automation and integration into broader data processing workflows.

Nmap was configured for device discovery using two specific scan types: *sP*, which performs a basic ping scan to find active devices without further testing such as port scanning or OS detection, and *PR*, which runs an Address Resolution Protocol (ARP) scan, which determines device activity by mapping IP addresses with their respective Media Access Control (MAC) addresses.

The next Section discusses how the Vulnerability Scanner was deployed and how the vulnerability risk was evaluated.

F. VULNERABILITY SCANNER

The purpose of the vulnerability scanner is to identify security vulnerabilities across various devices within the network. Once detected, these vulnerabilities are evaluated based on the potential risk they pose to normal network operations if exploited by cyberattacks. The used vulnerability scanning tool was GVM [22] which is the successor to OpenVAS.

GVM utilizes two scanners to assess devices on the network: the OpenVAS Scanner and the Notus Scanner. The OpenVAS Scanner is a comprehensive engine that executes individual vulnerability tests (VTs) sequentially, primarily using Nessus Attack Scripting Language (NASL) scripts against target systems. It draws on either the Greenbone Enterprise Feed or the Greenbone Community Feed to ensure up-to-date vulnerability information. While thorough, this method can be resource-intensive and slower, as each NASL-based local security check (LSC) is executed separately for every device.

In opposition, the Notus Scanner enhances performance by replacing the NASL-based LSC logic. Rather than executing individual scripts, it performs a bulk comparison of the installed software on a device against a known list of applications with vulnerabilities. This significantly reduces resource usage and scanning time, making the Notus Scanner a faster and more efficient option, particularly for LSCs.

GVM provides a variety of scan configurations tailored to different requirements and levels of detail. These configurations are designed to balance the depth of analysis with the potential impact on the normal operation of target systems. Users can choose between quick, low-impact scans or more exhaustive, potentially disruptive ones, depending on their specific needs. For the purposes of this research, the *Full and Fast* scan configuration was selected for the tests described in Section V, due to its efficiency and ability to deliver fast and reliable results without causing operational disruption to the target systems.

GVM supports exporting scan data in multiple formats, including PDF, CSV, and XML. For this research, XML was chosen due to its ability to encapsulate comprehensive scan details and facilitate seamless integration with other tools. GVM-generated XML files are extensive, containing numerous fields, some of these intended solely for internal system use. For practical analysis, it is recommended to focus on the report's critical data fields rather than processing all reported information.

G. VULNERABILITY RISK EVALUATION

The data in the report, obtained from the component in the previous section, are used to determine whether a device is considered vulnerable, as shown in Equation 1. This equation computes a vulnerability score for each finding by combining its Severity (S) with the corresponding Quality of Detection (QoD), normalized as QoD/100. The QoD parameter reflects the accuracy and confidence of the vulnerability detection provided by GVM.

GVM calculates QoD based on predefined criteria considering the reliability of the detection methods used. Specifically, vulnerabilities explicitly confirmed via reliable methods, such as direct version or patch-level checking, or clear proof of exploitation, are assigned high QoD values. Detections based on indirect evidence, such as service banners or other strong indicators, receive medium QoD values, whereas vulnerabilities inferred heuristically or through low-confidence indicators, such as generic behaviors or uncertain patterns, are given lower QoD values.

Multiplying Severity by QoD/100 thus effectively captures both the potential impact of the vulnerability and the accuracy of its detection. Therefore, QoD represents a structured measure of the reliability and accuracy of vulnerability iden-

tification, providing essential context and confidence levels for users making risk-based decisions.

For each device, the highest of these vulnerabilities scores is denoted as V. If V exceeds a predefined threshold, the device is classified as vulnerable. This threshold can be easily adjusted within the Mitigation Measure Automation script to accommodate varying risk assessment policies or operational requirements.

As future work we envision to replace the current high-risk decision mechanism, based on CVSS cores exceeding a predefined threshold, by more comprehensive alternatives. These include incorporating additional factors into the vulnerability risk evaluation process, such as: i) the vulnerability's exploitability value [3], [4]; and ii) the likelihood that a CVE vulnerability could be exploited, as obtained from a public API [23]; iii) the role or system's function of the affected device within the network; and iv) the number of interactions the affected device have performed with other network devices during a specific time interval. These factors are expected to significantly improve the precision of risk prioritization. Their omission in the current implementation was primarily due to the increased complexity in data acquisition, integration, and real-time processing within the orchestration pipeline. Nonetheless, their inclusion remains a key direction for future development.

$$V = \max_{i \in \{1,2,\ldots,n\}} \left(S_i \times \frac{QoD_i}{100} \right), \quad V > \text{Thresh} \qquad (1)$$

Where:

- V is the highest vulnerability score.
- S_i is the severity score of vulnerability i.
- QoD_i is the quality of detection value of vulnerability i.
- n is the total number of vulnerabilities.
- The operator $\max$ selects the highest score.
- The condition $V > Thresh$ means the device is considered vulnerable if the severity score exceeds the severity *Threshold*.

GVM supports remote task execution, such as initiating scans and retrieving reports, which enables its integration and management by the SSA. To facilitate automated communication with GVM, the `gvm-tools` toolkit was installed. Among the available tools, `gvm-script` was selected for its simplicity and efficiency. It allows direct interaction with GVM by executing Greenbone Management Protocol (GMP) commands via the command line, streamlining the automation of scanning tasks.

The gvm-script interface, which is intuitive and user-friendly, allows users to perform tasks such as initiating vulnerability scans, selecting scan configurations, and retrieving detailed reports using concise, single-line commands. This approach helps mitigate the complexity typically associated with API-based interactions.

Moreover, the use of gvm-script and its scripting-based design simplifies maintenance. By leveraging scripts, a clear separation of responsibilities was successfully implemented. This modularity facilitates easier updates and modifications, particularly when adjusting the scanning logic contained within the scripts. As a result, the integration process becomes more straightforward, while the risk of bugs and errors, which are common in other more complex API-based interactions, is significantly reduced.

The gvm-script method enhances the testing and debugging process, as each script can be independently validated, ensuring reliable and predictable behavior.

The next two sections debate the deployment of the SDN system.

H. SDN CONTROLLER

Ryu was chosen as the SDN controller [24], an open-source, modular platform developed by Nippon Telegraph and Telephone (NTT), a Japanese telecommunications company. In Japanese, "Ryu" means "flow", a name that effectively describes the controller's capability to dynamically manage network traffic flows. Written in Python and licensed under Apache 2.0, Ryu supports a wide range of network management protocols, including NETCONF, OF-Config, and the Open vSwitch Database Management Protocol. It also implements the standard SouthBound API OpenFlow, supporting versions from 1.0 to 1.5. In this implementation, OpenFlow v1.3 was used to control software-based switches via OpenvSwitch. Ryu provides a rich set of libraries for packet handling and supports various tunneling and encapsulation methods, such as VLAN.

The decision to adopt Ryu was driven by its open-source nature, extensive documentation, and Python-based implementation, which aligns well with the team's expertise. Additionally, Ryu's ability to integrate smoothly with different platforms and tools through its REST API (NorthBound interface), combined with strong community support and a flexible architecture, made it a robust choice for building a resilient SDN environment. In this setup, the Ryu controller operates within a Docker container.

I. SWITCH

The implementation has used Open vSwitch (OVS) [25], a robust, multi-layer virtual switch licensed under the open-source Apache 2.0 license. OVS is designed to support advanced network automation while maintaining compatibility with standard management protocols and interfaces. It integrates seamlessly with various virtualization platforms, such as Docker and VMware, and is particularly well-suited for virtualized environments involving multiple servers. These environments are characterized by dynamic endpoints, the need to preserve logical abstractions, and the offloading or delegation of tasks to specialized switching hardware. Version 3.1.0 of OVS was employed in this implementation.

The network topology was constructed using Mininet [26], a lightweight network emulator that creates virtual networks for testing and development purposes. Version 2.3.0 of Mininet was used. Additionally, Mininet enabled the configuration of an SDN router within one of its virtual hosts,

providing Internet connectivity to all emulated hosts. This virtualized testbed was crucial for simulating diverse network scenarios and assessing system behavior under controlled conditions before transitioning to more complex and realistic testing environments.

The next Section debates DHCP Server deployment.

J. DHCP SERVER

The DHCP service is provided through the implementation of the ISC DHCP server [27]. Version isc-dhcpd-4.4.3-P1 was selected for its proven stability and ease of use. This version is well-suited to the system's requirements, offering reliable performance and a straightforward configuration process without unnecessary complexity.

It is worth noting that, at the time of writing, ISC DHCP, while still supported, it is no longer receiving maintenance releases and it has been officially succeeded by Kea [28].

The next Section details how the Mitigation Module was built.

K. MITIGATION MODULE

In this research, the chosen mitigation strategy involved reassigning the VLAN of vulnerable devices to isolate them from the rest of the network. To achieve this, it was necessary to modify the behavior of the SDN controller by changing its source code. As illustrated in Fig. 4, when a device is identified as vulnerable, the SDN controller is responsible for updating the VLAN assignment (see step 11) on the device's interface and applying specific flow rules on the switch to enforce the new network policy. This effectively reroutes the device to a quarantine VLAN, isolating it from the main network.

Algorithm 1 is executed within the SDN controller after the SOAR platform triggers a VLAN change for a high-risk host via Ryu's REST API. This approach offers several advantages, as VLAN assignments are dynamically managed based on switch ports, allowing for straightforward and flexible updates. The algorithm relies on two dictionaries: `port_to_vlan` (line 1) and `ip_to_switch_port` (line 2). The `port_to_vlan` dictionary maintains real-time mappings of VLAN IDs to switch ports, while `ip_to_switch_port` links IP addresses to their corresponding switch IDs and source ports.

Upon receiving an IP address (line 3) and a new VLAN ID (line 4), the function first checks whether the IP exists in the `ip_to_switch_port` dictionary. If found, it retrieves the associated switch ID and port (line 7); otherwise, it returns a 404 error (line 9), indicating the IP is not recognized. Once the relevant switch and port are identified, the function updates the `port_to_vlan` mapping with the new VLAN ID (line 11). It then removes any existing flow rules for that port to avoid conflicts (line 13), ensuring the new VLAN configuration is correctly applied in the data plane. Finally, the function returns the updated `port_to_vlan` dictionary (line 16), reflecting the current VLAN assignments.

The next section presents and analyzes the results obtained through a comprehensive set of evaluation tests conducted to assess the proposed system.

Algorithm 1 Change VLAN ID

Require: Dictionaries port_to_vlan and ip_to_switch_port.

1: port_to_vlan: Tracks which VLAN each port is assigned to.
2: ip_to_switch_port: Maps an IP address to its associated port and switch.
3: ipaddress: IP address of the device whose VLAN ID needs to be changed.
4: vlanid: New VLAN ID to be assigned.
5: {Step 1: Check if IP address exists in ip_to_switch_port dictionary.}
6: **if** ipaddress exists in ip_to_switch_port **then**
7: Retrieve switch ID and source port.
8: **else**
9: **return** 404 (Not Found).
10: **end if**
11: {Step 2: Update port_to_vlan dictionary with the new VLAN ID.}
12: Update port_to_vlan for the source port with vlanid.
13: {Step 3: Clean all flow rules associated with the source port.}
14: Use the clean_flows method to clear the flow rules.
15: {Step 4: Return the updated port_to_vlan dictionary.}
16: **return** Updated port_to_vlan dictionary.

V. RESULTS

This section presents and analyzes the results obtained from testing the proposed system in two distinct environments, aiming to comprehensively evaluate its performance under varied conditions. Due to infrastructure limitations, the physical testbed lacked SDN-capable switches, rendering it unsuitable for validating the VLAN-based mitigation mechanism. In traditional switching environments, VLAN assignments are typically static and tied to physical port configurations, which restricts the dynamic and programmable VLAN reassignments necessary for responsive host isolation. Therefore, a virtualized SDN-enabled environment was employed to assess the mitigation mechanism, offering the flexibility to simulate and observe VLAN transitions in a controlled setting.

Conversely, the physical environment was instrumental in validating the vulnerability scanning process within a real-world network context, where interaction with actual hardware devices (referred to as hosts or machines) could be evaluated more realistically. Specifically, tests A to D, focused on performance analysis of the vulnerability scanner, were conducted in the physical environment, while test E, which evaluates the orchestration performance, including automated response execution, was performed in the virtual environment, where more control over the network setup was possible. Together, these complementary testbed environments

enabled a well-rounded assessment of the system's functionality, scalability, and responsiveness.

The physical environment tests were conducted in the computer networks laboratories at Iscte – University Institute of Lisbon [29]. These tests aimed to evaluate the performance of the vulnerability scanner integrated into the developed system. The laboratories provided access to 32 machines distributed across 16 workbenches in two separate rooms. By conducting tests with varying numbers of hosts, the scalability and performance of the scanner were evaluated under different load conditions. Incrementally increasing the number of target hosts allowed for an analysis of how resource consumption and scan duration scaled with the number of devices being scanned.

Each laboratory machine hosted a virtual machine (VM) designated as a scan target. Table 2 details the specifications of both the physical machines and the target VMs.

TABLE 2. Hardware and software specifications of the laboratory machines and their corresponding target virtual machines used in the physical testbed. The laboratory machines hosted Windows 10 Enterprise with virtualized Windows Server 2003 targets for vulnerability scanning.

Specification	Laboratory Machine	Target VM
OS	Windows 10 Enterprise	Windows 2003
CPU	Intel i5-6600	8 CPU Cores
RAM	32 GB	1 GB

The physical machine designated as the Security Tools Server hosts a VM running Kali Linux, where the vulnerability scanner, GVM, is installed and operational. This server is connected to the laboratory switch, providing it with access to the network. Within the VM, a DHCP server is configured to assign IP addresses to the laboratory machines. Table 3 presents the hardware specifications of the physical server and the configuration details of the server VM.

According to our current best knowledge, we have not identified any prior SOAR–SDN proposals with publicly available implementations that would enable a direct, quantitative comparison of orchestration complexity and response times. Most related works provide only conceptual architectures or partial implementations, which limits the possibility of benchmarking our proposal performance against them. As an alternative, and to promote transparency and reproducibility, we have made our full implementation publicly available on [15]. This allows the community to replicate our results and provides a reference point for future work to compare orchestration complexity and response times against our proposal.

A. SCAN TIME ANALYSIS

The objective of this test was to evaluate the resource consumption of GVM and its impact on scan duration. The time required to complete a scan directly influences how frequently scans can be performed and how quickly vulnerabilities can be detected and mitigated. Long scan durations may delay vulnerability identification, thereby increasing exposure to potential threats.

TABLE 3. Hardware and software specifications of the Security Tools Server and its associated virtual machine used for running GVM in the physical testbed. The server hosted a Kali Linux VM configured with DHCP and vulnerability scanning services.

Specification	Server Machine	Server VM
OS	Windows 11 Home	Kali Linux 2022.2
CPU Model	AMD Ryzen 5 5600H	8 CPU Cores
RAM	24 GB	12 GB

Assessing the scanner's impact on system resources, specifically CPU, RAM, and network bandwidth, ensures that scanning operations do not degrade overall system performance or lead to unexpected downtime. Bandwidth usage was also monitored to verify that the scanning process does not saturate the network, which could disrupt host communication or worse the performance of other services. The parameters CPU, RAM, and bandwidth, were selected to monitor the system performance in the tests described below. The goal was to identify any resource bottlenecks that might correlate with increased scan durations and limit the system's overall performance.

Tests were conducted in the laboratory environment using 1, 2, 4, 8, 16, and 32 hosts. Fig. 5 illustrates the relationship between the number of scanned hosts and the corresponding scan duration.

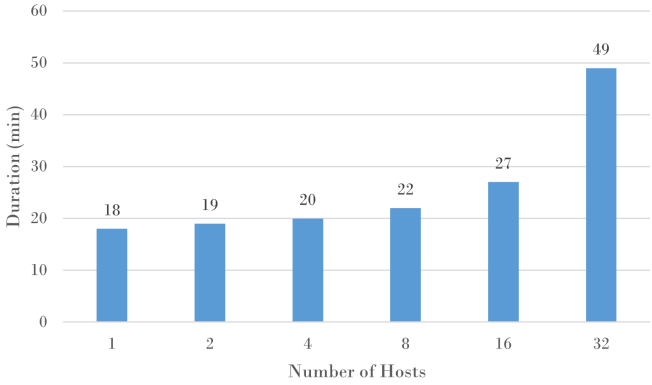

FIGURE 5. Duration of vulnerability scans with 1 to 32 hosts in the physical testbed. Tests were executed with GVM (Full and Fast configuration) running on Kali Linux, scanning Windows-based virtual machines connected via laboratory switches.

The results indicate a noticeable increase in scan duration as the number of scanned hosts grows, with a particularly sharp rise observed beyond 8 hosts. This escalation can be attributed to the complexity of the Vulnerability Tests (VTs) performed by GVM on each individual host. As the number of targets increases, the scanner must handle a proportionally

greater volume of data and processing tasks, leading to longer execution times.

Further insights into this behavior are provided in subsections V-B, V-C, and V-D, where the scanner's resource usage is examined in greater detail. These analyses help identify external limiting factors that may contribute to increased scan durations.

B. CPU LOAD

This subsection analyzes CPU usage across the different test scenarios. Table 4 presents the maximum, average, and standard deviation values of CPU utilization recorded during scans to a variable number of hosts. The standard deviation indicates the variability of CPU usage over time, providing insight into the consistency of resource consumption. All CPU usage values were calculated from a representative sample, with the CPU initially operating at 1%.

For tests involving up to 2 hosts, maximum CPU usage remained moderate, and average usage was relatively low. Although the test with 4 hosts showed a peak CPU usage nearing 100%, the increase in scan duration was minimal, only about two minutes longer than the single-host test (see Fig. 5). This can be attributed to the average CPU usage still being within acceptable limits, allowing the system to maintain efficient operation (see Table 4).

However, starting from 8 hosts, the CPU experienced full saturation, accompanied by a significant rise in average usage, indicating a notable strain on server resources. With 16 hosts, the impact on scan duration became more pronounced. The CPU not only reached its maximum capacity but also remained at high utilization levels for extended periods. In the 32-host scenario, the situation further deteriorated, with average CPU usage approaching 90%, signaling a critical overload. This sustained high usage reduced the scanner's efficiency, preventing it from effectively handling all hosts and resulting in a substantial increase in scan completion time.

An analysis of the standard deviation values presented in Table 4 reveals a notable variation in CPU usage consistency as the number of hosts increases. For tests involving 1 and 2 hosts, the standard deviation remains relatively low, indicating stable and predictable CPU usage with minimal fluctuation around the mean. This suggests that under lighter loads, resource consumption is more consistent.

However, for scenarios with 4 and 8 hosts, the standard deviation increases significantly, reflecting greater variability in CPU utilization. This implies that the system experienced intermittent peaks in resource demand. Despite the high average CPU usage, the increased variability suggests that the system was frequently operating near capacity, with only brief and infrequent periods of reduced load.

These results highlight a substantial rise in CPU demand as the number of hosts grows, underscoring the scalability challenges associated with running vulnerability scans in environments with a high number of connected devices.

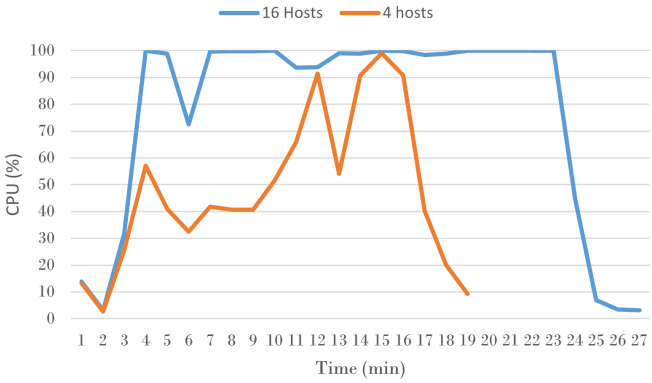

FIGURE 6. CPU utilization during vulnerability scanning with 4 and 16 hosts in the physical testbed. Scans were performed using GVM with the Full and Fast configuration, and measurements were collected from the Security Tools Server running Kali Linux on a virtual machine.

Fig. 6 illustrates CPU utilization over time for workloads involving 4 and 16 hosts, respectively. In both scenarios, the initial CPU load is approximately 13%, which is attributed to the initiation of tasks within the GVM system. In the 16-host scenario, CPU usage subsequently exhibits a marked increase, particularly at minute four, reaching saturation at 100%. This sharp increase likely corresponds to the initial intensive execution phase of vulnerability tests (VTs). Following these peaks, CPU utilization decreases around minute six, with further reductions observed at minute eleven. Toward the conclusion of the scan, resource usage progressively declines as expected. Conversely, in the 4-host scenario, CPU utilization reaches saturation only once at minute fifteen, demonstrating overall more efficient resource management throughout the measurement period. The observed fluctuations in CPU utilization are presumably correlated with the timing of specific VTs, reflecting their varying computational complexity and intensity. Comparing periods of highest CPU utilization across both scenarios, it is evident that the 16-host scan requires approximately seven additional minutes to conclude compared to the 4-host scenario, a result consistent with the findings presented in Fig. 5.

C. MEMORY USAGE

This subsection analyzes RAM usage during the various test scenarios. As shown in Table 5 and Fig. 7, memory consumption remains relatively stable despite the increase in the number of scanned hosts. Although there is a slight upward trend in RAM usage as more hosts are added, the variation is minimal and does not follow the same increasing pattern observed in scan duration results.

These findings suggest that GVM handles memory efficiently, even as the workload scales, indicating that RAM is not a limiting factor in the scanning process. Instead, the CPU appears to be the primary contributor to the increased scan times. RAM usage values were derived from a representative sample, with the initial memory usage recorded at 37% (approximately 4.3 GB).

TABLE 4. CPU utilization results during vulnerability scans with 1 to 32 hosts in the physical testbed. Values include maximum, average, and standard deviation of CPU load measured on the Security Tools Server running GVM with the Full and Fast configuration.

CPU load	1 Host	2 Hosts	4 Hosts	8 Hosts	16 Hosts	32 Hosts
Max (%)	27.5	58.6	99.0	100	100	100
Avg (%)	14.3	24.3	47.8	78.0	82.1	87.7
STD Deviation (%)	6.7	13.9	29.0	35.2	32.9	27.0

TABLE 5. RAM usage results during vulnerability scans with 1 to 32 hosts in the physical testbed. Maximum, average, and standard deviation values were recorded on the Security Tools Server to assess the impact of scaling on memory consumption.

RAM memory	1 Host	2 Hosts	4 Hosts	8 Hosts	16 Hosts	32 Hosts
Max (%)	41.6	42.3	43.2	46.7	50.4	52.9
Avg (%)	41.1	41.6	42.1	43.4	44.4	44.9
STD Deviation (%)	0.5	0.6	0.8	1.7	2.7	2.8

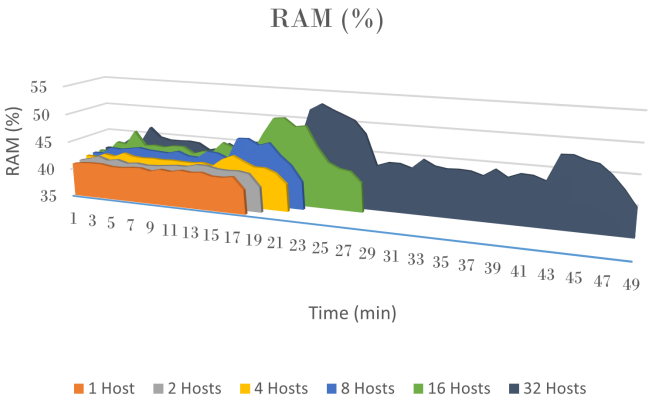

FIGURE 7. Comparison of RAM usage during vulnerability scans with with 1 to 32 hosts in the physical testbed. Scans were conducted using GVM (Full and Fast configuration) on Windows-based target virtual machines, with memory consumption monitored on the Security Tools Server.

The analysis of RAM usage over time, as illustrated in Fig. 7, revealed no significant outliers, confirming the consistency of the results throughout the testing process. This stable behavior indicates that no anomalous or unexpected events occurred during the test runs, thereby reinforcing the reliability of the experimental data. The absence of abrupt fluctuations in memory usage suggests that the system maintained predictable performance, even under varying load conditions.

D. NETWORK OVERHEAD

This subsection examines the network bandwidth utilized during the different test scenarios. Fig. 8 presents bandwidth usage as a function of the number of scanned hosts. Throughout the tests, upload bandwidth consistently remained lower than download bandwidth, indicating that the scanner received more data than it transmitted.

Following an initial increase, the growth in bandwidth usage begins to plateau once the number of hosts exceeds 8. This stabilization is likely due to the system reaching 100% CPU utilization, which limits the scanner's ability to process and execute vulnerability tests in parallel. Both upload and download bandwidth usage show a positive correlation with the number of hosts, although the increase in download bandwidth is more pronounced. The average bandwidth values were calculated from a representative sample. Initial average values were recorded at 0.7296 Kbps for upload and 1.0203 Kbps for download.

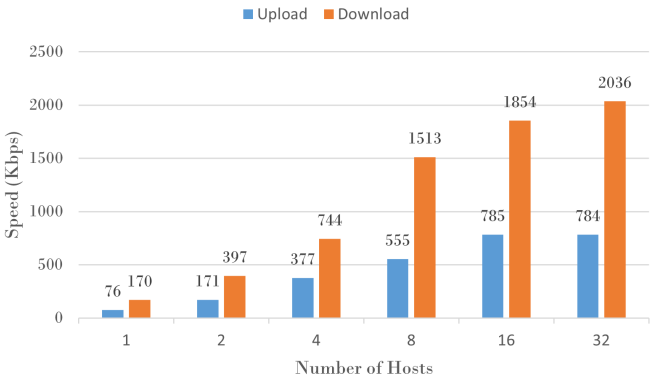

FIGURE 8. Average upload and download bandwidth usage during vulnerability scans with 1 to 32 hosts in the physical testbed. Tests were conducted on a 100 Mbps LAN, using GVM with the Full and Fast configuration. Bandwidth consumption was monitored between the Security Tools Server and the scanned hosts.

Although bandwidth usage increased over time, the values remained well within acceptable limits and did not represent a bottleneck for the vulnerability scanning process. The tests were conducted on a LAN with a capacity of 100 Mbps, which is significantly higher than the peak bandwidth observed in Fig. 8, approximately 2 Mbps. This means that the scanning process utilized only about 2% of the available network capacity, confirming that bandwidth was not a limiting factor in this context.

E. PERFORMANCE AND ORCHESTRATION TIMING

The tests conducted in the virtual environment involved running vulnerability scans on a single host running the Windows 2003 operating system (see Table 2), hosted within a VMware virtual machine. The objective of these tests was to measure the execution time of each step in the workflow, not to benchmark different operating systems, but rather to assess the orchestration performance of the proposed SOAR-based framework. Using a host with well-documented vul-

nerabilities allowed for a consistent and efficient evaluation of detection, response, and mitigation sequences. To support this, a logging mechanism was implemented on both the SOAR Server and the Security Tools Server, enabling precise tracking of when the system transitioned from one task to the next (refer to Fig. 4).

Table 6 presents the time taken for each step. By analyzing these durations, it becomes possible to identify stages that may be contributing disproportionately to the overall execution time. The first entry, "Device Discovery," represents the time between initiating the network discovery request and receiving a response. The second column, "Prepare Vuln Scan," shows the time required to prepare the vulnerability scan request, which includes queuing the device information and creating an automation ticket, until the scan request is initiated.

Following this, the vulnerability scan itself is executed for a single host, lasting approximately 18 minutes as shown in Fig. 5; this duration is not included in the Table 6. The "Request Report" column reflects the time between requesting and receiving the vulnerability report, while "Parsing Report" visualizes the time taken to process and interpret the report data. Lastly, the "Change VLAN" column indicates the time required to reassign the VLAN, and the diverse values associated to the "Change IP" column are available in Table 7.

The communication time between the SDN controller and the switch during the VLAN change process was also evaluated. This was done by measuring the time interval between the transmission of the `FLOW_MOD` message and the receipt of its corresponding acknowledgment (ACK). However, some discrepancies were observed in the recorded timings during message capture.

In one instance, the controller sent three separate `FLOW_MOD` messages, each corresponding to a distinct command (ADD, DELETE, and ADD), resulting in a total communication time of 0.000075302 seconds. In another case, all command headers were encapsulated within a single message, yielding a shorter time of 0.000024276 seconds. Despite these variations, both durations are extremely short and do not introduce any meaningful delay to the system, as expected.

Referring to Table 6, we observe that once the vulnerability is detected by GVM, the combined time to request (16 ms) and parse (3 ms) the vulnerability report totals only 19 ms. This swift processing enables the system to rapidly identify the need for mitigation, promptly isolating the vulnerable host and minimizing exposure to potential security threats.

An additional issue identified during testing in the virtual environment was the delay in reestablishing network connectivity at the host level following a VLAN change. The expected behavior was for the host's network interface to automatically disconnect and reconnect after the VLAN switch. While the VMware virtual switch correctly detected the interface as inactive, it failed to trigger any action on the host side. As a result, the host retained its previous IP address and did not initiate a new DHCP request.

Consequently, even though the host was now connected to a switch port in the new VLAN, it continued operating with the old IP address until the DHCP lease expired and was renewed. To address this issue, the DHCP server's lease time was significantly reduced, enabling faster reassignment of IP addresses and ensuring that hosts could more promptly adapt to the new network context.

While reducing the DHCP lease time represents a practical workaround, it incurs additional network overhead due to more frequent lease renewal traffic. Although alternatives such as leveraging SDN to trigger DHCP renewals or implementing 802.1X-like port-based authentication may offer more direct control, they typically require either root access, specialized client-side software, or tight integration with specific networking infrastructures. In our opinion, such approaches compromise transparency and limit general applicability. By contrast, the proposed solution avoids the installation of agents or host-level code execution, preserving the non-intrusive nature of the system. It also remains agnostic to specific network configurations or authentication mechanisms, operating effectively across generic switches and diverse topologies, thus maximizing its deployability and ease of integration.

As previously discussed, VLAN switching was executed almost instantaneously at the SDN controller level, effectively isolating the vulnerable host by applying the appropriate OpenFlow rules to the topology switch. However, despite the rapid VLAN reassignment, the host remained disconnected until it obtained a new IP address. Adjusting the DHCP lease time helped mitigate this limitation, enhancing the overall responsiveness of the mitigation process.

To evaluate this behavior, two tests were conducted using DHCP lease times of one and five minutes, respectively. Each test included 40 measurements to determine the average time required for lease renewal. Theoretically, the expected average renewal time is half the lease duration, assuming that the VLAN change request occurs randomly within the lease cycle. Under a uniform distribution of waiting times, the average remaining time until lease expiration should converge to this midpoint.

A critical aspect of the testing methodology was ensuring that VLAN change requests were issued at random points within the lease interval. If requests were made immediately after one another, the host would consistently wait the full lease duration before acquiring a new IP address, skewing the results. To ensure accurate and representative sampling, requests were randomized throughout the lease cycle.

The results, presented in Table 7, confirm that the observed average renewal times align with theoretical expectations, falling within the standard deviation range.

F. SCALABILITY AND PERFORMANCE BOTTLENECKS

The evaluation results demonstrated the proposal's responsiveness, with no excessive delays observed throughout the complete SOAR workflow (Table 6). The vulnerability detection process was divided into several critical functional steps, including device discovery, preparation for vulnera-

TABLE 6. Execution time of each orchestration step in the virtual testbed during a single-host vulnerability scan. Measurements include device discovery, vulnerability scan preparation, report retrieval and parsing, and mitigation actions (VLAN and IP reassignment).

Device Discovery	Prepare Vuln Scan	Request Report	Parsing Report	Change VLAN	Change IP
1,5 sec	0,98 sec	0,016 sec	0,003 sec	$\sim$ 0 sec	Table 7

TABLE 7. Delay in IP renewal following VLAN reassignment, measured under different DHCP lease times (60 s and 300 s) in the virtual testbed. Results show the average renewal time and standard deviation across 40 measurements.

Lease Time (sec)	Average (sec)	STD Deviation (sec)
60	38	8
300	212	62

bility scanning, and report generation, all of which exhibited minimal execution times. The mitigation measure, i.e. Change VLAN in the present scenario, was executed almost instantaneously by the SDN controller, indicating that the tested architecture can timely isolate susceptible hosts. Only the vulnerability scanning phase showed a noticeably extended duration due to its inherent job complexity.

The laboratory tests revealed that GVM exhibits high CPU utilization when scanning more than eight hosts concurrently, which can significantly prolong the overall scanning duration. These reported results reflect a worst-case scenario in which all hosts were scanned simultaneously to stress-test the system and expose its upper performance bounds. In practical deployments, however, scalability can be improved through several strategies. These include enforcing a predefined concurrency limit with queued jobs, distributing workloads across scanning clusters, or applying scan scheduling to stagger vulnerability assessments. Alternatively, lightweight preliminary scans could be employed to reduce the load on the main vulnerability assessment engine. Finally, the modular design of the Security Tools Server enables parallelization across multiple instances, further enhancing scalability and performance. We identify the application of these techniques as an important direction for future work.

G. IMPLICATIONS FOR SYSTEM ENHANCEMENT

The current solution operates within the internal boundaries of the organization's network, benefiting from existing in-situ security mechanisms, such as, firewall policies. While the proposed SOAR-based approach does not currently interface directly with these legacy assets, real-world deployments often require coordination with broader enterprise security ecosystems, including SIEM platforms and firewalls. To this end, future work will explore integration strategies, such as, the use of RESTful APIs provided by either SIEM platforms or firewalls, enabling the SOAR correct orchestration with legacy security infrastructure. Such orchestration is essential for enabling a scalable, cohesive, and responsive security management in complex enterprise environments. The current solution relies uniquely on the Nmap tool [20] to discover new hosts connected to the network. Additionally, in future investigation, other system components, such as the DHCP server or the SDN controller could be used to discover a new arriving host (or in general, detect any change in the network topology) and inform the SOAR framework about that. As an example, after the SOAR is notified about a new host, it could trigger right away a vulnerability scan process on that specific host, avoiding the SOAR resort to a more resource-intensive active scanning in the network.

The risk evaluation of each host vulnerability should combine the CVSS score with other future compensating factors, such as the likelihood of exploitation by a malicious actor. In this way, a high CVSS score associated with a specific vulnerability may be adjusted downward if the probability of successful exploitation is low. Such a scenario may occur, for example, when an attacker would need local access or elevated remote privileges on the host before being able to exploit the vulnerability. Additionally, other adjustment factors available from public vulnerability databases, e.g. [23], they can be also incorporated on the final calculation of the host vulnerability risk. Some previous proposals [3], [4] studied the vulnerability's exploitability value by an attacker and they should be considered in future work. Other interesting idea to modulate the host risk is to factorize the average number of interactions the affected device have performed with other network devices during a specific time window.

Let us assume now the scenario of the current SOAR-based proposal to select a possible mitigation action like moving a high-risk host to a quarantine VLAN. This host isolation should remain in place until corrective actions are performed on the host to eliminate its vulnerabilities. Nevertheless, this solution could have a drawback. For instance, isolating in this way a critical server simply because it crosses the severity risk threshold could result in service disruptions or broader network issues. In this way, as future work, the mitigation solution selected by the SOAR should also consider the role or system's function of the affected device within the network.

The next Section concludes the paper and presents further guidelines for upcoming research.

VI. CONCLUSIONS AND FUTURE WORK

This section presents the conclusions drawn from this research and proposes further future work, complementing what was already debated in V-G.

A. CONCLUSIONS

This research explored the proactive and automated detection and mitigation of vulnerabilities in network environments managed by SDN, using a SOAR platform to orchestrate var-

ious open-source tools. The system automates processes such as discovering devices, assessing vulnerabilities, analyzing results, and executing mitigation measures, ensuring interoperability, proactivity, and adaptability. It integrates tools like Nmap and GVM through a Security Service Adapter, enabling workflow automation and continuous monitoring. Vulnerable devices are promptly isolated via VLAN switching implemented by the SDN controller, with a modular architecture allowing future enhancements. Beyond the integration of existing tools, this work presents a cohesive and novel framework, where the orchestration logic and SOAR playbook are central to its innovation. These elements enable adaptive and context-aware responses that surpass current approaches in the literature, highlighting the system's unique contribution to the field.

Tests conducted in laboratory and virtual environments validated the system's functionality and performance. Laboratory results highlighted GVM's significant CPU demands when scanning more than eight devices, with CPU usage increasing from 14–48% for 1–4 devices to 78–88% for 8–32 devices. RAM usage remained stable (41–45%), while bandwidth ranged from 76 Kbps (TX) and 170 Kbps (RX) for one device to 784 Kbps (TX) and 2036 Kbps (RX) for 32 devices. Scan durations increased from 18 minutes for one device to 49 minutes for 32 devices. The VLAN switching mitigation was applied almost instantly, taking just 19 ms, demonstrating the system's responsiveness and efficiency.

The research successfully developed a proactive system that automates vulnerability detection and mitigation by integrating SDN, open-source tools, and a SOAR platform. The system efficiently identified vulnerabilities, implemented VLAN switching as a mitigation measure, and with effective orchestration among components, addressing all initial goals.

B. FUTURE WORK

Complementing what was already discussed in Section V-G, below are presented some suggestions for future work, as follows: i) node classification and scanning; ii) applying more mitigation measures; iii) vulnerabilities in IoT sensor environments; iv) automated intelligent generation of system security tests; and v) multiple AI agents managing vulnerabilities in network domains controlled by SDN.

Node Classification and Scanning: The SDN controller can be further enhanced to classify network nodes based on their contextual relevance within the topology, such as, their proximity to critical assets or the nature of the services they provide. This would enable prioritization of vulnerability scans for devices that perform essential functions, such as servers or nodes with elevated privileges. Moreover, the framework could benefit from the implementation of customizable scan configurations. For instance, it could dynamically adjust scan aggressiveness based on system load, conducting lighter scans during peak usage periods to minimize performance impact, and reserving more intensive scans for off-peak hours. Additionally, the scanning strategy could be adapted based on a device's position within the network and the number of its active direct neighbors. This would allow the system to fine-tune the frequency and intensity of scans, focusing resources where they are most needed and improving overall efficiency and responsiveness.

Applying more mitigation measures: To ensure a robust defense posture, the implementation of additional mitigation measures is essential. The system developed in this research was designed with flexibility in mind, allowing for the seamless integration of new mitigation strategies. Deep Packet Inspection (DPI) serves as a valuable complement to VLAN isolation, particularly for devices with low-risk vulnerabilities. By monitoring network traffic in real time, DPI enables the detection of potentially malicious activity without restricting the device's access to system resources, thus maintaining operational continuity while enhancing security. Moving Target Defense (MTD) is another mitigation approach that can be applied in cases where immediate isolation is not required. Vulnerable devices could be placed in a dedicated VLAN where MTD techniques are employed, such as, periodically changing the device's IP address—to create a dynamic and less predictable network presence. This provides an intermediate level of containment that is less disruptive than full isolation. Additionally, a more granular analysis of the CVSS vector could be leveraged to tailor mitigation strategies. By interpreting individual metrics within the vector, the system can derive more context-aware insights, enabling the application of mitigation measures that are better aligned with the specific characteristics and severity of each vulnerability. Additional mitigation strategies could also draw upon architectures designed for media-independent handovers [30] and flow mobility management, as exemplified in systems combining PMIPv6 with IEEE 802.21 for simultaneous multi-access across heterogeneous networks [31]. These approaches enable dynamic routing of traffic flows across multiple interfaces (e.g., WiFi and cellular), providing flexible control and continuous service even during access transitions, capabilities that could support the deployment of mitigation zones or adaptive quarantine in mobile IoT contexts.

Vulnerabilities in IoT Sensor Environments: A key direction for future work is to extend the framework's capabilities to IoT environments, where devices are often deployed without sufficient security controls, making them potential attack vectors. IoT scenarios introduce new technical considerations: devices typically have constrained processing power and battery life, and they utilize diverse communication protocols that may offer limited bandwidth or intermittent connectivity. Integrating support for these technologies will likely require adapting our scanning and mitigation techniques to operate efficiently under such constraints. For example, one promising approach is to leverage SDN-based admission control for IoT endpoints, similar to the framework proposed in [11], which automatically scans IoT devices for known vulnerabilities before allowing them to join the network. If a device is found to be vulnerable, that system attempts automated remediation or isolates the device (e.g.,

using firewall rules) to prevent it from exposing the network to risk. Notably, this type of pre-admission scanning was shown to incur only minimal performance overhead (on the order of a few milliseconds of added latency), suggesting that our architecture could incorporate similar IoT device checks without significantly impacting overall performance.

We plan to explore this direction in crowd-monitoring scenarios, where sensors are deployed in urban environments, heritage sites, and public spaces to collect occupancy and movement data [32], [33]. Adapting the current scanning and mitigation system to support such devices would demonstrate its applicability to smart city infrastructures, where ensuring the secure operation of sensor networks is essential for maintaining data integrity, system resilience, and public trust.

Self-generation of automatic system security tests: Generative AI can produce security tests to perform more efficient automatic pentesting procedures, reducing human intervention and enabling a more proficient detection and posterior mitigation of eventual security vulnerabilities [34]. Pentesting is also referred as penetration testing [35] or ethical hacking [36]. Another interesting topic to investigate is explainable artificial intelligence (XAI) [37], [38]. The main aim for using XAI in scenarios where AI/ML models artificially self-produce security tests for high complex systems is to guarantee to the humans as network security managers to always trust on all the (learned) management security decisions made by artificial agents, because these agents never could make a wrong automatic decision. In this way, AI models must satisfy the following range of criteria to boost in humans a sufficient level of trust on the automatic AI decisions [38]: reliability, safety, privacy, explanatory justifiability, impartiality (fairness), and usability. A recent review [39] on interpretability in AI offers a new strategy on how humans can develop trust in the automated decisions made by a specific AI model. This trust can be established if people understand how the AI algorithm is trained and how it arrives at reliable security management decisions across various system scenarios. The authors of [40] provide theoretical foundations of XAI, clarifying important definitions and identifying research goals, open issues, and future research lines related to turning opaque ML outputs into more transparent decisions to humans. In [41], two XAI methods, i.e. LIME and SHAP, were used in a machine learning-based intrusion detection system. They have conducted a survey analysis in which participants answered questions evaluating the degree of interpretability increase when each XAI method was used, directly comparing both methods. The authors conclusions are in [41].

Autonomous vulnerability management in SDN: The autonomous [42] detection and mitigation of device security vulnerabilities [1] in SDN environments [2] should be viewed as a cross-cutting capability that interacts with all primary SDN control actions, namely, flow management, traffic engineering, topology control and, fault detection and recovery. Rather than functioning as an isolated task, AI agents responsible for the autonomous security vulnerability management could influence and modify each one of these control loops. For instance, flow rules may be updated to isolate malicious traffic, traffic paths may be rerouted to avoid compromised nodes, topology may be reconfigured to disable affected links, and anomaly detection mechanisms may trigger security responses. As such, AI-based vulnerability management acts as a longitudinal layer, enhancing the overall resilience and trustworthiness of the network.

REFERENCES

[1] João Polónio, José Moura, and Rui Neto Marinheiro. On the road to proactive vulnerability analysis and mitigation leveraged by software defined networks: A systematic review. *IEEE Access*, 12:98546–98566, 2024.

[2] Rahim Masoudi and Ali Ghaffari. Software defined networks: A survey. *Journal of Network and Computer Applications*, 67:1–25, 5 2016.

[3] Mengmeng Ge, Jin-Hee Cho, Dongseong Kim, Gaurav Dixit, and Ing-Ray Chen. Proactive defense for internet-of-things: moving target defense with cyberdeception. *ACM Transactions on Internet Technology (TOIT)*, 22(1):1–31, 2021.

[4] Seunghyun Yoon, Jin-Hee Cho, Dong Seong Kim, Terrence J Moore, Frederica Free-Nelson, and Hyuk Lim. Attack graph-based moving target defense in software-defined networks. *IEEE Transactions on Network and Service Management*, 17(3):1653–1668, 2020.

[5] Ankur Chowdhary, Adel Alshamrani, Dijiang Huang, and Hongbin Liang. Mtd analysis and evaluation framework in software defined network (mason). In *Proceedings of the 2018 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, page 43–48, 2018.

[6] Sukwha Kyung, Wonkyu Han, Naveen Tiwari, Vaibhav Hemant Dixit, Lakshmi Srinivas, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Honeyproxy: Design and implementation of next-generation honeynet via sdn. In *2017 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9. IEEE, 2017.

[7] George Stergiopoulos, Panagiotis Dedousis, and Dimitris Gritzalis. Automatic analysis of attack graphs for risk mitigation and prioritization on large-scale and complex networks in industry 4.0. *International Journal of Information Security*, 21(1):37–59, 2022.

[8] Johnson Kinyua and Lawrence Awuah. Ai/ml in security orchestration, automation and response: Future research directions. *Intelligent Automation & Soft Computing*, 28(2), 2021.

[9] Upendra Bartwal, Subhasis Mukhopadhyay, Rohit Negi, and Sandeep Shukla. Security orchestration, automation, and response engine for deployment of behavioural honeypots. In *2022 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8, 2022.

[10] Tao Zhang, Fanyu Kong, Dongshang Deng, Xiangyun Tang, Xuangou Wu, Changqiao Xu, Liehuang Zhu, Jiqiang Liu, Bo Ai, Zhu Han, et al. Moving target defense meets artificial intelligence-driven network: A comprehensive survey. *IEEE Internet of Things Journal*, 2025.

[11] Ruth M. Ogunnaike and Brent Lagesse. Toward consumer-friendly security in smart environments. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops*, pages 612–617, 2017.

[12] Yannis Nikoloudakis, Evangelos Pallis, George Mastorakis, Constandinos X Mavromoustakis, Charalabos Skianis, and Evangelos K Markakis. Vulnerability assessment as a service for fog-centric ict ecosystems: A healthcare use case. *Peer-to-Peer Networking and Applications*, 12:1216–1224, 2019.

[13] Yannis Nikoloudakis, Ioannis Kefaloukos, Stylianos Klados, Spyros Panagiotakis, Evangelos Pallis, Charalabos Skianis, and Evangelos K Markakis. Towards a machine learning based situational awareness framework for cybersecurity: an sdn implementation. *Sensors*, 21(14):4939, 2021.

[14] Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model. Technical report iso/iec tr 25010:2023, International Organization for Standardization (ISO), Geneva, Switzerland, 2023.

[15] J. Polónio. SDN-Vuln: Proactive Discovery and Mitigation of Security Vulnerabilities Leveraged by SDN. https://github.com/linuxer1337/sdn-vuln, 2024. Accessed: Apr. 12, 2025.

[16] Catalyst. Catalyst - Speed up your reactions. https://catalyst.security-brewery.com/, 2025. Accessed: Apr. 12, 2025.

[17] FastAPI. FastAPI framework, high performance, easy to learn, fast to code, ready for production. https://fastapi.tiangolo.com/, 2025. Accessed: Apr. 12, 2025.
[18] Tayyab Muhammad and Muhammad Munir. Network automation. *European Journal of Technology*, 7(2):23–42, 2023.
[19] Redis. Redis: The Real-time Data Platform. https://redis.io/, 2025. Accessed: Apr. 12, 2025.
[20] Nmap. Nmap: Discover your network. https://nmap.org/, 2025. Accessed: Apr. 12, 2025.
[21] Gordon Lyon. Nmap Network Scanning. https://nmap.org/book/, 2009. Accessed: Apr. 12, 2025.
[22] Greenbone AG. Vulnerability Management: Open Source and GDPR-compliant. https://www.greenbone.net/en/, 2025. Accessed: Apr. 12, 2025.
[23] FIRST.org. API that returns Exploit Prediction Scoring System (EPSS) Probabilities for CVE security vulnerabilities. https://api.first.org/, 2025. Accessed: Apr. 12, 2025.
[24] Ryu SDN Framework Community. Python Controller Ryu: Build SDN Agilely. https://ryu-sdn.org/, 2017. Accessed: Apr. 12, 2025.
[25] The Linux Foundation. Open vSwitch: Production Quality, Multilayer Open Virtual Switch. https://www.openvswitch.org/, 2016. Accessed: Apr. 12, 2025.
[26] Mininet Project Contributors. Mininet: An Instant Virtual Network on your Laptop (or other PC). https://mininet.org/, 2022. Accessed: Apr. 12, 2025.
[27] Internet Systems Consortium, Inc. ISC DHCP: Enterprise-grade solution for IP address-configuration need. https://www.isc.org/dhcp/, 2025. Accessed: Apr. 12, 2025.
[28] Internet Systems Consortium, Inc. Kea DHCP: Modern, open source DHCPv4 and DHCPv6 server. https://www.isc.org/kea/, 2025. Accessed: Apr. 12, 2025.
[29] ISCTE-IUL. Iscte-IUL: Laboratories. https://www.iscte-iul.pt/conteudos/research/1002/laboratories, 2025. Accessed: Apr. 12, 2025.
[30] Andreia Mateus and Rui Neto Marinheiro. A media independent information service integration architecture for media independent handover. In *2010 Ninth International Conference on Networks*, pages 173–178, 2010.
[31] Hugo Alves, Luís Miguel Silva, Rui Neto Marinheiro, and José André R S Moura. PMIPv6 Integrated with MIH for Flow Mobility Management: A Real Testbed with Simultaneous Multi-Access in Heterogeneous Mobile Networks. *Wireless Personal Communications*, 98(1):1055–1082, 2018.
[32] Rúben Dias da Silva, Rui Neto Marinheiro, and Fernando Brito e Abreu. Crowding detection combining trace elements from heterogeneous wireless technologies. In *2019 22nd International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 1–6, 2019.
[33] Tomás Mestre dos Santos, Rui Neto Marinheiro, and Fernando Brito e. Abreu. Wireless crowd detection for smart overtourism mitigation. In Elena Kornyshova, Rébecca Deneckère, and Sjaak Brinkkemper, editors, *Smart Life and Smart Life Engineering: Current State and Future Vision*, pages 237–258. Springer Nature Switzerland, Cham, 2025.
[34] Eric Hilario, Sami Azam, Jawahar Sundaram, Khwaja Imran Mohammed, and Bharanidharan Shanmugam. Generative ai for pentesting: the good, the bad, the ugly. *International Journal of Information Security*, 23(3):2075–2097, 2024.
[35] Qianyu Li, Miao Hu, Hao Hao, Min Zhang, and Yang Li. Innes: An intelligent network penetration testing model based on deep reinforcement learning. *Applied Intelligence*, 53(22):27110–27127, 2023.
[36] Yien Wang and Jianhua Yang. Ethical hacking and network defense: Choose your best network vulnerability scanning tool. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 110–113, 2017.
[37] Sajid Ali, Tamer Abuhmed, Shaker El-Sappagh, Khan Muhammad, Jose M Alonso-Moral, Roberto Confalonieri, Riccardo Guidotti, Javier Del Ser, Natalia Díaz-Rodríguez, and Francisco Herrera. Explainable artificial intelligence (xai): What we know and what is left to attain trustworthy artificial intelligence. *Information fusion*, 99:101805, 2023.
[38] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0210–0215, 2018.
[39] Ujjwal Singh Kathait, Anamika Rana, Rahul Chauhan, and Ruchira Rawat. A comprehensive review of interpretability in ai and its implications for trust in critical applications. In *2024 4th International Conference on Sustainable Expert Systems (ICSES)*, pages 1683–1693, 2024.
[40] Evandro S. Ortigossa, Thales Gonçalves, and Luis Gustavo Nonato. Explainable artificial intelligence (xai)—from theory to methods and applications. *IEEE Access*, 12:80799–80846, 2024.
[41] Diogo Gaspar, Paulo Silva, and Catarina Silva. Explainable ai for intrusion detection systems: Lime and shap applicability on multi-layer perceptron. *IEEE Access*, 12:30164–30175, 2024.
[42] Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2):895–943, 2022.

JOÃO POLÓNIO received his B.Sc. in Telecommunications and Computer Engineering from Iscte – Instituto Universitário de Lisboa, Portugal, in 2021. During his undergraduate studies, he developed a strong interest in computer networks and network security. He completed his M.Sc. in Telecommunications and Computer Engineering at Iscte in 2024. Throughout his time at Iscte, he has actively contributed to teaching activities in computer network architectures and has been involved in the management and support of departmental laboratories.

JOSÉ MOURA received the B.Sc. degree in Electronics and Telecommunications from the Universidade de Aveiro, Portugal, in 1989; the M.Sc. degree in Computer Networks from the Faculdade de Engenharia, Universidade do Porto, Portugal, in 2001; and the Ph.D. degree in Computer Science from Lancaster University, U.K., in 2011. From 1989 to 2000, he worked as an Engineer in Supervisory Control and Data Acquisition (SCADA) systems and industrial automation at EFACEC Sistemas Electrónica, Portugal. From 2000 to 2001, he was a Researcher at INESC, Porto, Portugal. Since 2001, he has been with Iscte – Instituto Universitário de Lisboa, Portugal, where he teaches Computer Networks, and he is also a Researcher with the Instituto de Telecomunicações, Portugal. He serves as an active reviewer for several Quartile 1 journals. His current research interests include network management, edge computing, optimization, virtualization, software-defined networking, security and resilience in networked systems.

RUI NETO MARINHEIRO is an Associate Professor at Iscte – Instituto Universitário de Lisboa, Portugal, and a researcher at the Instituto de Telecomunicações. He holds a Ph.D. in Multimedia Information Systems from the University of Southampton, United Kingdom, and an M.Eng. in Electrical and Computer Engineering with a specialization in Telecommunications from the Faculty of Engineering, University of Porto, Portugal. Dr. Marinheiro has extensive experience in teaching and research in the fields of Telecommunications, Computer Networks, Security, and the Internet of Things. He has coordinated and contributed to numerous national and international research projects.