Journal of Information Security and Applications 60 (2021) 102878

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Cyberprotection in IoT environments: A dynamic rule-based solution to defend smart devices

Pantaleone Nespoli [a,*], Daniel Díaz-López [b], Félix Gómez Mármol [a]

[a] Department of Information and Communications Engineering, University of Murcia, 30100 Murcia, Spain
[b] School of Engineering, Science and Technology, Universidad del Rosario, Bogotá, Colombia

ARTICLE INFO

Keywords:
COSMOS
Cybersecurity
IoT security
Resource-constrained device
Smart home

ABSTRACT

Undoubtedly, modern human digital lives are every day more and more connected, and the revolution of "everything connected" is already becoming a reality. Indeed, humans live in the age of the Internet of Things (IoT), and one of the most usual IoT contexts is a smart home. Unfortunately, such significant enhancement also means that common home devices, such as fridges, cameras, or even bulbs, are exposed to malevolent entities whose primary goal is to threaten the confidentiality, integrity, and availability of the automatically-exchanged information. Aiming at fine-tuning the protection of the smart devices, this paper proposes a novel dynamic rule management solution adaptable to the current status of the IoT environment, so to protect it against cyberattacks. Experiments demonstrated that a notable reduction in the CPU and RAM consumption was achieved when applying this novel scheme. Additionally, the number of packets processed per second increased substantially, inducing a meaningful enhancement also from a security perspective.

1. Introduction

Digital threats may be considered as prevailing and ubiquitous nowadays, targeting personal phones, laptops, and electronic devices in general [1,2]. Thus, cybersecurity plays a vital role in fighting against those threats [3,4], as it protects societal digital assets from threats that impact the ongoing digital transformation [5].

Smart devices are massively being incorporated in homes thanks to IoT [6]. Thus, a "smart home" hosts IoT devices that exchange data with remote web servers and provide IoT services to users. Due to the value of such exchanged information, these IoT devices become attractive targets for malicious entities that aim to gain access to sensitive information, violating confidentiality, integrity, or availability of such users' data [2]. In this regard, intrusion detection systems (IDS) may be proposed as a solution to protect IoT devices, as IDS would be able to monitor a scenario, process and extract information of interest and finally determine the presence of malicious traffic.

The protection of smart homes is strongly impacted by the limited computational capabilities of most of the composing smart devices, such as RAM, CPUs, internal storage, and communication throughput. So, a suitable proposal of a security solution for smart homes should guarantee an efficient but effective use of the available resources.

Also, a highly-desired feature of IoT security proposals is the capability of dynamically adapting to the changes in the scenario in an effort to utilize the number of resources that are strictly necessary to protect the surrounding scenario. Thus, the defense of the smart devices against the threats should avoid the underprotection and overprotection situations where an insufficient or excessive amount of resources, respectively, are being employed.

Furthermore, security proposals for IoT scenarios need also to consider that IoT services being consumed should not be impacted drastically. Thus, security solutions designed for smart homes should avoid introducing a notable latency in the user experience, focusing on offering non-intrusive security that supports its adoption under the premise of usability.

Additionally, the design of security solutions for smart homes should consider that common users within such scenarios have little or no experience in IoT management or cybersecurity skills. So, it is crucial to develop security solutions able to protect the digital assets inside a smart home appliance with no or minimal user intervention, but guaranteeing that the user is informed of the overall security situation [1,2].

All these previously mentioned challenges require innovative cybersecurity proposals that bring computational efficiency, dynamic adaptability to the current scenario, ease of use, and non-intrusiveness, so that the attacks over user data and devices may be detected promptly, and the effects may be appropriately contained, avoiding a negative impact on the smart ecosystem.

* Corresponding author.
E-mail addresses: pantaleone.nespoli@um.es (P. Nespoli), danielo.diaz@urosario.edu.co (D. Díaz-López), felixgm@um.es (F. Gómez Mármol).

https://doi.org/10.1016/j.jisa.2021.102878

Available online 20 May 2021

1.1. Objectives

In our previous work, we proposed COSMOS [7], a seamless sentinel to shield surrounding IoT devices leveraging several protection levels, as we will see in Section 3. In truth, COSMOS provides interesting security characteristics, and it is user-friendly and non-intrusive. Nonetheless, one could argue that it lacks computational efficiency since it would be desirable to achieve its cybersecurity purposes in a more resource-efficient way. Besides, COSMOS does not consider adaptation to the threats perceived in the scenario, which is a desired feature in an effort to protect the IoT devices and reduce resource utilization, preserving the same protection adequacy.

In light of the above, the primary purpose of the paper at hand is to enhance and optimize the COSMOS framework so that it can be deployed in resource-constrained devices in an flexible manner and still being able to shield the IoT devices. To achieve such ambitious goal, the following objectives are defined:

- **Objective 1**: Monitor the active surrounding IoT devices and services and identify potential vulnerabilities exposed by IoT devices in a timely fashion.
- **Objective 2**: Adapt the sentinel dynamically to the changes of the surrounding smart devices or services, deploying rules suitable for the possible newly discovered security flaws (e.g., vulnerabilities) in an effort to guarantee maximum protection uninterruptedly.
- **Objective 3**: Create a mechanism that allows the creation of dynamic mitigation rules by using the information about the detected threats and enforce them within the considered smart scenario (e.g., smart home).

The paper at hand is organized as follows: Section 2 (State of the art) gives an overview of the major solutions proposed in the literature on the IoT security ecosystem. Section 3 (COSMOS proposal) briefly describes the design of the COSMOS framework paving the path to the proposed improvements. Section 4 (Architecture) explains how our solution was developed with a particular focus on the main enhancements. Section 5 (Experiments) illustrates the exhaustive tests made to prove the feasibility of the presented framework, highlighting the comparison between COSMOS and our proposal. Finally, Section 6 (Conclusions and future work) concludes the paper and analyzes possible future works lines.

2. State of the art

Nowadays, it is possible to find a vast amount of proposals related to IoT security ecosystems, many of them approaching the issue uniquely. Additionally, different IoT frameworks exist nowadays [8,9], which have some security functionalities mainly to consider features like authentication, authorization, and secure communication, even though many security challenges are still opened. This section describes several proposed solutions, summarizing their features and highlighting their differences with the proposal presented in the paper at hand.

An IDS architecture for IoT environments (RPiDS) based on four main aspects is proposed in [10]. This proposal leverages Snort as the primary IDS running on a Raspberry Pi and employs three basic rule sets, manually selected, with different security levels. Application of a rule set for a particular risky scenario may be considered adaptive, even if it is a complex procedure that requires a broad knowledge of the network services and assets from the user.

In further analysis, a host-based Intrusion Detection and Mitigation framework named IoT-IDM is proposed in [11]. This proposal focuses on home scenarios and takes advantage of Software Defined Network (SDN) technology through an SDN controller, which allows to have network visibility and provides flexibility to configure, manage and secure the network remotely. Machine Learning (ML) techniques to detect suspicious network patterns and attacks are also incorporated.

Some methods based on deep learning techniques to guarantee cyber safety in smart home/office scenarios are proposed in [12]. Specifically, two anomaly detection models are built based on Long–Short Term Memory (LSTM) neural networks. The first anomaly detection model focuses on the features of the MQTT (Message Queuing Telemetry Transport) traffic flowing in the smart home/office. The second one operates on the measurement of the Total Apparent Power from smart devices and sensors deployed.

Another clever approach to cybersecurity for IoT environments is presented in [13], where a novel proposal able to predict malicious behavior and detect malevolent IoT nodes is proposed. This proposal is made of a machine learning model that learns the IoT-based network's behavior and identifies attacks such as network probing and SYN/UDP flood attacks. A rule-based component established according to a security policy is also part of this proposal.

An IDS architecture able to distinguish malicious network traffic, discover the type of attack, and identify the attacked device is proposed in [14]. Such IDS architecture is based on three models: (i) a ML module that profiles 8 IoT devices, (ii) a ML module that identifies anomalies in the network traffic of the IoT devices and may classify it as malicious or normal, and (iii) a ML module that recognizes which type of attack has been executed.

An IoT rule generation framework composed of a ML rule discovery, a threat prediction model and a monitor of rules violation and traffic behavior changes is proposed in [15]. The framework collects training data from sensors, extracts meaningful features, and then composes customized rules using a Random-Forest (RF) model.

Another innovative solution was proposed in [16], where the selection of the node where security services should be deployed inside an IoT network is performed using graph theory. Such a proposal considers the devices' capabilities to build a weighted graph and identify the dominating set, so the most suitable location to deploy a security service can be defined.

An interesting analysis about applying conventional IDS solutions in IoT networks is presented in [17], considering properties like resilience, redundancy, communication overhead, and energy consumption. Such proposal contains a central IDS agent and a cluster of distributed IDS agents that defend an IoT network in a collaborative way using IoT routing protocols.

An adaptable intrusion detection system for IoT is proposed in [18]. Such a system is implemented using an Odroid xu3 board without interfering with existing IoT devices and services, and it also offers adaptability to the mobility of nodes in the network through changes in signal strength.

A framework to protect IoT devices in healthcare scenarios is proposed in [19], which uses an attack–defense model to detect attacks and choose a defense strategy. Such work also uses evolutionary game theory to test the behavior of the proposal.

An IDS inspired by genetic algorithms is defined in [20], which is able to detect specific network attacks like DoS, probe, Remote-to-User (R2L), and User-to-Root(U2R). The overall detection rate of this proposal is 80% through the classification of protocols and the counting of features.

A scheme to detect attacks in IoT using deep learning is proposed in [21]. This proposal focuses on the training of fog nodes that compose a distributed architecture and are able to obtain a remarkable accuracy that depends on the number of fog nodes deployed.

Blacksite is an IDS based on artificial immune systems as described in [22]. This IDS classifies malicious traffic and is able to deny connections until a human may confirm the legitimacy of the traffic. In this proposal, the IoT nodes trained to warn about the presence of malicious traffic are called detectors.

A model to fight against DDoS (Distributed Denial of Service) attacks in IoT networks is proposed in [23], which includes four types

Table 1
Comparison of related works.

Related work	Scenario of application	Core technique	Main security function	Mitigation of the attack	Adaptative to changes	Require additional management	Main infrastructure required	Computational performance measures
[10]	General	Snort IDS rules	Detect networking attacks	No	No	No	Raspberry, SIEM server	Yes, CPU and Ram
[11]	Smart home	Host based IDS, ML model for anomaly detection	Detect networking attacks	Yes, blocking or redirecting intruder	No	Yes, it requires a third party manager	SDN controller with Openflow	No
[12]	Smart home, smart office	Anomaly detection LSTM model	Detect anomalies in MQTT protocol and in electricity measurements	No	No	No	Not defined	No
[13]	General	ML model for anomaly detection	Detect network scanning probing, DoS attack, known attacks based on IDS rules	No	No	No	Not defined	No
[14]	Smart home	3 ML models to identify normal behavior (profiling), identify malicious traffic and classify type of attack	Detect network based attacks (DoS, MIM, reconnaissance, replay) and multistage attacks	No	No	No	Not defined	No
[15]	General	2 ML models for IoT rule extraction and threat prediction	Detect network attacks	Yes, adapting the decision model	Yes, adapting the decision model	No	Server, devices	No
[16]	General (smart city)	Weight graph using device capabilities	Deploy IPsec service	No	Yes, adaptable to devices capabilities	No	Edge computing nodes	No
[17]	General	Random Geometric Graphs with IoT routing protocol (RPL)	Identify network attacks, particularly Sinkhole or MIM	No	Yes, adaptable to changes in topology	No	Central IDS agent, distributed IDS agents	Yes, communication overhead and energy consumption
[18]	General	IDS Signatures, Anomaly detection model	Detect network attacks	No	Yes, adaptable to mobility conditions	No	Odroid xu3 board	Yes, CPU and RAM
[19]	e-Health	Attack–defense model using evolutionary game theory, reinforcement learning, fuzzy logic	Detect network attacks	Yes, applying a selectable defense strategy	Yes, adaptable to type of attack	No	Not defined	No

(*continued on next page*)

of nodes: working, monitoring, legitimate, and attacker. Each of these nodes has specific functions that are simulated with Contiki in different IoT network scenarios, where the content and quantity of requests determine the existence of a DDoS attack.

A comparison between related works is presented in Table 1 according to the following factors: (i) proposal intended scenario, (ii) main technique(s) used by the proposal, (iii) main security functions provided, (iv) capacity to mitigate identified attacks, (v) automatic adaptation to changes, (vi) autonomous operation or required management, (vii) the main infrastructure required to work and (viii) evidence of some computational performance measures.

As indicated in Table 1, a bunch of related works define specific proposals for a smart home scenario, as this one represents a massive field of application that nowadays demands security functions. Regarding the core technique, some works are based on a type of IDS (e.g., rules or anomaly detection models), while others are more sophisticated using variances of neural networks, graph theory, game theory, fuzzy logic, artificial immune systems, or genetic algorithms. Also, most of the proposals offer detection capabilities for network attacks, but just a few of them offer some mitigation or contention. In terms of adaptation, most of the proposals are based on static conditions, and only a few ones are able to adjust their operation

Table 1 (*continued*).

Related work	Scenario of application	Core technique	Main security function	Mitigation of the attack	Adaptative to changes	Require additional management	Main infrastructure required	Computational performance measures
[20]	General	Dynamic IDS using Genetic Algorithm (GA)	Detect type of attack: Denial of services, Probe, Remote-to-User (R2L), User-to-Root (U2R)	Yes, blocking or redirecting traffic	Yes	No	SDN controller with Openflow	No
[21]	General	Deep learning	Detect type of attack: Denial of services, Probe, Remote-to-User (R2L), User-to-Root (U2R)	No	No	No	Distributed fog nodes	No
[22]	General	Artificial immune system with deep neural network model	Detect and classify malicious traffic	No	Yes, adaptable to new network traffic	Yes, it requires human manual confirmation	IoT nodes	No
[23]	General	Modeling IoT networks with working, monitoring and legitimate nodes	Detect and avoid DDoS attacks	Yes, refuting malicious requests	No	No	Working node	No
Our proposal	Smart home	Dynamic IDS rules	Detect network attacks	Yes, blocking malicious traffic	Yes, adaptable to identified threats	No	Raspberry board	Yes, CPU and RAM

to external factors like topology, mobility conditions, type of attack, network traffic, or devices capabilities. Also, most of the proposals offer an autonomous operation without requiring management labor aside from the initial deployment. In terms of infrastructure, there is a diversity between proposals that require the addition of new devices (Raspberry, Arduino, Odroid, etc.), the use of existing devices (endpoint nodes, controllers, etc.), and proposals that do not define such factor. Finally, very few proposals present experiments with computational performance measures.

From Table 1 it is possible to observe that the proposal described in the paper at hand is the only solution specialized in smart home scenarios, that has detection and mitigation capabilities, gets adapted to the existing threats in the scenario through dynamic management of rules, it is autonomous after deployment, may work on computationally-restricted devices and has been validated in terms of computational performance to demonstrate its convenience. All of these features included in our proposal are relevant in smart home scenarios, being the adaptability one of the most important ones in order to provide a tailored defense and increase the overall situational awareness.

3. COSMOS proposal

As mentioned before, unsolved challenges still appear in the fight against malicious actors in an effort to protect smart environments. Bearing such a challenge in mind, the paper at hand aims at proposing a a computationally efficient solution, offering an adaptive and dynamic rules management methodology that, in collaboration with a ML module, adapts itself to previously-unknown attacks and enforces mitigations with minimum latency. To achieve this target, we take advantage of the capabilities of our previous work, i.e., COSMOS [7].

Among several approaches for the protection of IoT scenarios, COSMOS (Collaborative, Seamless and Adaptive Sentinel for the Internet of Things)[1] can be seen as an innovative proposal that leverages the use of a plug-and-protect sentinel to protect smart environments from cyber threats. The main goal of such a proposal is to shield IoT devices using multiple defensive mechanisms that conform to a robust cyber defense strategy. COSMOS has been initially deployed in Raspberry Pi and is applicable in, but not limited to, smart home scenarios [24]. COSMOS is able to monitor the network traffic of the smart environment, analyzing the captured packets and inspecting them to search for potential threats. Moreover, COSMOS generates security events that are sent to a SIEM (Security Information and Event Management) server that processes them to identify malicious activities and anomalies [25]. A COSMOS sentinel may be deployed in every smart home appliance, while a common SIEM service could collect and analyze all security events to detect massive and distributed attacks. The analysis of such events allows the generation of cyber threat intelligence information to prevent new attacks, identify capacities of threat agents, and design an innovative countermeasures mechanism. A cybersecurity situational awareness could also be achieved with a massive deployment of COSMOS sentinels, which could be of interest for those CSIRTs (Computer Security Incident Response Teams) affiliated with a national ISP (Internet Service Provider).

COSMOS is an ecosystem of software tools that behaves holistically and detects attacks targeting smart networks. Specifically, COSMOS architecture is depicted in Fig. 1 and is composed of the following main components:

- **COSMOS sentinel**: The sentinel is in charge of monitoring the traffic generated within the network. By leveraging various security tools, it is able to provide defense-in-depth to the surrounding smart devices. In particular, the sentinel features four security rings that sequentially analyze potential malicious samples from different perspectives.

[1] https://webs.um.es/felixgm/projects/cosmos

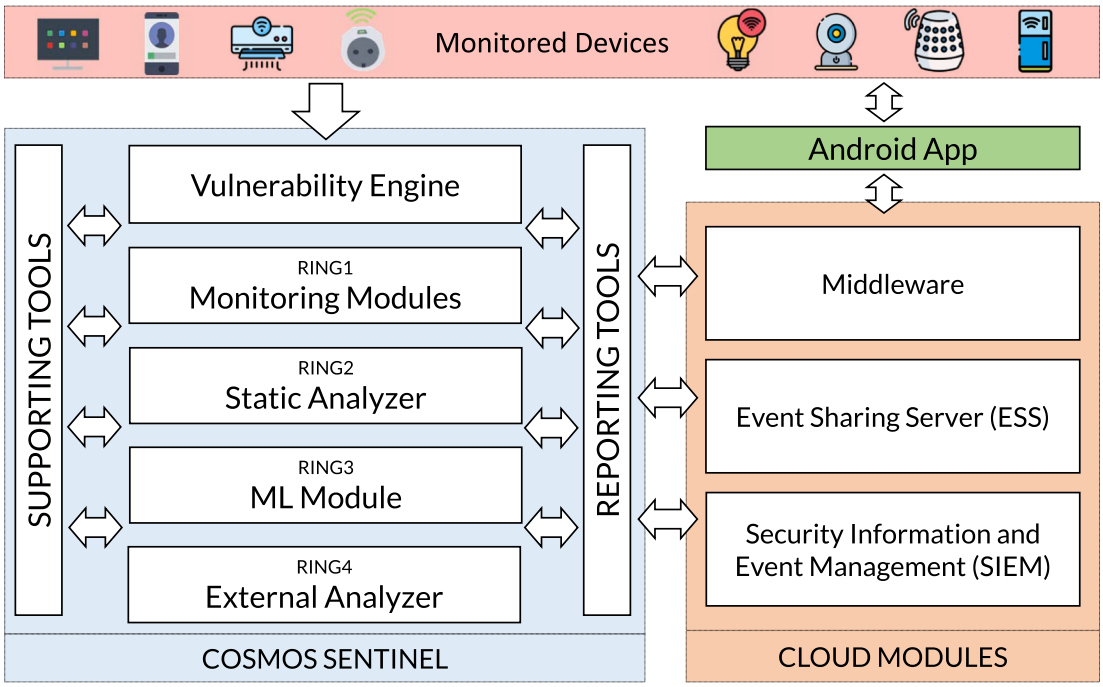


Fig. 1. COSMOS architecture.

- **Cloud modules**: These components represent the modules on the cloud exposing the services that the sentinel consumes during its operations. In particular, they manage the communication and correlation of security events with the deployed sentinels.
- **COSMOS App**: The sentinel communicates with a self-developed Android App through the Middleware. Such App allows end-users to control sentinels within the smart environment. Additionally, the App notifies the user about the current security situation of the smart devices.

All the previously-described modules support the cybersecurity functions of COSMOS and its communication capabilities. Nonetheless, the available COSMOS documentation indicates that the sentinel architecture can be further optimized from a computational viewpoint. For instance, COSMOS sentinel implements some functionalities through heavy processes that a home device may not support as they impact the speed, latency, and consumption of the resources of the host device. Furthermore, by adding the dynamic risk management capability, it would be possible to increase both security and computational features, using tailored security controls that shield the current risk status devices.

Thus, it is required to further optimize COSMOS in an effort to make it more computationally efficient, user-friendly and non-intrusive, so that it can be adopted and achieve its cybersecurity purposes in a better way. Also, COSMOS should achieve its security functions efficiently in order to allow its deployment in a low-powered and cheap device so most end-users can access it.

4. Proposed architecture

After a brief review on how the different COSMOS components work and communicate to perform the protection tasks, the main objective of this Section is to analyze and justify the made improvements and how they affect the overall performance and behavior of the framework.

It has to be stated that in the original version of COSMOS, all the functionalities were implemented in a single module, which is hard to scale and maintain over time. Therefore, this module was decomposed into several tiny components, one for each specific task, e.g., handling the static analyzer or OpenVAS events. In this way, we believe that the proposed framework features modularity, and it is easily extensible and simpler to maintain in the future.

Fig. 2 depicts how the system has been modified after the inclusion of new modules. At first glance, the major modifications have been conducted in the COSMOS sentinel component, where four extra modules have been added, namely, the Dynamic Rule Loader (DRL), the Vulnerability Detection System (VDS), the Event Processing Module (EPM), and the Mitigation Module (MM). In particular, these modules are in charge of performing the dynamic management of rules and conducting the mitigation process. Furthermore, another modification has been made to achieve an overall improvement of the performance. Specifically, the Scheduler (SCH) component has been added to schedule a *cron job* to download and update YARA and Suricata rule databases when the overall resource consumption of the framework is low. Besides, modifications have been implemented in 3 pre-existing components, i.e., the monitoring modules, the static analyzer, and the supporting tools.

Next, the implemented changes (either in the pre-existing COSMOS modules or in the new components) will be analyzed in a more detailed way, highlighting their operations and advantages.

4.1. Monitoring modules

The monitoring modules are responsible for intercepting the frames flowing within the monitored network and, consequently, analyzing them locally using powerful IDS rule-based engines. Indeed, they represent the first protection ring for smart devices. In the proposed architecture, two modules are deployed: the Ethernet monitor (i.e., Suricata) and the Wi-Fi monitor (i.e., Kismet).

Regarding Kismet[2] (i.e., the selected wireless IDS), no changes have been implemented apart from the adaptation needed to let the IDS correctly perform the improved COSMOS version.

Considering Suricata[3] (i.e., the proposed IDS) as standalone software, very few changes were made compared to the existing COSMOS

[2] https://www.kismetwireless.net/
[3] https://suricata-ids.org

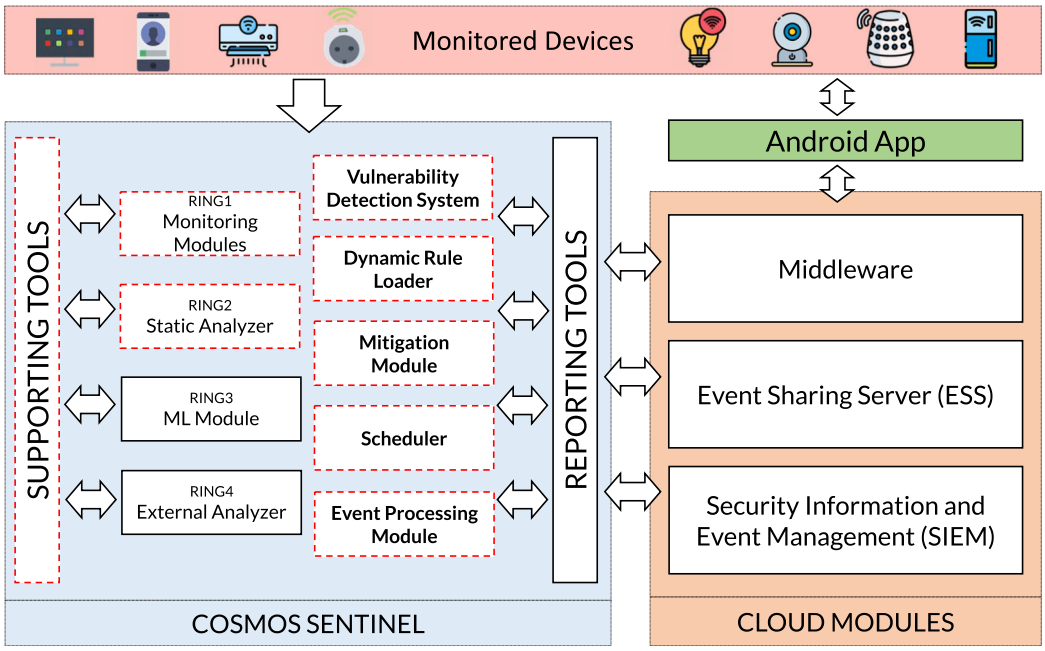


Fig. 2. Overview of modified COSMOS architecture, highlighting the new deployed modules (in dashed red boxes). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

framework. Among those, it is possible to highlight the modifications in the YAML configuration file to define the predefined rule set to use. More specifically, we changed the predefined YAML file[4] in order to add a new entry in the section `rule-files` as a new rule file named `local.rules`, where all dynamically deployed rules are going to be written.

Besides the above-mentioned changes, a new module dedicated to handling Suricata events in the Sentinel was developed, such as generate a new rule or update the current ruleset. This module downloads the new set of rules when the Scheduler triggers the update event (as reported in Section 4.4) and filters which subset of the whole local database of rules will be actively used at each iteration. In this sense, it is worth remarking that Suricata allows performing a hot reload of the rules, which are used without stopping the service. This feature is considered crucial for this case because the IDS must run uninterruptedly to assure the protection of the monitored devices. Thanks to such a non-blocking method of rules reloading, it is possible to provide a 100% protection time. The action itself of reloading the rules is achieved by using a utility of Suricata that allows it via command line after the selection of the correspondent rules to be deployed.

4.2. Static analyzer

The static analyzer is in charge of analyzing the malicious samples captured by the monitoring modules by using powerful static analysis rules searching for possible malware signatures. In the proposed framework, it represents the second protection ring against potential cyberthreats.

In the improved version of the framework, a custom module to handle YARA (i.e., the proposed static analyzer) requests was designed and deployed. In particular, this module is in charge of updating the rules repository and acts as a wrapper for YARA itself.[5] It is written in Python specifically for YARA and permits direct communication with it. More concretely, it allows to change the rules within the ruleset and compile those more efficiently. To achieve such a configuration, we designed a component following the Singleton Pattern.[6] In particular, this component allows one to handle the YARA module more efficiently and, additionally, it is easier to maintain in the long run because, since it is modular, it can be attached or detached from the core of the sentinel's modules without any substantial impact on the overall functioning. Besides, this module can be replaced at any time by any other tool that performs similar tasks. Further, it also allows keeping a simpler ruleset already compiled to achieve better performance, for example, when previously known analyzed file types will be analyzed, thus avoiding compiling rules unnecessarily.

4.3. Vulnerability Detection System (VDS)

In the proposed framework, OpenVAS[7] (i.e., the selected vulnerability scanner) has a crucial task, that is, to scan and detect vulnerabilities in the surrounding IoT devices or services within the smart network. Very few will oppose that the preventive detection of vulnerabilities is of primary importance in a dynamic context such as IoT smart appliance, where the security of the devices is often underestimated [26]. Since it is a standalone tool, no substantial changes have been made to the software itself, other than the predetermined ones to adapt the default installation to meet the network requirements. Nevertheless, a python module was designed in order to manage the connection with OpenVAS in a smooth fashion and to be able to handle events in programmatically (e.g., avoiding the user interface). Concretely, this module takes care of all OpenVAS connections, which are executed via Linux sockets since they are located on the same machine. To achieve proper communication with OpenVAS, we leverage Greenbone Vulnerability Management (GVM)[8] capabilities, which is recommended by the OpenVAS community. This protocol leverages its own wrapper in Python[9] to facilitate the entire communication pattern with OpenVas.

[4] https://suricata.readthedocs.io/en/suricata-4.1.3/configuration/suricata-yaml.html
[5] https://github.com/VirusTotal/yara-python
[6] https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Singleton.html
[7] https://www.openvas.org
[8] https://www.greenbone.net/en/vulnerability-management/
[9] https://pypi.org/project/python-gvm/

In detail, the principal tasks accomplished by this module are: (i) order to OpenVAS to start a network scan using parameters set by the user, the sys-admin or predetermined ones, and (ii) extract all the information regarding potential founded vulnerabilities.

The first of the actions mentioned above, i.e., *scanning the network*, can be started by two main events: (i) triggered by the scheduler, and (ii) fired by the user. Due to the changing and volatile nature of the IoT scenarios where the framework aims to be deployed, one could easily argue that performing scans with excessive frequency may represent a waste of resources. However, such a process cannot be done either with long delays between consecutive scans since new services o devices can be added to the network at any time. Therefore, these events try to cover this issue. On the one hand, if the user consciously adds new devices or services, he may inform this change via the Android app and, consequently, trigger a network scan to analyze the current status of the smart network. On the other hand, if no change has been notified by user in a given period, the Scheduler module will trigger a periodic scan to spot possible modifications in the network that the system is potentially unaware of.

The second action, i.e., *extract information about possible founded vulnerabilities*, involves the task that, when a vulnerability is detected, is in charge of extracting all available information about the security flaw. Using the GVM module in Python, it is indeed possible to extract an XML[10] with several necessary fields about the detected vulnerability, like, Common Vulnerabilities and Exposures (CVE)[11] information [27].

4.4. Scheduler (SCH)

The Scheduler serves a crucial role in the proposed architecture since it is designed to handle those tasks that need to be repeated in a particular time-lapse or over time, e.g., the vulnerability scanning and the update process of the IDS and static analyzer rules. This function is highly relevant toward the automation of the functioning of the COSMOS component modules since it avoids human intervention in the jobs launching through scripting or user interface commands. Thus, this module is in charge of orchestrating the functionalities within the proposed framework concerning the recurrence of tasks. To achieve such a goal, we adopted the Scheduler[12] module of Python, which features very versatile and powerful skills. By leveraging this module and a set of variable parameters, it is extremely easy and flexible to schedule the required tasks within the framework. This module was selected mainly because of its simplicity and its human-friendly syntax.

4.5. Supporting tools

These modules execute several actions when invoked by the other components of the COSMOS sentinel, helping the overall functioning of the framework.

Compared to the previous version of COSMOS, this component presents a considerable number of changes, especially in the sentinel core modules. In particular, the designed improvement was to remove the *busy waiting* for the files. Therefore, to address this shortcoming, a *watchdog* mechanism has been deployed. Specifically, the watchdog mechanism requests the system to be asynchronously notified when any change to the interested address of the file system appears. In this case, we used the Python module watchdog.[13] One can easily spot that this mechanism is more efficient and secure than the busy waitingbecause, in the latter case, it is mandatory to establish a waiting time that can result too high or too low. On the one hand, it can become a waste of resources and overload for the entire framework if it is too high.

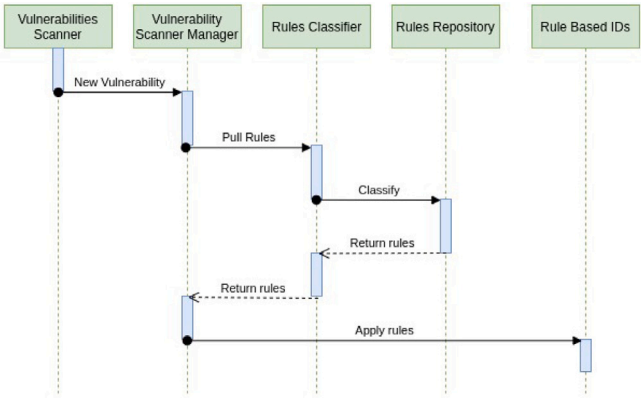


Fig. 3. Dynamic rules deployment within the proposed framework.

On the other hand, if it is too low, several files can be stacked in the queue, and, consequently, when the system notifies changes, there can be a huge queue to process. Thus, the proposed watchdog approach performs better given the requirements in which the framework is deployed.

4.6. Dynamic Rules Loader (DRL)

In order to achieve a dynamic risk management within the smart environment, an *ad-hoc* DRL was designed and deployed. The main idea of such a methodology is to charge the sentinel only with the detection rules that are strictly needed in a certain moment based on the surrounding environment, thus saving crucial resources and improving the overall functioning. Concretely, the proposed idea starts with the initialization of the VDS (i.e., OpenVAS), the IDS (i.e., Suricata), and all related software. After this initialization phase, OpenVAS scans the network, searching for potential vulnerabilities, while Suricata updates its local rules database and normalizes the ruleset to manipulate it more easily. Whenever OpenVAS spots a previously-unknown vulnerability, it extracts a metadata and information regarding such a security flaw (e.g., CVE attributes). If such information does not belong to a previously detected threat, that is, it is not in the same category, it is passed to a classifier module, where it is matched against one of the rules belonging to the ruleset of Suricata. Subsequently, the corresponding rule is dynamically inserted into Suricata in a non-blocking fashion thanks to its internal mechanism. For an easy reference, Fig. 3 illustrates the dynamic rules deployment implemented in the proposed framework and can be resumed as follows:

1. The vulnerability scanner detects a new vulnerability in the devices of the network.
2. It informs about this change to the Vulnerability Scanner Manager (VSM).
3. The VSM triggers the classification and deployment steps.
4. The classifier searches in the Rules Repository, available rules that may apply.
5. The classifier selects only those who detect possible attacks over the new vulnerability detected.
6. The classifier returns the rule or rules corresponding to the new vulnerability.
7. The VSM applies the new rule-set according to the surrounding scenario into the rule-based IDS.

The pseudocode of the proposed methodology can be seen in Algorithm 1, where the main flow of actions is executed, while Algorithms 2 and 3 complement the main one. In detail, the second algorithm (Algorithm 2) shows how the rules in the local repository of rules can

[10] https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction
[11] https://cve.mitre.org/
[12] https://pypi.org/project/schedule/
[13] https://python-watchdog.readthedocs.io/en/v0.10.2/

Algorithm 1 Dynamic Rules Loader (DRL)

```
1: procedure GENERAL_MECHANISM()                          ▷ General overview of dynamic rules loader behavior
2:     openVAS ← initialize()                             ▷ Initialize OpenVAS
3:     openvas.start_scanning()
4:     suricata_rules ← initialize()                      ▷ Initialize Suricata
5:     suricata_rules.download_and_update_rules()
6:     return_map ← rules_normalizer()
7:     active_threats ← ∅
8:     while scheduled or manually triggered scan do
9:         if (vuln ← openvas.get_vuln()) is ∅ and vuln is not in active_threats then
10:            active_threats += vuln
11:            suricata_rules.classify_vuln_rule(vuln, suricata_rules.actives)
```

Algorithm 2 Rules normalization.

```
1: procedure RULES_NORMALIZER()                           ▷ Standardize the names of the rules
2:     rules ← all_files_from_rules_dir
3:     for rule in rules do
4:         name ← family(rule.name)
5:         return_map[name] ← rule.absolute_path          ▷ "family(rule.name)" get the family of attacks for a rule
       return return_map
```

Algorithm 3 Classify new Vulnerabilities against existing rules.

```
1: procedure CLASSIFY_VULN_RULE(vuln, active_rules, all_rules)   ▷ New vulnerability to classify
2:     for all rule in all_rules do
3:         family ← vuln.family                           ▷ vuln is obtained from OpenVAS scan, and Family is an attribute
4:         for all key in family do
5:             if key in rule then                        ▷ Match vulnerability type against rule
6:                 active_rules[rule] ← all_rules[rule]
7:         return_map[name] ← rule.absolute_path
```

be normalized, and the third one (Algorithm 3) illustrates a simple way to classify those according to the family of threat that it detects. The normalization process is necessary because, in a real deployment, there should be several sources of rules, and, possibly, not all of them possess the same structure for naming. Consequently, to handle this situation correctly, a possible way is to represent those in a standard format in memory (i.e., with a unique way to identify them). Also, the classification system is essential since it is the one in charge of determining which rule matches with which vulnerability and discarding useless rules for a given scenario, thus achieving better performance.

The same process of classification and updating rules of the IDS (i.e., Suricata) can be applied to the static analyzer (i.e., YARA) with the same logic but in a different way since YARA is more dedicated to file types in contrast to network traffic. In YARA, rules are updated with the help of the Scheduler Module, and whenever a file is extracted by Suricata, it is sent to YARA in order to be analyzed. In addition, Suricata also creates a metadata file used to determine the type of the file, and, according to it, a determined ruleset of YARA is loaded, compiled, and used to analyze that file.

4.7. Event Processing Module (EPM)

EPM processes all events generated by the IDS and filters only those of the "alert" type, considered of interest in our context. Such a component also features high significance within the proposed framework since it is able to highlight only the critical events happening in the protected network. In fact, due to the hyper-connectivity of the smart appliances, the number of events generated by the intelligent devices is huge, and possibly not all of them are relevant from a security viewpoint. So, the EPM is in charge of filtering those events to make COSMOS security tools process the alerts.

Such characteristic was deployed in the Ethernet IDS by leveraging the `eve.json`[14] file from Suricata, in which it writes all logs it process, e.g., warnings, alerts, types of traffic, and more relevant information. More in detail, all the logs in the `eve.json` file have a determined structure[15] that gives one several information about the current processed events. These events are generated when a network packet triggers the corresponding match against any of the rules loaded in Suricata's kernel. At this point, Suricata is in charge of writing the corresponding information to a file[16] through an entry containing relevant details about the packet. Later, using a dedicated pipeline, this entry is directed to Suricata, and in this place, the event is processed, extracting only the interesting events. These entries are json-like[17] objects. In this work, a Python module was used named simple-json[18] for easy processing of the events.

Similarly, the same reasoning may be applied to the wireless IDS of the proposed architecture (i.e., Kismet) by filtering the events produced by the real-time logging.

4.8. Mitigation module (MM)

The mitigation module implemented in this framework features high simplicity, although it is very powerful. Indeed, it is able to block potentially malicious traffic flowing within the network by applying tailored filtering rules. To this extent, it has to be remarked that the selected IDS (i.e., Suricata) can also perform as an Intrusion Prevention

[14] `/var/log/suricata/eve.json`
[15] https://suricata.readthedocs.io/en/suricata-4.1.4/output/eve/eve-json-output.html#alerts
[16] /var/log/suricata/eve.log
[17] https://www.json.org/json-es.html
[18] https://github.com/simplejson/simplejson

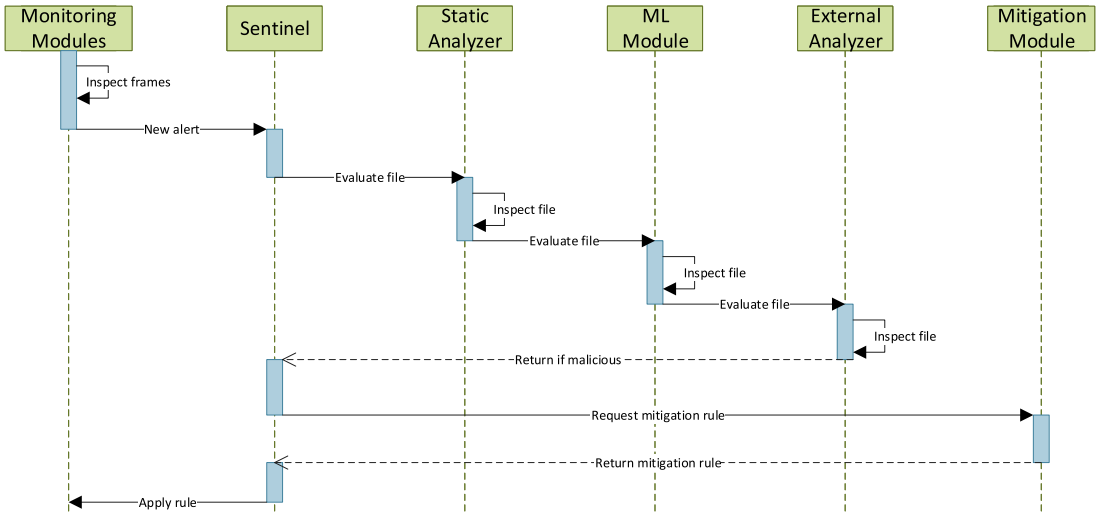


Fig. 4. Intrusion detection and mitigation flow.

Table 2
Modules used in the COSMOS deployment for the smart home scenario..

Module	Main Tool	Purpose	Deployed on
Forensic	Radare[a]	Analyze samples captured by the monitoring module	Sentinel
Network IDS	Suricata	Detect intrusions and malicious traffic in the smart home	Sentinel
Wireless IDS	Kismet	Sniff wireless traffic so it can be analyzed	Sentinel
VDS	OpenVAS	Detect vulnerabilities in the smart home devices	Sentinel
Static analyzer	YARA[b]	Detect malware based on signatures	Sentinel
Exchange	ActiveMQ[c]	Communicate sentinel with server	Sentinel - Server
Exchange	SpringBoot[d]	Communicate sentinel with mobile app	Server
Mobile App	Google Android SDK[e]	Support the mobile app	User smartphone

[a] https://github.com/radareorg/radare2.
[b] https://virustotal.github.io/yara/.
[c] http://activemq.apache.org/.
[d] https://spring.io/projects/spring-boot.
[e] https://developer.android.com/studio.

System (IPS) and control the network traffic in an effort to protect the smart devices from cyberthreats. By leveraging this capability, Suricata can be efficiently employed to mitigate the effect of the malicious frames within the network. More specifically, due to its ample community support and vast industry usage, Suricata features an extensive rules set that is able to cover most of the modern attacks (excluding the zero-day exploits [28]). Particularly, when previously unknown attacks are detected by, say, the correlation capabilities of the SIEM [25] or the employment of anomaly-based IDSs [29], a new rule that drops such suspicious traffic is generated and dynamically enforced in the current ruleset. In this way, the malicious traffic is dropped on successive attempts to invade the system.

Nonetheless, the dynamic rules management to mitigate attacks is not limited to block files traveling through the network. It goes beyond that, since a new functionality has been developed to achieve more flexibility during the mitigation process. Concretely, whenever a new alert is generated by the IDS, a dedicated pipeline attached to the sentinel core is activated thanks to the EPM. In particular, the sentinel is continuously listening to such pipeline and, when any event is noticed, the sentinel immediately processes it using the security rings and generates the corresponding rule by setting the *action* field to *drop*. From this event, the *src_ip*(source IP) and *dest_ip*(destination IP) are also extracted in order to generate a rule that prohibits the flow of that specific traffic within the protected network. Afterward, an alert is generated and sent to the user application for logging and further analysis.

For easy reference, a snapshot of the mitigation process flow can be seen in Fig. 4. Specifically, the monitoring modules (both ethernet and wireless) are in charge of constantly inspecting the frames flowing within the network, thus representing the first protection ring for the smart devices. Whenever a suspicious file is detected, an alert is generated and passed to the sentinel. Consequently, the sentinel passes the sample to the other detection rings in sequential order. First, the static analyzer (i.e., YARA) inspects the file using the enabled rules searching for malware samples. In case the static analysis declares the file as benign, it is delivered to the ML module that, in turn, is in charge of classifying the sample as goodware or malware. If it is the first case, the file is passed to the external analyzer (i.e., VirusTotal), which leverages several malware databases to classify the sample. Whenever one of the mentioned rings marks the sample as malware, an alert is generated and returned to the sentinel. In this case, the sentinel requests a mitigation rule to the MM to block the suspicious traffic. Such a rule is then applied directly to the monitoring modules in an effort to avoid the invasion of the same threat at any time.

5. Experiments

As mentioned in the previous sections, the COSMOS architecture improvements aim to enhance the overall performance from both security and performance perspectives. To prove the capabilities of the meliorated framework and effectively argue on the performance, several experiments are described throughout this Section. In particular, the settings are presented in Section 5.1, and the results are analyzed in Section 5.2.

Table 3
Metrics used in the experiments.

Metric	Description
CPU Consumption	CPU consumption among all cores
RAM Consumption	RAM usage for a given experiment
Rules	Amount of actives rules in every moment
Dropped packets	Amount of packets dropped
Kernel captures	Amount of packets captured and processed

5.1. Setup of the experiments

Experiments were conducted using VirtualBox version 6.1.6 r137129[19] and Ubuntu 16.04 as guest image. The Ubuntu VM was equipped with 6 GB of RAM and 4 CPU cores to host the improved version of the COSMOS framework. The host machine was a Laptop HP Probook G5 with 8th Generation Intel Core i5 (with four physical cores, divided into eight logical cores). Table 2 contains the main tools used in the deployment of COSMOS for the designed smart home scenario. Additionally, the metrics defined in Table 3 were used in the set of experiments.

Two scenarios were prepared, so it could be possible to argue on the improvements compared to the previous version of the COSMOS framework.

- **Scenario 1**: The first scenario consists in performing the experiments loading COSMOS with an entire ruleset.
- **Scenario 2**: The second scenario consists in replicating the experiments of the scenario 1 but enabling the dynamic rules mechanism, so it can be possible to measure how the designed strategy impacts COSMOS operations and performance. The only rule loaded in COSMOS from the start was the file extraction rule, as it is needed to extract the captured files and guarantee the correct intrusion detection behavior.

In both scenarios, the steps that described the experiments are presented next:

1. Prepare a smart home scenario.
2. Deploy COSMOS to protect the smart home.
3. Startup COSMOS modules.
4. Expose vulnerabilities in the smart home scenario.
5. Measure the behavior of COSMOS through the defined metrics.

In this sense, it has to be noted that the experiments are not synchronized, thus the steps may not be exactly lined up.

In particular, the vulnerabilities exposed in step 4 of the experiments can be seen as common security flaws belonging to a typical smart home scenario. To this extent, those vulnerabilities are detailed next:

- A downstream traffic containing malicious files.
- An Apache HTTP server (2.4.18) exposing the 80 port which serves the previously mentioned malicious files.
- An Apache ActiveMQ Web Console with default login credentials.
- A SSL/TLS protocol utilization without HTTP Strict Transport Security (HSTS).
- A MQTT Broker that does not require authentication.

In particular, the vulnerabilities exposed during the experiments can be seen as common security flaws belonging to a typical smart home scenario.

Malicious files mentioned in step 1 were obtained from Drebin[20], a dataset with a massive collection of malicious files. These malicious files were used to test COSMOS capacity to monitor the smart home network, detect vulnerabilities in the surroundings and mitigate the imminent risk.

For both scenarios, the 55,718 rules available in Emerging Threats[21] were used as COSMOS ruleset. Emerging Threats ruleset was selected because it focuses on detecting and blocking advanced threats, is daily updated, and is available in Suricata format. The Emerging Threats rules also cover more than 40 different categories of network behaviors, malware associated with command and control functionalities, DoS attacks, botnets, informational events, exploits, vulnerabilities, exploit kit activity, and more.

Finally, each test lasted for 120-200 s, however, in the future, we plan to extend their duration to better grasp how COSMOS works in a long period of time and better test its stability.

5.2. Results

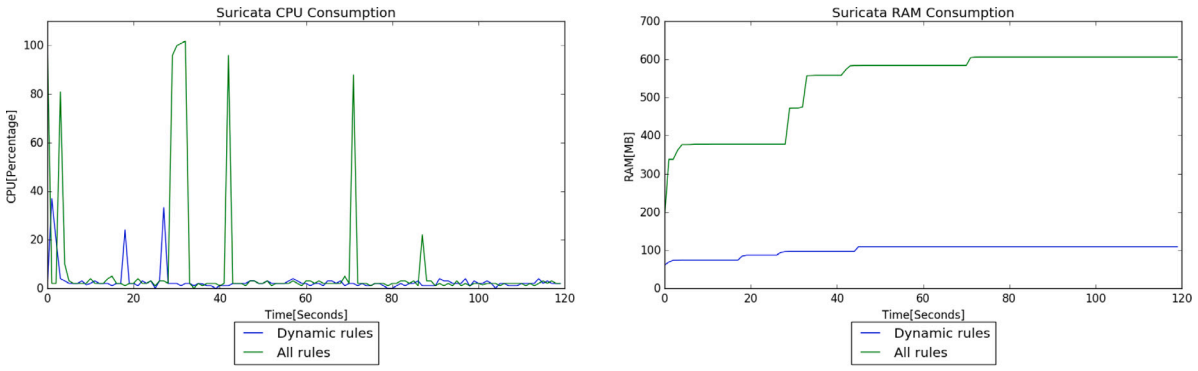
In the execution of the experiments under scenario 1, COSMOS was able to discover the exposed vulnerabilities thanks to the work of the VDS described in Section 4.3 and then used the collected vulnerabilities information to create a new rule to mitigate the security risk identified in the downstream of the malicious Drebin files, increasing in this way the number of COSMOS rules to 55,719.

On the other hand, in the execution of the experiments under scenario 2, COSMOS started up with the one rule. After vulnerabilities were exposed, COSMOS loaded 552 rules that matched those vulnerabilities extracted from the Emerging Threats ruleset. COSMOS also loaded one additional rule to mitigate the identified risk downstream of the malicious Drebin files. Besides, the adaptation capacity of COSMOS was seen in the selection of the rules that exactly protect the scenario from the vulnerabilities, avoiding under-protecting or over-protecting the smart home scenario. From this adaptability, COSMOS was also able to process more packets, which increased its detection capability, providing cyberprotection to the intelligent devices in the smart appliance at any time.

Fig. 5(a) shows the CPU consumption in both scenarios 1 and 2 regarding the selected IDS (i.e., Suricata). More specifically, in scenario 1 (i.e., the IDS using the entire ruleset), Fig. 5(a) depicts several CPU peaks close to 100%. Instead, in scenario 2, it never passes over 40% of CPU utilization, representing a great overall improvement. It has to be remarked that the observed peaks in scenario 1 correspond to the moments when vulnerabilities are discovered and mitigated since it implies a matching of a set of rules from the full ruleset. Nevertheless, in scenario 2, peaks are considerably lower compared to scenario 1. Such an enhancement is due to the dynamic rules management charging the IDS with the rules effectively needed to cover the detected vulnerabilities.

Besides, Fig. 5(b) depicts the RAM usage in both scenarios 1 and 2 regarding the selected IDS (i.e., Suricata). In particular, the RAM usage in scenario 1 overpasses the 600 MB of utilization since the entire ruleset is charged in order to face the threats within the smart network. Nonetheless, by leveraging the dynamic rule management, the RAM usage decreases to 100MB at its peak, representing an improvement of 83%. One could easily say that, as expected, RAM saving can be seen as the major enhancement. More specifically, it shows that the original version of COSMOS IDS starts with relatively low consumption at the bootstrap, but it is skyrocketed when all rules are loaded in the core and gets even bigger when it starts to analyze the network traffic. In contrast, the proposed IDS with dynamic rule management starts smoothly with a small increment in the RAM usage, and then, when new rules are added, it increments slightly and stabilizes around 100 MB. It is worth analyzing the reason why the RAM usage is higher with more rules enabled. Concretely, for all rules, Suricata needs to

[19] https://www.virtualbox.org/
[20] https://www.sec.cs.tu-bs.de/~danarp/drebin/download.html
[21] https://rules.emergingthreats.net/open/suricata-5.0/



(a) IDS CPU consumption.

(b) IDS RAM consumption.

Fig. 5. Resources consumption of the adaptive rules COSMOS deployment.

keep an internal representation of those in memory, thus the more rules enabled, the more space will occupy in the RAM.

The outcomes on the IDS resource consumption experiments show an overall decrease in both CPU and RAM usage thanks to the dynamic rules management. Such an improvement has to be considered complementary to the experimental results presented in [7], where authors tested the previous version of the COSMOS framework as both intrusion and malware detector. Generally, the decomposition of the COSMOS original single-module functionalities into several smaller components also leads to a global advancement, especially looking at the modularity, extensibility, and maintainability of the framework, without impacting the performance. Furthermore, from Fig. 5(a) and Fig. 5(b), it is possible to appreciate the reduced time lapse between the detection of the threats (i.e., spotted security flaws in the network) and the generation of the mitigation rule to cover such threats. More specifically, whenever the COSMOS framework discovers a potential threat by means of the detection performed by the VDS or the security rings, the sentinel generates the corresponding rule in a timely fashion. The entire process (i.e., detection plus mitigation) matches with the peaks in both CPU and RAM usage.

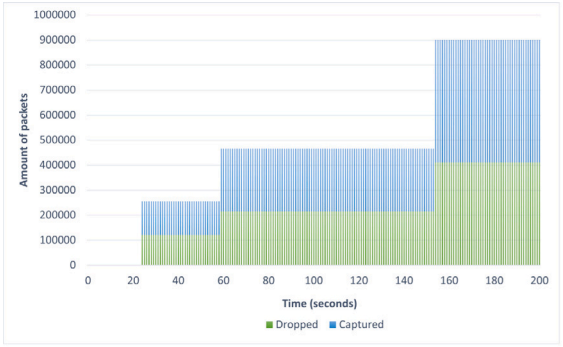
Another important perspective can be captured by evaluating the kernel of Suricata, using the file /var/log/suricata/stat.log, where Suricata saves all stats about the kernel, and the tool Suri-Stats,[22] which is in charge of analyzing it. Specifically, we were able to perform an analysis of the behavior of a critical parameter of the IDS, that is, the amount of processed and dropped packets. According to 4.1, Suricata is not always able to process all incoming packetsSome of them are dropped or just not analyzed because, to get the best performance, Suricata needs to run in "workers" mode. This effectively means that there are multiple threads, each running a full packet pipeline and receiving packets from the capture method. Thus, we rely on the capture method to distribute the packets over the various threads. One critical aspect of this is that Suricata needs to get both sides of a flow in the same thread, in the correct order. The problem here is that by having both sides of the traffic in different queues, the processing of packets becomes unpredictable. Timing differences on the NIC (Network Interface Controller), the driver, the kernel, and Suricata will lead to a high chance of packets coming in at a different order than on the wire. This is specifically about a mismatch between the two traffic directions. One could argue that this feature affects the overall quality of detection since the more packets are analyzed, the more secure the environment may be. Thus, this is a critical parameter to optimize as well. Since Suricata generally has fewer rules to apply to each packet with the proposed dynamic rule management, it is able to analyze those faster, therefore no need to drop them.

In Fig. 6(a), Suricata behavior in scenario 1 is shown, with all rules loaded, and it drops around 400,000 packets and is only able to deliver and analyze around 500,000 packets. Nevertheless, with the dynamic rule management enabled, in scenario 2, it achieved 750,000 packets analyzed and only dropped around 300,000 (Fig. 6(b)). It is important here to highlight that the improvement in the dropped packets is around 25%. On the other side, regarding the processed packets, the improvement was 33% which is a very great increase. This can be translated into detecting more malicious traffic since analyzing more packets increases the possibility of spotting malevolent actions flowing within the network. Also here, the enhancement of the performance implies an increment of security.

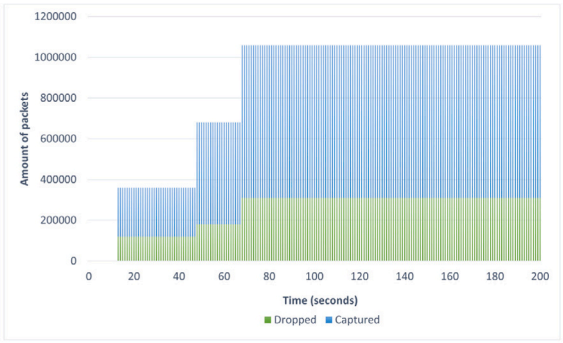
Experimental results demonstrate that the proposed version of COSMOS performs an accurate and fast detection across the different protection layers with a noticeable improvement. More specifically, the best improvement was obtained in RAM consumption of the IDS (i.e., Suricata) with a reduction of 500 MB on average. In addition, thanks to our modification in the mitigation strategy, we achieved the second-best improvement in the rate of IDS dropped and analyzed packets, reaching a reduction of 100,000 fewer dropped packets and an increase of 250,000 more analyzed packets during the experiments. Also, we obtained a better usage of CPU in the IDS, with lower peaks of consumption because the improved IDS runs with significantly less rule with which compare the incoming traffic. Such a trick increases the performance w.r.t. resources consumption and the quality of detection because it is able to process more packets compared to the previous version. As a side effect, the sentinel core modules were using more RAM compared to the previous version, due mainly to the more processing power needed to retain in memory the full state of the surrounding scenarios. In particular, when some changes are made, the sentinel core module analyzes the differences and recalculates the optimal rules to apply for the new state. Nevertheless, such an increase is negligible since it caused only 75 MB of difference, and compared with the 500 MB of saved resource of the IDS, one can conclude that it is worth it. Thanks to the novel dynamic rule management, the resource consumption is quite low compared to the previous version, where the entire ruleset was applied to maintain accurate performance in detection.

It is important to remark that experiments were run many consecutive times to avoid possible out-layers and interference. However, the results were steady, with little or negligible variations. In the executions of scenario 2, COSMOS was stable, and it was able to detect all vulnerabilities, adapting the loaded rule set to the necessary. Experiments results show that CPU consumption, RAM consumption, and dropped packets for scenario 2 were lower than in scenario 1. Also, the COSMOS capacity to process traffic was improved, as it analyzed a considerably higher amount of packets. These improvements suggest that COSMOS may be deployed in the future in computational-constrained hardware, still guaranteeing an adequate level of protection of a smart scenario.

[22] https://github.com/regit/suri-stats



(a) IDS captured and dropped packets with all rules enabled.



(b) IDS captured and dropped packets with dynamic rules management enabled.

Fig. 6. Comparison regarding captured and dropped packets with normal and dynamic rules management.

Finally, COSMOS implementation could still be improved to be more efficient computationally through the optimization of the code used, for instance, in the rule representation.

6. Conclusions and future work

The IoT paradigm of "everything connected" has become more present in our daily life. Consequently, smart objects may be located in several daily-used appliances, exchanging massive amounts of information and providing innovative services to final users. Unfortunately, this phenomenon implies that cyber-threats targeting IoT devices are getting more and more advanced and frequent. Thus, several security issues have raised around the IoT ecosystem, requiring innovative solutions to address that challenges.

In the paper at hand, we presented a new and improved version of COSMOS [7], a novel framework to protect IoT environments against cyber-threats, making it adaptive to the risk scenario without losing its capability to cyberprotect IoT devices. Such improvement was achieved by making COSMOS aware of the current vulnerabilities and exposures in the surrounding smart network and dynamically adapting its ruleset to the circumstances. The adaptive strategy also allowed an optimization in the COSMOS resource consumption.

To demonstrate the feasibility and the produced improvement of the proposed framework, exhaustive experiments have been conducted. During the above-mentioned tests, the sentinel and the IDS have been heavily stressed with several malicious frames, reproducing two main attack scenarios to better highlight the improvements of the framework. Experimental results demonstrate that the COSMOS architecture's developed changes generate substantial improvements from both security and performance viewpoints.

As future works, we plan to deploy COSMOS in a programmable home router to validate its operation under realistic and domestic computational-constrained hardware. Techniques to correlate the identified vulnerabilities with the available rules set are also oncoming research. Finally, we plan to develop more experiments that include more advanced attacks to determine the COSMOS capacity to protect vulnerable IoT environments.

CRediT authorship contribution statement

Pantaleone Nespoli: Conceptualization, Methodology, Software, Validation, Investigation, Data curation, Writing - original draft, Writing - review & editing. **Daniel Díaz-López:** Software, Investigation, Data curation, Writing - original draft. **Félix Gómez Mármol:** Conceptualization, Resources, Writing - review & editing, Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by an FPU predoctoral contract granted by the University of Murcia, Spain, and by a Ramón y Cajal research contract (RYC-2015-18210) granted by the MINECO (Spain) and co-funded by the European Social Fund.

References

[1] Khan MA, Salah K. IoT security: Review, blockchain solutions, and open challenges. Future Gener Comput Syst 2018;82:395–411. http://dx.doi.org/10.1016/j.future.2017.11.022.

[2] Ray AK, Bagwari A. IoT based Smart home: Security Aspects and security architecture. In: 2020 IEEE 9th international conference on communication systems and network technologies. IEEE; 2020, p. 218–22. http://dx.doi.org/10.1109/CSNT48778.2020.9115737.

[3] Gómez Mármol F, Gil Pérez M, Martínez Pérez G. I don't trust ICT: Research challenges in cyber security. In: 10th IFIP WG 11.11 international conference on trust management. IFIP AICT, vol. 473, Darmstadt, Germany; 2016, p. 129–36. http://dx.doi.org/10.1007/978-3-319-41354-9_9.

[4] Nespoli P, Papamartzivanos D, Marmol FG, Kambourakis G. Optimal countermeasures selection against cyber attacks: A comprehensive survey on reaction frameworks. IEEE Commun Surv Tutor 2018;20(2):1361–96. http://dx.doi.org/10.1109/COMST.2017.2781126.

[5] Khan TS, Khan NU, Juneio HF. Smart city paradigm: Importance, characteristics, and implicatios. In: 2020 advances in science and engineering technology international conferences. IEEE; p. 1–6. http://dx.doi.org/10.1109/ASET48392.2020.9118352.

[6] Alalade E. Intrusion detection system in smart home network using artificial immune system and extreme learning machine [Ph.D. thesis], University of Cincinnati; 2020.

[7] Nespoli P, Useche Pelaez D, Díaz-López D, Gómez Mármol F. COSMOS: Collaborative, seamless and adaptive sentinel for the internet of things. Sensors 2019;19(7):1492. http://dx.doi.org/10.3390/s19071492.

[8] Ammar M, Russello G, Crispo B. Internet of Things: A survey on the security of IoT frameworks. J Inf Secur Appl 2018;38:8–27. http://dx.doi.org/10.1016/j.jisa.2017.11.002.

[9] Díaz-López D, Blanco Uribe M, Santiago Cely C, Tarquino Murgueitio D, Garcia Garcia E, Nespoli P, et al. Developing secure IoT services: A security-oriented review of IoT platforms. Symmetry 2018;10(12):669. http://dx.doi.org/10.3390/sym10120669.

[10] Sforzin A, Mármol FG, Conti M, Bohli J. RPiDS: Raspberry Pi IDS — A fruitful intrusion detection system for IoT. In: 2016 intl IEEE conferences on ubiquitous intelligence computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress. 2016, p. 440–8. http://dx.doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0080.

[11] Nobakht M, Sivaraman V, Boreli R. A host-based intrusion detection and mitigation framework for smart home IoT using openflow. In: 2016 11th international conference on availability, reliability and security. 2016, p. 147–56. http://dx.doi.org/10.1109/ARES.2016.64.

[12] Vakakis N, Nikolis O, Ioannidis D, Votis K, Tzovaras D. Cybersecurity in SMEs: The smart-home/office use case. In: 2019 IEEE 24th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). 2019, p. 1–7. http://dx.doi.org/10.1109/CAMAD.2019.8858471.

[13] Anthi E, Williams L, Burnap P. Pulse: an adaptive intrusion detection for the internet of things. In: Proceedings of living in the internet of things: cybersecurity of the IoT. London, UK: IET; 2018, p. 110–9. http://dx.doi.org/10.1049/cp.2018.0035.

[14] Anthi E, Williams L, Slowinska M, Theodorakopoulos G, Burnap P. A supervised intrusion detection system for smart home IoT devices. IEEE Internet Things J 2019;PP:1. http://dx.doi.org/10.1109/JIOT.2019.2926365.

[15] Domb M, Bonchek-Dokow E, Leshem G. Lightweight adaptive Random-Forest for IoT rule generation and execution. J Inf Secur Appl 2017;34:218–24. http://dx.doi.org/10.1016/j.jisa.2017.03.001.

[16] Godquin T, Barbier M, Gaber C, Grimault J-L, Le Bars J-M. Applied graph theory to security: A qualitative placement of security solutions within IoT networks. J Inf Secur Appl 2020;55:102640. http://dx.doi.org/10.1016/j.jisa.2020.102640.

[17] Qurashi MA, Angelopoulos CM, Katos V. An architecture for resilient intrusion detection in ad-hoc networks. J Inf Secur Appl 2020;53:102530. http://dx.doi.org/10.1016/j.jisa.2020.102530.

[18] Midi D, Rullo A, Mudgerikar A, Bertino E. Kalis — A system for knowledge-driven adaptable intrusion detection for the internet of things. In: 2017 IEEE 37th international conference on distributed computing systems. 2017, p. 656–66. http://dx.doi.org/10.1109/ICDCS.2017.104.

[19] Boudko S, Abie H. Adaptive cybersecurity framework for healthcare internet of things. In: 2019 13th international symposium on medical information and communication technology. 2019, p. 1–6. http://dx.doi.org/10.1109/ISMICT.2019.8743905.

[20] Mansour A, Azab M, Rizk MRM, Abdelazim M. Biologically-inspired SDN-based intrusion detection and prevention mechanism for heterogeneous IoT networks. In: 2018 IEEE 9th annual information technology, electronics and mobile communication conference. 2018, p. 1120–5. http://dx.doi.org/10.1109/IEMCON.2018.8614759.

[21] Diro AA, Chilamkurti N. Distributed attack detection scheme using deep learning approach for internet of things. Future Gener Comput Syst 2018;82:761–8. http://dx.doi.org/10.1016/j.future.2017.08.043.

[22] Brown J, Anwar M. Blacksite: human-in-the-loop artificial immune system for intrusion detection in internet of things. Human-Intell Syst Integr 2021;1–13. http://dx.doi.org/10.1007/s42454-020-00017-9.

[23] Zhang C, Green R. Communication security in internet of thing: preventive measure and avoid DDoS attack over IoT network. In: Proceedings of the 18th symposium on communications & networking. 2015, p. 8–15. http://dx.doi.org/10.5555/2872550.2872552.

[24] Pelaez D, Díaz-López D, Nespoli P, Gomez Mármol F. TRIS: a Three-Rings IoT Sentinel to protect against cyber-threats. 2018, http://dx.doi.org/10.1109/IoTSMS.2018.8554432.

[25] Díaz-López D, Blanco Uribe M, Santiago Cely C, Vega Torres A, Moreno Guataquira N, Moron Castro S, et al. Shielding IoT against cyber-attacks: An event-based approach using SIEM. Wirel Commun Mob Comput 2018;2018. http://dx.doi.org/10.1155/2018/3029638.

[26] Atlam HF, Wills GB. IoT security, privacy, safety and ethics. In: Farsi M, Daneshkhah A, Hosseinian-Far A, Jahankhani H, editors. Digital twin technologies and smart cities. Cham: Springer International Publishing; 2020, p. 123–49. http://dx.doi.org/10.1007/978-3-030-18732-3_8.

[27] Blinowski GJ, Piotrowski P. CVE based classification of vulnerable IoT systems. In: International conference on dependability and complex systems. Springer; 2020, p. 82–93. http://dx.doi.org/10.1007/978-3-030-48256-5_9.

[28] Collier B, Clayton R, Hutchings A, Thomas DR. Cybercrime is (often) boring: maintaining the infrastructure of cybercrime economies. University of Edinburgh; 2020.

[29] Fernández Maimó L, Huertas Celdrán A, Gil Pérez M, García Clemente FJ, Martínez Pérez G. Dynamic management of a deep learning-based anomaly detection system for 5G networks. J Ambient Intell Hum Comput 2019;10(8):3083–97. http://dx.doi.org/10.1007/s12652-018-0813-4.