Kalis2.0 - a SECaaS-based Context-aware Self-adaptive Intrusion Detection System for the IoT

Antonino Rullo, Daniele Midi, Anand Mudjerikar, Elisa Bertino

***Abstract*—The wide variety of application domains makes the Internet of Things (IoT) quite unique among other types of computer networks: IoT networks can be made of devices of different types, i.e., characterized by different hardware, functionalities, computing capabilities, and also network topology and communication protocols may drastically change from one IoT application to another. Such a heterogeneity requires ad-hoc security solutions, as security techniques that are effective in one IoT context may not be so in another context. Furthermore, IoT networks are ever-evolving by their very nature as smart devices can be easily added or removed. These factors call for the design of security tools capable of adapting themselves to the specific IoT instance, but also to the continuous network changes. In this paper we propose a context-aware, Security-as-a-Service based approach for intrusion detection whereby an IDS *(i)* autonomously collects information about the monitored system, *(ii)* chooses the best detection strategy accordingly, and *(iii)* modifies the detection strategy as the network evolves over time. This comprehensive approach to intrusion detection is an attempt to face the heterogeneity which characterizes the IoT in all its aspects, making it possible the design of a security tool able to be self-adaptive and context-aware, that is, effective in different and evolving IoT scenarios with little or no human intervention.**

***Index Terms*—Internet of Things; Intrusion Detection System; IDS; context-awareness; Security-as-a-Service; SECaaS; network features; device features; software architecture.**

I. Introduction

The security issues of the Internet of Things (IoT) are caused by a number of factors. First, since security requires investments, for business reasons manufacturers may sell vulnerable products, leaving users with security issues that are unlikely to be resolved. For instance, widespread consumer IoT devices, as well as the routers that connect such devices to the Internet, leave the factory with default credentials that final users are not going to change, and that attackers will exploit to login and take over the system remotely. Also, due to the poor computational capabilities, most IoT devices are unable to host intrusion prevention and/or intrusion detection systems as laptops and desktop computers do. As a matter of fact, IoT devices do not have enough computing power to run an antivirus or even do not allow to install it. Despite some sort of security is implemented by IoT enabling technologies such as authentication and encryption, attackers still find the way to compromise connected devices. Indeed, the numerous hardware and software vulnerabilities of IoT devices, and the multitude of malware that exploit such vulnerabilities, are evidence of this. Therefore, the key to protect IoT devices is to monitor for threats at the network level through an Intrusion Detection System (IDS) installed on the border router or on a dedicated device.

To date, researchers have focused their attention on the development of IDSes for specific IoT technologies/scenarios, possibly with a distributed architecture that requires agents to be installed on IoT devices [13], [21], [42], [43]. These approaches, however, lack of flexibility as they need to be manually reconfigured when the monitored IoT network evolves, and they are too invasive for particularly constrained devices that barely have the capabilities of running an IDS instance along with the task they are designed for, which represents a further opportunity for attackers to easily perform DoS attacks on those devices.

On the other hand, the industry has come up with centralized security solutions for the IoT (e.g. Norton Core [7], ZingBox [14], F-Secure SENSE [9], Bitdefender Box [3]) in the form of routers with multiple functionalities such as antivirus, firewall, IDS, and the ability of taking countermeasures upon the detection of intrusions (e.g., isolating suspicious devices, or shutting down the traffic to keep sensitive information from leaving the network). One of the main strengths of these products is the federated architecture they belong to, whereby the parent company can keeps up to date all of its devices against the most recent attacks thanks to the thousands of security reports that they constantly deliver.

However, the heterogeneity which characterizes the IoT increases the difficulty of deploying all-encompassing security solutions. Such a heterogeneity mainly concerns network topology, communication protocols, provided services, and hardware, that may be of different types even within the same IoT network, as in the case of smart homes. This variety of technologies expands the attack surface compared to wireless sensor networks that are composed of devices homogeneous with respect to hardware and software. Furthermore, the ever-evolving nature of IoT networks, i.e. the possibility of easily add/remove (potentially different) devices into/from the mon-

Manuscript received MONTH DAY, YEAR; revised MONTH DAY, YEAR. The work reported in this paper has been supported by NSF under grant 2112471 "AI Institute for Future Edge Networks and Distributed Intelligence (AI-EDGE)", and by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

Antonino Rullo is with the Department of Computer Engineering, Modeling, Electronics, and Systems, University of Calabria, 87036 Rende, Italy, and with the Institute for High Performance Computing and Networking, National Research Council (ICAR-CNR), Via P. Bucci, 87036 Rende (CS), Italy (e-mail: n.rullo@dimes.unical.it)

Anand Mudjerikar and Elisa Bertino are with the Lawson Computer Science Department, Purdue University, West Lafayette, IN, USA (e-mail: amudgeri@purdue.edu, bertino@purdue.edu)

Daniele Midi is with Google, San Francisco, CA, USA (e-mail: midi@google.com)

itored area, requires an IDS that is optimal at a certain point in time that continues to be so even when changes occur.

Nevertheless, both the academy and the industry have paid very poor attention to the importance of having a security tool able to autonomously adapt to current network settings, that is, choosing the best security strategy in relation to the features of the system under monitoring. Such a characteristic is called *context-awareness* [41], and refers to the ability of an IDS to collect information about its surroundings at any given time, and adapt its behavior according to the current needs. Context-awareness may improve IDSes performances mainly because *when simply observing events that may be symptoms of security incidents, a deep knowledge about the IoT network (i.e., specific network features) can improve either the detection and the attack classification accuracy*. In other words, a deep knowledge of the system under monitoring allows an IDS to focus only on the threats that may actually occur at the present time, to spend the maximum computing capabilities for the detection of such threats, and, as a consequence, to better classify ongoing attacks. In particular, accurately classifying detected attacks is extremely important when remedy actions have to be performed in relation with the (potential) damage. In such cases, a misclassification may produce an incorrect, potentially harmful attack response.

In this work, we first analyze the characteristics that make IoT a unique domain, we investigate the relationship between different network/device features and related attacks, and we outline the advantages that context-awareness entails for intrusion detection. Then, we propose a hybrid Security-as-a-Service (SECaaS) based approach for context-aware intrusion detection, whereby the risk assessment and the selection of the best detection strategy are accomplished on a cloud basis, while the discovery of the network features and the detection task are performed locally. We also present Kalis2.0, a Service-Oriented software Architecture (SOA) [31] which enables the security devices that operate at the local level to discover the characteristics of the monitored IoT network and perform intrusion detection.

Kalis2.0 is an improvement of Kalis, a preliminary version of the architecture that we presented in [25]. In this work, we rearrange the Kalis design to conform to a more functional architecture, whereby each component carries out a specific job for the purposes of system efficiency only, while all the security tasks are exclusively carried out by detection modules. Beside the architectural improvements, we also address a significant limitation of Kalis lying in its local character, whereby a Kalis node is manually configured by the local administrator(s), who bases decisions on local evidence, and the level of security is bounded by the administrator experience and capabilities. In the model we propose in this article, instead, the administrator is only required to subscribe at a service provider and enroll one or more security devices, while the service provider takes care of deploying all necessary security solutions. The service provider is built according with the SECaaS paradigm, whereby security experts exploit formal and automated procedures for assessing risk and determining defensive strategies accordingly. Figure 1 shows the differences between the two security models.

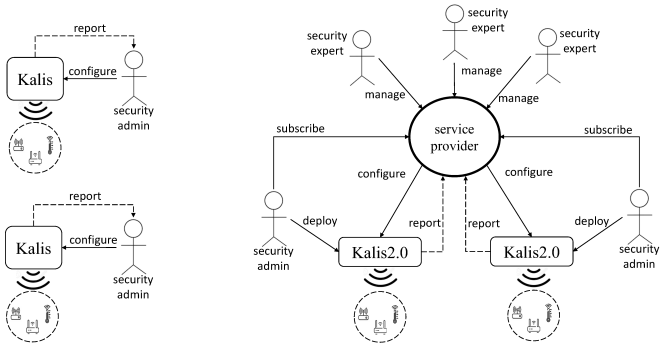


Fig. 1. Kalis (left) vs the SECaaS-based approach (right).

Finally, we propose a set of experiments to demonstrate how context awareness is essential for enhancing intrusion detection efficacy and efficiency.

Our contributions can be summarized as follows:

- the conceptual modeling of a Security-as-a-Service based approach for the accomplishment of context-aware intrusion detection tasks in IoT networks;
- the definition of IoT attack taxonomies that show the relationship between network/device features, vulnerabilities, and potential attacks as the basis of risk assessment;
- a systematic approach for assessing risk and selecting the most appropriate detection strategy accordingly;
- the design of Kalis2.0, a software architecture for a self-configuring, context-aware IDS;
- the evaluation of a Kalis2.0 prototype over a wide range of network features and attacks.

The rest of the paper is organized as follows. In Section II we discuss related work. In Section III we discuss some background concepts on the IoT domain and on context-aware intrusion detection. In Section IV we present our conceptual model for context-aware intrusion detection. In section V we discuss feature discovery techniques. In sections VI and VII we present a systematic method for risk assessment, and for the selection of the best detection strategy, respectively. In section VIII we present Kalis2.0, a software architecture to implement a self-adaptive, context-aware IDS. In Section IX we talk about the potential advantages of using our context-aware approach. In Section X we report experimental results on the difference between context-aware and non-context-aware intrusion detection. Finally, in Section XI we draw conclusions and propose future work.

II. Related Work

The literature on context-aware security for the IoT can be classified in *anomaly-based* and *risk-based* approaches. Anomaly-based approaches take security decisions depending on whether or not specific events occur in their usual context. Often the user is also considered as part of the system and its behavior contributes to the definition of the context. In risk-based approaches, instead, security actions are triggered based on how potentially harmful the context is.

TABLE I
LITERATURE COMPARISON.

Work	Approach	Context	Context focus	Objective	Self-adaptive	Scope
Sikder et al. [50]	anomaly-based	What the sys. does	sensors and user data	intrusion detection	no	sensor-based IoT nets
Sikder et al. [51]	anomaly-based	What the sys. does	sensors and user data	intrusion detection	no	smart homes
Khanpara et al. [17]	anomaly-based	What the sys. does	sensors and user data	real time assessment of user permissions	no	smart homes
Sylla et al. [52]	risk-based	What the sys. does	sensors and user data	real time enforcement of security/privacy policies	no	smart homes/cities
Pan et al. [36]	anomaly-based	What the sys. does	sys log, GPS, protocols	intrusion detection	no	BAC nets
De Matos et al. [8]	risk-based	What the sys. does	sensors data	real time enforcement of security/privacy policies	no	generic IoT nets
Park et al. [37]	anomaly-based	What the sys. does	sensors data, sys log, net log, CPU usage, memory usage, network usage	intrusion detection	no	smart factories
Hameed et al. [12]	anomaly-based	What the sys. does	device ID, location and activity	intrusion detection	no	mobile IoT nets
Moshin et al. [27]	risk-based	What the sys. looks like	network topology, user security policies	recommendations to increase resiliency	no	sensors-controllers-actuators nets
Jia et al. [15]	anomaly-based	What the sys. does	UID/GID, UI activity, control flow, runtime value, data flow	real time enforcement of app functionalities	no	appified IoT platforms
Rullo et al. [44]	risk-based	What the sys. looks like	protocols and protocols embedded prevention systems	real time enforcement of detection techniques	yes	generic IoT nets
This work	risk-based	What the sys. looks like	network topology, mobility, protocols, open ports, OS, prevention systems, network services, device model	real time enforcement of detection techniques	yes	generic IoT nets

A further classification can be made on the basis of how the context is defined. In this regard, most of the work (typically anomaly-based approaches) define the context based on *what the system does*, that is, the context is learned on historical data such as sensed data, user location, sensors-controllers-actuators interaction patterns, temporal patterns, etc. On the other hand, other work (either risk-based and anomaly-based) define the context based on *what the system looks like*, that is, the context is defined as the current state of the system in terms of structural characteristics such as topology and mobility, user security policies in place, communication protocols, network services, protocol built-in security mechanisms etc.

Research has been conducted on the above topics under various settings, and main challenges and results achieved so far in the literature are discussed in the following subsections. In Table I, related work are compared on the basis of a set of features relevant for this topic.

A. Anomaly-based approaches

Sikder et al. [50] proposed *6thsense*, an IDS for multi-sensors devices that observes changes in sensor data and creates a contextual model that distinguishes benign and malicious behavior of the sensors in relation with the current user task. The same authors proposed *AEGIS* [51], an IDS for smart homes which correlates the devices' operational patterns with user activities, and builds a Markov Chain-based machine learning model to define benign user behavior. Similar to [51], Khanpara et al. [17] proposed a permission framework for smart homes where the user is considered as part of the monitored system, and the normal system behavior (i.e., the context) is characterized by means of historical data, such as devices' locations, user's locations, time usage patterns, and actions. The user is granted with the permission of using home devices prior authentication, and if his/her current request pattern matches one of those that were labeled as normal. Pan et al. [36] proposed an IDS for building automation and control networks, where the concept of context modeling is to represent the information acquired from the sensors and resources during the execution of the system. A vectorial representation is generated for each resource/sensor as a description of its normal behavior, then Bayesian Network and RIPPER algorithms are trained on historical data and used to detect abnormal behavior. Paerk et al. [37] proposed an IDS for smart factories, which exploits machine learning techniques to build a model of the normal system's behavior. Hameed et al. [12] proposed a distributed clone node attack detection technique for mobile IoT networks. When the clone node detection is to be initiated, devices can either assume the role of prover or the role of verifier. The prover produces a proof of presence as a signature of its context information, and the verifier checks the signature to confirm or deny the prover's physical presence. The context information includes the device's identifier, timestamp, device's location, and device's activity. Jia et al. [15] proposed *ContextIoT*, a permission system for appified IoT platforms that helps users to perform access control, where the context is defined as the program path of an app functionality, that is, the execution flow of the code at runtime along with the data flowing through the execution path. In these settings, app functions are allowed only when the permissions granted by the user match a particular usage context. ContextIoT learns the user security preferences for each functionality and allows or denies their enforcement on the basis of the current context.

B. Risk-based approaches

Moshin et al. [27] proposed *IoTSAT*, a formal framework for security analysis of IoT networks that are based on the "sensors-controllers-actuators" model. IoTSAT defines the context as the system current state on the basis of the network topology and user security policies, and outputs a set of

recommendations for increasing the system resiliency against potential attacks (e.g., the increase of sensors redundancy). De Matos et al. [8] proposed a reasoning engine architecture designed to execute pre-defined security directives such as IF *contextA* AND *contextB* THEN *action*, where the contexts are defined as sets of past events, and *action* is the enforcement of a security/privacy policy. Similarly, Sylla et al. [52] presented *SETUCOM*, a risk assessment framework which selects security and privacy policies (e.g., the application of two factor authentication, the implementation of secure communication, etc.) on the basis of both user's and devices' behavior. Rullo et al. [44] proposed *PAST* a self-adaptive IDS which activates specific defense techniques on the basis of the attacks targeting the protocols adopted by the devices under monitoring, and of the security features provided by the protocols themselves.

C. Other work

A detailed survey on context aware computing [39] describes diverse kinds of techniques for the collection, modelling, reasoning, and distribution of context data of IoT networks. The authors distinguished approaches to context modelling into six categories, based on the data structures used for the design and exchange of the context: *key-value based* (e.g., text or binary files), used to model limited amount of data such as network configuration and features; *markup scheme based* (e.g., *xml*), useful for context exchange; *graphical* (e.g., databases), good for long term and large volume of permanent data archival; *object-oriented* (e.g., programming languages), used to model data using class hierarchies and relationships; *logic based* (e.g., facts, expressions, and rules), primarily used to express policies, constraints, and preferences, it enables for reasoning tasks; *ontology based* (semantic technologies such as RDF, RDFS and OWL), used to model domain knowledge and context structure.

Context awareness was the core concept also for the design of techniques to calculate the optimal placement of security resources in IoT networks. In this regard, Rullo et al. [45] showed how to determine Pareto-optimal resource allocation strategies on the basis of the network topology and the characteristics of the available security resources. This approach was later adapted for large-scale [46] and mobile [47] networks.

Out of the IoT context and in a more general computer setting, More et al. [28] stated that ontological reasoning can confer an IDS the ability to link and infer means and consequences of cyber threats and vulnerabilities whose signatures are not yet available. In this regard, they presented a knowledge-based approach to intrusion detection modeling, where data from different sources (e.g., host logs, network logs, hardware utilization), along with IDS/IPS-originated information, data from different sensor streams, vulnerability description feeds, and domain expert knowledge is collected into an ontological form and fed into a reasoner with the purpose of detecting ongoing attacks.

D. Discussion and Comparison

Though there is no direct comparable work to compare with, differences between existing context-aware based security solutions for the IoT and our framework can be noted as follows.

First, from Table I we observe that a method that enables the security approach to be automatically adjusted to changes in the context is not taken into account in the works thus far discussed, except for our previous work [44] which is focused on IoT protocols features only. Despite some of these approaches are designed in a way that makes it simple to implement the self-adaptive property, instructions on how to do it have not been explicitly provided.

Second, *risk-based* approaches [8], [27], [52] do not focus on intrusion detection, but rather on the real time enforcement of attack prevention systems such as security and privacy policies. On the other hand, works focusing on intrusion detection [12], [36], [37], [50], [51] presented *anomaly-based* approaches whereby the context is defined based on *what the system does*, i.e., data collected from multiple sources which characterize the behavior of the system in a global manner, and fed into a predefined detection algorithm.

Finally, approaches whereby the context is defined based on *what the system looks like* consider very few system features, that are network topology and security policies in [27], and protocols in [44].

Our work differs in the following things: *(i)* we provide the guidelines for the design of a context-aware, *risk-based* IDS architecture which does not rely on a predefined detection algorithm, rather it enforces an ad-hoc detection strategy determined on the basis of *what the system looks like*; we stress that these characteristics make our framework orthogonal to the approaches presented so far, rather than alternative; *(ii)* we consider a wider range of system features, enabling a detailed characterisation of the system under monitoring and, as a result, an accurate assessment of the risk in terms of potential attacks; and *(iii)* we consider the self-adaptiveness as a key property for an IDS in the face of the continuous evolutions that IoT networks experience throughout their life cycle.

III. Background

A. Internet of Things

The IoT has several characteristics that make it a challenging domain for security measures design. The wide range of hardware used for IoT devices results in a diverse set of communication mediums utilized. IoT applications serve for many different human activities, thus application-dependent features like network topology, network size, and mobility, may present different criticalities and points of failure.

Despite the heterogeneity that characterizes the IoT in many of its architectural aspects, several common security flaws put IoT systems under the same umbrella. The majority of IoT devices are low resource devices, in terms of power source, bandwidth communication, and computational capabilities. In contrast to standard computing systems, most IoT devices are less maintained and upgraded by the manufacturers. The technologies used for communication, operating system design, authentication, and firmware update share vulnerabilities that malicious users can exploit to infect IoT devices and launch a variety of attacks. These attacks may show different signatures (or may not even occur) according with the features of the targeted system, the most discriminating being mobility, network

topology, communication protocols, and prevention systems in use (e.g., access control, encryption, authentication).

B. Context-aware Intrusion Detection

Prevention mechanisms are proactive approaches that attempt to prevent security attacks in the first place. However, it is possible that security attacks succeed and proceed in an IoT system despite the adoption of prevention techniques. Thus, it is extremely important to detect such attacks at the earliest so that actions can be taken to stop further damage. To this end, reactive approaches like Intrusion Detection Systems (IDSes) can be adopted as the last line of defense, in an attempt to detect ongoing attacks.

The choice of the most effective techniques to be executed by an IDS is a fundamental task. However, this task if far from being simple as it requires a deep knowledge of the monitored system to be really effective. Moreover, since IoT instances are dynamic environments, a particular configuration of the IDS that is optimal at a certain point in time might not any longer be optimal later on. A possible, though naive solution, is activating as many detection techniques as possible in order to guarantee a good coverage against potential attacks. However, it presents two main drawbacks: First, the deployment of inappropriate detection techniques may be cause of inaccuracy, as they would produce a high number of false positives. Second, processing network events and traffic through all the detection techniques requires an unnecessary high amount of system resources, and can even introduce delays in the attack reaction.

In light of these considerations, an IDS would certainly benefit from an autonomous mechanism which collects the features of the system under monitoring, and selects the detection techniques that best fit with them. This working mode can be referred to as *context-aware intrusion detection*.

Two properties must be met by an IDS in order for it to be context aware, that are *selectivity* and *self-adaptivity*:

- **Selectivity:** An IDS must be able to drive the choice of the defense strategy (either for the inclusion and exclusion of detection techniques) based on the features of the monitored network and entities. As a matter of fact, an attacks can be carried out only when specific conditions are in place. For instance, certain attacks that multi-hop networks are vulnerable to do not affect single-hop networks; also, the vulnerabilities affecting IoT devices can be meant as the preconditions for specific attacks to occur.
- **Self-adaptivity:** IoT networks may evolve over time, i.e. devices may be introduced in/removed from the monitored area. This may lead to structural changes of the monitored environment, like topology changes, higher/lower network traffic, etc. In this setting, an IDS must be able to auto-configure itself in order to be effective anytime without the need of human intervention.

IV. Proposed Model

A. Conceptual Model

Our conceptual model for context-aware intrusion detection is based on the following key concepts:

Observation: a piece of information gathered by observing the available events (e.g., the frequency of a type of traffic, a special forwarding field in intercepted packets, a change in signal strength from a node, etc.);

Feature: an intrinsic characteristic of the monitored entities and networks (e.g., multihop vs. singlehop network, mobile vs. static network, open ports of monitored devices, communication protocols, etc.);

Symptom: a particular case of observation that could be associated with a potential security incident (e.g., data losses or inconsistencies, packet duplication or alteration, packet dropping, etc.);

Risk: the security flaws that can be exploited as attack vectors against an IoT system;

Detection technique: a mean to distinguish a security incident (known attack or anomaly) from benign system behavior, that may trigger response activities, such as an alert to a user, and/or automatic response actions, such as re-transmission of packets or device isolation.

With these notions in place, our context-aware intrusion detection approach follows this conceptual process:

Using the set of collected observations, the set of features F of the monitored system S can be determined. Based on the knowledge about F, the risk can be assessed in terms of the set of attacks A that can be potentially harmful for S. Based on the knowledge about A, the set of detection techniques to activate can be determined, which will process the available information to detect security incidents by the set of observed symptoms.

B. Context-aware Intrusion Detection-as-a-Service

Our conceptual model for intrusion detection requires four main tasks to be implemented, that are:

(1) the discovery of the features of the system under monitoring;
(2) the assessment of the risk in terms of potential threats based on the collected features;
(3) the selection of the most fitting detection strategy based on the assessed risk;
(4) the execution of the selected detection techniques.

We implement the above tasks according to the Security-as-a-Service (SECaaS) paradigm. In the SECaaS model a service provider supplies security services on a subscription basis, which offers subscribers a number of benefits: First, SECaaS offers ongoing protection since databases are regularly updated to offer the most recent security coverage. Second, when the overall cost of ownership is taken into account, it can be done more affordably than the majority of individuals can do on their own.

Typical SECaaS-based applications provide for their services to be executed on a cloud-basis. For instance, *PENTESTON* [38] is a platform designed to remotely collect infor-

mation and vulnerabilities about the targeted systems (task 1); also, the literature provides cloud-based solutions such as [20], [55] where the intrusion detection functions are executed remotely (task 4) with respect to where the data is collected. Due to the high computational and storage capabilities, cloud-based solutions can ensure high performances in terms of detection accuracy. Indeed, they can collect very large amount of data and fully exploit the potential of machine learning to build very accurate models of the monitored IoT instances. However, such an effectiveness does not come for free but at expense of efficiency: it requires IoT data to be migrated to the cloud, and the detection outcomes to be delivered far away from where the detection task takes place. This brings up various issues, including data privacy [54], significant communication overhead, and high communication latency. For these reasons we believe that a purely cloud-based security solution is not fully appropriate to address the IoT's security issues.

In light of these considerations, we propose a hybrid model composed of two interacting layers, namely, a *cloud* layer and a *local* layer. According to this model, tasks 1 and 4 are performed locally, i.e., in the IoT network area or at the most at the fog level [19], [22], while tasks 2 and 3 are implemented at the cloud level according to the SECaaS paradigm, that is, a service provider assesses the risk based on the information gathered by the security devices it interacts with and deploys them the most suitable intrusion detection techniques to perform in the monitored IoT networks. The SECaaS-based approach for the accomplishment of tasks 2 and 3 brings two main advantages, that are: *(i)* a context-aware, dynamic, flexible, and customized implementation of an intrusion detection strategy as a set of detection functions that are automatically provisioned and dynamically migrated based on real-time security requirements; and *(ii)* the availability of a wide variety of detection techniques to address the heterogeneous nature of the IoT, which solves the problem of individuals having to equip their security tools with a limited set of intrusion detection modules that would never be sufficient in the face of the large and ever-evolving attack surface that characterizes the IoT.

Figure 2 depicts the interaction steps between the actors of our SECaaS-based model, namely, the user, the security devices, and the service provider. First, the user subscribes its security devices at the service provider (1), which provides them with a procedure for the discovery of the features of the IoT networks under monitoring (2). Each device installs and executes the feature discovery procedure (3) – task 1 – and sends the collected data to the service provider (4), which we refer to as *feature discovery report*. At this point the service provider performs tasks 2 and 3 of our conceptual model, i.e., based on the set of collected features it assess the risk and chooses the most fitting detection strategy as a set of detection modules (5). Such modules are then sent to the security devices deployed on the field (6), that will install and execute them (7) – task 4.

The security devices repeat the discovery process at regular time intervals, or when cues of network changes are detected. The discovery of new features causes further connections with

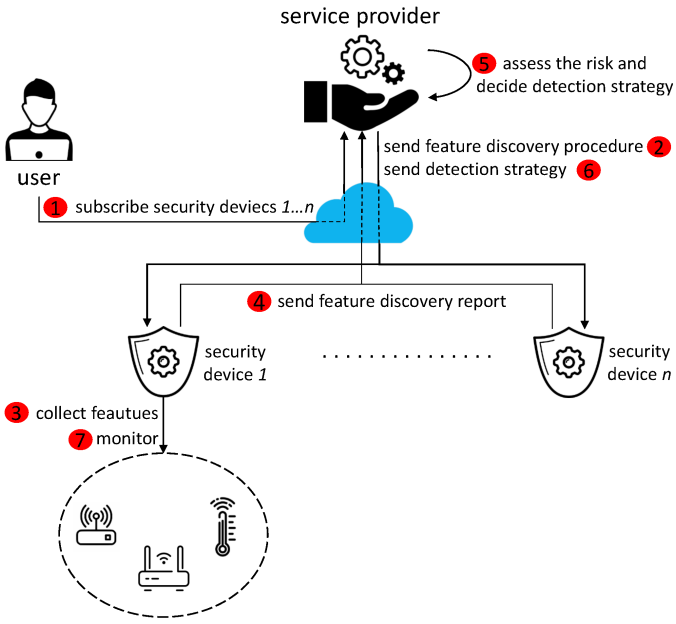


Fig. 2. The proposed SECaaS-based model.

the service provider and the consequent potential installation of new detection modules, and/or the uninstallation of the detection modules in execution that are no longer useful in the current network setting.

In order for security devices to accomplish these tasks, we propose Kalis2.0, a service oriented software architecture that enables security devices to discover the characteristics of the monitored IoT network (task 1) and perform intrusion detection accordingly (task 4). Kalis2.0 is a modular architecture, whose components interact with each other to efficiently handle sniffed network traffic in relation to the detection modules in execution. In particular: a module collects information about the monitored entities; a module generates and maintains a concise representation of the feature discovery report; a module supplies the detection algorithms in execution with only the network packets of interest for the specific detection task; a module routes sniffed traffic to the other components; and a module in is charge of configuring the whole system and coordinating the interactions between components. A representation of Kalis2.0 is depicted in Figure 5. In Section VIII a more detailed description of each architectural component is provided.

V. Feature Discovery

Feature discovery is the key activity that enables intrusion detection to be context sensitive. The outcome of feature discovery is a report on the characteristics of the monitored system either at the network and device level. In the following we describe three techniques for feature discovery, namely, *port scanning*, *banner grabbing*, and *penetration testing*, and provide some hints on how they must be performed in an IoT scenario.

A. Port Scanning

Port scanning consists in sending requests to a range of port addresses on a host, with the goal of finding active ports. Considered that there are 65536 distinct port addresses, many port scanning tools only test the ports most commonly associated with most commonly used network services, in order to avoid keeping busy the computational resources of the tested device. However, modern malware are aware of that, and purposely scan on uncommon port numbers in order to pass undetected by IDSs (see Table VII). In light of this, a TCP scan can be performed on all the 65536 ports only during the time slots in which the tested devices are in an idle state, so as to not affect their normal operation. The TCP port scan is preferred to other types of scan (e.g. UDP and SYN scans) as, besides enumerating open ports, it also allows to collect information about the network services running behind such ports.

B. Banner Grabbing

Banner grabbing allows to gain information about a networked system, which include device's related information such as communication protocols in use, Operating System, network services (and their version), MAC address, device type, model and manufacturer, and network's related information such as topology and routing scheme. Banner grabbing can be conducted either in a passive and in an active manner: the passive mode provides for the inspection of the header of sniffed packets, while the active one involves probing the tested device with a port scan, and then analyze the response data. The data gathered by banner grabbing can be used to further investigate on the tested devices in order to collect additional information, that in turn may reveal further security flaws. Also, some vulnerabilities can be traced back from the knowledge about the device type and the network services it hosts.

C. Penetration Testing

A penetration test is an offensive security approach in which the tester probes the target system in order to gain information about its security flaws [49]. Penetration tests are fundamental as they allow to determine the security status of the monitored IoT devices, that is, how vulnerable are them and to which attacks. Such information can be collected by means of a process enumeration task, a vulnerability scan, and by simulating attacks (without performing malicious actions) to discover further vulnerabilities. The analysis of running processes can be leveraged to trace back to vulnerabilities affecting those processes. A dictionary attack can be performed to check whether the tested devices have weak credentials. Such an attack provides for the tester to login by trying different default usernames and passwords, in particular the username-password pairs typically used by malware to gain user privileges (e.g., Mirai [2] uses 62 pairs), plus the default credentials typically assigned by devices' manufacturers. The tested devices are affected by such a vulnerability if one or more of the following situations occur:

- the tester authenticates with default credentials (weak password vulnerability);
- the tested device does not ask for a second authentication step after the tester guesses a username-password pair (no-two factor authentication vulnerability);
- after a low number of failed login attempts (typically 3 or 5) the tested device allows to continue sending authentication attempts (account lockout vulnerability);
- the tested device leaks useful information in response of each failed authentication attempt (username enumeration vulnerability).

The vulnerability scan assesses the tested devices for known vulnerabilities. There exist two types of vulnerability scans, namely *authenticated* and *unauthenticated*. The first type authenticates using login credentials and performs a more accurate test by accessing low-level data, such as specific services and configuration details of the Operating System. The unauthenticated scan, instead, cannot go as deep as its authenticated counterpart, thus it retrieves fewer information. A vulnerability scan can be performed following the simulation of a dictionary attack. In this regard, if the dictionary attack succeeds, then an authenticated scan can be performed with the guessed username-password pair, otherwise an unauthenticated scan can be initiated.

VI. Risk Assessment

Risk assessment is the process of identifying potential threats on the basis of the characteristics of a target system. As we mentioned in Section II, contrary to the majority of work on context-aware IoT security, we do not characterize the target system in terms of what it does, but rather in terms of how it is made as the knowledge obtained by means of a feature discovery task. In particular, we distinguish two types of knowledge, namely *structural* and *security*. The former is about the anatomy of the monitored system, such as network topology, mobility, protocol stack, devices' type, devices' operating system/firmware, devices' open ports, MAC addresses, devices' running processes, attack prevention systems and network services provided by the IoT technologies in place. It is collected by means of port scanning, banner grabbing and process enumeration, and its analysis allows to assess the risk in terms of which attacks the target system can undergo and which not. In fact, there are relationships between IoT technologies and security incidents that allow to assess the possibility and impossibility for an attack to happen in presence of a specific feature.

Security knowledge is about the security weaknesses affecting the monitored entities, such as weak authentication, weak encryption, and weak update mechanisms, as well as the set of vulnerabilities retrieved by the vulnerability scan. As opposed to structural knowledge, which must instead be analysed offline to uncover clues about potential hazards, such security flaws are instantly detectable through the online analysis of sniffed network traffic and penetration testing techniques. Structural and security information allows to obtain a full picture of the monitored IoT system as the starting point for a risk assessment task.

TABLE II
A CLASSIFICATION OF IOT VULNERABILITIES BASED ON FEATURE DISCOVERY TECHNIQUES.

Category	Vulnerabilities	Consequences	Discovery technique
V1	**V1.1** - Username Enumeration **V1.2** - Weak Passwords **V1.3** - Account Lockout **V1.4** - No Two-factor Authentication	After the attacker takes control of the device s/he can launch any kind of attack against other entities, or put the device out of service.	penetration testing
V2	**V2.1** - Unencrypted Services **V2.2** - Poorly Implemented Encryption **V2.3** - Update Sent Without Encryption	Data could be loss, stolen or modified and, depending on the data exposed, could lead to complete compromise of the device or user account. Architecture of a device, it's file system, any buffer overflows and secret information such as passwords, certificate information or database addresses could be disclosed if updates are sent without been encrypted. The attacker can inject extra code in firmware so that unauthorized operations can be performed.	passive banner grabbing
V3	**V3.1** - Update Location Writable **V3.2** - Firmware and storage extraction	Attackers can intercept OTA updates, or even download the firmware from the manufacturer web page. In both cases attackers can modify the firmware and distribute it to all users. Insecure firmware could lead to compromise of user data, control over the device and attacks against other devices.	offline analysis of the structural knowledge collected with banner grabbing
V4	**V4.1** - Insecure Components **V4.2** - Insecure Network Services	Attackers can gain access to the device by means of ports unnecessary exposed to the internet. Insecure network services and running processes, such as out of date versions of busybox, openssl, ssh, web servers, etc., may be susceptible to buffer overflow or denial of service. Denial of service attacks against other users may also be facilitated when insecure network services are available.	offline analysis of the structural knowledge collected with port scanning and process enumeration

As it receives a feature discovery report, the service provider determines the set of vulnerabilities the monitored system is affected by, that are the necessary conditions and the points of entry for attackers and malware to perform malicious tasks. There are vulnerabilities in almost every aspect of IoT, including operating system, communication protocols, and APIs, and their exploitation causes either the control of the affected devices and the leakage of exchanged data, opening the door to a variety of attacks at network level such as eavesdropping, man in the middle, spoofing, flooding, routing attacks, but also to malware infections, leading to consequences like denial of service, data theft, and data loss.

The Open Web Application Security Project (OWASP) [33] has identified the most common vulnerabilities of IoT devices, among which we have selected those that can be retrieved by means of the feature discovery techniques discussed in Section V, while excluding those that are detectable by means of human intervention only, such as the susceptibility to side channel attacks like *glitching* [30], which requires the physical access to a device. We grouped them into four categories on the basis of the technique used for their discovery. Table II maps OWASP vulnerabilities with the four categories, and for each category it shows the consequences that could arise if the related vulnerabilities were exploited, along with the technique required for their discovery.

A. Taxonomies

In order to meaningfully characterize the IoT threats in relation with the feature discovery reports it receives, the service provider uses a set of taxonomies that look at the IoT security from different perspectives, namely, the *network* perspective and the *device* perspective.

We hereby show instances of such taxonomies for the two perspectives. Since a complete characterization of IoT threats is beyond the scope of this paper, they should not be considered exhaustive or accurate, rather, they must be intended only as illustrative examples.

1) Network Perspective: There exist a number of attacks that can be performed at the network layer, and as such, the possibility for these attack to happen/success strictly depends on the presence of specific network features. In Table III we show a taxonomy for the most common network features and network attacks, with dots and crosses indicating the possibility and impossibility, respectively, for an attack to happen in presence of a specific feature. For instance, *sinkhole* attacks cannot be performed when routing paths are static, while *smurf* attacks are not possible in single-hop topology. We also include the presence of prevention systems. Although they could be considered as a device feature, prevention systems decrease the success rate of most network attacks. The knowledge about the prevention mechanisms in place is fundamental for risk assessment since they serve as a first line of defense, preventing or thwarting some malicious activities and allowing the IDS to account for a smaller number of threats. Data tampering attempts, for example, are made ineffective if authentication mechanisms such as signatures or Message Authentication Codes (MAC) are enforced by some of the monitored devices.

TABLE III
TAXONOMY FOR NETWORK ATTACKS AND NETWORK FEATURES.

	NETWORK FEATURES											
	topology		routing				prevention sys.					
NETWORK ATTACKS	single hop	multi hop	static	dynamic	shortest path	multy-path	MAC	encryption	FHSS	ECC	signatures	seq. number
selective forwarding	×	●	●	●	●	×	●	●	●	●	●	●
replay	●	●	●	●	●	●	×	●	●	●	×	×
sinkhole	×	●	×	●	●	×	●	●	●	●	●	●
sybil	●	●	●	●	●	●	×	×	●	●	×	●
wormhole	×	●	●	●	●	●	●	●	●	●	●	●
data alteration	×	●	●	●	●	●	×	●	●	●	×	●
delay	×	●	●	●	●	●	●	●	●	●	●	●
jamming/collision	●	●	●	●	●	●	●	●	×	×	●	●
flooding	×	●	●	●	●	●	×	●	●	●	×	●
smurf/fraggle	×	●	●	●	●	●	×	●	●	●	×	●
spoofing	●	●	●	●	●	●	×	×	●	●	×	●
eavesdropping	●	●	●	●	●	●	●	×	●	●	●	●

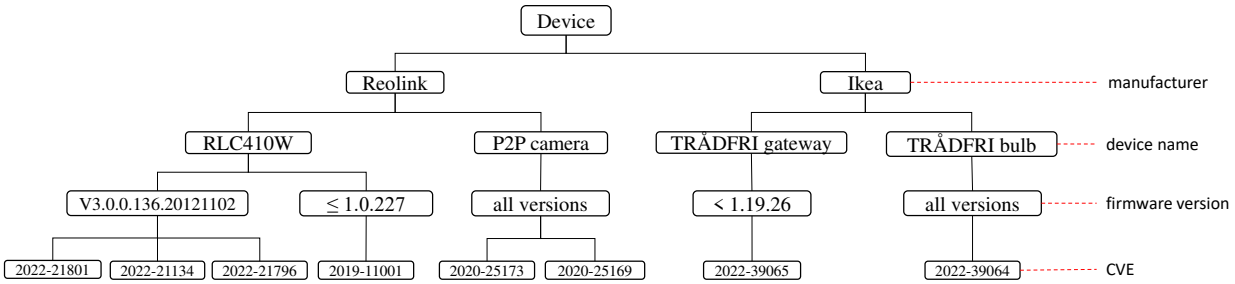


Fig. 3. Taxonomy for device model and related threats.

TABLE IV
TAXONOMY FOR IOT OPERATING SYSTEMS AND RELATED THREATS.

OS	Attack	CVE	Protocol
RIOT	break encryption	2021-41061	none
	buffer overflow	2021-31664	none
	DoS	2019-15702	TCP
	DoS	2019-16754	MQTT
	buffer overflow	2019-1000006	DNS
Tizen	arbitrary code execution	2021-25437	none
	arbitrary code execution	2021-25436	none
Mbed	memory leaks	2020-12887	CoAP
	buffer overflow	2020-12886	CoAP
	resource consumption	2020-12885	CoAP
	DoS	2019-17210	MQTT
Ubuntu Core	privilege escalation	2017-6507	none
	privilege escalation	2016-1576	none
Fuchsia	modify data	2022-0882	none
	privilege escalation	2021-22566	none
HarmonyOS	memory heap overflow	2021-22480	none
	DoS	2021-22479	none
	information leakage	2021-22478	none
NuttX	remote code execution	2021-26461	none
	memory corruption	2020-17529	TCP
FreeRTOS	privilege escalation	2021-43997	none
	buffer overflow	2021-42553	none
	information leakage	2018-16603	TCP
OpenWRT	XSS	2022-41435	none
	memory leaks	2022-38333	none
Contiki-NG	buffer overflow	2021-21281	TCP
	remote code execution	2018-19417	MQTT
	buffer overflow	2022-35927	RPL
	buffer overflow	2022-36054	6LowPAN
	buffer overflow	2023-23609	BLE
	buffer overflow	2022-36053	IPv6

2) Device Perspective: We characterize IoT devices on the basis of the features that can be linked to known attacks and security flaws, namely: manufacturer, device name, operating system/firmware version, protocols, open ports, and network services. Most of information about the relationships between devices features and cyber threats are publicly available in online repositories such as NVD [29], CERT [5], and MITRE [26].

Figure 3 shows a taxonomy for **manufacturer**, **device name**, **firmware version** and vulnerabilities. Device name and firmware version uniquely identify a device model and thus, the set of vulnerabilities it is affected by. On the other hand, feature discovery may not reveal complete information on the device model, which may result in a larger range of vulnerabilities, that is, the leaves belonging to the sub-tree rooted to its manufacturer or device name node.

The analysis of **Operating Systems** (OSes) related threats plays an important role for risk assessment since OSes are the main software that devices run. Moreover, since the implementation of the IoT protocol stack is different for each OS, some OS vulnerability can be exploited only if specific protocols are in place and not otherwise. As an example, Table IV reports a taxonomy of the relationships between OSes, protocols and vulnerabilities.

While non-IoT networks run mostly on top of the TCP/UDP and IP protocols, the IoT depends on a diverse set of communication **protocols**. In Table V we show a taxonomy for the IoT protocol stack and the attacks IoT protocols are vulnerable to. Some protocols feature intrinsic security flaws, while other ones are vulnerable to a variety of attacks depending on specific devices' features. For instance, the RPL routing protocol is subject to the *rank* and *version number* attacks [1] by design, i.e., regardless of device model or OS, whereas the MQTT application protocol can be an attack vector when certain device models are used, or when specific MQTT implementations are in place. For the MQTT protocol, indeed, several implementations are available which can be deduced by looking at the broker's address (which is always visible), and some of which have their own security issues as shown in Table VI.

For a complete understanding of the circumstances under which a malware assault can be successful, devices' **open ports** and **network services** are meant as the prerequisites for malware propagation, along with other device features such as type, model, and CPU architecture. Malware can have a variety of negative effects, including the interruption of devices or network's regular operation (local DoS), the steal of private data like user credentials or user sensitive information (privacy attack), or distributed attacks against a remote host/server (DDoS). Table VII shows a taxonomy of the relationships between devices' features and malware. For each malware, the class of the vulnerability it exploits for infection is indicated in round brackets. For a more exhaustive behavioral analysis of IoT malware we remind the reader to [11].

Notice that features such as device type, protocol, and OS are covered by more than one taxonomy. Despite it may looks inefficient, on the contrary such a redundancy allows to get to vulnerabilities from different perspectives, allowing a vulnerability that is shared by multiple features to be disclosed if at least one of those features is detected. For instance, the vulnerability CVE-2023-24157 affects the device *TOTOLINK T8 V4.1.5cu* when the application protocol is MQTT (see Table V). Whether the service provider is aware of both features or only one of them (i.e., device model and App protocol),

TABLE V
TAXONOMY FOR IOT PROTOCOLS AND RELATED THREATS.

Protocol	Layer	Attack	Device Feature → Feature value
CoAP	Application	amplification attack	-
		memory corruption (CVE-2022-33211)	device model → Qualcomm 9207 LTE Modem
		read memory (CVE-2020-12886)	device OS → Arm Mbed OS 5.15.3
		buffer overflow (CVE-2019-17212)	device OS → Arm Mbed OS 5.14.0
		DoS (CVE-2020-10063)	device OS → Zephyr RTOS > 2.2
XMPP	Application	XMPPbomb	-
		Authentication bypass (CVE-2018-15721)	device model → Logitech Harmony Hub
AMQP	Application	CSRF (CVE-2023-24447)	network service → RabbitMQ message broker
		DoS (CVE-2020-4931)	network service → IBM MQ messaging system
		MITM (CVE-2018-11087)	running process → Pivotal Spring
MQTT	Application	arbitrary command execution (CVE-2023-24157)	device model → TOTOLINK T8 V4.1.5cu
		improper access control (CVE-2023-22600)	device model → InHand Networks InRouter 302
		DoS (CVE-2019-17389)	device OS → RIOT OS before 2019.07
		remote code execution (CVE-2018-19417)	device OS → Contiki-NG OS before 4.2
DDS	Application	privacy laeks (CVE-2019-15135), buffer overflow (CVE-2022-41838)	-
NTP	Application	DoS [10]	-
		Command Injection (CVE-2022-36786)	device OS → D-link router DSL-224
TCP	Transport	TCP SYN Flood	-
		buffer overflow (CVE-2021-21281)	device OS → Contiki-NG OS before 4.6
		DoS (CVE-2019-15702)	device OS → RIOT OS before 2019.07
RPL	Network	local repair, DAO/DAG inconsistency, rank attack, worst parent, routing table overload, DIO/DIS flooding, version number [53]	-
		buffer overflow (CVE-2022-35927)	device OS → Contiki-NG OS before 4.7
MPL	Network	Suppression attack (DoS) [40]	-
OLSR	Network	MPR flooding attack, incorrect ANSN generation, incorrect TC/MID/HNA message generation	-
AODV, DSR	Network	rush attack, RREP flooding,modification of RREP/RREQ/RRER messages	-
6LoWPAN	Physical	buffer reservation, fragment duplication, authentication attack [53]	-
		buffer overflow (CVE-2022-36054)	device OS → Contiki-NG OS
ZigBee	App/Tran/Net	CSRF (CVE-2019-20480)	device model → MIELE XGW 3000
		DoS (CVE-2022-39064)	device model → TRÅDFRI bulb
		DoS (CVE-2019-15915)	device model → Xiaomi DGNWG03LM
Z-WAVE	All	key reset attack, S0 downgrade attack, route modification, sniffing, DoS, impersonation	-

TABLE VI
MQTT IMPLEMENTATIONS WITH RELATED THREATS

Name	Broker Address	CVE
Mosquitto	mqtt.eclipse.org	2021-41039 (DoS) 2021-34432 (DoS)
HiveMQ	broker.hivemq.com	2020-13821 (credential theft)
Flespi	mqtt.flespi.io	-
Dioty	mqtt.dioty.co	-
Fluux	mqtt.fluux.io	-
EMQ X	broker.emqx.io	2021-46434 (username enumeration) 2021-33175 (DoS)

TABLE VII
TAXONOMY FOR DEVICES' FEATURES AND MALWARE.

Malware	Device feature→Value	Port-Service
Hajime (V1.2)	type→DVR, IP cam, router	TCP 23-Telnet
Satori (V4.2)	model→d-link router	52869-UPnP SOAP
	model→Huawei HG532	37215-N/A
Mirai (V1.2)	type→IP cam, router	TCP 23-Telnet
Okiru (V1.2)	processor→ARC	TCP 23-Telnet
Masuta (V4.2)	model→D-Link EDB 38722	N/A-HNAP,SOAP
Wicked (V4.1)	model→Netgear DGN-1000	8080-HTTP
	type→CCTV-DVR	81-HTTP
	model→Netgear R7000	8443-N/A
Sora (V4.1)	model→Rasilient PixelStor	N/A-N/A
Amnesia (V4.1)	type→DVR	N/A-N/A
Remaiten (V1.2)	type→router	TCP 23-Telnet
BrickerBot (V4.2)	any	TCP 23-Telnet
Darlloz (V4.1)	type→router, IP cam, type→set-top box	N/A-N/A
Aidra (V1.2)	processor→ARM	TCP 23-Telnet
Bashlite (V4.2)	model→D-Link 850 L Router	N/A-N/A
Moose (V1.2)	processor→ARM, MIPS	TCP 23-Telnet
PNscan (V1.2)	processor→x86, ARM, MIPS, MIPSEL	TCP 22-SSH
Routrem (V1.2)	type→router	TCP 23-Telnet
XOR DDoS (V1.2)	processor→ARM, x86, x64	TCP 22-SSH
Persirai (V4.2)	type→IP cam	TCP 81-HTTP
Hide 'N Seek (V1.2)	type→IP cam	8080, 2480 80, 5984, 23-N/A
IRCTelnet (V1.2)	type→DVR, IP cam	TCP 23-Telnet
Rakos (V1.2)	any	TCP 22-SSH
Kaiji (V1.2)	any	TCP 22-SSH
IOTroop (V1.2)	type→IP cam	N/A-N/A
Linux Rabbit (V1.2)	any	TCP 22-SSH

it can nonetheless take into account this vulnerability, even though with different degrees of confidence, which would be high if both features were known, and lower if just one of them was. We will discuss this aspect more in detail in the next subsection.

B. The Inference Process

Risk assessment is accomplished by cross-referencing the feature discovery report delivered by an enrolled security device with the collection of taxonomies that map IoT features to known threats. This process takes the name of *inference process*, and outputs a collection of security flaws and potential attacks, which we refer to as the *risk model*.

The inference process is quite straightforward when it comes to assessing risk connected to network features and devices models. In the first case, a boolean relation as the one shown in Table III can be used to carry out a top-down

approach, whereby a subset of network attacks is excluded from the set of all possible attacks based on actual network features, to obtain the set of attacks that can actually happen. A tree-based taxonomy as the one shown in Figure 3, instead, can be used to identify a device's security flaws as the leaves belonging to the sub-tree rooted to its manufacturer, name, or firmware version node.

Other features, instead, such as protocols, Operating Systems and open ports, may pose risks either on their own and in combination with other features. For these, a more powerful knowledge representation scheme is required for the risk to be quantified. To this end, we implement the taxonomies for protocols, Operating Systems and open ports as a set of *directed acyclic attack graphs* (DAAGs). In a DAAG, start and intermediate nodes represent features, and terminal nodes represent attacks and security flaws. Two examples are shown in Figure 4, which depicts an excerpt from the DAAGs used to identify the security flaws affecting application protocols (left) and the malware that exploit port 23 to infect devices (right). According to this method, the taxonomy for protocols as the one shown in Table V can be represented with four DAAGs, one for each layer of the IoT protocol stack (Application, Transport, Network, Physical).

The threats to an IoT system caused by a feature f are identified by the set of terminal nodes of a DAAG rooted in f, that are reachable via the intermediate nodes labeled with the name of the features for which some information has been collected. Information are provided by feature discovery reports either at network and device level as a set of feature-value pairs, with value "N/A" denoting the absence of information for a given feature. We formally define a DAAG as follows.

*Definition 1 (**directed acyclic attack graph**): A* directed acyclic attack graph *(DAAG) is a tuple* $\langle s, N_F, N_V, N_T, E, \delta \rangle$*, where:*

- $s \in N_F$ *is the* start node*, i.e., a node with in-degree 0;*
- N_F *is the set of* feature nodes*, i.e., nodes labeled with feature names;*
- N_V *is the set of* value nodes*, i.e., nodes labeled with feature values;*
- N_T *is the set of* terminal nodes*, i.e., nodes with out-degree 0, and labeled with threats names;*
- $E \subseteq \{N_F \times N_V\} \cup \{N_V \times N_F\} \cup \{N_V \times N_T\}$ *is the set of edges;*
- $\delta : N_V \times N_T \rightarrow (0, 1]$ *is an edge labeling function that associates a value in the range* $(0, 1]$ *to each edge originating from a value node labeled with "N/A" and ending at a terminal node.*

In Figure 4, feature nodes and value nodes are represented as pointy rectangles and rectangles, respectively. From now on, for ease of explanation, we may denote a node x labeled with "l" simply as the node l.

Given a value node N/A, and a terminal node t, $\delta(N/A, t)$ tells how likely threat t affects the monitored device according to the information collected about the features that can cause t. Further details on δ will be provided later in this section. The edges going from a feature node f to the value nodes $v_1, \ldots, v_n$ represent the n possible values for feature f, the N/A value included.

*Definition 2 (**active edge**): Let* $G = \langle s, N_F, N_V, N_T, E, \delta \rangle$ *be a DAAG, and* $e = (x, y) \in E$ *be an edge.* e *is defined as* active *in the following two cases:*

- $e \in N_F \times N_V$ *and there exists a feature with name* x *and value* y *in the feature discovery report;*
- *there exists an active edge* $(z, x) \in N_F \times N_V$*.*

Terminal nodes of a DAAG can be reached starting from the start node and following paths involving active edges only. A threat t joins the risk model if there exists a path as a sequence of active edges from s to a terminal node t. In Figure 4, active edges and selected thrates are highlighted in green.

DAAGs and feature discovery reports guide the inference process in determining a risk model for the enrolled security devices. However, a security device may fail to gather all or part of the relevant information about the IoT entities under observation, or rather, it may discover features that are unknown to the service provider. For example, it is reasonable to consider a newly released device model as a feature that is ignored by the service provider for a while, at least until security experts fill in the missing data. In these cases, it's imperative that risk models are also built upon missing knowledge in addition to learned knowledge, otherwise ineffective detection settings could be deployed. Therefore, the absence of information is not be interpreted as the absence of risk, but rather, it must encourage the generation of *redundant* risk models, i.e., that include threats that are mutually exclusive because they originate from the same feature when this is given alternative values, but whose current value is unknown. The fundamental rule to comply with when assessing risk is thus: *When there is no knowledge on the target system or a part of it, a redundant risk model must be generated; if information are provided instead, a more specific risk model can be endorsed*. The deployment of redundant detection settings as a result of a redundant risk model still guarantees adequate protection against potential attacks, but is poorly efficient due to its redundant nature. Redundant settings, for instance, may include detection techniques for the attacks targeting different routing protocols, even though a set of interacting devices do not use more than one routing protocol at the same time. But in absence of information about routing, all available means are required to identify any potential threat to routing tasks, preferring a higher false positive rate to some, but way harmful, false negatives. If the routing protocol is known, instead, a more specific risk model can be generated in favour of a more customized security solution.

To deal with this aspect, a DAAG is designed so that the edges originating from a value node N/A directly reach all the terminal nodes that are also reachable via a feature node f such that $(f, N/A) \in E$. This construction allows to include into the risk model all the threats caused by a given feature in the case no information is provided about it.

Example 1: Figure 4 (left) shows an excerpt of the DAAG which guides the inference process in finding the security flaws linked to IoT application protocols. The active edge going from the start node to the value node `MQTT` is an

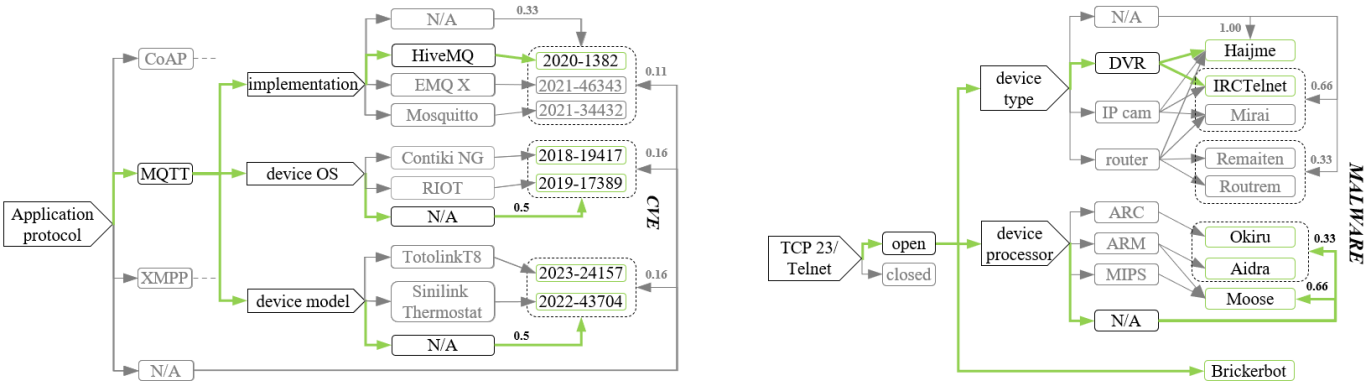


Fig. 4. An excerpt of the DAAGs that guide the inference process in finding the security flaws of IoT protocols (left), and in determining the set of malware that can threaten an IoT device when a set of open ports is given (right).

indication that the feature discovery report includes the pair *(app_protocol, MQTT)*. According with Definition 2, all the edges outgoing from the `MQTT` node are active. The feature nodes `implementation`, `devicesOS` and `device_model` denote features that, in combination with the MQTT protocol, may expose an IoT device to a number of risks depending on their value. The active edges going from nodes `deviceOS` and `device_model` to the `N/A` nodes are an indication that the feature discovery report does not provide any information about such features, while it does for the feature `implementation` with the value `HiveMQ`. As a consequence, the edges linking the `N/A` nodes to the security flaws caused by `deviceOS` and `device_model` features become active. The terminal nodes in green represent the security flaws that will join the risk model.

Example 2: Figure 4 (right) shows an excerpt of the DAAG which guides the inference process in determining what are the malware that can threaten an IoT device when TCP port 23 is open. According with this example, the risk model will include *(i)* the malware that exploit the Telnet service via the TCP port number 23 and affect DVR devices for which no knowledge about their processor is given, plus *(ii)* Brickerbot, a malware that exploits the Telnet service via the TCP port number 23 and spreads without any additional device-specific feature.

With the above concepts in mind we can now clarify how function δ works and why it is relevant. As we have explained earlier in this section, the lack of knowledge about the characteristics of the monitored entities results in a redundant risk model, that is, a collection of threats that are mutually exclusive because they derive from the same feature when this is given values that exclude one another, but that are unknown. For such threats, function δ derives a real value in the range $(0, 1]$ as a measure of the degree of confidence of their existence. More precisely, given the set of active edges $\{(x, t_1) \dots (x, t_n)\}$ where x is labeled with N/A, the value obtained by applying $\delta(x, t_i)$ denotes how likely threat t_i affects the device being monitored, considering the possibility that one of the other $n - 1$ threats may be in its place.

The behavior of function δ is described by the pseudocode sketched in Algorithm 1. For this purpose we use the two utility functions in and $siblings$, defined as:

$$in(y) = \{x | (x, y) \in E \wedge x \neq \text{“}N/A\text{”}\}$$
$$siblings(y) = \{x | \exists z : (z, x) \in E \wedge (z, y) \in E\}$$

where E is the set of edges. Given a node y, function $in(y)$ returns the set of nodes $x \neq N/A$ such that there exists an edge (x, y), while function $siblings(y)$ returns the set of nodes x such that $in(x) \cap in(y) \neq \emptyset$.

Algorithm 1 function δ

Input: $x \in N_V$ labeled with “N/A”, $y \in N_T$ such that $(x, y) \in E$
Output: a label $c \in (0, 1]$ for the edge (x, y)

```
I = in(y)
S = ∅
c = 1
while x ∉ S do
    z ← I                          ▷ z is a node sampled from I
    S = siblings(z)
    if y ∈ N_F ∨ y ∈ N_T then
        c = c · |I|/|S|
    y = z
    I = in(y)
return c
```

Example 3: Consider the set of security flaws highlighted in green in the DAAG of Figure 4 (left). Vulnerability $\texttt{CVE} - 2020 - 1382$ is certainly a security flaw that affects the target IoT device as it belongs to a path which does not include `N/A` nodes. On the other hand, there is a 0.5 degree of confidence that the other vulnerabilities affect the device being monitored. The rationale behind the value 0.5 (computed as the result of the evaluation of function δ) is that each CVE is enabled when the feature which gives rise to it assumes a specific value out of two potential values. Note that, in the case we were not aware of protocol being used, these vulnerabilities would have a degree of confidence of $0.5/3 = 0.16$, where 3 is the number of different application protocols (MQTT, XMPP amd CoAP) provided in this DAAG.

Example 4: Consider the set of malware highlighted in green in the DAAG of Figure 4 (right). Malware `Okiru`, `Aidra` and

`Moose` join the risk model as endings of paths that include an `N/A` node, and thus with a degree of confidence smaller than 1. Either malware `Okiru` and `Aidra` can manifest with a degree of confidence of $1/3 = 0.33$. The value 0.33 is because they can threaten devices having ARC and ARM processors, respectively, that are one of the three potential values (ARM, ARC and MIPS) of the feature `device_processor`. Malware `Moose`, instead, has a degree of confidence of $2/3 = 0.66$, since it can threaten devices having either an ARM or a MIPS processor.

Every threat included in a risk model is represented as a pair $\langle t, c\rangle$, where t is the threat identifier, and c is a real value in $(0, 1]$ obtained as the result of the evaluation of function δ. For threats to which δ does not apply but that still belong to the risk model, $c = 1$ by default (e.g., CVE-2020-1382 in the DAAG of Figure 4 (left)).

As we are going to see in the next section, function δ can help tuning detection settings for efficiency purposes. Security tools deploying detection settings derived from highly redundant risk models, indeed, may incur in a significant computational burden. To avoid this, more lightweight detection settings can be obtained by excluding from the risk model some of the threats characterized by very low c values.

VII. Selection of the Detection Strategy

Based on the risk assessed in terms of vulnerabilities and potential attacks, the service provider determines the intrusion detection settings for enrolled security devices, that we refer to as *detection strategies* (DSes). A DS is a collection of *detection modules* (or simply *modules*), i.e., detection algorithms that analyze network traffic and raise an alert if security incidents or anomalies are recognized. We assume, without loss of generality, that there exists a 1-to-n mapping from the set of detection modules to the set of threats, i.e., a detection module addresses one or more threats. This is a reasonable assumption since attacks that cause the same symptoms (i.e., thet share the same signature) can be detected by the same detection technique. This is the case, for example, of many malware such as Mirai, Remaiten, Okiru, etc., that establish a connection with the target device on the TCP port 23 and perform DDoS attacks, thus generating a large amount of outbound traffic.

A detection module dm is identified by a triple $\langle T, w, D\rangle$, where T is the set of threats dm deals with, $w = \max_{\langle t,c\rangle \in T} c$ is a real value in $(0, 1]$, and D is the set of IoT devices to be monitored by dm to address threats in T. w is the highest c value among the threats $\langle t, c\rangle \in T$, and denotes how likely at least one threat in T will affect the IoT devices in D.

The first step for determining a detection strategy is selecting the detection modules that address the threats provided in the risk model. A detection strategy generated from a redundant risk model, however, may also be redundant, and as such it may experience performance issues like computational overhead, high false positive rate, and low attack classification accuracy. The redundancy degree of a detection strategy DS can be derived by evaluating an aggregate function $\phi : DS \rightarrow [0, 1)$ over the w values of the detection modules in DS, with value 0 meaning a non-redundant strategy, and value 1 a very high redundant one. Two candidate aggregate functions are:

$$\phi(DS) = 1 - \frac{1}{|DS|} \cdot \sum_{\langle T,w,D\rangle \in DS} w$$

$$\phi(DS) = 1 - \min_{\langle T,w,D\rangle \in DS} w$$

that generate a real value as the complement to 1 of the average (top) and the minimum (bottom) w values computed over the set of modules. The aggregate function can be either fixed or user-dependent, i.e., chosen or defined by the user at enrollment time depending on his/her security requirements. If the redundancy degree r of a detection strategy DS goes above a certain threshold value $h \in (0, 1)$, the service provider seeks to generate a more lightweight strategy DS^* with $r^* < h$. The process by which from DS we get to DS^* follows an incremental approach, whereby a strategy DS' is generated from a risk model M', that in turn is obtained by removing from M some threat $\langle t, c\rangle$ having c under a threshold value h^*, where M is the risk model which originated DS; now, if $r' < h$ then $DS^* = DS'$, otherwise the same process is applied to M' until a strategy characterized by a redundancy degree $< h$ is found.

The policy to decide which threats, among those having $c < h^*$, must be removed from a risk model, can be chosen by the user at enrollment time. For instance, criteria for threat removal can be derived from the metrics used in the NVD to characterize vulnerabilities. As an example, the user may want to remove vulnerabilities having *low* impact on the target system as first filtering method, and if this does not provide a strategy with redundancy degree $< h$, then s/he can request that vulnerabilities having *medium* impact be removed as well.[1] Similarly, the vulnerability *consequences*, i.e., the security area that is violated when a vulnerability is exploited (e.g., integrity, availability, etc.), can be used as an alternative to, or in cascade with, the aforementioned impact-based approach. In this case, the user may be more interested in keeping modules for the detection of attacks to the integrity of data rather than for those that threaten the availability of IoT devices. As well, the user can choose to apply specific policies to routers, and different ones to other kind of devices. In general, whatever policy for threat removal which provides a good trade-off between effectiveness and efficiency can be adopted.

The service provider maintains the list of enrolled security devices $s_1, s_2, \ldots, s_n$ along with the detection strategies $DS_1, DS_2, \ldots, DS_n$ they enforce. When a security device s_i delivers a new feature discovery report, the service provider determines a new detection strategy $DS_i^{'}$ and sends s_i the sets DM^{in} and DM^{out}, where:

- $DM^{in} = DS_i^{'} \setminus DS_i$ is the set of detection modules to install;
- $DM^{out} = DS_i \setminus DS_i^{'}$ is the set of detection modules to uninstall;

[1]In NVD, vulnerabilities are classified as *critical, high, medium* or *low*, to quantify the impact they have on a target system when they are exploited.

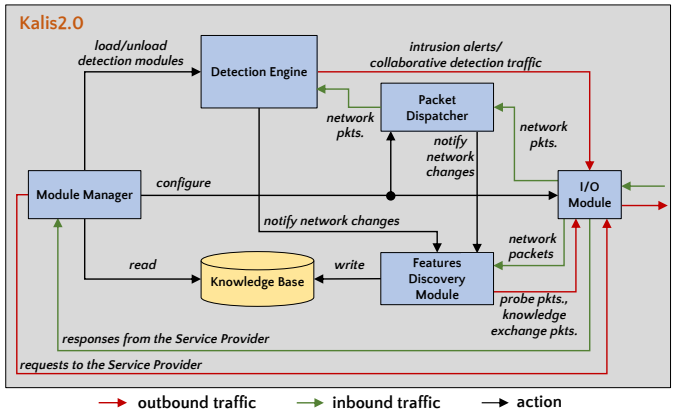


Fig. 5. The architecture of Kalis2.0.

Note that either DM^{in} or DM^{out} can be empty, and that DM^{in} contains the (encrypted) executable code of the detection modules to install, while DM^{out} contains the identifiers of the detection modules to uninstall.

VIII. Kalis2.0 – A Service-oriented Software Architecture for Security Devices

We hereby present Kalis2.0, a service oriented architecture for a context-aware, self-adaptive, network-based, centralized IDS for the IoT. Kalis2.0 enables security devices to enforce the intrusion detection as-a-service approach at the local layer by implementing the feature discovery and the intrusion detection tasks.

Figure 5 depicts the architecture of Kalis2.0, which consists of 6 main components: the *I/O Module* routes inbound traffic to the other architectural components, and outbound traffic (generated for feature discovery) to the IoT devices under monitoring using the appropriate physical layer protocol; the *Feature Discovery Module* enforces the feature discovery techniques described in Section V; the *Module Manager* is responsible for the configuration of the other architectural components; the *Detection Engine* enforces the detection strategy; the *Packet Dispatcher* forwards network packets to the detection modules in execution; and the *Knowledge Base* holds the feature discovery report generated by the Feature Discovery module. Kalis2.0 enables collaborative detection to be performed by security devices monitoring the same IoT system, that is, distributed detection tasks can be carried out by exchanging locally collected knowledge and network events relevant to the security goal.

In the following we describe in detail each component and how they interact with each other.

1) Module Manager: The Module Manager is in charge of setting up the whole system to operate in accordance with the IoT environment being monitored. It reads the Knowledge Base at regular time intervals, and if its content has changed with respect to the last access, the Module Manager connects with the service provider for sharing the new collected knowledge. This responds with a new detection strategy, which entails the configuration of the other architectural components. In particular, the Detection Engine is updated with the new detection strategy; the I/O Module is set up to enable the communication with new IoT devices, if any; and the Packet Dispatcher is configured so as each detection module is fed only with network packets originated from, or destined to the devices they are monitoring.

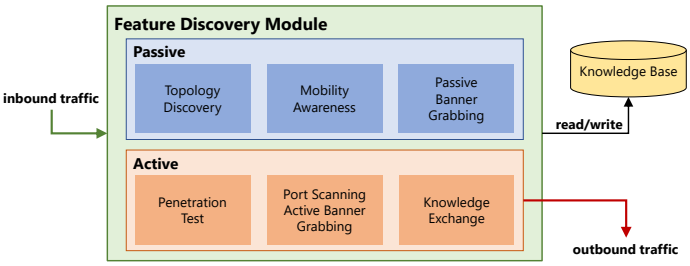


Fig. 6. The Feature Discovery module.

2) Feature Discovery Module: The Feature Discovery Module (Figure 6) enforces the autonomous knowledge discovery mechanism of Kalis2.0. As explained in Section V, feature discovery can be performed either in a passive and an active manner. The passive mode provides knowledge to be extracted from the sniffed network traffic, the active one, instead, enforces penetration testing, port scanning and active banner grabbing techniques. The submodules that work in passive mode are: the *Topology Discovery* module, which reconstructs the topology of the network under monitoring; the *Mobility Awareness* module, which detects mobility based on some criteria such as the changing of devices' signal strength beyond a certain threshold; ad the *Passive Banner Grabbing* module, which analyzes the data contained in the sniffed packets.

The active knowledge extraction concerns penetration tests and active banner grabbing on the IoT devices under monitoring, as well as the generation (detection) of advertisement packets to manifest itself to (detect the presence of) other security devices to share collected knowledge with. Sharing knowledge among different security devices can enable the discovery of features that single nodes cannot find out locally. As an example, being aware that other security devices are noticing changes in signal strength for specific IoT devices can enable a security device to correlate such changes with those experienced locally and detect mobility in the network with higher confidence.

The information collected either in the active and passive modes are stored into the Knowledge Base. In particular, for each IoT device the IP address, the protocols it uses, the signal strength, the list of devices it communicates with, the Operating System/firmware version, its open ports with associated network services, and the vulnerabilities it is affected by are reported.

The discovery process is activated in three circumstances: *(i)* a timeout expires; *(ii)* the Packet Dispatcher detects a new IP address; *(iii)* a detection module receives no packets for a long time. In the first case, the timeout duration is fixed, and the timeout is reset every time one of the other two circumstances occurs. The second and third cases occur when some device joins, and is removed from the network,

respectively. In this cases the discovery task is re-executed in order to update the network topology, and to figure out which technologies are being used by the new devices, if any.

When the discovery process terminates, the Feature Discovery module updates the Knowledge Base by replacing the obsolete data with the fresh one.

3) I/O Module: The I/O module handles inbound and outbound traffic. Inbound traffic can be of three types: *(i)* traffic originated by the monitored IoT network, which is forwarded to the Packet Dispatcher for detection purposes, and to the Feature Discovery module for discovery purposes; *(ii)* inbound connections coming from the service provider, that are routed to the Module Manager; and *(iii)* packets originated by other security devices either for manifesting their presence, or for exchanging collected features and network events, that are forwarded to the Feature Discovery module in the first and second cases, and to the Packet Dispatcher in the third case.

Outbound traffic can be of four types: *(i)* intrusion alerts generated by the detection modules in execution; *(ii)* outbound connections established by the Module Manager to transmit the collected knowledge to the service provider; *(iii)* probe packets originated by the Feature Discovery module; and *(iv)* traffic originated by the collaborative detection modules and destined to other security devices. Outbound connections require a preliminary configuration phase to be performed. This is done by the Module Manager which provides the I/O Module with a set of pairs $\langle dest\_IP, PHY\_protocol \rangle$. When the I/O Module receives an outbound connection request by another architectural component, it looks for a matching between the destination IP carried into the request and the $dest\_IP$ of one of the pairs it holds. If a match is found, it encapsulates network layer packets into physical layer packets according with the physical layer protocol the destinations device $dest\_IP$ uses. The pairs $\langle dest\_IP, PHY\_protocol \rangle$ are defined as a result of the feature discovery phase, and are stored into the Knowledge Base.

4) Packet Dispatcher: The Packet Dispatcher receives network packets from the I/O Module, and forwards them to the detection modules being executed by the Detection Engine. The Packet Dispatcher is configured by the Module Manager so that each detection module only receives packets strictly necessary for its detection task. When the Packet Dispatcher detects a packet with a new IP address, then it notifies the Feature Discovery Module, which in turn initiates a new discovery task to identify potential network changes.

5) Knowledge Base: The Knowledge Base holds all the information about the features of the monitored entities and networks, and makes these available to the Module Manager. We refer to an individual piece of knowledge as *knowgget* ("**know**ledge nu**gget**"). We model each knowgget as a label, describing the information represented, and its associated value. Each knowgget has a "creator" field – representing the security device that created it (useful for knowledge sharing). Knowggets may in turn be composed by several different pieces of data; for example the knowledge about the current traffic frequency (as packets per second) can include several sub-pieces of information for each different packet type, such as TCP SYN, TCP ACK or TinyOS CTP. We refer to these as *multilevel knowggets*. The label of a multilevel knowgget is thus not associated with a single value, but with a group of other knowggets, in a tree-like structure. A knowgget k is formally defined as a tuple $\langle l, v, s \rangle$, where l is the label, v is either a primitive value or a set of knowggets (for multilevel knowggets), and s is the identifier for the security device creator of k. Figure 7 shows an example of Knowledge Base.

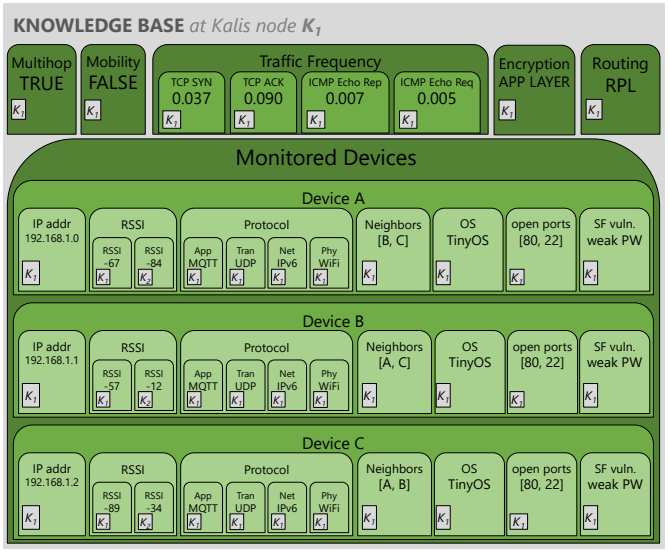


Fig. 7. An example of Knowledge Base with heterogeneous knowggets, each showing label, value, and creator field. The `RSSI` multilevel knowgget is composed of two knowggets generated by two different security devices.

To enable the knowledge-sharing mechanism, the Knowledge Exchange module sends the knowggets to other security devices, making sure to mark the appropriate identity in the "creator" field denoting the security device that generated the knowgget. At the same time it receives knowggets from other security devices. Note that this mechanism does not provide a way for a security device to overwrite or alter the Knowledge Base of another security device. When a security device, say s_1, receives a new or updated collective knowgget k from a different security device, say s_2, the Feature Discovery Module of s_1 checks whether the label *and* creator of k matches any existing knowgget in the Knowledge Base. Therefore, s_1 can only update those knowggets in s_2 that were originally generated by itself.

6) Detection Engine: The detection Engine enforces the detection strategy. Detection modules analyze the captured traffic and detect anomalies and security incidents. The Module Manager coordinates all the modules, loading/unloading them as needed, depending on changes in the Knowledge Base.

The deployment of two or more security devices in the same network area enables the execution of *collaborative* detection techniques. Each security device detects the presence of the other ones, and send this information to the service provider. This, in turn, will respond with a set of detection modules to be executed in a distributed way, which allows to detect security incidents that involve wide network areas. Collaborative detection modules perform a distributed detection task by exchanging locally collected knowledge and network events

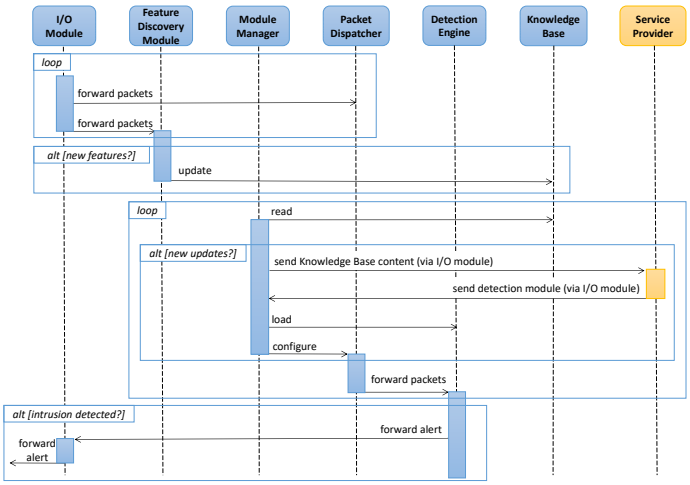


Fig. 8. Sequence diagram of Kalis2.0.

relevant for the security goal they pursue. The communication between collaborative detection modules requires the sender to broadcast detected network events along with the identifier of the detection module to which data is intended. Collaborative detection typically comes into play in large scale networks where attacks may affect a portion of the network larger than that monitored by a single security device. An example is the Wormhole attack in which an attacker captures packets at one point in the network by announcing a shorter route to the destination, tunnels them to another point far in the network, and then replays them from that point. If the two points belong to network areas monitored by different security devices, a single security device would not be able to detect such an attack unless it receives additional information from the security device which monitors the other side of the network.

A. Operational Phase

Figure 8 shows the sequence diagram of the operational phase of Kalis2.0. In particular, it depicts the situation in which the Feature Discovery module discovers a new network feature. This causes the interaction between the Module Manager and the service provider and the consequent installation of a new detection module. Then, the Module Manager configures the Packet Dispatcher to deliver incoming packets to the new detection module. Notice that the Packet Dispatcher does not deliver any packet before it has received instructions about how to do it from the Module Manager. The detection module starts working as soon as it receives incoming traffic from the Packet Dispatcher, and activates the I/O module to raise an alert following the detection of an intrusion. The detection module will work until the Module Manager unloads it. All the components in Kalis2.0 run independently. When a new packet is captured on any protocol, all the interested parties are asynchronously notified of the new packet event, and can independently and concurrently process the new information.

IX. Benefits of the Proposed Approach

The benefits of the proposed context-aware detection approach mainly reflect in the following aspects: *(i)* higher detection rate; *(ii)* higher detection accuracy; *(iii)* higher classification accuracy; and *(iv)* reduced risk of unauthorized disclosure of security algorithms. In the following subsections, each aspect is discussed with the help of practical examples of real IoT technologies. Next, in the experimental section we show the benefits of context-awareness with practical tests.

A. Higher Detection Rate

The knowledge of the features of the monitored IoT network allows to obtain higher detection performances in terms of detection rate. Indeed, the selection of the most appropriate detection method according with the network characteristics is crucial for ensuring that all ongoing attacks are detected, and insufficient enforcement of these criteria would limit the detection capabilities of an IDS. As an example, distinguishing factors include the ability to detect device mobility, as the various types of mobility patterns in IoT guide the way malware propagate [6] as well as the efficacy of other types of attacks. Mobile IoT finds successful application in areas such as healthcare, logistic, smart homes, smart cities, etc. In these scenarios, it is important to detect mobility patterns and determine the most appropriate security solutions that are effective for enforcing a high security level. Literature on IoT security provides numerous examples of security techniques for mobile IoT scenarios, such as [18], [23], [48].

Beside mobility, the knowledge of other types of features plays an important role as well. For instance, communication protocols at different layers of the protocol stack suffer from specific attacks that cannot be intercepted unless IDSes run detection techniques that are designed specifically for them. Examples are: the XMPPbomb attack to the XMPP protocol, NDP spoofing attack to the NDP protocol, Version Number attack, Rank attack and Worst Parent attack to the RPL protocol, and so on.

As a further example, consider the Contiki-NG Operating System which, depending on the specific OS version, is subject to buffer overflow attacks at different layers of the protocol stack, in particular at the Physical layer with the 6LowPan protocol (CVE-2022-36054), at the Transport layer with the TCP protocol (CVE-2022-36054), and at the Network layer with the RPL protocol (CVE-2022-35927). In all these cases, without knowledge about the protocol, the OS, and the OS version, a buffer overflow attempt is unlikely to be detected.

In general, there are many aspects of IoT systems that can be exploited to determine the most effective detection strategy. For all of them, selecting the most appropriate security techniques can avoid attacks that occur under specific circumstances to go undetected.

B. Higher Detection Accuracy

The knowledge of the enabling technologies allows for a more accurate detection task. For instance, an IDS may confuse legitimate protocol activities with malicious actions

when it does not have in-depth knowledge of how the protocols work, thus the knowledge of protocols behavior may help limiting the number of false positives. For example, the RPL, AODV and DSR protocols require network nodes to drop received packets when they do not satisfy certain requirements. With this in mind an IDS can lower the number of false positives because more likely to distinguish between malicious and benign drops.

C. Higher Classification Accuracy

The knowledge about network features can be used to distinguish attacks sharing similar symptoms, therefore hard to distinguish to a passive external observer. Recognizing the true nature of an attack can be of paramount importance, especially when attack reaction mechanisms are to be initiated. For instance, both *ICMP Flood* and *Smurf* attacks show a high amount of ICMP Echo Reply messages sent to a victim node. However, *Smurf* attacks cannot be carried out in single-hop networks, and this information represents the distinguishing factor that allows to recognize the true nature of the ongoing attack.

Another fitting example are malware. Malware classification is a critical task as it drives the clean up process of infected devices. In fact, the method used to sanitize a device relies on where the infection is located on the device itself. As shown in Table VIII, after infection a malware can hide in the device memory, in the device file system, or in the device firmware. The infected device must be restarted to remove malware that reside in memory, formatted to remove malware that reside in the file system, while a firmware update is necessary to remove malware that reside in the device firmware.

TABLE VIII
IOT MALWARE CLASSIFIED BASED ON THE LOCATION OF THE INFECTION

Memory-Resident	File System-Resident	Firmware-Resident
Mirai	Bashlite	VPNFilter
Reaper (IoTroop)	Remaiten	Lillin Scanner
Masuta (PureMasuta)	Torii	TheMoon
Persirai	Linux.MulDrop.14	RubyMiner
Hajime	BrickerBot	Anarchy
Gafgyt	ELF Linux/Mirai	Amnesia
Hide 'N Seek (HNS)	ELF Linux/IRCTelnet	AESDDoS
Okiru	Kaiten (KTN)	Linux.Darlloz
Satori	LuaBot	Linux.ProxyM
Tsunami	Mukashi	Linux.Moose
Echobot	Manga	Ttint
	FBot	Sbidiot

Uniquely identifying a malware by cross-referencing device features and observed symptoms, however, may not always be possible for two main reasons: first, the feature discovery task may not retrieve all relevant information about the monitored devices (e.g., the device model is known but its firmware version is not), and second, malware may be indistinguishable due to some commonalities (e.g., the majority of malware perform TCP flood based DDoS attacks). Nonetheless, it is often possible to narrow the range of possibilities by excluding malware that do not cause the observed symptoms and that target devices with characteristics different from those collected. The device model or manufacturer, in particular, are features that significantly contribute to this assessment: when certain network activities involve specific device models, there is a good chance of determining the associated malware class.

As an example, consider a Huawei router with the port Telnet 23 open as the set of collected features. The detection of patterns that represent a dictionary attack on port 23 may be the symptom of an ongoing malware infection. However, this knowledge is not sufficient to uniquely identify the type of malware since there exist many of them, especially Mirai variants, that perform dictionary attacks on the Telnet port 23. The further detection of outgoing traffic towards a blacklisted IP address on the TCP port 7000 is another evidence of the presence of a malware exchanging messages with its C&C server. In this case, this additional symptom allows to restrict the range of possibilities. Indeed, the only (known) malware that affects Huawei router via Telnet port 23 and managed via TCP port 7000 is Satori, which is memory-resident and thus removable via device rebooting.

D. Reduced Risk of Unauthorized Disclosure of Security Algorithms

When developing an IDS, it's crucial to guarantee the anonymity of the detection methods. In fact, if malicious users manage to obtain the implementation details, they may be able to devise and carry out detection-proof attacks with success and remain undetected. In addition, there's a chance that the implementation details would be made public, which would cause significant financial losses for the company that owns the revealed technologies.

Even while security devices may have firmware upgrades and their entire file system encrypted, attackers still have a chance of learning the encryption keys. For instance, they can perform a brute force attack to guess the encryption key and then decrypt the content of the hard disk. Alternatively, they can perform reverse engineering from the executable code in the central memory (which is in cleartext), and reconstruct the security techniques in execution.

Present-day commercial products are prone to this kind of attacks. Security devices that host Kalis2.0, instead, are initially "vacant", and subsequently "stuffed" by the service provider with a selection of detection modules according with the features of the network under monitoring. This mode ensures higher confidentiality for the organization which holds the rights of the security techniques. Indeed, in order to acquire necessary information, an attacker would be compelled to carry out a number of onerous jobs: first, s/he has to deploy a Kalis2.0 node into an IoT network, and wait for it to download the detection modules from the service provider; second, s/he would be able to disclose the detection modules selected for that specific IoT network only; third, s/he must deploy the Kalis2.0 node into a network with different characteristics in order to gain further information. Thus, since the set of security techniques is restricted by the actual network features, an attacker cannot disclose more than what a Kalis2.0 node actually uses to monitor the current network.

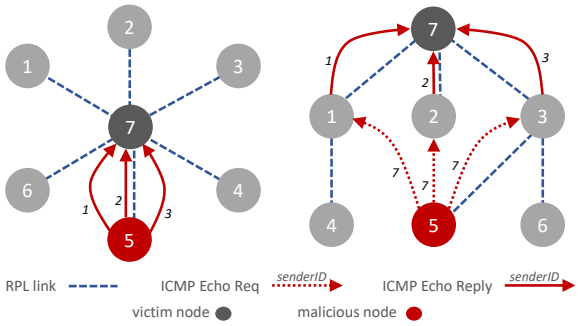


Fig. 9. ICMP Flood attack (left) vs. Smurf attack (right).

X. Experimental Results

We performed a wide set of experiments to show the benefits of the proposed context-aware approach compared with a non-context-aware IDS (NCA-IDS). To this end, we carried out a number of attacks that are representative of situations where context awareness offers better detection performance, in particular:

- attacks that present identical or very similar symptoms, and thus that are difficult to classify correctly (Section X-A);
- attacks for which the detection technique to use depends on specific network features (Section X-B);
- attacks that can be confused with legitimate network activities (Section X-C);
- attacks for which their detection is easier when the knowledge of specific protocol features is exploited (Section X-D);

In these settings, we show how Kalis2.0 provides better detection performances than the NCA-IDS, basing its decisions on both **network** and **device features** such as **network topology**, **device mobility**, **protocols**, and **intrusion prevention systems**.

The two systems are evaluated in terms of: **detection rate** – percentage of adverse events detected out of all the adverse events in the test scenario; **classification accuracy** – percentage of correctly classified attacks out of all the detected attacks; **false positive rate** (where applicable) – percentage of false alarms out of all the benign events in the test scenario; **CPU usage**; and **RAM usage**.

A. Attack scenario 1: Same symptoms different threats – the case of ICMP Flood and Smurf attacks

1) Description: The knowledge about network features may be decisive for the correct classification of detected attacks under the countermeasures that have to be performed. A well fitting example is the case of ICMP Flood and Smurf attacks that, even though they show same symptoms, can be distinguished from each other because of the topology of the target network.

In an ICMP Flood attack (Figure 9-left) the attacker node floods the victim with ICMP Echo Reply messages using several different identities as sender. In a Smurf attack (Figure 9-right), the attacker sends ICMP Echo Request messages to several neighbors of the victim using the victim's identity as sender; those neighbors will thus respond with ICMP Echo Reply messages directed to the victim. We note that, to an external observer these two attacks show the same symptom, that is a high amount of ICMP Echo Reply messages sent to the victim node. However, in contrast to the ICMP Flood attack, the accomplishment of a Smurf attack needs a specific network topology to be in place, that is the one with a number of network nodes close enough to both the malicious node and the victim, that can act as reflectors of malicious traffic. In this test, this knowledge is leveraged by Kalis2.0 to achieve an accurate attack classification.

2) Implementation: We use the Cooja simulation environment [35] to simulate an IoT system made of 7 TelosB motes with the TinyOS Operating System, the BLIP protocol stack [32] (the Berkeley Low-power IPv6 stack for TinyOS), and the TinyRPL implementation of the RPL routing protocol. The nodes are placed in a 2000 m^2 area, and feature a communication rate of *1 pkt/sec*. The simulation time is of 15 minutes.

We implemented two distinct network instances: in the first one (we call it *star*), IoT nodes are too far apart to communicate with each other, thus the RPL protocol builds a star topology in which each node communicates with the sink only (Figure 9-left); in the second network instance (we call it *tree*), instead, IoT nodes are close enough to each other for the RPL protocol to set up a tree-like topology (Figure 9-right). In both networks a node acts as a sink.

In these settings, we note that the Smurf attack can only be accomplished in the *tree* network where the malicious node is close enough to other nodes to send them forged ICMP Echo Request messages. The ICMP Flood attack, instead, can be successfully performed in both networks as the malicious node only needs to reach the victim, that is the sink node in the *star* network, and any node falling in its action range in the *tree* network. In light of this, 30 Smurf attacks and 30 ICMP Flood attacks are performed on the *tree* network, whereas the *star* network experiences 30 ICMP Flood attacks and no Smurf attacks. For each attack instance the malicious node is chosen at random.

Both Kalis2.0 and the NCA-IDS are set with either the ICMP Flood and the Smurf attack detection modules active.

3) Results: In both network scenarios either Kalis2.0 and the NCA-IDS detect all attacks, demonstrating a 100% detection rate, however, in the *tree* network they are unable to recognize the attack class since for every attack instance, regardless of which one, either an ICMP Flood attack and a Smurf attack alarm is raised. In the *star* network, instead, Kalis2.0 leverages its knowledge of the current network topology to disable the Smurf attack detection module, and as a consequence it is able to correctly classify all the ICMP Flood attacks, while the NCA-IDS keeps running both modules due to its inability to match its detection capabilities with the context.

For what concerns computing resources, Kalis2.0 is more efficient than the NCA-IDS when monitoring the *star* network because it executes one detection module only (for the detection of ICMP Flood attack), while the NCA-IDS keeps running both modules. In contrast, the NCA-IDS uses less computing

resources than Kalis2.0 when monitoring the *tree* network because, besides the two detection modules, Kalis2.0 runs other architectural components in background, in particular the Topology Discovery module and the Packet Dispatcher. Table IX summarizes the obtained results.

TABLE IX
PERFORMANCES COMPARISON FOR ICMP FLOOD AND SMURF ATTACKS

	tree network		star network	
	NCA-IDS	**Kalis2.0**	**NCA-IDS**	**Kalis2.0**
detection rate	**100%**	**100%**	**100%**	**100%**
classification accuracy	0%	0%	0%	**33%**
CPU usage	**0.55%**	0.61%	0.49%	**0.35%**
RAM usage	**16406 Kb**	18406 Kb	15707 Kb	**13809 Kb**

B. Attack scenario 2: Same threat different detection methods – the case of the Version Number attack

1) Description: The RPL protocol (IPv6 Routing Protocol for Low-Power and Lossy Networks) creates a network topology based on the concept of Destination Oriented Directed Acyclic Graph (DODAG), which defines a tree-like, loop-free structure that outlines paths between nodes. A network can have one or more DODAG at once, which operate together to generate an *RPL Instance*. The combination of *RPLInstanceID*, *DODAGID*, and *DODAGVersion* identifies a DODAG.

By sending a DIO message (Destination Information Object) to the neighbor nodes, the root node initiates the DODAG formation. The neighbors figure out how much it will cost to join the DODAG and create a path that leads to the root. The adjacent nodes multicast DIO messages until the DODAG is formed. The network becomes stable after the DODAG is established. When some changes in the network occur, such as a node changing its location, the formation of a new DODAG with a higher version number is initiated.

The Version Number Attack (VNA) consists in a malicious node sending a larger version number in the DIO message, breaking off the network's consistency. When the root node gets a DIO message with a new version number, it initiates a global repair operation to re-set the DODAG, which involves all the network nodes.

2) Implementation: For this evaluation we use the Cooja simulation environment, where 30 TelosB motes with the same characteristics of those described in the previous experiment are randomly distributed in a 200 m^2 area. To handle nodes mobility, we use the Cooja-Mobility-Plugin, with node speeds set to 0.5 m/s. The network behavior alternates from static to mobile for 50 times (25 static and 25 mobile), keeping the same behavior for 1 minute during which one VNA is performed at random times. The whole experiments lasts 50 minutes.

Many detection techniques exist for the VNA attack, however each one is specific to a network with certain characteristics, e.g. mobility. In this evaluation, we provide two different detection modules for VNA attacks, one suitable for static networks [34], and the other for mobile networks [48]. The NCA-IDS randomly selects one of the two modules for each of the 50 experiment runs, closely simulating a static module library configuration that does not adapt to the changes in network features. Kalis2.0, instead, leverages the knowledge by the Mobility Awareness module, and dynamically selects the appropriate detection technique for the current network mobility setting. To infer node mobility we use the Receive Signal Strength Indicator (RSSI), which is supported by Cooja.

3) Results: The NCA-IDS misses attacks when the active module is not the one suitable for the current mobility profile of the network. Kalis2.0, instead, uses the right module in the majority of cases, making mistakes only when the attack starts immediately before or immediately after the moment in which the network switches from mobile to static, and vice-versa, since the Mobility Awareness module requires some seconds to detect the change.

Compared to the NCA-IDS, Kalis2.0 uses more computing resources due to the execution of the Mobility Awareness module which activates 50 times, one for each experiment run. Table X summarizes the obtained results.

TABLE X
PERFORMANCES COMPARISON FOR THE VERSION NUMBER ATTACK

	NCA-IDS	**Kalis2.0**
detection rate	34%	**96%**
classification accuracy	**100%**	**100%**
CPU usage	**2.59%**	4.26%
RAM usage	**44862 Kb**	61998 Kb

C. Attack scenario 3: Symptoms disambiguation – the case of the Selective Forwarding attack

1) Description: In Selective Forwarding attacks, a malicious device forwards only a fraction of the received packets. A widely used approach against dropping attacks is the *watchdog* [24], whereby a security device overhears the packets sent by the nodes falling in its action range, and launches an alarm as soon as a node does not forward a received packet. Some routing protocols (e.g., RPL, AODV and DSR) enable nodes to discard routing control packets when these do not satisfy certain requirements. An effective watchdog would leverage the knowledge of the routing protocol in use in order to not confuse packets dropped by malicious nodes with those dropped by the protocol itself.

The RPL routing protocol implements an optional delay/replay protection mechanism, whereby a node discards a packet if it does not pass the delay/replay check, i.e., if it is delivered after a certain time threshold (delay), or if it is a copy of a packet already delivered in the past (replay). An incremental counter in the header of RPL secure control messages is sequentially increased for each transmission, and used for message replay attack protection. The counter can alternatively store a timestamp in which case delay protection is provided as well. According with [4], upon the reception of a RPL secure control message, "*if the timestamp indicates a transmission time prior to the locally maintained transmission time counter for the originator address, a replay violation is indicated and a node must discard the packet. If the received timestamp indicates a message transmission time that is earlier than the current time less the acceptable packet delay, a delay violation is indicated and the node must discard the incoming packet*".

2) Implementation: For this evaluation we use a real testbed made of 10 Sky Mote XM1000 sensor boards with the Contiki-NG Operating System. Contiki-NG provides an implementation of the IoT protocol stack with the CoAP protocol at the application layer, UDP at the transport layer, IPv6/ICMPv6 at the network layer, RPL as routing protocol, and 6LoWPAN at the adaptation layer. One mote is connected via USB to a laptop to act as a sink (Figure 10). The laptop runs an instance of the Kali Linux OS [16] which is an open-source, Debian-based Linux distribution, which offers several hacking tools. The laptop also hosts Kalis2.0 and the NCA-IDS, that use the Wireshark sniffing tool to capture and inspect exchanged packets, which comes as a Kali Linux facility.

After node deployment, the RPL routing protocol takes about 15 seconds to set the network topology as depicted in Figure 10. The attacking node $n1$ is programmed to randomly drop 30% of packets (either application and RPL packets) coming from nodes $n3$ and $n4$. These, in turn, are programmed to delay 30% of the RPL messages coming from their child nodes, in order to make $n1$ dropping such packets according with RPL specifications. The delay time ranges from 0.1 to 1 *sec*. The whole experiment lasts 10 minutes.

Both IDSes run a detection module which works according with the watchdog principle, that is, it computes the maximum transmission time $MTT_{c\to p}$ for each link between a child node c and its parent p during a 10 *sec* attack-free period, as the maximum time between when c sends a packet to p and when p forwards the same packet to its parent, and raises an alarm as soon as a node does not forward a received packet within $MTT_{c\to p}$ (we call this module simply *Watchdog*). Additionally, Kalis2.0 is provided with a further detection module (we call it *WatchdogRPL*) to be executed in parallel with *Watchdog* if the RPL routing protocol is in use. *WatchdogRPL* exploits the RPL built-in delay/replay protection mechanism described above as a mean to distinguish between malicious and benign drops, in particular: It computes $MTT_{c\to p}$ as explained above, then for each child node c it stores the timestamps t_c^i and t_c^{i+1} of the last two forwarded RPL packets RPL_c^i and RPL_c^{i+1}, respectively.[2] When the parent node p does not forward upward a RPL message, *WatchdogRPL* verifies whether the packet drop complies with the RPL specifications described above, and if not it raises an alarm. In particular, it checks whether the packet drop is not the result of the RPL delay check, i.e. if $T > t_c^{i+1} \geq T - MTT_{c\to p}$, neither the result of the RPL replay check, i.e. if $t_c^{i+1} > t_c^i$, where T is the actual time. If both verifications are successful then it concludes that p is arbitrarily dropping packet RPL_c^{i+1}.

3) Results: Both Kalis2.0 and the NCA-IDS detect almost all Selective Forwarding attacks, however the NCA-IDS generates some false positives because it raises an alarm for every benign drop. In contrast, Kalis2.0 enables *WatchdogRPL* to work in parallel with *Watchdog*, which allows to distinguish between malicious and benign drops. The Packet Dispatcher is instructed to forward RPL packets to the *WatchdogRPL* module, and application packets to the *Watchdog* module.

[2]RPL nodes use Destination Advertisement Object (DAO) messages to make known the routing tables of their descendants to their parent, which in turn, updates and forwards them upwards.

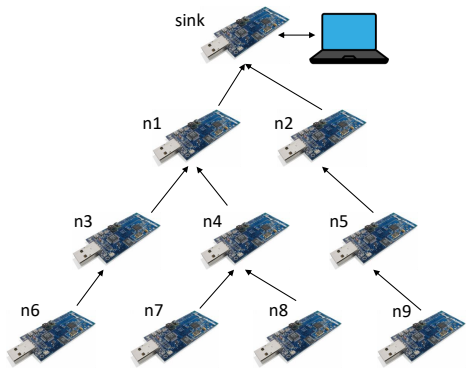


Fig. 10. Testbed with 10 Sky Motes X1000, one is connected to a laptop to act as a sink.

With these settings, the *Watchdog* module is hindered from handling RPL packets, which prevents it from raising false alarms like the NCA-IDS does. As a result, Kalis2.0 raises an alarm when the *WatchdogRPL* module reports the detection of a RPL packet drop, or when the *Watchdog* module reports the detection of an application packet drop. In few cases either Kalis2.0 and the NCA-IDS miss attacks because of the coarse value of the maximum transmission time $MTT_{c\to p}$. Indeed, $MTT_{c\to p}$ is computed over the first 10 seconds of network operation, which is a limited time period compared with the duration of the whole experiment, and thus the actual transmission time between a child node and its parent is sometimes higher.

Kalis2.0 uses much more computing resources than the NCA-IDS due to the execution of two detection modules in place of one, in addition to the other architectural components. Table XI summarizes the obtained results.

TABLE XI
PERFORMANCES COMPARISON FOR THE SELECTIVE FORWARDING ATTACK

	NCA-IDS	**Kalis2.0**
detection rate	**97%**	**97%**
classification accuracy	**100%**	**100%**
false positive rate	11%	**0%**
CPU usage	**0.87%**	2.06%
RAM usage	**18442 Kb**	2966 Kb

D. Attack scenario 4: Exploiting protocol features – the case of Delay and Replay attacks

1) Description: During a Delay attack, a malicious node delays the packets it receives from its neighbors in order to slow down the network operations. In a Replay attack, a malicious node stores received packets to forward them later in the future. Both delayed and replayed packets may fool benign nodes which, as a result, may behave incorrectly with respect to network rules. As explained in the previous attack scenario, the RPL routing protocol provides a sort of protection against delay/replay violations, whereby delayed and replayed RPL packets are dropped. In this test, the knowledge of this protocol mechanism is exploited by Kalis2.0 to provide better detection performances with respect the NCA-IDS.

2) Implementation: For this evaluation we use the testbed depicted in Figure 10 which we already described in the previous attack scenario. Node $n3$ is programmed to delay 30% of the packets coming from its child nodes, while node $n4$ is programmed to replay one packet randomly chosen among those received from its child nodes in the 30% of transmissions to $n1$. The whole experiment lasts 10 minutes.

The NCA-IDS runs a Replay attack detection module which stores in a buffer the packets sent by each node, and raises an alert if any of the packets sent at a later time are the same as one in the buffer. As the buffer has a fixed size, once it is filled up, it is updated according to a First-In-First-Out (FIFO) policy. The Delay attack detection module computes the maximum transmission time $MTT_{c\to p}$ for each link between a child node c and its parent p during a 10 *sec* attack-free period as the maximum time between when c sends a packet to p and when p forwards the same packet to its parent. At runtime, it computes the current transmission time $CTT_{c\to p}$ for each transmission between a pair of nodes (c, p), and raises an alarm whenever $CTT_{c\to p} > MTT_{c\to p}$.

Kalis2.0 is provided with the same detection algorithms as those used by the NCA-IDS, plus two additional modules (we call them *DelayRPL* and *ReplayRPL*), that handle RPL packets only, while non-RPL packets are handled by the other modules. The Packet Dispatcher takes care of forwarding sniffed RPL packets to the *DelayRPL* and *ReplayRPL* modules. The *ReplayRPL* module stores the timestamps t_c^i and t_c^{i+1} of the last two forwarded RPL packets RPL_c^i and RPL_c^{i+1}, respectively, for each child node c, and raises an alarm if $t_c^{i+1} \leq t_c$. The *DelayRPL* module computes the maximum transmission time $MTT_{c\to p}$; stores the timestamps t_c^i and t_c^{i+1} of the last two forwarded RPL packets RPL_c^i and RPL_c^{i+1}, respectively; and raises an alarm if $t_c^{i+1} < T - MTT_{c\to p}$, where T is the actual time.

We recall that the timestamp is available in RPL packets' headers, while it is not in lower layers' packets, and as a consequence an IDS would not be able to exploit this information unless provided with context-aware capabilities.

3) Results: Both the NCA-IDS and Kalis2.0 detect almost all Delay attacks, providing a 95% and a 97% detection rate, respectively, however Kalis2.0 is more efficient in getting the desired goal. Indeed, while the NCA-IDS must compute the current transmission time $CTT_{c\to p}$ for each transmission between each pair child-parent (c, p), Kalis2.0, in contrast, only reads the *timestamp* feature of RPL packets' header, which entails much less computation. Both IDSes do not achieve 100% detection rate because of the coarse value of $MTT_{c\to p}$ which is computed over the first 10 seconds of network operation, thus resulting smaller than the actual transmission time during 5% of attacks. Kalis2.0 has a 2% detection rate higher than NCA-IDS since delayed RPL control packets are handled by the *DelayRPL* module which does not rely on $MTT_{c\to p}$.

For what concerns the Replay attack, the NCA-IDS and Kalis2.0 provid a 96% and a 97% detection rate, respectively. The detection module adopted by the NCA-IDS maintains a buffer to store network packets sent by all nodes, which is memory consuming, and compares the current packet with all packets in the buffer, which is time and CPU consuming. Furthermore, since the buffer has limited size, it is constantly updated according to a FIFO policy and as a consequence, the detection module misses replayed packets that are no longer stored in the buffer, which is the reason for the 4% and 2% of false negatives generated by NCA-IDS and Kalis2.0, respectively. In contrast, Kalis2.0 is more efficient in handling RPL packets since the *ReplayRPL* only needs to read and compare timestamp values pairs without having to store and compare entire network packets. For the same reason Kalis2.0 has a 1% detection rate higher than NCA-IDS. Indeed, the *ReplayRPL* module does not use a buffer to store network packets, thus missing replayed packets that are not currently buffered is not an issue with it. Table XII summarizes the obtained results.

TABLE XII
PERFORMANCES COMPARISON FOR DELAY AND REPLAY ATTACKS

	delay attack		**replay attack**	
	NCA-IDS	**Kalis2.0**	**NCA-IDS**	**Kalis2.0**
detection rate	95%	**97%**	96%	**97%**
classification accuracy	**100%**	**100%**	**100%**	**100%**
CPU usage	1.39%	**0.72%**	1.95%	**0.35%**
RAM usage	13608 Kb	**1406 Kb**	21309 Kb	**1839 Kb**

XI. CONCLUSIONS

In this paper we presented a context-aware, self-adapting, SECaaS-based IDS for the IoT, able to collect information about its surroundings and adapt its behavior accordingly. As opposed to most context-aware-based approaches, whereby the context is built upon historical data, we defined the context in terms of system features. We proposed a hybrid SECaaS-based model composed of a cloud layer and a local layer, whereby a service provider (cloud layer) takes care of assessing the risk based on the system features discovered by the security devices (local layer) it interacts with, and of deploying them the most suitable intrusion detection techniques to perform in the monitored IoT networks.

For the cloud tier, we defined *(i)* a method to quantify the risk as a set of threats, each associated with the likelihood of affecting the monitored devices based on the information collected about the devices themselves, and *(ii)* a method to determine the best detection strategy based on the assessed risk as a set of detection algorithms with a good trade-off between effectiveness and efficiency. For the local tier, we showed *(i)* how to discover device and network features and how these can be leveraged to collect further information useful for the selection of the best detection strategy, and *(ii)* Kalis2.0, a service oriented software architecture which dialogues with the cloud layer and performs context-aware intrusion detection.

Finally, we discussed the advantages of using our context-aware approach, which were also illustrated by the findings of a series of experiments that show how effective context-aware based detection is compared to non-context-aware detection in terms of detection accuracy, attack classification accuracy, and computing resource usage.

As future work we aim at extending our framework with an Artificial Immune System to allow the IDS autonomously evolve hand in hand with attack patterns over time.

REFERENCES

[1] Z. A. Almusaylim, N. Jhanjhi, and A. Alhumam. Detection and mitigation of rpl rank and version number attacks in the internet of things: Srpl-rp. *Sensors*, 20(21):5997, 2020.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In *26th {USENIX} security symposium ({USENIX} Security 17)*, pages 1093–1110, 2017.

[3] Bitdefender. Bitdefender Box. https://www.bitdefender.com/smart-home/. Accessed: 2023-11-13.

[4] A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. Rfc 6550: Rpl: Ipv6 routing protocol for low-power and lossy networks, 2012.

[5] Carnegie Mellow University. CERT. https://www.kb.cert.org/vuls. Accessed: 2023-11-13.

[6] S.-M. Cheng, P.-Y. Chen, C.-C. Lin, and H.-C. Hsiao. Traffic-aware patching for cyber security in mobile iot. *IEEE Communications Magazine*, 55(7):29–35, 2017.

[7] CNET. Norton Core. https://www.cnet.com/products/norton-core/specs/. Accessed: 2023-11-13.

[8] E. de Matos, R. T. Tiburski, L. A. Amaral, and F. Hessel. Providing context-aware security for iot environments through context sharing feature. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 1711–1715. IEEE, 2018.

[9] F-Secure. SENSE. https://community.f-secure.com/sense-router-en. Accessed: 2023-11-13.

[10] J. J. Gondim, R. de Oliveira Albuquerque, and A. L. S. Orozco. Mirror saturation in amplified reflection distributed denial of service: A case of study using snmp, ssdp, ntp and dns protocols. *Future Generation Computer Systems*, 108:68–81, 2020.

[11] I. Gulatas, H. H. Kilinc, A. H. Zaim, and M. A. Aydin. Malware threat on edge/fog computing environments from internet of things devices perspective. *IEEE Access*, 11:33584–33606, 2023.

[12] K. Hameed, S. Garg, M. B. Amin, B. Kang, and A. Khan. A context-aware information-based clone node attack detection scheme in internet of things. *Journal of Network and Computer Applications*, 197:103271, 2022.

[13] A. Hassanzadeh and R. Stoleru. On the optimality of cooperative intrusion detection for resource constrained wireless networks. *Computers & Security*, 34:16–35, 2013.

[14] IoT-Inc. ZingBox. https://www.iot-inc.com/portfolio_page/zingbox/. Accessed: 2023-11-13.

[15] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. Unviersity. Contexlot: Towards providing contextual integrity to appified iot platforms. In *NDSS*, volume 2, pages 2–2. San Diego, 2017.

[16] Kali. Kali Linux. https://www.kali.org/docs/introduction/what-is-kali-linux. Accessed: 2023-11-13.

[17] P. Khanpara, K. Lavingia, R. Trivedi, S. Tanwar, A. Verma, and R. Sharma. A context-aware internet of things-driven security scheme for smart homes. *Security and Privacy*, page e269, 2022.

[18] I. Kotenko, I. Saenko, and A. Branitskiy. Framework for mobile internet of things security monitoring based on big data processing and machine learning. *IEEE Access*, 6:72714–72723, 2018.

[19] P. Kumar, G. P. Gupta, and R. Tripathi. A distributed ensemble design based intrusion detection system using fog computing to protect the internet of things networks. *Journal of ambient intelligence and humanized Computing*, 12:9555–9572, 2021.

[20] H. Lin, Q. Xue, J. Feng, and D. Bai. Internet of things intrusion detection model and algorithm based on cloud computing and multi-feature extraction extreme learning machine. *Digital Communications and Networks*, 2022.

[21] C. M. Liu, R. Chen, and C. Chen. An artificial immune-based distributed intrusion detection model for the internet of things. In *Advanced Research on Material Engineering, Architectural Engineering and Informatization*, volume 366 of *Advanced Materials Research*, pages 165–168. Trans Tech Publications, 1 2012.

[22] D. Man, F. Zeng, W. Yang, M. Yu, J. Lv, and Y. Wang. Intelligent intrusion detection based on federated learning for edge-assisted internet of things. *Security and Communication Networks*, 2021:1–11, 2021.

[23] V. Manjula and D. C. Chellappan. Replication attack mitigations for static and mobile wsn. *International Journal of Network Security & Its Applications (IJNSA), Vol.3, No.2, March 2011*, 3(2), 2011.

[24] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 255–265, 2000.

[25] D. Midi, A. Rullo, A. Mudgerikar, and E. Bertino. Kalis—a system for knowledge-driven adaptable intrusion detection for the internet of things. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 656–666. IEEE, 2017.

[26] MITRE. CVE. https://cve.mitre.org. Accessed: 2023-11-13.

[27] M. Mohsin, Z. Anwar, G. Husari, E. Al-Shaer, and M. A. Rahman. Iotsat: A formal framework for security analysis of the internet of things (iot). In *2016 IEEE conference on communications and network security (CNS)*, pages 180–188. IEEE, 2016.

[28] S. More, M. Matthews, A. Joshi, and T. Finin. A knowledge-based approach to intrusion detection modeling. In *2012 IEEE Symposium on Security and Privacy Workshops*, pages 75–81. IEEE, 2012.

[29] National Institute of Standards and Technology. National Vulnerability Database (NVD). https://nvd.nist.gov. Accessed: 2023-11-13.

[30] NewAE Technology Inc. ChipWhisperer. https://wiki.newae.com/Main_Pag. Accessed: 2023-11-13.

[31] N. Niknejad, W. Ismail, I. Ghani, B. Nazari, M. Bahari, et al. Understanding service-oriented architecture (soa): A systematic literature review and directions for further investigation. *Information Systems*, 91:101491, 2020.

[32] K. Nithin. Blip: An implementation of 6lowpan in tinyos, 2010.

[33] Open Web Application Security Project. OWASP. https://owasp.org. Accessed: 2023-11-13.

[34] M. Osman, J. He, F. M. M. Mokbal, N. Zhu, and S. Qureshi. Ml-lgbm: A machine learning model based on light gradient boosting machine for the detection of version number attacks in rpl-based networks. *IEEE Access*, 9:83654–83665, 2021.

[35] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings. 2006 31st IEEE conference on local computer networks*, pages 641–648. IEEE, 2006.

[36] Z. Pan, S. Hariri, and J. Pacheco. Context aware intrusion detection for building automation systems. *Computers & Security*, 85:181–201, 2019.

[37] S.-T. Park, G. Li, and J.-C. Hong. A study on smart factory-based ambient intelligence context-aware intrusion detection system using machine learning. *Journal of Ambient Intelligence and Humanized Computing*, 11(4):1405–1412, 2020.

[38] Penteston. Penteston. https://penteston.com. Accessed: 2023-11-13.

[39] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, 16(1):414–454, 2013.

[40] C. Pu and X. Zhou. Suppression attack against multicast protocol in low power and lossy networks: Analysis and defenses. *Sensors*, 18(10):3236, 2018.

[41] J. L. H. Ramos, J. B. Bernabe, and A. F. Skarmeta. Managing context information for adaptive security in iot environments. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*, pages 676–681. IEEE, 2015.

[42] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt. Lithe: Lightweight secure coap for the internet of things. *Sensors Journal, IEEE*, 13(10):3711–3720, 2013.

[43] S. Raza, L. Wallgren, and T. Voigt. Svelte: Real-time intrusion detection in the internet of things. *Ad Hoc Netw.*, 11(8):2661–2674, Nov. 2013.

[44] A. Rullo, E. Bertino, and D. Saccá. Past: Protocol-adaptable security tool for heterogeneous iot ecosystems. In *2018 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8. IEEE, 2018.

[45] A. Rullo, D. Midi, E. Serra, and E. Bertino. Strategic security resource allocation for internet of things. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pages 737–738. IEEE, 2016.

[46] A. Rullo, D. Midi, E. Serra, and E. Bertino. Pareto optimal security resource allocation for internet of things. *ACM Transactions on Privacy and Security (TOPS)*, 20(4):1–30, 2017.

[47] A. Rullo, E. Serra, E. Bertino, and J. Lobo. Shortfall-based optimal placement of security resources for mobile iot scenarios. In *European Symposium on Research in Computer Security*, pages 419–436. Springer, 2017.

[48] G. Sharma, J. Grover, and A. Verma. Qsec-rpl: Detection of version number attacks in rpl based mobile iot using q-learning. *Ad Hoc Networks*, 142:103118, 2023.

[49] S. Siboni, V. Sachidananda, Y. Meidan, M. Bohadana, Y. Mathov, S. Bhairav, A. Shabtai, and Y. Elovici. Security testbed for internet-of-things devices. *IEEE Transactions on Reliability*, 68(1):23–44, 2019.

[50] A. K. Sikder, H. Aksu, and A. S. Uluagac. 6thsense: A context-aware sensor-based attack detector for smart devices. In *26th USENIX Security Symposium*, pages 397–414, 2017.

[51] A. K. Sikder, L. Babun, H. Aksu, and A. S. Uluagac. Aegis: A context-aware security framework for smart home systems. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 28–41, 2019.

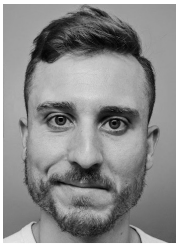
[52] T. Sylla, M. A. Chalouf, F. Krief, and K. Samaké. Setucom: Secure and trustworthy context management for context-aware security and privacy in the internet of things. *Security and communication networks*, 2021, 2021.

[53] A. Verma and V. Ranga. Security of rpl based 6lowpan networks in the internet of things: A review. *IEEE Sensors Journal*, 20(11):5666–5690, 2020.

[54] Q. Yang, Y. Liu, T. Chen, and Y. Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[55] W. Yassin, N. I. Udzir, Z. Muda, A. Abdullah, and M. T. Abdullah. A cloud-based intrusion detection service framework. In *Proceedings Title: 2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*, pages 213–218. IEEE, 2012.

Antonino Rullo received his PhD in Computer and Systems Engineering from the University of Calabria, Italy, in 2015. He was Visiting Researcher at the Lawson Computer Science Department, Purdue University, West Lafayette, IN, USA, from August 2015 to June 2016. In the period 2016-2020 he covered a postdoctoral position at the Department of Computer Engineering, Modeling, Electronics and Systems, University of Calabria, where he is currently Assistant Professor. His main research interests include IoT security and Process Mining.



Daniele Midi is a Senior Software Engineer at Google in San Francisco, CA, USA. He obtained his PhD in Computer Science from Purdue University, USA, in 2016. He got his M.Sc. in Computer Science Engineering from University of Roma Tre (Rome, Italy). His current work focuses on whole-home intelligent algorithms and AI for the smart home, developing ML models and algorithms as well as the large-scale infrastructure to support them.

Anand Mudjerikar is an applied machine learning scientist at Microsoft Security Research in Redmond, USA. He received his PhD in Computer Science from Purdue University, USA in December 2021. Before this, he got his Bachelors degree in Information and Communication Technology from DA-IICT, India and his Masters degree in Information Security from CERIAS at Purdue University, USA. His current research focus is on the combination of Machine Learning and Artificial Intelligence techniques with Computer and Network Security.

Elisa Bertino is Samuel Conte professor of Computer Science at Purdue University. She serves as Director of the Purdue Cyberspace Security Lab (Cyber2Slab). Prior to joining Purdue, she was a professor and department head at the Department of Computer Science and Communication of the University of Milan. She has been a visiting researcher at the IBM Research Laboratory in San Jose (now Almaden), at Rutgers University, at Telcordia Technologies. She has also held visiting professor positions at the Singapore National University and the Singapore Management University. Her recent research focuses on security and privacy of cellular networks and IoT systems, and on edge analytics for cybersecurity. Elisa Bertino is a Fellow member of IEEE, ACM, and AAAS. She received the 2002 IEEE Computer Society Technical Achievement Award for "For outstanding contributions to database systems and database security and advanced data management systems", the 2005 IEEE Computer Society Tsutomu Kanai Award for "Pioneering and innovative research contributions to secure distributed systems", the 2019-2020 ACM Athena Lecturer Award, and the 2021 IEEE 2021 Innovation in Societal Infrastructure Award. She received a Honorary Doctorate from Aalborg University in 2021 and a Research Doctorate in Computer Science from the University of Salerno in 2023.