Kalis - A System for Knowledge-driven Adaptable Intrusion Detection for the Internet of Things

Daniele Midi
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
dmidi@purdue.edu

Antonino Rullo
DIMES Department
University of Calabria
Rende, Italy
nrullo@dimes.unical.it

Anand Mudgerikar
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
amudgeri@purdue.edu

Elisa Bertino
Dept. of Computer Science
Purdue University
West Lafayette, IN, USA
bertino@purdue.edu

Abstract—**In this paper, we introduce Kalis, a self-adapting, knowledge-driven expert Intrusion Detection System able to detect attacks in real time across a wide range of IoT systems. Kalis does not require changes to existing IoT software, can monitor a wide variety of protocols, has no performance impact on applications on IoT devices, and enables collaborative security scenarios. Kalis is the first comprehensive approach to intrusion detection for IoT that does not target individual protocols or applications, and adapts the detection strategy to the specific network features. Extensive evaluation shows that Kalis is effective and efficient in detecting attacks to IoT systems.**

I. Introduction

As the Internet of Things (IoT) enables many novel applications, it also increases the risk of cyber security attacks [6], [15], [44], [45]. Most applications – such as mission-critical tasks, industrial control or medical monitoring – have stringent requirements with respect to reliability, data privacy, and trustworthy data delivery. IoT security is thus critical.

Designing security measures for IoT is challenging. First, while traditional wireless sensor networks (WSNs) are composed of devices homogeneous with respect to hardware and software, the IoT landscape is heterogeneous, fragmented, and not supportive of interoperability. Such a heterogeneity expands the attack surface, while at the same time increases the difficulty of deploying all-encompassing security solutions. Second, unlike WSNs, IoT devices are susceptible not only to attacks from other devices in the network, but also from more powerful attackers from the untrusted Internet.

Cryptographic techniques for IoT [19], [20], [31]–[33] help in ensuring confidentiality and authenticity of in-network traffic; however such techniques are not able to protect against all attacks. Therefore, additional security tools specific to IoT are needed. One such tool is represented by the Intrusion Detection System (IDS). IDSes are vital to maintain the IoT functional. Detecting an undergoing attack is the first line of defense for *always-on* IoT systems; however, most devices lack logging and reporting mechanisms present instead in enterprise security products [12].

Different approaches can be taken when designing an IDS for the IoT. Existing solutions deploy a custom IDS on each device or group of devices [23], [34]. This approach has the major drawback of being "too local", i.e. each IDS only has a local view of the security situation. Moreover, it does not account for interoperation of separate IoT devices and delegates security to the manufacturers of individual devices. On the other hand, while a global solution can address security at a general level, a preset IDS will not be flexible enough against the complex and heterogeneous IoT ecosystem. Simply adapting existing IDSes, designed for traditional computing networks or WSNs, is not viable. Approaches such as full network scanning to look for threats, used by well known tools like SNORT [35], are not suitable for IoT [40] as most IoT standards are based on IPv6 [37]. Also, traditional networks IDSes do not handle mobility and dynamicity. On the other hand, while WSN IDSes handle mobility, they do not support heterogeneous networks which normally rely on a host software module running on the monitored devices (unfeasible for common closed-source IoT devices), and need a central device through which all communication passes (not always available in IoT networks).

There are, however, several characteristics of IoT that can be leveraged to design an IDS fit for IoT. First most IoT devices communicate on standard mediums and protocols (such as IEEE 802.15.4 [16], WiFi or Bluetooth for mediums, and ZigBee [46] or 6LoWPAN [14] for protocols). Therefore, as long as a device is able to communicate by using several of these mediums and protocols, techniques such as promiscuous overhearing and watchdog-based mechanisms [13], [29] can be deployed. If such a device were able to host a modularly-designed IDS, new detection capabilities could be added as soon as new communication interfaces were available. Moreover, when simply observing events that may be symptoms of security incidents, specific network features can help in ruling out particular attacks, improving accuracy, and increasing detection performance. We leverage these observations in the design of our system.

In this work, we first analyze the different attack scenarios that make IoT a unique domain, and investigate the relationship between different network and devices features, and related attacks. Then, we introduce Kalis[1], a self-adapting, knowledge-driven IDS for IoT able to detect attacks in real time across IoT systems running different communication

[1]Kalis is acronym of **K**nowledge-driven **a**daptable **l**ightweight **i**ntrusion detection **s**ystem, but is also a traditional double-edged Filipino sword.

protocols. Kalis autonomously collects knowledge about the features of the monitored network and entities, and leverages such knowledge to dynamically configure the most effective set of detection techniques. To the best of our knowledge, Kalis is the first approach to intrusion detection for IoT that does not target an individual protocol or application, and adapts the detection strategy to the specific network features.

To minimize the impact on performance and to support IoT devices to which new software cannot be added, Kalis can be deployed as a standalone tool on a separate, external device, providing "security-in-a-box". In that setting, Kalis does not require changes to existing IoT software and has no performance impact on the applications running on the IoT devices. We also developed an implementation of Kalis for smart routers, such as those running OpenWRT [1], to leverage its knowledge-based approach as smart firewall for filtering suspicious incoming traffic from untrusted Internet sources to IoT devices in the local network. Kalis can be easily extended for new emerging protocol standards and leverages a customizable library of intrusion detection techniques. Last, it provides a knowledge sharing mechanism that enables collaborative incident detection, and can act as data source for multisource security information management (SIEM) systems [28].

Our contributions are: **(i)** the introduction of an IoT attack taxonomy and an analysis of the relationship between potential attacks and network/device features; **(ii)** the conceptual modeling of a knowledge-driven intrusion detection approach, and its application to the design of a self-configuring, knowledge-driven IDS for the IoT; **(iii)** the development and evaluation of a complete IDS prototype, with modules covering a wide range of network features and attacks.

II. Background

A. IoT

The IoT has several characteristics that make it a challenging domain for security measure design. First, the wide range of hardware used for IoT devices results in a diverse set of computing capabilities as well as in terms of communication mediums utilized. Second, while traditional networks – for which IDS techniques have been developed – run mostly on top of the TCP/UDP and IP protocols, the IoT depends on a diverse set of communication protocols. In some cases, a device communicates on multiple protocols and interfaces at the same time to perform its tasks. In this regard, IoT systems have different communication patterns. *Device-to-device* communication among different IoT devices (also referred to as "things") – often from different manufacturers – is almost always carried out over the Internet, through cloud services designed for device interoperability. At the same time, though, groups of devices collaborating for a common task – usually from the same manufacturer and part of the same product – form a "master-slaves" structure that we refer to as *hub-to-subs*; in this case, a powerful device coordinates several constrained devices through wireless protocols that are more constrained in power or bandwidth, such as IEEE 802.15.4 or Bluetooth.

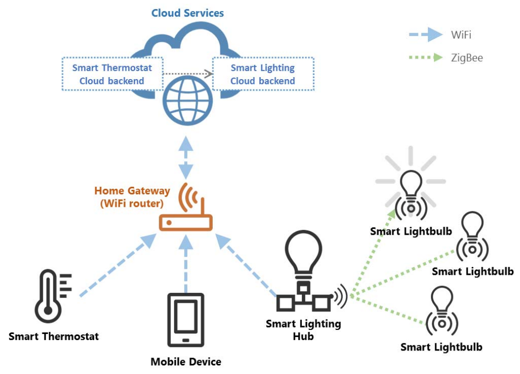


Fig. 1: A common home automation scenario, depicting the different patterns of IoT communication.

Consider a simple home automation scenario (Figure 1) composed of a smart lighting system and a smart thermostat controlled via a smartphone. The smart lighting system consists of an Internet-connected IoT device serving as hub and a set of wireless-enabled light bulbs powered by constrained microprocessors. When the smartphone issues the command of turning on a light, the command hits the cloud services, reaches the hub device through the Internet, and is then propagated locally via a ZigBee-like protocol to the actual light bulb. Conversely, interoperability between separate systems is not usually achieved via local communication. When the smart thermostat becomes aware that the user is at home, it can set the correct temperature and also require the smart lighting system to turn on the lights. Even though both devices are in the same household, such communication is achieved with the thermostat pushing a command to its own cloud service, then having the cloud services of the two systems communicating, and finally having the smart lighting system's cloud service propagating the command to the hub device.

B. Intrusion Detection Systems

Most IDSes have a common structure: a data gathering module that collects data possibly containing evidence of an attack, an analysis module that processes such data to detect attacks, and an attack reporting mechanism. The main differences in design choices for IDSes lie in [36]: **(i) Data source:** host-based, network-based, hybrid **(ii) Detection methods:** signature-based, anomaly-based **(iii) Time of detection:** online, offline **(iv) Architecture:** centralized, distributed **(v) Environment:** wired/wireless/ad-hoc network, etc.

Network-based IDSes perform their tasks by analyzing network traffic, whereas host-based IDSes require software modules, called *agents*, that run on the monitored devices themselves. Intrusion detection techniques can be broadly classified into *signature-based* and *anomaly-based*. Signature-based approaches monitor the network and try to match patterns of events or data to known attack signatures; these approaches are simpler to develop but cannot detect attacks for which the signature is unavailable. Conversely, anomaly-based techniques monitor network traffic and compare it against an established “normal” baseline – which can be dynamically learned by the system or statically set by the administrator – to detect anomalous behaviors. Such approaches are more versatile, as they can detect unknown attacks, but are harder to implement and more inaccurate, potentially yielding high false positive rates [8].

III. Knowledge-driven Intrusion Detection

IDSes typically leverage a library of detection techniques. Activating all those detection techniques guarantees a good coverage against potential attacks, but leads to inaccuracy and wasted resources. Consider attacks sharing similar symptoms: to a passive external observer – which is the case for network-based IDSes – such attacks will be indistinguishable. Moreover, processing network events and traffic through all the detection techniques requires an unnecessarily high amount of system resources, and can even introduce delays in the attack reaction. We observe, however, that some features of the monitored network and entities allow one to rule out specific attacks; for example, a selective forwarding attack cannot be carried out in a single-hop network. Collecting knowledge about the network's features thus enables the selection of the optimal set of detection techniques. Such knowledge acquisition can be carried out autonomously by the IDS, thus avoiding the need of providing the IDS with predetermined information about the network features. Such an approach also removes the configuration burden from the user, often not expert in IoT or network security especially in domestic settings. Moreover, since IoT is a dynamic environment, a particular configuration of the IDS that is optimal at a certain point in time might not any longer be optimal later on. Supporting continuous security enforcement despite environment changes is critical.

A. Conceptual Model

Our conceptual model for knowledge-driven intrusion detection is based on several 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, ...);

Feature: an intrinsic characteristic of the monitored entities and networks (e.g., multihop vs. singlehop network, mobile vs. static network, powerful vs. constrained devices, ...);

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, ...);

Detection Technique: the detection technique for a known security incident, attack or anomaly, carried out purposefully or not, that should trigger response activities, such as an alert to a user, and/or automatic response actions (such as re-transmission of packets, and device isolation).

With these concepts in place, our knowledge-driven intrusion detection approach follows this conceptual process: *using observation O, the system can determine feature F of the monitored entities and network. Given the knowledge about F, the system can determine which one(s) among detection techniques $\{D_1, D_2, ...D_n\}$ to activate. When only the right detection techniques are active, they will process the available information to detect the security incidents by symptom(s) S, thus improving the accuracy of the system.*

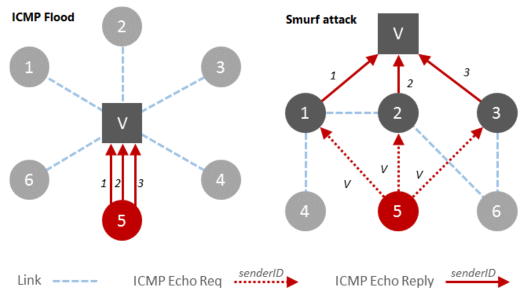


Fig. 2: ICMP Flood attack vs. Smurf attack.

1) Working Example: We illustrate our model walking through an example with two possible attacks: ICMP Flood attack and Smurf attack (see Figure 2). In a ICMP Flood attack, a single attacker node sends many ICMP Echo Reply messages to the victim, using several different identities as sender. In a Smurf attack, 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. To an external observer, these two attacks show the same symptoms: a high amount of ICMP Echo Reply messages sent to a victim node. However, the Smurf attack is not possible in single-hop networks, and this knowledge can be leveraged to achieve an accurate detection. Consider the attack and the topology shown in the left-hand side of Figure 2 and suppose that node 5 carries out an ICMP Flood attack on victim node V. Our knowledge-driven model can then be instantiated for this example as follows. *By observing the traffic, the system can reconstruct the portion of the topology in the monitored network, and determine that it is a single-hop network. Given that knowledge, the system activates the detection technique for ICMP Flood attacks and not that for Smurf attacks. Upon the detection of an unusually high amount of ICMP Echo Reply messages to the node, the only active module will unambiguously detect the undergoing ICMP Flood attack.*

B. Taxonomies

In order for our knowledge-driven model to be effectively employed, it is necessary to categorize the threats in IoT. We thus propose two taxonomies that look at IoT security threats from different perspectives.

1) Attack Patterns: Taxonomy By Target: In the IoT, different entities have different capabilities and potential to cause security incidents. We propose a classification and nomenclature from the point of view of attack patterns, considering the source and the destination of each pattern. Table I reports our taxonomy. It indicates the attack sources on the rows and the targets on the columns.

		TARGET			
		Internet Service	**Hub**	**Sub**	**Router**
SOURCE	**Internet**	Denial of Service	Remote Denial of Thing	-	-
	Hub	Denial of Service	Control Denial of Thing	Denial of Thing	Denial of Routing
	Sub	-	-	Denial of Thing	-
	Router	-	Control Denial of Thing	-	Denial of Routing

TABLE I: Taxonomy of IoT attacks by target.

Smart routers/gateways are becoming widely available commercial products, and fit well in the IoT ecosystem; thus, we include them in the categorization not only as target, but also as potential source of attacks. Note that some source-target pairs are not possible. For example, a sub would not typically be able to attack a router or an Internet service directly, as it lacks the communication hardware necessary to reach them.

Attacks aimed at Internet services are usually Denial of Service (DoS) attacks. Recent news have reported the use of IoT devices for botnet attacks [11]. However, other attacks may target IoT systems and/or IoT devices. Thus, we introduce the term *"Denial of Thing"* (DoT) for attacks aimed at disrupting the functionality of a *thing*, and consider some sub-types of DoT attacks in our taxonomy. For example, attacks targeting an IoT hub are typically aimed at denying the execution of some functions of that hub, including the control of all its dependent subs, if any. Therefore, we refer to such attack as a *"Control DoT"*. Last, we use the term *"Denial of Routing"* for all attack patterns that target IoT routers. Such attacks will typically aim at blocking the normal functionality of all the IoT devices on the local network. Note that attacks from the Internet to a local smart router cannot be addressed by any local solution, and are therefore out of the scope of our work.

2) Leveraging Knowledge: Taxonomy By Features: The development of a knowledge-driven intrusion detection model requires also an understanding of the relationships between monitored network/entity features and security incidents. Therefore, we propose a taxonomy for the most common features and attacks in the IoT. Figure 3 shows our classification, with dots and crosses indicating the possibility and impossibility, respectively, for an attack to happen in presence of a specific feature, and circles indicating instances in which the appropriate detection technique for an attack depends on the specific feature. The specific instances we choose for both attacks and features are not to be considered exhaustive, but cover a wide range of common attacks and features. Note that in many cases the detection technique for a specific attack depends on the characteristics of the network of interest; for instance, for attacks such as sybil and sinkhole the detection techniques for single-hop networks are significantly different from those adopted for multi-hop networks. Thus, for such attacks, it is important that all and only the appropriate detection techniques are activated depending on the knowledge about the various network features.

Also note that the IoT is unique in being susceptible to attacks ranging from those proper of constrained WSN nodes to those proper of traditional computer networks. Therefore, our taxonomy includes a wide set of features to accommodate all potential attacks (even though it is not possible to be exhaustive in the actual instantiation of this classification.). Last, note that we include as features also the presence of prevention techniques; for example, cryptographic techniques deployed on some of the monitored devices make the latter immune to attacks such as data alteration.

IV. Design of Kalis

A. Design Requirements

The design of an IDS for IoT systems must fulfill several important design requirements. Following are the ones we considered in the design of Kalis.

No software changes. As IoT software is often proprietary and heterogeneous, an IDS should monitor as an external entity through overhearing and environment sensing.

Multi-medium and Multi-protocol. IoT devices use different communication mediums and protocols, and all attack patterns (see Section III-B1) must be considered. An IoT IDS must comply with several standards, and be extensible to new technologies as they emerge.

No performance overhead. IoT applications have different requirements in terms of quality of service. An IDS should not impact the performance of the devices' applications.

Collaborative. A single point of view is not always sufficient for detecting security incidents. An IDS should enable knowledge sharing and collaborative detection techniques.

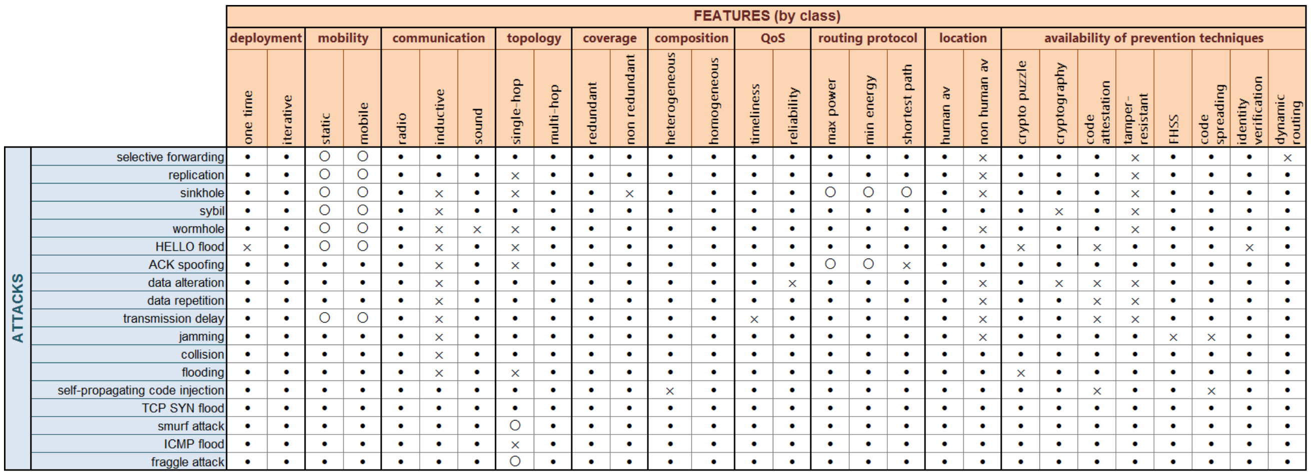
These design requirements allow Kalis to be effective against network-level attacks. We do not focus on application-level attacks, as most consumer IoT devices encrypt their communications, thus making the payload opaque to Kalis.

B. Architecture

Given the design choices of IDSes discussed in Section II-B, Kalis (see figure 4) is a *network-based*, hybrid *signature/anomaly-based*, *hybrid centralized/distributed*, *online* IDS that adapts to *different environments*.

1) Communication System: The Communication System interfaces with the external world. Specialized subcomponents take care of interacting with traffic on different protocols. The Communication System overhears all traffic on all the supported interfaces, satisfying the breadth requirement of considering all attack patterns discussed in our taxonomy by target. Our current design includes ZigBee/XBee/6LoWPAN (on IEEE 802.15.4), WiFi (on IEEE 802.11), and Bluetooth.

2) Data Store: The Data Store listens for events from the Communication System on newly captured packets, manages the history of recent traffic for modules to access, and logs all traffic on disk or memory, if required by the user. In order to appropriately utilize memory, only a sliding window of configurable size of the most recent packets is kept in memory. Logs from disk can also be replayed for traffic analysis by the network administrator in case security incidents are detected. The Data Store abstracts the traffic sources by replaying traffic



	FEATURES (by class)																											
	deployment		mobility		communication			topology		coverage		composition		QoS		routing protocol			location		availability of prevention techniques							
ATTACKS	one time	iterative	static	mobile	radio	inductive	sound	single-hop	multi-hop	redundant	non redundant	heterogeneous	homogeneous	timeliness	reliability	max power	min energy	shortest path	human av	non human av	crypto puzzle	cryptography	code attestation	tamper-resistant	FHSS	code spreading	identity verification	dynamic routing
selective forwarding	•	•	○	○	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	×	•	•	•	×
replication	•	•	○	○	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	×	•	•	•	•
sinkhole	•	•	○	○	•	×	•	×	•	•	×	•	•	•	•	○	○	○	•	×	•	•	•	×	•	•	•	•
sybil	•	•	○	○	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	×	•	×	•	•	•	•
wormhole	•	•	○	○	•	×	×	×	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	×	•	•	•	•
HELLO flood	×	•	○	○	•	×	•	×	•	•	•	•	•	•	•	•	•	•	•	•	×	•	×	•	•	•	×	•
ACK spoofing	•	•	•	•	•	×	•	×	•	•	•	•	•	•	•	○	○	×	•	•	•	•	•	•	•	•	•	•
data alteration	•	•	•	•	•	×	•	•	•	•	•	•	•	•	×	•	•	•	•	×	•	×	×	×	•	•	•	•
data repetition	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	×	•	•	×	×	•	•	•	•
transmission delay	•	•	○	○	•	×	•	•	•	•	•	•	•	×	•	•	•	•	•	×	•	•	×	×	•	•	•	•
jamming	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	•	×	×	•	•
collision	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
flooding	•	•	•	•	•	×	•	×	•	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	•	•	•	•
self-propagating code injection	•	•	•	•	•	•	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	×	•	•	×	•	•
TCP SYN flood	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
smurf attack	•	•	•	•	•	•	•	○	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
ICMP flood	•	•	•	•	•	•	•	×	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
fraggle attack	•	•	•	•	•	•	•	○	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Fig. 3: Taxonomy of relationships between IoT network/device features and attacks. Dots and crosses indicate the possibility and impossibility, respectively, of an attack in presence of a specific feature; circles indicate that the appropriate detection technique for the attack depends on the specific feature.

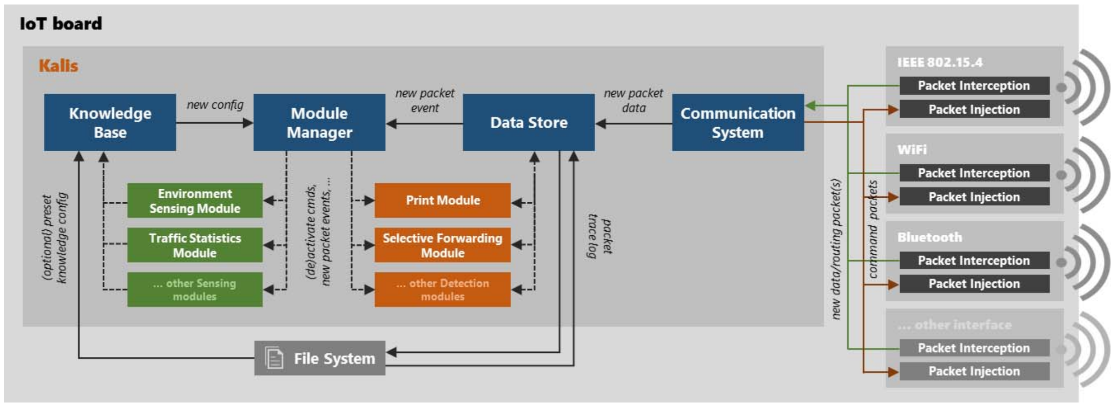


Fig. 4: The high-level architecture of Kalis.

transparently to the detection modules, which will perform their tasks as if operating on live traffic.

3) Knowledge Base and Collective Knowledge Management: This component stores all the available information about the features of the monitored entities and networks in a unique, centralized place, and makes this information available to all the parties requiring it, such as Detection Modules and the Module Manager. We refer to an individual piece of knowledge as *knowgget* (i.e. "knowledge nugget").

Knowledge Modeling. The pieces of knowledge representing features managed by the Knowledge Base can vary significantly with respect to the type of data that they are represented by. For example, the knowledge about a portion of the network being multi-hop can be modeled as a Boolean data type, but the knowledge about the total number of nodes monitored by the IDS has to be represented by an integer. We thus model each knowgget as a label, describing the information represented, and its associated value of any type. Each knowgget has a "creator" field – representing the Kalis node that created it (useful for knowledge sharing) – and an optional "entity" field – in case it is specific to an individual monitored entity (e.g., the detected signal strength for a monitored sensor node). 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 knowggets is thus not associated with a single value, but with a group of other knowggets, in a tree-like structure. Last, as Kalis does not know in advance all the knowggets to be collected, and new modules may want to store new, previously unknown knowggets, the set of labels is not fixed, and is dynamically managed as a multi-level map data structure. Figure 5a shows an example of Knowledge Base containing some knowggets about the monitored network. A knowgget k is formally defined as the tuple $k = \langle l, v, c, e \rangle$, where l is the label, v is either a primitive value or a set of knowggets (for multilevel knowggets), c is the identifier for the Kalis node creator of k, and e is the entity related to k (or `null` if none).

Collective Knowledge. The collected knowledge represents the view of the network portions surrounding the Kalis node. In most cases, this knowledge is exactly what is needed to efficiently perform intrusion detection in that portion of the network. For example, while other parts of the network might have a multi-hop topology, this is not relevant and potentially harmful to an accurate detection as compared to the knowledge that the local area – for which this Kalis node is responsible – is single-hop. In some cases, however, sharing knowledge among different Kalis nodes can enable the detection and discovery of global features useful for intrusion detection. As an example, being aware that other Kalis nodes are noticing changes in signal strength for specific devices can enable the local Kalis node to correlate such changes with those experienced locally and detect mobility in the network. Kalis' mechanism for collective knowledge management allows for sharing and synchronizing selected information across Kalis nodes. To enable this mechanism, a module inserting a knowgget of collective interest into the Knowledge Base can mark the knowgget as "collective". The Knowledge Base then takes care of communicating changes in that knowgget to the other Kalis nodes for storage in their Knowledge Bases, making sure to mark the appropriate identity in the "creator" field denoting the Kalis node that generated the knowgget. Note that this mechanism does not provide a way for a Kalis node to overwrite or alter knowledge in another Kalis node. When a Kalis node, say T_1, receives a new or updated collective knowgget k from a different Kalis node, say T_2, the Knowledge Base of T_1 will check to see if the label *and* creator of k matches any existing knowgget in the Knowledge Base. Therefore, T_1 can only update those knowggets in T_2 that were originally generated by itself.

Static Knowledge. Kalis also allows one to specify initial or static knowggets. For example, if the network is static and will always remain so, it makes sense to provide Kalis with this information, as it can be leveraged to avoid trying to detect mobility in the network. For this purpose, the Knowledge Base optionally loads a configuration file providing initial settings and a-priori available knowggets. Figure 6 shows the (JSON-inspired) grammar for the language used for configuration files, and Figure 7 shows an example of a configuration file to activate by default two modules and to insert the a-priori knowledge that the network is static. Note that in this case the knowggets might specify an "entity" field, but not a "creator" field, as they will be assigned the local Kalis node's identifier as such.

4) Modules: In Kalis any network feature-specific or attack-specific functionality is implemented as an independent module. The Module Manager component coordinates all the modules, activating/deactivating them as needed, depending on changes in the Knowledge Base, routing new packet events to all the interested parties, and collecting alerts about detected incidents. Each module is able, given a particular instance of the Knowledge Base, to determine whether its services are required and, thus, whether it should be active at that particular point in time. Kalis includes two different types of modules: Sensing modules and Detection modules.

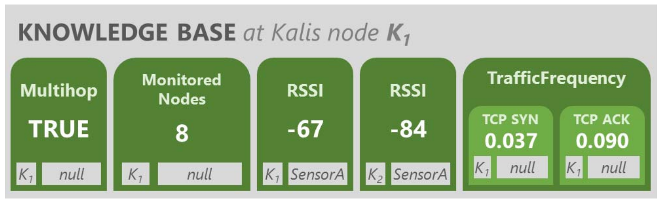
Sensing Modules. Sensing modules are the core of the autonomous knowledge-discovery mechanisms of Kalis. We include in our prototype three sensing modules: a Topology Discovery module, a Traffic Statistics Collection module, and a Mobility Awareness module. The first uses the captured traffic to reconstruct the local topology and differentiate between multi-hop and single-hop networks. The second continuously collects statistics about the traffic load, differentiated for each type of packet (e.g., ZigBee data, ZigBee routing, TCP SYN, TCP ACK, ...). The third leverages the signal strength to dynamically detect whether the network is static or mobile (see Section V for details). Whenever a sensing module finds a relevant change in network features – such as the discovery that a portion of the monitored network is multi-hop – it will store this new knowgget into the Knowledge Base. The Knowledge Base will in turn notify the Module Manager that recent changes to the available knowledge might require activating or deactivating specific modules.

Detection Modules. Detection modules analyze the captured traffic – together with the available knowggets – and detect anomalies and security incidents. Each module is specialized on a specific attack, but some techniques could be generalized to detect attacks with similar symptoms but different severity or root causes – e.g. selective forwarding attack vs. blackhole attack. As discussed in Section II-B, intrusion detection techniques are either *signature-based* or *anomaly-based*. However, the data and knowledge made available by Kalis makes it possible to include detection modules of both kinds, increasing the accuracy in detecting known attacks while at the same time being able to react to unknown attacks. In our prototype, we include several detection modules for common attacks, such as selective forwarding, SYN flow, ICMP flood, and replication.

Scalability. IoT networks can be large thus raising the issue of scalability for our dynamic module loading mechanism. However, because of the locality of the knowledge acquired by each Kalis node, different IDS nodes can load different (and locally-optimal) sets of modules depending on their surroundings, thus allowing the system to scale to arbitrarily large networks just by means of adding new IDS nodes throughout the network.

V. Implementation

Development Environment. We implement Kalis using Java on an Odroid xu3 development board. In order to interact with the IEEE 802.15.4 traffic, we leverage a TelosB [27] wireless sensor mote with a custom TinyOS [21] application as bridge. Moreover, we integrate into Kalis the `tcpdump` utility – which internally uses the *libpcap* library – in order to promiscuously monitor all WiFi traffic. Our implementation uses Java Reflection in various parts of the system. For example, when the configuration file is parsed and a module is specified to be



(a) An example of Knowledge Based with heterogeneous knowggets, each showing label, value, creator field and entity field.

```
KnowledgeBase {
  "K1$Multihop" = "true",
  "K1$MonitoredNodes" = "8",
  "K1$SignalStrength@SensorA" = "-67",
  "K2$SignalStrength@SensorA" = "-84",
  "K1$TrafficFrequency.TCPSYN" = "0.037",
  "K1$TrafficFrequency.TCPACK" = "0.090"
}
```

(b) Key-value pair representation of the Knowledge Base in the implementation of Kalis.

Fig. 5: Comparison of abstract vs. implemented representation of knowledge in Kalis.

```
⟨config⟩         ::= ⟨modules⟩ ⟨knowggets⟩
⟨modules⟩        ::= 'modules = {' ⟨module-list⟩ '}'
⟨module-list⟩    ::= ⟨module-def⟩ ',' ⟨module-list⟩
                  |  ⟨module-def⟩
⟨module-def⟩     ::= ⟨module-name⟩ [ '(' ⟨param-list⟩ ')' ]
⟨param-list⟩     ::= ⟨key-value-pair⟩ ',' ⟨param-list⟩
                  |  ⟨key-value-pair⟩
⟨knowggets⟩      ::= 'knowggets = {' ⟨knowgget-list⟩ '}'
⟨knowgget-list⟩  ::= ⟨key-value-pair⟩ ',' ⟨knowgget-list⟩
                  |  ⟨key-value-pair⟩
⟨key-value-pair⟩ ::= ⟨key⟩ '=' ⟨value⟩
```

Fig. 6: Grammar for Kalis configuration files.

```
modules = {
  TopologyDetectionModule,
  TrafficStatsModule (
    activationThresh=1,
    detectionThresh=2
  )
}
knowggets = {
  mobility = false
}
```

Fig. 7: Example of Kalis configuration files.

activated, the corresponding class is dynamically instantiated by name. This implementation makes it possible to add new modules without the need to recompile the entire system as long as those modules implement the required interfaces.

Event-driven Architecture. All the components in Kalis (see Figure 4) run independently. For example, 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. In the same way, when a detection module detects a potential attack, it raises a detection event that is then routed to all the subscribed parties. This also allows Kalis to interoperate with cloud-based monitoring dashboards, automated response systems, and real-time user notification mechanisms.

Knowledge Representation. To implement the Knowledge Base, we model each knowgget as a key-value pair, in which both the key and the value are represented as strings. When querying the Knowledge Base, the modules can either retrieve the raw value and parse it independently, or specify what is the data type they expect in return for a given key and the Knowledge Base will attempt to parse the string as that data type. Our encoding of the key allows for fast queries. Given a knowgget $k = \langle label, value, creator, entity \rangle$, Kalis encodes it as a key `"`$creator\$label@entity$`"` and a value `"`$value$`"`. Looking up local (or collective) knowggets only requires searching for the prefix matching (or not matching) the identifier of the local Kalis node. Instead, looking up knowggets related to a specific entity only requires searching for keys with a suffix matching the identifier of the entity of interest. Last, finding a single specific knowgget is done by matching the key exactly. This model allows Kalis to uniformly represent multilevel knowggets by flattening the hierarchy of labels in dot notation; that is, the sub-information of the "TrafficFrequency" knowgget about TCP SYN packets created by Kalis node T_1 is represented as an individual knowgget with key `"T1$TrafficFrequency.TCPSYN"`. Figure 5b shows an example of how the Knowledge Base in Figure 5a is represented in our implementation of Kalis.

Collective Knowledge Synchronization. The implementation of the synchronization mechanism relies on a few building blocks. First, the discovery of peer Kalis nodes is carried out by periodical beaconing on the local network. Each Kalis node listens for advertisement broadcast packets from other Kalis nodes, and adds newly-discovered nodes to a peer list, in a commonly used *discovery-through-advertisement* pattern effective for local networks with a moderate number of peers (reasonable assumption for a Kalis deployment). All communications among the nodes are encrypted, and only enable a one-way communication (in each direction) between pairs of nodes, without the need for interaction beyond the acceptance of incoming new or updated collective knowggets.

Sensing Modules. Kalis current implementation includes several sensing modules. The Topology Discovery module detects multi-hop and single-hop topology by analyzing the captured traffic. The features used for this analysis include the communication medium used (IEEE 802.15.4 or WiFi), the detection of known protocols (such as RPL in 6LoWPAN or Collection Tree Protocol in TinyOS), the inclusion of specific forwarding/next-hop headers in packets, and more. The range of characteristics that are leveraged to understand the topology of the network can be extended when new protocols or mediums are standardized for the IoT. The Traffic Statistics Collection module maintains statistics about the frequency of the various types of traffic overheard in the network, both on a global and per-monitored-device level. Our implementation maintains statistics for several different types of traffic, including TCP SYN, TCP ACK, ICMP Requests, ICMP Responses, ZigBee plain packets, and Collection Tree Protocol packets. For each traffic type, the module records the number of packets per unit of time (configurable but set to 5 seconds by default); the frequency of each type of traffic is recorded both for the whole network, and for each individual monitored device (to support an accurate detection of targeted DoS-like attacks and subsequent potential response actions.) This sensing module supports, for example, the use of anomaly-based detection modules that can detect unknown attacks, even when their signature is not predetermined. The Mobility Awareness module uses a simple approach that detects mobility when any node's signal strength changes more than a certain threshold. More complex techniques could also be employed, but are out-of-scope for this work.

Dynamic Detection Module Configuration. We implement the dynamic activation and deactivation of detection modules based on the changes in the Knowledge Base via a *publish-subscribe* mechanism. Each detection module can subscribe to changes on one or more knowggets by key, and is automatically notified; the modules will therefore notify the Module Manager when their services are no longer required, according to the latest knowledge.

Smart Firewall Deployment. We have developed a version of Kalis for deployment on smart network routers running the OpenWRT firmware [1]. This enables Kalis to act as a smart firewall. To do so, we first create a runnable JAR file of Kalis containing all the required libraries and packages. For the deployment on smart routers, we had to address several technical challenges. For instance, due to memory constraints, it is not possible to execute a full-fledged Java Virtual Machine (JVM). We overcome this problem by executing the JAR file using *JamVM* [24], an open-source compact JVM implementation for embedded devices. Normally, JamVM would need to be used in conjunction with Java Class Libraries, such as GNU Classpath [7], to obtain a full Java Runtime Environment. In our scenario, due to the router's space constraints, we only use the standalone JamVM executable to run Kalis, without adding a Java class library but packaging instead all the minimum required libraries into the Kalis JAR file itself.

VI. Evaluation

In our experiments, we evaluate (a) the breadth of the IDS coverage over heterogeneous networks, devices and protocols, (b) the benefits of the knowledge-driven approach on the detection accuracy in terms of false positives and detection rate, as well as resource consumption, (c) the reactivity of the IDS in the dynamic discovery of a changing environment, and (d) the benefits of the collaborative knowledge mechanisms.

A. Experimental Setup

We evaluate Kalis by placing the IDS node near a network of heterogeneous, real-world IoT devices. Our setup includes a small WSN of 6 TelosB nodes, a Nest Thermostat, an August SmartLock, a Lifx smart lightbulb, an Arlo security system, and an Amazon Dash Button. All the WSN nodes are programmed with a TinyOS application that sends a data message every 3 seconds towards a node acting as base station, using the Collection Tree Protocol (CTP) [10]. Concerning the WSN traffic, the Kalis node is placed near the middle portion of the WSN, able to overhear intermediate hops of data packets. Since compromising commodity IoT devices – especially to carry out various controlled attacks in a repeatable way – is difficult, we choose to record and replay actual traces of network traffic from these devices, enhanced with additional packets representing symptoms of such attacks. For each attack scenario, we run the systems on 50 symptom instances, representing the ground truth for detection. We believe that this setup truthfully represents the complexity of the IoT ecosystem, including both multi-hop and single-hop networks, different protocols on different mediums (CTP on IEEE 802.15.4, TCP/IP on WiFi), and different devices in terms of computational power and functionality. Furthermore, to simulate potential response actions upon an IDS detection, we program as a simple countermeasure the temporary revocation from the network of any node identified as suspect by the IDS.

B. Benefits of Knowledge-Driven Approach

In our experiments, we compare our knowledge-driven approach to that of a traditional IDS. For total fairness with respect to the detection techniques, we emulate a traditional IDS by running our system without Knowledge Base, and with all the modules active at all times. We also compare Kalis with Snort [35], using custom rules along with the default community ruleset for Snort to detect attacks in our simulated scenarios. Please note that a direct comparison with current IoT-specific solutions is not possible, since their assumptions differ significantly from those of Kalis (as detailed in Section VII, e.g., they target a single group of devices part of the same system, require software changes to the monitored devices, are limited to a single protocol, ...). We therefore choose to compare against Snort as it is a general purpose IDS as well as the *de facto* standard for intrusion detection. We choose attack scenarios as representative of two classes of attacks: (a) different attacks that present identical or very similar symptoms, and (b) attacks for which the detection technique to use depends on the network features. By evaluating our system in these two scenarios, we show how our knowledge-driven approach significantly improves the detection of all the attacks that fall into those two broad categories. We compare the systems on several metrics: (i) **Detection Rate** – number of adverse events detected out of all the adverse events in the test scenario; (ii) **Classification Accuracy** – number of correctly classified attacks out of all the detected attacks; (iii) **Countermeasure effectiveness** – how positive a response action based on the detections of Kalis is for the overall network; (iv) **CPU usage**; (v) **RAM usage**.

1) ICMP Flood attack on single-hop network: The first scenario is that of an ICMP Flood attack on a single-hop network (see Section III-A1). In this setting, the traditional

IDS identifies all the attacks (high detection rate); however it generates false positives as it is not able to disambiguate the ICMP Flood attack from a Smurf attack. Snort shows a high detection rate for attacks, but it is not able to distinguish between the Smurf and ICMP Flood attacks.

With respect to countermeasures, the ICMP Flood detection module considers as suspect all nodes within one hop from the victim; conversely, the Smurf attack detection module considers as suspect all nodes at a 2-hop distance from the victim; both attempt an approximate disambiguation through a comparison of the signal strength with previous overheard communications [5], [42]. We observe that, in this scenario, Kalis correctly revokes only the attacking node, while the traditional IDS attempts to revoke the only node two hops away from the victim, which in a simplistic graph exploration is the victim node itself, therefore disconnecting the entire network.

2) Replication attack on static vs. mobile network: The replication attack is an application-independent attack unique to wireless networks of constrained devices. In such attack, malicious devices are added to the network as replicas of some legitimate node(s). Many detection techniques exist for this attack; however each one is specific to a network with certain characteristics, e.g. mobility [25]. In this evaluation, we provide two different detection modules for replication attacks, one suitable for static networks, and the other for mobile networks. The network in this evaluation randomly changes between a static and mobile behavior of the nodes over time. We repeat the evaluation 100 times, each time carrying out 3 replication attacks (i.e., sending data packets from 3 nodes that are replicas of legitimate nodes in the network). As the communications for this experiment run on ZigBee, Snort is unable to intercept and analyze the traffic, and thus unable to detect attacks in this scenario. The traditional IDS randomly selects one of the two modules for each of our experiment runs, closely simulating a static module library configuration that does not adapt to the changes in network features. Kalis, instead, leverages the knowledge by the Mobility Awareness module, and dynamically selects modules for the current network mobility setting. We observe the detection rate and accuracy of both Kalis and the traditional IDS approach.As expected, while Kalis always uses the right modules, the traditional IDS approach misses some attacks when the active module is not the one suitable for the current mobility profile of the network.

3) Overall Results: We summarize the experimental results in Table II for effectiveness and performance metrics. Kalis achieves 100% accuracy, as it leverages the optimal set of modules for detection. As the detection techniques used cannot always detect all attacks, the detection rate is not perfect; however, this is independent from Kalis, and the comparison with the traditional IDS approach using the same detection techniques still shows the benefits of Kalis. The results show that our prototype is lightweight in terms of CPU and RAM requirements, and that the knowledge-driven approach of Kalis outperforms the traditional IDS and Snort on all the metrics.

	Trad. IDS	**Snort**	**Kalis**
Detection Rate	48%	89%	91%
Accuracy	75%	76%	100%
CPU usage	0.22%	6.3%	0.19%
RAM usage (kb)	23961.06	101978.24	13978.62%

TABLE II: Average effectiveness and performance across both experimental scenarios in Section VI-B1.

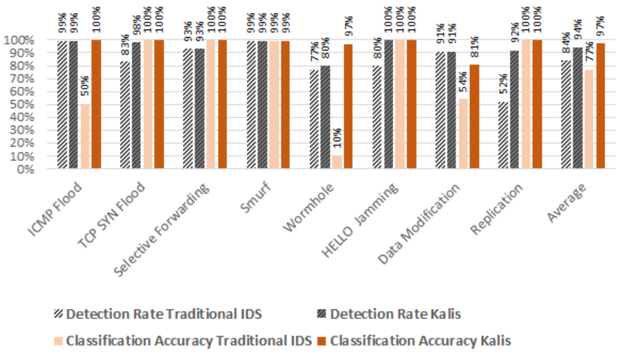


Fig. 8: Effectiveness comparison for Kalis vs. traditional IDS approaches across all experimental scenarios (averages).

C. Reactivity to Environment Changes

To evaluate the reactivity of Kalis, we run it with a configuration file that does not activate any detection modules by default and does not contain any a-priori knowgget. We then let Kalis monitor a ZigBee network with one node programmed to carry out selective forwarding attacks, and measure how soon Kalis detects the first attack. The selective forwarding detection module only activates upon discovering a multi-hop network; the Topology Discovery sensing module detects such feature from the first CTP packets intercepted. Thus, we verify that Kalis correctly identifies 100% of the selecting forwarding attacks from the very beginning of the communications, even with no detection modules initially active.

D. Knowledge Sharing

We evaluate Kalis in a scenario where collaborative knowledge enables the selection of the appropriate set of detection modules. In this scenario, two Kalis nodes monitor two different portions of a ZigBee network. One node in each portion is malicious, namely nodes B_1 and B_2, and they collude in carrying out a wormhole attack. In such attack, B_1 does not correctly forward traffic, transmitting it instead directly to B_2. The Kalis node observing the behavior of B_1 would, by itself, detect a blackhole attack, while the Kalis node observing B_2 would, without further information, consider it a source of traffic. However, correlating the events between the two Kalis nodes, they are able to correctly identify such attack as a wormhole.

E. Breadth of Attack Detection

We evaluated Kalis over a wide range of attacks, including the attack scenarios introduced in the previous subsections.

Overall, we consider 8 attack scenarios; Figure 8 summarizes the results. Snort is not shown as it could not run on any of the ZigBee-based attacks scenarios. The accuracy of Kalis relies in part on the overall accuracy of the loaded detection modules. For transparency in the evaluation, we reflect in the reported data also the cases in which the accuracy of the underlying detection techniques does not reach 100%; however, we observe that Kalis is always more effective than traditional IDS approaches and, on average, achieves significant improvements.

VII. Related Work

Extensive research has been carried out on IDSes for traditional networks. Two popular open source IDSes are SNORT [35] and Bro [30]. Both IDSes rely on network information gathered by a packet sniffer, and detect attacks using signature matching over this information. However, their techniques are not applicable to the IoT domain. For example, techniques such as host scanning or port scanning would be ineffective on most IoT standards using IPv6. Also, they only work on traditional IP-only networks (wired and wireless), while Kalis supports a wide variety of mediums and related protocols. Most traditional IDSes rely on a list of rules to detect attacks; one of the most updated and popular open source rule list is by Talos [39]. However, while running through a large rule list is sustainable for a traditional network, small IoT networks would incur heavy overhead on the performance of the IDS. Also, running all the rules usually results in more false positives. Kalis, instead, uses an attack detection mechanism that adapts according to the network environment, thus avoiding unnecessary effort. Several IDSes have been developed for WSNs [2], [4], [9], [38], [41]; however, they suffer from one or more limitations with respect to IoT: inability to adapt, applicability only to a single platform and protocol, small and specific range of detection techniques, complete dependency on collaboration, reliance on the existence of a central control point. Thus, while we do not assert that no security tool exists built for WSN or traditional networks that could work in an IoT environment, we believe that the premises used to design those IDSes are significantly different from, and do not fully apply to, the domain of IoT. We note, however, that existing techniques can be incorporated in Kalis as detection modules.

Recent research efforts exist focusing on IDSes for IoT. SVELTE [34] is the most relevant of such efforts. It is both centralized (at the hub of an IoT system/group of devices) and distributed (at each sub). It is composed of a *6Mapper* module, which reconstructs the topology of the subs with respect to the hub, and an intrusion detection module, which analyzes data and detects intrusion. In comparison to Kalis, SVELTE (a) is host-based and thus requires modifications to the IoT devices' software, (b) targets a single IoT system (hub and subs), (c) is primarily designed for devices communicating via the RPL protocol (IPv6 Routing Protocol for Low-Power and Lossy Networks) [43] and cannot be expanded to multiple heterogeneous protocols and mediums, (d) leverages an extensible but predefined set of detection techniques, with no dynamic activation, (e) does not adjust to environmental changes. Liu *et al.* propose the application of Artificial Immune Systems to IoT IDSes [22], [23] to design a detection mechanism based on datagram signature analysis, with *vaccine* sharing among IDS nodes when a new attack signature is generated. In such an approach it is unclear how to determine the ground truth about legitimate datagrams. Moreover, the administrator is still required to understand which attack a new detector is capturing and supplement the attack library knowledge. In Kalis, different nodes, guarding different network portions, all have access to the same library of Detection Modules as with vaccines sharing, but Kalis activates only the modules needed for the monitored network portion, improving efficiency and accuracy. Jun *et al.* propose using Complex Event-Processing (CEP) techniques for intrusion detection in IoT [17]. Such work focuses on improving the IDS performance online rather than offline. Our work also leverages CEP techniques, but uses the Knowledge Base to avoid processing unnecessary rules. We note that approaches focusing on adaptable IDSes have been applied to other domains; for example, Karaarslan *et al.* investigated network awareness for intrusion detection for Web services [18]. This encourages us to believe that our proposed system is feasible and effective. Finally, some research efforts have focused on taxonomies of IoT threats. Babar *et al.* [3] proposed a taxonomy based on the attacker goal, such as Physical Threat, Communication Threat, Identity Management. Mayzaud *et al.* [26] propose an attack taxonomy for RPL-based IoT networks, thus focusing only on such routing protocol. In our taxonomies, we classify attacks from the high-level perspective of specific attacks and communication patterns, and aim at finding relationships between the features of the monitored devices/networks and the attacks, since these classifications are critical for comprehensive IDS for IoT.

VIII. Conclusions

We designed Kalis, the first self-adapting, knowledge-driven IDS for IoT, able to detect attacks across a wide range of protocols. The experimental evaluation shows that Kalis is effective and efficient in detecting attacks in IoT networks. As future work, we will investigate the use of machine learning techniques for knowledge collection and extend Kalis with specialized cloud-based intrusion detection services. Moreover, while the current prototype of Kalis is designed to be deployed on less constrained IoT boards, some scenarios could benefit from the deployment of specialized Kalis nodes on constrained devices. We envision the possibility of selecting a specific module configuration – based on the knowledge collected by Kalis in a network – and to deploy that configuration at compile-time on very small devices such as WSN nodes.

Acknowledgments

The work reported in this paper has been partially supported by NSF under award ACI-1547358.

References

[1] OpenWRT. https://openwrt.org/.

[2] N. A. Alrajeh, S. Khan, and B. Shams. Intrusion detection systems in wireless sensor networks: a review. *International Journal of Distributed Sensor Networks*, 2013, 2013.

[3] S. Babar, P. Mahalle, A. Stango, N. Prasad, and R. Prasad. *Recent Trends in Network Security and Applications: Third International Conference, CNSA 2010, Chennai, India, July 23-25, 2010. Proceedings*, chapter Proposed Security Model and Threat Taxonomy for the Internet of Things (IoT), pages 420–429. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[4] R.-C. Chen, C.-F. Hsieh, and Y.-F. Huang. A new method for intrusion detection on hierarchical wireless sensor networks. In *International Conference on Ubiquitous Information Management and Communication*, ICUIMC '09, New York, NY, USA, 2009. ACM.

[5] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee. Identifying unique devices through wireless fingerprinting. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, pages 46–55, New York, NY, USA, 2008. ACM.

[6] M. U. Farooq, M. Waseem, A. Khairi, and S. Mazhar. A critical analysis on the security concerns of internet of things (iot). *International Journal of Computer Applications*, 111(7), 2015. Copyright - Copyright Foundation of Computer Science 2015; Last updated - 2015-03-13.

[7] G. Foundation. Gnu classpath, 2012.

[8] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[9] K. Gerrigagoitia, R. Uribeetxeberria, U. Zurutuza, and I. Arenaza. Reputation-based intrusion detection system for wireless sensor networks. In *Complexity in Engineering (COMPENG)*, June 2012.

[10] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ACM SenSys, pages 1–14, 2009.

[11] A. Gonsalves. New toolkit seeks routers, internet of things for ddos botnet. http://www.csoonline.com/article/2687653/data-protection/new-toolkit-seeks-/routers-internet-of-things-for-ddos-botnet.html, 2014. Accessed: May 2016.

[12] A. Grau. Intrusion detection software lowers internet of things risk. http://www.controleng.com/single-article/intrusion-detection-software-lowers-internet-of-things-iot-risk/e1ca58c3af94e41f26bc4836c21803f5.html, 2015. Accessed: May 2016.

[13] Y.-a. Huang and W. Lee. A cooperative intrusion detection system for ad hoc networks. In *ACM Workshop on Security of Ad Hoc and Sensor Networks*, New York, NY, USA, 2003. ACM.

[14] J. Hui, D. Culler, and S. Chakrabarti. 6lowpan: Incorporating ieee 802.15. 4 into the ip architecture. *IPSO Alliance White Paper*, 3, 2009.

[15] Y. H. Hwang. Iot security &#38; privacy: Threats and challenges. In *Workshop on IoT Privacy, Trust, and Security*, IoTPTS '15. ACM, 2015.

[16] IEEE. Ieee 802.15 wpan task group 4 (tg4). http://www.ieee802.org/15/pub/TG4.html.

[17] C. Jun and C. Chi. Design of complex event-processing ids in internet of things. In *2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*, pages 226–229, Jan 2014.

[18] E. Karaarslan, T. Tuglular, and H. Sengonca. Does network awareness make difference in intrusion detection of web attacks. 2006.

[19] T. Kothmayr, W. Hu, C. Schmitt, M. Bruenig, and G. Carle. Poster: Securing the internet of things with dtls. In *ACM Conference on Embedded Networked Sensor Systems*. ACM, 2011.

[20] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle. Dtls based security and two-way authentication for the internet of things. *Ad Hoc Networks*, 11(8):2710–2723, 2013.

[21] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In W. Weber, J. Rabaey, and E. Aarts, editors, *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg, 2005.

[22] C. Liu, J. Yang, R. Chen, Y. Zhang, and J. Zeng. Research on immunity-based intrusion detection technology for the internet of things. In *Natural Computation (ICNC), 2011 Seventh International Conference on*, volume 1, pages 212–216, July 2011.

[23] 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.

[24] R. Lougher. Jamvm – a compact java virtual machine, 2014.

[25] V. Manjula and D. C. Chellappan. Replication attack mitigations for static and mobile wsn. *arXiv preprint arXiv:1103.3378*, 2011.

[26] A. Mayzaud, R. Badonnel, and I. Chrisment. A taxonomy of attacks in rpl-based internet of things. *Int'l Journal of Network Security*, 2016.

[27] Memsic. Telosb datasheet. http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf.

[28] D. Miller, S. Harris, A. Harper, S. VanDyke, and C. Blask. *Security information and event management (SIEM) implementation*. McGraw Hill Professional, 2010.

[29] A. Mishra, K. Nadkarni, and A. Patcha. Intrusion detection in wireless ad hoc networks. *IEEE Wireless Communications*, 11(1):48–60, Feb 2004.

[30] V. Paxson, S. Campbell, J. Lee, et al. Bro intrusion detection system. Technical report, Lawrence Berkeley National Laboratory, 2006.

[31] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. Securing communication in 6lowpan with compressed ipsec. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pages 1–8, June 2011.

[32] S. Raza, S. Duquennoy, J. Hglund, U. Roedig, and T. Voigt. Secure communication for the internet of thingsa comparison of link-layer security and ipsec for 6lowpan. *Security and Communication Networks*, 7(12):2654–2668, 2014.

[33] 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.

[34] 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.

[35] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999.

[36] F. Sabahi and A. Movaghar. Intrusion detection: A survey. In *Systems and Networks Communications, 2008. ICSNC'08. 3rd International Conference on*, pages 23–26. IEEE, 2008.

[37] Z. Shelby and C. Bormann. *6LoWPAN: The wireless embedded Internet*, volume 43. John Wiley & Sons, 2011.

[38] B. Sun, L. Osborne, Y. Xiao, and S. Guizani. Intrusion detection techniques in mobile ad hoc and wireless sensor networks. *IEEE Wireless Communications*, 14(5):56–63, October 2007.

[39] C. Systems. Talos – official snort rule set, Accessed: November 2016.

[40] The Security Ledger. Ids and the iot: Snort creator marty roesch on securing the internet of things. https://securityledger.com/2014/04/ids-and-the-iot-snort-creator-marty-roesch-on-securing-the-internet-of-things/.

[41] M. Tiwari, K. V. Arya, R. Choudhari, and K. S. Choudhary. Designing intrusion detection to detect black hole and selective forwarding attack in wsn based on local information. In *Computer Sciences and Convergence Information Technology, 2009. ICCIT '09. Fourth International Conference on*, pages 824–828, Nov 2009.

[42] J. Wang, G. Yang, Y. Sun, and S. Chen. Sybil attack detection based on rssi for wireless sensor network. In *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pages 2684–2687, Sept 2007.

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

[44] Z.-K. Zhang, M. C. Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen, and S. Shieh. Iot security: ongoing challenges and research opportunities. In *Service-Oriented Computing and Applications (SOCA)*. IEEE, 2014.

[45] K. Zhao and L. Ge. A survey on the internet of things security. In *Computational Intelligence and Security (CIS)n*, Dec 2013.

[46] ZigBee Alliance and others. Zigbee specification, 2006.