SENTINEL: A Robust Intrusion Detection System for IoT Networks Using Kernel-Level System Information

Adrien Cosson[1], Amit Kumar Sikder[2], Leonardo Babun[2], Z. Berkay Celik[3], Patrick McDaniel[1], A. Selcuk Uluagac[2]
[1]Penn State University - amc1103@psu.edu, mcdaniel@cse.psu.edu
[2]Florida International University - {asikd003, lbabu002, suluagac}@fiu.edu
[3]Purdue University - zcelik@purdue.edu

ABSTRACT

The concept of Internet of Things (IoT) has changed the way we live by integrating commodity devices with cyberspace to automate our everyday tasks. Nowadays, IoT devices in the home environment are becoming ubiquitous with seamless connectivity and diverse application domains. Modern IoT devices have adopted a many-to-many connectivity model to enhance user experience and device functionalities compared to early IoT devices with standalone device setup and limited functionalities. However, the continuous connection between devices and cyberspace has introduced new cyber attacks targeting IoT devices and networks. Due to the resource-constrained nature of IoT devices as well as the opacity of the IoT framework, traditional intrusion detection systems cannot be applied here. In this paper, we introduce SENTINEL, a novel intrusion detection system that uses kernel-level information to detect malicious attacks. Specifically, SENTINEL collects low-level system information (CPU usage, RAM usage, total load, available swap, etc.) of each IoT device in a network and learns the pattern of device behavior to differentiate between benign and malicious events. We evaluated the efficacy and performance of SENTINEL in different IoT platforms with multiple devices and settings. We also measured the performance of SENTINEL against five types of real-life attacks. Our evaluation shows that SENTINEL can detect different attacks to IoT devices and networks with high accuracy (over 95%) and secure the devices in different IoT platforms and configurations. Also, SENTINEL achieves minimum overhead in power consumption, ensuring high compatibility in resource-constraint IoT devices.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**.

KEYWORDS

Internet of Things, Intrusion detection, Mirai Botnet, IoT security.

ACM Reference Format:
Adrien Cosson[1], Amit Kumar Sikder[2], Leonardo Babun[2], Z. Berkay Celik[3], Patrick McDaniel[1], A. Selcuk Uluagac[2]. 2021. SENTINEL: A Robust Intrusion Detection System for IoT Networks Using Kernel-Level System Information. In *International Conference on Internet-of-Things Design and Implementation (IoTDI '21), May 18–21, 2021, Charlottesvle, VA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3450268.3453533

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
IoTDI '21, May 18–21, 2021, Charlottesvle, VA, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8354-7/21/05…$15.00
https://doi.org/10.1145/3450268.3453533

1 INTRODUCTION

Internet of Things (IoT) is a new technology domain that has been steadily increasing in popularity in the past decade. It is today a multi-billion dollar industry and has penetrated in different application domains. The most commonly encountered use of IoT is in smart homes, where IoT devices (such as light bulbs, cameras, or door locks) are installed and connected to provide a way of automating daily tasks and overseeing the state of the home environment [9, 10]. Another main application of IoT is industrial environment monitoring, where wireless sensors are deployed to provide real-time feedback of the state of a production line or warehouse [42, 56]. A more recently emerging application domain is healthcare, where patient care and monitoring machines are connected to provide a centralized and more accessible view of their current condition [34, 35].

Over the past decade, security issues have been frequently found in the majority of IoT devices and frameworks. Attacks specialized against IoT devices have been developed and deployed massively in recent years [10, 48]. The most notable one is the Mirai botnet, which was first detected in late 2016, infected more than half a million devices in a span of a few months, and used this network to launch a series of DDoS attacks. This includes the largest one to date, with a throughput of 623 Mbps [5]. These security issues are mostly due to economics; as IoT devices have to be relatively cheap to be competitive, manufacturers and application developers often forgo good security practices as a way to keep costs down. Furthermore, as the IoT market is fast-moving, with new products announced every month, a short time-to-market is often crucial for manufacturers, causing security concerns to be delegated to low-priority or even post-launch items. Although intrusion detection for standard networks is a mature field, it is still developing for IoT devices and networks due to devices having different constraints such as low computing power, meshed and ad-hoc structure, or limited battery capacity. These properties require the use of intrusion detection methods that do not incur an overhead in processing and power usage. Since traditional intrusion detection methods rely heavily on local pre-processing work, they are not applicable to IoT systems [8].

Indeed, IoT framework implementations are too opaque and high-level to provide a useful level of insight into the low-level state of the devices that obstruct the integration of traditional intrusion detection systems. The IoT frameworks abstract away the hardware's specifics by presenting a simple software interface layered on top of an embedded platform, to which the user is not granted

direct access. This lack of visibility inside the devices is a key advantage for adversaries, allowing them to hide their actions among the high-level traffic and evade traditional intrusion detection systems (IDSs). Moreover, most state-of-the-art IoT intrusion detection systems rely on protocol-specific network information which are only effective for specific IoT platforms. [11, 27, 37, 50]. Hence, a protocol-agnostic platform-independent intrusion detection system is needed, which can utilize low-level system information to detect malicious attacks in an IoT network.

To address these emerging threats and shortcomings of existing intrusion detection systems, we present Sentinel, a robust intrusion detection system that leverages low-level system information to detect network attacks in the IoT framework. Sentinel is a protocol-agnostic IDS and is built upon the observation that low-level data obtained by accessing IoT devices' OSes contain specific patterns for different benign device behavior that can be used to detect malicious behavior. Rather than replacing existing security mechanisms, Sentinel introduces a new data aggregation layer to collect system and process-level data (e.g., RAM usage, CPU usage, load, runnig processes, etc.) from the user-space of every IoT device for a low performance cost and a scalable fashion. Sentinel extracts data by installing a Linux Kernel Module on each device, retrieves the latest values of the queried metrics from the kernel using a polling application, and stores PostgreSQL database allowing both local and concurrent access. Sentinel observes the change patterns of the system-level information for different device actions and trains machine learning-based detection techniques to distinguish between benign and malicious device behavior. In Sentinel, the framework utilizes several Machine Learning-based detection techniques to detect malicious attacks including Naive Bayes, Decision Tree, Logistic Regression, and Random Forest classifiers. To test the efficacy, we implemented Sentinel in two different IoT platforms (Home Assistant and WebThings) and collected data from nine different IoT devices. Furthermore, we considered different IoT configurations and ML model parameters to evaluate the performance of Sentinel against five different threats. Our extensive evaluation shows that Sentinel can achieve high accuracy and F-score (over 96%) in detecting threats in an IoT network. Additionally, Sentinel achieves minimum overhead in terms of power consumption indicating efficiency in real-life deployment in IoT devices and networks.

Contributions: Our main contributions are noted as follows.

- We introduce Sentinel, a real-time low-level system monitoring framework for IoT devices that can efficiently collect system and process-level data to detect malicious attacks.
- We designed Sentinel as a platform and protocol-agnostic intrusion detection that can support different IoT frameworks and configurations. Sentinel allows easy integration of new devices by customizing the IoT kernel automatically from the centralized hub.
- We tested Sentinel against five different threats to IoT networks and achieved high accuracy and F-score (over 96%). Our evaluation also indicates that Sentinel introduces low overhead in IoT devices and hubs making it suitable for real-life deployment.

Organization: The rest of the paper is organized as follows: In Section 2, we present the background information and provide an overview of existing threats and solutions to IoT networks. Then, we discuss the problem scope and threat model in Section 3. Section 4 details Sentinel's architecture and Section 5 details the implementation of Sentinel in real-life IoT platforms. In Section 6, we test the efficacy of Sentinel in detecting different attacks in IoT networks. Section 7 discusses the benefits of Sentinel and outlines future research directions. Finally, Section 8 concludes the paper.

2 BACKGROUND AND RELATED WORK

In this section, we first discuss the components of the IoT framework to explain the design approach of Sentinel. We also discuss the related works by outlining threats to the IoT environment and shortcomings of existing intrusion detection systems available for IoT platforms and devices.

2.1 IoT Framework

An IoT framework is a set of systems that connect and establish communication between multiple devices and a centralized access point while providing a unified user interface. All IoT frameworks typically consist of the same core elements:

(1) A hub, which is a device located at the center of the network, in charge of maintaining and controlling a list of connected nodes.
(2) A device API that allows smart things manufacturers to expose their functionalities in a way that can be used by the hub.
(3) A user interface, usually in the form of a smartphone app or a web-app.This interface allows the user to see the state of their devices, control them, add new ones, and create rules to automate their behavior.

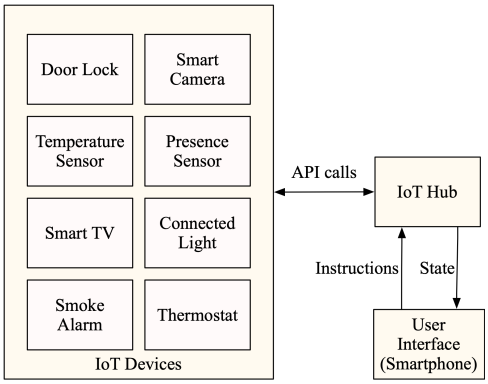


Figure 1: Architecture of an IoT framework

This architecture is presented in Figure 1. In an IoT environment, it is common for the nodes (devices and sensors) to have limited computation power and energy capacity. As these devices are very specialized, they are designed just to fit their power requirements, often leaving very little room for additional software. This implies that any additional added security feature must be as minimal as possible and offload some work to a sturdier machine, be it the hub or a dedicated device.

2.2 Intrusion Detection Systems in IoT

Intrusion detection is a domain of computer security dedicated to monitoring a system for any malicious behavior. Intrusion Detection Systems (IDS) exist to cover different systems and are generally separated into two architectures. Network-based IDS (NIDS) monitor the state of an entire network in search of malicious agents by gathering network-level metrics and processing them at a central location. NIDS can be implemented in the IoT hub [23] or in the cloud [4] to minimize resource overhead at the device level while monitoring the IoT network. Host-based IDS (HIDS) run on a specific host and search for malware operating inside of it through

the use of system-level and process-level information [51]. As IoT devices are usually resource-constrained devices (low processing and power capacity), NIDS is more prevalent in IoT environments because of centralized and cloud implementation capability.

There are three approaches an IDS can use to detect malicious behavior in an IoT environment. Signature-based detection can be performed by comparing the collected data pattern to a list of known malicious signatures of known threats. While very efficient, this type of detector is powerless against zero-day attacks [25]. Anomaly-based detection takes a different path by building an internal representation of the system compared to an expected baseline state. This baseline has to be learned by the system through the observation of known benign behavior. Any discrepancy can then be detected and handled. However, this method is more prone to false positives, as it relies heavily on a statistical approach to the detection [28].

The third intrusion detection logic is specification-based. Similar to anomaly-based detection, the system possesses a set of baseline and threshold values compared to the current situation. While the previous method infers these values from observation, they are manually defined by a human expert in this method. This allows the system to start acting immediately after being turned on, as it does not require any training. Further, the false positive rate is usually lower compared to anomaly-based detection. However, the need for human intervention makes this system poorly scalable, as any change made to the infrastructure will force the rules to be updated [28].

All three approaches rely on the same base principle: collecting actionable data from which decisions are made. However, due to their heterogeneous nature, IoT systems seldom present a standardized access method to this crucial information.

2.3 IoT Network Attacks

In recent years, several attacks in IoT networks and devices have been reported by the research community and developers. IoT attacks are usually sorted into three different categories: node-level, network-level, and application-level attacks [3].

2.3.1 Node-level. These attacks focus on targeting a single device. Due to their low-power nature, IoT devices are very vulnerable to DoS (Denial of Service) attacks. This class of attacks is characterized by an attacker rendering a device unresponsive. This is often achieved by flooding it with requests to saturate its CPU [54]. For battery-powered devices, this can also be done through a "battery-draining" attack, where the attacker sends a constant flow of requests to the device. This prevents it from entering sleep mode, which exhausts its battery at a much higher rate than normal, and causes the device to shut down once it is depleted [22, 52].

Physical attacks take a different path to node-disruption. In this instance, the attacker needs to have physical access to the device to compromise it, but the resulting attacks are much harder to detect. A common type of physical attack is RFID tampering, where the adversary leverages common vulnerabilities in the RFID protocol to disrupt the network [29].

2.3.2 Network-level. Network-level attacks correspond to attacks having an influence on the IoT network as a whole. The most often encountered attacks of this type are routing attacks, which use various methods to manipulate the network flow to the adversary's advantage. These attacks only work in meshed networks, where every device can be responsible for routing packets to their destination. The simplest routing attack is the black hole attack, where a compromised node will advertise itself as the optimal route to every other node and drop every packet received, effectively stopping communications in the network. A grey hole attack operates in the same way but only drops a fraction of received traffic [53].

Another class of network-level attacks, encountered both in traditional networks and in IoT networks, is passive listening attacks. In this scenario, the attacker, having taken over a device, uses it to eavesdrop on the network traffic to gather insufficiently protected sensitive information [1]. This data gathering can also be achieved by exploiting network side-channels, such as packet timings or channel bands.

2.3.3 Application-level. Finally, application-level attacks are designed to disrupt or take down a specific application running on a node (so-called "edge computing" nodes). This can be achieved in many different ways, depending on the application running on the node. For instance, a node can be tricked into downloading a malicious file from the Internet, allowing an attacker to take it over [19]. Another way of targeting an application is by running a Man-in-the-Middle attack. This is done by spoofing a service, the device needs to connect to and using this connection to sniff sensitive information [49]. As these attacks are very dependent on the actual application running on the device, our work does not focus on them.

2.4 Related Work

The idea of collecting low-level host data for intrusion detection purposes is not new. Garfinkel & Rosenblum [16] proposed an IDS architecture relying on this very idea by running the host's application in a virtual machine and exposing low-level information about the application to a local IDS. Such an architecture provides isolation between the potentially compromised host application and the IDS while still giving access to a fine-granularity level of detail. Forrest et al. [15] showed that observing sequences of privileged syscalls made by an application could be used to detect certain classes of attacks reliably. However, it seems that so far, no similar approach has been made for IoT environments.

Many works focused on building IoT intrusion detection systems, most of them based on a network-level approach. *SVELTE* is an IDS focused on 6LoWPAN networks, developed by Raza et al. [37]. This system aims to detect routing attacks, build a map of the meshed network during a learning phase, and then monitor network flow for any behavior not matching this mapping. *INTI* [11] is another 6LoWPAN IDS that detects attacks by having the nodes first categorize themselves in clusters based on their proximity. Packet flow monitoring is then performed at every node and used to calculate a level of trust for each one. When a node's trust dips too low, it is considered compromised and is eliminated from its cluster. *Kalis*, proposed by Midi et al. [27] is a more general-purpose IoT IDS. The core idea is: while most attacks can be detected reliably by an existing IDS, there are no IDS that can identify all attacks on its own. The system learns the specifics of the network (device type, network layout) and uses this knowledge to select the IDS that makes sense in this scenario. Moustafa et al. [31] presented an NIDS for IoT based on statistical flow features. From a given network features dataset, a subset is picked based on the features with

the lowest cross-correlation (i.e. the features the most independent from every other). These features are then passed on to a set of three different machine learning models, which each determine whether the traffic observed is malicious or benign. Their decisions are used in a weighted vote to arrive at a final decision. Anthi et al. proposed a NIDS that monitors the packets in the IoT networks to learn the benign device behavior and detect any malicious packet that could lead to a potential network attack [4]. In a recent work, Sikder et al. developed a context-aware intrusion detection framework *AEGIS*, which observes the user activities in an IoT environment to detect malicious behavior and apps [45, 47]. However, *AEGIS* only considers malicious behavior generated from malware/malicious apps installed in the IoT devices and cannot detect network-level attacks. Several prior works also focused on mitigating Mirai Botnet using network-specific defense strategies [14, 21]. Gopal et al. suggested application whitelisting in an IoT node using program hashes to mitigate the spread of Mirai bot [17]. Al-Qerem et al. proposed a network-based detection method of Mirai botnet using random forest algorithm and features extracted from network packets [2]. Kumar et al. presented a network filtering approach to mitigate DDoS attacks initiated by Mirai [24]. However, no platform and protocol-independent solutions are proposed in prior works that efficiently address different malicious intents of Mirai botnet in the IoT environment.

Differences from existing works. Compared to these prior works, Sentinel offers a platform-agnostic intrusion detection system. The main differences between Sentinel and existing solutions (although they are useful) can be articulated as follows. (1) While most of the prior works focus on network packets to detect attacks, Sentinel considers system and process-level information to detect IoT network attacks. (2) Sentinel does not rely on network protocols, which makes it suitable as a protocol-independent IDS. (3) Sentinel offers both local and remote/cloud-implemented detection methods, which can minimize the overhead significantly in resource-constraint IoT devices. (4) Sentinel only considers system and process-level information to learn benign device behavior, which does not rely on user-defined configurations and installed apps. (5) Sentinel does not store or utilize any user-related information that ensures user privacy in an IoT network.

3 PROBLEM SCOPE AND THREAT MODEL

In this section, we introduce the problem scope and articulate the threat model considered in Sentinel

We assume an IoT home environment (smart home environment) with several devices and sensors connected via a centralized hub. We assume that the attacker has access to the network and can run arbitrary code on any compromised node. Further, the attacker can take over the IoT application itself to make it run arbitrary code. However, we assume that the attacker does not have privileged access, nor can they disrupt the kernel, as it would then be trivial to replace the kernel module with a malicious one that would only report fake information. Regarding the hub, we assume that the attacker does not have any access to the component of Sentinel running on it and cannot alter its behavior nor kill it. We make no assumption regarding the attacker's capability to get access to the information exposed by Sentinel.

Based on the above-mentioned assumptions, the attacker can have a wide array of goals - (1) use the corrupted device to reach other connected device to perform a DDoS attack, (2) generate fake outbound packets to exfiltrate the IoT server/hub, (3) keep the compromised nodes always alive by sending false ping messages periodically to drain the battery, (4) generate a large amount of inbound traffic to make the compromised node unavailable, (5) create packet drop intentionally to disregard any user command or benign device operation. The detailed implementation of these attacks is given in Section 5.4.

4 SENTINEL FRAMEWORK

Sentinel is a novel framework designed to detect node-level and network-level attacks on the IoT environment. The purpose of Sentinel is to serve as a data aggregation platform and train an ML-based IDS using collected system-level information. By providing an IoT framework-agnostic system, it relieves IDS designers from the burden of adapting their methods to the low-level specifics of the network. As IoT is a fundamentally constrained domain, both in computation power and power consumption, Sentinel aims to be as lightweight as possible on the network nodes by offloading most of the heavy work to the hub, a centralized, higher power device.

Sentinel is built upon a Linux Kernel Module running on every IoT device in the monitored network (① in Figure 2), and providing various low-level metrics to the userspace. Each of these devices then runs a data sampling application ② that periodically collects these metrics and sends them over to the IoT hub. In the hub, a data collection component ③, receives and stores this data, and forwards it through an API, to allow an ML-based IDS ④ to use this information.

While the idea of running some part of Sentinel on every device in an IoT network is a reminder of a host-based Intrusion Detection System (HIDS), due to the constraints inherent to the IoT devices, we cannot afford to perform any detection work locally. Instead, we centralize all the data at the hub and run an IDS either there or in the cloud, making our architecture more closely related to a network-based IDS (NIDS).

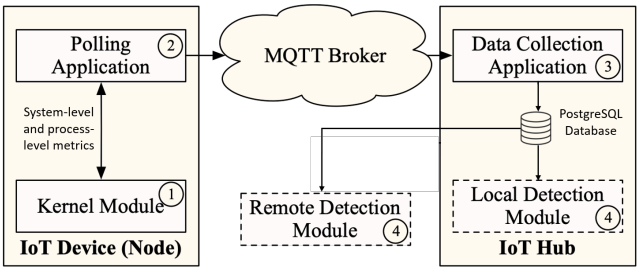


Figure 2: Sentinel framework.

4.1 Kernel Module

The Sentinel Kernel Module (SKM) is in charge of exposing useful low-level metrics to userspace in a machine-friendly and centralized location (e.g., a hub in centralized IoT architecture). The intrusion detection system can then access this data to train the ML model and detect malicious attacks in real-time.

The SKM is a Linux Kernel Module installed on every node that needs to be monitored. The main motivation for choosing a Linux-powered platform is the wide adaptation of Linux OS in IoT devices and open-source functionalities [12]. Currently, Linux has

the highest market share with 43% of all IoT gateways and nodes using different versions of Linux as OS [20]. However, the Sentinel kernel module can be easily implemented in other open-source IoT operating systems (e.g., Contiki, TinyOS, RIOT OS, etc.). For the consumer IoT platforms such as Samsung SmartThings, SKM can be installed as an external app that can access system-level information via IoT hub. Once installed, SKM creates a set of entries in the `sysfs` filesystem, each corresponding to a metric, with its contents mappable to a standard C type. Whenever a file is read, the SKM is informed and fetches the wanted value directly from the relevant kernel data structure. `sysfs` is a RAM-based filesystem, introduced in the Linux Kernel as a replacement for the legacy `profcs`. It provides a file-based view of the kernel data-structures by giving developers an easy interface to export `kobjects`. Each file in `sysfs` corresponds to a single item (e.g. the temperature of a device). These files can be either read-only (if it would not make sense to write to them) or can allow writing to configure the device they relate to [30].

We chose to use a kernel module for two main reasons. First, a kernel module introduces a lower performance overhead than a userspace application, as the transition between userspace and kernel space does not provoke a context switch. Second, a kernel module needs less computing power to perform its task than a regular application, as all the data it needs already exists in the kernel memory and simply needs to be read. Meanwhile, a userspace application would have to parse the output of specific commands or kernel files. SKM uses `sysfs` as this filesystem is considered to be the standard way to expose kernel information to userspace [26].

The implementation of the SKM allows any user on the device to access the exposed data raising privacy concerns as the data consist of sensitive information such as RAM usage, CPU usage, power consumption, etc. One way to remediate this issue would be only to allow certain users to access it, limiting access to only the monitoring application. Another route would be to encrypt the data inside the SKM so only the hub could interpret it after receiving it. This would provide end-to-end confidentiality of the data at the expense of higher computation cost on the node.

The metrics exposed by the SKM are available in Table 1. In addition to the system-level parameters made available by the SKM, we also allow exposing information about an arbitrary process. The target process can be dynamically changed at any time. The process-level values we collect and make available are presented in Table 1. We choose to expose these specific metrics as they are easy to find in the kernel and do not require any complex computation to be obtained, allowing us to keep the performance hit of the SKM minimal. Further, these metrics have been proved to provide actionable information for intrusion detection, as seen in the related work [4]. Adding new metrics to be reported by the SKM is straightforward. It only requires adding a new `sysfs` entry in the module source code and writing a function that fetches the corresponding data from a relevant kernel object. Hence, SKM allows easy customization and adaptability for new configurations of IoT environments and devices.

4.2 Polling Application and Data Sampling

With the data exposed by the SKM, Sentinel can collect it from the node's userspace using the polling application. The polling application (② in Figure 2) periodically reads all the data available and forwards it to the hub.

Type	Metric	Unit
System-level	Number of logical CPUs	N/A
	Frequency of each CPU	kHz
	Total, free, and available RAM	kB
	Total, free, and available swap	kB
	Number of running processes	N/A
	1, 5, and 15 minutes loads	N/A
Process-level	Current physical memory used	kB
	Current virtual memory used	kB
	High-water physical memory	kB
	High-water virtual memory	kB
	Number of file descriptors open	N/A

Table 1: Metrics exposed by Sentinel.

The communication between the nodes and the hub is done via a Message Queue Telemetry Transport (MQTT) layer, a publisher-subscriber protocol commonly found in home IoT networks [38]. In an MQTT system, clients can publish and subscribe to topics (e.g. `living_room/thermostat`). Any message published on a topic will be relayed to its subscribers. In our implementation, the entire MQTT traffic is secured with SSL certificates, both for the server and the clients. This ensures that the reported data is not sent to an attacker masquerading as the broker and prevents an attacker from injecting malicious values.

As MQTT is almost ubiquitous in home IoT environments, having Sentinel piggyback on it helps reduce the need for extra software installation and maintenance. It also provides a scalable platform, allowing an arbitrary number of nodes to send their information to the hub.

The device polling rate plays a vital role in detecting malicious events in IoT networks as it allows the detection module to understand the network's overall status. Setting this rate too high can cause an overabundance of data at the hub and cause high CPU use on the nodes. On the other hand, setting it too low may cause an attack to be missed or detected later than it could have been. To mitigate the variable polling rate, the polling application of Sentinel offers a way to *a fortiori* change the polling rate to allow an IDS to implement some drill-down policies. For instance, the polling rate can be set to a low value in standard conditions and increased whenever suspicious behavior is detected. Sentinel offers both specified and dynamic polling rates in its design. For dynamic polling rate, the low value is chosen based on the training dataset. Sentinel learns the pattern of benign and malicious behavior from the training dataset and identifies minimum possible polling rate to detect attacks with high accuracy.

4.3 Data Collection Module

Once the nodes' data is sent to the MQTT broker via the polling application, the hub needs to notify the broker it wants to receive it. As an IoT network needs to be flexible to allow for devices being frequently added and removed, the data collection application dynamically detects data coming from unknown nodes and can handle nodes reconnecting.

Upon receiving a data record from a node over the MQTT connection, the hub unpacks it and inserts it into a local PostgreSQL database instance. We choose to use a PostgreSQL database as it provides a reliable, production-tested data storage solution. This database also provides an open interface for remote connections. The use of a PostgreSQL database was dictated by the need for

a data access method providing concurrent access, as well as remote access. This allows the intrusion detection module to access and perform real-time monitoring while the data collection application is still collecting more measurements. PostgreSQL being an industry-tested framework, also helps the stability of Sentinel. Hence, Sentinel provides a user-customization option to change and configure the IoT network based on user needs. Sentinel also uses a python library based on SQLAlchemy to specify the data format stored in the database. This library presents an API for easy data retrieval and manipulation, and new custom functions can be designed to fit a user's needs. Having the database instance run locally is convenient as it provides faster access time. However, this causes the hub to be a centralized point of failure, as the loss of the device will cause all the stored data to be lost. For this reason, the data aggregation component is able to use a remote database instead, at the expense of performance.

4.4 Intrusion Detection Module

The intrusion detection module of Sentinel receives the data by accessing the PostgreSQL database. To provide flexible security options and minimize the overhead, Sentinel offers both local detection module and remote detection module for real-time monitoring. For the analytical model, we utilize different machine learning-based detection techniques to differentiate between benign and malicious events in an IoT environment. The main advantage of using machine learning-based techniques is that they are easy to implement in resource-limited devices such as IoT devices [44]. Also, ML-based detection techniques can be trained before implementing in a real-life environment that offers fast detection. For Sentinel, we consider five different ML-based detection techniques.

Naive Bayes: Naive Bayes model is a widely used probabilistic classifier that uses Bayes' method to determine independent relation between features [43]. In Sentinel, the Naive Bayes classifier observes the change of each individual feature in the devices to calculate an event's probability in the IoT environment. In the testing mode, the classifier checks for the presence of a set of specific features and determine the nature of the event (benign/malicious).

Rule-based Learning: In rule-based ML, a set of relational rules are identified between features of the IoT devices [18]. These rules are used to build a single prediction model to determine whether the current status of the IoT environment is benign or malicious. For Sentinel, we used the PART algorithm for rule-based learning.

Regression Model: Regression model implemented in Sentinel builds a prediction model by observing the changes to the dependent variables (benign/malicious state) due to the changes in independent variables (kernel-level features). We use the logistic regression (LR) model in Sentinel, which is commonly used in IDSs [55].

Neural Network: In neural network-based intrusion detection, the relationship between features is compared with the biological neurons, and a relationship map is created based on the effect of variable features in the overall system. As the device features in an IoT environment are non-linear in nature, we chose multi-layer perception (MP) algorithm, which can achieve high accuracy in non-linear dataset [36].

Tree-based classifiers: Tree-based classifiers operate by traversing a binary tree built during a training phase [57]. Starting at the root of the tree, each node corresponds to a comparison of a specific value of the datapoint. Based on the result of this comparison, one of the two branches of the node is taken, and the operation is repeated until it reaches a leaf node. This leaf node corresponds to the class the sample is predicted to belong to. For Sentinel, we considered three different tree-based classifiers (decision tree (DT), random forest (RF), and logistic model tree (LMT)) to compare the outcome of our proposed IDS.

The intrusion detection module collects the data from the database, trains the ML model, and learns the benign device behavior in an IoT network. In the testing phase, the collected data is compared with the benign instances and the intrusion detection module pushes a notification to the user interface via the hub in case of a malicious event.

5 SENTINEL IMPLEMENTATION

We developed Sentinel as a centralized intrusion detection system for the IoT environment. To implement and test the effectiveness of Sentinel, we chose two different IoT platforms - Home Assistant and WebThings. In the following subsections, we provide details of Sentinel's implementation steps in real-life IoT platforms and explain how we orchestrate IoT device behavior during data recording sessions. We also discuss our attack implementation and malicious data collection approaches.

5.1 Testbed Environment

To assess the real-world usefulness of Sentinel, we install Sentinel on two different IoT networks, each of them running a different IoT framework and with a similar device architecture. The two frameworks considered are Home Assistant and WebThings. We only chose open-source hubs as it allows us to customize and implement Sentinel through MQTT broker. Several popular IoT framework such as Samsung SmartThings and Amazon Alexa also offers hub-centric IoT architecture. Though these platforms offer user-defined app development using web IDE, they only offer pre-approved commercial hubs that are closed-source by nature. Hence, it is not possible to install arbitrary kernel modules or software. However, as the commercial hubs run on Linux, we believe it would be possible for Sentinel to be integrated by their manufacturers.

Home Assistant is an open-source framework, providing integration with most commercial IoT devices. A strong emphasis is put on user freedom, allowing them to create their own devices and guaranteeing fully local processing. This framework's hub can be installed on any main OS and provides extensive configuration. Connecting a device to the hub is done by creating an "integration", which defines what interactions are possible with the device, and how the hub can perform them. As creating an integration is a complex task, a common way for enthusiasts to create their own devices is to use the pre-existing MQTT integration. This integration allows the user to define a new device (e.g. a new light) by simply listing its MQTT topic, as well as its capabilities. The only work to be done is then to write the client-side handler that reacts to the MQTT messages coming from the hub and sends back acknowledgments.

WebThings is Mozilla's open-source implementation of the Web of Things (WoT), an initiative aiming to standardize IoT. As with Home Assistant, this framework focuses on giving users the tools they need to configure and control their networks as they want. The WebThings Gateway (WebThings name for a hub) can be run on any Linux machine and provides a local data-processing application. Adding a device to the network simply is as simple as implementing

a few API endpoints and running a web-server on the device. The Gateway then automatically detects and connects to the server. The user only has to program the device to perform its work when a callback function is called.

Figure 3: Floor plan of the experimental testbed

5.2 IoT Network Layout

The network we simulate runs on a set of Raspberry Pi 4, each one representing a different IoT device. At the center of the network is the hub. Every other device runs an implementation of its role in the used IoT framework, as well as Sentinel. To help visualize the set of devices used in our testbed, Figure 3 illustrates an example floor plan of a studio using all of them. The hardware we use to simulate the devices are listed in Table 2.

Device type	Hardware attached
Color light bulb	WS2812 LED strip
Smoke detector	Smoke sensor
Door lock	Servo-motor
Smart TV	N/A
Thermostat	BMP280 sensor and power relay
Weather station	BMP280 sensor
Presence detector	PIR motion detector
Physical switch	Double-throw switch
Outlet	Power relay

Table 2: Types of devices present in the network

5.2.1 Home Assistant Implementation Details. In Home Assistant platform, all the IoT devices are implemented with the default MQTT component. Each device is essentially a Python application that subscribes and publishes to the relevant MQTT topics. For instance, a door lock will interact with the following topics:

- `home/mqtt_lock/available`: whether the device is connected and available
- `home/mqtt_lock/set`: listen for commands coming from the hub
- `home/mqtt_lock/state`: publish its current state, used as an acknowledgment for the hub

All the messages (both from the hub and the devices) are set to be retained by the MQTT broker, so if any party restarts, it is able to assess the current state of the system immediately and set its internal representation accordingly. For instance, if a device stops for any reason, when it restarts and connects to the broker, it will be informed of the last command the hub sent it, and will be able to act accordingly.

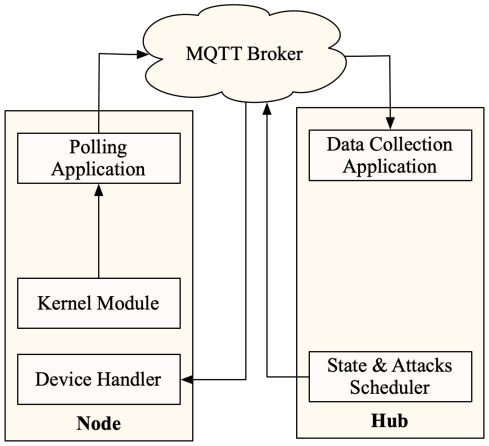


Figure 4: Sentinel instrumented for the experiments

5.2.2 WebThings Implementation Details. WebThings handles devices differently from Home Assistant. While the latter needs to have prior knowledge of the device type and already have a server-side handler (called "component"), the former only needs the nodes to create a simple web-server that exposes a predefined API. Once the hub has been detected and connected to the node's server, it can learn the properties the devices possess (such as OnOffProperty, ColorProperty, etc.). For a device to be recognized as a specific type (e.g. a lamp), it needs to expose a specific set of properties [32].

For instance, a thermostat needs to implement the `TemperatureProperty` and `TargetTemperatureProperty` properties in order to have the "Thermostat" capability. Additionally, the `HeatingCoolingProperty` and `ThermostatModeProperty` properties will also be part of the interface provided by the "Thermostat" capability.

5.3 Simulating the IoT Network

In order to generate the training dataset, we build a home IoT network with standard devices, which we cycle through all the possible combinations of logical states. This is done by enumerating all these combinations in a text file, referred to as a "trace file". This trace is then fed to a scheduling engine that parses it, and executes all the events founds inside. As Sentinel only considers system and process-level data, the user schedule used in a real-life IoT daily routine is irrelevant to the collected data. Instead, the main focus of the training dataset to include any possible events that are logically possible within the IoT network. Hence, we generate and record every device's behavior for every possible configuration it can have. This gives us a clearly labeled set of data that an automated detection system can use. For each combination of attack, device state, and framework, we run each device for 20 minutes and record its metrics with Sentinel. We only collected 20 minutes of traces for attack scenarios for each device. We simulated the same attacks multiple times with different targeted devices, resulting in more than 100 minutes of traces for each attack scenario. For normal operation, we collected 4 hours of devices traces from each device which we used as benign events in machine learning model.

A complete overview of Sentinel instrumented for these experiments is available in Figure 4. The devices used to represent the IoT network in our experiments are Raspberry Pi 4 Model B, with 4GB of RAM. All the Raspberry Pi use a custom Raspbian Lite image,

created with the `pi-gen` tool [13]. The image consists of a Raspbian Lite, to which we add Wi-Fi credentials, customize username and password, enable SSH, and install all the tools that compose Sentinel. Each Python application has a `virtualenv` created where all the required packages are installed. The SKM is compiled, installed, and set to load at boot time. The nodes are connected through Wi-Fi, supported by a D-Link DIR-605L router running firmware v2.09.

5.4 Simulated Attacks

To evaluate how Sentinel can help detect IoT network attacks, we need to observe the data it collects while running an attack, or at least simulating the side effects of said attack. Here, we turn to Mirai to identify the side effects of said attacks.

Mirai is the latest significant IoT botnet in history, having infected hundreds of thousands of devices over a few months and using them to launch large-scale DDoS attacks. Mirai infects devices by using default or common login-password combinations. A Mirai bot works by scanning for vulnerable machines. Once a vulnerable machine is found, it is reported to the Mirai "C&C" server. The C&C server then sends this information to a different server that hosts an SQL database. Finally, it notifies a third server, the "loader", which exploits the vulnerability, sends the payload, and executes a new instance of the bot[5]. The separation of C&C and loader is done in order to prevent the botnet from being taken down if the C&C is blocked. Furthermore, the loader is not referenced by IP address but by the domain name in the bot executable, allowing for a short downtime in case the C&C is targeted. The attacker only has to spawn a new machine and update the DNS records to point to it, restoring the connection to the entire botnet.

From the Mirai botnet attack, we can extract three main side effects of interest: the network scanning phase, the C&C connection, and the new target reporting. We then focus on these three behaviors, which also happen to be common traits of IoT network attacks in our evaluation. We decided to simulate Mirai's side effects rather than actually run it because the minute details of the attack are not what Sentinel focuses on. Rather, we implemented these behaviors' side effects, as would be seen on an infected device. Further, we also implement the effects of a black/grey hole attack, as it is a commonly encountered behavior in IoT network intrusions.

(1) **Network Scan / Pivoting.** A pivoting action consists of scanning the newly reachable network with Nmap. To be as realistic as possible, we do the same here: the attack ping device continuously scans a server (attack 1).

(2) **Exfiltration** For this behavior, the side effects are somewhat the inverse of the previous one: a large amount of outbound traffic and no increase in inbound. We simulate this by sending large UDP packets to a server that discards them (attack 2).

(3) **C&C Keep-alive** For the attacker to keep control of its infected devices, they need to periodically exchange a heartbeat message to confirm that it is still reachable and compromised. This is simulated by periodically pinging a remote machine that responds with an empty payload (attack 3).

(4) **Black/Grey Hole Attack** The side effect of such an attack is a large amount of inbound network traffic and a small amount of outbound traffic. This is simulated by having the device connect to a server and the server sending a large message (≥1MB) in response (attack 4). In addition, for a grey hole attack, a random amount of received messages will be sent out to simulate the partial packet drop created by the attack (attack 5).

6 EVALUATION

Our evaluation aims to test the efficacy of Sentinel in detecting known IoT vulnerabilities and overhead in real-life IoT systems. Here, we specifically focus on the following research questions:

RQ1 What is the performance of Sentinel in detecting different attacks in IoT environment? (Sec 6.2)

RQ2 What is the impact of detection methods' parameters on the performance of Sentinel? (Sec. 6.3)

RQ3 What is the impact of IoT devices and platform configuration on the performance of Sentinel? (Sec. 6.4)

RQ4 What is the impact of Sentinel in IoT devices in terms of power consumption? (Sec. 6.5)

6.1 Evaluation Setup and Methodology

As explained in Section 5, we built an IoT environment consisting of nine different types of devices and implemented Sentinel to collect kernel-level data. We observed that several features (`nb_cpus`, `free_ram`, `total_ram`, `free_swap`, `total_swap`, `tracked_pid`) remain constant regardless of device state and ongoing activity. Hence, we performed a feature pruning process to remove constant features from the collected data. We created a dataset for each node/device and normalized each data point between 0 to 1 based on the minimum and maximum values. The only two exceptions are RAM and swap usage, which are expressed as a percentage of total use. The datasets created contain the samples recorded every second over the time window of the experiment and are labeled if there is an attack or not (binary classification), or which type of attack is underway with a "no attack" type (multi-class classification).

For the attack dataset, we simulated the behavior of the Mirai botnet in the IoT environment and collected data for four different types of attacks. We identified three main effects of Mirai botnet (network scanning, C&C connection, new target reporting) and simulated the behaviors in both WebThings and Home Assistant platforms. We do acknowledge that the experiments used to simulate the Mirai botnet are artificial. Our main objective was to have complete control over the experimental parameters and prevent the need to run actual malware, with all the risks it could involve. While each simulation does not represent a complete attack on its own, they all embody a characteristic trait of the Mirai botnet. Focusing on one side-effect at a time allows us to demonstrate that all of them are susceptible to detection by an IDS based on our framework. In reality, Mirai botnet can have evasive techniques such as generating a lower amount of and smaller UDP packets and sending very low-frequency heartbeat messages. However, Sentinel can still detect these evasive techniques as Sentinel learns benign behavior from the training dataset and correlates the subtle change in network traffic-based kernel information. Hence, our simulated attacks can successfully replicate the behavior of the Mirai botnet.

The collected data by Sentinel are used to train different ML algorithms to detect attacks in the IoT environment. Here, we selected four types of ML classifiers - Naive Bayes, rule-based learning (PART), regression model (logistic regression), neural network (multi-layer perception), and decision tree (decision tree, random forest, and logistic model tree). For our purposes, we trained the ML models using stratified k-fold validation. In this training method, the data is split into multiple subsets (called "folds"), while making sure that each class is evenly represented across each set. Each of the folds is then split randomly into an 80%/20% partition. We

	WebThings							Home Assistant						
ML Algorithm	**TPR**	**FNR**	**TNR**	**FPR**	**Acc.**	**F-Score**	**Avg. CT (s)**	**TPR**	**FPR**	**TNR**	**FNR**	**Acc.**	**F-score**	**Avg. CT (s)**
Naive Bayes	0.8	0.2	0.94	0.06	0.87	0.864	21.6	0.77	0.23	0.92	0.08	0.845	0.838	27
PART	0.85	0.15	0.94	0.06	0.895	0.892	24.5	0.75	0.25	0.88	0.12	0.815	0.809	34.6
LR	0.91	0.09	0.9	0.1	0.905	0.905	34	0.88	0.12	0.91	0.09	0.895	0.894	48
MP	0.89	0.11	0.95	0.05	0.92	0.919	68.5	0.86	0.14	0.94	0.06	0.9	0.898	81.7
DT	0.95	0.05	0.97	0.03	0.96	0.959	35.6	0.92	0.08	0.95	0.05	0.935	0.934	51.5
RF	0.95	0.05	0.98	0.02	0.965	0.964	87.9	0.91	0.09	0.97	0.03	0.94	0.939	94
LMT	0.94	0.06	0.92	0.08	0.93	0.92	102.5	0.92	0.08	0.95	0.02	0.93	0.929	112

Table 3: Performance of Sentinel in binary classification.

	Decision Tree						Random Forest					
	Attack 1	**Attack 2**	**Attack 3**	**Attack 4**	**Attack 5**	**No Attack**	**Attack 1**	**Attack 2**	**Attack 3**	**Attack 4**	**Attack 5**	**No Attack**
Attack 1	98.76	0.17	0.02	0.00	0.00	1.06	98.51	0.42	0.05	0.00	0.00	1.02
Attack 2	0.167	96.13	0.74	0.20	0.11	2.65	0.27	97.42	0.63	0.17	0.11	1.40
Attack 3	0.00	0.00	96.19	0.35	0.02	3.33	0.00	0.00	96.84	0.47	0.02	2.67
Attack 4	0.00	0.17	0.48	96.56	0.15	2.65	0.00	0.17	0.89	96.71	0.15	2.08
Attack 5	0.02	0.00	0.04	0.07	97.46	2.41	0.00	0.00	0.14	0.15	97.03	2.69
No Attack	0.05	0.26	0.18	0.17	0.20	99.15	0.08	0.39	0.12	0.17	0.27	98.97

Table 4: Confusion matrix for WebThings multi-class classification.

	Decision Tree						Random Forest					
	Attack 1	**Attack 2**	**Attack 3**	**Attack 4**	**Attack 5**	**No Attack**	**Attack 1**	**Attack 2**	**Attack 3**	**Attack 4**	**Attack 5**	**No Attack**
Attack 1	99.35	0.13	0.00	0.00	0.00	0.52	99.12	0.13	0.00	0.00	0.00	0.75
Attack 2	0.00	91.31	0.41	0.00	0.00	8.28	0.00	93.87	0.74	0.00	0.00	5.39
Attack 3	0.04	0.43	96.67	0.04	0.00	2.83	0.06	1.06	97.08	0.12	0.00	1.74
Attack 4	0.00	0.00	0.13	99.11	0.02	0.74	0.00	0.00	0.17	98.75	0.14	0.94
Attack 5	0.00	0.00	0.00	0.00	98.15	1.85	0.00	0.00	0.06	0.07	98.09	1.78
No Attack	0.04	1.36	0.15	0.06	0.09	98.31	0.09	0.87	0.16	0.08	0.12	98.68

Table 5: Confusion matrix for Home Assistant multi-class classification.

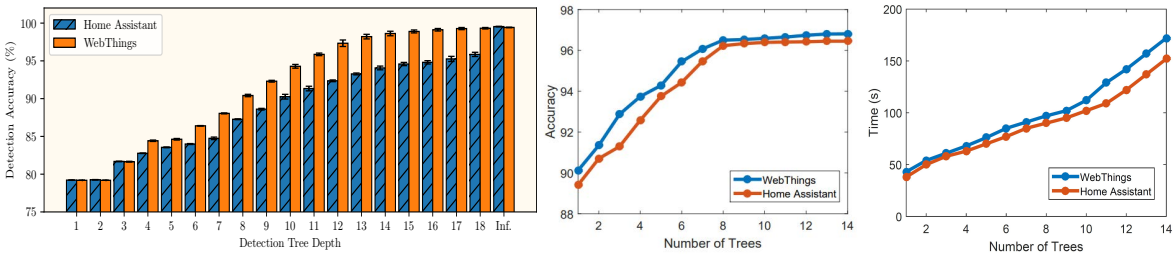
train the model on the first part, and evaluate it on the second. This method of training is useful in proving that the results obtained are not due to random chance and are consistent across multiple independent training passes. Finally, to evaluate Sentinel, we considered seven performance metrics: True Positive Rate (TPR), False Negative Rate (FNR), True Negative Rate (TNR), False Positive Rate (FPR), Accuracy, F-score, and Average Computaiton Time (Avg. CT)

6.2 Detecting Attacks in IoT Environment

We built Sentinel as an ML-based IDS to detect different attacks in IoT environment due to fast computing, easy implementation, and high detection accuracy of machine learning algorithms. In this section, we evaluate the performance of Sentinel using different machine learning algorithms against five different attacks. For ML-based IDS, both supervised and unsupervised learning are popular among the research community. We observe that kernel level information in IoT devices are correlated and has a small degree-of-change for benign and malicious activities. Also, malicious activities in IoT system follow a specific pattern which can be detected by supervised learning with high accuracy. Hence, we chose supervised learning as the analytical model of Sentinel. We consider two important criteria (1) performance using binary classification (2) performance using multi-class classification. For both binary and multi-class classification, we selected seven different ML algorithms. Our selected ML algorithms cover Naive Bayes, Rule-based learning, Regression model, Neural Network, and tree-based classifiers. These ML algorithms are widely used to build intrusion detection system and are suggested by researchers and security practitioners [36, 43].

Binary classification: In binary classification, our main goal is to determine whether the state of the IoT environment is benign or malicious. Here, the state of the IoT environment represents the overall status of all the installed IoT devices. We tested the performance of Sentinel in two different IoT platforms - WebThings and Home Assistant. Table 3 presents the overall performance of Sentinel in different binary ML classifiers. We can observe that Sentinel achieves the highest accuracy and F-score of 96.5% and 96.4%, respectively, using the Random Forest classifier in the WebThings platform. However, the computation time for RF is 87.9s, which higher than the other classifiers. Compared to the RF classifier, the decision tree achieves similar accuracy (96%) with a low computation time of 35.6s. RF classifier also outperforms other ML algorithms in the Home Assistant platform with accuracy and F-score of 94% and 93.9%, respectively. In Home Assistant platform, the decision tree also achieves the accuracy and F-score of 93.5% and 93.4% with lower computation time (51.5s) than the RF classifier. Among other classifiers, LMT, MP, and LR achieve accuracy and F-score over 89% in both WebThings and Home Assistant platforms. We can also observe that Naive Bayes and PART can achieve accuracy and F-score over 81% and 80%, respectively, with lower computation time than other ML algorithms. In summary, Sentinel can achieve high accuracy and F-score (over 94% and 93%, respectively) in detecting different attacks using tree-based classifiers.

Multi-class classification: In multi-class classification, Sentinel aims to detect specific attack in an IoT environment. Here, we considered decision tree and random forest classifiers as Sentinel achieves high accuracy in binary classification using these ML algorithms. Table 4



(a) (b) (c)

Figure 5: Impact of model parameter in Sentinel: (a) tree depth vs accuracy using decision tree, (b) number of tree vs accuracy using random forest, and (c) number of tree vs computation time using random forest.

and 5 presents the performance of Sentinel using a multi-class classification in WebThings and Home Assistant platform, respectively. We can observe that Sentinel can detect different attacks in the IoT environment with high accuracy (average of 97% using DT and RF) in the WebThings platform. From the confusion matrix presented in Table 4, it is evident that Sentinel reports low false positive and negative cases (less than 4% FPR and FNR) in multi-class classification compared to binary classification. Table 5 also shows the high accuracy of Sentinel in the Home Assistant platform in multi-class classification. Here, the average accuracy of Sentinel is over 96% in both decision trees and random forest classifiers. We can also observe that in both WebThings and Home Assistant platform Sentinel achieves the highest accuracy in detecting network scan/pivoting actions (Attack 1) while the lowest accuracy is achieved in detecting exfiltration behavior (attack 2). In summary, Sentinel achieves better performance in multi-class classification than binary classification.

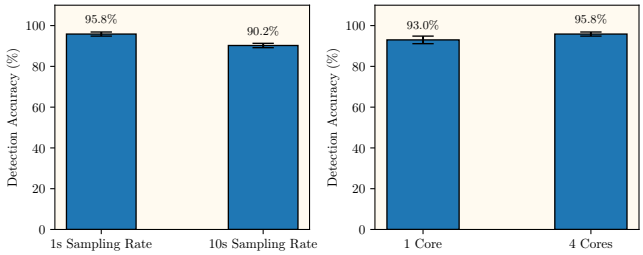
6.3 Impact of Model Parameters

In this sub-section, we test the efficacy of Sentinel in different model parameters. As we built Sentinel as a ML-based intrusion detection framework, model parameters (e.g., tree-depth, number of layers in neural network, etc.) have impact in accurately detecting attacks in IoT environment. Here, we selected random forest and decision tree classifiers as detection methods for Sentinel. To determine the best value for the depth of the Decision Trees, we trained multiple models: one per possible depth value (between one and the total number of metrics), and one with no depth limit. Figure 5a shows the accuracy of Sentinel in Home Assistant and WebThings platforms with different decision tree depths. Here, one can observe that the accuracy of Sentinel increases with the number of tree depths. For WebThings platforms, accuracy converges after a depth of 14. However, in the Home Assistant platform, The highest accuracy is achieved with no depth limit (near 97% accuracy). For random forest classifiers, the accuracy of Sentinel also increases with the number of trees (Figure 5b). However, the computation time increases significantly with the number of trees in random forest classifiers. Figure 5c shows that the accuracy of Sentinel becomes steady (96.5%) after eight trees in random forest classifiers with an average computation time of 102s. However, random forest achieves the highest accuracy of 96.83% with 12 trees, which increases the computation time to 163s. Here, the improvement in accuracy is insignificant compared to the computation time. In short, Sentinel can perform with high steady accuracy without increasing the computation time significantly.

6.4 Impact of IoT Platform Configurations

To evaluate the performance of Sentinel in different IoT configurations, we considered variable sampling rate and processing capacity of the IoT devices. Here, we considered the decision tree model as it achieves high accuracy in both binary and multi-class classification with low computation time.

Impact of sampling period: In order to characterize how the accuracy of *Sentinel* is impacted by its sampling period, we run two data collection runs. The first one has a polling rate of one second, and each state is held for one minute. The second is done with a period of ten seconds and a state duration of ten minutes. These configurations provide us with the same amount of measurements in both sets, guaranteeing that any difference found is not caused by the variability of the training dataset sizes. The results of this experiment are presented in Figure 6a.



(a) (b)

Figure 6: Detection Accuracy for (a) different polling rate (1s and 10s), (b) different computation power (1 and 4 cores).

When the polling rate is decreased to ten seconds, the detector accuracy decreases by 5.6 percentage points. While the more extended sampling period comes with a potentially increased reaction time to an attack, Sentinel comes with a drill-down capacity, allowing dynamically changing individual devices' polling rate. It is then possible to design a logic that temporarily decreases the sampling period if suspicious behavior is noticed.

Impact of processing capacity. The Raspberry Pi 4, which was used to represent an IoT device, is significantly more powerful than the average IoT device, which is usually a single-purpose, single-core device. The Raspberry Pi 4 has a 1.5GHz four-core CPU, which could have an effect on *Sentinel*. To evaluate how this impacts its performance, we perform a complete experiment (collecting training and test data) while disabling three of the four cores of each node. The data is gathered with one second between each sample and one minute per state. We then train a new Decision

Tree on this dataset and compare its precision to the four core dataset. The results of this experiment are available in Figure 6b. With a core count reduced to one per device, the attack detection accuracy is decreased by 2.8 percentage points. This difference comes from the fact that, as the device now has to share a single core for all its processes, some metrics are not as insightful as with four cores. For instance, the `cpu_load` metric that records the one-minute CPU load of the device now has an increased value outside of an attack. When an attack occurs, its increase is consequently less noticeable. However, it is to be noted that the detector performance was not degraded to useless a level by this change. In summary, *Sentinel* can effectively run on a low core-count IoT device.

6.5 Power Consumption Analysis

One major constraint for IoT devices is power use. Some devices are designed to be battery-powered and be able to last for months or years on their supply. To inspect how *Sentinel* can impact the battery life expectancy of these devices, we perform a power consumption analysis. This is done by running the device handler on a Raspberry Pi with and without *Sentinel*, in different logical states and with various reporting periods. The power draw is measured at the outlet with a wattmeter in which only the Pi is plugged in. The results of the experiment are available in Figure 7. For a reporting period of one second, *Sentinel* causes an increase of power use of about 10%, which will definitely impact the lifetime of a battery-powered device. However, as the polling frequency decreases, the power consumption overhead incurred decreases too. As such, for ten seconds reporting delay, the overhead dips below 1%. This means it is possible to find a trade-off between battery life and enhanced device security by varying the sampling rate.

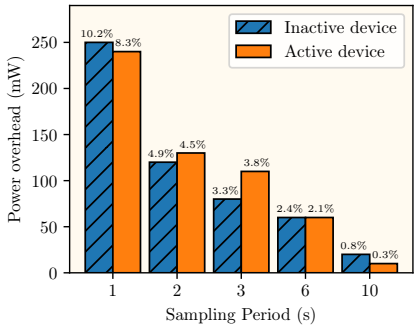


Figure 7: Power overhead caused by Sentinel for various polling periods, expressed as absolute and relative values

One interesting observation is that power overhead is higher in inactive devices for some cases. Most IoT devices switch to sleep mode in order to save power in inactive phases. This certainly improves the performance of IoT devices in terms of resource consumption. However, for collecting kernel-level data, SENTINEL changes the device state from sleep mode, spiking power consumption. This results in high power overhead in inactive devices in certain cases. SENTINEL can minimize this overhead by simply correlating running processes with devices to invoke a high polling rate.

6.6 Effect of Dynamic Polling Rate

As mentioned earlier, the performance of SENTINEL depends on the polling rate of SKM. This introduces an accuracy-overhead trade-off in SENTINEL. From Section 6.4 and 6.5, we observe that accuracy and power consumption are proportional for different polling rates. While higher sampling frequency provides high accuracy, it introduces high power consumption in IoT systems. To mitigate this trade-off, SENTINEL introduces a dynamic polling rate where sampling frequency automatically adjusted based on detected malicious state. At initial stage, SENTINEL sets low sampling frequency (polling rate 10s) and switches to high sampling frequency (polling rate 1s) after detecting a malicious event. Figure 8 reports accuracy and power consumption of fixed and dynamic polling rate. We can observe that while a fixed polling rate of 1s provides over 95% accuracy, it introduces additional 438 mW power consumption as overhead. Compared to this dynamic polling rate (from 10s to 1s transition) ensures over 93% accuracy with only 110 mW power consumption as overhead. In summary, the dynamic polling rate introduced by SENTINEL can effectively detect malicious events in IoT platforms with high accuracy and low overhead.

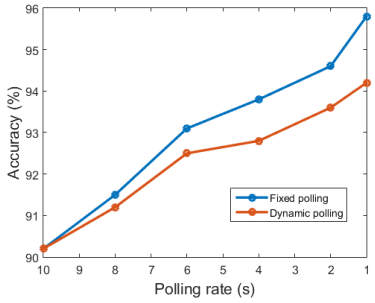


Figure 8: Fixed polling vs dynamic polling in SENTINEL

7 DISCUSSION

This section illustrates the benefits of SENTINEL in real-life deployment and how deploying SENTINEL in a smart home can help different groups of consumers using several use scenarios and discuss different benefits of SENTINEL.

7.1 Benefits of SENTINEL

Data-centric IoT applications. SENTINEL proposes an IDS that includes a polling application to collect kernel-level information. Hence, SENTINEL can be implemented as a lightweight framework-agnostic data-collection system as well as an IDS. Also, SENTINEL can be used as a system monitoring framework for large IoT networks such as industrial IoT systems [39].

User-customized Apps. SENTINEL implements an SQL database to collect and store data from the IoT devices. This provides a well-known low-level API to allow any user to create their own data access functions without relying on the methods provided by SENTINEL. Hence, users can use the implemented database to build customized apps.

Easy deployability in real-life. IoT environment such as smart home varies in platforms, installed devices, and apps. As the IoT environment depends on users' needs and choices, a centralized IDS like SENTINEL needs to be easily scalable for different configurations. Adapting *Sentinel* for a new set of devices may require new features to be added to detect malicious device behavior efficiently. This can be done relatively easily by editing the source code of SKM. Further, as `sysfs` is passive (i.e. the SKM only does some work when a file is read), there is no issue with adding more features than required, as the data polling application can simply ignore the unused ones.

Device independent. As the iteration of Sentinel depends on every device running Linux and being able to communicate over MQTT, it does not faithfully represent the totality of the IoT devices available currently. However, we do not believe this to be an issue in real-life deployment for multiple reasons:

- Any IoT device that does not run Linux will most likely run a very limited firmware, making it irrelevant to the attacks considered in this paper.
- Sentinel could be modified to work not only with MQTT but also with other non-WiFi-based IoT protocols (e.g., Zigbee, BLE).

Hence, Sentinel can be easily deployed in IoT environments irrespective of operating system and communication protocols.

Inherent privacy features. The MQTT component of *Sentinel* relies on server-side and client-side certificates in order to provide adequate security properties to ensure data privacy. For the purposes of our experiments, the certificates were manually generated and distributed, for the sake of convenience. However, in a real-world environment, where the IoT network would need more flexibility, the certificate distribution can be performed through a centralized authority [7]. As Sentinel collects sensitive device information to detect malicious attacks to the IoT devices and network, the inherent MQTT security component ensures complete user privacy.

Fault detection in the IoT environment. Fault detection is an active domain of IoT research, focusing more on safety, where intrusion detection focuses on security. However, both domains share features, as faults can have similar side effects as network intrusions. For instance, a device getting stuck in an infinite loop can resemble a sleep deprivation attack. As Sentinel collects and observes system-level features to detect malicious attacks, the collected data can be utilized to build an effective fault detection system for the IoT environment.

Data security. Our MQTT setup allows any device with the proper credentials to publish and subscribe to any topic, potentially allowing an attacker to obtain sensitive information by corrupting a node. Most MQTT brokers provide an access control List feature, through which read/write access to topics can be restricted. However, this ACL is static and needs to be edited whenever a new device joins, as it will need its own set of topics to publish [46]. Also, the Sentinel kernel module transmits system-level information from IoT nodes to monitor and detect malicious activities from the centralized hub. This opens up the possibility of passive attacks such as eavesdropping or device fingerprinting which can result in sensitive information leakage [6, 40]. A potential alternative to this unwieldy method could be to add an end-to-end encryption layer on the data by encrypting it in the SKM and decrypting it at the hub. This would prevent any possibility of eavesdropping but would require designing a key agreement method between the kernel module and the hub.

Centralized IDS. We built Sentinel as a centralized IDS to address the resource-constraint problem in IoT platforms. In an IoT environment, several devices may connect with each other to perform various user-defined tasks. Installing standalone IDS for each device is costly in terms of resources (computation power, memory, etc.) as IoT devices are usually resource-constraint devices by nature. Also, most of the consumer IoT platforms (e.g., Samsung SmartThings, Amazon Alexa, WebThings, etc.) offer hub-centric design where IoT devices are connected to a centralized hub. Hence, it is easier and practical to deploy Sentinel as a centralized IDS in a real-life IoT environment. The Sentinel kernel module can also be implemented in each IoT node to collect data which ensures intrusion detection for single device failure. Additionally, Sentinel can be implemented as a cloud-based service to avoid any attacks targeting IoT hub.

7.2 Future Work

Falsified IoT data. Our threat model assumes that the adversary does not have privileged access to the nodes, and as such cannot falsify the data reported by the SKM. However, it could be interesting to investigate if an attacker with root access forcing the node to misreport the system information could impact the detection abilities. In particular, when a device changes state, the attacker would have to adapt its falsification to remain covert. At first glance, this is not trivial, as it requires the knowledge of how the device should behave in every scenario. However, using adversarial machine learning, it is possible to mimic the device behavior to evade ML-based IDS [33, 41]. In future, we will study the effect of adversarial ML on Sentinel and analyze mitigation strategies.

Real-life IoT device behavior. As stated previously, Sentinel can be considered as an automated intrusion detection solution as well as a data-gathering tool that can be leveraged by a centralized monitoring system for the IoT environment. In order to be improved, Sentinel could provide a larger variety of data, which would allow it to be tailored to fit a specific scenario, where some metrics are more relevant than others. However, in order to determine which metrics could benefit from being added to the monitoring capacities of Sentinel, some feedback from actual field experience would be required. Hence, one future research direction can be the empirical study of heterogeneous IoT devices to expand the feature space of Sentinel to improve real-life deployment performance.

8 CONCLUSION

Wide utilization of IoT devices in different application domains have attracted the attackers to target IoT devices and platforms. Due to resource-constraint and opaque implementation, traditional security frameworks fail to detect attacks specific to IoT devices. Also, the lack of security measures from the vendors exacerbates the situation. In this paper, we presented Sentinel, a novel platform and protocol-agnostic intrusion detection system for IoT devices and networks. Sentinel collects low-level system and process-level information by accessing the IoT kernel and learning benign device behavior to detect IoT network attacks. Sentinel utilized different machine learning-based detection techniques to distinguish benign and malicious device behavior. We evaluated Sentinel in different IoT platforms and achieved high accuracy and F-score (over 96%) against different IoT attacks. Moreover, Sentinel introduces low overhead, making it suitable for real-life IoT devices. In future work, we will expand our framework by considering new IoT attacks as well as implementing on commercial IoT devices.

9 ACKNOWLEDGMENT

This work is partially supported by the US National Science Foundation (Awards: NSF-CAREER-CNS-1453647, NSF-1663051, NSF-1705135), US Office of Naval Research grant Cyber-physical Systems, and Cyber Florida's Capacity Building Program. The views expressed are those of the authors only, not of the funding agencies.

REFERENCES

[1] Abbas Acar, Hossein Fereidooni, Tigist Abera, Amit Kumar Sikder, Markus Miettinen, Hidayet Aksu, Mauro Conti, Ahmad-Reza Sadeghi, and Selcuk Uluagac. 2020. Peek-a-Boo: I see your smart home activities, even encrypted!. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 207–218.

[2] Ahmad Al-Qerem, Bushra Mohammed Abutahoun, Shadi Ismail Nashwan, Shatha Shakhatreh, Mohammad Alauthman, and Ammar Almomani. 2020. Network-Based Detection of Mirai Botnet Using Machine Learning and Feature Selection Methods. In *Handbook of Research on Multimedia Cyber Security*. IGI Global, 308–318.

[3] Ioannis Andrea, Chrysostomos Chrysostomou, and George Hadjichristofi. 2015. Internet of Things: Security vulnerabilities and challenges. In *2015 IEEE symposium on computers and communication (ISCC)*. IEEE, 180–187.

[4] Eirini Anthi, Lowri Williams, Małgorzata Słowińska, George Theodorakopoulos, and Pete Burnap. 2019. A supervised intrusion detection system for smart home IoT devices. *IEEE Internet of Things Journal* 6, 5 (2019), 9042–9053.

[5] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the mirai botnet. In *26th USENIX security symposium*. 1093–1110.

[6] Leonardo Babun, Hidayet Aksu, Lucas Ryan, Kemal Akkaya, Elizabeth S Bentley, and A Selcuk Uluagac. 2020. Z-iot: Passive device-class fingerprinting of zigbee and z-wave iot devices. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.

[7] Leonardo Babun, Z Berkay Celik, Patrick McDaniel, and A Selcuk Uluagac. 2021. Real-time analysis of privacy-(un) aware iot applications. *Proceedings on Privacy Enhancing Technologies* 2021, 1 (2021), 145–166.

[8] Elhadj Benkhelifa, Thomas Welsh, and Walaa Hamouda. 2018. A critical review of practices and challenges in intrusion detection systems for IoT: Toward universal and resilient systems. *IEEE Communications Surveys & Tutorials* 20, 4 (2018), 3496–3509.

[9] Z Berkay Celik, Leonardo Babun, Amit Kumar Sikder, Hidayet Aksu, Gang Tan, Patrick McDaniel, and A Selcuk Uluagac. 2018. Sensitive information tracking in commodity IoT. In *27th USENIX Security Symposium*. 1687–1704.

[10] Z Berkay Celik, Earlence Fernandes, Eric Pauley, Gang Tan, and Patrick McDaniel. 2019. Program analysis of commodity iot applications for security and privacy: Challenges and opportunities. *ACM Computing Surveys (CSUR)* 52, 4 (2019).

[11] Christian Cervantes, Diego Poplade, Michele Nogueira, and Aldri Santos. 2015. Detection of sinkhole attacks for supporting secure routing on 6LoWPAN for Internet of Things. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 606–611.

[12] Sam Cook. 2021. *60+ IoT statistics and facts.* https://www.comparitech.com/internet-providers/iot-statistics/

[13] Adrien Cosson. 2020. Fork of Pi-Gen Used to Generate the Sentinel Raspbian Images.

[14] Michele De Donno, Nicola Dragoni, Alberto Giaretta, and Angelo Spognardi. 2018. DDoS-capable IoT malwares: Comparative analysis and Mirai investigation. *Security and Communication Networks* 2018 (2018).

[15] Stephanie Forrest, Steven A Hofmeyr, Anil Somayaji, and Thomas A Longstaff. 1996. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*. IEEE, 120–128.

[16] Tal Garfinkel, Mendel Rosenblum, et al. 2003. A virtual machine introspection based architecture for intrusion detection.. In *Ndss*, Vol. 3. Citeseer, 191–206.

[17] Tatikayala Sai Gopal, Mallesh Meerolla, G Jyostna, P Reddy Lakshmi Eswari, and E Magesh. 2018. Mitigating Mirai malware spreading in IoT environment. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2226–2230.

[18] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. 2007. Bothunter: Detecting malware infection through ids-driven dialog correlation.. In *USENIX Security Symposium*, Vol. 7. 1–16.

[19] Tobias Heer, Oscar Garcia-Morchon, René Hummen, Sye Loong Keoh, Sandeep S Kumar, and Klaus Wehrle. 2011. Security Challenges in the IP-based Internet of Things. *Wireless Personal Communications* 61, 3 (2011), 527–542.

[20] Eclipse IoT. 2021. *IoT Developer Survey 2020.* https://iot.eclipse.org/community/resources/iot-surveys/assets/iot-developer-survey-2020.pdf

[21] LES Jaramillo. 2018. Malware detection and mitigation techniques: lessons learned from Mirai DDOS attack. *Journal of Information Systems Engineering & Management* 3, 3 (2018), 19.

[22] MHR Khouzani and Saswati Sarkar. 2011. Maximum damage battery depletion attack in mobile sensor networks. *IEEE Trans. Automat. Control* 56, 10 (2011), 2358–2368.

[23] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. 2019. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems* 100 (2019), 779–796.

[24] CU Om Kumar and Ponsy RK Sathia Bhama. 2019. Detecting and confronting flash attacks from IoT botnets. *The Journal of Supercomputing* 75, 12 (2019), 8312–8338.

[25] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. 2013. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications* 36, 1 (2013), 16–24.

[26] Linux. 2020. *Rules on how to access information in sysfs.* https://august.com/pages/how-it-works

[27] Daniele Midi, Antonino Rullo, Anand Mudgerikar, and Elisa Bertino. 2017. 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)*. IEEE, 656–666.

[28] Robert Mitchell and Ing-Ray Chen. 2014. A survey of intrusion detection techniques for cyber-physical systems. *ACM Computing Surveys (CSUR)* 46, 4 (2014), 1–29.

[29] Aikaterini Mitrokotsa, Melanie R Rieback, and Andrew S Tanenbaum. 2010. Classification of RFID attacks. *Gen* 15693, 14443 (2010), 14.

[30] Patrick Mochel and Mike Murphy. 2021. *Sysfs - The Filesystem for Exproting Kernel Objects.* https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

[31] Nour Moustafa, Benjamin Turnbull, and Kim-Kwang Raymond Choo. 2018. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things. *IEEE Internet of Things Journal* 6, 3 (2018), 4815–4830.

[32] Mozilla. 2018. *WoT Capability Schemas.* https://iot.mozilla.org/schemas/

[33] AKM Newaz, Nur Imtiazul Haque, Amit Kumar Sikder, Mohammad Ashiqur Rahman, and A Selcuk Uluagac. 2020. Adversarial Attacks to Machine Learning-Based Smart Healthcare Systems. *arXiv preprint arXiv:2010.03671* (2020).

[34] AKM Newaz, Amit Kumar Sikder, Mohammad Ashiqur Rahman, and A Selcuk Uluagac. 2020. A survey on security and privacy issues in modern healthcare systems: Attacks and defenses. *arXiv preprint arXiv:2005.07359* (2020).

[35] AKM Iqtidar Newaz, Amit Kumar Sikder, Leonardo Babun, and A Selcuk Uluagac. 2020. Heka: A novel intrusion detection system for attacks to personal medical devices. In *IEEE Conference on Communications and Network Security (CNS)*. IEEE.

[36] AKM Iqtidar Newaz, Amit Kumar Sikder, Mohammad Ashiqur Rahman, and A Selcuk Uluagac. 2019. Healthguard: A machine learning-based security framework for smart healthcare systems. In *Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. IEEE, 389–396.

[37] Shahid Raza, Linus Wallgren, and Thiemo Voigt. 2013. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad hoc networks* 11, 8 (2013), 2661–2674.

[38] MQTT Github Repository. 2020. *Mqtt/Mqtt.Github.Io.* https://github.com/mqtt/mqtt.github.io

[39] Luis Puche Rondon, Leonardo Babun, Ahmet Aris, Kemal Akkaya, and A Selcuk Uluagac. 2020. PoisonIvy: (In) secure Practices of Enterprise IoT Systems in Smart Buildings. In *Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 130–139.

[40] Shammya Saha, Nikhil Ravi, Kári Hreinsson, Jaejong Baek, Anna Scaglione, and Nathan G Johnson. 2021. A secure distributed ledger for transactive energy: The Electron Volt Exchange (EVE) blockchain. *Applied Energy* 282 (2021), 116208.

[41] Md Hasan Shahriar, Nur Imtiazul Haque, Mohammad Ashiqur Rahman, and Miguel Alonso. 2020. G-ids: Generative adversarial networks assisted intrusion detection system. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 376–385.

[42] Amit Kumar Sikder, Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, Kemal Akkaya, and Mauro Conti. 2018. IoT-enabled smart lighting systems for smart cities. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 639–645.

[43] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. 2017. 6thsense: A context-aware sensor-based attack detector for smart devices. In *26th USENIX Security Symposium*. 397–414.

[44] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. 2019. A context-aware framework for detecting sensor-based threats on smart devices. *IEEE Transactions on Mobile Computing* 19, 2 (2019), 245–261.

[45] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. 2019. Aegis: a context-aware security framework for smart home systems. In *Proceedings of the 35th Annual Computer Security Applications Conference*. 28–41.

[46] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. 2020. Kratos: multi-user multi-device-aware access control system for the smart home. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 1–12.

[47] Amit Kumar Sikder, Leonardo Babun, and A Selcuk Uluagac. 2021. Aegis+ A Context-aware Platform-independent Security Framework for Smart Home Systems. *Digital Threats: Research and Practice* 2, 1 (2021), 1–33.

[48] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A Selcuk Uluagac. 2018. A survey on sensor-based threats to internet-of-things (iot) devices and applications. *arXiv preprint arXiv:1802.02041* (2018).

[49] Ivan Stojmenovic, Sheng Wen, Xinyi Huang, and Hao Luan. 2016. An overview of fog computing and its security issues. *Concurrency and Computation: Practice and Experience* 28, 10 (2016), 2991–3005.

[50] M Surendar and A Umamakeswari. 2016. Indres: An intrusion detection and response system for internet of things with 6lowpan. In *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE, 1903–1908.

[51] John R Vacca. 2013. *Computer and Information Security Handbook*. Elsevier Science & Technology Books, San Diego.
[52] Eugene Y Vasserman and Nicholas Hopper. 2011. Vampire attacks: Draining life from wireless ad hoc sensor networks. *IEEE transactions on mobile computing* 12, 2 (2011), 318–332.
[53] Linus Wallgren, Shahid Raza, and Thiemo Voigt. 2013. Routing attacks and countermeasures in the RPL-based internet of things. *International Journal of Distributed Sensor Networks* 9, 8 (2013), 794326.
[54] Anthony D Wood and John A Stankovic. 2002. Denial of service in sensor networks. *computer* 35, 10 (2002), 54–62.
[55] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *Seventh Asia Joint Conference on Information Security*. IEEE, 62–69.
[56] Jacob Wurm, Khoa Hoang, Orlando Arias, Ahmad-Reza Sadeghi, and Yier Jin. 2016. Security analysis on consumer and industrial IoT devices. In *21st Asia and South Pacific Design Automation Conference*. IEEE, 519–524.
[57] Yanfang Ye, Dingding Wang, Tao Li, and Dongyi Ye. 2007. IMDS: Intelligent malware detection system. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1043–1047.