An Attack-Resilient Architecture for the Internet of Things

Hussain M. J. Almohri, *Member, IEEE,* Layne T. Watson, *Life Fellow, IEEE*, David Evans

***Abstract*—With current IoT architectures, once a single device in a network is compromised, it can be used to disrupt the behavior of other devices on the same network. Even though system administrators can secure critical devices in the network using best practices and state-of-the-art technology, a single vulnerable device can undermine the security of the entire network. The goal of this work is to limit the ability of an attacker to exploit a vulnerable device on an IoT network and fabricate deceitful messages to co-opt other devices. The approach is to limit attackers by using device proxies that are used to retransmit and control network communications. We present an architecture that prevents deceitful messages generated by compromised devices from affecting the rest of the network. The design assumes a centralized and trustworthy machine that can observe the behavior of all devices on the network. The central machine collects application layer data, as opposed to low-level network traffic, from each IoT device. The collected data is used to train models that capture the normal behavior of each individual IoT device. The normal behavioral data is then used to monitor the IoT devices and detect anomalous behavior. This paper reports on our experiments using both a binary classifier and a density-based clustering algorithm to model benign IoT device behavior with a realistic test-bed, designed to capture normal behavior in an IoT-monitored environment. Results from the IoT testbed show that both the classifier and the clustering algorithms are promising and encourage the use of application-level data for detecting compromised IoT devices.**

***Index Terms*—Internet of Things, Intrusion detection, Unsupervised learning, Network security**

I. Introduction

The Internet of Things (IoT) connects physical devices to the Internet. Applications deployed on the Internet of Things include devices that are capable of monitoring and capturing events, making decisions according to specifications set by vendors or system administrators, and coordinating with other devices to apply application-specific policies. IoT device vendors typically use conventional open-source software, mainly modified to suit the purpose of IoT applications, to operate the devices. For example, Amazon FreeRTOS [1], Android Things [2], Raspbian [3], Embedded Linux [4], and Riot [5] are among the growing number of modifiable and open-source IoT operating systems.

H. M. J. Almohri is with the Department of Computer Science, Kuwait University, Kuwait. Email: almohri@ieee.org

L. T. Watson is with the Departments of Computer Science, Mathematics, and Aerospace and Ocean Engineering, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061. Email: ltw@ieee.org

David Evans is with the Department of Computer Science, University of Virginia, Charlottesville, VA 22903. Email: evans@virginia.edu

This work was supported and funded by Kuwait University, Research Project No. (RQ01/18)

Open-source software modules, although widely deployed and well-tested, often have known and zero-day vulnerabilities which attackers can exploit to achieve unrestricted remote code execution. For example, in 2019, embedded Linux was reported to have a root account vulnerability [6]. Such vulnerabilities may endure on many devices as system administrators miss the opportunity to patch their devices promptly. Attackers can perform reconnaissance operations on the target network to discover the type of software used and find remote exploitation vulnerabilities [7], [8]. With access to a vulnerable IoT device, the attacker can launch attacks from malicious nodes, disturbing trust in the network. Precisely, attackers can manipulate the application on the compromised device to send deceitful messages to other devices that will cause them to behave in ways that achieve the attacker's goals, even though those devices have not been compromised directly. Thus, a single remotely compromised device can undermine the security of the entire IoT subnetwork.

This work addresses the problem of securing an IoT subnetwork by using *application-level* data generated on individual IoT subnetworks and providing an architecture for limiting and controlling access to the devices. Our solution aims to tackle the security problem at the application layer and leverages the role of the IoT subnetwork in monitoring the security status, detecting vulnerable devices and healing from attacks. IoT devices primarily exchange application data with remote clients, which can be used to reason about the security of the device itself. As suggested in several previous works [9]–[17], the data generated on IoT devices can be used to learn models for the behavior of malicious and benign devices. The hypothesis is that machine learning techniques can be utilized in an IoT subnetwork to determine the security status of each device (secure or compromised). Once a compromised device is detected, the IoT subnetwork can defend itself against intruders by controlling the interface to IoT devices (for example, by using virtual drivers [18]).

Several previous works provide models for analyzing data collected from IoT network traffic. Meidan et al. propose to use device white-listing by using classification [9]. DÏoT [10] (and other similar works, such as [11]–[17]) improve anomaly detection by using probabilistic classification of network packets. Choi et al. present a sensor activity correlation model that requires no manual labeling but uses a bit-wise representation of normal sensor activity and activity transition probabilities as ground truth [19]. Sikder et al. describe several machine learning models for analyzing sensor-level data collected from mobile devices [20]. Their work is closest to the modeling part of our work as it uses actual user behavior data in a

classification model against malicious behavior.

In addition to data analysis, architectural solutions are needed to secure networks against malicious devices. Barrera et al. argue that well-established models for limiting network-level activities by profiling IoT application behavior can be effective [21]. Deadbolt uses virtual drivers to protect IoT devices, for example, by providing encryption for network traffic out of IoT devices [18].

We present an *attack-resilient IoT architecture*, which extends the state of the art in two directions. First, the existing IoT data analysis frameworks mainly focus on network traffic data and apply conventional anomaly detection techniques. While these solutions are useful, they miss the information from the application layer. Our hypothesis is that application-level data assists in reasoning about the validity of data exchanges across the network. Compared to miscellaneous network traffic, application-level data is more vibrant and has significantly less noise. Application-level data modeling can be coupled with other models, such as analyzing malicious activities (e.g., DÏoT [10]) to provide a comprehensive data model for preventing large-scale attacks against IoT subnetworks. Second, previous works do not integrate data analysis with restricted network architecture. Restricting the network architecture simplifies the data analysis to enable a simple policy system to isolate suspicious IoT devices. The architecture and data model provided in this article attempt to address these two issues.

In our attack-resilient IoT architecture, a trusted controller machine acts as a proxy to capture all device communication, as well as application data and metadata (such as file attributes) from all IoT devices. Security of the network depends on that controller machine not being compromised, and preventing a compromised device within the network from being able to compromise other devices. The collected data represents events, such as motion, that are captured on individual IoT devices. Each IoT device sends events data in the form of event sequences. Each event sequence represents several events within a specific interval. Our model of event sequences provides rich metadata that can be used to guide the analysis of anomalous behavior across the IoT subnetwork. Compromised devices are assumed to generate deceitful event sequences to pollute the data collected from devices and confound the trusted controller. Detecting compromised devices requires controlling the traffic to and from devices and authenticating data from devices. In our model, the controller authenticates event sequences by requiring application data to be signed using verified device keys. As opposed to application instrumentation proposed by Wang et al. [22], our attack resilient IoT architecture uses a simple message signing protocol for authenticating application data.

In summary, the contributions of this work are:

1) an attack-resilient IoT architecture in which a centralized controller dynamically controls and manages IoT device interactions, to provide security for the IoT network even when some devices are vulnerable to remote exploitation attacks (Section III),
2) a model of IoT application data and a data provenance verification method based on a simple message signing protocol (Section IV-A),
3) two anomaly detection models, based on classification and clustering (Section IV-B), and
4) a prototype implementation (Section VI) and experimental evaluation using an IoT testbed (Section VII). Evaluation results show that on average 76% of the attacker's messages originating from a compromised device can be detected. The best success rate of our models was 91% on a 30-day dataset collected from an actual IoT environment.

Next, we present the necessary background and related work on securing IoT subnetworks.

II. Background and Related Work

This section presents background on the Internet of Things, followed by a discussion of the related work.

A. Internet of Things

We adopt a practical definition of the Internet of Things (IoT), among the existing many [23], that characterizes target IoT networks. We define *things* as devices that are designed to be connected to the Internet with full or semi-automated interactions with external entities (devices or humans). These things are interchangeably referred to as *devices* or *nodes* throughout this article. Because of Internet connectivity, IoT devices enable a range of applications, for example, providing real-time monitoring for human operators outside the target network. An *IoT subnetwork* is formed by a set of collaborative devices that are interconnected to serve a specific purpose.

IoT subnetworks often use a centralized control engine entity that interacts with all IoT devices (Figure 1). The control engine is responsible for data collection from the devices, enforcing access control policies, and enable interaction with client applications. Some IoT subnetworks allow direct client application access to individual devices. Alternatively, communication with an IoT device can be restricted to designated devices. Many IoT subnetworks are set up with assistance from cloud computing providers. For example, the AWS IoT Things Graph [24] provides cloud assistance for connecting various IoT subnetworks, enabling improved collaboration.

IoT Security Challenges. IoT subnetworks face several security challenges. First, IoT subnetworks rely on the security of individual devices and the software components that execute on these devices. These devices are often developed with severe cost pressures and time-to-market requirements that result is failure to follow known best security practices in their implementations. Attackers who can find vulnerable devices may be able to compromise the entire IoT subnetwork. Attackers can use the compromised subnetwork to inject false data into other devices, and thus compromise the overriding application. In the example IoT environment of Figure 1, an attacker-controlled neighboring subnetwork attempts to gain access to critical IoT infrastructure, the surveillance cameras, to inject false data and influence access control policies employed by the central IoT control engine. Second, IoT subnetworks can be large, comprising hundreds of nodes.

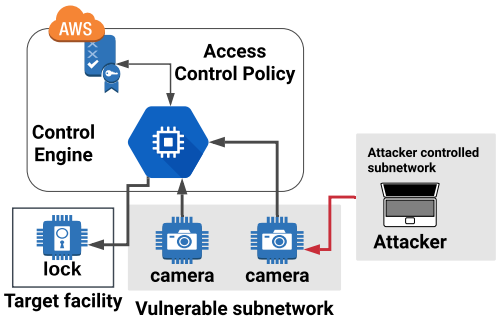


Fig. 1: A simplified IoT architecture with a cloud-assisted access control policy enforcement. An attacker subnetwork can exploit surveillance devices and cause undesired incidents.

With large interconnected subnetworks, maintaining security is challenging. A system of device interaction is required to control message exchanges across IoT devices. Third, IoT devices may occasionally be set as part of a fog computing [25] environment. IoT subnetworks that participate in a fog allow a portion of the computation to be carried on individual IoT devices. Heavier computations are offloaded on the cloud. This requires continuous interaction of various IoT subnetworks, effectively increasing exposure to remote attacks.

B. Related Work

Numerous works have considered various strategies for securing IoT subnetworks. The main approaches use machine learning models, blockchain-based access control, and program analysis.

Machine Learning. Many existing works employ machine learning techniques for IoT device identification. Device identification is useful and can aid in securing IoT subnetworks. However, there is much to be done in terms of detecting security breaches from active attackers that compromise selected vulnerable IoT devices.

Classifying network data is used in a model proposed by Meidan et al. [9]. In this model, devices are identified with the assistance of manually labeled data. Based on a trained classifier, devices are allowed access in an IoT subnetwork if the device type is white-listed. Nguyen et al. developed DÏoT [10], which uses Gated Recurrent Units (GRU) for analyzing network-level data for anomaly detection in IoT devices. The proposed model maps packets to symbols with probabilities calculated by the GRU. The probabilities below a threshold are marked as anomalies. In this work, we follow a similar approach by using probabilistic reasoning to detect anomalies. However, our anomaly detection model differs in that it uses application-level data and meta as opposed to network packets, it uses both classification and clustering to learn a model of *benign* behavior, and detects vulnerable devices by inspecting the signatures on application data. Note that IoT devices are required to supply their corresponding device proxies with signed checksums of the generated data.

Wang et al. [22] and Choi et al. [19] use formal graph-based approaches for detecting faults and security breaches in IoT devices. Wang et al. [22] use a form of a directed acyclic graph (DAG) to represent the interactions of various entities and events across an IoT platform. Their main concern is designing a code instrumentation mechanism to implement the DAG and collect provenance data to answer questions such as "How did a device turn on at a specific time?" Choi et al. [19] also use machine learning for detecting deviations in data transmissions across sensors and actuators. The data is grouped using correlation groups, where the groups are checked against real-time activities. The scope of event sequences in the attack resilient IoT architecture (Section III-C), however, is more general and includes security-sensitive events that are carried by devices. Serror et al. [26] use network filtering and device behavioral rules to detect anomalies in IoT devices. As specifying rules can be practically a limiting factor, we propose to use machine learning with minimal user intervention to harden an IoT subnetwork against remote attackers.

Blockchain. Some of the previous works proposed the use of the block-chain technology for developing access control in vast IoT networks [27]–[35]. Blockchain technologies use cryptographic methods for providing decentralized verification across a distrusting network of nodes. One way to secure IoT subnetworks is to use blockchain as an access control scheme [28], [31]–[33], [36]. An IoT network can maintain a cryptographically verifiable blockchain to control access to any group of nodes in the network. For example, the blockchain provides access control verification for nodes with long-term membership in the network (such as management nodes). In this work, we do not use the blockchain technology because of the increasing cost of verifying a chain. Instead, we use short chains of signatures to prove the provenance of IoT messages across a small IoT subnetwork.

Program Analysis. Other approaches use program and information flow analysis for ensuring IoT security. Soteria proposes to use static analysis and model checking to make sure if an IoT app enforces security policies correctly [37]–[41]. IoT-Guard [38], and similar systems such as [42] and [43], use program instrumentation in IoT applications to detect flawed applications and enforce appropriate security policies. Balliu et al. developed a semantics framework to reason about program interactions for IoT systems [44]. Program analysis can complement a data-oriented model by securing vulnerable software components that are used to operate IoT devices. While this approach is valuable, we focus on using data to analyze IoT device behavior. Thus, we assume that vulnerable software components are inevitable and can be exploited remotely.

III. Model and Architecture

This section starts by presenting the threat model (Section III-A), an overview of the architecture (Section III-B), and a detailed design description of the introduced components (Section III-C).

A. Threat Model

Several components are involved from both the defense (i.e., the target IoT subnetwork) and the attack fronts. The defense

front is an IoT subnetwork that includes a trusted computing base, the physical IoT devices and their installed software, and the network that connects the IoT devices. The IoT subnetwork serves an application that requires several IoT devices to cooperate and supply data. The system administrators (who own and manage the defense front) specify policy rules for the IoT subnetwork, which are applied according to data received from IoT devices. The attacking front includes software for remotely collecting reconnaissance data about the target IoT subnetwork.

The attacker can use the reconnaissance data to locate an available exploit and launch a remote attack. If the attacker compromises an IoT device and modifies messages sent from the device or redirects messages to an attacker-controlled device, the attack has succeeded. If the attack fails, the attacker attempts a new offense until the attack succeeds or the attacker's patience or resources are exhausted.

The trusted computing base consists of two components: (1) a controller, and (2) a network of device proxies. The controller is a system that is assumed to be secure against known attacks, and regularly monitored for patches and security updates. The controller includes software components and storage for collecting data from the IoT subnetwork. The controller has full access to each IoT device and must maintain connections with all IoT devices in the subnetwork. The controller can manage security policies for the IoT application. The controller also manages firewall rules across the network or for individual IoT devices. The controller can deploy and control device proxies. Device proxies are implemented as software components and are deployed within the network infrastructure of the IoT subnetwork.

B. Overview

IoT devices are exposed to vulnerabilities that enable remote attackers to gain unauthorized privileged access. Attackers can manipulate data generated on the compromised IoT devices and generate and send arbitrary messages to other devices. As discussed earlier, attackers are assumed to succeed in compromising IoT devices that are directly accessible through the Internet. Our approach is to (1) dynamically narrow the interface to each IoT device with a specification restricting access only to the trusted computing base, and (2) detect anomalous data exchanges between IoT devices by collecting and analyzing event data that passes through the network. In this work, messages refer to data (containing events captured by an IoT device), which is sent to another device in the network.

We introduce an attack-resilient IoT architecture to achieve the first goal. The architecture includes a controller component that installs a software-based *device proxy* for each physical IoT device. The device proxy does not reside on the IoT device itself. All device proxies are deployed on a coordinating device in the network. Each device proxy is set to interact with a single physical IoT device. The corresponding IoT device is also restricted to interact with the designated device proxy. A message m generated by a device P_i is sent to a device proxy V_j^i. If the message is a response to an external request (for example, from a client application), m is forwarded to the external requesting client. m is also forwarded to the controller component to be included as part of a device exchange dataset. For example, consider a camera P_i in a network N_k (written $P_i \in N_k$) that is connected to a device proxy $V_j^i \in N_k$ (forming a relation). P_i is set to only communicate with V_j^i if the request is outside N_k. This restriction is applied in the network firewall rules. All messages to and from P_i are relayed through and cached in V_j^i.

To achieve the second goal, we use an anomaly detection system based on classification and clustering of IoT application data. We collect and analyze events data from IoT devices. An event represents the data produced from a function on an IoT device. The functions of interest are the ones that serve the application for which the IoT subnetwork operates. For a surveillance application, an event can be a motion, a sound, or physical observations such as sensing smoke in the monitored environment. These events form the data that is used to analyze network traffic for anomalies. We are concerned with two problems. First, the system administrators should be able to distinguish fabricated deceitful messages generated by compromised IoT devices. Detecting fabricated data prevents the controller from making misguided decisions. Second, compromised IoT devices should be identified and considered for treatment or quarantine. Detecting and treating weak IoT devices assists in gradual healing and security improvement of the overall IoT subnetwork.

Analyzing events data can be done in two ways. System administrators can physically monitor the environment to collect data, label events as benign (in case of a certain event recorded by a benign camera) or malicious (in case of a fabricated event generated by a compromised camera). The collected labels can be used in a supervised learning algorithm to develop a model of the environment and detect deviations from the model. This method requires significant human intervention.

Our alternative is to use application data events as the basis for modeling device behavior, saving the need for manual labeling. During a training phase, application data are collected in the attack resilient IoT subnetwork. The data is used to train a binary classifier (Section IV-B). The output of the classifier captures the occurrence of events during daily time intervals. The normal behavior is then compared with real-time data generated from devices. Anomalous events are detected and used to limit devices that are suspected to be compromised. We also introduce a clustering approach to detect anomalies in events data. Our clustering approach employs a density-based clustering algorithm (DBSCAN [45]), which is designed to isolate data points that differ from the general trend. Our experiments on a real IoT testbed (Section VII) show that clustering application event frequencies can successfully isolate simulated anomalies with at least 70% success rate.

Using the results of the data analysis, the attack-resilient architecture can adjust policies on device proxies to limit or completely isolate compromised devices, enabling it to quickly recover from device compromises and mitigate the damage they can cause to other devices and the IoT application. While policy specification for IoT subnetworks is an important problem, we defer the discussion of precise policy specifications

based on anomaly detection results for future work.

C. Attack-Resilient IoT Architecture

In this section, we define an IoT subnetwork, the controller component of the attack resilient IoT architecture, the device proxies, and the attack resilient IoT subnetwork. The architecture is used in a data analysis model that is described later in Section IV-B.

IoT Subnetworks. An Internet of Things subnetwork is a digraph $G_k = (U_k, E_k)$ where k is the subnetwork's universal identifier, distinguishing it from other subnetworks, U_k is its set of nodes, E_k a set of edges (ordered pairs of distinct nodes). Two subnetworks G_k and G_l of G are neighbors in G if G has an edge $(u, v) \in (U_k \times U_l) \cup (U_l \times U_k)$. A node in a subnetwork represents an IoT device. Nodes are assumed to be at least connected to one subnetwork, through an Internet protocol. The set $U_k = V_k \cup W_k$ of nodes in a network may contain both proxy nodes V_k and physical nodes W_k. An IoT subnetwork serves a particular application and provides one or several interfaces for authenticated remote clients. Remote clients, including possible attackers, are also modeled as subnetworks.

The Controller. The controller (Figure 2) is a software module that is hosted on a machine in an isolated subnetwork. The controller is only accessible by the system administrators and accepts no incoming connections. The controller only accepts incoming network traffic when the connection is initiated by the controller itself. A small subnetwork G_c includes the controller node $c \in V_c$. The subnetwork G_c is a neighbor of all subnetworks in G that include at least one device proxy, having permanent bidirectional connections with all device controllers in all subnetworks. The controller can create or destroy device proxies, receives data from device proxies, and stores them in an isolated datastore $s \in V_c$ node within the subnetwork G_c. The controller can also query the third node $a \in V_c$ that is responsible for generating a model of events behavior according to the data in s. Bidirectional edges in E_c connect all the three nodes c, s, and a.

The controller receives analysis results from a, consults a policy file, and applies the appropriate policy. The policies are rules that affect device proxies. A policy can be either

1) terminate (or launch) device proxy x,
2) connect (or disconnect) device proxy x to (or from) IoT device d,
3) enable (or disable) remote traffic from IoT device d connected to device proxy x, or
4) limit remote access to an IoT device d connected to proxy x to a set of remote clients.

The policy file is a table of conditional statements that indicate the application of policies according to the results of data analysis in a. Because the controller depends on the analysis of an anomaly detection method (described in Section IV-B), false positives can occur. Hence, the controller should only limit access to a suspicious IoT device. The limitation can include disconnecting some of the remote clients. The policy is applied through the device proxy. When the controller has high confidence about the suspicious device being compromised, a more restrictive policy can be applied. For example, the corresponding device proxy is terminated. In this case, the device can no longer interact with any other device in the subnetwork.

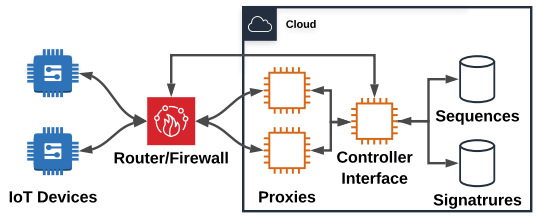


Fig. 2: Implementation of the attack resilient IoT architecture using AWS EC2 instances and a modified Linux machine as the firewall router.

Device Proxies. A device proxy is a software module that is installed on a virtual machine within an IoT subnetwork G_k. The purpose of using device proxies is to isolate physical IoT devices by narrowing their interfaces with remote clients. Also, device proxies ensure that events data, as generated on IoT devices, are properly reported to the controller. Each device proxy is represented as a node in U_k. Each device proxy is launched on a separate virtual machine hosted on a physical machine (such as a router). The controller sets each device proxy u to connect to a device $x \in U_l$ via an edge (u, x), making G_l and G_k neighbors. Note that to isolate a device proxy from network scans and local area network attacks (such as ARP poisoning attacks [46]), device proxies and their corresponding physical devices are in separate IoT subnetworks.

Attack-Resilient IoT Subnetwork. An *attack-resilient* IoT subnetwork $\mathcal{A}$ is an IoT subnetwork that (1) has device proxies connected to every physical IoT device, that is, $|V_\mathcal{A}| = |W_\mathcal{A}|$, and $\forall x \in W_\mathcal{A}$, $\exists u \in V_\mathcal{A} : (u, x) \in E_\mathcal{A}$, and (2) contains an incomplete bipartite subgraph between the two node subsets $V_\mathcal{A}$ and $W_\mathcal{A}$, with edges $E^*_\mathcal{A} \subset E_\mathcal{A}$ that only connect $V_\mathcal{A}$ and $W_\mathcal{A}$. The attack resilient IoT subnetwork transmits normal device messages, miscellaneous network protocol messages, and specially signed messages that carry event data headed towards the controller node. The network structure is dynamic and is managed by the controller node. Nodes and edges are added and removed according to the policy file maintained by the controller node (described above).

The data analyzed from individual devices (Section IV-B) are used by the attack-resilient IoT subnetwork as the bases of a defense strategy. The defense uses the model developed from benign sensor activities to detect anomalies in the sensor data. When an event sequence S (defined precisely below) arrives, the controller compares S with the normal data. If S is detected to be an anomaly, the controller increments a counter for the physical device $d^\star$ from which S originated. The counter is used to decide if the physical device $d^\star$ should be restricted. When the controller decides that $d^\star$ may have been compromised, one of the defense policies (described earlier) can be applied. This work focuses on providing the

architecture and the data analysis models and defers the discussion of policy design to future work.

IV. Analyzing IoT Device Data

In this section, we describe the data needed to recognize misbehavior of IoT devices (Section IV-A). This data comprises event sequences that include the events captured by IoT devices in a time interval. After introducing events, we present a model for analyzing event data using a binary classifier and a clustering approach (Section IV-B) as highlighted in Figure 3. A Support Vector Machine (SVM) classifier and a density-based clustering algorithm are used to detect deceitful sequences generated by devices.

A. Event Sequences

We are concerned with fabricated messages that are generated on compromised IoT devices. The fabricated messages influence the decision by the access control system that applies security policies to smart IoT devices (for example, locking or unlocking smart door locks). The core problem is that the access control system is unable to distinguish the fabricated messages. Our approach is to enable provenance verification of messages across IoT subnetworks by tracking message origins as they are passed through each individual IoT device or device proxy. For tracking messages, we define events and event sequences and present a model for monitoring sequences of events throughout the network.

Definition of an event. An event a_i has the form $(\mathcal{M}, d, \sigma, \alpha)$, where i is the sequence index, $\mathcal{M}$ is a list of metadata attributes for the event, d is the identifier of the device that generated the event, σ is the operating system service associated with the event, and α is the data supplied with the event.

Formally, an *event sequence* is a tuple $S = (A, \delta, o)$ where $A = (a_1, a_2, \ldots, a_n)$ is a sequence of events in order of time, δ is the time interval for which the sequence was created, and o is the device identifier representing the original creator of the sequence. For any sequence $S \in \mathcal{S} = \{$all event sequences in an attack resilient IoT subnetwork $\mathcal{A}\}$, a unique combination of (δ, o) identifies the sequence. That is, no two sequences are created at the same time on the same device.

Recording events. The rationale for using a time-based sequence model is the assumption that several correlated events occur in temporal proximity. Events at various intervals can also be correlated. Accordingly, devices record events in sequences that are only valid during δ. All physical devices and device proxies record, transmit and receive event sequences across the IoT network. Each device either creates a new sequence for each system service or receives a sequence from another neighboring device. The device sets a timer for each new sequence. When the timer expires, the device creates a new sequence and waits for events (file creations) to occur. In this model, files created for several consequent physical movements can be recorded in a single sequence during the lifetime of the sequence.

The relationship among events in separate sequences is established when analyzing the sequences (Section IV-B). Note that the duration of each time interval δ is platform-dependent and can be arbitrary. The value of δ can significantly affect the analysis of events as evaluated in Section VII.

Authenticity of events. An attacker can generate and transmit sequences with arbitrary origins from compromised devices. To prevent attackers from faking events, each device must sign event sequences transmitted to other devices. Assume that each device has a pair of public and secret keys (PK, SK) such that the public key PK and the device identifier are registered with the cloud controller. To prove authenticity of an event sequence S, the device sending the sequence must sign the message containing the event sequence. The sending device can append S with a message authentication code (MAC). Using a fast one-way hash function $H(x)$, the device appends S with $y = H(S)$ and signs y using its secret key: $e = \mathsf{Enc}(y)_{SK}$. The encrypted checksum, the signature e, is the message authentication code for S. A device proxy receives the signed message from the corresponding IoT device and forwards it to the controller. Note that the attack resilient IoT architecture may destroy a device proxy and replace it with a different proxy. Further, an attacker may send messages from a compromised device to another IoT device and convince the recipient device to send the message through its proxy. Preventing this attack requires authenticating the proxy that forwards the message. Thus, when a recipient device receives S, the device appends a new subsequence of events to S_1, giving a new (3-tuple) event sequence S^*. To authenticate the additional events in S^*, a new MAC is appended to S^* which includes the previous signature e and the checksum of the newly added subsequence: $e^* = \mathsf{Enc}(e \| H(S^*))_{SK^*}$ (where SK^* is the secret key for the recipient of S).

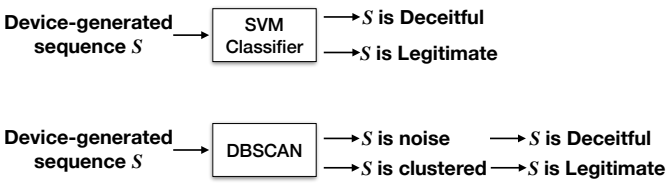


Fig. 3: Overview of the two approaches used to detect deceitful device-generated sequences. SVM classifier is the Support Vector Machine classifier and DBSCAN is the density-based clustering algorithm.

B. Event Anomalies

Our work is distinguished from previous work since we use application data to detect and analyze anomalies (Section III-B). In contrast with network traffic data, application data is specific and is challenging to generalize. Our learning model is based on the observation that (1) IoT application data is temporal as captured by event sequences, and (2) attackers generate fabricated events at particular times. Thus application data anomalies can be captured by isolating fabricated sequences either based on the probability of occurrence or with respect to the relationship with benign data points. Here,

we present a general definition of anomaly followed by two models to distinguish benign and anomalous behaviors.

Anomaly Model. Let $f(S_i) \in \mathbb{R}$ be a characterization function that receives a sequence S_i. The sequence S_i is an anomaly if

$$f(S_i) \leq \tau,$$

where τ is a generic threshold parameter. The function f is generically defined to accommodate the two anomaly detection techniques that we use for our architecture. The rationale for using two models is to compare the effectiveness of each one in increasing the performance of the anomaly detection process as described next.

Classification Approach. First, we use a binary classifier (similar to [47]) to decide if a sequence should occur during fixed time intervals. Fixed time intervals are specified for each sequence of events in the attack resilient IoT architecture. The classifier's input has two features: an event sequence and a time interval. The output of the classifier is 1 if the dataset supports that the event sequence is expected to occur during the given time interval, and 0 otherwise. A rigid anomaly detection scheme would directly use the output of the classifier to decide if an event is anomalous by setting $\tau = 0$. Instead, we use a flexible anomaly detection scheme by using the posterior probabilities of the classifier's output [48]. We train the classifier with a dataset $\mathcal{D}$ of benign event sequences. The training results in a model $M(\mathcal{D})$. In real time, the sequences generated from IoT devices are labeled using the learned model. To detect if a sequence is an anomaly, we set the value of the function f to the posterior probability obtained from the classifier:

$$f(S_i) = P(y = a|S_i),$$

where y is the output of the classifier based on the model $M(\mathcal{D})$, and $0 \leq a \leq 1$. The choice of a depends on the expectation that the event occurs. Here, a low probability that S_i is expected to occur during a specific time interval determines a low value of f. The sequence is labeled as anomaly when f falls below τ.

Clustering Approach. We also cluster the observed sequences using a density-based clustering approach [45]. The reason for using density-based clustering is to isolate random behavior by attackers in generating event sequences. In a density-based clustering method, noise is defined as data points that do not belong to any clusters. We use the definition of noise in density-based clustering to detect anomalies. Note that we do not base the anomaly detection decision on data attributes from within the IoT devices. That is because such attributes can be fabricated on compromised devices. We also require that IoT devices be memory-less. In the attack-resilient architecture, the IoT device communication history is only cached on device proxies. Thus, attackers cannot learn the previous communication patterns. As a result, the number of events and the time interval of sequences are arbitrary and may appear as noise.

To form the clustering problem, a quantitative value for an event sequence is defined. Recall from Section IV-A that for an event sequence S_i, $|A|$ is the number of events recorded from device $d^\star$ during the interval δ. Let x_i be the pair $(d^\star, |A|/m(\delta))$, where $m(\delta)$ is the length of the time interval δ. Cluster first on $d^\star$ (sort by physical device), then on the event frequencies $|A|/m(\delta)$ (hierarchical clustering).

The input $X = \{x_1, x_2, \ldots, x_n\}$ to the clustering algorithm is a set of event sequence pairs for a single IoT subnetwork application. Assume X is collected during a time interval with a negligible probability of attack. The event sequences are generated on multiple devices and a diverse set of time intervals. When the input X is clustered, most sequences are expected to join a cluster. The remaining noise, which is assumed to contain anomalies, is not clustered. We modify the density-based clustering to include additional auxiliary single-member clusters for each unclustered point. Let C denote the cluster set containing the sequence S_i pair x_i, computed using a density-based clustering algorithm. Then the value of the characterization function $f(S_i)$ is computed as the cardinality of the cluster set C relative to the total number of data points (pairs):

$$f(S_i) = \frac{|C|}{n}.$$

A low value of $f(S_i)$ either indicates that x_i is noise or x_i has joined a sparse cluster. As anomalies are assumed to be the minority events, the function f is expected to detect them.

Handling Cluster Noise. A challenge in detecting the anomalies using the clustering approach is with the noise in the training dataset of the normal behavior. To detect anomalous behavior as noise, the already existing noise in the normal behavior can result in a high rate of false positives. To remedy this problem, the noise from the normal behavior is separated from the clustered behavior. In real time, if an event sequence is close to a cluster, the sequence is marked as normal. Otherwise, if the sequence is noise, the controller attempts to find a near neighbor in the noisy normal data. If a *close enough* normal noise data point is detected, the sequence is marked as normal. Otherwise, the sequence is an anomaly.

V. Security Analysis

In this section, the security of the proposed architecture and data analysis model is presented. The mitigated attacks and security limitations are discussed.

A. Mitigated Attacks

Remote attackers are the primary concern of this work. A remote exploitation attack vector exists because of the usability of the target IoT subnetwork. Enabling remote clients requires allowing remote traffic to reach the IoT devices in the target subnetwork. Thus, the legitimate user can connect directly to an IoT device from any IP address using a client application. Our model does not restrict the access of remote clients. Instead, the network traffic to each IoT device is restricted to a device proxy that is created in a neighboring subnetwork (as virtual machines).

Although the traffic is restricted, remote attackers can gain access to a vulnerable IoT device. The remote access can be through a victim application that accepts attacker manipulations, for example, through payloads. A more powerful remote

access could be through gaining escalated privileges, allowing for a stable connection with the victim device.

The data analysis model is used to find IoT devices that are compromised through remote exploitation attacks. The data model relies on events that are generated on each IoT device. The attack resilient architecture requires that an event sequence S be signed by all devices that transmit S. The remote attacker should continue signing events even though they might be fabricated. Otherwise, the controller does not accept an unsigned S or one with an unknown key. This signing protocol allows the resilient architecture to check the origin of sequences.

Stealth Attacker. The remote exploitation attacker can remain in stealth mode, collect data from the compromised IoT device, and imitate the victim to attack the classifier or the clustering algorithms that are used to predict device behavior. The objective of this attack imitating the victim is to supply the controller with fake event contents (for example, replacing the image of an event on a camera with a fake one). This type of attack can occur in two ways. One is to use the victim device itself as the storage for event behavior. The other way is to record each normal event sequence that is generated by the victim and transmit it back to a more powerful device that is controlled by the attacker.

The use of device proxies allows us to mitigate the first type by limiting the storage of the actual IoT device to the minimum, essentially disabling the accumulation of data on the device, and using device proxies as a cache for the data, in the case of generating many events within a short time interval. This approach does not eliminate the data collection on the device completely. However, it will severely limit the capability of the attacker to build a complete image of the victim's behavior.

The second type of attack is eliminated by the data analysis approach. The advantage of the controller on the attacker is the existence of an abundance of data from all connected IoT devices. This data is collected in a controlled environment, for example, by disabling remote access completely. The data, as demonstrated in the experiments of Section VII, enables the anomaly detection to detect if a device is suspicious or malicious quickly. In this case, the controller instructs the device proxy to limit remote access to the device (for example, disabling outbound traffic). To overcome this mitigation, the attacker may avoid interfering with the activity of the device, allowing it to behave normally. The attacker will only send a copy of the collected events back to an attacker-controlled machine. This attack can be mitigated by learning the legitimate client's behavior or IP address and comparing the real-time connection's behavior to the learned behavior. Also, strict policies can mitigate the attack, for example, by limiting connection to a single client application. Further investigation of a complete solution to this problem is left for future work.

B. Limitations

The resilient architecture depends on the trustworthiness of device proxies, and the controller. Attackers with full knowledge of the target environment may simulate the behavior of IoT devices with a predictive model without transmitting anomalous data from a compromised device. The model presented in this work does not provide an immediate solution to such an attack.

One important challenge in using data input from multiple devices is establishing trust amongst the devices. We assume this can be done before the system is exposed, and that devices can establish secure key pairs with the controller. This may not be possible in typical deployments. One approach to establishing trust is to use blockchain technologies, as proposed by Fortino et al., to maintain a record that certifies the reputation capital of each device [49]. This use of blockchain technology can further aid the process of collecting and analyzing reliable data from cooperating devices.

We assume that the model of benign behavior is developed during a period that can be guaranteed to be free of attacks, but otherwise closely resembles normal use. This may be difficult to achieve in practice. Our anomaly detection scheme may have a decreased success rate when the benign behavior of IoT devices differs substantially from that of the training period. This requires retraining the anomaly detection system to represent the new behavior. Detecting the change in the benign behavior can be challenging and requires careful consideration since it opens the design to poisoning attacks where an attacker can inject data that corrupts the trained model.

Our model could also be extended using the concept of causal reasoning [50] to further enhance the detection strength by the controller. In this scenario, an attacker with full knowledge of a subset of the devices, potentially with physical access to them, can be detected by using data from other cooperating devices. For example, a causal relationship graph can be constructed to represent various conclusions using the sequences of events generated by our architecture. Once a pattern of causal relations is learned, the attacker with a full mimicking capability on a device A can still be detected, when a cooperating trustworthy device B generates an event subsequence that contradicts the one generated at A.

VI. Implementation

This section provides the implementation details of the attack-resilient IoT architecture. The prototype uses Raspberry Pi 4 as the hardware module for IoT devices. The required software modifications are minimal and require no changes to the operating system kernel or the network software stack. Apache web server, MySQL database, and AWS Elastic Cloud Computing (EC2) are used to develop the required software components. The software components include a sequence manager for the IoT device, the controller, which is implemented on the Amazon Web Services cloud, and a proxy service that is installed on each device proxy. The architecture also includes a signature store and a data store. The signature store is a relational database table of device IP, device MAC address, and the corresponding binary representation of the public key for the device. The data store is a relational database table with a table entry for each sequence generated.

The sequence manager receives commands from the device proxy, monitors a set of services on the IoT device, records

sequences, and signs each sequence and sends it to the device proxy. The sequence manager is a daemon process that automatically launches when the IoT device boots. The threat model does not require trusting the sequence manager. When the attacker tampers with a victim device, the attacker either stops the process and prevents it from transmitting data or leaves the sequence manager to normally transmit fabricated sequences.

The firewall rules are set and managed by the controller. A basic set of firewall rules are applied on the network router that connects the physical IoT devices to the Internet. The router restricts each IoT physical device to only communicate with the corresponding proxy by setting the inbound and outbound rules. The firewall rules enforce restrictions on the MAC address of the IoT device. The attacker can forge the MAC address of a victim IoT device. Forging MAC addresses is not useful since it prevents the victim from communicating to the device proxy. The only benefit is in causing a denial of service, which is not the concern of this work. The router is implemented as a regular Linux machine. The firewall rules are applied as `iptables` rules. These rules are updated directly by the controller.

Although firewall rules limit interactions with the IoT devices, remote attackers can still find ways to exploit a target IoT device. Suppose that a target IoT device u_1 is in a subnetwork G_1. A neighboring subnetwork G_2 has regular machines that are accessible through the Internet with an open traffic rule. If a machine u_2 in G_2 is compromised by the attacker, then the attacker might access the subnetwork G_1. u_1 is behind the firewall, but the attacker might find vulnerabilities in the router and gain access to u_1. The attacker might also gain access to u_1 by sending malicious requests that pass through the corresponding device proxy. The assumption here is that device proxies are trusted, but we do not trust the requests that are processed through device proxies.

The IoT devices send sequences through the router to device proxies (Steps 1 and 2 in Figure 4). Each device proxy runs a web server. Upon receiving a sequence as a request, the web server modifies the request to include a signature and forwards the request to the controller (Step 3). The proxy just signs this message with its own key, and forwards it to the controller, which verifies both signatures (Step 4). The signatures are generated using Python's cryptography library [51].

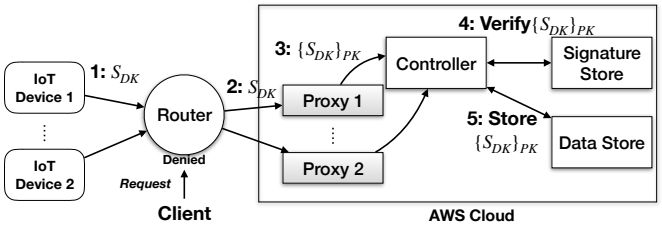


Fig. 4: An implementation of the resilient architecture. The implementation uses Elastic Computing Cloud (EC2) machines in AWS. Rectangles show EC2 machines and shaded shapes are trusted components. Each IoT device communicates through a distinct proxy machine. Requests are only routed to and from the designated proxy machine. S_{DK} is a sequence signed by the device key DK. $\{S_{DK}\}_{PK}$ is a sequence, which is signed by the proxy.

The controller has a web server interface that interacts with routers and device proxies. Device proxies send sequences asynchronously to the controller. The controller receives each request and sends the request to a temporary database table for verifying the signature. A thread pool checks the signatures on the sequences and sends the verified signatures to the signature store database. A MySQL database is used to handle the store requests. The sequences are also recorded (Step 5) to train the machine learning models.

Device proxies are created by the controller. Each device proxy is a virtual machine with a predefined machine image that has a copy of the web-based proxy as described above. For implementation simplicity, AWS EC2 is used to create virtual machines for device proxies. To reduce network round trip time, virtual machines can be created internally on a VMware ESX server.

VII. Evaluation of Anomaly Detection

We evaluate the two anomaly detection methods using benign data generated in an IoT testbed. The attacker data is simulated based on two attack strategies that have different views of the target environment. The classifier is trained on the benign training set and is evaluated using the benign test set and the attacker test set. With the benign test set, the classifier achieves high precision and recall. With the attacker test set, the classifier has low recall in many cases (below 0.5). Depending on the behavior of the attacker and the length of the interval in which sequences are generated, the classifier produces different precision and recall values. The clustering method is evaluated using the percentage of noise generated when clustering attack data points. The clustering algorithm has an average success rate of detecting 76% of attacker-generated messages. In the worst case, the clustering method marks 70% of attacker points as noise, thereby detecting them as anomalies. In the best case, the controller marks 91% of attacker points as noise. The presented models may suit different application and data settings and the choice of parameters affects the final results. In practical settings, system administrators can adjust the system parameters to produce the best results.

A. Data Collection

The experimental data was collected from a testbed in two physically separate locations: a home office and a research lab. For the home office, three sensors were installed: an office room, the server and storage room, and the corridor. A single sensor was installed in the research lab. The sensors capture pictures of any moving objects. The hardware for sensors is Raspberry Pi 4 machines with an internal camera module attached to the board. Each machine has four cores and 4 GB of memory and a 32 GB of SD storage card. Each sensor runs an Ubuntu distribution and the `motion`[1] open-source library for detecting motion using the camera.

[1] https://motion-project.github.io/

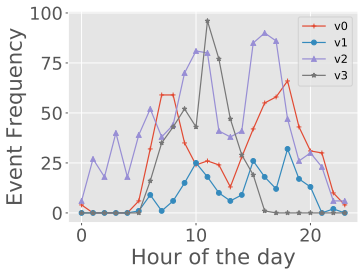


Fig. 5: Hourly frequency of motions for each sensor.

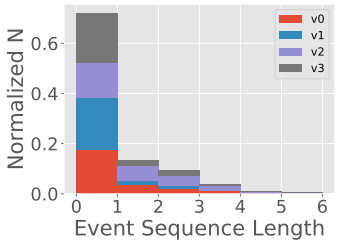


Fig. 6: Frequency histogram for each sensor with unified four-minute intervals ($m(\delta) = 4$). Each interval corresponds to an event sequence. The y-axis shows the normalized frequency of sequences. N is the normalized frequency.

Sensors are labeled v_i for $0 \leq i \leq 3$ in the order mentioned previously. The data was collected 24 hours a day for about 30 days, generating approximately 2500 data points. Some of the sensors were installed during the entire data collection period and some with shorter periods of time. Because of the experimental nature of the testbed and the changes needed in the location of the sensors, noisy motions due to changing the location were detected. Also, some motions were not captured by the sensors because they were occasionally turned off.

The activity generated on the sensors corresponds to less than ten people. The sensor's view frame was facing the entrance in all locations but with arbitrary calibration. Further, the corridor sensor was not able to detect far away motion (its range is limited to about 15 meters). However, motion of closer objects were instantly recorded on the sensor's internal storage with the file metadata capturing the exact time of the motion. The sensors also sometimes mistakenly interpreted changes in the lighting as a movement (for example, detecting the natural light appearing from the window).

The collected data serves as the training data for modeling normal (benign) behavior in the monitored environment. Figure 5 shows the frequency of events within one-hour intervals for each sensor. The figure shows a clear pattern of activity according to the time of day. Figure 6 shows the frequency histogram of the event sequences with length ≤ 5 in four-minute intervals. The data for Figure 6 provides a key insight that underlies the application data analysis in this work. The assumption is that the attacker misses the expected size of event sequence lengths, and the event sequences provided by the attacker are clustered as noise.

B. Design of the Experiments

The collected data is assumed to be benign with no malicious activity. No device was compromised during the experiments, and all the users having access to the cameras were trustworthy. Thus, the data is used as a basis for the category of benign activities. For experimenting with fabricated messages, we present two attacker strategies. Each strategy uses a model to generate fabricated messages.

Naïve Random Attacker. The random attacker is the simplest attacker that is assumed to have a minimal view of the target IoT subnetwork. The attacker knows the IP addresses of a subset of the sensors and the purpose of the application (e.g., surveillance). The attacker does not have prior knowledge of the normal behavior of the sensors. The attacker knows the type of the physical environment in which the IoT subnetwork is installed (e.g., a home office). The attacker's goal is to pollute the data collected from sensors and influence the decisions made by a high privilege IoT subnetwork that interacts with the subnetwork of sensors. The attacker can gain remote access to a target physical device by using a vulnerable service on the target. The attacker can disable reporting a typical event sequence, modify a normal event sequence to include less or more events, or generate fabricated event sequences when there are no actual events captured by the compromised sensor.

The simulation of a random attacker tests the attacker techniques mentioned earlier. The simulation starts assuming a trained classifier and a clustered dataset with a large dataset of the benign behavior. The simulation measures the attacker's success by generating attacking sequences during 24 hours. The time at which event sequences are generated is chosen randomly. Also, the number of events reported in a single sequence is chosen randomly.

Mimicry Attacker. The mimicry attacker can leverage reconnaissance data in an attempt to mimic the behavior of benign nodes. This attacker uses an accurate estimation of file size and a better estimation of sequence length. In particular, we use statistics collected from the dataset of each device to simulate the estimated average file size associated with each event sequence. We also limit the random number generator for the attacker to the maximum and minimum sequence length from the actual dataset. The assumption here is that the attacker is able to collect more reconnaissance data, for example, by spending more time in stealth mode on the compromised device, but is not able to build its own accurate model of the subnetwork so sends random data within the observed parameters.

C. Classification Results

The results are displayed for each IoT device in the testbed (Section VII-A). The three home office devices are referred to as v_0 for the office room, v_1 for the server and storage room, and v_2 for the corridor. The lab device is referred to as v_3.

The results of the experiments show the performance of the anomaly detection methodology using various parameters.

We use precision (positive predictive value) P and recall (sensitivity) R to measure the accuracy of the classifier:

$$P = \frac{T_p}{T_p + F_p}, \quad R = \frac{T_p}{T_p + F_n},$$

where T/F denote true/false and p/n denote positive/negative. We first test the relevant features that could be used to capture attacker behavior accurately. We use a support vector machine (SVM) classifier to classify the input with the output values $Y = 1$ indicating an event was reported and $Y = 0$ indicating no event was reported. The input includes the hour of an event sequence, the average time difference between any two events in the sequence, and the average size of the event files in the sequence.

We measure the performance of both the attacker and the training dataset using the trained classifier. A shuffled 30% of the training dataset is used as a test set. The attacker-generated dataset is also used as a test dataset for the classifier. The attacker is successful when both precision and recall values are high. That is, if precision and recall are high, the attacker can convince the controller that the compromised device is behaving normally. The attacker's benefit is in manipulating the contents of the event towards malicious intents. Detecting manipulations of the event contents are out of the scope of this work.

Feature Selection. Tables I and II summarize the results of the classifier's performance using the training data compared with data generated by the naïve random attacker. In Table I, the results are reported with the input to SVM including the file size averages. Here, both the precision and recall values are high enough to provide encouragement that the proposed defense could be made to work in practice. In Table II, the file size averages were excluded from the classifier for both the training and attacker data. Here, the attacker fails with devices v_1 and v_3 but has high precision with the other two while keeping a low recall. This indicates that if file sizes are not used to train the classifier, the attacker suffers from a high false negative rate. Based on these results, we proceed with only training the metadata of events.

Detection Accuracy. Figure 8 shows the performance of the classifier using the training and the mimicry attacker. Recall that the mimicry attacker only speculates about the number of events reported within a sequence. The mimicry attacker has an accurate estimate of the minimum and the maximum number of events reported within a sequence and the maximum and minimum values for the average distance between two events. These estimates are supplied directly from the training dataset. The dataset for the mimicry attacker might be challenging to obtain. However, simulating this behavior aims to stress test the classification approach by giving considerable advantage to the attacker.

The results show the computed precision and recall for each device for both the training dataset and the attacker-generated dataset using various event sequence interval lengths. The bars show precision and recall for both the naïve and the mimicry datasets in each figure. Note that the classifier has a high true positive and low false positive for the training datasets of all devices. The attacker generated data has the best classifier performance in v_0 and v_2. Even with the achieved performance, the attacker data classifier performance still has a relatively low recall value. In this simulation, we also tested the accuracy of the classifier by modifying the intervals during which the sequences are captured. The x-axis shows the length of sequence intervals in minutes. Longer sequences show better attacker performance by achieving higher recall values. This is a natural consequence as longer sequences are likely to capture the output value of $Y = 1$.

M	Training		Attack	
	Precision	Recall	Precision	Recall
v_0	0.904	0.89	0.911	0.954
v_1	0.948	0.945	0.909	0.953
v_2	0.817	0.797	0.898	0.948
v_3	0.927	0.921	0.918	0.958

TABLE I: Results of attacker data and training data classifier accuracy including file size estimation. Each v_i refers to an IoT device in the testbed (defined earlier)

M	Training		Attack	
	Precision	Recall	Precision	Recall
v_0	0.883	0.877	0.854	0.048
v_1	0.938	0.934	0.002	0.047
v_2	0.788	0.771	0.898	0.104
v_3	0.927	0.921	0.003	0.055

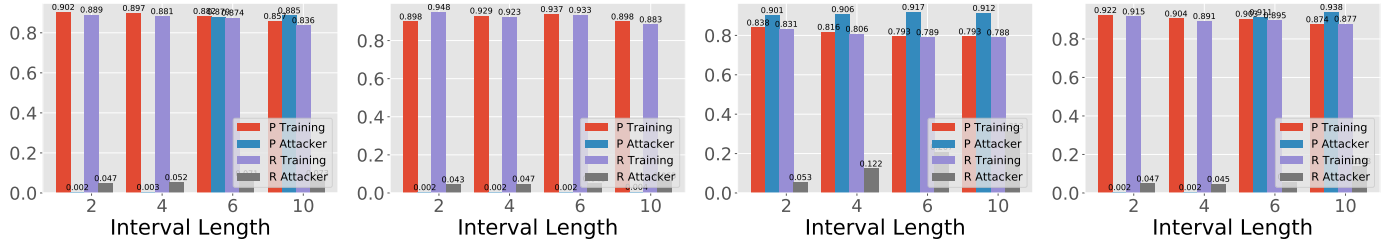
TABLE II: Results of attacker data and training data classifier accuracy excluding file size estimation.

The naïve random attacker (with no realistic estimation of the input data) has a lower success rate compared to the mimicry attacker (Figure 7). For example, in the dataset of the first IoT device, the mimicry attacker has a considerably high precision and recall. In contrast the naïve random attacker has low precision and recall. This indicates that the naïve random attacker performs poorly in predicting the benign behavior of the compromised IoT device.

D. Clustering Results

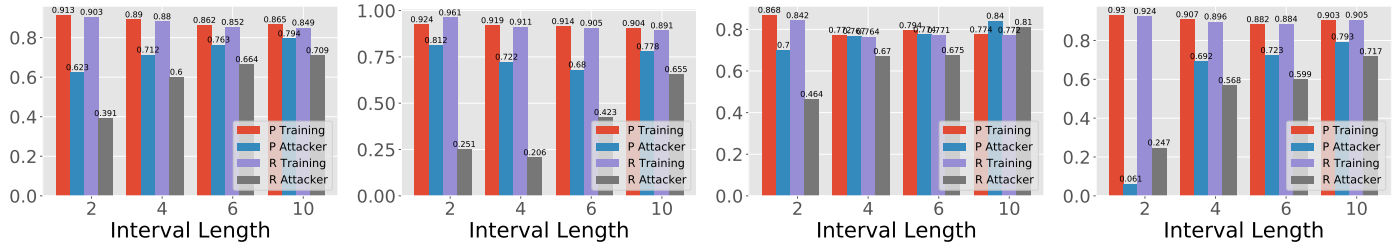
The clustering-based anomaly detection uses the density-based clustering algorithm DBSCAN [45]. This clustering algorithm attempts to find core samples of high density and clusters other data points accordingly. The algorithm also finds *noise* data points that do not belong to any cluster. Two parameters influence the results of the algorithm: the radius ϵ used to cluster points, and the minimum number m of samples within the radius ϵ to form a cluster. In our experiments, Euclidean distance was used to measure the distances.

We first tabulate the ratio of data points found to be noise by DBSCAN in Table III. In this table, for each interval length of $m(\delta)$, the dataset is clustered and the number of noise data points divided by the number of clustered points is computed. Ideally, the clustering algorithm should return no noise. The chosen parameters of DBSCAN for the experiments are $\epsilon = 0.5$ and $m = 3$. The results demonstrate the false positive rates. These results are subject to change based on different parameter selections and different datasets.



(a) Office (v0) (b) Server Room (v1) (c) Corridor (v2) (d) Lab (v3)

Fig. 7: Performance of the classifier using the training dataset compared to naïve random attacker-generated sequences. P stands for precision, and R stands for recall. The y-axis shows the value of the metric.



(a) Office (v0) (b) Server Room (v1) (c) Corridor (v2) (d) Lab (v3)

Fig. 8: Performance of the classifier using the training dataset compared to mimicry attacker-generated sequences. P stands for precision, and R stands for recall. The y-axis shows the value of the metric.

Interval Length	Machine			
	v_0	v_1	v_2	v_3
2	0.104	0.036	0.136	0.058
4	0.174	0.067	0.216	0.098
6	0.208	0.094	0.273	0.118
10	0.282	0.122	0.356	0.157

TABLE III: Result of density-based clustering (capturing false positives) with varying interval lengths. The values show the ratio of number of noise points to number of clustered points in the dataset. The ideal ratio is 0.0.

To measure the performance of anomaly detection on the collected dataset, we report the fraction of attacker sequences that are detected as anomalies. In the simulations, the threshold $T = 1/n$ (size of a single-point cluster over the total number of data points) is used, which only considers the noise as anomalies. Algorithm 1 is used to simulate the naive attacker behavior. Let X be a matrix of inputs (implemented as a list) to the clustering algorithm with each row having two columns: the interval of the sequence, and the number of events in the sequence. Let I be the set of all intervals, and f_l and f_h be the minimum and maximum attacker estimations of the number of events in all sequences. Let `random`$(a,\ b)$ be a function that generates a random data point object with a value in $[a, b]$. The function `DBSCAN`(X) generates a list of labels L by executing the DBSCAN clustering algorithm on the input set X. `noise`(p) marks the attacker data point p as a noise point. A label with a negative value marks a noise point.

Algorithm 1 essentially reclusters each attacker-generated data point with the rest of the training dataset. The ratios

Algorithm 1 Compute the performance of clustering-based anomaly detection.

Require: X, I, f_l, f_h
1: **for** $i \in I$ **do**
2: $p \leftarrow$ `random`$(f_l,\ f_h)$
3: `push`$(X, (i, p))$
4: $L \leftarrow$ `DBSCAN`(X)
5: $l \leftarrow$ `pop`(L)
6: **if** $l < 0$ **then**
7: `noise`(p)
8: `push`$(P,\ p)$
9: **end if**
10: `pop`(X)
11: **end for**
12: **return** P

of the number of noise points to the number of data points generated by the attacker are demonstrated in Table IV. Higher values indicate better anomaly detection performance. Note that the minimum achieved noise ratio is 0.7; that is, 70% of the attacker generated data points are marked as noise points. The performance of the mimicry attacker is slightly better than that of the naive random attacker for all devices in all intervals.

Comparison of Accuracy with Existing Works. We compare the detection performance of our model with 6thsense [20], a context-aware IoT intrusion detection system. The objective of 6thsense is to analyze user behavior and detect intrusions with the aid of an IoT network. 6thsense uses sensor-level detection as opposed to application-level detection. It finds

Interval Length	Machine (Mimicry Random)				Machine (Naive Random)			
	v_0	v_1	v_2	v_3	v_0	v_1	v_2	v_3
2	0.701	0.739	0.73	0.711	0.892	0.908	0.882	0.898
4	0.772	0.788	0.749	0.75	0.879	0.912	0.835	0.885
6	0.787	0.745	0.723	0.749	0.851	0.894	0.827	0.871
10	0.819	0.825	0.792	0.812	0.842	0.884	0.803	0.857

TABLE IV: Result of using density-based clusters to detect anomalies from the naive random and mimicry random attacker-generated data. The results show the fraction of attacker sequences that are marked as anomalies.

Interval length (hours)	Attack Data Points	Verification Time (s)	Time Per Sequence (ms)
2	5040	112.30	22
4	2520	27.76	11
6	1680	11.98	7
10	1166	4.56	4

TABLE V: Time to cluster attack points over the entire data sets with various interval lengths.

sensor activity that is malicious as a result of executing malware. We use the F-score to compare the performance of our model to that of 6thsense:

$$F = 2 \times \frac{(P \times R)}{(P + R)}.$$

The value of F ranges from 0 to 1, with $F = 1$ indicating the best value. Our model achieves an F-score of 0.74–0.94 (on average, 0.87) for the random attacker, while achieving 0.76–0.94 (on average, 0.86) for the mimicry attacker. This compares favorably with 6thsense, which achieves (depending on the model parameters) a F-score of 0.4615–0.9899 (on average, 0.8040) for their Markov Chain based model. For their Naive Bayes model, the F-score range is 0.7615–0.8235 (on average, 0.7859). The performance of our models can potentially be improved by using a feature-rich dataset from the underlying IoT sensors. Coupled with the application-level dataset, the accuracy of the models would likely be improved.

We compared our results previous works [9], [11], [21] using the metrics from Barrera et al. [21], true positive rate (TPR) and false positive rate (FPR). TPR is identical to the recall metric used in our work. A maximum reported FPR of 0.1% produces a TPR of 94.01%. Unfortunately, the reported data does not enable the calculation of precision, and consequently the F-score. However, comparing the recall values of our work and the reported TPR, our work achieves a minimum of 0.72 and a maximum of 0.94 recall, similar to the results reported by Barrera et al. [21]. Meidan et al. reported a classification accuracy of 95%–99% [9]. Bai et al. [11] reported an accuracy of at least 74.8% and at most 80.1% when classifying network traffic data. These results are not directly comparable to our work. However, the achieved accuracy values indicate a high possibility of detecting malicious traffic, similar to the one achieved by our clustering method.

Execution Time. When detecting a deceitful attacker-generated sequence S using clustering, the execution time depends on the speed of the clustering algorithm to find the best-fit cluster for S. When using a classification method, the time depends on the speed of classifying S according to the trained model. In both cases, the time required to detect if a single sequence S is benign or deceitful is negligible (running on a machine with a 6-core Intel i7 CPU and 32GB of memory). Here, the execution time for the clustering method is presented since it is expected to be a more expensive operation than the classification method. The detection time involves finding the cluster to which S belongs and detecting if S is a noise point (Algorithm 1). To show the time required for processing a number of sequences, we measured the time to process the attacker sequences generated for the experiments. Note that in our experiments, the simulations were conducted by dividing the data by days of the week and then by intervals of a day, then aggregating over the days of a week. That is, each dataset measures the number of events for one interval of a day over a week. Depending on the interval length and the number of attack data points, the required verification time varies. Table V shows the execution times needed to process the entire attacker dataset for each interval. Processing a single attacker sequence takes at most 22 milliseconds.

E. Discussion

The results for anomaly detection show the effectiveness of using application metadata with the sensor data of the testbed. While the anomaly detection methods are not perfect in detecting fabricated attacker messages, given enough time, anomaly detection methods can detect malicious behavior from compromised devices. For example, the clustering approach could detect 70% of data generated from the mimicry attacker on the device v_0 (when $m(\delta) = 2$). This behavior could be monitored by the controller to first limit interaction with the device by enforcing more restrictive policies using the device proxy. If the behavior persists, the controller can decide to segregate the compromised device by using the device signature on the sequences. Also, the controller can isolate the data generated by the compromised device since the time at which malicious behavior was observed.

VIII. Conclusion

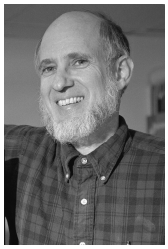
We presented an anomaly detection method using application data along with an architecture that uses device proxies to control access to devices and collect the relevant data. The architecture uses the results of the presented anomaly detection

method to decide which devices are compromised. Our results demonstrate the potential feasibility of the approach through an experimental case study using data generated from a typical IoT subnetwork environment with no specific controls on the environments except for the locations of the sensors. While the results are promising, many more settings should be tested to adjust the anomaly detection methods better and develop a more comprehensive IoT anomaly detection system. Our architecture could be extended to include a full policy engine that can directly use the results of the anomaly detection model to adjust policies across the network. One possible application of the attack-resilient IoT architecture model is in enabling interactions among various IoT devices belonging to neighboring subnetworks. In this case, the exchanged data in a controlled environment can be used to construct a model of inter-device interactions. An anomaly detection method can then be used to detect and isolate malicious exchanges.

ACKNOWLEDGEMENTS

This work was partially supported by a grant from the National Science Foundation SaTC Program (#1804603).

Hussain M. J. Almohri received the BS degree in Computer Science from Kuwait University and the Ph.D. degree in Computer Science from Virginia Tech in 2013. He is currently an assistant professor of computer science at Kuwait University. He has co-founded a mobile payment startup and has advised a number of software startups in the Gulf region. His research focuses on systems and network security. He has served as a reviewer for several IEEE and IET journals and Kuwait Journal of Science.



Layne T. Watson (F '93) received the B.A. degree (magna cum laude) in psychology and mathematics from the University of Evansville, Indiana, in 1969, and the Ph.D. degree in mathematics from the University of Michigan, Ann Arbor, in 1974.

He has worked for USNAD Crane, Sandia National Laboratories, and General Motors Research Laboratories and served on the faculties of the University of Michigan, Michigan State University, and University of Notre Dame. He is currently a professor of computer science, mathematics, and aerospace and ocean engineering at Virginia Polytechnic Institute and State University. He serves as senior editor of Applied Mathematics and Computation, and associate editor of Computational Optimization and Applications, Evolutionary Optimization, Engineering Computations, and the International Journal of High Performance Computing Applications. He is a fellow of the National Institute of Aerospace and the International Society of Intelligent Biological Medicine. He has published well over 300 refereed journal articles and 200 refereed conference papers. His research interests include fluid dynamics, solid mechanics, numerical analysis, optimization, parallel computation, mathematical software, image processing, and bioinformatics.

David Evans is a Professor of Computer Science at the University of Virginia. He is the author of an open computer science textbook, a children's book on combinatorics and computability, and a textbook on secure multi-party computation. He has SB, SM and PhD degrees in Computer Science from MIT and has been a faculty member at the University of Virginia since 1999.