ARGUS: Context-Based Detection of Stealthy IoT Infiltration Attacks

Phillip Rieger Marco Chilese Reham Mohamed

Markus Miettinen Hossein Fereidooni Ahmad-Reza Sadeghi

Technical University of Darmstadt

Abstract

IoT application domains, device diversity and connectivity are rapidly growing. IoT devices control various functions in smart homes and buildings, smart cities, and smart factories, making these devices an attractive target for attackers. On the other hand, the large variability of different application scenarios and inherent heterogeneity of devices make it very challenging to reliably detect abnormal IoT device behaviors and distinguish these from benign behaviors. Existing approaches for detecting attacks are mostly limited to attacks directly compromising individual IoT devices, or, require predefined detection policies. They cannot detect attacks that utilize the control plane of the IoT system to trigger actions in an unintended/malicious context, e.g., opening a smart lock while the smart home residents are absent.

In this paper, we tackle this problem and propose ARGUS, the first self-learning intrusion detection system for detecting *contextual attacks* on IoT environments, in which the attacker maliciously invokes IoT device actions to reach its goals.

ARGUS monitors the contextual setting based on the state and actions of IoT devices in the environment. An unsupervised Deep Neural Network (DNN) is used for modeling the typical contextual device behavior and detecting actions taking place in abnormal contextual settings. This unsupervised approach ensures that ARGUS is not restricted to detecting previously known attacks but is also able to detect new attacks. We evaluated ARGUS on heterogeneous real-world smart-home settings and achieve at least an F1-Score of 99.64% for each setup, with a false positive rate (FPR) of at most 0.03%.

1 Introduction

IoT devices are becoming an integral part of modern life in many application domains like smart homes, smart buildings, smart city infrastructure, and smart factories. Increasingly IoT devices are also providing access to rich contextual information, making it possible to realize intelligent ambient environments in which whole systems of interconnected IoT devices are controlled in a coordinated and adaptive way. For instance, IoT devices can sense data about the movements and behavior of smart home users and automatically adapt lighting, heating, or air conditioning settings accordingly.

In 2020, more than 11.3 billion IoT devices were deployed in smart homes and more than 27 billion devices are expected for 2025 [8]. In the US, 23% of all broadband households have already 3 or more connected devices [46] and one can expect this number to increase since more and more functions will be controlled by IoT devices.

Attacks on IoT Systems. The continuously growing number and diversity of IoT devices from different device manufacturers enlarges the potential attack surface in many IoT networks. While different approaches have been proposed to detect attacks that directly compromise IoT devices, e.g., through IoT malware [32, 36, 40], more stealthy attacks compromising the *contextual integrity* [13] of IoT networks by misusing the *control plane* of IoT devices, i.e., local and remote systems and applications like vendor-provided smartphone apps or cloud services that are used to control the IoT devices, have not been sufficiently addressed yet.

Detecting attacks on contextual integrity is highly challenging. Since both, benign actions and *contextual attacks*, use regular control commands for triggering legitimate actions and also the network traffic does not necessarily differ, benign and attack-actions are as such indistinguishable. The only difference between them is the context of their invocations, identified through the environmental factors and stages of the other devices in the smart-home. Therefore the attack-actions are invoked in a situation, where the user does not desire it.

For instance, an attacker who infiltrated the vendor's cloud service, e.g., due to weak credentials, could instruct a smart lock in the apartment of the victim user to open the door while the user is absent so that the attacker can then break into the user's home. The attacker could also compromise the safety of a smart home by sending a command to turn on the smart stove, causing a potential fire hazard. Also other, non-intentional device or system failures may compromise the safety of the target environment, e.g., if the IoT-controlled

heating system fails and turns off completely while the user is on vacation, such that the indoor temperature falls below the freezing point. Unlocking the door, turning off the heating, or, turning on the stove, considered individually, are perfectly normal actions. Consequently, if the attacker uses the control plane to trigger these actions, the related commands look like perfectly benign commands. Since the commands are sent in these examples by the vendor-specific cloud platform, the network traffic does not differ from the traffic of benign actions. Therefore, in order to detect such attacks, also the *context* of actions that IoT devices undertake needs to be considered.

Existing Defenses. Existing approaches for IoT intrusion detection systems often focus on analyzing the network traffic [24, 36, 37]. However, these approaches cannot detect contextual attacks, as the network traffic of attack commands is indistinguishable from benign commands. The remaining approaches that consider contextual information fall into different categories. The most relevant categories are: 1) Validation of Sensor Values, 2) Policy Enforcement (either defined rules or dynamically through graph representations), and 3) systems for Contextual Anomaly Detection. The systems in the first category [9,35,49,50] often focus on detecting anomalies only with regard to a specific sensor. This enables them to detect wrong measured sensor values, but making them fail to recognize abnormal physical system states. For example, if the door of an apartment is physically opened during the users' absence, then the values of the sensors are correct when indicating this, although opening the door while nobody is at home should be considered anomalous.

Approaches in the second category use fixed policies [15, 20, 25, 33, 47, 48] that are often determined by non-dynamic processes. A user must either extract these policies from the source code/description of apps that control IoT devices, ignoring the actual user behavior, or they must be manually defined by the users, which is inconvenient.

Existing approaches that fall into the third category by training anomaly detection modules based on captured training data [10,26,41–43] also suffer from the requirement of semantic information about the IoT devices [26], being restricted to having example data for the attacks [43], to analyzing only the commands [10], or cannot model the relationships between the individual devices accurately [41, 42].

An autonomous approach that monitors the context of the IoT devices without requiring access to source code is therefore needed. If the attack is detected in time, the user can take countermeasures in time to mitigate the attack, e.g., by calling the police if the door lock is opened while she is absent.

Our Approach. In this paper, we propose ARGUS, a novel approach for detecting contextual attacks against IoT networks, i.e., attacks which perform benign actions in a wrong context. ARGUS is a complementary solution augmenting existing network traffic monitoring-based intrusion detection approaches that focus on detecting direct attacks on IoT devices (e.g., IoT malware attacks) but cannot detect contextual attacks utilizing the control plane. To detect such stealthy, contextual attacks, ARGUS is inspired by the notion of *contextual integrity* [13] and makes use of the observation that the acceptability and permissibility of an action are highly dependent on the contextual setting in which the action is taking place. It models the context in terms of events, user actions, device actions, and triggered automation rules to overcome the limitation of network-traffic-based approaches. Another challenge that ARGUS addresses is that what is considered "normal" behavior is highly dependent on individual networks and users, preventing the use of simple static policies for distinguishing between malicious and benign actions. To address this challenge, ARGUS trains a Deep Neural Network (DNN) for capturing the interdependence between contextual factors and events and device actions. With the help of the trained DNN, ARGUS evaluates the actions of IoT devices in the network and determines for each action an anomaly score to detect anomalous situations in which an invoked action is not consistent with the *contextual setting* in which it is occurring. Our contributions are as follows:

- We present ARGUS, a context-based intrusion detection framework for IoT networks capable of detecting IoT infiltration attacks in which the adversary compromises the control plane of the network, e.g., cloud servers or mobile apps, to stealthily manipulate the behavior of devices to achieve malicious goals. Thus, ARGUS detects contextual attacks where the individual device actions are normal but are performed in a wrong context (§3.1).
- We develop a dynamic tuning scheme for the classification boundary of events' anomaly scores that automatically adapts to different setups (§4.3).
- We collect and provide the first real-world dataset, capturing the behavior of different smart-homes to be utilized by research community for conducting future research in this area[1] (§5.1).
- We extensively evaluate ARGUS on the collected real-world dataset, consisting of 5 heterogeneous smart-home setups (§5.2).

2 Problem Setting

In this section, we first briefly elaborate on recent IoT attacks and then explain our system model and design to mitigate such attacks. Afterward, we describe the contextual threat model and challenges that ARGUS is designed to solve.

2.1 Recent IoT Attacks

In the course of IoT market proliferation, an increasing number of attacks on IoT devices have been reported [38]. Some prominent attacks were related to Mirai botnet [11] and its

[1] https://github.com/TRUST-TUDa/argus-data

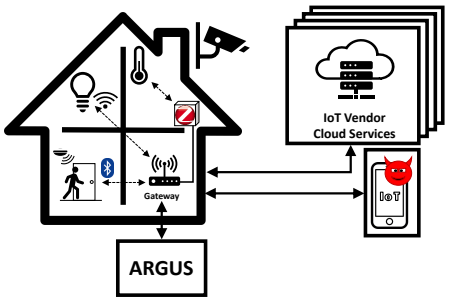


Figure 1: ARGUS system model

successors [29], where a massive number of compromised IoT devices was used to stage one of the largest DDoS attacks ever recorded on the internet. However, recently other attacks have emerged that specifically target functions controlled by IoT devices, for instance, a botnet of IoT devices was able to effectively incapacitate the heating system of a large residential building while the temperature outside was far below freezing [22]. Obviously, such attacks could have even more damaging consequences, if instead of a heating system, e.g., vital devices in a hospital would be targeted.

These examples demonstrate the need for effective countermeasures against attacks targeting the *control plane* of IoT networks, i.e., the systems, protocols, and mechanisms controlling the functionality of the IoT devices.

2.2 System and Context Model

Our system model, shown in Fig. 1, considers a heterogeneous IoT network consisting of different IoT devices controlling various functions related to their ambient environment and measuring different parameters of the environment and their operational context. Some of the IoT devices may be associated with vendor-specific IoT cloud services and may utilize associated mobile applications for allowing the user to remotely control these devices. All devices are connected to the local network, which typically also provides access to the internet via an access gateway. Devices connect to the local network using Ethernet or WiFi, or, a specific hub device providing IP-connectivity for devices using wireless proximity protocols like Bluetooth, ZigBee, or Z-Wave.

Via the internet, the IoT devices are connected to the control plane, consisting of all local and remote systems and applications that are used to legitimately control the IoT devices. In the rest of the paper, we assume that the attacker compromises one of these entities (cf. §2.3).

An example of such a setup is shown in Fig. 1. Here, the smart-home consists of a light-bulb (connected through WiFi), a smart-lock (connected through Bluetooth), an IP-Camera (connected through LAN), and a smart thermometer (connected through ZigBee via a hub). Each device uses a different cloud platform and the user controls them via vendor-specific applications on its mobile phone. The cloud platforms and the mobile applications represent here the control plane of the setup. The goal of ARGUS is to monitor the actions taken by individual IoT devices, e.g., turning on the light or unlocking the door, and notify the user in case it detects device actions that are not consistent with the current contextual setting.

The main focus of ARGUS are *contextual attacks* where the attacker uses benign functions of IoT devices triggered in incorrect contextual settings to stage attacks for achieving its attack goals. For detecting such attacks, we model the context of the IoT network in terms of a number of context features characterizing the system's contextual state. The considered contextual features can be roughly classified into three main categories: 1) Ambient and temporal features describing the environment (noise level, luminosity, humidity, temperature, time of the day, etc.), 2) Features indicating the context of the user (asleep/awake, present/absent, etc.), 3) Device states (device state changes, triggered automation rules, event notifications, device alarms, etc.).
The context features can be automatically harvested from the monitored IoT system using appropriate APIs of individual IoT devices, their associated cloud services, and possible home automation systems installed in the user's network. As described in more detail in §3, ARGUS aggregates these contextual factors and feeds them to a machine learning algorithm used to profile the contextual state of the local IoT setup and perform anomaly detection.

2.3 Adversary Model

We consider an adversary $\mathcal{A}$ that compromises a part of the IoT control plane to trigger normal-looking actions on the IoT devices but in the wrong context. Therefore, the legitimate user does not want these actions to be executed. Note that for staging the attacks $\mathcal{A}$ does not need to actually compromise the targeted IoT devices and execute malicious code locally on the IoT device itself. It is sufficient to abuse the compromised control plane to invoke commands. The invoked commands look, when considering them individually, benign and might, in another context, be also invoked by the user itself. Therefore, only the context of invocation allows to distinguish benign actions and attacks.

For compromising the control plane, there exist a number of attack vectors, e.g., when the IoT device or the related cloud service use insufficient authentication (weak, default [3], or even missing passwords [5]), or a malware placed on the smartphone of the user targeting specific IoT apps.

An example of a contextual attack is shown in Fig. 1. $\mathcal{A}$ compromises an app on the user's mobile phone, which is part of the control plane and uses it to post a control command to the smart lock, causing it to open the door while the user is asleep or not at home. In general, unlocking the door using the mobile phone is a benign event, which is also invoked by the user. However, the normal context for such actions would be, e.g., that the lock opens when the user is returning home and approaching the door. The fact that the user is asleep, or,

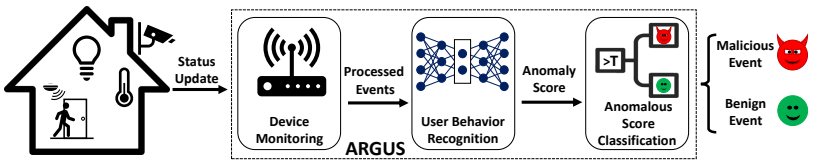


Figure 2: High-Level Overview of the components of ARGUS

absent from home clearly represents an abnormal context for the smart lock action of opening the door.

The motivation of $\mathcal{A}$ for the attack is to invade the users' privacy, cause financial damages to the user, or harm the user in some other way.

However, we assume that $\mathcal{A}$ neither compromises the actual IoT devices nor ARGUS, as we will elaborate in our trust model (cf. §3.2).

2.4 Requirements and Challenges

For detecting contextual attacks in realistic real-world settings in an effective and user-friendly manner, the ARGUS system should satisfy the following requirements:

R1 Fast detection: Since contextual attacks can potentially lead to significant physical damages or monetary loss to the user, attacks must be detected in near real-time when new incoming events take place in the system in order to allow sufficient reaction time for taking appropriate countermeasures.

R2 Cause identification: The system must be able to identify the device or event that is causing an alarm to be triggered. This is necessary to allow the user to understand what the root cause is and choose appropriate countermeasures to mitigate the attack.

R3 Minimizing false alarms: The system must not generate many false alarms to ensure that the user is not overwhelmed with false alarm notifications. Otherwise, the user will likely start ignoring incoming alerts, or, disable the protection system altogether to avoid unnecessary inconveniences.

R4 Autonomous operation: The system must run with minimal configuration input from the user. Users should be required to perform only the basic configurations while the system should take care of training and applying the necessary contextual models for attack detection. Otherwise, the system will not be practical as users are not likely to have sufficient expertise and be willing to spend considerable effort in configuring the system.

To cater to the above requirements, the system must thus solve the following technical challenges:

C1 Detecting attacks consisting of benign actions: As discussed in §2.2, contextual attacks constitute situations, in which the adversary triggers actions in wrong contextual situations where the actions themselves represent legitimate benign actions of the device. The main challenge is how one can detect an attack consisting of *benign* actions? Some existing approaches monitor the network traffic of IoT devices to detect known attack patterns, or, deviating traffic patterns caused by potential attacks [24,28,32,36,37,40]. In the setting of contextual attacks, these approaches are, however, not applicable, as the traffic patterns used to stage the attack represent in essence 'benign' operations of the devices.

ARGUS seeks to resolve this challenge by utilizing a context model detailed in §4 that explicitly links device actions with the contextual setting they take place in. This allows the detection model to enforce contextual integrity by learning what sequences of events are benign, normal event sequences, and what represent potential attacks. This also allows to detect attacks immediately when they occur (R1) and to specifically indicate, which particular action it was that triggered detection, thus helping in identifying what device action caused a potential alarm (R2).

C2 Autonomous defense personalization: For minimizing the number of false alarms (R3), the detection system must be tailored to the local IoT set-up and personalized to consider personal preferences and habits of users. The challenge here is, how this can be done without requiring extensive manual configuration of enforcement policies to enable autonomous operation of the system (R4)? Earlier systems addressing contextual attacks heavily depend on explicit input from the user to define policies determining permissible and undesired IoT device actions in particular contextual situations [16, 23, 28, 44]. Such approaches are unlikely to work in general in real-life settings, since regular IoT users are very unlikely to have the required experience or motivation to spend a considerable amount of time and effort in setting up secure and effective security policies for their IoT devices that sufficiently accurately match their IoT set-up and personal preferences. ARGUS seeks to tackle this challenge by replacing pre-specified static policies by a trained detection model that can be trained requiring only minimal explicit inputs from users.

3 System Design

In the following, we provide a high-level overview of ARGUS, the first context-based IoT intrusion detection framework that monitors connected IoT devices together with their context to detect abuse. ARGUS uses an anomaly detection approach that compares newly captured events against the modeled behavior to detect anomalous actions (e.g., unseen attacks). In §3.2 we elaborate on the considered trust model and security assumptions.

3.1 High-Level Overview

A high-level overview of ARGUS is depicted in Fig. 2. It involves the following entities: Device Monitoring, Context Modeling, and Anomaly Score Classification components to monitor the actions of the IoT devices and detect anomalous behavior. In the following, we outline the role of each component. Details for each component are discussed in §4.

Device Monitoring. The monitoring component collects the status updates and event notifications from the observed IoT devices and preprocesses them for the following components. For managing the different APIs of different IoT ecosystems, ARGUS makes use of a home automation platform that connects to the individual APIs of the various IoT devices installed in the system.

Context Modeling. This component utilizes an Auto-Encoder (AE) architecture [14] which is an unsupervised DNN approach, since it uses only benign data for training. The AE predicts anomaly scores for each new incoming event that is captured by the Device Monitoring component. The score indicates the similarity of the captured events with the modeled expected behavior.

Anomaly Score Classification. The final component of ARGUS uses a dynamically calculated threshold as classification boundary to discriminate each event based on its anomaly score as benign or malicious. The threshold is based on the anomaly scores of the previous day as well as the previous threshold. An event is considered to be an attack, if the anomaly score is higher than the threshold.

3.2 Security and Trust Assumptions

We make following assumptions with regard to the capabilities of the adversary $\mathcal{A}$ and the trusted system components:

- Aligned with existing work ARGUS considers only attacks that compromise the control plane of the IoT network but not direct attacks against the IoT devices themselves [33]. This is because there exists a large body of work focusing on detecting direct attacks against IoT devices [24, 28, 32, 36, 37, 40]. ARGUS should thus be seen as a complementary approach augmenting these defenses by providing the ability to detect also contextual attacks targeting the control plane. Consequently, since $\mathcal{A}$ cannot compromise the IoT devices, e.g., to run code locally, it also cannot suppress or fake status updates. It also cannot utilize IoT devices to impersonate other IoT devices [27, 42], as this would require compromising the utilized IoT devices first.
- Aligned with previous work [16, 47], we assume the individual components of ARGUS to be trusted. While during the development of IoT devices there is less focus on security considerations, ARGUS is specifically designed to increase the security, such that security experts are involved in its implementation and also it is reasonable to expect the system to be hardened for security. Because of the focus on security considerations during the implementation and the limited set of functions, we assume that $\mathcal{A}$ cannot compromise the components of ARGUS.
- Aligned with previous work [26, 36], we assume the local IoT setup to be not compromised during training time.

4 ARGUS

In the following we describe the individual components of ARGUS (Device Monitoring, Context Modeling, Anomaly Score Classification) in detail.

4.1 Device Monitoring

The Device Monitoring component collects status updates from the individual IoT devices, allowing ARGUS to also consider the context of an invoked action to determine whether the action is benign (addressing C1). A challenge here is the very heterogeneous IoT device landscape. Different devices from different manufacturers use different protocols, are connected to different cloud platforms and mobile applications, and might even use different communication technologies. For example, one device might use WiFi to connect to the local network while other devices might use ZigBee and must use a dedicated ZigBee hub device as a gateway to connect to the local network. For these reasons, most existing intrusion detection systems (IDS) consider only the LAN and WiFi traffic but do not consider other wireless communication protocols [24, 36, 37].

The Device Monitoring component of ARGUS addresses this challenge by making use of a home automation system. A number of such systems that integrate various IoT devices and allow to control them in one central place have been developed in the past [1, 2, 4, 6]. While some manufacturers might put less focus on security aspects, an easy integration and access is vital for deployment and user acceptance of IoT devices creating a strong motivation for manufactuers to allow a simple integration into existing HAPs. The Device Monitoring of ARGUS exploits this by utilizing the automation system for the integration of the devices, maintaining the adapters to use the APIs of the different IoT ecosystems, and

Table 1: Model Hyperparameters

Variable	Setting
Encoder Layer Type	[GRU, GRU]
Decoder Layer Type	[GRU, GRU]
Encoder Hidden Units	[GRU:256, GRU:64]
Decoder Hidden Units	[GRU:64, GRU:256]
Optimizer	Adam
Loss Function	MSE
Learning Rate	Decaying from 10^{-3} to 10^{-6}
Dropout Value	0.3
Epochs	Max 35 000
Batch Size	64

collecting the events from IoT devices of different vendors and technologies. This allows ARGUS to use all available devices for monitoring the context.

When a device changes its status, the device reports this status update to the home automation system. The remaining part of the Device Monitoring component then records the status update, parses the status if necessary to ensure a standard data format, adds the new event to the sequence of previous events, orders all events by their occurrence time, and forwards them to the Context Modeling component.

4.2 Context Modeling

In order to distinguish between a normal scenario and a suspicious event in view of the usual user and system behavior, the Context Modeling component of ARGUS models the expected behavior and its context based on the previously collected training data. In the following subsections, we will describe the data preprocessing as well as the architecture of the Deep Neural Network (DNN) that is used for modeling the user's expected behavior.

4.2.1 Data Preprocessing

The data preprocessing is performed in the following steps:

1. Parsing log data: The captured events are parsed in order to store the information about the devices' and sensors' status update events. For the ease of presentation, in the following device states refer to the states of the individual IoT devices as well as the captured values of the individual sensors.

2. States' value mapping: In order to deal with the multitude of devices' states and for facilitating the subsequent phase of ML, all device states are mapped into numeric values in a restricted range. Devices that have only a limited or nominal set of states (e.g., "on" and "off") are mapped in range $[0,1]$. Each observed state is first mapped to a cardinal number state_{id}, which is then normalized corresponding to:

$$S_i = \frac{1}{|\text{states}|} \cdot \text{state}_{i_{\text{id}}} \tag{1}$$

where S_i is the state i in the set of states for that device, $|\text{states}|$ is the set of all states, and $\text{state}_{i_{id}}$ is the cardinal number of that state in the set. The state S_0 is reserved for new values that were not observed during the training phase. Continuous values (e.g., temperature and humidity values) are mapped to 10 values in range $[0,1]$ corresponding to $[\text{state}_{min}, \text{state}_{max}]$. The interval r_i for the values that are mapped to value $i/10$ is given by :

$$r_i = \left[S_{\min} + i\frac{S_{\max} - S_{\min}}{10},\ S_{\min} + \frac{S_{\max} - S_{\min}}{10}(i+1) \right] \tag{2}$$

where $S_{\max|\min}$ is the max or min value of the states' set of that device, and $i \in \{0,\ldots,9\}$ is the numerical value of each bucket of the new mapping, reserving S_0 for future unseen values.

3. Event chain construction: For each moment in time of the recorded events, the state of each device in the system is reconstructed from the status updates, for having a complete view of the system at every moment in time. The resulting chain is characterized by a list of events: $[\text{event}_0, \text{event}_1, \ldots, \text{event}_n]$ where

$$\text{event}_i = [\text{state}_{\text{device}_0}, \text{state}_{\text{device}_1}, \ldots, \text{state}_{\text{device}_m}] \tag{3}$$

in a particular moment in time i, where event_0 is the first event recorded and event_n is the last one.

4. Sequence building: The event chain is converted into event windows of size l. Therefore, the chain is split accordingly in groups of size l for producing feature vectors with shape $(l, N_{devices})$.

4.2.2 Deep Learning Model

ARGUS models the users' normal behaviors and the context to distinguish between abnormal and normal behaviors by using an Auto-Encoders (AE) architecture [14] for the Deep Neural Network (DNN). AEs are widely used in anomaly detection tasks [18, 34, 52]. They consist of two parts, an Encoder and a Decoder. We are using an undercomplete AE, so the size of information decreases layer by layer in the Encoder until a "bottleneck" where information reaches the point where the model has extracted all the hidden patterns in data (information at this stage is known as "encoded data"). From this point on, the information will be reconstructed by the Decoder, expanding it layer by layer, to reproduce the input data. Finally, the amount of error made in the reconstruction is measured by using the mean squared error (MSE). The reconstruction error of the AE is used as the anomaly score of an event.

Since the kind of data we are dealing with has a temporal structure we have designed an undercomplete AE made of recurrent unit layers, in particular, based on Gated Recurrent Units (GRU) [19]. The choice of using GRU layers is guided by the fact that we need to be able to learn latent patterns in temporal context for being able to recognize user's behavior in benign scenarios. This enables ARGUS to learn even desired random behavior, e.g., randomly turning on/off lights during specific times.

Encoder and Decoder are made of two recurrent layers each one, respectively with decreasing and increasing number of hidden units. The architecture of our AE model is depicted in Fig. 3. The hyperparameters are shown in Tab. 1.
The training task concerns reconstructing input data. To do so, the model learns the latent patterns and hidden representation

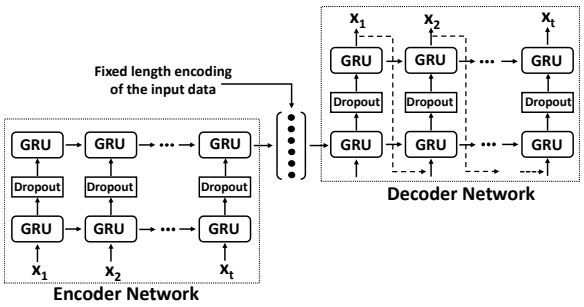


Figure 3: ARGUS Auto-Encoder (AE) architecture

of data, i.e., learning the user's behavior. The AE architecture allows to use only benign data for training the DNN such that the model at end of the process will only be able to reliably reconstruct the benign data (i.e., benign scenarios), while abnormal data, i.e., suspicious events, cannot be encoded effectively (resulting in a larger reconstruction error) since they are not exposed to the model during the training. A large reconstruction error that exceeds a threshold allows the ARGUS system to recognize an event as malicious.

4.3 Anomaly Score Classification

The Anomaly Score Classification uses the anomaly scores, i.e., the reconstruction errors, which were predicted by the Context Modeling component to determine whether the given status update is benign or malicious. To discriminate these values, a threshold T_d is determined dynamically based on the previously observed anomaly scores and used as classification boundary. An event is classified as benign iff its reconstructions error is smaller or equal to T_d, otherwise an alarm is raised.

For calculating the threshold T_d that is used on day d + 1, for each previous day $d^* \leq d$ a so-called threshold candidate C_{d^*} is calculated:

$$C_{d^*} = \max(E_{d^*}) + \beta \cdot (\max(E_{d^*}) - \min(E_{d^*})) \quad (4)$$

where E_{d^*} is the set of reconstruction errors of all events that were collected on day d^* and β represents the security level. To prevent that exceptional high reconstruction errors significantly affect the threshold, a momentum [31] is used to combine C_d with the previous threshold T_{d-1}. The threshold T_d is then given by:

$$T_d = \begin{cases} \alpha \cdot T_{d-1} + (1-\alpha) \cdot C_d & d > 0 \\ C_0 & d = 0 \end{cases} \quad (5)$$

where α is the aging factor that determines the impact of the previous threshold. The higher α is, the more impact the previous threshold has.
Since ARGUS captures the events and calculates the anomaly score without any delay, ARGUS fulfills by design R1 (Fast detection). As an anomaly score is determined for every new event, it is possible to identify the reason of the anomaly by presenting the last event as well as a short list of preceding events to the user such that ARGUS also fulfills R2.

4.4 Implementation

In the following, we describe the implementation details of the evaluation of ARGUS in §5. Further, we define the values for the parameters of ARGUS, such that the only adaption that users need to make is the decision, which devices shall be monitored. Therefore, ARGUS fulfills R4 (Autonomous operation).
Device Monitoring. For the monitoring, ARGUS utilizes a home automation platform (HAP) that connects different devices and collects their states. In addition, the user can specify and create specific automation rules or configurations for his own network. Using a HAP enables ARGUS to monitor a large variety of devices from different manufacturers and ecosystems for observing the context. The HAP keeps track of the status of each connected device or sensor in the local home network. Thus, it enables ARGUS to monitor the contextual features automatically, without requiring it to set any feature manually.

For the experiments, we used Home Assistant, an open-source platform that supports at the time of writing more than 1000 devices and also protocols like MQTT [4]. Users can also control the connected devices from outside the network, e.g., via a mobile app.

A further advantage of the remote access to the HAP is that it allows a flexible deployment of ARGUS. Due to the separation of individual components, the Context Modeling and Anomaly Score Classification can be placed outside the user's home, e.g., on some trusted cloud servers. Then, only the Device Monitoring needs to be placed inside the user's network, such that this component can be installed, e.g., on a low-performance device. Alternatively, Especially in the case of privacy concerns of the user, the design and implementation of ARGUS also allow a completely local deployment, e.g., on a low-performance device like a RaspberryPi (cf. App.B). By this, ARGUS also allows a network setup where no data related to the IoT devices leaves the local network.

Context Modeling. As described in §4.2.2, the ARGUS AE architecture is generally applicable and depends only on the number of devices involved in the analysis and on the width of the event window size $l = 16$. By this, ARGUS addresses C2. Encoder and Decoder are made of two layers each with a fixed number of hidden units, respectively, decreasing (from 256 down to 64) and increasing (from 64 up to 256 for compressing and then reconstructing the input.

For analyzing the user's behavior, the temporal context encoded in the event sequences, needs to be considered. So, we need to take into account the temporal structure of data

in order to have a model that is able to learn those patterns. For that reason, we use recurrent layers, in particular, Gated Recurrent Unit (GRU) layers. We used GRUs units rather than alternatives such as LSTMs [30] due to their capability of faster convergence, requiring less memory (i.e., less trainable parameters) and better dealing with long-term memory problems (e.g., vanishing/exploding gradient).

The size of the final model depends on the number of devices in the system. In our setups, the number of devices varies from 18 to 40, so the model size varies from 1.2 to 2.7 million trainable parameters.

The designed learning process for such models uses a decaying learning rate (from 10^{-3} down to 10^{-6}) once the number of epochs reaches some specific epochs. Furthermore, we make use of early stopping with patience monitoring validation loss in order to prevent overfitting behaviors.

Anomaly Score Classification. The threshold that is used as classification boundary to discriminate benign events from attacks makes use for two parameters, the aging factor α and the security level β.

α is used for performing a trade-off between allowing the threshold to dynamically react to changes in the behavior and preventing high changes in the threshold, which could cause incorrect classifications (cf. App. E). We therefore set $\alpha = 0.8$.

The security level β ensures an additional margin to prevent unusual behavior of the user from causing false alerts. Too high values prevent ARGUS from detecting attacks, while too low values cause false alerts. We empirically set $\beta = 0.2$ (cf. App. E).

5 Evaluation and Discussion

In this section, we discuss how different attacks are applied to the network and how our approach is able to detect such attacks. For the evaluation, we evaluate 5 real-world IoT setups. The first subsection shows the modeling of the benign data and the distribution of the testing and training data. The next subsection discusses the real network and how the attacks are performed and detected. For the evaluation, we use the well established performance metrics F1-Score, Precision, Recall and False-Positive-Rate (cf. App. G). In App. I we describe the computational setup, in App. B we evaluate the runtime performance of ARGUS.

5.1 Dataset

For our experiments, we equipped 5 different smart-home settings with a variety of IoT devices and sensors that were used by residents on a daily basis. From each setup, we used the first 7 days for training the model and the remaining data for testing. The data for training were split into 90% of actual training data and 10% validation data.

Table 2: Performance of ARGUS on Real-World Setups, all values in percentage.

Dataset	FPR	Pr	Re	F1-Score
Home 1	0.03	99.22	100.00	99.64
Home 2	0.00	100.00	100.00	100.00
Home 3	0.00	100.00	100.00	100.00
Home 4	0.00	100.00	100.00	100.00
Home 5	0.00	100.00	100.00	100.00

5.1.1 Dataset Collection

For collecting the dataset, we captured the status updates of IoT devices in 5 different smart-home environments (referred to as Home 1 - Home 5). Each setup consisted of multiple sensors (temperature, humidity, brightness and motion, door and window sensors) and actors (light bulbs, thermostats). To make the individual setups differ from each other and evaluate the ability of ARGUS to generalize, each setup also had some additional sensors and actors, making the dataset heterogeneous. For example, in the setup Home 1 also a CO_2 sensor was installed, while in the setups Home 4 and Home 5 also a number of smart thermostats were installed. In App. A, we show the deployed sensors and actors for each setup. For the data collection, the popular open-source smart-home control system Home Assistant was used (cf. §4.4).

The devices were installed in different homes, covering a one-person room in a shared apartment (Home 1), an one-person apartment (Home 2), as well as shared homes with 4 inhabitants each (Home 3, Home 4, Home 5). The experiments included ten different male and female participants (teenagers, students, and adults up to approximately 49 years). Initially, controlled experiments incorporating a number of simulated attack scenarios were executed in the simpler attack settings in Home 1 and Home 2, since these environments included only one inhabitant and were therefore easy to control. The more complex contextual settings incorporating several persons in Home 3 and Home 4 were used for passive data collection, without active attacks, mainly to test the sensitivity of the approach for false alarms under a more challenging setting. Finally, the most complex IoT set-up was implemented in the multi-person setting Home 5, where also controlled experiments with attacks were implemented to test the full performance of ARGUS in complex real-world settings. Each setup used the home automation platform for automatically triggering actions, e.g., turn off the camera when the user comes home, turn of the heating when the window is opened, or reduce the heating temperature during the night.

5.1.2 Ethical Considerations

The dataset collection raised ethical concerns, as the recorded behavior of the users might contain sensitive data. We addressed these concerns by ensuring that all affected persons, i.e., the users as well as all guests, were aware of the data

Table 3: Categorization of evaluated attacks into Event Spoofing (ES), Event Interception (EI), Command Spoofing (CS), and Command Interception (CI)

Attack	ES	EI	CS	CI
Door open while absent		•	•	•
Lights on while absent	•		•	
Movement while absent	•			
Camera off while absent	•		•	
Light flickering			•	
Heating on while open windows	•	•		•
Lights on during night			•	
Fake Fire closed windows	•			
Fake Fire open windows	•			

collection and gave their consent. Further, we limited the approach to non-privacy-sensitive sensors and excluded the other sensors like the geolocation or the SSID of the WiFi network that the mobile phone is connected to. In addition, all potentially sensitive data items were anonymized. Our experimental set-up has been reviewed and approved by the ethics board of our university.

5.2 Experimental Results

The performance of ARGUS for the individual setups is shown in Tab. 2. The table shows the results in terms of FPR and for the setups where attacks were performed (Home 1, Home 2, Home 5), also the Precision (Pr), Recall (Re), and F1-Score. As Tab. 2 shows, ARGUS recognizes almost all benign events correctly (FPR $\leq 0.3\%$) and thereby fulfills R3 (Minimizing false alarms). Tab. 2 also shows that ARGUS detects almost all the attacks (Re $\geq 99.64\%$). The detailed results for the individual attacks are shown in Tab. 4 and discussed in §5.2.1.

5.2.1 Attack Detection

To evaluate the performance of ARGUS, we performed various attacks. The malicious behavior for these attacks can be categorized as follows:

1) **Event Interception (EI)**: $\mathcal{A}$ intercepts (i.e., suppresses) an event, for instance, $\mathcal{A}$ intercepts the event "`window open`" to not turn off the heating, creating a potential monetary damage to the user.

2) **Event Spoofing (ES)**: $\mathcal{A}$ spoofs (i.e., creates) a fake event. For instance, $\mathcal{A}$ creates the fake event "`user at home`" when the user is not at home to trigger the command "`turn off the camera`", e.g., for breaking into the house without being recorded.

3) **Command Interception (CI)**: $\mathcal{A}$ intercepts (i.e., suppresses) a command. For instance, when the user is leaving and triggering the "`lock the door`" command, $\mathcal{A}$ can maliciously intercept the command leaving the door open.

4) **Command Spoofing (CS)**: $\mathcal{A}$ spoofs (i.e., creates) a fake command. For instance, $\mathcal{A}$ can trigger fake commands, e.g., to turn on the light while the user is sleeping or absent, creating a potential damage (device damage and/or electricity costs) to the user.

As Tab. 3 shows, some attacks such as "Door open while absent" may actually fall into multiple categories: i) EI: when the user is leaving it is automatically propagated the event "`user not at home`". $\mathcal{A}$ can intercept the event avoiding that the apartment door is locked. ii) CS: when the user is not at home, $\mathcal{A}$ can spoof the command "`open the door`", having physical access to the home. iii) CI: when the user leaves, $\mathcal{A}$ can intercept the command "`lock the door`", preventing it to be locked.

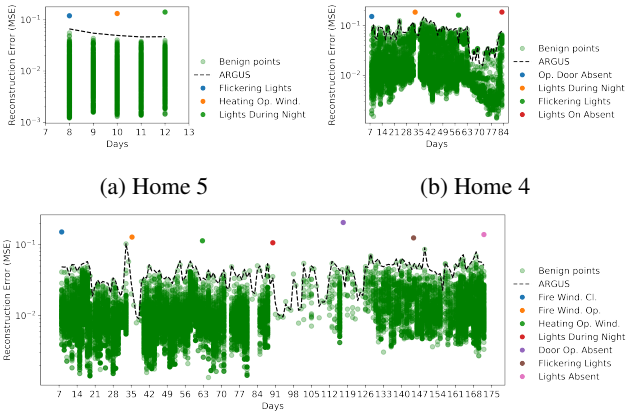
Table 4: Performance of ARGUS on Real-World Attacks, all values in percentage.

Attack	Dataset	Pr	Re	F1-Score
Door Open During Absence	Home 1	100.0	100.0	100.0
	Home 4	100.0	100.0	100.0
Lights On During Absence	Home 1	100.0	100.0	100.0
	Home 2	100.0	100.0	100.0
	Home 4	100.0	100.0	100.0
Movement During Absence	Home 2	100.0	100.0	100.0
	Home 4	100.0	100.0	100.0
Light Flickering	Home 1	100.0	100.0	100.0
	Home 2	100.0	100.0	100.0
	Home 3	100.0	100.0	100.0
	Home 4	100.0	100.0	100.0
	Home 5	100.0	100.0	100.0
Heating while Windows Open	Home 1	100.0	100.0	100.0
	Home 2	100.0	100.0	100.0
	Home 5	100.0	100.0	100.0
Lights On During Night	Home 1	100.0	100.0	100.0
	Home 2	100.0	100.0	100.0
	Home 3	100.0	100.0	100.0
	Home 4	100.0	100.0	100.0
	Home 5	100.0	100.0	100.0
Fake Fire Open Windows	Home 1	100.0	100.0	100.0
	Home 2	100.0	100.0	100.0
	Home 4	100.0	100.0	100.0
Fake Fire Closed Windows	Home 1	100.0	100.0	100.0
	Home 2	100.0	100.0	100.0
	Home 4	100.0	100.0	100.0

5.2.2 Internal Components

Context Modeling. The Context Modeling component predicts the reconstruction loss that is used by the Anomaly Score Classification component to discriminate benign events and attacks. Fig. 4 shows the predicted anomaly scores for benign events and the performed attacks in the setups Home 1, Home 2, and Home 5. As the figure shows, the Context Modeling component separates the attacks well from the benign events, allowing the threshold to effectively distinguish between both event types. In App. F we evaluate alternative design choices for the Context Modeling component.

As Subfig. 4c shows, 7 days of training data are sufficient for ARGUS to model the expected context, such that only very few FPs are predicted (FPR $\leq 0.03\%$) even for long periods (80 days) without any adaption or retraining.



(a) Home 5 (b) Home 4

(c) Home 1

Figure 4: Anomaly Scores for different real-world homes for benign behavior and attacks.

This demonstrates the effectiveness of the Context Modeling component for modeling the user's behavior.
To evaluate the amount of training data that ARGUS needs to effectively model the user's behavior, we performed multiple experiments with different duration for capturing the training data in the setup Home 1. As Fig. 5 shows, 7 days of training data are sufficient for ARGUS to achieve a F1-Score of 99.64% on 80 days of test data. This demonstrates that ARGUS is able to learn users' behavior without requiring a long training data collection phase.

Anomaly Score Classification. The Anomaly Score Classification component uses the dynamically determined threshold (cf. §4.3) for recognizing attack events based on their anomaly scores.

Tab. 5 compares the proposed threshold tuning approach with different alternative options. The threshold "Mean of max" realizes a static threshold, calculated as the average of the per-day maximal anomaly scores for the training data. However, it causes FPs, such that the precision is only approx. 37.5% and the threshold cannot adapt dynamically. Also, using the mean plus standard derivation of the previous day causes many FPs (FPR = 5.9%) and as well as using the maximal anomaly score of the previous day as threshold or (Pr=87.8%). We also evaluated different choices for the aging factor α. For $\alpha = 0.0$, resulting in using only the threshold candidate C_d, although no FPs were caused here, the threshold reacted too much on changing behavior, such that it was not able to detect all attacks (Re = 99%). On the other side, a too high value for α, resulting in a static threshold also failed to detect all attacks since the anomaly scores were higher than normal on the first day (Pr = 94.7%). In comparison, ARGUS, setting $\alpha = 0.2$, performed best, achieving Re = 100.0%, FPR = 0.0% and F1-Score = 100.0%. In App. D we evaluate the difference between the threshold and

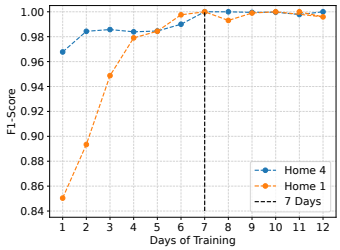


Figure 5: Evaluation of ARGUS depending on the duration of training data for the setup Home 1 and Home 4

threshold candidate in detail, in App. E we evaluate different choices for the α and security level β.

6 Security Considerations

The main goal of ARGUS is to detect contextual attacks and prevent adversary $\mathcal{A}$ from abusing the deployed IoT devices. As we demonstrated in §5, ARGUS effectively detects abnormal behavior. However, if adversary $\mathcal{A}$ is aware of the presence of ARGUS, it may attempt to manipulate the system in a way that it can avoid detection while still being able to successfully execute the attacks.

6.1 Avoidance by Context Manipulation

Since ARGUS detects malicious actions based on the context of the action taking place, $\mathcal{A}$ may attempt to spoof the contextual status updates of specific devices in a way that would make the actions falsely seem to happen in a legitimate context (e.g., $\mathcal{A}$ spoofs the window sensor state as being 'closed' whereas in reality the window is open and subsequently turns on the heating to waste energy). However, this would make the attack significantly more challenging for the adversary, since it would not be sufficient to compromise the control plane of the targeted IoT devices but it would be necessary to compromise and run code on specific devices in order to impersonate them or spoof context updates. As discussed in §3.2, we consider such attacks to be outside of the scope of ARGUS, since there already are numerous state-of-the-art approaches for detection of direct attacks against IoT devices that are orthogonal to ARGUS and can complement its detection capabilities with regard to such attacks.

6.2 Manipulating Model Training Data

If adversary $\mathcal{A}$ is present in the targeted network already during the initial training phase of the system, it can trivially execute any attacks and remain undetected. In accordance with similar work, we therefore make the assumption that the system is not compromised during the initial training of the system (cf. § 3.2). However, we also evaluated a setting in which the adversary manages to start a data poisoning attack when the initial training has not been entirely completed,

Table 5: Evaluation of Different Choices for the Classification Threshold for the setup Home 5, all values in percentage.

Threshold	FPR	Pr	Re	F1-Score
Mean of max	0.1	37.5	100.0	54.5
Mean+Std of prev. day	5.9	0.7	100.0	1.4
Max of prev. days	0.1	87.8	100.0	93.5
ARGUS ($\alpha = 0.0$)	0.0	99.0	99.0	99.9
ARGUS ($\alpha = 1.0$)	0.002	94.7	99.9	97.3
ARGUS	0.0	100.0	100.0	100.0

or, the attack is executed during a re-training phase of the detection model. We evaluated the impact of small amounts of attack data in the training datasets to examine the ability of ARGUS to detect attacks (cf. App. H), showing that ARGUS is resilient against the evaluated attack.

7 Discussion

We presented ARGUS that detects contextual attacks. We demonstrated in §5 that it effectively detects attacks without raising a significant number of false positives (FPR $\leq 0.03\%$) even in complex scenarios, such as homes with multiple inhabitants. In §2.4 we discussed different requirements that an effective intrusion detection system needs to fulfill. We showed in §4 that ARGUS fulfills our requirements with regard to fast detection (R1), cause identification (R2), and autonomous operation (R4). Our evaluation in §5.2 also showed that ARGUS generates a very low number of false alarms, thereby satisfying requirement R3.

7.1 Limitations of ARGUS

The focus of ARGUS is to recognize behavior that deviates from the normal behavior of the legitimate user. In §5 we demonstrated that ARGUS can also handle very complex scenarios like a home with multiple inhabitants. However, if the change in the normal behavior is too large to be handled by ARGUS, e.g., when additional persons move into the apartment, a retraining of the model must be initiated. This could be done automatically if the user labels raised alerts as false positives (e.g., using their smartphone). If the False-Positive-Rate is too high, a retraining can be initiated automatically. Another option would be to continuously adapt the model slightly, according to the monitored behavior. We leave determining suitable strategies for this to future work.

Another limitation is that, since ARGUS's definition of anomalous behavior depends on the behavior of the user during training time, actions that the user performed frequently during the training time might not be detected. This might be a problem if the user wants to use ARGUS, to unlearn bad habits, e.g., learning to turn of the heating when the window is opened. As this action is performed by the user, using ARGUS to change the user's behavior is out-of-scope of this paper. To handle those scenarios, ARGUS could be extended by a policy-based detection component. As discussed in §1, this policy-based detection is not suitable to detect attacks in general but would be able to detect a single, specific, previously defined situation (like an active thermostat while the windows are open).

Further, ARGUS focuses on observing the state of the system to recognize illegitimate states. Therefore, it can be seen as complementary to policy-based approaches, that, e.g., supervise the execution of critical automation rules in few, well-defined situations. An example here would be an automation rule that locks the door exactly 10 minutes after the user left the house. Since this behavior does not depend on other circumstances but only on the time since the user left the home, it can be easily supervised by a manually crafted policy or state-of-the-art policy-based approaches. In comparison, ARGUS focuses on complex situations. For example, turning on the light depends on many different contextual factors, e.g., the presence of the user, whether he is sleeping, etc.

Moreover, since ARGUS uses the observed behavior for modeling the expected behavior, ARGUS is limited to detecting abnormal situations but cannot supervise the behavior of IoT devices to handle these abnormal situations. An example would be an automation rule that automatically notifies the user, when a flood sensor notifies water (cf. work of Celik *et al.* [16]). Since the flooding situation is unlikely to occur in the training data, ARGUS cannot detect if the attacker suppresses the sending-notification action, as suppressing actions is outside of our adversary model (cf. §3.2). However, since as the flooding situation is already anomalous itself, ARGUS will notify the user about the flooding, such that the user can take effective counter measures himself.

7.2 False-Alerts

ARGUS only creates very few False-Positives (FPs) as it is 0.03% for Home 1, and 0% for all of the other homes (Home 2, Home 3, Home 4, Home 5). Thus, even for a high number of daily events (e.g., 3000), this would result only in a single FP per day. Considering the high number of notifications that users receive on their smartphones each day, e.g., socials and webservices as Google, one false alert per day is negligible.

8 Related Work

There is a large body of literature on IoT security with many diverse approaches, although not all are relevant for our paper. We categorized them as follows: i) Network Traffic Inspection, ii) Command Authentication, iii) Data Provenance, iv) Local Intrusion Detection, v) Sensor Validation, vi) Policies & Transition Graphs, vii) Contextual Anomaly Detection.

Approaches that inspect the network traffic [24, 36, 37] are not effective for detecting contextual attacks that compromise the control plane. Since here, the attacker activate the functions of the IoT devices over the regular control infrastructure, the network packets that are used for transmitting commands are indistinguishable from the traffic of benign commands.

Table 6: Comparison of Approaches for Contextual Anomaly Detection. The symbol indicates the presence (✓) or absence (✗) of the respective ability, while the color indicates whether the presence/absence is desired (green) or undesired (red).

Approach	Restricted to known Attacks	Bound to Policies	Considers Normal User Behavior	Handles unknown Devices	Infers Hidden Correllations	Handles Event Spoofing	Handles Event Interception	Handles Command Spoofing	Handles Command Interception
Amraoui *et al.* [10]	✗	✗	✓	✓	✗	✗	✗	✓	✗
HomeGuardian [21]	✓	✗	✓	✓	✓	✓	✓	✓	✓
HAWatcher [26]	✗	✓	✓	✗	✓	✓	✓	✓	✓
6thSense [41]	✗	✗	✓	✓	✗	✓	✓	✓	✓
Aegis [42]	✗	✗	✓	✓	✗	✓	✓	✓	✓
Tang *et al.* [43]	✓	✗	✓	✓	✓	✓	✓	✓	✓
ARGUS	✗	✗	✓	✓	✓	✓	✓	✓	✓

By analyzing the context of the invocation, i.e., states of other devices, ARGUS can detect contextual attacks.

Command Authentication approaches [16, 32, 44] focus on apps that are responsible for automating various tasks. Malicious apps could try to activate functions on devices which they are not supposed to control, e.g., an app that is responsible for measuring the humidity could try to unlock the door. To prevent such attacks, these approaches authenticate the source of a triggered command and check if the respective apps is actually allowed to control the targeted device. However, compared to ARGUS they cannot detect attacks, where the attacker uses the regular control infrastructure, e.g., use the cloud service of the smart-lock to unlock the door.

Local intrusion detection approaches need to be installed on each IoT device locally [12, 27, 40]. However, these approaches assume that the device manufacturer allows and supports installing software on the devices and that low-performance devices can execute additional software, while, ARGUS does not require any modification of the devices.

Data Provenance Approaches analyze the source of a command. One example for approaches falling into this category is ProvThings, being developed to explain attacks in the retrospective. By analyzing the flow of actions that were automatically triggered, the attack helps especially to explain complex attacks, where the attacker did not trigger the targeted action directly but exploited and concatenated multiple automation rules [45]. However, since this scheme focuses on explaining a performed attack rather than detecting the attack while it is being performed, ProvThings is complementary to ARGUS.

Hence, in the rest of this section, we focus on those proposals (iv-vii) that consider the whole *context* of the underlying IoT system in various ways such as validating the individual sensor values [9, 35, 49, 50], using policies or modeling the system states as nodes of a graph and analyzing the state transitions [15, 20, 25, 33, 47, 48], or focusing on detecting contextual anomalies [10, 26, 41–43].

8.1 Sensor Value Validation

Adkisson *et al.* use auto-encoders to detect anomalous values of sensors that measure the environment for plants [9]. Kotevska *et al.* compare the values of co-located sensors [35]. Yasaei *et al.* compare the values of different sensors where the measured values directly effect each other, e.g., the water temperature and the Nitrate concentration, to detect malfunctions of individual sensors [49]. Yin *et al.* analyze time series of the individual sensor values [50].

However, all these approaches consider only simple scenarios without human-caused randomness or are limited to detecting sensor malfunctions without being able to detect anomalous system states that occur in the real world. For example, if the door of an apartment is opened while the inhabitant is absent, then these approaches will not create an alert, as the co-located or correlated sensors consistently show that the door is opened. In comparison, ARGUS can detect that the door is opened in the wrong context (while the inhabitant is not at home) and inform the user.

8.2 Policies & Transition Graphs

Other approaches verify the system state against pre-defined policies [33, 47, 48]. However, these rules can easily be analyzed by the adversary and also create additional overhead for the user to define these policies. For example, for HomeEndorser [33] and Expat [47] the policies have to be defined manually, making the system unpractical. Yamauchi *et al.* monitor the users' behavior during the setup time and create an alert if the performed action was not invoked during the setup phase, in a situation where all sensors measured the exact values as they do at the moment [48]. Feng *et al.* automatically craft invariants from logfiles of industrial IoT systems [25], which are significantly less complex then smart-homes and can only learn linear relations between different devices, increasing the risk of wrong predictions. In comparison to these approaches, ARGUS is trained automatically from the data that is captured during the setup phase. The deep learning-based behavior modeling of ARGUS enables it to learn hidden/non-linear relationships between system states, e.g., it does not require all sensors to measure the same states as in the setup phase but learns which sensors can actually differ (e.g., temperature sensors) and which sensor values must be exactly the same (e.g., the presence of the inhabitant).

Homonit [51] and Soteria [15] both extract a Deterministic Finite Automaton (DFA) from the source code of mobile apps, where each node represents a system state. They create an alert if an invalid transition from one state to another state is taken. However, since these approaches use only the source code as data source, they consider only a theoretical legitimate behavior and not the actual user's behavior for recognizing contextual attacks. For example, the app might allow to turn on the light while the user is sleeping. However, if the user does not do this, then ARGUS can still detect this attack. DICE uses a Markov Chain to model the transitions of a system [20]. However, it models each system state as all sensor values that occurred in a time frame of 1 minute, such that it cannot distinguish, e.g., between turning on the light or the light-flickering attack that was discussed in §5.2.1.

8.3 Contextual Anomaly Detection

Table 6 shows on overview of state-of-the-art approaches for Contextual Anomaly Detection. It systematize them based on different criteria, 1) if the system can only recognize attacks that were known at setup time, 2), whether the system uses policies for recognition and is restricted to detect behavior that is allowed/forbidden by these policies, 3) whether the actual user behavior is considered or only expected behavior that is, e.g., extracted from app descriptions, 4) whether custom IoT devices, where no semantic information are available can be handled, 5) if also hidden relationships can be learned or if the system is restricted to recognizing state changes that have been exactly occurred in the training data, 6) if the system can handle attacks that spoof events (e.g., fake temperature sensor values), 7) if event interceptions are recognized (e.g., intercepting the user-coming-home event before the door is opened), 8) if spoofed commands can be recognized (e.g., if the light is turned on during the night), and 9) if command interceptions are recognized (e.g., the command to turn of the heating before opening the window).

HAWatcher uses semantic information, extracted from the app descriptions and source code, to match devices with the correlated attribute. For example, it determines that a sound sensor and a media player both are connected via the sound property. HAWatcher then generates correlation rules that need to be fulfilled [26]. This structure enables HAWatcher to enforce strict policies (cf. §7.1). However, requiring the presence of semantic information limits the applicability as such information are not always available. For instance, the temperature, humidity, and CO_2 sensor in our experimental setup were built by the respective inhabitants. Therefore, for these devices neither apps nor semantic information exists. In comparison, ARGUS uses only the actually occurred sensor values for learning the expected behavior, such that it is also effective without any semantic information.

Amraoui *et al.* train a One-class SVM for analyzing the triggered commands but do not consider the values of the sensors [10]. Thus, it cannot detect, e.g., the Movement During Absence attack.

Tang *et al.* provide an approach that trains binary classifiers on legitimate data as well as attack data to detect sensor failures [43]. Dai *et al.* use a Neural Network that is trained on attacks and benign behavior to distinguish between them [21]. However, by requiring attack training data these approaches are restricted to detecting known attacks, while ARGUS can detect arbitrary attacks.

6thSense [41] and Aegis [42] both consider the changes of the system states (e.g., sensor updates) as Markov Chains and require the probability for invalid state transitions, i.e., attacks, to be 0. However, the system then can only consider transitions that have occurred during the training phase and fails to learn hidden correlations/uncorrelations between sensor values, i.e., that some devices always change together or are unrelated, e.g., if they are located in different rooms. In comparison, the Behavior Modeling of ARGUS can learn relations, e.g., the independence of two sensors.

9 Conclusion

The number of devices being part of the Internet of Things is increasing rapidly, while at the same time these devices control more and more functions being part in our daily life. These factors make IoT devices an attractive goal for attacks that abuse these devices to cause financial damage or harm the users. We proposed ARGUS, the first dynamic system that can detect contextual attacks on connected device settings. It consists of a data collection component, monitoring devices even in a very heterogeneous landscape with devices from various manufacturers using different communication technologies and protocols. The designed Deep Neural Network in combination with the proposed dynamic threshold tuning scheme allows to distinguish between the expected user behavior and contextual attacks.

We extensively evaluated ARGUS for 5 different settings and showed that it effectively detects contextual attacks on IoT devices while raising only few false positives (FPR $\leq 0.03\%$), even in complex settings.

Acknowledgments

This work was funded in part by Intel as part of the Private AI center, HMWK within ATHENE project, and the European Union's Horizon 2020 research and innovation program under grant agreement No. 952697 (ASSURED).

References

[1] Apple home. https://www.apple.com/ios/home/. Accessed: 2022-01-25.

[2] Google assistant. https://assistant.google.com/smart-home/. Accessed: 2022-01-25.

[3] Hackers in hot water. pwning smart hot tubs, yes really. https://www.pentestpartners.com/security-blog/hackers-in-hot-water-pwning-smart-hot-tubs-yes-really/. Accessed 2022-01-29.

[4] Home assistant. https://www.home-assistant.io/. Accessed: 2022-01-25.

[5] Home security camera isn't secure. spotcam in the spotlight. https://www.pentestpartners.com/security-blog/home-security-camera-isnt-secure-spotcam-in-the-spotlight/. Accessed 2022-01-29.

[6] Smartthings. https://www.samsung.com/us/smartthings/. Accessed: 2022-01-25.

[7] Pytorch, 2022. https://pytorch.org.

[8] Single-family smart homes global market report 2022. https://www.reportlinker.com/p06193673/Single-Family-Smart-Homes-Global-Market-Report.html?utm_source=GNW, 2022. Accessed: 2022-01-30.

[9] Mary Adkisson, Jeffrey C Kimmell, Maanak Gupta, and Mahmoud Abdelsalam. Autoencoder-based anomaly detection in smart farming ecosystem. In *IEEE International Conference on Big Data (Big Data)*. IEEE, 2021.

[10] Noureddine Amraoui and Belhassen Zouari. An ml behavior-based security control for smart home systems. In *International Conference on Risks and Security of Internet and Systems*. Springer, 2020.

[11] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *USENIX Security*, 2017.

[12] Junaid Arshad, Muhammad Ajmal Azad, Muhammad Mahmoud Abdeltaif, and Khaled Salah. An intrusion detection framework for energy constrained iot devices. *Mechanical Systems and Signal Processing*, 136:106436, 2020.

[13] A. Barth, A. Datta, J.C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: framework and applications. In *S&P*. IEEE, 2006.

[14] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4), 1988.

[15] Z Berkay Celik, Patrick McDaniel, and Gang Tan. Soteria: Automated {IoT} safety and security analysis. In *USENIX Annual Technical Conference (USENIX ATC 18)*, 2018.

[16] Z. Berkay Celik, Gang Tan, and Patrick McDaniel. IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT. In *NDSS*, 2019.

[17] Yunqiang Chen, Xiang Sean Zhou, and Thomas S Huang. One-class svm for oldlearning in image retrieval. In *International Conference on Image Processing*, volume 1. IEEE, 2001.

[18] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. Autoencoder-based network anomaly detection. In *2018 Wireless Telecommunications Symposium (WTS)*. IEEE, 2018.

[19] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[20] Jiwon Choi, Hayoung Jeoung, Jihun Kim, Youngjoo Ko, Wonup Jung, Hanjun Kim, and Jong Kim. Detecting and identifying faulty iot devices in smart home with context extraction. In *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018.

[21] Xuan Dai, Jian Mao, Jiawei Li, Qixiao Lin, and Jianwei Liu. Homeguardian: Detecting anomaly events in smart home systems. *Wireless Communications and Mobile Computing*, 2022, 2022.

[22] Ryan Daws. Another IoT-based DDoS attack leaves Finnish properties without heating. https://iottechnews.com/news/2016/nov/08/another-iot-based-ddos-attack-leaves-finnish-properties-without-heating, 2016. Accessed: 2022-01-24.

[23] Wenbo Ding, Hongxin Hu, and Long Cheng. IOTSAFE: Enforcing safety and security policy with real iot physical interaction discovery. In *NDSS*, 2021.

[24] Yulin Fan, Yang Li, Mengqi Zhan, Huajun Cui, and Yan Zhang. Iotdefender: A federated transfer learning intrusion detection framework for 5g iot. In *IEEE International Conference on Big Data Science and Engineering (BigDataSE)*. IEEE, 2020.

[25] Cheng Feng, Venkata Reddy Palleti, Aditya Mathur, and Deeph Chana. A systematic framework to generate invariants for anomaly detection in industrial control systems. In *NDSS*, 2019.

[26] Chenglong Fu, Qiang Zeng, and Xiaojiang Du. HAWatcher:Semantics-Aware anomaly detection for appified smart homes. In *USENIX Security*, 2021.

[27] Tomer Golomb, Yisroel Mirsky, and Yuval Elovici. Ciota: Collaborative iot anomaly detection via blockchain. In *DISS@NDSS*, 2018.

[28] T. Gu, Z. Fang, A. Abhishek, H. Fu, P. Hu, and P. Mohapatra. IoTGaze: Iot security enforcement via wireless context analysis. In *IEEE INFOCOM - IEEE Conference on Computer Communications*, 2020.

[29] Stephen Herwig, Katura Harvey, George Hughey, Richard Roberts, and Dave Levin. Measurement and analysis of Hajime, a peer-to-peer IoT botnet. In *NDSS*, 2019.

[30] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[31] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. MIT Press, 2016.

[32] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Z. Morley Mao, and Atul Prakash. ContexIoT: Towards providing contextual integrity to appified IoT platforms. In *NDSS*, 2017.

[33] Kaushal Kafle, Kirti Jagtap, Mansoor Ahmed-Rengers, Trent Jaeger, and Adwait Nadkarni. Towards Practical Integrity in the Smart Home with HomeEndorser. *arXiv preprint arXiv:2109.05139*, 2021.

[34] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S Jensen. Outlier detection for time series with recurrent autoencoder ensembles. In *IJCAI*, 2019.

[35] Olivera Kotevska, Kalyan Perumalla, and Juan Lopez. Kensor: Coordinated intelligence from co-located sensors. In *IEEE International Conference on Big Data (Big Data)*. IEEE, 2019.

[36] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N Asokan, and Ahmad-Reza Sadeghi. DÏoT: A federated self-learning anomaly detection system for iot. In *ICDCS*. IEEE, 2019.

[37] TJ OConnor, Reham Mohamed, Markus Miettinen, William Enck, Bradley Reaves, and Ahmad-Reza Sadeghi. HomeSnitch: Behavior transparency and control for smart home iot devices. In *WISEC*. Association for Computing Machinery, 2019.

[38] Maire O'Neill et al. Insecurity by design: Today's iot device security problem. *Engineering*, 2(1):48–49, 2016.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python . *Journal of Machine Learning Research*, 2011.

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

[41] Amit Kumar Sikder, Hidayet Aksu, and A Selcuk Uluagac. {6thSense}: A context-aware sensor-based attack detector for smart devices. In *USENIX Security*, 2017.

[42] Amit Kumar Sikder, Leonardo Babun, Hidayet Aksu, and A Selcuk Uluagac. Aegis: A context-aware security framework for smart home systems. In *Annual Computer Security Applications Conference (ACSAC)*, 2019.

[43] Sihai Tang, Zhaochen Gu, Qing Yang, and Song Fu. Smart home iot anomaly detection based on ensemble model learning from heterogeneous data. In *IEEE International Conference on Big Data (Big Data)*. IEEE, 2019.

[44] Yuan Tian, Nan Zhang, Yueh-Hsun Lin, XiaoFeng Wang, Blase Ur, Xianzheng Guo, and Patrick Tague. SmartAuth: User-centered authorization for the internet of things. In *USENIX Security*, 2017.

[45] Qi Wang, Wajih Ul Hassan, Adam Bates, and Carl Gunter. Fear and logging in the internet of things. In *NDSS*, 2018.

[46] Jason Wise. Smart home statistics 2022: How many smart homes are there? https://earthweb.com/smart-home-statistics/, 2022. Accessed: 2022-01-30.

[47] Moosa Yahyazadeh, Proyash Podder, Endadul Hoque, and Omar Chowdhury. Expat: Expectation-based policy analysis and enforcement for appified smart-home platforms. In *ACM symposium on access control models and technologies*, 2019.

[48] Masaaki Yamauchi, Yuichi Ohsita, Masayuki Murata, Kensuke Ueda, and Yoshiaki Kato. Anomaly detection in smart home operation from user behaviors and home conditions. *IEEE Transactions on Consumer Electronics*, 66(2):183–192, 2020.

[49] Rozhin Yasaei, Felix Hernandez, and Mohammad Abdullah Al Faruque. Iot-cad: context-aware adaptive anomaly detection in iot systems through sensor association. In *IEEE/ACM ICCAD*. IEEE, 2020.

[50] Chunyong Yin, Sun Zhang, Jin Wang, and Neal N Xiong. Anomaly detection based on convolutional recurrent autoencoder for iot time series. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(1):112–122, 2020.

[51] Wei Zhang, Yan Meng, Yugeng Liu, Xiaokuan Zhang, Yinqian Zhang, and Haojin Zhu. Homonit: Monitoring smart home apps from encrypted traffic. In *CCS*, 2018.

[52] Chong Zhou and Randy C Paffenroth. Anomaly detection with robust deep autoencoders. In *ACM SIGKDD*, 2017.

A Devices

Table 7 shows for each setup in the collected dataset, a detailed list of the deployed IoT devices and measured values. In our experiments, we leveraged all values that were provided by the used home automation platform, covering the different categories of contextual features (cf. §2.2): i) Sensors/devices that measure ambient or temporal features (e.g., temperature, humidity, and luminosity), ii) user features (e.g., user presence and sleep confidence), and event features (e.g., states of the light bulbs, doors or windows).

B Runtime Performance

The real-time approach evaluates events without delay as soon as they are captured. Compared to the time the user needs to react manually, even an unlikely delay of multiple seconds would be negligible.
We performed additional experiments to evaluate ARGUS on a low-performance device (Raspberry Pi) without observing any delay. Further, we also restrict the main memory that ARGUS was allowed to use and observed that 330MB are sufficient for the implementation. It should be noted that the prototype of ARGUS was not optimized for runtime performance, such that by using, e.g., more efficient languages the runtime performance could be increased even further.

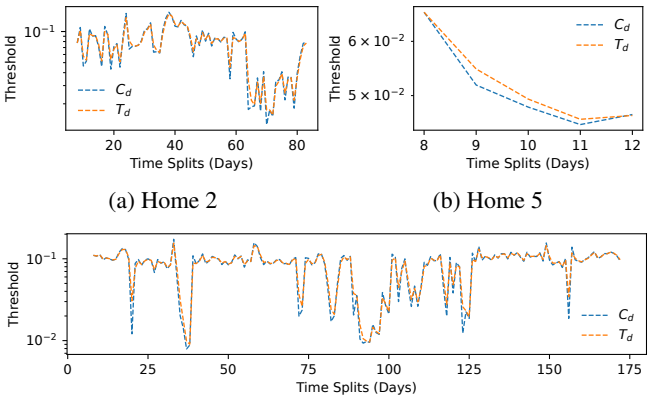
C Evaluation of Robustness of the Threshold

In the following, we investigate whether the injection of noise in devices' states would tamper ARGUS's threshold, therefore, making it more likely to miss attacks.
Considering 6 different levels of noise randomly sampled from a normal distribution, described by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{6}$$

Table 7: Deployed devices in the collected real-world IoT dataset. The deployment of a sensor/actor is indicated by ●, while the absence is indicated by ○.

Device	Home 1	Home 2	Home 3	Home 4	Home 5
Automation - All lights off	○	○	○	○	●
Automation - All lights on	○	○	○	○	●
Automation - Camera off when at home	○	○	○	●	○
Automation - Dinner lights	○	○	○	●	●
Automation - Dinner table light	○	○	○	○	●
Automation - Gaming mode	○	○	○	○	●
Automation - Heating boost off	○	○	○	○	●
Automation - Light off when no motion	○	○	○	●	●
Automation - Lights off in the evening	○	●	○	○	○
Automation - Lights off when too bright	○	●	○	●	○
Automation - Lights on in the morning	○	●	○	○	○
Automation - Lights on when motion detected	○	●	○	●	○
Automation - Piano Light	○	○	○	○	●
Automation - Sofa Lamp	○	○	○	○	●
Automation - Studio Light off	○	○	○	○	●
Automation - Studio Light on when motion	○	○	○	○	●
Automation: Camera on when user leave	○	○	○	●	○
Camera Status Sensor	○	●	●	○	○
Climate - Control access point 1	○	○	○	○	●
CO_2 Sensor Status	●	○	○	○	○
CO_2 Sensor	●	○	○	○	○
Control Access Room 1 Sensor	○	○	○	●	○
Door Sensor	●	●	●	●	●
Floor lamp	○	○	○	○	●
Heating - heater valve	○	○	○	○	●
Heating Temperature Sensor	●	●	○	●	○
Homematic - Radiator Thermostat Temperature Sensor	○	○	●	○	○
Humidity Sensor	●	●	○	●	●
IKEA Tradfri Roller Blind Sensor	●	○	○	○	○
IP Camera - Light Level	○	○	○	○	●
IP Camera - Motion	○	○	○	○	●
IP Camera - Motion Active	○	○	○	○	●
IP Camera - Pressure	○	○	○	○	●
IP Camera - Sound	○	○	○	○	●
Lamp consumption	○	○	○	○	●
Lamp consumption (daily)	○	○	○	○	●
Lamp consumption (total)	○	○	○	○	●
Lamp current	○	○	○	○	●
Lamp voltage	○	○	○	○	●
Light - Ceiling	●	●	●	●	●
Light - Desk Lamp	●	●	●	●	●
Light - Living Room	○	○	●	○	○
Philips Hue - Light Level Sensor 1	○	●	●	●	●
Philips Hue - Light Level Sensor 2	○	○	●	○	○
Philips Hue - Motion Sensor 2	○	○	●	○	○
Philips Hue - Temperature Sensor 1	○	●	●	●	●
Philips Hue - Temperature Sensor 2	○	○	●	○	○
Philips Hue - White Lamp 2	○	○	●	○	○
Philips Hue - White Lamp 3	○	○	●	○	○
Philips Hue - Motion Sensor 1	○	○	●	●	●
Piano lamp	○	○	○	○	●
Radiator Thermostat Sensor	○	○	○	●	●
Smartphone - Battery Life	○	●	○	○	○
Smartphone - Charging	○	○	○	○	●
Smartphone - Charging Sensor	●	○	○	○	○
Smartphone - Connected to WLAN	○	●	○	●	○
Smartphone - Detected Activity	●	●	○	○	●
Smartphone - Light Sensor	○	●	○	○	○
Smartphone - Locked	○	○	○	○	●
Smartphone - Phone Status	○	●	○	○	○
Smartphone - Sleep Confidence	●	●	○	●	○
Smartphone - Sleep Segment	○	○	○	●	○
Smartphone - Tracker	●	●	○	●	○
Studio lamp	○	○	○	○	●
Sun Sensor	●	●	●	●	●
Temperature Sensor (ESP)	●	●	○	●	●
User Presence	●	●	○	●	○
Weather - Home Location	●	●	●	●	●
Weather - Town	●	○	●	●	○
Window Sensor	●	●	●	●	○



(a) Home 2 (b) Home 5

(c) Home 1

Figure 6: Thresholds values comparison between Threshold Candidate (C_d) and ARGUS Threshold (T_d) for multiple real-world homes.

σ	Alerts %	Not affecting %	Affecting no alerts %
1	0.002	99.674	0.323
2	0.031	99.485	0.489
3	0.200	98.883	0.916
4	0.987	96.104	2.909
5	4.637	86.597	8.766
6	15.511	68.696	15.793

Table 8: Noise injection in events chain analysis on Home 1 dataset. Each entry counts 652,720×18 events evaluated. The values shown are the mean of all the devices attacked, for all the days considered per each level of noise considered (i.e., 18×6. In total are evaluated 70 493 760 events).

where μ is the mean and σ is the standard deviation, we attack all the devices, one per run, attacking its states with an amount of noise sampled from:[2]

$$f(x|\mu = 1, \sigma = i) \; \forall \; i \; \in \{1..6\} \tag{7}$$

considering all the days available for testing. For understanding if the attack is successful, and with which rate, we measure the number of alerts raised and how many times the benign maximum and minimum are exceeded (i.e., if the threshold computation would be affected by the injected noise).

As we can observe from Table 8, in order to affect the threshold, even with just a low probability, $\mathcal{A}$ has to inject a high amount of noise ($\sigma \in \{4,5,6\}$). However, if the level of noise gets severe the risk of raising an alert will raise up to 15.511%. Since $\mathcal{A}$ is assumed to be unable to compromise the ARGUS system (cf. §3.2) it cannot check in advance if a noised event will have no impact, raise an alert, or affect the threshold without raising an alert. Therefore, in order to damage the detection of ARGUS, it has to create a high number of attack events. Further, since the impact of a single day on the threshold is limited due to the aging factor α, $\mathcal{A}$ has to create many fake events over many days to damage ARGUS. However, this will also cause a high number of alerts, such that the user will become suspicious and detect the attack.

[2] We consider the mean of 100 samples sampled from the distribution.

D Comparison of Threshold and Threshold Candidate

To evaluate the impact of the momentum on the threshold in more detail, we measured the threshold candidate C_d as well as the resulting threshold T_d. As Fig. 6 shows, C_d fluctuates significantly, depending on the measured context of each day. However, if C_d would be used as classification boundary, these fluctuations might cause FNs or FPs on the following day. For example, when the user behavior for one day d differs from the usual behavior, C_d, which is used on the following day d+1 as classification boundary, is very high (as in Subfig. 6c). This would cause a risk that the anomaly scores of attacks on day d+1 are lower than C_d. On the other side, if the behavior of the user on day d is very similar to the expected behavior, C_d might be too low, causing FPs. In comparison, the momentum used by T_d smooths outliers and prevents that exceptional high or low anomaly scores on the previous day, e.g., caused by slightly differing user behavior, affect the system's performance.

E Ablation Study on α and β

Furthermore, we extensively evaluated the α and β parameters selection through 400 experiments. The results are summarized in Figure 7, showing how the selection of $\alpha = 0.2$ and $\beta = 0.2$ corresponds to an optimal choice.

F Alternatives for AE

We performed multiple experiments to compare the used DNN architecture (cf. §4.2.2 to other machine learning (ML) and deep learning algorithms, the results are shown in Tab. 9. As classical ML algorithm, we used a One-Class Support Vector Machine (One-Class SVM) [17]. The second approach uses also an Auto-Encoder (AE), which is also used by ARGUS. However, the evaluated AE model uses normal, linear layers, while ARGUS uses GRU layers to consider better the temporal order of the individual events. All approaches were trained using the same datasets and used to predict anomaly scores. However, to prevent any bias in favor of ARGUS, we opted for each of the alternatives a threshold that maximizes the F1-Score on the test data, while for ARGUS we used the threshold that we discussed in §4.3. However, even with this advantage, Re is significantly lower for the One-Class SVM and the AE

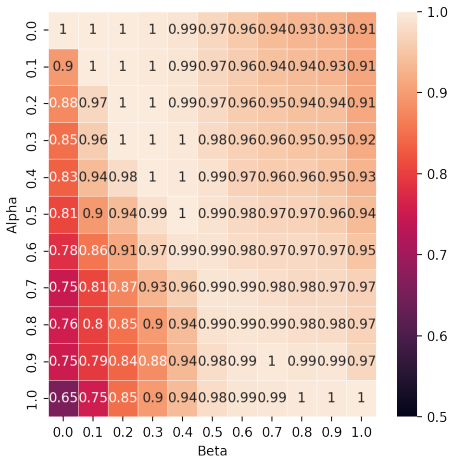


Figure 7: α and β parameters selection of ARGUS for the setup Home 4. For the sake of saving space, α and β are represented in the heatmap with a step of 0.1, the actual evaluation counts a step size of 0.05.

Table 9: Evaluation of alternative choices for the Machine Learning (ML) algorithm for modeling the expected behavior of Home 5, all values in percentage.

ML Algorithm	FPR	Pr	Re	F1-Score
One-Class SVM	0.0	96.2	52.9	68.2
AE without GRU	4.9	98.0	87.1	92.2
ARGUS	0.0	100.0	100.0	100.0

without the GRU layers, indicating that they miss many attacks. On the other side, especially the AE without the GRU layers also misclassifies many benign events (FPR = 4.9%) and the OneClassSVM misses 25 benign events (FPR = 0.016%), such that their F1-Scores are only 92.2% and 68.2%. In comparison, ARGUS detects all attacks in this setup (FPR = 0.0%) and the F1-Score is 100.0%.

G Evaluation Metrics

To evaluate the performance of the trained model we use common performance metrics such as False-Positive-Rate (FPR), Precision (Pr), Recall (Re), and F1-Score. For calculating these metrics, we count the number of benign events that are correctly classified (TN) or misclassified (FN) as well as the number of attacks that are correctly recognized (TP) or not recognized (FN). We define the performance metrics as follows:

False-Positive-Rate (FPR) indicates the risk to misclassify benign events. It is given by:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{8}$$

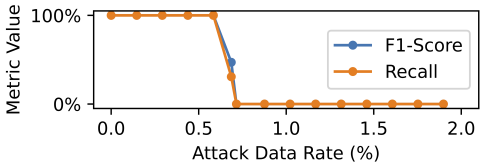


Figure 8: Evaluation of ARGUS depending on the amount of poisoned data in the training set of Home 5

Precision (Pr) indicates the probability that an event that is recognized as anomaly is actually an attack. It is given by:

$$\text{Pr} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{9}$$

Recall (Re) indicates the effectiveness of an approach to detect attacks. It is given by:

$$\text{Re} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{10}$$

F1-Score balances Pr and Re. It is calculated as:

$$\text{F1-Score} = 2 \cdot \frac{\text{Pr} \cdot \text{Re}}{\text{Pr} + \text{Re}} = \frac{\text{TP}}{\text{TP} + 1/2(\text{FP} \cdot \text{FN})} \tag{11}$$

H Robustness of ARGUS against Data Poisoning

We assume that the IoT system is not compromised during the data collection phase (cf. security assumptions in §3.2). In the following experiment, we evaluate the robustness of ARGUS if this assumption is violated, considering the data poisoning through the injection of attacks during the training process.

To do so, we conducted an experiment using the setup Home 5 and injected different numbers of events that are part of the Light-Flickering attack into the benign training data and evaluated the resulting model. In particular, we considered this attack as it was the attack with the lowest reconstruction scores (i.e., the closest to benign data and so the worst case possible).

As Fig. 8 shows, ARGUS demonstrates robustness characteristics against this kind of data poisoning while the overall poisoned data is lower than 0.6% of the total number of events. Therefore, $\mathcal{A}$ would need to manually perform this attack 42 times, until ARGUS cannot detect the Light Flickering anymore.

I Computational Setup

For the evaluation, the events were collected offline. All event traces were evaluated on a server running Debian 10, with 1 TB main memory, 64 physical cores, provided by an AMD EPYC 7742 processor, and 4 NVIDIA Quadro RTX 8000. For the machine learning experiments, we leverage the Scikit-Learn library [39] and Pytorch [7] for implementing the neural networks.