IoTGaze: IoT Security Enforcement via Wireless Context Analysis

Tianbo Gu*, Zheng Fang*, Allaukik Abhishek†, Hao Fu*, Pengfei Hu*, Prasant Mohapatra*
* Department of Computer Science, University of California, Davis, CA, USA
† ARM Research, Austin, TX, USA
Email: {tbgu, zkfang, haofu, pfhu, pmohapatra}@ucdavis.edu, Allaukik.Abhishek@arm.com

Abstract*—Internet of Things (IoT) has become the most promising technology for service automation, monitoring, and interconnection, etc. However, the security and privacy issues caused by IoT arouse concerns. Recent research focuses on addressing security issues by looking inside platform and apps. In this work, we creatively change the angle to consider security problems from a wireless context perspective. We propose a novel framework called IoTGaze , which can discover potential anomalies and vulnerabilities in the IoT system via wireless traffic analysis. By sniffing the encrypted wireless traffic, IoTGaze can automatically identify the sequential interaction of events between apps and devices. We discover the temporal event dependencies and generate the *Wireless Context* for the IoT system. Meanwhile, we extract the *IoT Context*, which reflects user's expectation, from IoT apps' descriptions and user interfaces. If the wireless context does not match the expected IoT context, IoTGaze reports an anomaly. Furthermore, IoTGaze can discover the vulnerabilities caused by the inter-app interaction via hidden channels, such as temperature and illuminance. We provide a proof-of-concept implementation and evaluation of our framework on the Samsung SmartThings platform. The evaluation shows that IoTGaze can effectively discover anomalies and vulnerabilities, thereby greatly enhancing the security of IoT systems.

***Index Terms*—Internet of Things, Anomaly Detection, IoT Security, Natural Language Processing, Wireless Context.**

I. Introduction

The rapid development of the Internet of Things (IoT) has an increasingly bigger impact on how we live and work. IoT technology enables interconnection, service automation, and other convenience in a variety of application scenarios, such as smart home, smart factory, and smart city, etc. By 2022, the number of connected IoT devices will reach to 29 billion [1]. The market value of IoT will reach to $1.2 trillion in 2022 with a compound annual growth rate of 13.6% starting from 2017 according to the IDC prediction [2]. To increase their market share, different companies develop their IoT platforms for third-party developers to build apps to realize service provision automation. The popular IoT program platforms include Samsung's SmartThings [3], Apples' HomeKit [4] and Google Home [5].

Despite the exploding devices and fast growth of platforms of IoT, the security and privacy solution is not keeping the pace. Emerging vulnerabilities and attacks in IoT have brought tremendous loss. Within 20 hours, 65,000 IoT devices were rapidly infected and utilized to launch Mira attacks leading to internet outage [6]. By exploiting a major bug in the implementation of the *ZigBee* light link protocol, the attacker can use one single malicious bulb to turn off all the city lights [7]. Most critical security and privacy threats come from the IoT platforms and their affiliated apps. For instance, despite the Samsung SmartThings platform has a capability separation model, the apps can still request the capabilities that they do not need. The platform lacks effective means to audit the requests. The authors [8] found that 55% of SmartApp did not use all the rights to device operations that their requested capabilities implied, and 42% of SmartApps were granted capabilities that were not explicitly requested or used. Once gaining access to the capabilities, the malicious apps may not follow the user expectation and their app descriptions, resulting in serious security issues.

To relieve the security and privacy threats, the researchers propose solutions from different perspectives. By embedding extra code, FlowFence [9] and Soteria [10] can monitor the data flows and related control flows to prevent all the implicit flows from IoT apps via static program analysis. ContextIoT [11] uses the runtime logging to extract the essential context for building a context-based permission system. SmarthAuth [12] collects the security-relevant context information from analyzing IoT apps' source code, annotations, and descriptions. IoTGuard [13] dynamically collects the apps' information to enforce safety and security policies. However, these approaches require good knowledge about the program framework and app code. They have to modify the apps' source code or patch the apps and platforms to realize the discovery and prevention of threats. As can be seen, most existing solutions focus on the program analysis for platforms and apps. Then we come up with a question: *Can we open a new path to enhance the defense of IoT security and privacy?*

In this work, we look outside the IoT platforms and apps, and rethink the IoT security and privacy problems from the wireless perspective, and propose a new concept `Wireless Context` in IoT. Distinct from the program-based context, the IoT *Wireless Context* is inferred from the wireless communication traffic. We propose and implement a novel IoT security enforcement framework called IoTGaze that can detect potential anomalies and vulnerabilities in the IoT system. First, IoTGaze extracts the wireless packet features to correlate the communication traffic with the interaction of events between apps and devices. IoTGaze constantly sniffs the encrypted wireless traffic and generates the interaction event sequence. Second, IoTGaze discovers the temporal event dependencies and builds the *Wireless Context* for IoT system. Third, IoTGaze extracts the actual user expected *IoT Context* from IoT apps' descriptions and user interfaces (UI). By comparing the detected *Wireless Context* with *IoT Context*, IoTGaze can discover the anomaly in current IoT system. Lastly, by exploring the wireless event dependencies, IoTGaze is able to discover the unknown vulnerabilities that are caused by the inter-app interaction chain via hidden channels, such as light, temperature, humidity, etc., and can be exploited by the attacker to launch attacks against IoT system.

Contributions: The contributions of our work are:

- Distinct from the existing solutions, we open a new path to rethink the IoT platform and app security issues from the `Wireless Context` perspective and propose a novel IoT anomaly and vulnerability detection framework called IoTGaze .
- We design a fingerprinting approach to detect the IoT events and generate the sequence of the events via analyzing the wireless packets. We also propose an effective mechanism to discover the temporal events dependencies and produce the event transition graph that represents the *Wireless Context* in IoT.
- We propose an approach that can extract the user expected `IoT Context` from apps' descriptions and UI using natural language processing (NLP). An algorithm is designed to detect the anomaly based on the comparison between the *IoT Context* and *Wireless Context*.
- By exploring *Wireless Context*, IoTGaze can discover the hidden vulnerabilities that are caused by the inter-app interaction, which is ignored by most exiting IoT security solutions.
- We prototype a proof-of-concept framework on the Samsung SmartThing platform, including 183 apps. The extensive evaluations show that our approach can achieve nearly 98% accuracy of anomaly detection. We also discover and provide a complete list of hidden vulnerabilities in the IoT system detected by IoTGaze .

II. Threat Model

In this paper, we consider the security and privacy problems on the typical IoT interaction chain: devices, apps, and IoT platform. Based on the program framework, the developers write apps that request the capabilities access privilege from devices, and then control the devices to implement service automation. For instance, the description of one app is *"Turn on the indoor surveillance if the householder leaves home, otherwise turn off the indoor surveillance"*. The related IoT devices are presence sensor and surveillance camera. The security and privacy issues for the IoT system we want to detect are: **(a)** `App misbehavior`. For instance, when the household is at home, the app should turn off the surveillance to prevent privacy leakage. But the app may not turn off the surveillance and still monitor the activities of the household and uploads the data to somewhere else. **(b)** `Event spoofing`. The attacker may spoof a *"presence.not_present"* command to the IoT hub and the hub turns off the surveillance. Then the intruder could break into the house. **(c)** `Over-privilege`. The app may request irrelevant capabilities from the platform, such as the lock control privilege. Then the app may unlock the door when the household leaves home, which triggers serious security issues. **(d)** `Device failure`. The hardware flaws and software bugs may cause device failure. The attacker can also launch an attack to make the device (e.g., surveillance camera) unresponsive. **(e)** `Hidden vulnerabilities`. This type of vulnerabilities is caused by some hidden channels that multiple apps interact with simultaneously. Consider the scenario where one heater control app can automatically turn on the heater in winter after 8:00 PM, and another app opens the window automatically if the room temperature is higher than 90°. The indoor temperature is the physical channel, and the attacker can spoof a command event to let the heater keep working, leading to an increase of the temperature, and finally open the window and break in. We design a novel anomaly and vulnerability detection framework called IoTGaze that addresses the above security and privacy threats in IoT system.

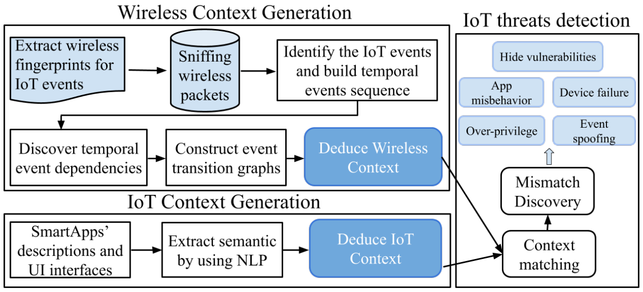

Wireless Context Generation
Extract wireless fingerprints for IoT events
Sniffing wireless packets
Identify the IoT events and build temporal events sequence
Discover temporal event dependencies
Construct event transition graphs
Deduce Wireless Context
IoT Context Generation
SmartApps' descriptions and UI interfaces
Extract semantic by using NLP
Deduce IoT Context
IoT threats detection
Hide vulnerabilities
App misbehavior
Device failure
Over-privilege
Event spoofing
Mismatch Discovery
Context matching

Fig. 1: Overview of IoTGaze

III. System overview

In this section, we provide an overview of IoTGaze , and describe key components and workflow, as shown in Fig. 1.

Wireless Context Generation. The challenge here is how to use the sniffed raw wireless packets to generate the IoT `Wireless Context`. We decompose this problem into three subtasks: (1) *How to correlate the wireless traffic with the IoT interaction events and generate the fingerprints for the events?* We utilize limited features extracted from the encrypted wireless traffic and generate effective fingerprints to detect IoT events. (2) *How to use the sequential packets to generate the sequential IoT events?* What we sniff is the wireless packet sequence, but for vulnerability detection we should work on events. Thus, we design an approach to automatically segment the packet sequence and generate the temporal event sequence. (3) *How to discover the temporal event dependencies and generate the wireless context?* We design an event dependency discovery method that accurately extracts the event dependencies and their causal relationship for building the wireless context graph.

IoT Context Generation. The `IoT Context` we define here is the event interaction chain between smart apps (which run in the cloud and interact with devices via the IoT gateway such as SmartThings Hub) and devices. The IoT context is the user expected app behaviors, which may not be the real execution behaviors of apps. The malicious apps may deceitfully inform users about their functionalities but surreptitiously execute some malicious activities. To accurately extract the IoT context, we analyze the apps' descriptions and UIs that are directly exposed to users and usually cannot deceit users compared with program code. We extract the IoT context from app description and UI using NLP techniques and build the corresponding event transition graph that represents the app work logic expected by the user.

Anomaly and Vulnerability Detection. The IoT context represents the IoT automation services expected by the user, while the wireless context reveals what practical automation services are happening. Each context is expressed by a set of event transition graphs. We propose an approach to discover the mismatch and anomaly by comparing the event transition graphs under different contexts. By further analysis, we can discover the hidden vulnerabilities that are caused by the inter-app interaction and can be used by an attacker to launch attacks. Then we can prevent the attacks before they happen. Next, we describe these components of IoTGaze in detail in the following section.

IV. Wireless Context Generation

We design and deploy a third-party *guardian* who *gazes* at the wireless communication traffic and detect potential anomalies and vulnerabilities. The *guardian* sniffs the encrypted wireless packets generated by the IoT activities and record the packets sequence $P = \{p_1, p_2, ..., p_i, ..., p_n\}$. Our goal here is to analyze the packet sequence and generate the wireless context. The wireless context is represented by a set of event transition graphs. In this section, we explain the procedures of wireless context generation in detail.

A. IoT Event Fingerprinting

Before generating the event sequence, we need to correlate the wireless traffic with the IoT events. We design the fingerprints to identify the IoT events. To realize service automation, event-driven smart apps receive data from various sensors (such as motion sensor, temperature sensor, and contact sensor) and issue commands to one or more actuators (e.g., smart bulb, smart power outlet, and smart lock, etc.) via the local IoT hub as the intermediary. We define IoT events as the activities that IoT hub interacts with sensors and actuators via wireless communication.

We use a packet sequence $P_{e_i} = \{p_1, p_2, ..., p_i, ..., p_{N_{e_i}}\}$ to represent the sniffed traffic for a unique event e_i. We extract the features from the packets' attributes except the encrypted data content to fingerprint the event. We list the features as following:(1) *Packet size*. A packet size could vary depending on what it transmits for which event. (2) *Packet direction*. A packet could be sent from the hub to a device or the opposite way. (3) *Packet interval*. The shorter packet interval signifies the higher transmission rate. Due to the difference of software and hardware, IoT devices may have distinct transmission rate, burst rate, response latency, and throughput, leading to varying packet interval. (4) *Packet layer*. Packets may be transmitted in different layers for a specific protocol. The above are common features across various wireless communication protocols, such as WiFi, Zigbee, Z-Wave, and Bluetooth lower Energy (BLE). Each protocol may have additional features. For example, The IP-based communication protocol may have features like IP source/destination address and source/destination port. By using the features set, we can generate the following fingerprint for a unique IoT event:

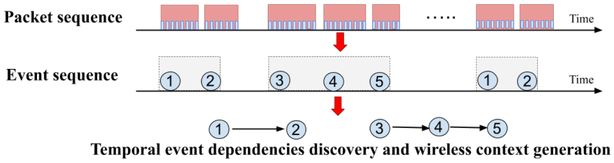

Packet sequence
.....
Time
Event sequence
1
2
3
4
5
1
2
Time
1
2
3
4
5
Temporal event dependencies discovery and wireless context generation

Fig. 2: Procedures of wireless context generation.

$$F_{e_i} = \begin{array}{c} \begin{matrix} p_1 & \cdots & \cdots & p_{N_{e_i}} \end{matrix} \\ \begin{pmatrix} f_{1,1} & f_{2,1} & \cdots & f_{N_{e_i},1} \\ f_{1,2} & f_{2,2} & \cdots & f_{N_{e_i},2} \\ \vdots & \vdots & \ddots & \vdots \\ f_{1,\mathcal{M}} & f_{2,\mathcal{M}} & \cdots & f_{N_{e_i},\mathcal{M}} \end{pmatrix} \end{array}$$

where N_{e_i} denotes the number of packets transmitted for event e_i, and $\mathcal{M}$ denotes the number of features we extract for a specific communication protocol.

We collect and create the fingerprints data set for each event, and use the Random Forest supervised machine learning model as the classifier $\mathcal{C}$. The value of N_{e_i} varies depending on the event e_i. In order to feed the fingerprints matrix F_{e_i} into the same machine learning model, we fix the number of packets to $\mathcal{N}$ and pad the matrix with zero if N_{e_i} is less than $\mathcal{N}$. The optimal value of $\mathcal{N}$ will be discussed and selected in the later evaluation section. Then we use the classifier $\mathcal{C}$ to classify the new, unlabeled fingerprints and identify the events.

B. Sequential Events Generation

We analyze the sniffed packets sequence:

$$P = \{(p_1, t_1), (p_2, t_2), ..., (p_i, t_i), ..., (p_n, t_n)\}, \qquad (1)$$

and identify the events using the generated fingerprints. Considering the packets for an event are sent within a short time. We use a sliding window with a maximum $\mathcal{N}$ packets within a fixed time interval of $\mathcal{T}$, and produce the matrix F. Then we feed the matrix F to the classifier $\mathcal{C}$ and output the probabilities for each event. If the maximum probability value predicted from event e_j is larger than the predefined classification threshold θ, then we think the packet sequence is created by event e_j. Otherwise, we will go to the next sliding window and continue to make the identification. The default step size is one, and the step size changes to the number of packets from event e_j once event e_j is detected.

C. Temporal Event Dependencies Detection

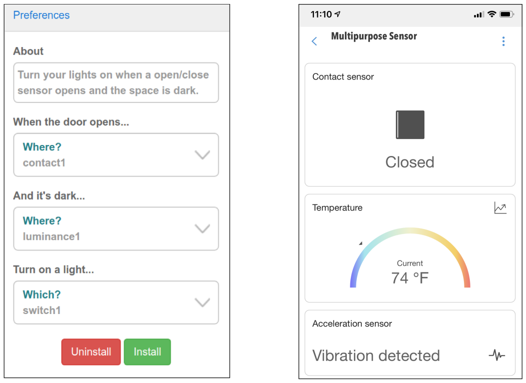
After identifying individual wireless events, we can construct the event stream $E = \langle(e_1, t_1), (e_2, t_2), ..., (e_n, t_n)\rangle$, where e_i $(i = 1, \ldots, n)$ is the wireless event happening at time t_i. Notice that e_i and e_j $(i \neq j)$ can be the same event happening at different time. A temporal event dependency means a set of events occur together with a chronological pattern. If event type a and b have a temporal dependency, then the time interval between them should follow a normal distribution $\mathcal{N}(\mu(a,b),\, \sigma^2)$ with σ being approximately equal to the standard deviation of the network delay. Thus, even though the expectation μ depends on the particular event type, the standard deviation is independent of event types. To determine if a and b are temporally dependent, we can collect all the samples of time interval between a and b from the event stream, and compute the sample standard deviation $\sigma(a,b)$ and compare it with the threshold τ (τ is a predefined parameter which is slightly larger than the standard deviation of network delay). If $\sigma(a,b) < \tau$, then we conclude a and b are temporally dependent, and vice versa.

Once we have identified all the pairs of events that are temporally dependent, we reconstruct the dependency sequence by concatenating these pairs. Formally, if $[a,b]$ and $[b,c]$ are dependent pairs, and $\mu(a,c) = \mu(a,b) + \mu(b,c)$, then we can get a dependency sequence $[a,b,c]$. Following such procedure, we iteratively check and concatenate sequences. In addition, we find that even if there is a dependency sequence $[a,b,c,d]$, $[c,d]$ itself could be a dependency sequence. We further discover these "subsequences of dependency sequences" using the number of occurrence in the input event stream. For example, if $[a,b,c,d]$ occurs 100 times, $[b,c,d]$ occurs 100 times, but $[c,d]$ occurs 150 times, then we know that $[b,c,d]$ is not a dependency sequence (since it is just a part of the dependency sequence $[a,b,c,d]$), but $[c,d]$ is a dependency sequence by itself and it occurs 50 times.

Generating Wireless Context. After discovering the event dependencies, we can build the event transition graph for each event dependency. As shown in Fig. 2, the event transition graph ① → ② represents a certain wireless context, such as *"If detecting a human presence, open the surveillance camera."*. The wireless context is extracted from the wireless traffic and can reflect the real activities of the currently installed apps. However, the wireless may not the expected IoT context from the user. We introduce the approach to extract the IoT context in the following section. If the wireless context violates the IoT context expected by the user, then it indicates potential anomalies in the current IoT system.

V. IoT CONTEXT GENERATION

In this section, we explain how to collect the IoT context that is the expected automation services from users. Due to the existing of malicious apps, the activities of smart apps cannot represent the actual IoT context. Some existing work conducts the static and dynamical analysis of the apps' code and checks if the actions of the program match what the apps describe. Instead of analyzing the apps' source code, we exploit the apps' description and UI that the apps usually do not tamper or spoof. Fig. 3(a) shows the installation interface of one SmartApp *Brighten-Dark-Places*. Based on the app description, the user chooses to install the app or not. Meanwhile, the needed capabilities are requested by the app, and the user needs to select the devices that can provide the capabilities. As we can see, the content in the installation interface is directly exposed to the user and tells the user what the app plans to do and is not usually tampered. If the user chooses to install the app, that means the app's description can reflect the user truly expected app service. If we know all the smart apps installed by the user, we can build the IoT context based on these apps' description and UI. Now, we introduce the IoT context generation approach and implement it on Samsung SmartThings platform [3].



About

Turn your lights on when a open/close sensor opens and the space is dark.

When the door opens...

Where?
contact1

And it's dark...

Where?
luminance1

Turn on a light...

Which?
switch1

Contact sensor

Closed

Temperature

Current
74 °F

Acceleration sensor

Vibration detected

Fig. 3: (a) The installation interface of the SmartApp *Brighten-Dark-Places* in Samsung SmartThings platform. (b) The capabilities that one multipurpose sensor has for the Samsung SmartThings platform.

A. App Description Analysis

The research work [14], [15] has revealed that most IoT applications following the "If-This-Then-That" (IFTTT) programming paradigm, which can also be reflected by their apps' description. The first step for analyzing apps' behaviors is to obtain the causal relationship from the description. One effective method is to identify the conditional and main clause from the description. The conditional clause involves some sensors' state change (e.g., the camera recognizes someone's face), and the main clause involves some devices' actions (e.g., unlock the door). Then, we can extract the related devices and their actions from the noun phrase and the verb phrase, respectively.

We use Stanford parser [16] to analyze the sentence structure of the app descriptions. To segment the description sentence into clauses, we build the constituency parse tree and split the sentence by label **S** (*Simple declarative clause*) or **SBAR** (*Clause introduced by a subordinating conjunction*). As shown in Fig. 4, the extracted three clauses are: "Turn on your lights", "a open/close sensor opens", and "the space is dark". The subordinating conjunction "when" signifies the causal relationship of the clauses via identifying the trigger and action. We analyze the dependency parse tree in Fig. 5 and extract the noun phrase and verb phrase from each clause. Considering the first clause as an example, "lights" is the accusative object of the verb "Turn" and this dependency is represented as *dobj*("Turn", "lights"). But the extracted

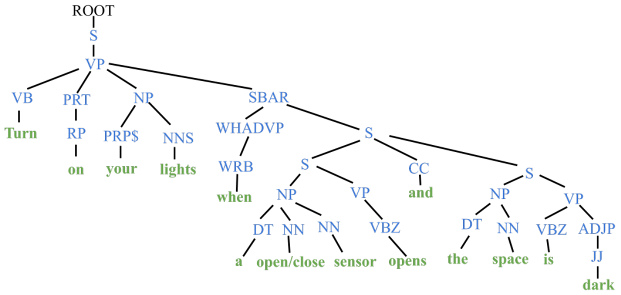

ROOT
S
VP
VB PRT NP SBAR
Turn RP PRP$ NNS WHADVP S
on your lights WRB S CC S
when NP VP and NP VP
DT NN NN VBZ DT NN VBZ ADJP
a open/close sensor opens the space is JJ
dark

Fig. 4: Stanford constituency tree representation of the description from the Brighten-Dark-Places SmartApp.

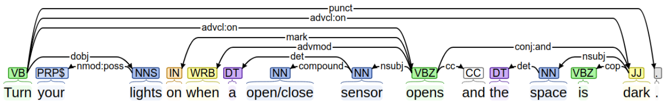

punct
advcl:on
advcl:on
mark
advmod
det
compound
dobj
nmod:poss
nsubj
conj:and
cc
det
nsubj
cop
VB PRP$ NNS IN WRB DT NN NN VBZ CC DT NN VBZ JJ
Turn your lights on when a open/close sensor opens and the space is dark .

Fig. 5: Stanford dependency tree representation of the description from the Brighten-Dark-Places SmartApp.

semantic may be human-readable and not machine-readable. We continue to do the capability matching process.

B. Capability Matching

The SmartApps interact with devices based on their capabilities. The capabilities have to be well decomposed in order to prevent over-privilege. The Samsung SmartThings platform maintains a capability list [17] that SmartApps can request. Fig. 3(b) shows the multipurpose sensor has three capabilities that can provide to apps: *capability.contactSensor*, *capability.temperatureMeasurement*, and *capability.accelerationSensor*. Although we have extracted the app behavior from the description, there could still be a semantic gap between the wording of the description and the capabilities. Hence, we need to establish the relationship between the non-phrases in the description and the capabilities. The verb phrase in the same clause may also provide useful information for the matching and could also be considered during the matching. For example, “is dark" is more related to “illuminance" than “the space".

We match noun phrases and verb phrases to the capabilities based on the similarity score computed by the Word2Vec [18] model trained on the part of Google News dataset (about 100 billion words). Because the Word2Vec only gives embedding for words, we split every phrase into a tuple of individual words. This operation is also performed for capability names. We take the highest score of all the possible word pairs between a phrase tuple and a capability tuple as the similarity of these two tuples. Once we have the similarity score for each phrase and capability pair, we match the clause to the most similar capability. For each clause, if the most similar capability is already taken by some other capabilities, the second most similar one is chosen. Taking BRIGHTEN-DARK-PLACES SmartApp as an example, the matching result is “Turn on your lights" ↔ *capability.switch*, “a open/close sensor opens" ↔ *capability.contactSensor*, and “the space is dark" ↔ *capability.illuminanceMeasurement*.

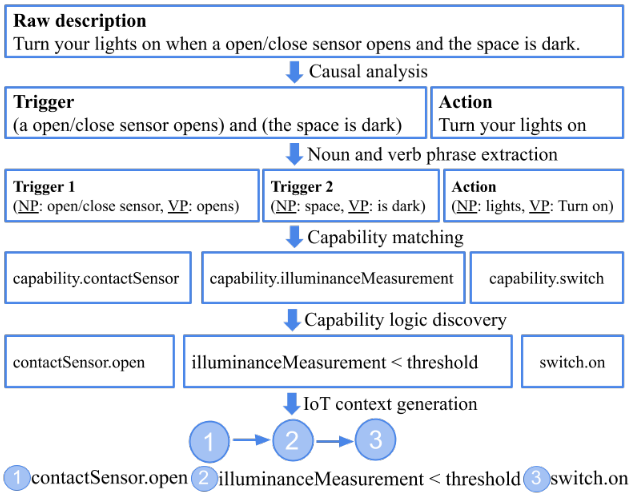

Raw description
Turn your lights on when a open/close sensor opens and the space is dark.
Causal analysis
Trigger
(a open/close sensor opens) and (the space is dark)
Action
Turn your lights on
Noun and verb phrase extraction
Trigger 1
(NP: open/close sensor, VP: opens)
Trigger 2
(NP: space, VP: is dark)
Action
(NP: lights, VP: Turn on)
Capability matching
capability.contactSensor
capability.illuminanceMeasurement
capability.switch
Capability logic discovery
contactSensor.open
illuminanceMeasurement < threshold
switch.on
IoT context generation
1
2
3
1 contactSensor.open
2 illuminanceMeasurement < threshold
3 switch.on

Fig. 6: IoT context generation from the *Brighten-Dark-Places* SmartApp.

C. Event Transition Graph Generation

After extracting the app logic and matching the verb and noun phrases to the actual capabilities, we discover the commands from the verb phrases. For example, “Turn on" clearly indicates the capability command *capability.switch.on()*. We construct the SmartApp's behavior as an event transition graph where each node represents the capability command. The complete workflow for our example SmartApp is shown in Fig. 6. The final event transition logic is: *contactSensor.open→illuminanceMeasurement* < *threshold→switch.on()*, which shows the app work logic expected by user. We build the event transition graphs for all the SmartApps installed by a user, which represent the `IoT context` in the current system.

VI. ANOMALY AND VULNERABILITY DETECTION

In this section, we introduce how to use the generated wireless context and IoT context to discover the anomalies and potential vulnerabilities in the IoT system. Each context is represented by a set of event transition graphs. We use $G = \{g_1, g_2, ..., g_i, ...\}$ and $G' = \{d_1, d_2, ..., d_j, ...\}$ to represent two sets of event transition graphs for IoT context and wireless context respectively. The nodes in the graphs g_i and d_j describe the corresponding IoT interaction events, which are represented by the unified capability commands. Meanwhile, all the events are numbered and given a global ID. The IoT context is the user expected app behaviors, and the wireless context is the actual app behaviors detected via the wireless communication traffic. If the wireless context violates the IoT context, that means potential anomalies and vulnerabilities. For each detected event transition graph d_j in the wireless context G', we check if we can find exactly the matched event transition graph g_i in the IoT context G. The match means the d_j and g_i should have identical event IDs and identical event dependencies. If there is no such match, we think there is a potential anomaly in the system.

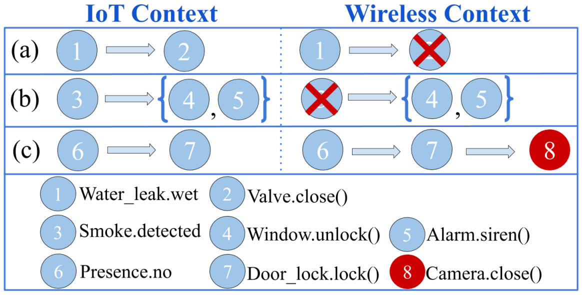


Fig. 7: Discovery of various anomalies.

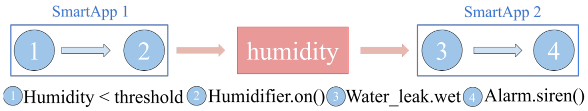


Fig. 8: Discovery of hidden vulnerability.

The Fig. 7 provide the examples of the typical anomalies we can detect via our approach. The first IoT context is "*If the water leak sensor detects the wet, close the valve*, which is represented by event transition *Water_leak.wet→Valve.close()*. For the wireless context, we only detect the first event and miss the second event. This anomaly could be caused by `device failure` or `app misbehavior`. The valve may not work due to its hardware flaws or software bugs. Also, the anomaly could be due to the app misbehavior. Once the app receives the wet alarm from the water leak sensor, it should send the command to close the valve. But from the wireless side, the app does not execute the second step. The second example is caused by `event spoofing`. Only when detecting smoke, the window is opened, and the alarm is triggered. But the attacker may spoof a fake smoke detected event and trigger subsequent actions. For the third example, we find an additional action *Camera.close()* is detected due to `over-privilege`. The actual IoT context is *If no presence is detected, lock the door.* The malicious app requests the non-necessary privilege for the camera and closes the camera after people leave the room. Then the attacker gets a chance to break in without the camera monitoring.

Furthermore, our approach can detect the potential vulnerabilities that are caused by the inter-app interaction chain. Most research work focuses on the local behaviors for one single app. They ignore the potential event interaction chain that crosses multiple apps. The chain is formed via some hidden channels, such as temperature, humidity, light, etc. The formed interaction chain could be leveraged by the attacker to launch attacks. Fig. 8 shows an example of how to discover the vulnerabilities via our approach. For wireless context, we detect a event chain ① → ② → ③ → ④. But we can only find the ① → ② and ③ → ④ in the IoT context. The first app opens the humidifier once the humidity is less than a threshold. Meanwhile, the humidity could influence the input of the second app. One malicious app can change the threshold and let the humidifier keeps working until triggering the water leak alarm. IoTGaze can detect such hidden vulnerabilities in advance and propose solutions to prevent such attacks.

Fig. 9: Smart devices and ZigBee packet sniffer used in our testbed for evaluating IoTGaze .

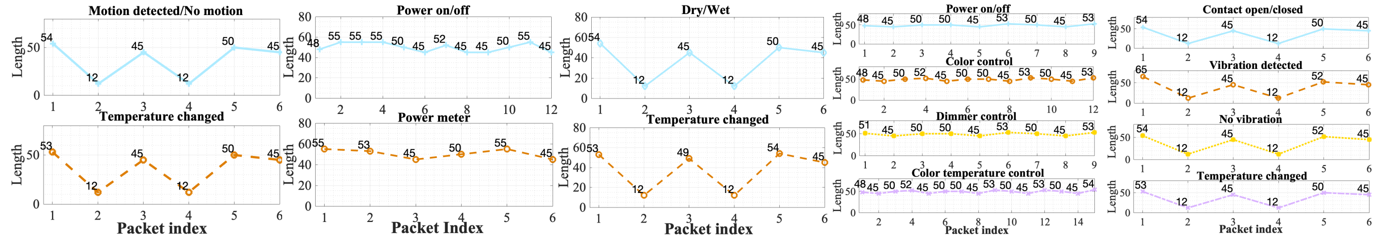
VII. Evaluation

In order to demonstrate the feasibility and effectiveness of IoTGaze , we implement our framework on the Samsung SmartThings platform. Fig. 9 exhibits the IoT devices we use in our testbed. All the devices are connected to a SmartThings hub with ZigBee wireless communication protocol. We use TI CC2531 USB Dongle [19] and install the Zigbee protocol sniffer [20] to sniff the wireless communication traffic between hub and devices. A set of SmartApps is installed to SmartThings to enable the provision of automation services. We design extensive experiments from various aspects to thoroughly evaluate our approach.

A. Anomaly Detection Evaluation

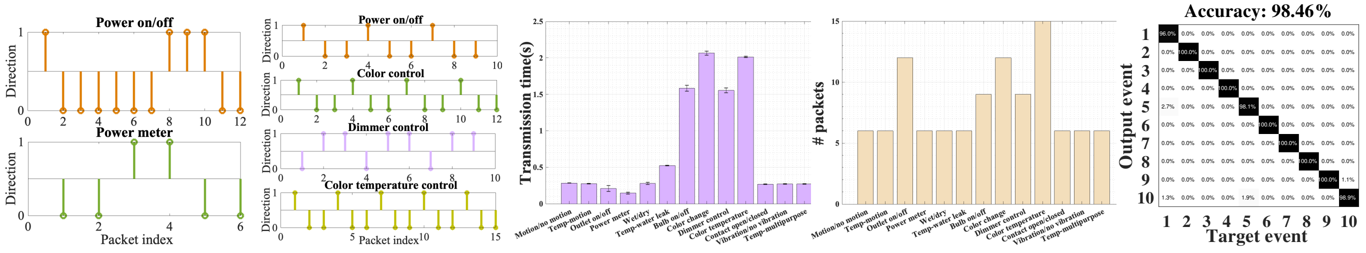
To verify the accuracy of our event fingerprinting approach, we use the five most commonly used IoT devices for our experiments: Motion sensor, Outlet, Water leak sensor, Philips Hue A19, and Multipurpose sensor. These devices can generate 19 types of events that can be found via the SmartThings iOS app. Each event corresponds to a SmartThings capability command. For example, Fig. 10(d) shows that the Philips Hue A19 can generate the following IoT events: *power on/off, color control, dimmer control,* and *color temperature control.* The corresponding capability commands are *switch.on(), switch.off(), colorControl.setColor(), switchLevel.setLevel(),* and *colorTemperature.setColorTemperature().* Our goal is to identify these events via sniffing the wireless packets.

Event Fingerprinting Analysis. We collect the sniffed Zigbee wireless traffic and correlate them with the downloaded event history from the SmartThings app. Here are observations from the experiments: (1) For most events, the packet size sequences are distinct, as shown in Fig. 10. Although the event pair *motion detected/no motion*, *power on/off*, and *dry/wet* have the same packet length with different data fields, events in each pair are contrary to each other, which implies that we can use one variable to record the device's status to distinguish these events. (2) Although some sensors use identical capabilities for the same purpose, their packet sequence and size are still distinct. For instance, the motion sensor, water leak sensor, and multipurpose sensor can all detect temperature change. Once the temperate change is detected, they all send the same event via capability command *temperature.value*. However, their packet size sequences are distinguishable and can be used for fingerprinting, as shown in Fig. 10(a)(c)(e). (3) The direction of the packet in the sequence can also be used to distinguish some events. We use **0** to denote the direction from



(a) Motion sensor (b) Outlet (c) Water leak sensor (d) Philips Hue A19 (e) Multipurpose sensor

Fig. 10: Packet size sequence from transmitted packets for different type events.



(a) (b) (c) (d) (e)

Fig. 11: (a) Packet direction for outlet related events. (b) Packet direction for Philips Hue A19 related events. (c) # packets transmission for different events. (d) Transmission time for different events. (e) Confusion matrix for event identification.

the device to the hub and use **1** for the reverse direction. Fig. 11(a)(b) show the packet directions for different events from outlet and Philips Hue A19, and verify the effectiveness of using packet direction as a fingerprint feature. (4) The inter-packet time interval and the transmission time can also be used to distinguish different events. The transmission time for each event is shown in Fig. 11(c), from which we can see that the shortest average transmission time is 0.1477s (for power meter event), while the longest average transmission time is 2.0656s (for color change event).

Event Collection and Model Training. To train the event classifier $\mathcal{C}$, we deploy the testbed in a typical office environment and continuously sniff and collect three weeks' wireless packet sequence and build the fingerprint matrix for each event. Fig. 11(d) shows that the maximum number of packets transmitted, for all kinds of events, is 15. So we set the parameter of $\mathcal{N}$ defined in section IV-A to be 15. We label the data by matching the recorded event history in the SmartThings app. For devices that are triggered very infrequently in a real office environment, such as water leak sensor, we manually wet and dry it to generate sufficient event samples for the training. We train a random forest classifier using the event samples.

Event Detection Analysis. Based on the devices' capabilities in our testbed, we select the existing apps in the SmartThings Public Github Repository [21] and also develop our customized apps to build a total of 35 apps library and install them in the SmartThings platform. We constantly sniff the wireless traffic for another one week and identify the event in a real-time manner and generate the event sequence. We set the value of parameter $\mathcal{T}$ and θ in Section IV-B to be 2.1s and 0.7 respectively. Before the identification, we remove the unrelated packets, including beacon packets, link-maintain packets, and acknowledgment packets. We still compare the detected events with the recorded events in the SmartThings hub and compute the detection accuracy. We select the ten most frequently occurred events and show their confusion matrix for classification in Fig. 11(e). The overall identification accuracy is 98.46%. The detection failure is mainly due to packet loss — the sniffer may miss some packets due to its limitation or other signal interference.

Wireless and IoT Context Discovery. After generating the sequence of the events, we mine the event dependencies using our algorithm and discover the wireless context. We successfully detect wireless context consisting of 35 event dependencies. The first eight items in Table I show part of the detected wireless context. We also use the proposed NLP approach to extract the IoT context from 35 apps installed, which exactly matches the detected wireless context. To further evaluate the applicability of our approach, we generate some complicated event dependencies, shown in the last four items in Table I. We insert these events to the sequence of the already existing events and verify that we can still discover these complicated wireless context. The experimental results demonstrate the effectiveness of our IoT context and wireless context discovery.

Anomaly Generation and Detection. For the installed 35 apps, we design and insert malicious code to the apps to generate anomalies. For each app, we modify its code to generate the following three types of anomalies: (a) Event Spoofing. We add the code in the app to spoof some events for triggering purpose. The first three items in Table I show the examples, such as event sequence changing from ① → ② to ~~①~~→ ②. (b) App Misbehavior. For the item 4-6 in Table

TABLE I: Anomaly detection via the discovery and comparison of IoT and wireless context.

No.	Event dependencies discovered in IoT context	Detected wireless context
1	$\boxed{\text{motion sensor}} \xrightarrow{\text{motion.active}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{Philips Hue}}$	~~①~~ → ②
2	$\boxed{\text{multipurpose sensor}} \xrightarrow{\text{temperature.value}} \boxed{\text{hub}} \xrightarrow{\text{colorControl.setColor()}} \boxed{\text{Philips Hue}}$	~~③~~ → ④
3	$\boxed{\text{outlet}} \xrightarrow{\text{power.value}} \boxed{\text{hub}} \xrightarrow{\text{switch.off()}} \boxed{\text{outlet}}$	~~⑤~~ → ⑥
4	$\boxed{\text{water leak sensor}} \xrightarrow{\text{water.wet}} \boxed{\text{hub}} \xrightarrow{\text{switch.off()}} \boxed{\text{outlet}}$	⑦ → ~~⑥~~
5	$\boxed{\text{multipurpose sensor}} \xrightarrow{\text{acceleration.active}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{Philips Hue}}$	⑧ → ~~②~~
6	$\boxed{\text{multipurpose sensor}} \xrightarrow{\text{contact.open}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{Philips Hue}}$	⑨ → ~~②~~
7	$\boxed{\text{multipurpose sensor}} \xrightarrow{\text{contact.close}} \boxed{\text{hub}} \xrightarrow{\text{switch.off()}} \boxed{\text{outlet}}$	⑨ → ⑥ → ②
8	$\boxed{\text{motion sensor}} \xrightarrow{\text{motion.inactive}} \boxed{\text{hub}} \xrightarrow{\text{colorControl.setHue()}} \boxed{\text{Philips Hue}}$	① → ⑩ → ②
9	$\boxed{\text{hub}} \xrightarrow{\text{switch.off()}} \boxed{\text{bulb}}, \boxed{\text{hub}} \xrightarrow{\text{lock.lock()}} \boxed{\text{lock}}, \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{camera}}$	⑪ → ⑫ → ~~⑬~~
10	$\{ \boxed{\text{multipurpose}} \xrightarrow{\text{contact.open}}, \boxed{\text{illuminance sensor}} \xrightarrow{\text{illuminance.value}} \} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{bulb}}$	{ ~~⑨~~, ⑭ } → ⑪
11	$\boxed{\text{multipurpose}} \xrightarrow{\text{temperature.value}} \boxed{\text{hub}} \{ \xrightarrow{\text{colorControl.setColor()}} \boxed{\text{Hue}}, \xrightarrow{\text{switch.on()}} \boxed{\text{heater}} \}$	③ → { ~~④~~, ~~⑮~~ }
12	$\{ \boxed{\text{thermostat}} \xrightarrow{\text{presence.not\_present}}, \boxed{\text{multipurpose}} \xrightarrow{\text{contact.closed}}, \boxed{\text{lock}} \xrightarrow{\text{lock.unlocked}} \} \boxed{\text{hub}} \xrightarrow{\text{lock.lock()}} \boxed{\text{lock}}$	{ ⑯, ⑰, ⑱ } → ~~⑫~~

TABLE II: Anomaly detection results for three types of anomaly.

	Spoofing	Overprivilege	Misbehavior
Precision	97.22%	98.55%	98.29%
Recall	94.82%	98.36%	95.20%

I, we modify the apps' code and make the app *not* execute the triggered actions, leading to the event sequence change from ⑦ → ⑥ to ⑦ → ~~⑥~~. (c) Over-privilege. The item 7-8 show the anomaly samples we generate. We modify the code to request the non-necessary capabilities and execute the addition actions, such as changing from ⑨ → ⑥ to ⑨ → ⑥ → ②. For each app, we generate 100 anomalies for each threat type and try to detect them via our approach.

The results of anomaly detection are shown in Table II. We can see that the detection for overprivilege has both high precision and recall. This is because the overprivileged event sequence is different from all of the normal sequences, so the precision and recall are just limited by the success rate of event detection. Spoofing and misbehavior can occasionally generate event sequence which match the normal app behavior. Therefore, their precision is high but recall is low.

B. Hidden Vulnerabilities Discovery

The current research mostly focuses on security vulnerability detection per SmartApp. The local behaviors of one single app may explicitly or implicitly affect the whole IoT system. The potential interactions between apps, devices, and the environment may produce vulnerabilities that cannot be discovered by per-app analysis. We propose to use the wireless context to discover the hidden vulnerabilities that also can be exploited by attackers.

In wireless context, we can find some wireless event dependencies spanning multiple applications. This is because these applications are somehow *correlated* together via some hidden channels. We thoroughly investigate the 183 apps in the SmartThings Public GitHub Repository [21] and analyze their interactions with other apps, devices, and the environment. We find that there are three kinds of channels that can cause potential vulnerabilities: (1) **Capability**. Two applications can interact if the first app's output is the trigger of the second app. For example, the SmartApp "NFC Tag Toggle" in the official SmartThings GitHub allows toggling of a switch, lock, or garage door. And another SmartApp "Door State to Color Light (Hue Bulb)" changes the color of Hue bulbs based on the door status. In this example, the two apps are directly chained via the capability *doorControl*. (2) **Physical channel**. The environment elements can be changed due to the input or output of some apps and cause potential interaction chains. We take one physical channel *smoke* as an example. The toaster may cause smoke, which makes alarm siren. (3) **System channel**. Some global variables in the IoT program framework may be shared by some SmartApps. The *location.mode* in SmartThings platform enables the devices to behave differently in different scenarios. For example, if the current location.mode is "Home" and the motion sensor detects motion, then turn on the light. But if the location.mode is "Away" and the motion is detected, then turn on the camera. All these three kinds of channels can generate unexpected application interaction, rendering system vulnerabilities.

We discover and provide the list of all the hidden vulnerabilities for each type of channel from the SmartThings platform. Based on the NLP approach in Section V, the capabilities related to the apps' input and output are extracted. We list seven capabilities that are shared by apps in Table IV. The capability *switch(light)* generates 127 potential inter-app interaction chains and has the highest risk score. We use Word2Vec [18] to establish the mapping between physical channels and apps' input and output. In total, nine physical channels are discovered, as shown in Table IV. The illuminance, energy, and temperature are the channels that bring the most of inter-app interactions. When we program the malicious apps, we show that system variable *location.mode* from SmartThings program platform *location.mode* are frequently used and modified by some apps, which can also cause security-relevant issues. We list the vulnerabilities found for each type of channel in Table III and show the statistics about vulnerabilities in Table IV.

TABLE III: Hidden vulnerabilities discovery via analyzing wireless context.

No.	Event dependencies discovered in wireless context
1	$\boxed{\text{time}} \rightarrow \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{heater}} \rightarrow \boxed{\textbf{temperature}} \rightarrow \boxed{\text{temperature sensor}} \xrightarrow{\text{temperature.value}} \boxed{\text{hub}} \xrightarrow{\text{window.open()}} \boxed{\text{window}}$
2	$\boxed{\text{temperature sensor}} \xrightarrow{\text{temperature.value}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{fan}} \rightarrow \boxed{\textbf{motion}} \rightarrow \boxed{\text{motion sensor}} \xrightarrow{\text{motion.active}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{light}}$
3	$\boxed{\text{water leak sensor}} \xrightarrow{\text{water.wet}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{light}} \rightarrow \boxed{\textbf{illuminance}} \rightarrow \boxed{\text{illum sensor}} \xrightarrow{\text{illuminance.value}} \boxed{\text{hub}} \xrightarrow{\text{windowShade.close()}} \boxed{\text{window shade}}$
4	$\boxed{\text{presence sensor}} \xrightarrow{\text{presence.not\_present}} \boxed{\text{hub}} \xrightarrow{\text{lock.lock()}} \boxed{\textbf{lock}} \xrightarrow{\text{lock.lock}} \boxed{\text{hub}} \xrightarrow{\text{colorControl.setColor()}} \boxed{\text{Hue}}$
5	$\boxed{\text{presence sensor}} \xrightarrow{\text{presence.present}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\textbf{bulb}} \xrightarrow{\text{switch.on}} \boxed{\text{hub}} \xrightarrow{\text{camera.take()}} \boxed{\text{camera}}$
6	$\boxed{\text{multipurpose sensor}} \xrightarrow{\text{temperature.value}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\textbf{AC}} \xrightarrow{\text{switch.on}} \boxed{\text{hub}} \xrightarrow{\text{switch.on()}} \boxed{\text{bulb}}$
7	$\boxed{\text{presence.not\_present}} \xrightarrow{\text{presence.not\_present}} \boxed{\text{hub}} \rightarrow \boxed{\textbf{location mode}} \rightarrow \boxed{\text{hub}} \xrightarrow{\text{switch.off()}} \boxed{\text{light}}$

TABLE IV: Statistics of hidden channels identified from official SmartApps.

Channel Type	Channel	# apps related	# interaction chains
Capability	swtich(light)	28	127
	doorControl	4	4
	lock	8	22
	switch(heater)	10	27
	switch(AC)	9	23
	colorControl	7	6
	thermostat	9	20
Physical	leakage	4	5
	illuminance	29	132
	energy	36	134
	contact	20	37
	acceleration	9	18
	smoke	10	17
	temperature	18	127
	motion	14	13
	humidity	4	3
System	location.mode	9	16

VIII. Related Work

IoT system is composed of protocols, devices, apps, platforms, and the environment. The complexity of the IoT system makes it challenging to resolve security and privacy issues. Each component in the IoT system can cause potential threats [22]. The device flaws [7], [7], [23] can be exploited by attackers to infiltrate the IoT networks. For smart apps, the static and dynamic program analysis [10], [11], [13], [24] are used to track the apps' control and data flow so as to prevent the sensitive data leakage and identify the potential app misbehavior. The research work [8], [9], [25] focus on the platform security and try to exploit the design flaws of exiting program frameworks and propose solutions to prevent app over privilege and sensitive information leakage. For instance, The authors [25] propose to collect provenance of events and data state changes to build provenance graphs of their causal relationships, enabling attack detection.

Some other techniques are also used to enhance the IoT security. The device fingerprinting technique is developed in [26], [27] to distinguish between legitimate devices and attacker devices. By analyzing the encrypted network traffic, [28] can build app fingerprints and [29], [30] can build the fingerprints for identifying the types of devices. Model checking is used in [31] as a building block to reveal "interaction-level" flaws by identifying events that can lead the system to unsafe states. Graph-based detection approaches [32], [33] can also be applied to IoT to detect anomalies. Natural language processing (NLP) is used in mobile apps [34]–[36] and IoT apps [12], [37] to automatically extract security-relevant information from apps' description, code, and annotations. The extracted semantics are compared to the tracked control and data flows in the program so as to detect apps' misbehaviors, which require complicated program analysis techniques. Our approach considers the anomaly detection starting from the view of wireless context. HoMonit [38] has a similar idea with us, and it compares the IoT activities inferred from the encrypted traffic with their expected behaviors dictated in their source code. But, our work does not need to analyze the source code and mainly focuses on the discovery of wireless context. We generate the sequential IoT events and mine their temporal event dependencies to explore all actual wireless context, and then compare with the IoT context inferred from apps' descriptions. By analyzing the wireless context, we can also provide a new approach to discover the hidden vulnerabilities, which HoMonit does not support.

IX. Conclusion

In this paper, we propose a novel IoT anomaly detection framework called IoTGaze . Instead of exploring the threats inside platform and apps, we deploy a third-party monitor IoTGaze , who gazes at the wireless traffic and detects the potential threats in the IoT system via analyzing the encrypted wireless packets. We propose a new concept called `wireless context` in IoT that represents the observed app logic from wireless sniffing. We design a fingerprinting based event detection approach and use it to generate the event sequence via sniffed wireless packets. We design an algorithm to discover the temporal event dependencies and build the wireless context. We also extract the IoT context that reflects user expected app behaviors via analyzing apps' descriptions via natural language processing techniques. By matching the wireless and IoT context, we can detect the anomalies that are happening in the IoT system. Furthermore, the event dependencies discovered by IoTGaze can reveal some potential vulnerabilities that are caused by the inter-app interaction via some hidden channels. We prototype our approach on the Samsung SmartThings platform and demonstrate the feasibility and effectiveness of IoTGaze .

REFERENCES

[1] Ericsson, "Ericsson mobility report: Internet of things forecast." https://www.ericsson.com/en/mobility-report/internet-of-things-forecast, 2018.

[2] IDC, "Worldwide semiannual internet of things spending guide." https://www.idc.com/getdoc.jsp?containerId=IDC_P29475, 2018.

[3] Samsung, "Smartthings." https://www.smartthings.com, Accessed: 2015.

[4] Apple, "Homekit." https://developer.apple.com/homekit/, Accessed: 2015.

[5] Google, "Google home." https://developers.google.com/iot/, Accessed: 2015.

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

[7] E. Ronen, A. Shamir, A.-O. Weingarten, and C. OâĂŹFlynn, "Iot goes nuclear: Creating a zigbee chain reaction," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 195–212, IEEE, 2017.

[8] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 636–654, IEEE, 2016.

[9] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 531–548, 2016.

[10] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, "Sensitive information tracking in commodity iot," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1687–1704, 2018.

[11] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, "Contexlot: Towards providing contextual integrity to appified iot platforms.," in *NDSS*, 2017.

[12] Y. Tian, N. Zhang, Y.-H. Lin, X. Wang, B. Ur, X. Guo, and P. Tague, "Smartauth: User-centered authorization for the internet of things," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 361–378, 2017.

[13] Z. B. Celik, G. Tan, and P. D. McDaniel, "Iotguard: Dynamic enforcement of security and safety policy in commodity iot.," in *NDSS*, 2019.

[14] C. Nandi and M. D. Ernst, "Automatic trigger generation for rule-based smart homes," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, pp. 97–102, ACM, 2016.

[15] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what?: Controlling flows in iot apps," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security(ACM CCS'18)*, pp. 1102–1119, ACM, 2018.

[16] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pp. 55–60, 2014.

[17] Samsung, "Capabilities reference for smartthings iot platform." https://docs.smartthings.com/en/latest/capabilities-reference.html, 2018.

[18] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[19] T. INSTRUMENTS, "Zigbee protocol packet sniffer." http://www.ti.com/tool/PACKET-SNIFFER, 2019.

[20] T. INSTRUMENTS, "Cc2531 usb evaluation module kit." http://www.ti.com/tool/cc2531emk, 2019.

[21] S. developer, "Smartthings public github repository." https://github.com/SmartThingsCommunity/SmartThingsPublic, 2019.

[22] Z. Fang, h. Fu, T. Gu, Q. Zhiyun, T. Jaeger, and P. Mohapatra, "Foresee: A cross-layer vulnerability detection framework for the internet of things," in *2019 IEEE 16th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*, IEEE, 2019.

[23] V. Sivaraman, D. Chan, D. Earl, and R. Boreli, "Smart-phones attacking smart-homes," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks(WiSec'16)*, pp. 195–200, ACM, 2016.

[24] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pp. 147–158, 2018.

[25] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *Network and Distributed Systems Symposium*, 2018.

[26] J. Han, A. J. Chung, M. K. Sinha, M. Harishankar, S. Pan, H. Y. Noh, P. Zhang, and P. Tague, "Do you feel what i hear? enabling autonomous iot device pairing using different sensor types," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 836–852, IEEE, 2018.

[27] D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. A. Beyah, "Who's in control of your control system? device fingerprinting for cyber-physical systems.," in *NDSS*, 2016.

[28] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 439–454, IEEE, 2016.

[29] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A.-R. Sadeghi, and S. Tarkoma, "Iot sentinel: Automated device-type identification for security enforcement in iot," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 2177–2184, IEEE, 2017.

[30] T. Gu and P. Mohapatra, "Bf-iot: Securing the iot networks via fingerprinting-based device authentication," in *2018 IEEE 15th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pp. 254–262, IEEE, 2018.

[31] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, "Iotsan: fortifying the safety of iot systems," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies*, pp. 191–203, ACM, 2018.

[32] J. Jiang, C. Jiuming, T. Gu, K.-K. Raymond Choo, C. Liu, M. Yu, W. Huang, and P. Mohapatra, "Anomaly detection with graph convolutional networks for insider threat and fraud detection," in *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, 2019.

[33] J. Jiang, C. Jiuming, T. Gu, K.-K. Raymond Choo, C. Liu, M. Yu, W. Huang, and P. Mohapatra, "Warder: Online insider threat detection system using multi-feature modeling and graph-based correlation," in *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*, 2019.

[34] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, "{WHYPER}: Towards automating risk assessment of mobile applications," in *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, pp. 527–542, 2013.

[35] X. Pan, Y. Cao, X. Du, B. He, G. Fang, R. Shao, and Y. Chen, "Flowcog: context-aware semantics extraction and analysis of information flow leaks in android apps," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1669–1685, 2018.

[36] M. Zhang, Y. Duan, Q. Feng, and H. Yin, "Towards automatic generation of security-centric descriptions for android apps," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security(ACM CCS'15)*, pp. 518–529, ACM, 2015.

[37] W. Ding and H. Hu, "On the safety of iot device physical interaction control," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security(ACM CCS'18)*, pp. 832–846, ACM, 2018.

[38] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "Homonit: Monitoring smart home apps from encrypted traffic," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security(ACM CCS'18)*, pp. 1074–1088, ACM, 2018.