2021 IEEE Conference on Communications and Network Security (CNS) | 978-1-6654-4496-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CNS53000.2021.9705050

Context-Aware IoT Device Functionality Extraction from Specifications for Ensuring Consumer Security

Upakar Paudel [*], Andy Dolan [†], Suryadipta Majumdar [‡], Indrakshi Ray [*]
[*] Colorado State University, Fort Collins, Colorado
{Upakar.Paudel,Indrakshi.Ray}@colostate.edu
[†]CableLabs, Louisville, Colorado
A.Dolan@cablelabs.com
[‡] Concordia University, Montreal, Canada
Suryadipta.Majumdar@concordia.ca

Abstract—**Internet of Thing (IoT) devices are being widely used in smart homes and organizations. An IoT device has some intended purposes, but may also have hidden functionalities. Typically, the device is installed in a home or an organization and the network traffic associated with the device is captured and analyzed to infer high-level functionality to the extent possible. However, such analysis is dynamic in nature, and requires the installation of the device and access to network data which is often hard to get for privacy and confidentiality reasons. We propose an alternative *static* approach which can infer the functionality of a device from vendor materials using Natural Language Processing (NLP) techniques. Information about IoT device functionality can be used in various applications, one of which is ensuring security in a smart home. We demonstrate how security policies associated with device functionality in a smart home can be formally represented using the NIST Next Generation Access Control (NGAC) model and automatically analyzed using Alloy, which is a formal verification tool. This will provide assurance to the consumer that these devices will be compliant to the home or organizational policy even before they have been purchased.**

Index Terms—**IoT, Smart Home, Device Functionality, NLP**

I. Introduction

We are moving towards a smart and connnected world, which is estimated to have 55.7 billion connected devices by 2025, out of which 75% will be IoT devices [1]. IoT devices contain many sensors and actuators and are able to perform multiple functionalities. For example, Nest Protect [2] whose primary functionality is to detect smoke also contains a microphone, rendering it to audio capture functionality. Consumers may install Nest Protect in their bedroom for smoke detection purpose, but Nest Protect's functionality of audio capture puts their security and privacy at risk. Thus, there is a need to understand the latent functionalities of IoT devices and their impact on security and privacy.

The most widely prevalent approaches [3]–[11] for obtaining device functionalities fall under the category of *dynamic approaches* which profile device behaviors by installing or simulating IoT devices and observing their network behaviors and inferring their functionalities. Often times, the network traffic is encrypted; consequently, such approaches are inaccurate and report high false positive and false negative rates. Moreover, deriving high-level functionality from low-level network data is non-trivial. Such approaches require set-up effort and time for capturing device information. If such set-up is done in real-world settings, device data may violate security or privacy concerns of consumers. One *static approach* [12] profiles device functionalities by analyzing vendor materials of IoT devices. This approach also suffers from accuracy issues, as they mainly rely on identification of device functionalities without considering the contextual information of words. For instance, the same word such as 'light' can have multiple meanings, and so it is hard to deduce whether the corresponding device can illuminate a room.

In this paper, we propose a static analysis approach. We associate a set of keywords with generic devices based on their functionality, check for the presence of these keywords and their context in the vendor materials using NLP techniques (e.g., [13], [14]), deduce the presence of relevant transducers in the IoT device, and use this knowledge to obtain the device functionalities. The use of contextual information in the vendor material reduces the number of false positives. We evaluate our approach on IoT products from different vendors, including Arlo and Samsung, and report encouraging results.

We further demonstrate how the knowledge of device functionalities can improve security and privacy in a smart home. Towards this end, we formulate device placement policies for the smart home IoT environment using an attribute-based access control model where access to a resource is contingent on the properties of the subject, the environment, and the resources. We use the NIST Next Generation Access Control (NGAC) model [15] because of its ability to model dynamic policies, where constraints can change while the policy is deployed. Also, policies may be complex and the constraints may conflict with each other. We need to analyze and demonstrate policy consistency and conformance. Manual analysis is tedious and error-prone. Towards this end, we demonstrate how to translate the NGAC policy to Alloy [16], which has tool support for automated policy verification.

We provide a framework that allows for extracting device

This work was supported in part by funds from NIST under award number 60NANB18D204, and from NSF under award number CNS 1822118 and from NIST, Statnett, Cyber Risk Research, AMI, and ARL.

functionality and checking conformance of device placement even before device purchase. Our contributions include the following. (i) We provide a context-aware approach for extracting IoT device functionalities from publicly-available resources using NLP. (ii) We illustrate how IoT device functionalities can be used for enhancing security. We show how our extracted functionalities and associated policies can be represented using standardized access control model (NGAC) and then be analyzed using formal verification methods (Alloy). (iii) We implement our proposed approach in the context of smart homes with diverse devices (Nest Camera, Arlo Ultra Cam, Samsung Smart Cam) from various vendors (Arlo, Nest and Samsung), and evaluate its efficiency, accuracy, and scalability, to demonstrate the feasibility of our approach.

II. Related Work

Dynamic Approach. [3]–[9] perform fingerprinting of IoT device behavior by analyzing network traffic. [3]–[5] use machine learning model trained on network traffic according to their service (e.g., DNS, HTTP) and the semantic behaviors of devices (e.g., detected motion) to automatically discover and profile device behaviors. [7]–[9] use unsupervised learning to build models for individual devices based on captured network traffic and automatically detect device identity. Hamza et al. [17]–[19] use Manufacturer Usage Description (MUD) profiles to fingerprint and detect device functionalities. These works detect network level device functionality based on network traffic. Particularly, Hamza et al. [17] develop a tool to detect and verify MUD profile based on network traffic, and device classification has been performed based on observed network traffic as well as generated MUD signature. It converts MUD policies to flow rules and uses those flow rules to detect an intrusion. [19] uses security testing methodologies to augment the MUD profile and increases its expressiveness by considering additional security aspects beyond just network traffic. MUD based solutions rely on MUD profiles and on vendors to provide complete and correct MUD profiles for IoT devices. For both ML and MUD based approaches device access is required and the device must be connected and be operational. Although these approaches can detect network-level behavior, inferring high level operation is error-prone because of the encrypted payload.

Static Approach. Our previous work [12] implements a technique of extracting IoT device functionalities based on design specifications from vendor materials. We use key term matching to detect transducers from vendor materials and map those transducers to their capabilities for identifying device functionalities. However, this work ignores the usage context of the key terms and hence results in high false positive rates.

Context-based approaches for IoT. [20] makes an effort towards understanding the context of the data generated by IoT sensors and actuators and to provide personalized recommendations to the users based on the captured data. Similarly, [21] exploits the heterogeneous contextual information (e.g., daily activities) that are captured from IoT devices. Then, those contextual information is leveraged to be used in personalizing care management process especially for elderly people. Our work aims at a different objective where we intend to extract device functionalities for consumer security without using any captured data.

III. Preliminaries

A. Vendor Material Description

This work considers various publicly available vendor materials including product webpages, technical specifications, and setup videos, for the purpose of extracting device functionalities. Product webpages (e.g., [22]) are the official marketing pages briefly describing a device, including its basic features. Technical specification pages (e.g., [23]) provide device specifications, including its hardware configurations. Setup videos (e.g., [24]) elaborate on how to install IoT devices including useful information about device functionalities. During our analysis, we experience on average 32-page long specification documents and 4:48 minute long setup videos.

B. Challenges of Context-Aware IoT Device Functionality Extraction

The challenges in extracting IoT device functionalities from vendor materials are as follows.

- *Understanding multimedia contents.* In addition to text, device vendors often use multimedia technology, such as picture, audio, and video. Thus, text processing alone becomes insufficient for extracting device functionalities. Automatically encoding these multimedia elements poses additional difficulty.
- *Minimal mention of key terms.* In many cases, key terms (i.e., that might indicate the presence of a transducer or functionality) are mentioned very few times in vendor materials with brief information about them. Therefore, obtaining contextual information about those key terms becomes challenging.
- *Ambiguous use of key terms.* The key terms are used in different ways by the various vendors. This makes it hard to identify the context and infer the presence of a functionality.
- *Lack of standardized template.* Each vendor follows different conventions in arranging their materials. Furthermore, different materials of the same vendor follow different formats. Also, vendors craft expressions and styles, and include non-standard terms and phrasing unique to only their line of products. The lack of standard format for vendors to describe generic features or hardware makes device functionality extraction across different vendors challenging.

C. Intuition behind Context-Aware Device Functionality Extraction

We initially conduct a feasibility study to demonstrate the effect of context in inferring device functionality. For example, the term *temperature* is used in different contexts, such as for measuring device temperature, for referring to operational temperature, and for sensing/adjusting a room temperature.

Thus, we need to understand the context before inferring device functionalities. In this feasibility study, we explore two major NLP techniques, N-gram and BERT.

N-gram [13]. N-gram is an NLP technique to predict the occurrence of a word based on its previous N words. [12] extracts the device specification from various vendor materials (overview page, technical specification page, and manuals). After extracting the device specification, then irrelevant contents (e.g., stop words, site navigation link, copyright information) are pruned to get the overall corpus for a specific device. We operate on this generated corpus to apply N-gram and detect local context for a specific term. Once we have a corpus for a specific device, we convert the corpus to trigram. We are interested in certain transducers (for example, *temperature sensor*) and the goal is to detect the contexts associated with the key terms of those transducers in the extracted corpus.

For example, in the Nest Thermostat [25] corpus, we have the phrase: *"nest temperature sensor room like"*. The trigrams generated from this phrase are `<nest temperature sensor>`, `<temperature sensor room>`, and `<sensor room like>`. From the Arlo Pro 3 Floodlight corpus, the phrase *"5 hrs operate temperature"* generates the following trigrams: `<5 hrs operate>` and `<hrs operate temperature>`. By analyzing, the preceding and following words in trigram for both Nest thermostat and Arlo Video Doorbell, we can see that the word `temperature` in Nest thermostat has the context of sensing temperature whereas the word `temperature` in Arlo video doorbell has the context of operational environment. We utilize this observation in our methodology in Section IV.

BERT [14]. BERT (Bidirectional Encoder Representation From Transformers) is a transformer-based language representation model and learns information from both sides of a word to learn the context of a specific word [26]. In this work, BERT is used to detect contextual semantics associated with key terms in vendor materials.

For example, in the following two sentences using the term *temperature*, BERT uses bidirectionality to understand their contextual meaning: (i) "check if the arlo app is warning that your doorbell temperature is too high", and (ii) "the temperature they sense is warmer or cooler than homeowner's feel". In *(i)*, we factor the context of the words mentioned before *temperature* to understand whether the current context is of type operating or notifying. In *(ii)*, to identify the context in which term *temperature* is being mentioned, we read the words following it which clearly indicates the context of 'sensing'. In our methodology (in Section IV), we leverage this strength of BERT to accurately extract functionalities using contextual meaning of a term.

D. Threat Model

This work is designed in the context of smart home, a growing IoT application. One focus of this work is to extract device functionalities using contextual information from vendor materials. The other focus is to demonstrate how the extracted functionalities are useful for security and privacy policies for consumers. The device functionalities can be used in access control scenario to verify device behavior according to required policy as well as to detect and prevent attacks that makes use of the sensors or actuators. In this work, we do not consider possible threat stemming from device misbehavior/malfunction and network attack that does not involve device transducers. This work does not rely on device access as well as network data. We enumerate device functionalities based on the vendor materials that are publicly and readily accessible. Missing information in those materials may affect the robustness of our approach.

IV. Proposed Framework

This section presents our proposed framework as follows.

A. Overview

Figure 1 displays a general overview of our proposed framework. Our framework can be categorized into four phases: (i) data collection and pre-processing, (ii) contextual embedding generation, (iii) clustering, and (iv) prediction. In the *data collection and pre-processing phase*, we first collect entire corpus of texts from vendor materials (e.g., overview page, technical specification page, manuals and setup videos) and then extract both key terms (i.e., the words that might be related to transducers) and their contextual information by leveraging NLP techniques. In the *contextual embedding generation* phase, we generate contextual representation of key terms based on the context of an entire sentence in which the terms are present. In the *clustering* phase, we cluster sentences with similar context together and also annotate those clusters. Finally, in the *prediction* phase, we predict if any sentences from a new device are clustered to the annotated cluster to identify the functionalities of that device. The output (i.e., device functionalities) of our framework can be used for different security applications, such as access control, auditing. In this paper, we obtain functionalities of several smart home devices from different vendors, and utilize those functionalities to specify and verify policies for consumers. Each phase of this framework is explained below.

B. Data Collection and Pre-processing

Data Collection. In this work, we extract all the text from various vendor materials (i.e., technical specifications, product overview pages, manuals or user guides, and setup videos). We leverage existing tools (e.g., beautifulsoup) to extract texts from the webpage (overview page and technical specification page), and manuals or user guide. We also extract texts from setup videos for a device by using Google Web Speech API [27]. Once all texts are extracted, we remove stop words and punctuation from those sentences as well as lemmatize our corpus so that we can leverage it further for understanding transducers and device functionalities.

Pre-processing. To extract the transducer information from the vendor materials, we first build an ontology, and then compare

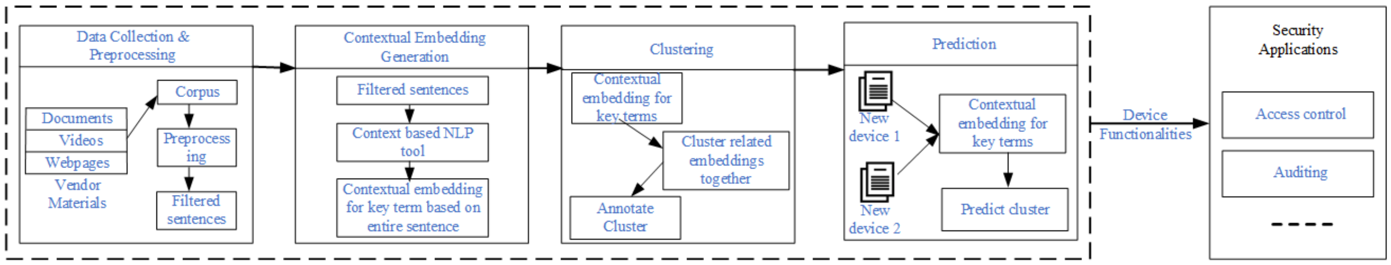


Fig. 1. Our proposed methodology for the context-aware IoT device functionality extraction

the obtained corpus and our ontology. Note that the ground truth is established based on this ontology, which is manually built from the analysis of vendor materials. To build an ontology of transducers (sensors and actuators), we perform a set matching algorithm between the key terms and words collected from corpus. Specifically, key terms are divided into two categories: *indicative* and *related*. The indicative terms are sufficient to conclude the presence of a transducer and related functionality in a device. The related terms are also about transducers but are rather vague and insufficient in reaching to a conclusion about the presence of transducer and related functionality. To match the obtained corpus and our ontology, we use one of the four algorithms as follows.

- *Device Cognisant Key Term Set Matching (dcKTSM):* In this algorithm, we select the most relevant key term set based on the presence of indicative terms. We extract matching key terms from our ontology based on the input corpus. This results in a subset of indicative terms for a candidate device. We then perform reverse mapping from these indicative terms to the transducer they refer to.
- *Device Cognisant Full Key Term Set Matching (dcfKTSM):* This algorithm is similar to dcKTSM. The only difference is that we consider both indicative and related terms to select most relevant key terms rather than just indicative terms as in dcKTSM.
- *Indicative Key Term Set Matching (iKTSM):* This algorithm directly considers corpus on all indicative terms from our ontology. The reverse mapping process from indicative term to transducer remains the same as above.
- *All Key Term Set Matching (aKTSM):* This algorithm is the most unguided algorithm. the iKTSM and aKTSM algorithms are very similar, except that in aKTSM, we evaluate corpus on both indicative and related terms as opposed to using only indicative terms in iKTSM.

However, solely applying any of the above algorithms on our corpus to elaborate device functionalities results in high false positive/negative rates (i.e., wrongly identifying the presence or absence of a functionality). This is mainly because of the naivety in our approach where we are directly mapping device to their functionalities based on the presence of indicative and related terms specific to a certain transducer in our collected corpus. For example, based on key term set matching algorithms, the presence of sentence "*Check if the Arlo app is warning that your doorbell temperature is too high*" would indicate the presence of temperature sensor in Arlo Video Doorbell. However, in reality, Arlo Doorbell does not contain a temperature sensor. Therefore, in this work, we should also consider the context in the above sentence so that we can understand the appropriate intent (e.g., unfavorable environment being generated for a device) of the term *temperature* in that sentence. Therefore, to prune the device functionalities obtained from the key term set matching algorithms, we enlist transducers that can be represented in different contexts by vendors in their materials. For the rest of the paper, we will use temperature sensor, light sensor and lock (in eight Smart Home products: Nest Camera, Nest Protect, Nest Thermostat, Nest x Yale Lock, Arlo Video Doorbell, Arlo Pro 3 Floodlight, and Samsung Smart Cam [2], [23], [25], [28]–[31]) as examples to present our context-aware approach.

C. Contextual Embedding Generation

This phase generates the contextual representation for key terms. As shown in Section III, we first build our intuition of context-aware extraction by conducting preliminary studies using both N-gram and BERT. Based on the outcome of that study, we conclude that even though N-gram (where $N = 3$) significantly improves the false positive rates over the basic key term set matching approach [12], this is an exhaustive process with low confidence. Therefore, we choose BERT to generate the contextual embedding for key terms as follows.

We extract all the sentences from vendor materials with key terms from the entire corpus on eight devices (as mentioned earlier). We then divide six devices for training purposes (where we cluster sentences with similar meaning to a key term together) and two devices for testing purposes (where we check if we could correctly predict the sentence from a new device and identify its functionalities based on the functionalities of the predicted cluster). We evaluate on all combinations (${}^{8}C_{2}$) of training and testing device, which totals to 28 combinations in Section VI. We then generate a BERT token for the key terms (e.g., *temperature* and *lock*) based on the context of the entire sentence. For example, the BERT token for the term *temperature* in the sentence "*Operating **temperature** -20 to 60 degree Celsius*" and sentence "*You adjust the **temperature** from your phone so they'll be cozy*" have different representations due to their different contexts. To generate the BERT token, we use Hianxao's BERT as a service tool [32]. We use the BERT model with 12 layers, 768 hidden units, and 12 attention heads for generating tokens.

D. Clustering

This phase clusters related token (that are generated by BERT) together by leveraging an affinity propagation algorithm, which is a graph-based clustering algorithm similar to K-Means that cluster the related data points together. In K-means, the value of 'K' (number of clusters) must be pre-specified whereas affinity propagation algorithm can automatically detect an optimal number of clusters [33]. After applying affinity propagation to BERT tokens, the contextually similar tokens are clustered together.

TABLE I
CLUSTER AFTER APPLYING AFFINITY PROPAGATION TO BERT TOKENS

Cluster 1
Now turn up the temperature and get comfortable
Put a Nest Temperature Sensor in any room, like the baby's room, and you can tell Nest to make that room a priority (sold separately)
The Nest Thermostat can use sensors and your phone's location to check if you've left, then sets itself to an Eco Temperature to save energy
You adjust the temperature from your phone so they'll be cozy
You may also be into Google Nest Mini From Google Nest Protect From Google Nest Temperature Sensor
The temperature they sense is warmer or cooler than homeowners feel
In a room that's used often, so Nest can read the right temperature and the homeowner can easily reach it
If your existing chime doesn't ring when someone presses your Video Doorbell, your Video Doorbell or Power Kit might not be wired correctly, or the temperature of your Arlo Video Doorbell might be too high
Check if the Arlo app is warning that your doorbell temperature is too high

Table I and Table II display the subsets of cluster after successfully implementing affinity propagation on BERT tokens obtained from the term *temperature*. In each cluster, the term *temperature* appearing in different rows are used in very similar contexts. The sentences in Cluster 1 (in Table I) have a notion of being able to sense/adjust temperature. Similarly, the sentences in Cluster 2 (in Table II) are providing information about suitable operating temperature for the device. These two clusters demonstrate that BERT and Affinity propagation can successfully cluster sentences based on the contexts. After clustering the related sentences, we manually annotate the cluster which refers to device functionalities. For example, we are interested in Table I, because this cluster is about sensing/adjusting temperature and we are interested in deriving device functionality of sensing temperature. We are not interested in Table II which is about favorable operating temperature for a device.

TABLE II
CLUSTER AFTER APPLYING AFFINITY PROPAGATION TO BERT TOKENS

Cluster 2
Operating Temperature 40ºF (4 ºC) to 100ºF (38 ºC)
Operating Temperature 32°–104°F (0°–40°C)
Operating Temperature –22° to 140°F (–30° to 60°C)
Battery temperature range: 14° to 131°F (–10° to 55°C)
Operating temperature 32° to 104°F (0° to 40°C)
Operating Temperature (F) 0°C +40°C (+32°F +104°F)
Operating Temperature -20 to 60 degree Celsius
Operating Temperature -20 to 45 degree Celsius
Operating Temperature -20 to 45 degree Celsius
The operating temperature or voltage is too low

E. Prediction

This phase predicts the functionalities of a new device. The previous phases remain the same for new devices. After context generation, we predict a suitable cluster for the sentences with key terms. If a sentence is predicted to fall under an annotated cluster, then we conclude the presence of a functionality in the given device, because the annotated cluster and current sentence both are explaining device functionality using similar semantics. Otherwise, we conclude the absence of that functionality. Our framework provides more precise information about device functionality by filtering based on the contextual meaning of transducers. An excerpt of device functionality output extracted by our framework is displayed in Table III. After extracting the device functionalities, we can use them for various security applications, e.g., access control and auditing. One such use case is described below.

TABLE III
EXTRACTION OF DEVICE FUNCTIONALITY FROM OUR APPROACH

Device Functionality/Device	Nest Protect	Nest Thermostat	Arlo Ultra	Samsung Smart Cam
Capture Image/Video			●	●
Detect Motion	●	●	●	●
Produce Infrared Light			●	●
Detect Light	●	●	●	●
Produce Light	●		●	
Capture Sound	●		●	●
Produce Sound	●		●	●
Detect Smoke	●			
Detect CO	●			
Measure Temperature	●	●		
Detect Contact				
Lock and Unlock door				
Control Thermostat		●		

V. USE CASE: APPLICATION TO CONSUMER SECURITY

Armed with the knowledge of device functionality, we now demonstrate how this knowledge can be used to protect consumer security. We demonstrate this in the context of a smart home application.

A. Security Policy Specification

A home has various rooms and areas, each of which is associated with a level of privacy. The owner may have a policy that provides constraints on the placement of devices in a home. The capabilities of a device and the sensitivity of a location will determine if the device can be placed in that location. One such policy involving three devices and a home having private areas is modeled using NIST Next Generation Access Control (NGAC) Model and illustrated in Figure 2. NGAC is useful for modeling policies where access to a resource depends on the properties (attributes) of the subject requesting access, the resources, and the environment. The policies are expressed in terms of relations over the attributes of the subjects, the resources, and the environment.

For our given example demonstrated in Figure 2, the policies express constraints on the placement of devices. The subjects in this case are devices, and the resources that need protection are the various locations in the house. *Nest Cam indoor* and *Nest Protect* are the two devices, and the various locations are *Meeting Room*, *Bathroom*, *Bedroom*, *Front Porch* all of which are shown using dotted boxes. The attributes of the devices correspond to their functionalities. The attributes of the location correspond to their sensitivities. The attributes are shown using solid boxes. The solid arrows from subjects/objects to attributes show the assignment relation which demonstrates possession of that attribute. The solid arrow between solid boxes show containment relation. For example, a device possessing *Image Capture* functionality also has *Visual Capture* functionality. Similarly, *Private Areas* in a home are a part of *All Locations*. The labeled dashed lines between subject attributes and resource attributes indicate allowance (labeled 'A') or prohibition (labeled 'X'). For example, *Audio Capture* is prohibited in *Meeting Room* whereas *Visual Capture* is allowed in it. The default policy allows all, as indicated by the dashed line labeled 'A' from *All Functionalities* to *All Locations*.

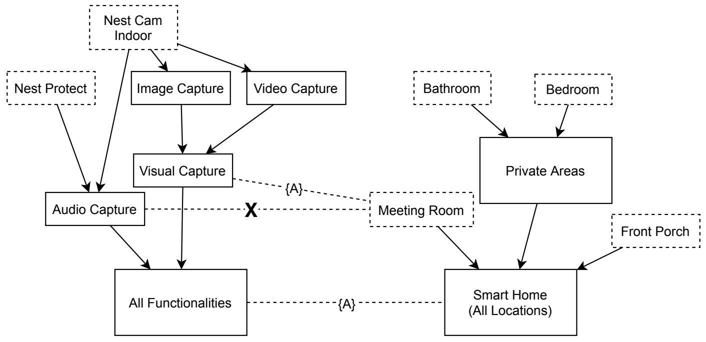


Fig. 2. Privacy policy of a homeowner

B. Security Policy Verification

In large scale applications, the number of devices, the number of functionalities, and the number of constraints may be large. It is important to ensure that the placement of devices conform to the policies and also that policies do not conflict with each other. Consequently, the specification must be analyzed. Manual analysis is tedious and error-prone. NGAC framework does not provide any tool support for automated analysis. Towards this end, we use Alloy [16] to formally analyze the policies expressed by our NGAC model. Alloy is based on first-order logic and has tool support for automated analysis. The key components of Alloy are signatures, facts, and assertions. *Signatures* are basic specifications for a type of object, similar to classes in an object oriented programming language. Signatures contain fields that represent relationships to other signatures. Facts and assertions contain constraints in first-order logic related to the signatures of a model. *Facts* specify constraints that always hold true for any instance of the model, while *assertions* specify constraints that are assumed to follow from the constraints defined by the facts of the model. Assertions must therefore be "checked" to determine if they hold for instances of the model. The Alloy Analyzer can be used to enumerate many instances of a model and check these assertions by converting them to boolean reasoning satisfiability problems. Assertions are verified to a certain depth of scope, meaning all instances of the model for a certain number of each signature type. Within that scope, if any instance invalidates any constraint of the assertion that is being checked, it is presented as a *counterexample* and indicates that the assertion is invalid.

1) Generalized Model: Our NGAC model is created in Alloy as a number of signatures and facts that constrain the structures and their relationships. *EnvGroup* signature represents the environmental attribute groups of the policy, and the *FunGroup* signature represents the functionality attribute groups of the policy.

Listing 1. General model group signatures

```
sig EnvGroup { parents: set EnvGroup }
sig FunGroup {
  parents: set FunGroup
  allows: set EnvGroup
  prohibitions: set EnvGroup }
```

Both of these signatures hold references to the hierarchical structures to which they belong by way of the *parents* field, a set which refers to the immediate parents of the particular group in the hierarchy. The relations between functionality and environmental groups that specify which functionalities are allowed or prohibited in which environmental group(s) are captured in the *allows* and *prohibitions* fields of each *FunGroup*. There are additionally two specialized signatures for both environmental and functionality groups, which act as the root and leaf members of the policy. The *AllEnv* and *AllFun* signatures, shown in Listing 2, exist as the roots in all policy instances. Incidentally, the *allows* and *prohibitions* fields of the *AllFun* group are used to specify the default policy, depending on which field contains the *AllEnv* group. For the policy leaves, we use the *Device* and *Location* signatures, also shown in Listing 2. These are functionally the same as the main signature that they extend, with the exception of the additional *location* field within the *Device* signature, which can be used to specify the device's location in an operating environment.

Listing 2. Root and leaf policy structures

```
one sig AllEnv extends EnvGroup { } { no parents }
one sig AllFun extends FunGroup { } {
  no parents
  allows = AllEnv
  prohibitions = none
}
sig Location extends EnvGroup { } { }
sig Device extends FunGroup {
  location: lone Location
} { }
```

The most significant fact shown in Listing 3, specifies that a device may not be in a location that is prohibited by way of the relationships between the device's functionality ancestors and the location's environmental ancestors, unless there is a corresponding allow.

Listing 3. Fact that constrains where a device may be located

```
fact {
  all d: Device, l: Location | (l in d.prohibitions) => (
       l not in d.location)
  all d: Device, l: Location |
    let prohintersect =
      (l.*parents & d.*parents.prohibitions) |
    let allowintersect =
      (l.*parents & d.*parents.allows) |
    ((allowintersect = none) and
      (prohintersect != none)) => l not in d.location
}
```

2) Instantiating and Validating Policies: We now demonstrate how to translate the exclusive policy depicted in Figure 2 to Alloy syntax. Each environmental group, functionality group, device, and location is represented as a signature which extends the *EnvGroup*, *FunGroup*, *Device*, and *Location* signatures, respectively. Listing 4 provides the Alloy representation for the *PrivateAreas*, *Bedroom*, and *VisualCapture* members of the policy, and the default policy.

Listing 4. Exclusive policy members translated to Alloy syntax

```
// Environmental group
one sig PrivateAreas extends EnvGroup { } { parents =
     AllEnv }
// Location
one sig Bedroom extends Location { } { parents =
     PrivateAreas }
// Functionality group
one sig VisualCapture extends FunGroup { } {
        parents = AllFun
        allows = MeetingRoom
        prohibitions = none
}
// Default policy
fact defaultpolicy {
  AllFun.allows = AllEnv
  AllFun.prohibitions = none
}
```

The Alloy assertion, shown in Listing 5, ensures that the instantiated policy conforms to the security goals specified by the user. The assertion specifies that no device that is related to the *AudioCapture* functionality group shall be in the *MeetingRoom* location.

Listing 5. Example policy assertion

```
assert policyassertion {
  all d: Device | (AudioCapture in d.^parents) =>
    MeetingRoom not in d.location
}
```

When the Alloy Analyzer checks this assertion, it will verify all policy instances within a scope of at most 1 of each signature type against the assertion to determine if it is valid. Because the policy specifies an explicit number of signatures to be considered, there is no need to consider a scope beyond 1; every valid configuration will be iterated over for verification.

3) Conflicts and Counterexamples: When the example policy assertion above is checked, Alloy Analyzer produces counterexamples that represent configurations that are in conflict with the constraints of the policy assertion, such as one that is depicted in Figure 3. In this configuration that Alloy automatically instantiated, the *NestCam* is located in the *MeetingRoom*, despite the fact that the policy assertion specifies that no device belonging to the *AudioCapture* functionality group should be located in the *MeetingRoom*.

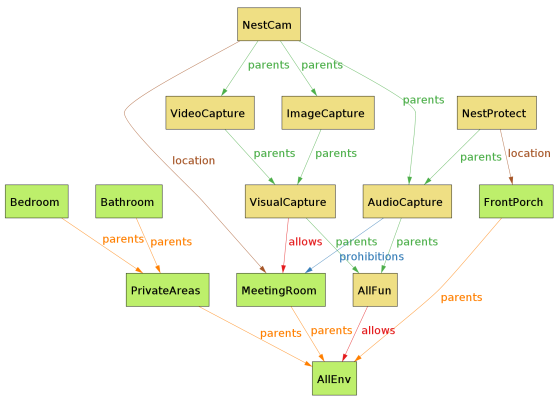


Fig. 3. A counterexample enumerated by Alloy Analyzer that shows a conflicting configuration for an exclusive policy

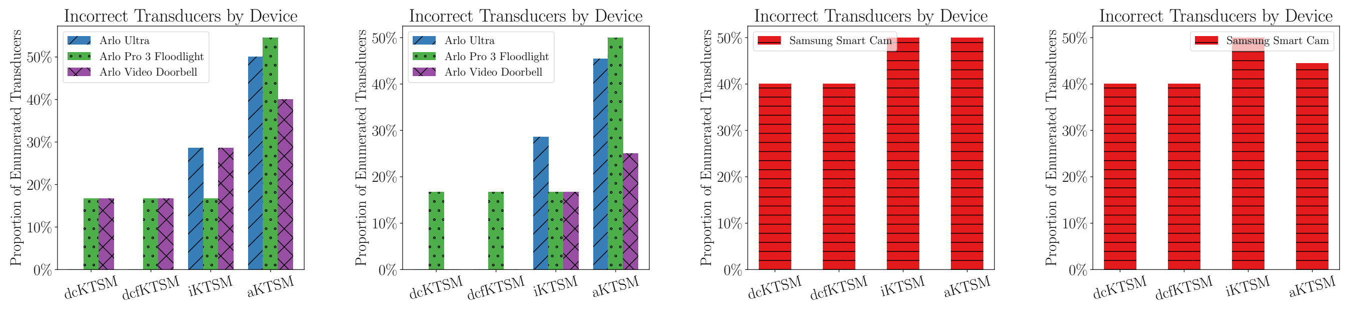
The policy conflict shown in this counterexample is the fact that the *NestCam* has *AudioCapture* functionality, yet is located in the *MeetingRoom*. However, a direct conflict exists in this configuration, since the *NestCam* is also explicitly allowed in the *MeetingRoom*. Using assertions such counterexamples can be enumerated. Once all location fields of all devices are specified, only one configuration can exist, and the same policy assertion can be checked to ensure that the chosen configuration does not conflict with the constraints of the policy. For example, if the *NestProtect* were specified to be located in the *Bedroom*, and the *NestCam* were specified to be located on the *FrontPorch*, then the assertion would be evaluated without any counterexamples.

VI. Experimental Results

A. Extraction Efficiency

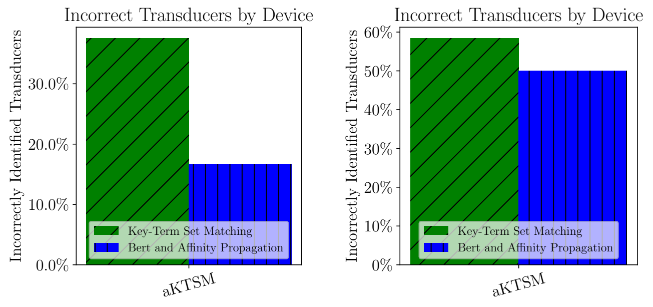
Figure 4 portrays our results for incorrect transducer identification for Arlo and Samsung devices with the various key term set matching approaches, with and without using the N-gram approach. In Figure 4a and Figure 4b, we see that Arlo video doorbell does not show any incorrect identification of transducer on dcKTSM and dcfKTSM with N-gram approach as opposed to when only the algorithms were applied without the N-gram approach. Similarly, the false-positive rate also decreases for the Arlo video doorbell with iKTSM when N-gram is applied. Also, we can see a slight improvement in false-positive rate for Samsung Smart Cam with aKTSM approach on when N-gram is applied as opposed to when N-gram is not applied. This preliminary study concludes that detecting the true context in which the transducer is mentioned in vendor material helps to robustly enumerate the transducer and map them to their respective functionalities.

Figure 5 compares result of aKTSM augmented with BERT for contextual embedding generation and Affinity propagation for clustering with that of aKTSM without BERT and Affinity propagation. The initial algorithms without BERT and Affinity propagation give false positive rates of 37.5% and 58.334% respectively for Nest Cam Indoor and Samsung Smart Cam



(a) Without N-gram (b) With N-gram (c) Without N-gram (d) With N-gram

Fig. 4. For Arlo (fig. 4a and fig. 4b), and Samsung(fig. 4c and fig. 4d) devices, grouped by vendor and for each vendor grouped by KTSM algorithm the proportion of matching transducers that are incorrectly identified with and without N-gram



(a) Nest Cam Indoor (b) Samsung Smart Cam

Fig. 5. Incorrect transducer by device for Nest Cam Indoor and Samsung Smart Cam Video Surveillance with aKTSM w/o BERT and Affinity Propagation and aKTSM augmented with BERT and Affinity Propagation

Video Surveillance which reduced to 16.667% and 50% with the use of BERT and Affinity propagation.

We apply our framework on detecting two sets of transducers (Temperature Sensor and Lock) and map them to respective device functionalities. We test our approach on aKTSM algorithm, because aKTSM algorithm was showing high false positive rate with key term set matching approach.

TABLE IV
CATEGORIZATION OF DEVICE EXTRACTION ACCURACY BASED ON CONTEXTUAL EMBEDDING OF KEY TERMS

Device Specific Accuracy	Vendor Specific Accuracy	Average Accuracy
Nest Protect: **100%**	Nest: **92.85%**	All vendors: **76.31%**
Nest Yale Lock: **100%**		
Nest Camera: **75%**		
Nest Thermostat: **100%**		
Arlo Pro 3 Floodlight: **83.33%**	Arlo: **50%**	
Arlo Ultra Cam: **60%**		
Arlo Video Doorbell: **0%**		
Samsung Smart Cam: **100%**	Samsung: **100%**	

We conduct our experiment on eight smart homes IoT devices which lead to ${}^{8}C_{2}$ i.e., 28 combinations, with a split ratio among devices of 75% (six devices) for training purpose and 25% (two) for testing purpose. We apply our framework based on BERT and Affinity Propagation to annotate the cluster that is about sensing/adjusting temperature using a training device. For each new testing device, we predict if any of the sentences with the word ‘temperature’ mentioned falls into our annotated cluster. If a sentence from the new device falls under our annotated cluster, we can conclude with high confidence that the device into consideration should have a temperature sensor. While training our model on some combination of devices, it yields no significant cluster (cluster that explains device behavior) as output. Such combination have been excluded from our result. We get a total of 19 combinations where we have a specific cluster explaining device behavior. Table IV displays the accuracy of our approach categorized into vendors. We only have one Samsung device into consideration. However, we can see Nest devices performing significantly well with 92.85% of vendor accuracy whereas Arlo vendor only have 50% accuracy. This difference in vendor accuracy stems from the clarity and conciseness with which the documents and materials pertaining to specific devices are provided by vendors. Nest provides more thorough documents for their device which yields to higher accuracy.

B. Verification Efficiency

Thanks to Alloy’s use of a highly efficient SAT solver, many different instantiations of an abstract model can be iterated over extremely rapidly. For example, running assertions to validate the soundness of the general model (e.g., that any instance of the model is acyclic) with an Alloy scope of 30, takes, on average over five trials, slightly over one second to complete. It is worth noting that these validations of the general model with such a scope produce an enormous number of instantiations, many of which would be far more complex than any that would actually be found in a smart home environment. For context, such an assertion at a scope of 30 creates 1.3 million Boolean satisfiability clauses.

In practice, a real-world application of the Alloy policies described in this work would only require validation of assertions of more specific configurations. These configurations, as described previously, are effectively bound to a scope of one, where there is at most one of each type of signature in any given instantiation. For example, the policy assertion shown in Listing 5, which represents the security goals of a user as they are represented by a configuration, can find a counterexample in under 10 milliseconds, on average, and produces only 7,145 clauses in the resulting Boolean expression. Even in the case of very loosely defined policies, where no specific locations are assigned to devices, and therefore a great number of instances are created, Alloy can iterate over these resulting instances extremely quickly. Note that, even the most complex configurations found in a smart home environment could be analyzed in a reasonable amount of time.

In a smart home environment where the device configurations do not change too often, the use of Alloy to monitor an IoT environment in real-time is reasonable, and even more so for proactive analysis. Alloy could be presented with the desired configuration by a user ahead of its installation, or an automated system could provide Alloy with the current configuration. In either case, Alloy would be able to efficiently notify the user of any potential violations to their security goals by validating the configuration against the user-defined policy. With Alloy, users can encapsulate their security goals in a policy that can be asserted against the IoT environment periodically, and in real-time as devices belonging to different functionality groups are placed in different locations.

VII. Conclusion and Future Work

We propose a context-based approach to detect transducer and the respective functionalities of IoT devices using vendor materials. We also show how the functionality of the devices plays an important role in an example smart home application. Specifically, we show policies where the placement of devices is contingent on the sensitivity of the location and the functionalities of the devices, and how such policies can be specified using NIST NGAC and automatically analyzed to check for conformance and consistencies using Alloy. Our future work involves expanding our scope to include more IoT devices from various vendors and also investigate the problem in the context of industrial IoTs. We would also like to investigate more use cases involving device functionalities, including its application to security audits.

References

[1] "IoT Growth Demands Rethink of Long-Term Storage Strategies, says IDC." [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prAP46737220

[2] "Nest Protect 2ng Gen - Installation and Tech Specs - Google Store." [Online]. Available: https://store.google.com/us/product/nest_protect_2nd_gen_specs

[3] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network Traffic Classifier With Convolutional and Recurrent Neural Networks for Internet of Things," *IEEE Access*, vol. 5, 2017.

[4] T. OConnor, R. Mohamed, M. Miettinen, W. Enck, B. Reaves, and A.-R. Sadeghi, "HomeSnitch: Behavior Transparency and Control for Smart Home IoT Devices," in *WiSec*, 2019.

[5] W. Zhang, Y. Meng, Y. Liu, X. Zhang, Y. Zhang, and H. Zhu, "HoMonit: Monitoring Smart Home Apps from Encrypted Traffic," in *CCS*, 2018.

[6] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray, "Behavioral Fingerprinting of IoT Devices," in *ASHES*, 2018.

[7] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma, "IoT SENTINEL: Automated Device-Type Identification for Security Enforcement in IoT," in *ICDCS*, 2017.

[8] S. Marchal, M. Miettinen, T. D. Nguyen, A. Sadeghi, and N. Asokan, "AuDI: Toward Autonomous IoT Device-Type Identification Using Periodic Communication," *IEEE J SEL AREA COMM*, vol. 37, 2019.

[9] J. Ortiz, C. Crawford, and F. Le, "DeviceMien: Network Device Behavior Modeling for Identifying Unknown IoT Devices," in *IoTDI*, 2019, pp. 106–117.

[10] V. Bhosale, L. De Carli, and I. Ray, "Detection of Anomalous User Activity for Home IoT Devices," in *IoTBDS*, 2021.

[11] A. Hamza, D. Ranathunga, H. H. Gharakheili, M. Roughan, and V. Sivaraman, "Clear as MUD: Generating, Validating and Applying IoT Behavioral Profiles," in *IoT S&P*, 2018.

[12] A. Dolan, I. Ray, and S. Majumdar, *Proactively Extracting IoT Device Capabilities: An Application to Smart Homes*, 2020.

[13] P. Kumar, "An Introduction to N-grams: What Are They and Why Do We Need Them?" Oct. 2017, section: AI. [Online]. Available: https://blog.xrds.acm.org/2017/10/introduction-n-grams-need/

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv:1810.04805 [cs]*, 2019, arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805

[15] D. Ferraiolo, R. Chandramouli, R. Kuhn, and V. Hu, "Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC)." New York, NY, USA: Association for Computing Machinery, 2016.

[16] D. Jackson, "Alloy: A Language and Tool for Exploring Software Designs," vol. 62, no. 9, 2019.

[17] A. Hamza, D. Ranathunga, H. H. Gharakheili, T. A. Benson, M. Roughan, and V. Sivaraman, "Verifying and Monitoring IoTs Network Behavior using MUD Profiles," *arXiv:1902.02484 [cs]*, 2019, arXiv: 1902.02484. [Online]. Available: http://arxiv.org/abs/1902.02484

[18] "Combining MUD Policies with SDN for IoT Intrusion Detection," Budapest Hungary.

[19] "Extending MUD Profiles Through an Automated IoT Security Testing Methodology," vol. 7.

[20] A. Otebolaku and G. M. Lee, "A Framework for Exploiting Internet of Things for Context-Aware Trust-Based Personalized Services," *Mobile Information Systems*, vol. 2018, 2018. [Online]. Available: https://www.hindawi.com/journals/misy/2018/6138418/

[21] L. Yao, B. Benatallah, X. Wang, N. K. Tran, and Q. Lu, "Context as a Service: Realizing Internet of Things-Aware Processes for the Independent Living of the Elderly," in *Service-Oriented Computing*, Q. Z. Sheng, E. Stroulia, S. Tata, and S. Bhiri, Eds. Cham: Springer International Publishing, 2016.

[22] "Nest Cam Indoor - Home Security Camera - Google Store." [Online]. Available: https://store.google.com/us/product/nest_cam

[23] "Nest Cam Indoor - Installation and Tech Specs - Google Store." [Online]. Available: https://store.google.com/us/product/nest_cam_specs

[24] Arlo Smart Home, "Arlo Video Doorbell: How To Install," Nov. 2019. [Online]. Available: https://www.youtube.com/watch?v=GMogHU9MyPI

[25] "Nest Learning Thermostat - Installation and Tech Specs - Google Store." [Online]. Available: https://store.google.com/us/product/nest_learning_thermostat_3rd_gen_specs

[26] R. Horev, "BERT Explained: State of the art language model for NLP," Nov. 2018. [Online]. Available: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

[27] "SpeechRecognition." [Online]. Available: https://pypi.org/project/SpeechRecognition/

[28] "Nest x Yale Lock - Key-Free Smart Deadbolt - Google Store." [Online]. Available: https://store.google.com/us/product/nest_x_yale_lock

[29] "Video Doorbell — Arlo Home Security — Arlo." [Online]. Available: https://www.arlo.com/en-us/products/arlo-video-doorbell/default.aspx

[30] "Arlo Pro 3 Floodlight Camera — Arlo Wireless & AC-Powered Security Cameras." [Online]. Available: https://www.arlo.com/en-us/products/arlo-pro-3-floodlight.aspx

[31] "Samsung Smartcam Video Surveillance Camera: SNH-P6410BN — Samsung US." [Online]. Available: https://www.samsung.com/us/smart-home/security/cameras/smartcam-hd-pro-1080p-full-hd-wifi-camera-snh-p6410bn/

[32] "hanxiao/bert-as-service: Mapping a Variable-Length Sentence to a Fixed-Length Vector Using BERT Model." [Online]. Available: https://github.com/hanxiao/bert-as-service

[33] R. Vink, "Algorithm Breakdown: Affinity Propagation - Ritchie Vink." [Online]. Available: https://www.ritchievink.com/blog/2018/05/18/algorithm-breakdown-affinity-propagation/