Multi-platform Application Interaction Extraction for IoT Devices

Zhao Chen, Fanping Zeng, Tingting Lu, Wenjuan Shu
School of Computer Science and Technology
University of Science and Technology of China
Email: chen95@mail.ustc.edu.cn, billzeng@ustc.edu.cn, {tingtlu, shuwj}@mail.ustc.edu.cn

***Abstract*—IoT devices used in smart home have become a fundamental part of modern society. Such devices enable our living space to be more convenient. This enables human interaction with physical environment, also happens between two applications or others third-party rules in addition, and causes some unexpected automation, even causes safety concerns. What's worse is that attackers can leverage stealthy physical interactions to launch attacks against IoT systems or steal user privacy. In this paper, we propose a tool called IoTIE that discovers any possible physical interactions and extract all potential interactions across applications and rules in the IoT environment. And we present a comprehensive system evaluation on the Samsung SmartThings and IFTTT platform. We study 187 official SmartThings applications and 98 IFTTT rules, and find they can form 231 hidden inter-app interactions through physical environments. In particular, our experiment reveals that 74 interactions are highly risky and could be potentially exploited to impact the security and safety of the IoT environment.**

***Index Terms*—IoT, multi-platform, application analysis and interaction extraction**

I. Introduction

In recent years, the Internet of Things(IoT) platforms and applications have been developed rapidly, and smart home applications have entered the lives of a large number of users which have made users' lives more intelligent, efficient and convenient. For example, water flow sensors and smart meters are used to improve energy efficiency, motion sensors and door locks which are connected to the Internet make it easier to control doors. In order to capture the large-scale market of smart homes, many IoT platforms have provide convenient device control mechanisms, such as Samsung's SmartThings [1] and Apple's Home Kit [2]. However some literature researches have shown that these smart applications are insufficient to protect users' security and privacy [3–5], and often cause users to fall into an insecure and unsafe situation. At the same time, interactive interfaces between a device and a third-party Trigger-Action platform are also designed to support user-defined device control rules, but this further exacerbates the security risks of the IoT system.

SmartThings: Current IoT systems generally consist of three main components: (1) a hub and some sensing devices, (2) a closed-source backend platform, and (3) a companion application for controlling home devices. Apart from these, SmartThings' backend platform further provides an online coding IDE and application emulation tool. The application which called **SmartApp** can subscribe device events or system variable to trigger the corresponding actions. The actions can be commands to a single or multiple devices, such as turning on the lights. It can also be an action to send a text message or a photo.

Due to the coarse-grained capability management policy, there are security issues such as privilege abuse and stealing device pin code on the SmartThings platform[3]. Because smart home applications are not sparsely distributed, they often affect each other and cause security risks. For example, one application turns on a heater in a room, another one detects smoke and then opens the windows. These two applications create insecure interactions due to the same environment. Ding et al. designed and implemented IotMon for this problem, which only can be used to detect the applications of the SmartThings platform [6].

IFTTT: The Trigger-Action platform is mainly used for user-defined control rules, including IFTTT [7] and Zapier [8]. These platforms allow users to customize device control rules, bind it to a specific devices and trigger actions of the device. For example, a rain warning is pushed to a user's mobile device based on a third-party weather service. In addition, when detecting an action, the device may capture a photo and post it to a social websites such as Twitter, thereby causing leakage of user privacy which means these platforms also introduce a long-term security risk[9]. Recently, the quantity of users using third-party platforms has gradually increased, this means cross-platform application interaction analysis is urgently needed.

Since the IoT platforms such as SmartThings support user-defined device control logic, and the rules provided by the third-party Trigger-Action platform can easily control authorized devices, the interactions between applications and rules due to the common physical environment are likely to bring more safety concerns. Performing the security analysis was challenging because the SmartThings platform is a closed-source system and the IFTTT rules are not standard. This makes it difficult to implement a unified analysis method to deal with both SmartApps and IFTTT rules. Thus, we propose an unified interaction extraction method for multi-platform IoT applications(rules) due to the shared physical environment, and implement an IoT Interaction Extraction(**IoTIE**) tool. Our main contributions are as follows:

1) We implement a prototype system to extract interaction for multi-platform IoT applications, which converts IFTTT rules into SmartApps and analyzes the interactions they formed.
2) We collect 244 rules on the IFTTT platform that can be used to control SmartThings devices, and 187 smartapps

978-1-7281-2583-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPADS47876.2019.00151

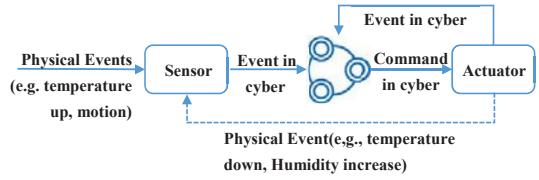

Fig. 1. Chain of events in an IoT System.

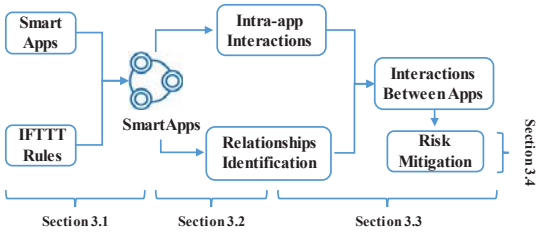

Fig. 2. System Overview.

from SmartThings' website. The quantity of interactions formed by only 120 smartapps is 177; after adding 41 IFTTT rules, they formed 231 interactions in total, and we find 74 high risky interactions.

3) We evaluate performance of **IoTIE** on SmartApps and IFTTT rules, and find that the average analysis time per application is 76.59 milliseconds, which is reasonable.

The remainder of this paper is organized as follows. Section II introduces the architecture design of **IoTIE**. Section III introduces the detailed design of each stage. Section IV introduces the results of the experimental evaluation while Section V introduces the related works. And the last section makes a conclusion for this paper.

II. System Overview

Figure 1 shows a high-level view of an event-driven IoT system [10]. Briefly, sensors convert the physical environment into data in the information system and generate events which are passed to the application subscribed to the events and trigger commands of a specified devices. The device causes an impact on the physical environment after executing the commands, such as heating, humidifying.

Figure 2 shows the process of our **IoTIE**. **IoTIE** consists of four phases: (I) application collection and rule conversion. (II) Intra-interaction and relations between capabilities and environments extraction. (III) Interaction generation between applications and rules. (IV) Interaction analysis and risk mitigation.

A. Application collection and rule conversion

In this phase, we collect IoT applications and third-party rules described by using natural languages, and convert rules into standard IoT application code by using the NLP method. Therefore, a unified method can be designed to analyze rules and IoT applications to extract interactions between applications and rules.

B. Intra-interaction and relations between capabilities and environments extraction

This phase consists of two tasks: (I) constructing the dependency between the event trigger conditions and the event handlers, and extracting the (trigger,action) tuples within the IoT application. Intra-interaction can be extracted by using static program analysis. The example of SmartApp is shown in Listing 1. It includes four main methods of *definition*, *preferences*, *installed*, and *update*. (II) extracting the description information within the application, such as the application name, description, and annotations. Because these pieces of information are likely to describe the physical environment affected by the triggered action of the application. Such physical environments include temperature, humidity, light, motion, and the others. Thus, the relations between the capabilities applied for by the application and the physical environments are constructed.

```
definition(
  name: "My First SmartApp",
  namespace: "mygithubusername",
  author: "Peter Gregory",
  description: "This is my first SmartApp. Woot!"
)
preferences {
  section("Turn on when motion detected:") {
    input "themotion", "capability.motionSensor",
    title: "Where?"
  }
  section("Turn on this light") {
    input "theswitch", "capability.switch"
  }
}
def installed() {
  initialize()
}
def updated() {
  unsubscribe()
  initialize()
}
def initialize() {
  subscribe(themotion, "motion.active",
    motionDetectedHandler)
  subscribe(themotion, "motion.inactive",
    motionStoppedHandler)
}
def motionDetectedHandler(evt) {
  log.debug "motionDetectedHandler called: $evt"
  theswitch.on()
}
def motionStoppedHandler(evt) {
  log.debug "motionStoppedHandler called: $evt"
  runIn(minutes, checkMotion)
}
```

Listing 1. SmartApp example

In our design, the 11 physical environments extracted from description in literature [6] and 2 newly found environments: color and voice are referred to, and the NLP method is used to calculate the similarity between the description information and the physical environment in each application, and so that a possible relations between capabilities and environments is determined based on the similarity.

C. Interaction generation between applications and rules

In this stage, we design an algorithm to generate the interaction between two IoT applications or the interaction between an application and a rule, using the intra-interaction and the relations between the capabilities and the physical environments in stage II.

D. Interaction analysis and risk mitigation

After obtaining the interactions between applications and rules, we discover the risk interactions according to the risk of the action in the interactions. For the found risky interactions, we propose two strategy to mitigate risk for application developers and users.

III. System Implementation

A. Application collection and rule conversion

We collect all open-sourced smart applications as our research object from Samsung's SmartThings platform. And we choose IFTTT as our Trigger-Action rule platform, because it already has 11 million users and 54 million rules, and it directly provides control rules for SmartThings devices. We collect 187 SmartApps and 244 IFTTT rules suitable for SmartApps in April 2019, but not all of them can be analyzed. In order to analyze applications and rules in a unified manner, we convert the IFTTT rules into the form of SmartApp for subsequent analysis. Since IFTTT rules are described by natural languages, and they have no standard format, which brings great challenges to the conversion at this stage.

IFTTT rule conversion: The goal of this process is to extract the trigger conditions and trigger actions of the IFTTT rules, and convert it into an application supported by the SmartThings platform. For example, for the rule "Start Skybell video record when motion is detected", we construct an application that subscribes to the event "motionSensor.active" and triggers the "videoCamera.on()" action when an action is detected. Thus, we design a code template with some blank to be filled. The name and description of the application are directly generated by the rule description, the corresponding sensing device's name is fixed, the device-subscribed event, the triggered action and the capability applied for are generated by using the NLP technology.

There are generally two category of IFTTT rules: one is action execution, such as "Lock your SmartThings device", and the second is rules containing keyword "when", which describes the event trigger conditions and the corresponding to-be-taken actions. In order to comply with the event semantics of the IoT platform, we only select 98 rules containing keyword "when" from 244 IFTTT rules. We use the word segmentation methods in NLTK [11] and remove stop words to divide the description into two parts. One is the condition part after keyword "when" and the other part is the action before keyword "when". We traverse all the token resulted from NLTK, and check whether the word or word list is an action or condition according to its part of speech and frequency of occurrence.

However, after implementing the foregoing method, we find that the final result is not accurate, so we manually check the generated application, to ensure that it can be simulated in the online IDE of SmartThing correctly.

B. Intra-interaction and relations between capabilities and environments extraction

We find all the applications subscribe to the events of devices and then trigger the action of other devices. The devices need to match the application with the corresponding capabilities during the installation. Only the device with the designated capabilities can obtain the sensed data and perform corresponding actions. The configuration information of these devices can be directly extracted from the SmartApps' source code, including the event subscribed by the application and the corresponding event handlers.

It can be seen from the example SmartApp code shown in Listing 1, SmartApp's infomation such as name, description is passed in by using *definition* method. The *preference* section claims all the capabilities and inputs of the application, this section is configured by device users. What's more, these configurations may be set incorrectly. And in the *installed* method, the *subscribe* method receive the parameters like the name of subscribed device, the trigger event and the event handler. We use the syntax analysis class **CompilationUnit** of the Groovy language to generate an abstract syntax tree(AST) of the application's source code, and traverse the AST to complete the extraction of the intra-interaction within the application.

We extract the list of capabilities and inputs from the *preference* method of AST, which contains sensor name and capabilities. The *subscribe*, *runIn* and *schedule* methods define the trigger conditions and actions of the sensors, so we traverse all these three methods to extract the trigger and action tuples. After extracting the triggers and actions, we link these information as our intra-interaction whose format is like $\langle capability.motionSensor : motion.detected, capability.switch : on \rangle$.

We also extract the name, description and the annotations of the application, because these pieces of information can determine the relations between the capabilities and the physical environments, for example, "Turn your lights on when it gets dark" means when it's sunset (about 6 o'clock pm), turn the light on. This description indicates that the capability *capability.switch* corresponds to light, and of course *capability.switch* can also have an impact on other environments, such as temperature.

As mentioned in literature [6], the environment involved in IoT devices includes temperature, humidity, light, location, motion, smoke, and leakage. In addition, there are four system environment variables: time, location mode, switch, and lock, 11 physical channels in all. Based on the method to extract physical channels in the literature [6], in addition to the above 11 channels, we newly discover 2 physical environments such as color and voice. Therefore, in order to determine the relations between capabilities and the physical environments,

we directly use the 11 environment channels and the newly discovered two channels. And then we calculate the similarity of the application name, description, annotations and enviroments by using the CountVectorizer provided in sklearn [12].

C. Interaction generation between applications and rules

Use the above extracted intra-interaction and the relations between capabilities and the environments, we can extract the interactions formed by applications and rules. We use a trigger-action tuple to represent a intra-interaction, where trigger is composed of capability and corresponding trigger conditions, and action is composed of capability and corresponding commands. The algorithm is shown in Algorithm 1. The algorithm first reads all the intra-interactions and the ralations between the capabilities and the environments, and then compares whether the two applications can form interactions in the same environment, and finally outputs the generated interactions between two applications or rules. For the system channels, we treat it like literature [6].

Algorithm 1 Algorithm for Extracting Interactions cross mutil-Apps and Rules

Input: intraactions_app; intraactions_rule; capabilityChannelMap
Output: interactions

for each l in $intraactions\_app$ **do**
  for each r in $intraactions\_rule$ **do**
    if $l == r$ **then**
      continue
    end if
    $lCapa = l.acCapa$
    $rCapa = r.triCapa$
    $channelsL \leftarrow capabilityChannelMap[lCapa]$
    $channelsR \leftarrow capabilityChannelMap[rCapa]$
    $chs \leftarrow channelsL \cap channelsR$
    if $chs! = \emptyset \quad and \quad lCapa == rCapa$ **then**
      for each channel in chs **do**
        $interactions \leftarrow [l, channel, r]$
      end for
    end if
  end for
end for

D. Interaction analysis and risk mitigation

We categories the discovered interactions by tagging higher-risk commands, devices, and capabilities (such as opening windows, turning on heaters, and sending photos) to identify risky interactions. These high-risk interactions not only create undesired automation, but also bring security and safety issues. Finally, based on these high-risk interactions, two strategies for risk mitigation are proposed.

Strategy I: Implementing the **IoTIE** on the indoor hub. When a new application is installed, it can detect whether the installed applications and the to-be-installed applications or rules can form insecure interactions. We will continue to study in this direction and hope to implement it.

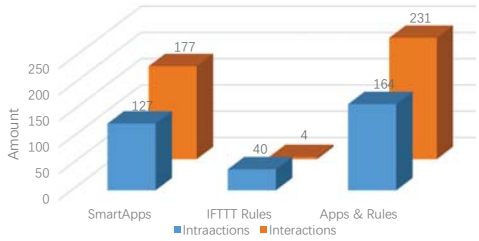

Fig. 3. Amount of Different Dataset's Interaction Dependency.

Strategy II: Let application developers pay more attention to this kind of problem. In the development process, try to combine the conditions of multiple sensor data and events to enhance the trigger conditions of the commands. What's more, we can do this by code instrumentation, and it is also compatible with older versions of the applications.

IV. Experimental evaluation

A. Setup

We obtain all 187 applications in the Samsung SmartThings platform, 14 of which are unable to analyze because of dynamic pages; 48 SmartApps are services for external services, there is no to-be-searched interaction defined in this paper, and 5 applications are about voice. The remaining 120 applications can be successfully analyzed. We collect all the 244 rules for the SmartThings applications from the IFTTT platform, and choose 98 rules with keyword "when", but some of the rules that are not supportted by SmartThings platform and some rules' semantic are the same, so finally the number of rules that are successfully converted into SmartApp is 41.

We conduct our experiments on a machine with Intel Core i5-4210M 2.60GHz CPU, 16.0GB RAM and Ubuntu 1604 operating system.

B. Experimental Result

To better assess our tools, we conducted experimental analysis from three perspectives.

- **RQ 1:** How effective is **IoTIE** in extracting interactions cross multi-platforms?
- **RQ 2:** After risk analysis of the interactions, what problems do we find?
- **RQ 3:** Compared with other tools, what are our strengths?

RQ 1: The Effectiveness to find the interactions. We conducted four groups of experiments. The datasets contains 120 SmartApps, 41 rules and mixed together respectively. The last group is that the two sides of an interaction are formed by SmartApps and rules.

The results are shown in Figure 3. 177 interactions were formed by 120 SmartApps; 4 interactions were formed by 41 rules; There are 231 interactions in the dataset of 120 SmartApps and 41 rules. It can be seen that when rules are added, there are more interactions that users do not expect

are formed, and some interactions are likely to cause safety issues. As we can see, our tool **IoTIE** is very effective for detecting interactions formed by multi-platform applications. And it takes less time, about 76.59 ms per application as shown in Table I.

TABLE I
TOTAL TIME OF INTERACTION ANALYSIS

	Amount	time (ms)	mean time(ms)
SmartApps	120	9174	76.45
IFTTT rules	41	752	18.34
SmartApps and rules	161	12331	76.59

RQ 2: Risk Analysis. We analyze the discovered interactions between applications and rules. We consider that the interactions whose last action includes "on, open, set" are high risky, because these operations usually trigger some things on, and then impact other things. Finally, we find 74 high risky interactions, some are shown in Table IV-B. The number of risky interactions formed by temperature, color, lock and location was 33, 14, 3 and 24 respectively.

No.1-5 is the interaction formed by SmartApps. In 1st interaction, the change of locationMode causes the indoor air conditioner or heater to be turned on. After the temperature rises, the window is opened or the heater is continuously turned on. No.6-8 are triggered by the application, and then the rule forms a high-risk interaction, in contrast No.9-10 are formed by rules and applications. This table shows that the interaction of applications and rules does create high-risk interactions that are unsafe or unsecurity for users.

Fortunately, we find that some rules are protective. If some dangerous event happened, it will send a warning notice to the user. But we also find some rules about voice will cause more dangerous situations. For example, if a song is configured as an alarm ring, smart speaker will misunderstanding it into a malicious commands, like "open the door" when there is no person in the house as described in the literature [13].

RQ3: Tool Comparison. Our tool is compared with IoTMon [6], ProvThings [14], and HomeGuard [15], as shown in Table II. **IoTIE** has more features than previous work, and can analyze single-application, multi-applications, and multi-platform applications, and there is no runtime intervention. ProvThings analyzes malicious behavior between applications by building application runtime log graphs. HomeGuard can detect CAI threats across multiple applications. IoTMon only supports analysis of applications on single platform.

C. Discussion

During the experiment, we find that the similarity calculation method used to extract relations between the capabilities and the physical environments resulted in inaccurate results. Therefore, we manually checked the data and removed the inaccurate results.

When converting IFTTT rules, since the description of the rule has no standard, the generated code needs to be verified to ensure that it can be simulated in the online IDE of SmartThing correctly. And the events and actions in the rules are not very compatible with the capabilities in the SmartThings platform. We plan to use more semantically NLP techniques to analyze rules.

TABLE II
TOOL COMPARATIONS

	Prov Things	Home Guard	IoT Mon	IoT IE
Single Application	✓	✓	✓	✓
Multiple Applications	✗	✓	✓	✓
No runtime Intervention	✗	✓	✓	✓
Multiple Platforms	✗	✗	✗	✓

Lastly, the **IoTIE** prototype system we implemented is currently only available for SmartThings and IFTTT platforms, and we plan to support more platforms such as Zapier [8] and Microsoft Flow [16].

V. RELATED WORK

In recent years, more and more researches focus on the security and sefety of the IoT from two aspects. One is about sensitive data protection. Celik et al. [4] designed a sensitive data flow analysis tool called SAINT, which can analyze the number of privacy leakages, but the tool can only analyze single applications. Fernandes et al. [5] designed a tag-based information flow control system that uses stain tracking to limit the flow of sensitive data. Jia et al. [17] proposed a context-based permission system for appified IoT platforms that provides contextual integrity by supporting fine-grained context identification for sensitive actions.

Another is the interactive security analysis of IoT applications. Ding et al. [6] proposed a tool that only extracted the interactions between SmartApps and analyzed their risks. Chi et al. [15] accurately extracted the automation semantics from the IoT applications by constructing the symbol execution module, then comprehensively considered the semantics of different IoT applications, and evaluated the interactions between applications. Celik et al. [18] developed a static analysis system, SOTERIA, to verify whether IoT applications follow existing security, and function attributes. Zhang et al. [19] attacked the Speech Recognition Systems(SRS) of the smart speaker using an ultrasonic command that could not be heard by the human, so that the speaker can be completely controlled using any command. Roy et al. [20] increased the attack range to $25ft$, made the attack easier to implement and provided a defense solution for SRS.

VI. SUMMARY AND CONCLUSIONS

We have proposed a tool that can analyze applications across multiple IoT platforms. It can extract unexpected interactions between applications due to environmental overlap. We analyzed 187 SmartThings applications and 98 IFTTT rules and found 231 interactions they formed, the quantity of SmartApp

TABLE III
SOME RISKY INTERACTIONS

No.	Trigger1 Capability	Action1 Capability	Potential Devices	Channel	Trigger2 Capability	Action2 Capability	Potential Devices
1	locationMode: mode	switch: on	AC, heater	temp	switch: switch.on	switch: on	window
2	motionSensor: motion	locationMode mode	multiple Devices	location	locationMode: mode	lock: each	lock
3	motionSensor: motion.active	switch: on	toaster	temp	switch: switch.on	switch: on	window
4	time: time	switch: on	Coffee Machine	temp	switch: switch.on	locationMode: mode	lock
5	contactSensor: contact.open	switch: on	bulb	light	illumiMeasure: illuminance	switch: on	curtain
6	switch: switch	locationMode: mode	app	location	locationMode: mode	thermostat: setHeating	rule[1]
7	thermostat: mode	thermostat: off	thermostat	temp	tempMeasurement: switch.on	switch: on	rule
8	switch: switch	locationMode: mode	AC	temp	tempMeasurement: temperature	switch: on	rule
9	smokeDetector: smoke.detected	thermostat: mode	rule	temp	tempMeasurement: temperature	switch: on	curtain, window
10	lock: lock.unlocked	switch: on	rule	smoke	switch: switch	locationMode: mode	app

[1] "rule" means this is formed by IFTTT rule.

triggering rule and rule triggering SmartApp are 28 and 22 respectively. The experimental results show that our tool is effective for cross-platform application interaction extraction. So, we hope this research can attract the attention of relevant companies and developers, and expect them to provide a security and safety IoT environment for users.

ACKNOWLEDGMENT

This work is supported partly by the National Key R&D Program of China 2018YFB0803400, 2018YFB2100300 and National Natural Science Foundation of China (NSFC) under grant 61772487.

REFERENCES

[1] "Smartthings," https://www.smartthings.com/.
[2] "Homekit," https://developer.apple.com/homekit/.
[3] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 636–654.
[4] 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)*, 2018, pp. 1687–1704.
[5] 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)*, 2016, pp. 531–548.
[6] 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, 2018, pp. 832–846.
[7] "Ifttt," https://ifttt.com/discover.
[8] "Zapier," https://zapier.com/.
[9] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decoupled-ifttt: Constraining privilege in trigger-action platforms for the internet of things," *arXiv preprint arXiv:1707.00405*, 2017.
[10] 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*. ACM, 2018, pp. 191–203.
[11] E. Loper and S. Bird, "Nltk: the natural language toolkit," *arXiv preprint cs/0205028*, 2002.
[12] "Countvectorizer," https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.
[13] X. Yuan, Y. Chen, Y. Zhao, Y. Long, X. Liu, K. Chen, S. Zhang, H. Huang, X. Wang, and C. A. Gunter, "Commandersong: A systematic approach for practical adversarial voice recognition," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 49–64.
[14] 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.
[15] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling." *CoRR*, 2018.
[16] "Microsoft flow," https://flow.microsoft.com/.
[17] Y. J. Jia, Q. A. Chen, S. Wang, A. Rahmati, E. Fernandes, Z. M. Mao, A. Prakash, and S. J. Unviersity, "Contexiot: Towards providing contextual integrity to appified iot platforms." in *NDSS*, 2017.
[18] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *2018 {USENIX} Annual Technical Conference ({USENIX} 18)*, 2018, pp. 147–158.
[19] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu, "Dolphinattack: Inaudible voice commands," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 103–117.
[20] N. Roy, S. Shen, H. Hassanieh, and R. R. Choudhury, "Inaudible voice commands: The long-range attack and defense," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 547–560.