Privacy Leakage Analysis for *Colluding* Smart Apps

Junzhe Wang
University of South Carolina
junzhe@cec.sc.edu

Lannan Luo
University of South Carolina
lluo@cse.sc.edu

***Abstract*—The rapid proliferation of Internet-of-Things (IoT) has advanced the development of smart environments. By installing smart apps on IoT platforms, users can integrate IoT devices for convenient automation. As smart apps are exposed to a myriad of sensitive data from devices, one severe concern is about the privacy of these digitally augmented spaces. The recent work SAINT [1] has been proposed to detect sensitive data flows in *individual* smart apps using taint analysis. But it has high false positives and false negatives due to inappropriate consideration of taint seeds and taint sinks.**

One important security issue ignored by existing work is that the IoT platform supports *parent-child* smart apps. Their ability to communicate, however, has a negative effect on security. We call the parent-child smart apps *colluding smart apps*. Unfortunately, no tool exists to detect smart app collusion. We propose PDCoLA, which addresses the limitations of SAINT, and more importantly, can detect privacy leakages by colluding smart apps. The evaluation results show that PDCoLA achieves higher accuracies than SAINT in detecting privacy leakages by *individual* smart apps, and is effective to detect privacy leakages by *colluding* smart apps.

I. Introduction

The rapid proliferation of Internet-of-Things (IoT) devices has advanced the development of smart homes and factories. The IoT market is expected to surpass $1.38 trillion by 2026 [2]. Popular IoT platforms include Samsung SmartThings, Amazon Alexa, Google Home, and Apple HomeKit.

By installing *automation apps* (also called *smart apps*) on IoT platforms, users can integrate heterogeneous IoT devices for convenient automation (e.g., turn on lights when the owner returns home). As smart apps are exposed to a myriad of sensitive data from IoT devices (e.g., device state like locked/unlocked) and user information (e.g., away/at home), one severe concern is about the privacy of these digitally augmented spaces.

IoT platforms currently provide coarse-grained controls for regulating whether a smart app can access private information, and provide little insight into how private information is actually used. For example, if a user allows a smart app to access a lock device's state, she has no idea whether or not the app will send the information to attackers. As a result, users must blindly trust that apps will properly handle their private data. Thus, what are urgently needed are analysis tools and techniques targeting IoT platforms that can identify privacy concerns in smart apps.

Taint analysis is a widely used technique for sensitive data tracking [3]. Many sensitive data tracking tools have been designed for mobile apps and other domains [4]–[10], which however are inadequate for analyzing smart apps [1], [11], [12]. The recent work SAINT [1] has been proposed to detect sensitive data flows in smart apps using static taint analysis. But taint seeds and taint sinks considered by SAINT are not appropriate, causing high false positives and false negatives, which are addressed by our work.

One important security issue, however, ignored by existing work is that the IoT platform supports *parent-child* smart apps [13]–[16]. The parent-child relationship is useful when a user wants to have multiple automations that act independently on separate IoT devices. In such a design, a parent app may have many child apps, and a child app has only one parent. More importantly, parents and children can communicate with each other. However, their ability to communicate has a negative effect on security and privacy as a smart app can send sensitive data to another smart app and eventually leak out. We call the parent-child smart apps *colluding smart apps*. Unfortunately, there are no effective tools to detect smart app collusion. Note that SAINT can only detect privacy leakages by *individual* smart apps.

We propose PDCoLA, which addresses the limitations of SAINT, and more importantly, extends the existing approach to detect privacy leakages by colluding smart apps. To make the work concrete, we showcase the idea and technique on SmartThings, one of the most popular IoT device integration platforms. We evaluate PDCoLA with different experiments and compare it to SAINT. Our experiment results show that PDCoLA achieves higher accuracies than SAINT in detecting privacy leakages by *individual* smart apps, and is effective to detect privacy leakages by *colluding* smart apps.

We made the following contributions.

- We identify one type of colluding smart apps, i.e., *parent-child* smart apps, whose ability to communicate has a negative effect on the security and privacy of the IoT ecosystem.
- Being the first in the literature, we present a system PDCoLA, extended from the existing work SAINT (focusing on detecting privacy leakages by *individual* smart app) to detect privacy leakages by *colluding* smart apps.
- PDCoLA identifies and addresses the limitations of the existing work SAINT.
- We have implemented and evaluated PDCoLA. It achieves higher accuracies than SAINT in detecting privacy leakages by *individual* smart apps, and is effective to detect privacy leakages by *colluding* smart apps.

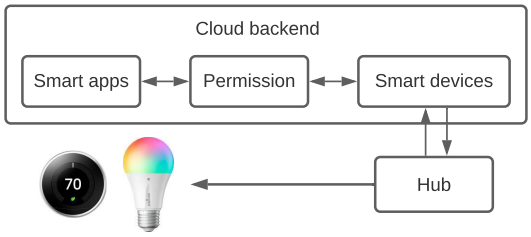

Fig. 1: The architecture of SmartThings IoT platform.

II. Background

A. SmartThings IoT Platforms

SmartThings is a proprietary platform owned by Samsung. It provides a software stack used to develop applications that control IoT devices, and includes four main components, as shown in Figure 1: *hub*, *smart apps*, *smart devices*, and *cloud*. The hub[1] bridges the communication between connected physical devices and the cloud backend. Smart apps are developed in Groovy (a dynamic, object-oriented language) [2] and run in the cloud backend.

SmartThings allows a user to specify devices and user inputs required for an app at installation. (1) User inputs are used to complete the app logic; e.g., a user input can select a device or set the time to turn on a light. (2) Devices have capabilities, consisting of *events* and *actions*. Actions represent how to control devices, and events are triggered when device states change. Apps subscribe to device events or other pre-defined events. When device states change, an event is triggered and the corresponding event handler of the subscribing app is invoked to take actions.

```
1  preferences {
2        input "contact1", " capability .contactSensor"
3        input "lock1"," capability .lock"
4  }
5
6  def installed () {
7        subscribe(contact1, "contact", eventHandler)
8  }
9
10 def eventHandler(evt) {
11       if (evt.value == "closed") {
12             log.debug "door locked"
13             lock1.lock()
14       }
15 }
```

Listing 1: A smart app.

Listing 1 shows a smart app which locks a door. It first asks a user to select two devices: one is *contact1* with the *contactSensor* capability, and another *lock1* with the *lock* capability (Line 2 and 3). Then, the app subscribes to the *contact* event of the *contact1* device, and registers the `eventHandler` method (Line 6). When the device *contact1*'s state changes, the *contact* event is triggered, which leads to the invocation of the `eventHandler` method (Line 8) to lock the device *lock1* if the *contact* event's current value is "*closed*" (Line 9-11).

B. Taint Analysis

Taint analysis is a widely used technique for sensitive data tracking [3], [17]. It tracks some selected data called *taint seeds* (e.g., data originated from untrusted sources), propagates them along program execution paths according to a customized policy called *taint propagation policy*, and then checks the taint status at certain critical locations called *taint sinks*.

Many sensitive data tracking tools have been designed for mobile apps and other domains [4]–[10], which however are inadequate for analyzing IoT apps [1], [11], [12]. The recent work SAINT [1] has been proposed to detect sensitive data flows in IoT apps using static taint analysis, which propagates taint values following all possible paths with no need for concrete execution [5], [7], [10].

III. Approach

We first present the limitations of SAINT and how PDCoLA addresses such limitations (Section III-A), and then discuss how we extend the techniques of SAINT to analyze colluding smart apps (Section III-B). Finally, a risk ranking model is proposed to evaluate the risk levels of detected privacy leakages (Section III-C).

A. Detecting Leakages by A Single Smart App

Similar to SAINT [1], we first convert the source code of a smart app to an intermediate representation (IR), which is used to construct an app's entry points, event handlers, and call graphs. Using such information, we model the lifecycle of an app and perform static taint analysis. For the details, please refer to SAINT [1].

A smart app does not have a *main* method. It declares entry points by subscribing to events. Each subscription includes a device name, an event, and an event handler (e.g., Line 7 in Listing 1). The event handler methods are commonly used to take device actions. A smart app may define multiple entry points by subscribing to multiple events. When the state of a device changes, an event is triggered and the corresponding event handler of the subscribing app is invoked to take actions. PDCoLA analyzes all subscriptions and finds their event handlers, which are considered as entry methods. Each entry method will be analyzed separately.

Below we mainly present the limitations of SAINT and how PDCoLA addresses them.

1) Taint Seeds: We consider user inputs and environment variables as taint seeds, which are explained as follows.

User Inputs. Smart apps often require user inputs (i.e., app configurations) to customize apps. User inputs may contain personally identifiable data that can be used to profile user behavior. For example, at a particular time specified by the owner, the air conditioner is turn on to wait for the owner

[1]More and more (WiFi-based) IoT devices do not require a hub to work.
[2]While SmartThings has recently started to support other languages, apps in Groovy are still the most popular and sophisticated ones.

back home; then, the particular time can be used to learn when the owner usually return back home. User inputs are declared by the keyword "*input*" (e.g., Lines 2 and 3 in Listing 1).

Environment Variables. Smart apps frequently interact with outside to retrieve environment data, such as location (whether a user is at home) and device status (e.g., whether a door is locked), which may be sensitive. We divide environment variables into four categories as shown below.

- ***Device state and information.*** When a user installs a smart app, she usually selects IoT devices (e.g., *contactSensor* at Line 2 in Listing 1). An IoT device is represented as a device object. Smart apps can interact with device objects to obtain device information. The SmartThings platform defines *interfaces* to access device information (e.g., `getId()` returns the device id). Moreover, smart apps can also directly access the fields of a device object to get device states (e.g., *switch.currentSwitch* returns the state of the *switch* device). The return variables of the interfaces and the fields of a device object are considered as taint seeds.
- ***Location.*** A location represents a user's geolocation or geographical location. The location information (such as timezone, longitude, and mode) can be accessed through either the interfaces or fields of the location object. The fields and return variables of the interfaces are considered as taint seeds.
- ***State.*** Smart apps do not store data about their previous executions; instead, they persist and retrieve data across executions via the state object (designed as a map). For example, a smart app may persist a "counter" to keep track of how many times a door is locked and store it in *state.counter*. We consider the state object and all its elements as taint seeds.
- ***Event.*** An event can contain different types of information, which may indicate the state of the device who generates the corresponding event. The event fields and the return variables of the event interfaces are set as taint seeds.

SAINT vs. PDCoLA. SAINT does not consider event information as taint seeds, while PDCoLA does. The reason we consider event information is that an event object may contain sensitive data that can be used to learn the status of a smart home or profile user behavior. E.g., by combining *event.value* (the value of this event) and *event.date* (the time when this event is created), attackers can profile user behavior—the owner may be back home when a *presence* event with the value of *present* is created, and after collecting this event for a period of time, the user behavior can be learned.

2) Taint Sinks: We consider two types of taint sinks.

Internet. Smart apps may send data to external services through HTTP interfaces (e.g., `httpGet()`, `httpPost()`, and `httpPut()`, etc.).

Message services. Smart apps use messaging APIs to deliver push notifications to users (e.g., `sendNotificationToContact()` and `sendNotification()`), and to send SMS messages to designated recipients (e.g., `sendSms()` and `sendSmsMessage()`). At each taint sink, for data to be sent outside, PDCoLA checks whether it is tainted and, if so, logs the data's taint label and destination.

SAINT vs. PDCoLA. We analyzed all HTTP interfaces and messaging APIs in the SmartThing API documentation, and exclude three messaging APIs from being considered as taint sinks, which are considered by SAINT. The first one is `sendNotificationEvent()`; it only displays a notification "*Hello Home*" but does not send out any notification. Another two are `sendPush()` and `sendPushMessage()`, which only send messages to *the user*'s mobile device.

B. Detecting Leakages by Parent-Child Apps

The type of colluding smart apps we focus on in this work is *parent-child* smart apps [13]–[16]. There may exist other types of colluding smart apps which will be explored as future work (Section VI).

The parent-child relationship is useful when a user wants to have automations that act independently on separate devices. A parent app may have many children; a child has only one parent. To create such a relationship, the parent app uses the "*app*" input to specify what app is a child. The child app should specify the "*parent*" option in its definition to specify what app should serve as the parent. Parents and children can communicate with each other. A parent can invoke the API `getChildApps()` to get all its children, or `findChildAppByName()` to get a particular child if the child's name is known, and then invoke a *public* method defined in the child. On the other hand, a child can get a reference to its parent by invoking `getParent()` and call a *public* method in the parent.

An Example. Figure 2 shows an example, where *smartblock-manager* is a parent app and has three child apps, *smartblock-chat-sender*, *smartblock-notifier*, and *smartblock-linker* (Lines 2-4 in Figure 2(a)).

In Figure 2(b), the child app *smartblock-chat-sender* sends the username (Line 17) and location mode (Line 16 and Lines 8-10) to a URL returned by `app.getParent().getServerURL()` (Lines 14-15), where `app.getParent()` returns the reference of the parent app, and `getServerURL()` is a public method defined in the parent app and returns the server URL (Line 8 in Figure 2(a)).

In Figure 2(c), the child app *smartblock-notifier* also sends the username (Line 17) and redstone signal strength (the value of an event; Line 16 and Lines 8-9) to the same URL. Note that as SAINT does not consider event information as taint seeds, the privacy leakage of redstone signal strength will be missed, causing false negative.

```groovy
section {
  app(appName:"SmartBlock Chat Sender")
  app(appName:"SmartBlock Notifier")
  app(appName:"SmartBlock Linker")
}

public getServerURL() {
  return "http :// ${ serverIp }:3333 "
}
```

(a) Parent app *smartblock-manager*.

```groovy
definition (name: "SmartBlock Chat Sender")

def initialize () {
  subscribe ( location , modeChangeHandler)
}

def modeChangeHandler(evt) {
  def mode = evt.value
  def message = "mode changed to: \"${
      mode}\""
  chatMessageToMC(message)
}

def chatMessageToMC(message) {
  def parent = app.getParent ()
  def url = "${parent.getServerURL()}/ "
  url += "chat?message=${message}"
  url += "&username=${username}"
  httpPost ( url ) {...}
}
```

(b) Child app *smartblock-chat-sender*.

```groovy
definition (name: "SmartBlock Notifier ")

def initialize () {
  subscribe (smartBlock, "
      redstoneSignalStrength ",
      redstoneSignalStrengthHandler )
}

def redstoneSignalStrengthHandler (evt) {
  def value = evt . value
  def message = "redstone signal is ${
    value}"
  chatMessageToMC(message)
}

def chatMessageToMC(message) {
  def parent = app.getParent ()
  def url = "${parent.getServerURL()}/ "
  url += "chat?message=${message}"
  url += "&username=${username}"
  httpPost ( url ) {...}
}
```

(c) Child app *smartblock-notifier*.

Fig. 2: An example of parent-child smart apps. The example code only includes the snippets related to app communication. The code has been modified from the original to ease the understanding.

Approach. To the best of our knowledge, *no work exists to detect privacy leakages by parent-child smart apps (i.e., colluding smart apps)*. We extend the techniques of SAINT—which focuses on detecting privacy leakages by *individual* smart apps—and design PDCoLA to detect privacy leakages by partent-child smart apps.

Specifically, for a parent app, PDCoLA finds its children through the "*app*" input. It then analyzes each entry method in each app separately. For each entry method in either the parent or child smart app, PDCoLA builds an Inter-procedural Control Flow Graph (ICFG) as in [12], and creates a call graph. After that, for an entry method, it constructs an *inter-App* Control Flow Graph (ACFG) by connecting ICFGs of all involved methods in different apps. For example, if a method M_1 in a child app invokes a method M_2 in the parent app, PDCoLA intercepts `app.getParent()` and directly connects M_1 with M_2. Similarly, if a method N_1 in the parent app invokes a method N_2 in the child app, PDCoLA intercepts `getChildApps()` or `findChildAppByName()` and connects N_1 with N_2. Finally, each ACFG is analyzed separately. We consider user inputs and environment variables in all apps as taint seeds.

C. Risk Ranking

We propose a risk ranking model to evaluate the risk levels of detected leakage instances for assisting users to prioritize handling higher-risk threats. It takes *data source-sensitivity* and *data destination-severity* into consideration.

Let a privacy leakage instance be $I = (d_s, d_d)$, where $d_s = \{d_s^0, ..., d_s^n\}$ denotes the set of data sources, and d_d the data destination (for a taint sink, there may be more than one kind of sensitive data being sent out to a destination). The risk level of I is represented as $Risk(I)$, which is determined by the risk levels of both data sources and data destination, as described below.

- *Data sources are related to taint seeds.* (1) As user inputs may contain personally identifiable data, which can be used to profile user behavior, we assign a high sensitivity value 2 to user inputs. (2) Some environment variables can reflect the status of a smart home (e.g., whether the door is locked/unlocked). We thus assign a high sensitivity value 2 to three types of environment variables, including *device*, *location* and *event*, and a low sensitivity value 1 to *state*.
- *Data destinations are related to taint sinks.* Smart apps can send data to different destinations, which are divided into three categories: (1) device manufacturer/app developer, (2) Samsung (if the URL contains "SmartThings" or "Samsung"), and (3) user/user's contacts (if the destination is specified by a user, e.g., an app may ask the user to input a phone number). We assign different severity values to them: 3 to app developer/device manufacturer, 2 to Samsung, and 1 to user/user's contacts.

Finally, the risk level of an instance I is represented as $Risk(I) = (Risk(d_s), Risk(d_d))$, where $Risk(d_s) = \max_{i=0}^{n} Risk(d_s^i)$, $i \in [0, n]$, is the sensitivity value of the data source(s), and $Risk(d_d)$ the severity value of the data destination. The higher a value is, the more risk the privacy leakage instance has. For each privacy leakage instance, PDCoLA outputs its risk level, including the sensitivity value of the data source(s) and severity value of the data destination.

IV. Evaluation

A. Experimental Settings

As SAINT focuses on a single smart app, we first compare PDCoLA to SAINT on detecting privacy leakages of each single smart app, and then evaluate PDCoLA under the parent-child smart app scenario.

Our experiments were conducted on a machine with an Intel Core i7-7700 CPU @ 3.60GHz with 16GB RAM.

Datasets. We use two datasets to evaluate PDCoLA.

- For fair comparison between PDCoLA and SAINT, we use *the same dataset* that was used by SAINT. Specifically, *Dataset-I* contains 168 official apps from the SmartThings GitHub repo [18] and 62 third-party apps from the SmartThings community forum [19].
- We create *Dataset-II* containing 18 pairs of hand-crafted parent-child smart apps, which is used, together with *Dataset-I*, to evaluate whether PDCoLA can successfully detect sensitive data flows of parent-child smart apps.

B. Comparison with SAINT

We use *Dataset-I* to compare PDCoLA with SAINT. As PDCoLA and SAINT consider taint seeds and taint sinks differently (see Section III-A), their results of privacy leakage detection may not be the same.

Differences due to Taint Seeds. PDCoLA considers events as taint seeds (as they may contain sensitive data that can be used to profile user behavior), but SAINT does not. There are 16 privacy leakage instances where the taint seeds are the event information, which are manually verified. However, these instances are missed by SAINT, causing false negatives. For example, an app sends the current humidity level, which is the value of the *humidity* event, to a recipient by invoking `sendSmsMessage()`.

Differences due to Taint Sinks. Three messaging APIs, which are impossible to send messages to a destination with potential risks, are excluded from being considered as taint sinks by PDCoLA, which however are considered by SAINT. There are totally 84 privacy leakages instances whose taint sinks are one of the three messaging APIs. We manually verified these instances. However, these instances are reported by SAINT, causing false positives.

Risk Analysis. There are totally 272 privacy leakage instances detected by PDCoLA. SAINT detects 340 leakage instances, where 84 are false positives due to it considers three messaging APIs as taint sinks. Moreover, 16 instances are missed by SAINT as it does not consider events as taint seeds. We manually checked the reported privacy leakage instances and confirmed the results. The manual checking is feasible since smart apps are comparatively smaller than other programs such as mobile apps and web applications. Table I shows the breakdown of the detected privacy leakage instances by PDCoLA in terms of different values of data source-sensitivity and data destination-severity (the meaning of these values is discussed in Section III-C).

TABLE I: Privacy leakages breakdown.

(a) Data source-sensitivity.

Sensitivity value	2	1
# of instances	246	26

(b) Data destination-severity.

Severity value	3	2	1
# of instances	116	9	147

C. Study of Parent-Child Smart Apps

We next use both *Dataset-I* and *Dataset-II* to evaluate PDCoLA's capability of detecting privacy leakages by parent-child smart apps.

Results of *Dataset-I*. Three sets of parent-child smart apps are found in *Dataset-I*.

- The first set is shown in Figure 2. In *smartblock-char-sender*, the data sources of the privacy leakage include location information and user name, and the destination is the device manufacturer's URL. As a result, the risk level is (2, 3). Similarly, in *smartblock-notifier*, the data sources of the leakage include event infromation and user name, and the destination is the device manufacturer's URL. Thus, the risk level of the leakage is also (2, 3).
- In the second set, the parent app *ecobee-connect* first generates a message including the child devices' status (i.e., *connected* or *disconnected*). This message is then shared with its children and sent by its children to SmartThings and device manufacturer. The risk level is (2, 3) if the message is sent to device manufacturer, and (2, 2) if sent to SmartThings.
- In the third set, the parent app sends a value, which is depended on a user input, to its child app. Based on the value, the child app then updates the color temperature of an IoT device, and reports the updated color temperature to SmartThings. Thus, the risk level is (2, 2).

Note that only when both parent and child apps are analyzed together, a precise detection result about *what sensitive data is sent to which destination* can be obtained, which however cannot be achieved by SAINT.

Results of *Dataset-II*. *Dataset-II* contains 18 sets of hand-crafted parent-child smart apps. Below we describe how we created these parent-child smart apps.

- We first randomly selected 12 smart apps from *Dataset-I*, which are considered as parent apps, and another 12 smart apps as child apps. We then inserted a line of code into each parent app to indicate which app is its child, and modified the parent app code to let it invoke a method defined in the child (or modified the child app code to let the child invoke a method defined in the parent). Moreover, we make sure that the selected method to be invoked is involved in a sensitive data flow. This resulted in 12 pairs of parent-child smart apps.

- We also randomly selected 12 third-party smart apps from the SmartThings community forum [19], and followed the same way above to create another 6 pairs of parent-child smart apps.

Through this, each pair of parent-child smart apps comes with the ground truth of what data leaks are in these apps. The evaluation result shows that PDCoLA can successfully find all the data leaks and determines the data sources and destinations correctly.

V. Related Work

This section discusses the related work. We first discuss the related work on analyzing IoT applications, and then discuss the related work on privacy leakage detection.

Analysis of IoT Applications. Many approaches have been proposed to analyzed IoT applications [1], [11], [12], [20]–[26], [26], [27], [27], [28], [28]–[30], [30]–[36]. These works centered on the security of IoT programming platforms and IoT devices. For example, ContexIoT provides contextual integrity for smart apps [12]. HAWatcher extracts semantics from smart apps for anomaly detection [29].

Some aim to detect unsafe interactions between IoT apps, which are caused by multiple IoT apps control the same device or influence the same physical channel [31], [37]–[40]. For example, a user may install an app A_1 to open the window if the temperature is too high, and another app A_2 to close the window when the user leaves the house. Considering the scenario: if the temperature is too high, A_1 opens the window; if the user leaves home, A_2 closes the window. The the two apps conflict.

However, these approaches differ from PDCoLA in three aspects. (1) *Research problems are different*: PDCoLA aims to detect private leakages rather than unsafe interactions. (2) *Techniques are different*: PDCoLA applies taint analysis, while these approaches apply other kinds of techniques like model checking [21], [41], [42]. (3) *Apps are different*: PDCoLA analyzes parent-child apps which do not necessarily control the same device or influence the same physical channel.

Privacy Leakage Detection. Many sensitive data tracking tools have been designed for mobile apps and other domains [4]–[10], [43], [44], which however are inadequate for analyzing smart apps [1], [45], [46].

SAINT [1] is the first system that detects sensitive data flows in smart apps. However, as taint seeds and taint sinks considered by SAINT are not appropriate, it has high false positives and false negatives. Moreover, it can only detect privacy leakages by *individual* smart apps. In thiw work, we design PDCoLA, which addresses the limitations of SAINT, and extends the technique to detect privacy leakages by parent-child smart apps.

VI. Discussion

Smart apps run in a sandboxed environment [47], which limits apps' ability to share data. We have identified one type of colluding smart apps, i.e., parent-child smart apps, which can share data via *overt channels*. There may exist other types of colluding smart apps. For example, two smart apps, unlike parent-child apps, cannot communicate with each other but send data to the *same destination*. While each piece of data does not cause harms, the aggregated data is dangerous and can be used to learn the status of smart homes. We plan to investigate this problem, specifically, which insensitive data if aggregated would cause high privacy risks, and notify users if those smart apps are installed together.

Moreover, we also plan to investigate whether or not smart apps can communicate via *covert channels*. Many approaches on detecting colluding mobile apps' communication via *covert channels* have been proposed [48]–[51]. We will investigate, whether covert channels are possible between smart apps, and if so, whether the existing approaches can be applied for the smart app scenario.

This research identifies a new problem of smart app collusion. It demonstrates the importance of analyzing parent-child smart apps to achieve precise privacy leakage detection. We hope it can inspire and lead to new ideas and techniques for this critical problem.

VII. Conclusion

To the best of our knowledge, no work exists to detect smart app collusion. We propose PDCoLA, being the first one in the literature, to detect privacy leakages by parent-child smart apps. PDCoLA addresses the limitations of SAINT, and extends the technique of SAINT for analyzing parent-child smart apps. The evaluation shows that PDCoLA achieves higher accuracies than SAINT in detecting privacy leakages by *individual* smart apps, and is effective to detect privacy leakages by *parent-child* smart apps.

Acknowledgments

This work was supported by the US National Science Foundation (NSF) under grants CNS-1850278 and CNS-1953073.

References

[1] 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 *USENIX Security*, 2018.

[2] Hashedout, "Re-hashed: 27 surprising iot statistics you don't already know," https://www.thesslstore.com/blog/20-surprising-iot-statistics-you-dont-already-know/.

[3] J. Newsome and D. X. Song, "Dynamic taint analysis for automatic detection, analysis, and signaturegeneration of exploits on commodity software." in *NDSS*, vol. 5. Citeseer, 2005, pp. 3–4.

[4] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[5] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *Acm Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[6] M. G. Kang, S. McCamant, P. Poosankam, and D. Song, "Dta++: dynamic taint analysis with targeted control-flow propagation." in *NDSS*, 2011.

[7] X. Wang, Y.-C. Jhi, S. Zhu, and P. Liu, "Still: Exploit code detection via static taint and initialization analyses," in *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 2008, pp. 289–298.

[8] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman, "Taj: effective taint analysis of web applications," *ACM Sigplan Notices*, vol. 44, no. 6, pp. 87–97, 2009.

[9] W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*. ACM, 2014, pp. 1–6.

[10] Z. Yang and M. Yang, "Leakminer: Detect information leakage on android with static taint analysis," in *2012 Third World Congress on Software Engineering*. IEEE, 2012, pp. 101–104.

[11] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks," in *USENIX Security 16*, 2016.

[12] 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.

[13] Samsung, "Parent-child smartapps," https://docs.smartthings.com/en/latest/smartapp-developers-guide/parent-child-smartapps.html.

[14] ——, "New parent/child smart app documentation," https://community.smartthings.com/t/new-parent-child-smart-app-documentation/26153.

[15] ——, "Parent-child smartapp."

[16] ——, "Parent-child communication," https://community.smartthings.com/t/parent-child-communication/156058.

[17] L. Luo, Q. Zeng, C. Cao, K. Chen, J. Liu, L. Liu, N. Gao, M. Yang, X. Xing, and P. Liu, "Tainting-assisted and context-migrated symbolic execution of android framework for vulnerability discovery and exploit generation," *IEEE Transactions on Mobile Computing*, vol. 19, no. 12, pp. 2946–2964, 2019.

[18] SmartThings, "SmartThings Official App Repository," https://github.com/SmartThingsCommunity, 2020.

[19] ——, "SmartThings Community Forum For Third-party Apps," https://community.smartthings.com/, 2020.

[20] V. Sivaraman, H. H. Gharakheili, A. Vishwanath, R. Boreli, and O. Mehani, "Network-level security and privacy control for smart-home iot devices," in *WiMob*. IEEE, 2015.

[21] Z. B. Celik, P. McDaniel, and G. Tan, "Soteria: Automated iot safety and security analysis," in *USENIX Security'18)*, 2018, pp. 147–158.

[22] Q. Wang, W. U. Hassan, A. Bates, and C. Gunter, "Fear and logging in the internet of things," in *NDSS*, 2018.

[23] Z. B. Celik, E. Fernandes, E. Pauley, G. Tan, and P. McDaniel, "Program analysis of commodity iot applications for security and privacy: Challenges and opportunities," *arXiv preprint arXiv:1809.06962*, 2018.

[24] O. Alrawi, C. Lever, M. Antonakakis, and F. Monrose, "Sok: Security evaluation of home-based iot deployments," in *IEEE S&P*, 2019, pp. 208–226.

[25] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," *arXiv preprint arXiv:1808.02125*, 2018.

[26] H. Chi, Q. Zeng, X. Du, and L. Luo, "Pfirewall: Semantics-aware customizable data flow control for home automation systems," in *The 28th Annual Network and Distributed System Security Symposium (NDSS)*, 2021.

[27] X. Li, Q. Zeng, L. Luo, and T. Luo, "T2pair: Secure and usable pairing for heterogeneous iot devices," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 309–323.

[28] X. Li, F. Yan, F. Zuo, Q. Zeng, and L. Luo, "Touch well before use: Intuitive and secure authentication for iot devices," in *The 25th annual international conference on mobile computing and networking*, 2019, pp. 1–17.

[29] C. Fu, Q. Zeng, and X. Du, "Hawatcher: Semantics-aware anomaly detection for appified smart homes," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[30] X. Liu, Q. Zeng, X. Du, S. L. Valluru, C. Fu, X. Fu, and B. Luo, "Sniffmislead: Non-intrusive privacy protection against wireless packet sniffers in smart homes," in *24th International Symposium on Research in Attacks, Intrusions and Defenses*, 2021, pp. 33–47.

[31] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 411–423.

[32] L. Luo, Q. Zeng, B. Yang, F. Zuo, and J. Wang, "Westworld: Fuzzing-assisted remote dynamic symbolic execution of smart apps on iot cloud platforms," in *Annual Computer Security Applications Conference*, 2021, pp. 982–995.

[33] Q. Zeng, L. Luo, Z. Qian, X. Du, and Z. Li, "Resilient decentralized android application repackaging detection using logic bombs," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018, pp. 50–61.

[34] L. Luo, Y. Fu, D. Wu, S. Zhu, and P. Liu, "Repackage-proofing android apps," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 550–561.

[35] H. Chi, C. Fu, Q. Zeng, and X. Du, "Delay wreaks havoc on your smart home: Delay-based: Automation interference attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2022, pp. 1575–1575.

[36] C. Fu, Q. Zeng, H. Chi, X. Du, and S. L. Valluru, "Iot phantom-delay attacks: Demystifying and exploiting iot timeout behaviors," Technical Report, Tech. Rep., 2021.

[37] M. Alhanahnah, C. Stevens, and H. Bagheri, "Scalable analysis of interaction threats in iot systems," in *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*, 2020, pp. 272–285.

[38] M. O. Ozmen, X. Li, A. C.-A. Chu, Z. B. Celik, B. Hoxha, and X. Zhang, "Discovering physical interaction vulnerabilities in iot deployments," *arXiv preprint arXiv:2102.01812*, 2021.

[39] H. J. Kang, S. Q. Sim, and D. Lo, "Iotbox: Sandbox mining to prevent interaction threats in iot systems," in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2021, pp. 182–193.

[40] M. Balliu, M. Merro, and M. Pasqua, "Securing cross-app interactions in iot platforms," in *2019 IEEE 32nd computer security foundations symposium (CSF)*. IEEE, 2019, pp. 319–31 915.

[41] 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*, 2018, pp. 191–203.

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

[43] L. Luo, "Heap memory snapshot assisted program analysis for android permission specification," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 435–446.

[44] Q. Zeng, L. Luo, Z. Qian, X. Du, Z. Li, C.-T. Huang, and C. Farkas, "Resilient user-side android application repackaging and tampering detection using cryptographically obfuscated logic bombs," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2582–2600, 2019.

[45] A. Mandal, P. Ferrara, Y. Khlyebnikov, A. Cortesi, and F. Spoto, "Cross-program taint analysis for iot systems," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 2020, pp. 1944–1952.

[46] A. M. Alashjaee, S. Duraibi, and J. Song, "Iot-taint: Iot malware detection framework using dynamic taint analysis," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2019, pp. 1220–1223.

[47] 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.

[48] S. Chandra, Z. Lin, A. Kundu, and L. Khan, "Towards a systematic study of the covert channel attacks in smartphones," in *International Conference on Security and Privacy in Communication Networks*. Springer, 2014, pp. 427–435.

[49] A. Al-Haiqi, M. Ismail, and R. Nordin, "A new sensors-based covert channel on android," *The Scientific World Journal*, vol. 2014, 2014.

[50] W. Qi, Y. Xu, W. Ding, Y. Jiang, J. Wang, and K. Lu, "Privacy leaks when you play games: A novel user-behavior-based covert channel on smartphones," in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*. IEEE, 2015, pp. 201–211.

[51] W. Qi, W. Ding, X. Wang, Y. Jiang, Y. Xu, J. Wang, and K. Lu, "Construction and mitigation of user-behavior-based covert channels on smartphones," *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 44–57, 2017.