Some Recipes Can Do More Than Spoil Your Appetite: Analyzing the Security and Privacy Risks of IFTTT Recipes

Milijana Surbatovich
University of Rochester

Jassim Aljuraidan
Carnegie Mellon University

Lujo Bauer
Carnegie Mellon University

Anupam Das
Carnegie Mellon University

Limin Jia
Carnegie Mellon University

ABSTRACT

The use of end-user programming, such as *if-this-then-that* (IFTTT), is becoming increasingly common. Services like IFTTT allow users to easily create new functionality by connecting arbitrary Internet-of-Things (IoT) devices and online services using simple *if-then* rules, commonly known as *recipes*. However, such convenience at times comes at the cost of security and privacy risks for end users. To gain an in-depth understanding of the potential security and privacy risks, we build an *information-flow model* to analyze how often IFTTT recipes involve potential integrity or secrecy violations. Our analysis finds that around 50% of the 19,323 unique recipes we examined are potentially unsafe, as they contain a secrecy violation, an integrity violation, or both. We next categorize the types of harm that these potentially unsafe recipes can cause to users. After manually examining a random selection of potentially unsafe recipes, we find that recipes can not only lead to harms such as personal embarrassment but can also be exploited by an attacker, e.g., to distribute malware or carry out denial-of-service attacks. The use of IoT devices and services like IFTTT is expected only to grow in the near future; our analysis suggests users need to be both informed about and protected from these emerging threats to which they could be unwittingly exposing themselves.

Keywords

End-user programming; IFTTT service; Internet of Things (IoT); Information-flow

1. INTRODUCTION

Use of Internet-of-Things (IoT) devices, especially in the context of smart homes, is growing rapidly [5]. Nest thermostats allow users to control the temperature of their home remotely. Philips Hue lights can be programmed via mobile phones to change the color of the light to create personalized ambiance. Smart cameras can detect motion and alert users of break-ins. These are some of the many examples of IoT devices that are currently being used by users in their everyday life. Gartner estimates that around 20 billion IoT devices will be installed by 2020 [3].

©2017 International World Wide Web Conference Committee (IW3C2), published under Creative Commons CC BY 4.0 License.
WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052709

Moreover, services such as IFTTT allow end users to interface between different channels (comprised of different IoT devices and online services) [6], thus making it easy for users to connect the different functionalities of their deployed IoT devices to each other and to those offered by various online services. With IFTTT, users can join together any two of its 364 channels using *recipes*, simple *if–then* rules, that connect specific *trigger* events to desired *action* events.[1] An example of an actual recipe is "If my Fitbit logs 10,000 steps, then update my Twitter with a new post." Without much technical skill, users can in this way easily customize the behavior of their devices to better suit their needs.

However, like many emerging technologies, IFTTT recipes can also create security and privacy risks for their end users. Security and privacy risks could arise from recipes that inadvertently leak private information, recipes that can be triggered from an untrusted source but execute a potentially dangerous *action*, or recipes that chain together unexpectedly (where the action executed by one recipe serves as a trigger for another). Next, we show concrete examples of security and privacy risks.

A user can create a recipe that publishes his daily physical activity (such as step count, distance, and calories burned) to Facebook. Such a recipe explicitly broadens the audience of what was previously information private to the user, and could inadvertently cause embarrassment or other harms. Suppose, for example, the user claims to be unwell or intentionally tries to avoid a social gathering; if the user is at the same time engaged in physical activity that is automatically reflected in a Facebook post, the claims of illness will be, perhaps embarrassingly, refuted. To give another example, a user, during a road trip, adds a recipe that uploads any photo taken by her Android device to her Flickr account to document her journey. Unfortunately, the user forgets to remove this recipe after the trip, and, soon after, photographs that she takes of her passport are unintentionally made public on Twitter.[2]

Some recipes lead to security risks and potential harm by allowing an attacker to exploit certain trigger channels. For example, a user could write a recipe to open the window if the temperature rises above a certain threshold. This recipe enables an attacker who can affect the temperature of the user's house to cause the window to open and hence give him access to the house. The attacker could affect the temperature of the house in different ways: for example, if there is an outside fuse box, the attacker could flip the breaker and turn off the air conditioning; if the air conditioning has an exhaust to the outside, the attacker could cover it. To give another exam-

[1] IFTTT in November 2016 introduced *applets*, an enhanced version of recipes. We discuss applets in Section 7.3.

[2] Unless explicitly stated otherwise, these and all other examples we use in the paper are of actual IFTTT recipes, although we only hypothesize about their intended uses.

ple, to conveniently manage email attachments, a user could write a recipe that uploads all attachments from newly received emails to her OneDrive folder. Later, if the user receives an email with a malicious attachment and uses OneDrive to sync multiple devices, then the malicious attachment would automatically be copied to multiple devices, increasing the likelihood that the user will mistakenly execute the malicious program.

These examples show that making it possible to almost arbitrarily connect smart devices and online services also introduces many new opportunities for users to harm themselves, whether through unintentionally leaking information, undermining their physical security, or exposing themselves to cyber threats. To investigate the extent to which users may be creating recipes that expose them to potential security and privacy risks, we examined a set of 19,323 unique published IFTTT recipes—most recipes are published so that they can be reused—collected by Ur et al. [31]. Based on our manual inspection of the recipes, we defined an information-flow lattice consisting of labels that specify the secrecy and integrity levels of recipe components. We analyzed individual recipes based on this information-flow model. Concerningly, we found that around 50% of the recipes involve either a secrecy or an integrity violation (or both). By manually examining a random sample of these *violating* recipes, we categorized the potential harms to users into four broad categories. In doing so we also validated the results of our information-flow violation analysis: while the recipes that our analysis flags as violations are sometimes likely consistent with users' intentions, they in general do have the potential to cause or increase the risk of harms such as embarrassment, leaking behavioral data, or even physical harm.

We also observe that recipes can inadvertently be chained together, with the outcome of one recipe causing another recipe to be triggered. The existence of such chains does not appear to be part of the IFTTT programming model, and both the possibility and the existence of specific chains among a user's recipes is opaque to the user. We examined the prevalence of recipe chains in our dataset and how they affect users' risk. We found that for users who use 30 recipes—slightly above the norm—on average at least two of their recipes will form a chain, and more than half of these chains contain a potentially unsafe recipe.

Our paper makes the following contributions:

- We defined a multi-level information-flow lattice for labeling the secrecy and integrity characteristics of IFTTT triggers and actions.
- We apply our model to publicly shared IFTTT recipes: we decorate recipes with security labels and encode them in Prolog for automated analysis.
- We quantitatively analyze the recipes for secrecy and integrity violations, providing the first insight into the extent to which published recipes may involve privacy or integrity violations.
- We develop a categorization of potential harms that violating recipes can inflict by manually inspecting a random selection of violating recipes.

Roadmap. In Section 2, we provide a brief overview of the IFTTT framework. Section 3 describes our information-flow model. We describe the findings of our analysis in Section 4, and the possible harms that result from violations in Section 5. Section 7 discusses the implications of our findings and the limitations of our approach. We conclude in Section 8.

2. BACKGROUND ON IFTTT

If-this-then-that (IFTTT) [6] is an end-user programming framework to connect smart devices and online services, based on a trigger–action paradigm. The connectible devices and services are known as channels; examples are Twitter, Google Drive, SmartThings, and Nest thermostats. Each channel has a specific set of events that can trigger a recipe, and another set of events that can form the response to the trigger being executed. For instance, the triggers in the Nest thermostat channel are setting the thermostat to *Home*, *Away*, or *Eco* modes, or the temperature falling below or rising above a threshold. The actions that can be triggered include setting the thermostat to the different modes or setting the temperature to a specific value. To create a recipe, the user selects the trigger and action channels and events, and then fills in any necessary parameters, known as *ingredients*, to fully specify the trigger and action events. Ingredients can be an email address, a phone number, a link to Dropbox folder, or a specific value (e.g., to which to set the temperature). In other words, ingredients are the personalized components of a recipe.

IFTTT is constantly adding new channels. In October 2016 there were 364 channels [8], for which we provide a broad, informal categorization in Fig. 1. The *smart home* category includes climate control, home security, lighting, and other smart appliances. The *personal* category covers fitness wearables, smart watches, smartphones, and photo apps. *Social and online media* channels include social media sites and news networks, while *business and communication tools* span workflow trackers, email, finance apps, document reviewers, and cloud storage. Many channels deal with calendars, to-do-lists, notifications, and task management; we call this category *task tracking*. The *smart home* category contains by far the largest number of channels. As we discuss later, however, the majority of the most popular channels belongs to the *social and online media* category (see Section 4.1).

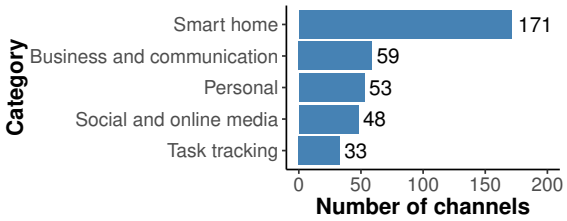

Figure 1: Breakdown of IFTTT channels into five broad categories.

IFTTT has seen a large increase in its user base in recent years; in 2015, IFTTT was reported to have more than one million unique users [4]. With the emergence of the IoT, the use of IFTTT is expected to continue growing [1].

3. INFORMATION-FLOW MODEL

We next describe the information-flow model we developed to reason about the security and privacy risks of IFTTT recipes. By examining a subset of recipes, we develop two sets of information-flow labels; we arrange them in a secrecy lattice and an integrity lattice to describe the types of information that flow from trigger events to action events (Section 3.1). Then, we give examples of potentially unsafe recipes identified by associating labels to events (Section 3.2). Finally, we formalize the notion of recipe chains and describe their contribution to potential security and privacy risks (Section 3.3). Results of analyzing our dataset of IFTTT recipes using this information-flow model are presented in Section 4.

3.1 Security Lattices

There are two types of information-flow policy violations: secrecy violations and integrity violations. Secrecy violations occur if information that should only be known only by a specific set of people becomes available to a larger audience, potentially leaking

private information [28, 20]. Integrity violations occur if information from less trusted sources influences information from more trusted sources, potentially corrupting it [28, 10].

To reason about information-flow properties of IFTTT recipes, we label each IFTTT trigger and action with one or more secrecy and integrity labels. In our model, secrecy labels denote who could know that the event took place or the details of the event, and integrity labels denote who could cause the event. As is standard, we define our security lattices based on a partial order, written $\sqsubseteq$ between security lables [33]. It is safe to allow information to flow from a lower (public or trusted) label to a higher label (private or untrusted), but not the other way around. If a recipe starts at a private source and ends at a public sink or if it starts at a untrusted source and ends at a trusted sink then it is a violation of secrecy or integrity policies.

To determine which security labels would be effective at describing the secrecy and integrity characteristics of IFTTT trigger and action events, we examined recipes to develop candidate labels, and then iteratively refined the candidate labels through applying them to additional recipes. This process culminated in the construction of two lattices, shown in Fig. 2 and 3. The secrecy lattice has three levels. At the top there is the *private* label, denoting information that only the user of a recipe knows, such as Fitbit activity, or received texts and emails; the middle level is composed of two somewhat privileged groups (discussed next); and the lowest level is *public*, describing information with unrestricted access (e.g., that has been publicly shared). In the middle level, *restricted physical* describes events that take place in partially privileged spaces, such as a home or office. Anyone in close proximity is privy to such events and could potentially observe them; but these events are invisible to users not in proximity. For instance, an event in this category could involve a phone ringing, lights blinking, or a thermostat being adjusted. Similarly, *restricted online* describes events that take place online with a restricted audience, primarily through social media. Events in this category could be Facebook or Twitter posts, updates on a work management app, or Spotify activity.

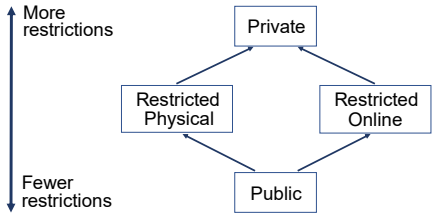

Figure 2: Secrecy lattice. A violation occurs when the corresponding labels of a trigger–action pair go from more restricted to less restricted or if they go between the middle groups.

The integrity lattice has a similar structure to the secrecy lattice. The most restricted label is now *trusted*, referring to events that only the user should be able to cause (i.e., trigger), and the least secure label is *untrusted*, used for events that could potentially be caused by anyone. There is a slightly less trusted variant of the trusted group, *trusted other*, which describes sources that the user does not control but would be extremely hard to manipulate by others, such as natural phenomena (e.g., weather or time of day, as reported by an authoritative source) or a reputable website that is unlikely to be easily manipulated (e.g., The New York Times). Similarly, there is a slightly more trusted variant of untrusted called *untrusted group*, which describes events that can be manipulated by unknown groups of people (e.g., the event that captures that a new post has become the most popular post on a subreddit can be triggered by an arbitrary collection of people who collude to upvote that post). *Restricted online* and *restricted physical* refer to similar events as in the security lattice.

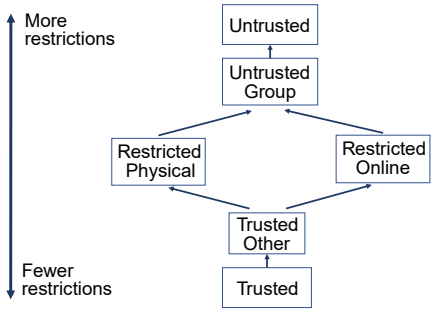

Figure 3: Integrity lattice. It has a similar structure as the secrecy lattice with additional variants of trusted and untrusted sources.

We allow an event to have more than one label to account for different contextual situations, such as a motion sensor that could be indoors (*restricted_physical* integrity) or outdoors (*untrusted* integrity). We chose these labels for our lattice because they represent realistic groupings of the behaviors of devices and services according to their intended use, while still being general enough to perform user-independent analysis. Because these groupings are broad, our estimate of risk is conservative; some violations we report would disappear with more accurate knowledge of a user's environment. For instance, a cloud storage folder could either be private or shared to a group; for our analysis, we assume the latter, even though it leads to reporting secrecy leaks that do not exist if the user specified a private folder. Making the labels more precise (e.g., by instantiating them with ingredients, as described in Section 2) would require us to know exactly where individuals have installed their devices, what privacy settings they have on their social media accounts, etc. Such fine-grained labeling would allow a more accurate analysis of risks, but is not compatible with a large-scale analysis, which is our focus.

3.2 Examples of Unsafe Recipes

A recipe has a security violation if a more restricted trigger is linked to a less restricted action [33] or if the labels of the recipe's trigger and action aren't comparable (i.e., the middle-level labels *restricted physical* and *restricted online*). Since each trigger and action can potentially have a set of labels, a recipe can be categorized into one of three groups: *definite violation* (all combinations of labels violate information-flow constraints, as described by the secrecy and integrity lattices), *maybe violation* (some combinations of labels violates information-flow constraints), or *safe* (no combinations of labels violates information-flow constraints).

Following are examples of four violating recipes in the context of our information-flow model.

Definite secrecy violation: *private* → *public*. *If I take a new photo with the front camera of my phone, add it to Flickr as a public photo.* Photos taken with a phone camera are by default only seen by the user, so the trigger label is *private*. Anyone browsing Flickr can see a public photo, so the action label is *public*. This recipe could be harmful if it causes a user to unintentionally upload pictures of private documents.

Maybe secrecy violation: *restricted_physical* → (*private*, *restricted_online*). *If I enter a specific area, upload a file to Google Drive.* Nearby people can see you enter the area, so the trigger is *restricted_ physical*. The label of the action depends on the setting of the Google Drive folder. If it is shared with a group (e.g., housemates

or family), the user's location becomes visible to an online group; if it is private, there is no violation.

Definite integrity violation: *untrusted* → *restricted_physical*. *If there is a new Instagram photo by anyone in the area, turn my smart switch on, then off.* If the area specified is open to the public, anyone with an Instagram account could take the photo, so the trigger label is *untrusted*. Only people in my house should be able to toggle my smart switch, so the action label is *restricted_physical*. This recipe could be harmful if someone takes many photos in short succession, as it could cause a light connected to the switch to blink rapidly, harming the switch and disturbing people in the house.

Definite integrity violation: *restricted_online* → *trusted*. *If I am tagged in a photo, create a new Facebook status.* Any of a user's friends can tag her in a photo, so the trigger is *restricted_online*, but only the user can update her status, so the action label is *trusted*. This recipe could have undesired effects if a friend uploads a large album and tags a user in many of the photos, causing her account to spam others with status updates.

3.3 Chaining Recipes

To fully understand the ways in which recipes may cause security and privacy risks, it is important also consider *recipe chains*. There are two ways recipes can be linked to form chains; we call them *direct linking* and *physical connections*. Recipes A and B are directly linked if A's action channel and B's trigger channel are the same and A's action fires B's trigger. E.g., the action "Send an email to my email account" will directly fire the trigger "New email received" if both are from the user's Gmail channel. Two recipes can be physically linked if the action of the first recipe affects a physical medium or channel such as temperature, sound, or light, and the trigger of the second recipe is fired by changes to the same medium or channel.

Defining chains To define chains we first start by defining *paths*. A path of length N, where N is a integer and $N \geq 2$, is a sequence of recipes $R_1, \ldots, R_N$ such that each adjacent pair of recipes are linked, i.e., $\forall 1 \leq i < N : \texttt{linked}(R_i, R_{i+1})$, and no two recipes are the same. As discussed above two recipes are linked either directly or if there is a physical connection between them, i.e., formally $\texttt{linked}(R_i, R_{i+1})$ if $\texttt{action}(R_i) = \texttt{trigger}(R_{i+1})$ or $\exists m \in M : \texttt{changes}(\texttt{action}(R_i), m) \wedge \texttt{monitors}(\texttt{trigger}(R_{i+1}), m)$ where M refers to all possible physical media.

A *max-chain* of length N is a path of length N that cannot be extended by prefixing or suffixing it by any other recipes in the recipe set that is not already in the path. For convenience, in this document we will refer to max chains as just *chains*.

4. ANALYZING IFTTT RECIPES

Using our information-flow model, we can answer interesting questions about the IFTTT framework. We first describe our dataset of publicly available IFTTT recipes (Section 4.1) and our analysis methodology (Section 4.2). Then, we report the results of analyzing the dataset as a whole for information-flow violations (Section 4.3). Finally, we report on a similar analysis that focuses on subsets of recipes that may be adopted by individual users (Section 4.4).

4.1 Dataset

Our dataset consists of all publicly shared IFTTT recipes as of May 2016 (an updated collection from Ur et al. [31]). In this section we present key characteristics of the all the unique recipes and channels in our dataset.

Number of trigger channels	251
Number of trigger events	876
Number of action channels	218
Number of action events	470
Number of unique recipes	19,323

Table 1: Number of trigger and action channels/events in our dataset.

Recipes: Each recipe in our dataset contains the user name of the author, the number of times the recipe has been adopted, and a natural language description of the recipe. It also includes the trigger channel, the specific trigger, an auto-generated description of what the trigger does, and the same information for the action. For our analysis, we only need to consider the trigger channel, trigger event, the action channel, and the action event. For example, for the recipe "if it gets too hot, then open the window," we extracted the trigger channel as `nest_thermostat`, trigger event as `temperature_rises_above`, action channel as `smartthings`, and action event as `unlock`. From the original dataset we extract all the unique recipes, trigger channel and event pairs, and action channel and event pairs. Table 1 summarizes the number of channels and events available in our dataset.

As recipes can be made public by their authors for others to use, we wanted to see how often recipes were adopted by other IFTTT users. To analyze this we first aggregate the adoption numbers for equivalent recipes (i.e., recipes with the same channel and event pair for both action and trigger); our analysis examines only unique recipes. Fig. 4 shows the distribution of recipes by adoption number. Recipes differ sharply in the number of times other users adopt them. Very few recipes were adopted more than 100,000 times, with the majority adopted between 1 and 100 times.

Channels: We identify the most popular channels for which users were writing recipes. Fig. 5 shows the top eight trigger channels used in recipes from our dataset. Users predominantly write recipes for social media and online services with Weather being the most popular trigger channel. The most popular action channels are similarly biased towards online services and social media (Fig. 6), with Google Drive being the channel for most actions.

4.2 Methodology

We encoded both our analytical model and the dataset in Prolog [9]. We chose Prolog because recipes resemble Prolog clauses and the way recipes are triggered resembles proof search in Prolog. Our Prolog encoding is an executable model for the recipes and our analysis is written as Prolog queries. Given our model, we analyzed the risks posed by individual recipes as well as recipe chains. In the remainder of this subsection, we describe how we assigned labels to the actions and triggers and how we actually linked recipes together to identify chains.

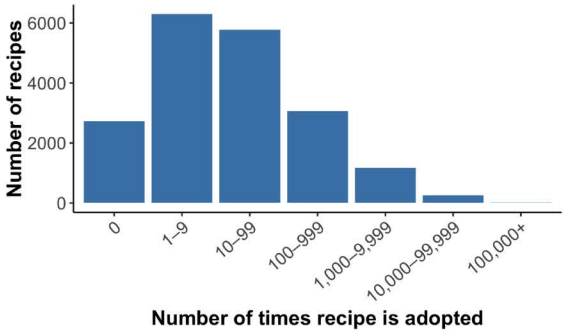

Figure 4: Adoption distribution of IFTTT recipes.

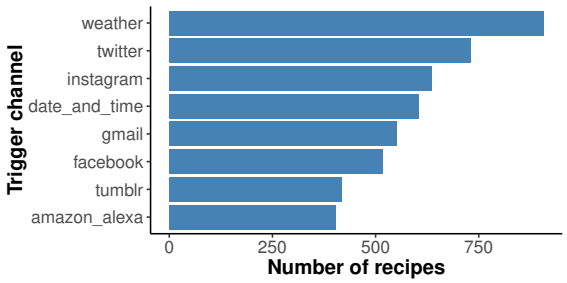

Figure 5: Top eight trigger channels used by users in our dataset.

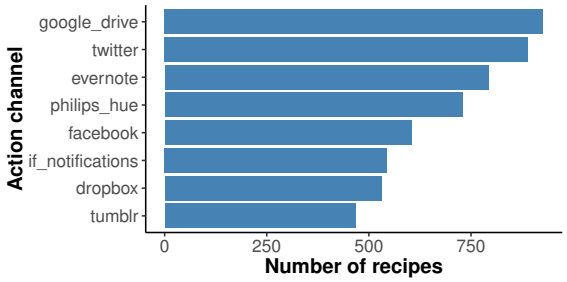

Figure 6: Top eight action channels used by users in our dataset.

4.2.1 Assigning labels to triggers and actions

After devising a preliminary set of labels, we assigned one or more labels to each trigger and action based on the recipe description and what we know about the particular channels. After running initial analyses, and after revising the labels, we revisited and adjusted the assignments. As a final confirmation that the label assignments were reasonable, two of the authors took a random sample of 100 recipes and relabeled them. They used the first 20 recipes as a training exercise, discussing the labeling together, and then labeled the remaining 80 recipes independently. The coders used a scoring system that awarded one point for full agreement, 0.5 points for partial agreement (in the case of multiple labels), and zero points for full disagreement. 78% of the time coders agreed on both labels (or label sets); the remaining 22% of the time one coder's labels were a superset of those applied by the other coder (there was no instances of full disagreement among the coders).

4.2.2 Identifying linked recipes

Although recipe chains are simple to define in the abstract, identifying chains among our actual recipes is less straighforward.

Direct linking: Determining whether two recipes are directly linked requires us to match the text description of an action to that of a trigger. For example, in the chain formed by forwarding one's Twitter post to Facebook and then from Facebook to LinkedIn, the action of posting to Facebook and the trigger of posting to Facebook are "Create a link post" and "New link post by you", respectively. While the link between these two descriptions can be discovered easily by manual inspection, identifying such links automatically is challenging. Our approach was to manually rewrite some of the actions to match their corresponding triggers. For example, if an action is "turn switch on" and a matching trigger is "switch turned on", we would rewrite the action as "switch turned on". This is feasible because the match can only happen for actions and triggers in the same channel and the number of channels in the dataset is limited.

Physical connections: Connections between recipes through the physical medium are often subtle. For example, turning off a fan via a smart plug will cause the temperature in a room to rise, which may trigger a recipe whose trigger is the temperature rising above a threshold. However, the smart plug and the thermostat are different IFTTT channels, and the action and trigger have seemingly unrelated descriptions, making it difficult to automatically recognize such connections. Nevertheless, physical connections are likely and interesting events within a smart home.

To track physical connections, we labeled relevant triggers and actions with a physical channel (e.g., temperature, sound, or light) and a physical event (e.g., `level_change_up`, `turn_on`). Physical events fall roughly into two categories: those that change the level of a physical medium and those that check for a change in level, e.g., turning on a heater, and a Nest thermostat checking if the temperature is above a threshold. Some triggers depend on the external state of a device, such as an alarm system. If an alarm system is armed, and something happens to trigger motion, it could also cause a recipe that responds to the alarm system being triggered to fire; but if the alarm is not armed nothing would happen. Tracking situations like these required that we explicitly model the state related to some of the triggers and actions. For our dataset, alarm is the only trigger we found whose state influences linking between recipes.

One particular issue we faced was the possible relationship between power and temperature events. Based on our observation of recipe descriptions, the devices being controlled by power switches, such as heaters and fans, often affect temperature. Therefore, in our model we associate actions that turn power switches on or off with triggers that fire on temperature changes. This assumption leads to false positives in our analysis, since a precise analysis depends on what device the power switch controls.

Another issue with physical connections is that connections often depend on the location of the devices themselves. If there is motion in the backyard, but the user has a motion sensor in the front yard, nothing will be triggered. Further, many connections depend on the values that users specify for their recipes. For example, a user can have a recipe with an action that sets his thermostat to a specific temperature chosen by the user when the recipe is created. The same user has another recipe that triggers when the temperature inside his home is above a certain threshold. These two recipes will connect only if the temperature he selected for the first recipe is above that threshold used in the second recipe. Without knowledge of specific values of the ingredients, we can only model potential connections, not necessarily actual ones.

4.3 Recipe-level Violations

Out of the 19,323 unique recipes, we found that 9,637 recipes (49.9%) were *unsafe*, as they involved either a secrecy or an integrity violation. 4,432 recipes (22.9%) contained only integrity violations, 3,220 recipes (16.7%) only secrecy violations, and 1,985 recipes (10.3%) both secrecy and integrity violations. These numbers include the recipes with both *definite* and *maybe* violations (defined in Section 3.2). If we consider only definitely violating recipes, there are 7,150 (37.0%) unsafe recipes: 3,605 (18.7%) with only integrity violations, 1,927 (10.0%) with only secrecy, and 1,618 (8.4%) with both. Fig. 7 illustrates these results. There is no correlation between the the number of times a recipe is shared and the probability that it is unsafe.[3]

We next investigate the number of violations that occur between specific pairs of labels. For this, we focus on total violations, i.e., including maybes; results based on only definite violations are roughly similar. When counting violations for a recipe with multiple combinations of trigger and action labels, we count each violating combination as one whole *maybe* violation for that particular pair of trigger and action labels. For instance, if the trigger's

[3]Nagelkerke's pseudo $R^2 \approx 0$

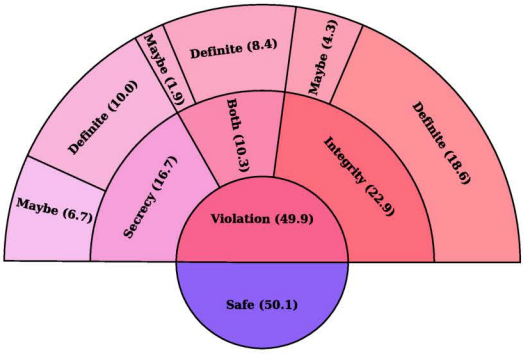

Figure 7: Distribution of safe and unsafe recipes (in %).

secrecy label can be either *private* or *restricted-online*, and the action's is *public*, then we will count this as two maybe violations, one for the flow from *private* to *public* and one for the flow from *restricted-online* to *public*. This does mean that the number of violations will be larger than number of violating recipes.

The majority (around 66%) of the 5,685 secrecy violations are leaks from triggers labeled *private* to actions labeled *restricted-online* or *restricted-physical*, as shown in Fig. 8. An example of such violation is the recipe which changes a user's Facebook status based on word search in her Google calender, potentially leaking (by accident) private information about her appointments to the public or to her Facebook friends, depending on the privacy settings used for the post. In addition, 18% of the secrecy violations are leaks between triggers and actions in the restricted circles. In sum, 84% of secrecy violations involve recipes that have their action labeled as sharing information with a restricted group, which suggests that IFTTT users do not care about or do not notice when their private information could be observed by restricted circles. On the other hand, only 16% of the secrecy violations have destinations with the *public* label, which might suggest that IFTTT users are usually hesitant to share their private or somewhat private information on public channels.

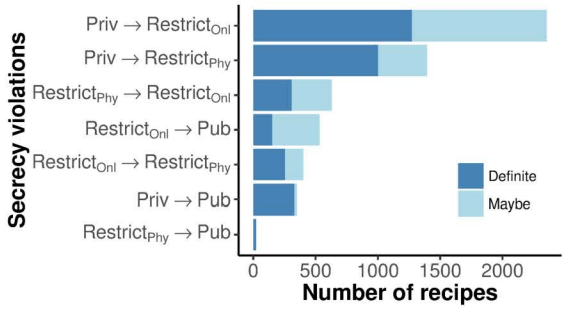

Figure 8: Secrecy violations by trigger and action label.

Most of the 7,782 integrity violations (83%) are to actions that are labeled *trusted*, with half (50% in total) originating from the *trusted other* or *untrusted* sources, as shown in Fig. 9. This fact along with the fact that integrity violations are more frequent than secrecy violations, could indicate that IFTTT users either are not aware of integrity violation risks or are not concerned about them.

Consistently with this, the most frequent violation, from *trusted-other* to *trusted* is usually benign. In most cases, it involves a recipe that is triggered by an action on a well known website, such as the New York Times or ESPN websites. Although relatively benign, such recipes are nevertheless technically unsafe, since each effectively gives a somewhat trusted source access to possibly sensitive resources, such as a user's indoor lighting or her phone's wallpaper.

More concerning is the second largest category of integrity violations, from *untrusted* to *trusted*. This kind of violation is more serious because the recipe can be triggered by a larger set of people, usually with relative ease. For example, the recipe that updates a user's smartphone wallpaper whenever an image is posted in a *subreddit* enables anyone with a Reddit account to change the user's device wallpaper to an embarrassing photo.

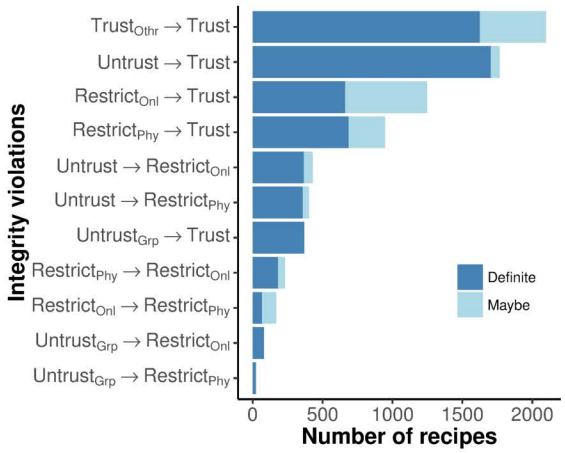

Figure 9: Integrity violations by trigger and action label.

4.4 User-level Recipe Violations

An average IFTTT user will have a small set of recipes active in his or her account. Unfortunately, aside from recipe adoption statistics, we do not have any data on what sets of recipes individual users actually use in their account. Therefore, to understand the risks for a single IFTTT user we first need to identify small sets of recipes that resemble the recipe sets of individual users. Another motivation to analyze small sets of recipes is examining recipe chains. Studying recipe chaining in the whole dataset is uninsightful, since in practice chaining can take place only among the recipes adopted by an individual user.

To construct recipe sets we need to answer the following questions: (1) What is the average number of recipes an individual user uses? (2) How do we choose the recipes themselves?

To answer the first question we investigate articles reporting average number of unique users using IFTTT recipes and average number of recipes run by IFTTT on a daily basis. In early 2015, several articles reported that IFTTT had around 20 million recipes executed daily and approximately 600 million each month [7, 2]. In the same time frame, IFTTT was reported to have more than one million unique active users who run recipes daily [4]. Given this, we assume that in early 2015, each user would, on average, run about 20 recipes every day. The actual number would vary by user, and may have increased since the time these numbers were reported; however, for our purposes we consider this rough approximation sufficient, and analyze recipes in sets of 20, 30, and 40.

To answer the second question we used two strategies to sample recipes from our dataset. In the first strategy we selected a set of $n + m$ recipes, including the n most-frequently adopted recipes and m recipes selected uniformly at random from the remaining recipes. The reasoning behind this strategy is that a user is likely to select some of her recipes from the most used recipes, but will also select, or create, the remaining recipes based on her unique needs. Our second strategy was to select all the recipes randomly but weighted by the frequency of adoption of each recipe. We report on the results of analyzing recipes selected through each of

	Recipe Sampling	Weighted 30			Top10+Rand10			Top20+Rand10			Top30+Rand10		
		mean	med	stddev	mean	med	stddev	mean	med	stddev	mean	med	stddev
Recipes	Unsafe Recipes	8.4	8	2.4	6.6	7	1.5	10.6	11	1.6	14.5	14	1.5
	inc. maybes	*13.2*	*13*	*2.5*	*12.0*	*12*	*1.6*	*19.0*	*19*	*1.6*	*26.0*	*26*	*1.6*
	Sec. Viol.	2.2	2	1.4	2.5	2	1.1	2.5	2	1.2	3.5	3	1.1
	inc. maybes	*6.8*	*7*	*2.2*	*6.6*	*7*	*1.4*	*7.7*	*8*	*1.4*	*12.7*	*13*	*1.4*
	Int. Viol.	7.2	7	2.3	4.5	4	1.4	8.5	8	1.4	11.5	11	1.4
	inc. maybes	*9.3*	*9*	*2.4*	*7.3*	*7*	*1.5*	*13.3*	*13*	*1.5*	*16.2*	*16*	*1.5*
Chains	Number	1.2	1	1.4	0.3	0	0.6	1.7	1	1.0	2.0	2	1.1
	Avg. Length	2.0	2	0.2	2.0	2	0.0	2.0	2	0.2	2.0	2	0.2
	Unsafe	0.6	0	0.9	0.2	0	0.4	1.5	1	0.8	1.7	1	0.9
	inc. maybes	*1.0*	*1*	*1.2*	*0.3*	*0*	*0.5*	*1.6*	*1*	*0.9*	*1.8*	*2*	*1.0*

Table 2: Simulating user-level violations for different size of recipe sets. Considered different sampling techniques for selecting the recipes.

these strategies. We found that the choice of strategy does not make a significant difference to the results of our analysis.

For each sampling strategy, the experiment was repeated 500 times. We summarize our results in Table 2. The average number of unsafe recipes was above 28% (44% including *maybes*) for all strategies. This tells us that an average IFTTT user will have a significant number of recipes that potentially impose security and privacy risks. Another interesting finding is that the average user who has at least 30 recipes will have at least one recipe chain. The average length of recipe chains is two regardless of how we select the set of recipes to analyze. This means that the likelihood of more than two recipes chaining together is low, unless a user intentionally adopts recipes that chain. In fact, among the top 50 recipes there are no unsafe chains. Finally, more than half of the chains include unsafe recipes. If an unsafe recipe is a part of a chain then its consequences could be exacerbated, especially if the other recipe is also unsafe. An example of recipes chaining, and potentially exacerbating risk through doing so, is as follows: One recipe allows any Facebook user who can tag the victim in a picture to cause that picture to be added to the victim's iOS album; another posts any new photo in the iOS album publicly to the victim's Flickr account.

5. IMPLICATIONS OF VIOLATIONS

Once we generated the list of recipes that involved information-flow violations, we wanted (1) to confirm through manual examination that such recipes could often cause users harm; and (2) to categorize the types of harm that could be inflicted.

To do this, we started by examining some recipes manually, and we found that we can categorize the potential harms (risks) into four broad groups as described below–

- Personal: Cause embarrassment or leak behavioral data
- Physical: Damage physical health or property or goods
- Cyber Security: Disrupt online service or distribute malware
- Innocuous: Seemingly harmless

The most noticeable harm was of personal nature, like causing personal embarrassment by leaking sensitive pictures or current location. We also found recipes that can cause—or increase the risk of—physical harm such as making break-ins easy, damaging the physical hardware of IoT devices, or triggering migraines or other health conditions. Other recipes could be exploited by a malicious attacker to distribute malware through emails or even enable him to carry out denial-of-service like attacks to disrupt online services. For some recipes we could not envision a realistic harm (i.e., any possible harm simply seemed too far fetched) even though the assigned labels seemed logically correct. There is no way to get rid of these violations without having contextual information about the user's exact situation, so we categorize them as false positives and call them *innocuous*.

Another interesting aspect of the threat model is that not all violating recipes need an explicit attacker to have an adverse consequence to the user. A recipe that uploads daily fitbit statistics to twitter has the potential to embarrass the user without any interference from a third party, whereas the recipe that uploads all email attachments to OneDrive will only be harmful if someone is spamming or sending malicious attachments. To capture this distinction, we give each harmful recipe an additional label of *self* or *external*.

For illustration, we provide one example recipe for each category of potential harm.

Personal: *If I take a new photo, then upload on Flickr as public photo.* This recipe could leak sensitive or embarrassing information if one took a picture of a check to send to landlord, or a picture of one's romantic partner. This harm is labeled as *self-inflicting* as any harm is the result of the user's own behavior.

Physical: *If the last family member leaves home, then turn off lights.* This recipe, by turning off the lights in a predictable fashion, signals that your home is empty, making it easier for a burglar to plan the opportune time to rob the place. This harm is labeled as *external* as a third-party can potentially inflict the harm.

Cyber Security: *If there is a new email in your inbox with an attachment, then add that file to OneDrive.* This recipe could be used to spread malware to all devices synched with a OneDrive account. If a malicious attachment gets propagated to all synched devices, it increases the probability that the file will be opened by the user, especially since it is removed from the suspicious context (i.e., the email). This harm is labeled as *external* as a third-party can potentially inflict the harm.

Innocuous: *If a smart switch turns off, then append to a text file in Dropbox.* Technically, there is an integrity violation here because anyone who can access the switch can fill your Dropbox space, however, this recipe cannot be spammed as a single misuse would only generate a few bytes of data (assuming the attacker does not control your smart switch, in which case the attacker can inflict greater harm anyway).

After determining the potential categories of harm, we next try to quantitatively measure how many of the violating recipes fall into each these categories. To do this, we manually examined randomly

selected 200 recipes from all the violating recipes in our dataset. The first 20 recipes were used for training; two coders together assigned one or more harm labels to each recipe and discussed the rationale for their decision. For the remaining 180 recipes, the two coders independently categorized each recipe's harm(s) and came together to compare results. If the labels matched exactly, we awarded one point of agreement, for partial match (in the case of multi-labels) we awarded 0.5 points of agreement, and if the coders decided one of them had fully mislabeled the harm category for a recipe, we awarded zero points. Initially the coders agreed on 87% of the categorization and were able to fully agree after discussing the remaining conflicts. There were no disagreements over whether a recipe needed a third party to be harmful (i.e., in the task of assigning *self* or *external* labels).

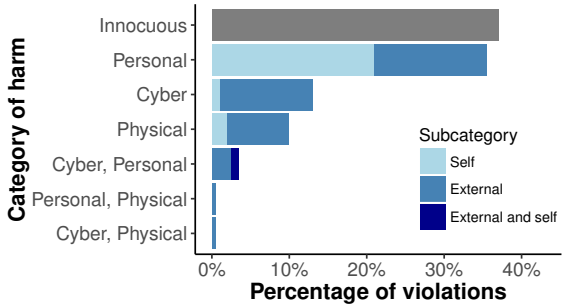

Figure 10: Categorical analysis of the implications of information-flow violations. *External and self* denotes that the *cyber* harm is external and the *personal* harm is self-inflicted.

Fig. 10, highlights our findings after performing inter-coder agreement. As seen in Fig. 10, a little more than a third of the recipes were innocuous, or not likely to cause harm. Most of the harmful recipes had personal consequences, leaking unintended information or causing embarrassment or agitation. The next largest category is cyber security harms, dealing with damage to digital devices and cloud services, at around 13%. Recipes that may increase the risk of physical damage are less common, accounting for around 10% of the violating recipes, and there are a handful of recipe that have multiple types of harm.

It is worth noting that there can be overlaps between categories and side effects that we could not determine just from the recipes. For example, damage to a device will probably cost some money to fix or replace, and embarrassments that significantly effects one's public image can have far-reaching repercussions, including financial ones. If a recipe leads a user to accidentally post a politically controversial tweet, this could cost her friendships and other potential opportunities. It is also interesting that most violations have outcomes that involve data being saved or published online, primarily through social media or cloud services, and therefore can have diverse indirect consequences. For example, many recipes involve posting private information to online services where this information may further be used in different ways; if the original posting of private information was unintended, the consequences of that accident may be far-reaching and hard to quantify.

6. RELATED WORK

With the widespread use of various online services and IoT devices, end-user programming has received much research attention in the last decade as it allows users to easily interface different devices with other devices or online services. Researchers have looked at how an average user can customize their smart home using trigger-action programming [30, 13, 32], while others have also extended such automation techniques to commercial buildings [26]. Researchers have also studied the usability of existing trigger-action programming frameworks and have propose guidelines for developing more user-friendly interfaces [13, 21, 11, 29, 15]. Recently, there have been efforts to built semantic parsers that automatically map IFTTT style recipes described in natural language to actual executable codes [27, 12].

While end-user programming makes it easy for users to write automation rules, it does complicate things when multiple users share the same environment and try to enforce their own set of rules. In such scenarios not only conflicts among individual rules arise, but also unforeseen chaining of rules start to emerge. Researchers have studied such conflicts and have proposed ways to resolve such conflicts for home and office environments [24, 26, 22]. A similar kind of conflict resolution study has also been done in the context of designing a smart city [23].

Information flow based security analysis has long been an active field of research. Information flow controls enable us to track the propagation of information in and across systems, and thus help us prevent sensitive information from being released. Dorothy E. Denning's work on using a lattice model to guarantee secure information flow in a computer system [14] was seminal in this field. Later on, Sabelfeld et al. provide a comprehensive survey of the past three decades of research on information-flow security [28]. Myers et al. show how to incorporate language-based information flow controls into simple imperative programming languages [25]. In the era of smartphones, information-flow based analytic frameworks such as TaintDroid [17] and PiOS [16] enabled users to track how their private data is being used by third-party applications. With the fast adoption of IoT devices in the recent years, researchers are now focusing on analyzing the security and privacy risks of IoT devices. But like any emergent technology, IoT too is rife with potential security risks [18]. Moreover, current IoT programming frameworks only support permission-based access control on sensitive data and are ineffective in controlling how sensitive data is used once access is gained. FlowFense [19] framework provides such control by imposing developers to declare the intended data flow patterns for sensitive data. Our work looks at analyzing the security and privacy risks of end-user programming frameworks like IFTTT without requiring fine-grained user settings.

7. DISCUSSION AND LIMITATIONS

We next discuss some implications of our results, the limitations of our approach, and how applets, the new form of recipes, impact our overall analysis.

7.1 Levels of Concern and Intended Leaks

Our breakdown of violating recipes based on the labels of triggers and actions reveals some interesting insights. One insight is that violating recipes vary significantly in the amount of risk to which they expose users. For example, an information flow from *private* to *public* is more concerning than from *private* to one of the restricted groups. Sharing one's health related information (e.g., a report generated by a Fitbit device) with one's Facebook friends might be indiscreet, but it is probably less risky than sharing it on a public web forum. Likewise, an integrity violation caused by an *untrusted* trigger is more concerning than one caused by a restricted trigger. A family member or a friend figuring out he can open a window by increasing the temperature inside your home may not use this knowledge maliciously, whereas a stranger might.

As we discuss in Section 4.3, for most pairs of labels that are indicative of a secrecy violation, there is a significant number of recipes that use them as trigger and action labels. The largest such sets are recipes that pass information from *private* to *restricted on-*

line and from *private* to *restricted physical*. Users are leaking private information, but mostly to specific groups of people rather than the public at large. The types of integrity violations are less varied: there are few recipes that connect restricted groups, with the vast majority of violating recipes going from *trusted_other* to *trusted* and from *untrusted* to *trusted*.

In retrospect, these are the types of violations to be expected. Two major uses of IFTTT recipes are controlling smart home devices and pushing information to social media. It is easy for the latter to result in leaking private information. In the former case, many IFTTT actions are configured so that the action appears to come from the user, or at least an account controlled by the user. Indeed, this seems inherent in the idea of automation for convenience: a user gives up direct control of something he would do manually and is instead allowing a program to automate it, sometimes based on untrusted input.

Although many violating recipes are likely consistent with what users intended, their behavior is not necessarily innocuous. Harmful side effects are probably not at the forefront of the user's mind, especially since much of the conversation surrounding the ability to arbitrarily link devices and services focuses on the convenience and novelty of it. Since the concept *is* innovative, IoT-specific security principles and behaviors are not yet in the public awareness. For example, people generally do not have to worry about any *new* emerging threats for an old, non-smart home. While insecurities exist for conventional homes, attack vectors and their respective defenses are generally well known. Connecting one's home with recipes can introduce new methods of attack of which the user might not be cognizant, and, thus, for which good safeguards might not be in place. As shown in our harm analysis, even recipes seemingly working as intended can have harmful consequences that the user might not have considered when adopting the recipe. This suggests that it is important to educate users about the potential risks inherent in many of their recipes, so that they can make rational decisions about whether the harms outweigh the benefits or at least be aware that new safety measures may be necessary.

7.2 Limitations

Some major limitations of our work follow from its abstract nature. We made design choices that resulted in loss of some detail in order to keep the model general enough to be useful without user-specific parameters (which we do not have). We used fairly broad security labels, labeled device APIs as opposed to data, and chained together recipes that might not actually interact, all of which leads to overapproximating the number of violating recipes. We also lacked concrete data about which specific sets of recipes individual users adopted and so had to approximate such sets.

Based on these limitations, we see a few directions for future work. We would like to make these analyses more applicable to an individual user, perhaps by creating a web interface that would inform users of security and privacy risks as they write their recipes. Making such a tool would remove some of the weaknesses of our current approach, as users' specific instantiations of recipes could be used to perform the analyses more precisely. In this interface, we could also create explicit declassification and endorsement functions that users would utilize to indicate when they are knowingly creating or adopting recipes that contain information-flow violations. In addition to developing such a tool or interface, it would be helpful to conduct user studies to collect data on the number and types of recipes that specific users adopt, as well as to elicit information about their awareness and perception of violations among their recipes, and of IoT security concerns in general.

7.3 Applets: Enhanced Recipes

In November 2016 IFTTT introduced enhanced recipes called *applets* and converted existing recipes to applets. Applets add three major capabilities, which we discuss below, along with their implications for our analysis. Only registered IFTTT partners can create applets with these new features; regular users can only adopt existing applets or create new ones that are equivalent to recipes.

Multiple actions: While recipes were limited to only one action per recipe, applets can have multiple actions. This does not impact our analysis, as an applet with n actions can be replaced with n recipes with the same trigger.

Queries: Previously, an action would only contain information from the trigger channel. Applets can execute *queries* to acquire additional information about the state of *query channels*. For example, an applet that sends a daily email about the user's energy consumption could query the user's Harmony remote, a query channel, to check if the TV is on or off. We may need to augment our model to include new secrecy and integrity labels for query channels and assign each query channel appropriate labels.

Conditions: With applets one can include code to conditionally execute the actions. For example, a recipe that sends you an SMS whenever it is raining can be made to not execute during the night. We need to augment our model and analysis to consider flow of information from conditions to actions. In particular, if the results from queries can be used in conditions, then we need to make sure that the query channel labels are also considered to identify information leakage from query channels to the action.

8. CONCLUSION

Recipes with potential insecurities are endemic on IFTTT. Many recipes are seemingly benign, but can cause personal, digital, or even physical harm. While many of these risks exist in some form without using recipes, such as oversharing to social media or smart devices taking away control, the fact that they are happening through channels that the user might not have considered harmful is of significant importance. If users do not realize that there is a danger, they will do nothing to guard against it, rendering them more susceptible to attacks. The prevalence of potential risks among the recipes we examined strongly suggests that users need to be informed about the security and integrity violations that their recipes can potentially create, and of the potential consequences of such violations, so that they can make well-informed decisions. In this context, our work provides a foundation for tools and practices to better inform and help users manage the risks they face.

Acknowledgments

This work was supported in part by NSF grants CCF-1560137 and CCF-1320470, a gift from Google, the Army Research Laboratory under Cooperative Agreement number W911NF-13-2-0045 (ARL Cyber Security CRA), and DARPA and the Air Force Research Laboratory under agreement number FA8750-15-2-0277. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsement, either expressed or implied, of DARPA, the Air Force Research Laboratory, the Army Research Laboratory, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation here on.

9. REFERENCES

[1] 2016 on IFTTT. https://ifttt.com/blog/2017/01/year-in-review, Accessed Jan. 2017.

[2] The future is this one-click remote for everything in your life. http://qz.com/346767/ifttt-pares-down-its-automation-service-to-prepare-for-the-one-click-smartwatch-future/ Accessed Oct. 2016.

[3] Gartner says the Internet of Things will transform the data center.

[4] How IFTTT is taking a big swing at being a connective tissue for IoT. http://www.techrepublic.com/article/how-ifttt-is-taking-a-big-swing-at-bringing-connectedness-to-a-connected-world/ Accessed Oct. 2016.

[5] How IoT and smart home automation will change the way we live. http://www.businessinsider.com/internet-of-things-smart-home-automation-2016-8 Accessed Feb. 2017.

[6] IFTTT. https://ifttt.com, Accessed Oct. 2016.

[7] IFTTT launches 3 "Do" apps to automate photo sharing, tasks, notes; rebrands main app "IF". https://techcrunch.com/2015/02/19/ifttt-launches-3-do-apps-to-automate-photo-sharing-tasks-notes-rebrands-main-app-if/ Accessed Oct. 2016.

[8] IFTTT services. https://ifttt.com/search/services, Accessed Feb. 2017.

[9] Prolog. http://www.swi-prolog.org/, Accessed Oct. 2016.

[10] K. J. Biba. Integrity considerations for secure computer systems. Technical report, MITRE Corp., 04 1977.

[11] F. Cabitza, D. Fogli, R. Lanzilotti, and A. Piccinno. End-user development in ambient intelligence: A user study. In *Proceedings of the 11th Biannual Conference on Italian SIGCHI Chapter*, CHItaly 2015, 2015.

[12] X. Chen, C. Lu, R. Shin, M. Chen, and D. Song. An end-to-end approach for natural language to IFTTT program translation. In *Proceedings of the 2016 Neural Information Processing Systems (NIPS)*, NIPS '16, 2016.

[13] L. De Russis and F. Corno. HomeRules: A tangible end-user programming interface for smart homes. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, 2015.

[14] D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, May 1976.

[15] A. K. Dey, T. Sohn, S. Streng, and J. Kodama. iCAP: Interactive prototyping of context-aware applications. In *Proceedings of the 4th International Conference on Pervasive Computing*, 2006.

[16] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting privacy leaks in iOS applications. In *Proceedings of Network and Distributed System Security Symposium*, 2011.

[17] 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)*, 32(2):5, 2014.

[18] E. Fernandes, J. Jung, and A. Prakash. Security analysis of emerging smart home applications. In *Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[19] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash. FlowFence: Practical data protection for emerging IoT application frameworks. In *Proceedings of the 25th USENIX Security Symposium*, 2016.

[20] J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, 1982.

[21] J. Huang and M. Cakmak. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, 2015.

[22] C.-J. M. Liang, B. F. Karlsson, N. D. Lane, F. Zhao, J. Zhang, Z. Pan, Z. Li, and Y. Yu. SIFT: Building an internet of safe things. In *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, 2015.

[23] M. Ma, S. M. Preum, W. Tarneberg, M. Ahmed, M. Ruiters, and J. Stankovic. Detection of runtime conflicts among services in smart cities. In *Proceedings of 2016 IEEE International Conference on Smart Computing*, 2016.

[24] S. Munir and J. A. Stankovic. DepSys: Dependency aware integration of cyber-physical systems for smart homes. In *Proceedings of the 5th International Conference on Cyber-Physical Systems*, 2014.

[25] A. C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification. In *Proceedings of the 17th IEEE Computer Security Foundations Workshop*, 2004.

[26] A. A. Nacci, B. Balaji, P. Spoletini, R. Gupta, D. Sciuto, and Y. Agarwal. BuildingRules: A trigger-action based system to manage complex commercial buildings. In *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, 2015.

[27] C. Quirk, R. J. Mooney, and M. Galley. Language to code: Learning semantic parsers for If-This-Then-That recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

[28] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on selected areas in communications*, 21(1):5–19, 2003.

[29] K. Tada, S. Takahashi, and B. Shizuki. Smart home cards: Tangible programming with paper cards. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct*, UbiComp '16, 2016.

[30] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, 2014.

[31] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman. Trigger-action programming in the wild: An analysis of 200,000 IFTTT recipes. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, 2016.

[32] J.-b. Woo and Y.-k. Lim. User experience in Do-it-yourself-style smart homes. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '15, 2015.

[33] S. A. Zdancewic. *Programming Languages for Information Security*. PhD thesis, 2002.