Identifying Privacy Weaknesses from Multi-party Trigger-action Integration Platforms

Kulani Mahadewa
School of Computing,
National University of Singapore,
Singapore
kulani41@comp.nus.edu.sg

Yanjun Zhang
School of ITEE,
University of Queensland, Australia
yanjun.zhang@uq.edu.au

Guangdong Bai∗
School of ITEE,
University of Queensland, Australia
g.bai@uq.edu.au

Lei Bu
State Key Laboratory for Novel
Software Technology,
Nanjing University, China
bulei@nju.edu.cn

Zhiqiang Zuo
State Key Laboratory for Novel
Software Technology,
Nanjing University, China
zqzuo@nju.edu.cn

Dileepa Fernando
School of IT and Computing,
Sri Lanka Technological Campus,
Ingiriya Road, Padukka, Sri Lanka
dileepaf@sltc.ac.lk

Zhenkai Liang
School of Computing,
National University of Singapore,
Singapore
liangzk@comp.nus.edu.sg

Jin Song Dong
School of Computing,
National University of Singapore,
Singapore
dcsdjs@nus.edu.sg

ABSTRACT
With many trigger-action platforms that integrate Internet of Things (IoT) systems and online services, rich functionalities transparently connecting digital and physical worlds become easily accessible for the end users. On the other hand, such facilities incorporate multiple parties whose data control policies may radically differ and even contradict each other, and thus privacy violations may arise throughout the lifecycle (e.g., generation and transmission) of triggers and actions. In this work, we conduct an in-depth study on the privacy issues in multi-party trigger-action integration platforms (TAIPs). We first characterize privacy violations that may arise with the integration of heterogeneous systems and services. Based on this knowledge, we propose Taifu, a dynamic testing approach to identify privacy weaknesses from the TAIP. The key insight of Taifu is that the applets which actually program the trigger-action rules can be used as test cases to explore the behavior of the TAIP. We evaluate the effectiveness of our approach by applying it on the TAIPs that are built around the IFTTT platform. To our great surprise, we find that privacy violations are prevalent among them. Using the automatically generated 407 applets, each from a different TAIP, Taifu detects 194 cases with access policy breaches, 218 access control missing, 90 access revocation missing, 15 unintended flows, and 73 over-privilege access.

∗Guangdong Bai is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM Reference Format:

1 INTRODUCTION

Various trigger-action services, such as IFTTT [27], Zapier [44] and Microsoft Flow [35], add further capability to integrate the Internet of Things (IoT) systems with online services. We call these new multi-party facilities trigger-action integration platforms (TAIPs). A TAIP consists of the trigger-action service, the participating trigger and action service, and the interfaces provided by them to support applet creation and execution. With the flexible approaches provided by various TAIPs, new functionalities like “if the user is tagged in a photo on Facebook, then blink the Philips Hue bulb in the living room” can be easily realized with almost no programming skill required. The cores of the TAIPs, i.e., the trigger-action services, are on the rise in the current internet market. According to recent statistics [28], IFTTT, the most popular trigger-action service, has 19 million users, more than 700 services integrated and over 90 million active connections as of January 2021.

Same as in all other IoT-related systems, security and privacy of the TAIPs should be a perennial concern. Previous studies have focused on the individual components of trigger-action applications, including IoT devices [10, 18, 24, 32, 37], application frameworks [3,
In this work, we study the privacy preservation of the multi-party TAIP from an integration perspective. We focus on the weakness in the TAIP implementations that may lead to privacy violations when they integrate heterogeneous parties to fulfill particular functionalities. A typical case of such violations is the cross-party policy enforcement, which occurs when the data access control policy enforced by one party is infringed by others intentionally or unintentionally. For example, a user can upload his file to Dropbox with a restriction of sharing within a group of friends, and the file is well protected under Dropbox’s access control mechanism. Nevertheless, most of the current trigger-action services simply grab the file and send it to the action service, once any applet declares to use “file uploaded to Dropbox” as a trigger, regardless of the user’s intention.

One may argue that the user should be responsible to control the flow of their private information. For example, Facebook’s data policy says “When you (the users) choose to use third-party apps, websites, or other services that use, or are integrated with, our Products, they can receive information about what you post or share” [14]. The challenge is that even though the users have been aware of the flow of their private information from one service to another, existing TAIPs fail to provide users with complete control over their own data. For example, our study finds that after pulling a file from a trigger service (e.g., a photo taken by an Android device), IFTTT makes a copy on its server and exposes a URL to the action services (e.g., a smart device or a social media service). The copy is stored permanently and there is no way for the user to delete it. More seriously, even if the original file residing in the trigger service is deleted, IFTTT’s copy remains accessible through the URL. Unfortunately, we find that such misunderstandings on the shared responsibility are not uncommon.

The cause for these issues are at least twofold. First, the parties involved in a TAIP belong to different manufacturers and service providers. Each individual party may have applied policies that are sufficient in their context, but these policies may be violated by others due to misunderstandings or even complete unawareness. Second, a single trigger-action service usually supports hundreds of online services/IoT devices available in the market. The number of combinations can easily reach thousands, and each of them may takes numerous trigger events as inputs. For instance, IFTTT supports 485 trigger services, 414 action services, which results in 200,790 service-level possible ways of integration without even considering the event-level combinations. As a result, it is challenging, if not impossible, for the trigger-action service provider to have comprehensive end-to-end testing.

We propose a black-box testing approach to identify weaknesses that may lead to privacy violations from the TAIP. We develop our approach into a trigger-action integration platform fuzzer (Taufu). Taifu adopts a two-phase workflow similar to that in compiler testing—it first generates the applets (also known as the trigger-action programs [45] and recipes [40] in the literature) that are the inputs to the trigger-action applications (analogous to the programs to a compiler under testing), and then generates triggers to drive the executions of the applets (analogous to the program inputs to the compiler-generated executable).

The first phase uses a technique that consists of a protocol inference to identify undocumented interfaces for application creation, and a natural language processing (NLP)-based corpus creation to produce expected data values for applet configuration. It aims to explore service combinations and data flows as many as possible. The second phase simulates the trigger events with another set of test cases generated using a hybrid technique including generation based on extracted protocols and test case mutation. The behavior of the trigger-action applications in response to the applet execution is monitored and checked against a set of pre-defined rules for weakness detection.

To characterize the privacy weaknesses in real-world TAIPs, we have tested Taifu against the TAIPs that comprise the IFTTT and several widely used online services, e.g., Instagram, Twitter, Android Photo and Facebook. Taifu generates applets for 407 TAIPs by synthesizing the trigger/action types from 52 services that we find may handle privacy-sensitive data. It has identified 194 cases with access policy breaches, 218 access control missing, 90 access revocation missing, 15 unintended flows, and 73 over-privilege access.

**Contributions.** To summarize, we make the following main contributions in this paper:

- We characterize the privacy weaknesses in TAIPs. To our best of our knowledge, this is the first study that explores the fundamental cause of the continuously reported privacy issues in trigger-action applications [4, 40, 43].
- We propose Taifu, a novel approach of identifying privacy weaknesses out of TAIPs. Taifu uses the automatically generated applets as inputs to activate data flows in the TAIPs as many as possible.
- We have applied Taifu to characterize the privacy weaknesses in TAIPs that take IFTTT as the core trigger-action service. It has found multiple privacy weaknesses. We release our implementation and experiments online [41].

## 2 MULTI-PARTY TAIPS AND A RUNNING EXAMPLE

In this section, we summarize a generic architecture of TAIPs which shows the integration of multiple services and the data flows among them. We also present a running example that is used later to explain the privacy violations in Section 3 and our approach in Section 4.

### 2.1 Trigger-action Integration Platforms

Figure 1 shows a generic representation of the architecture and workflow of a trigger-action application. It is summarized from a manual study on the IFTTT-based TAIPs, with a focus on the documentations, and reverse engineered workflows of several mostly integrated services. A TAIP typically involves three parties, i.e., the trigger-action platform, the trigger system, and the action system. In the following we brief them.

**Trigger-action Platform.** The trigger-action platform provides interfaces for the services and devices to interact with each other for automation tasks. It consists of two main components, i.e., the trigger-action service (e.g., a cloud service) and the trigger-action...
Client (e.g., a mobile or a web client). The trigger-action service facilitates channels for the online trigger services and action services to communicate with each other. The interfaces between any two services are subject to the pre-established agreements between their providers.

Trigger System and Action System. A third-party system can be integrated with the trigger-action platform to initiate triggers (i.e., the trigger system), complete actions (i.e., the action system), or both. Similar to the trigger-action platform, each system consists of a cloud service and a client (e.g., a device, a mobile or a web client). The trigger service initiates the automation tasks and the action service completes them.

### 2.2 A Running Example

Given an automation task "IF a photo is uploaded with a hashtag on Facebook, THEN create a post on Blogger", the TAIP takes the following three steps to realize it.

#### Step 1: Service Connection

The user authorizes the trigger-action service to connect and communicate with the selected trigger service (i.e., Facebook) and action service (i.e., Blogger) through the trigger-action client (step 1 in Figure 1).

#### Step 2: Applet Creation

An applet is a small conditional program which specifies "IF this trigger happens, THEN fire that action". It involves a pair of events selected from the connected services as the trigger and the action (step 2). The behavior of an applet can be customized according to the user’s preferences through parameters given to the chosen events. We refer to this process as event configuration in the rest of the paper. In an applet, a set of special arguments called ingredients are used to carry trigger data. If the ingredients are provided as arguments when configuring the action, the action service can receive the trigger data. To create the applet in our example, the user selects the trigger of "New photo post by you with a hashtag" from Facebook and the action of "Create a post" from Blogger. The ingredients of the trigger include the photo and title/body of the Facebook photo post.

#### Step 3: Applet Execution

The created applet can be executed by initiating a trigger event at the trigger client (step 3). In our example, when the user uploads a photo post with a hashtag through the Facebook client, the Facebook service receives a trigger. It then informs the trigger-action service that the trigger event has occurred (step 4). The applet that is associated with the trigger event would be executed by the trigger-action service (step 5), which then sends the defined action actuation (including the data body if any is received from the trigger service) to the action service (step 6). In response, the action service completes the action (i.e., "create a post"), and pushes the update to the action client (i.e., Blogger) (step 7). We note that the trigger-action client also receives the execution log, which contains the applet identity, the execution status, and the trigger data (e.g., texts and URLs) received from the trigger service (step 8).

### 3 PRIVACY VIOLATIONS IN TAIP

In order to characterize the privacy weaknesses in the multi-party TAIPs, we seek to explore possible channels that sensitive information may exfiltrate. In this section, we discuss the threats against the user privacy (Section 3.1), and then identify sensitive flows (Section 3.2) and possible privacy violations (Section 3.3).

#### 3.1 Threat Model

The security threats caused by malicious or compromised entities [17], malicious applets [4, 9] and undesirable rule chaining [43, 45] have been studied in the literature. In this work, we explore the user data protection by the service providers as data processors and data controllers. This is motivated by a recent trend that many countries start imposing strict administrative procedures, such as the EU General Data Protection Regulation (GDPR) [21]. This has resulted in a significant development of privacy policy regulations in the domain of mobile applications [22]. The user data protection issues in TAIPs deserve equal or more attention due to their cross-party nature—whenever data transmission is involved, the regulations have highly strict requirements.

Since there is no existing taxonomy of privacy violations in a TAIP that can be used by our work, we refer to EU GDPR [21] for a guideline when we define privacy violations. GDPR regulates the protection of user data by imposing legal obligations on every data controller (i.e., all parties in our architecture except the user), as shown in Table 1. In the current design of IFTTT-based TAIPs, the user is only prompted to authorize the service connection (see step 1 in Section 2.2). Disclosing privacy policy is not compulsory at this stage, although many service providers like Twitter may inform the user of the implications. During the process of applet creation and execution, the user is not acknowledged the contents of the data to be associated with the trigger events, and the purpose and means of processing the data. This actually forwards to the user the accountability of any privacy violations (e.g., access control policy breaches) happening during/after the execution of the applet. We

| Table 1: GDPR Principles of Personal Data Processing (an excerpt from Art. 5 GDPR [20]) |
|---------------------------------|-------------------------------------------------|
| **Principles**                  | **Requirements related to this work**          |
| P1     Lawfulness, fairness and  | processed lawfully, fairly and in a transparent manner |
| P2     Purpose limitation       | "collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes" |
| P3     Data minimisation        | "limited to what is necessary in relation to the purposes" |
| P4     Accuracy                 | "accurate... erased or rectified without delay" |
| P5     Storage limitation       | "kept no longer than is necessary for the purposes" |
| P6     Integrity and confidentiality | "processed in a manner that ensures appropriate security" |

Identifying Privacy Weaknesses from Multi-party Trigger-action Integration Platforms
thus treat such scenarios where service providers fail to provide appropriate countermeasures as weaknesses.

3.2 Sensitive Data

The data flows can be categorized into explicit and implicit flows [11, 30]. The “if-then” paradigm of applets is a typical control flow based on information flow, i.e., an implicit flow. Such type of flows are pervasive in all applets, such that an action always indicates the occurrence of the corresponding trigger. A previous study by Zhang et al [45] has explored the privacy issues arising due to the chaining of applets. In this work, we aim to examine the control mechanisms of the service providers. Therefore, we exclude implicit flows out of our scope.

We then put our focus on the cross-party data flows that the data is sourced from the trigger system and transmitted to other two services when an applet is executed. We instrument the network APIs in the mobile clients and deploy network proxies between clients and services, and capture the network traffic while executing 30 most installed IFTTT applets. By manually analyzing the captured traffic, we have identified two types of trigger data as the critical assets that may raise privacy concerns of the users.

**Textual Contents.** These contents are considered sensitive as they may involve users’ social media posts/tweets, date and time, and text description of the devices/services that may reveal their identities.

**URLs.** The applets may involve users’ private photos, videos and other objects. Due to their big sizes, they are not directly included in the trigger data; instead, the URLs linking to them are attached. Therefore, the URLs are considered sensitive.

3.3 Types of Privacy Violations

With the sensitive data types identified, we then investigate how each party in the TAIP may mismanage the sensitive data. Based on the GDPR principles, we identify five types of privacy violations in the context of multi-party TAIPs.

**V1: Access Control Policy Breach** (rel. P6). This refers to the violation that the access control policy for the trigger data, defined by the user at the trigger service, is altered by the TAIP during the transmission of the trigger data to either the trigger-action service (at step 1 in Figure 1) or the action service (at step 2). This violation may lead to personal data being unintentionally shared on a service with public audience.

**Example.** Consider the automation task in our running example. After the user uploads a private photo on Facebook, the trigger-action service obtains a publicly-accessible URL of the photo, rather than a URL protected with authentication.

**V2: Lack of Access Revocation** (rel. P4 and P5). This refers to the violation that once the data reaches the trigger-action service or action service, the user loses all control over it. In consequence, it may be permanently stored by them unless they proactively delete it. Trigger-action services like IFTTT request the user to grant them the access to the trigger data, using multi-party authorization protocols like OAuth. We remark that security mechanisms such as OAuth token revocation [25] must be employed to enable the user to control over the trigger data.

**Example.** After the user deletes the photo on Facebook, the URL obtained by the trigger-action service or the action service remains accessible, and there is no any interface for the user to revoke it.

**V3: Unintended Data Flow** (rel. P1 and P2). This refers to the violation that the trigger-action service accepts and retrieves trigger data from the trigger service, out of the user’s intention and expectation. This may lead to personal data being unintentionally shared on another service.

**Example.** The applet is executed when the user uploads a video (instead of a photo) on Facebook, and the trigger-action service retrieves the video and shares it on Blogger.

**V4: Lack of Fine-grained Access Control** (rel. P1 and P6). This violation occurs when a trigger service or an action service provides fine-grained privacy configuration options, but the TAIP fails to retain them. The trigger-action service therefore does not provide the user with the same configuration options, upon the applet creation. This may lead to access control being altered or even downgraded when cross-party data transmission occurs.

**Example.** Facebook provides privacy configurations of “private” and “public” to its users. However, when creating the applet, the TAIP does not provide options for the user to restrict the trigger data sharing to only public photos. Hence, when the user uploads a private photo on Facebook, the TAIP shares it on Blogger.

**V5: Violation of the Least Privilege Principle** (rel. P2 and P3). This refers to the violation that trigger-action service acquires more trigger data from the trigger service than the minimum required to perform the actuation at the action service. In particular, upon the event configuration, the user specifies which trigger data (ingredients) should be transmitted to the action service. However, at the execution time, the trigger-action service may extract extra data. This may lead to the extra data being shared out.

**Example.** The action event is configured to receive the title of the Facebook photo post. However, during the execution, the trigger-action service acquires the entire photo in addition to the title.

4 APPROACH

In this section, we present our approach. Our discussion takes IFTTT-based TAIPs as an example, but it is applicable in other TAIPs based on other trigger-action services.

4.1 Challenges and Approach Overview

The core idea of Taifu is to construct and execute applets as the test cases to trigger possible data flows among involved entities in the TAIP. The generated data flows are then analyzed to identify privacy violations. We target to automate this process, so the following challenges have to be alleviated.

- **Challenge 1: Handling Login and Authorization GUIs**. To connect the online services with IFTTT, most of them require the user to log in with a registered account (step 1 in Section 2.2). This process is in general an OAuth procedure, in which the trigger-action service needs to be granted the permissions to access the trigger and fire the action. All these require much user interaction, and thus Taifu should be able to recognize GUI items and simulate responses to complete it. This is addressed in Section 4.2.

- **Challenge 2: Generating Valid Applets**. The event configuration (step 2 in Section 2.2) requires Taifu to provide
As is illustrated in Figure 2, Taifu is designed as a two-phase approach that consists of the generation of new applets and the execution of the generated applets. Below we briefly each component in these two phases and leave details to the remaining sections.

Phase 1: Applet Generation. This phase aims to generate applets. It includes two components: the service authenticator and authorizer and the applet generator. The former logs into the involved trigger and action services, and authorizes the applet to access the trigger and fire the action (detailed in Section 4.2). The latter extracts possible protocol steps involved in generating an applet, and then takes them as seeds to create new applets for each possible trigger-action pair (detailed in Section 4.3).

Phase 2: Applet Execution and Privacy Analysis. This phase executes the generated applets, and in the meantime, it identifies privacy violations based on a set of privacy rules. It includes three components: the applet executor, the behavior monitor and the privacy checker. The applet executor executes the applets by feeding them with trigger events. Unlike a traditional fuzzier which mutates all editable fields to generate as many test cases as possible, Taifu’s strategy is to mutate the privacy policy related fields such that the triggered data flows may reveal privacy issues. The behavior monitor extracts information (i.e., textual data and URLs) for detecting privacy issues. It involves the service authenticator and authorizer to obtain session cookies and access tokens, so that Taifu can access data in each service (detailed in Section 4.4). The privacy checker infers the policy enforced on the sensitive data from the execution traces, and then checks the privacy violations (V1-V5) using a set of pre-defined rules (detailed in Section 4.5).

As invalid values are given, the applet would be rejected straightaway by IFTTT, so Taifu should be “aware” of the protocol format and meaningful values for event configuration. A straightforward way might be to replay pre-recorded requests, but to create unknown applets (so as to trigger more data flows), Taifu needs to automatically recognize the editable fields and feed them with valid values. This is addressed in Section 4.3.

• Challenge 3: Monitoring Behaviors for Policy Decision. While an applet is being executed (step 3 in Section 2.2), Taifu needs to collect information (e.g., data flows) to decide privacy compliance. However, this is a non-trivial task, as the involved parties usually do not provide explicit protocol to fetch the status of the actions. This is addressed in Section 4.4.

4.2 Automated Login and Authorization

Taifu’s module of service authenticator and authorizer handles the login and authorization processes (see challenge 1), and connects the trigger and action services with the trigger-action service. Although nearly all services follow the OAuth procedure, due to the diversity of their implementations, the automation is a non-trivial task. As a first step, we have performed a manual analysis of randomly selected 30 online services compatible with IFTTT. We examined their protocols based on the captured network traffic, and identified three main types of OAuth procedures, i.e., auto-connect, simply-connect, and auth-required, as is illustrated by the examples in Figure 3. The auto-connect means no explicit actions are required to connect, simply-connect means a single button click is needed to connect, and auth-required means an authentication followed by an authorization is required to connect.

Among these three, our effort is mostly spent on the simply-connect and auth-required categories. Observing that web pages (HTML documents) often use domain-specific keywords in the attributes of HTML tags, we let Taifu exhaustively search for the keywords in all attributes to identify tags of interest. In particular, it uses an HTML parser to process each web page and search for all tags that potentially involve user inputs or user interactions, e.g., form, input, div, button and a. Its input element identifier then queries our manually crafted corpus to identify the type of each tag, i.e., username field, password field, navigation button, sign-in button, or authorization button, based on its attributes, e.g., id, class, name, type, or placeholder. The input and div tags are filtered to identify username or password fields; the div, a and
button tags are filtered to identify sign-in or authorization actions. Once the correct tags are identified, Taifu uses a web automation tool called Selenium to automatically perform authentication, such as filling in the test account information, and authorization, such as clicking the authorization button.

4.3 Applet Generation

Given a pair of trigger and action events, the module of applet generator creates an applet to incorporate them (see challenge 2). The applet creation process involves much user interaction for event configuration. It seems our approach of automating the login and authorization process could be reused. Nevertheless, creating an applet involves much more complex interfaces such as option fields and text fields. They may have interdependence with each other—for example, selecting one option may enable another, and their values are less predictable than HTML tags. We therefore use an alternative way of extracting the creation protocol from the communication between the trigger-action client and its server, and then utilizing it to create applets by concretizing the input fields.

4.3.1 Extracting Creation Protocols. We instrument the IFTTT Android app to capture the communication traces (in terms of HTTP requests and responses) between the app and the IFTTT server. This is done based on the Xposed framework on a Samsung Galaxy S4 device with custom Android Nougat. We instrument the Java classes that are used by the IFTTT app for network communication (e.g., the okhttp library and com.ifttt.ifttt.graph.Query). With the instrumentation, we manage to capture the traces for the randomly selected 30 services. We then perform a differential analysis [2, 33] on the traces to identify unique REST API calls. Each API call includes an HTTP request-response pair that we refer to as a protocol step in the rest of the paper.

In Table 2, we list the extracted protocol steps (responses are omitted). Each protocol step includes constant parts (e.g., the request URL and the field names in the request body) and variable parts (e.g., the input parameter and the field values). Column 3 lists the variable parts of HTTP request body of each protocol step in <> brackets. The rest of the HTTP request body is constants. Variable X in step 1 is the current trigger module name of the format <trigger service>..<trigger> (e.g., facebook.new_photo_with_hashtag.by_you), Y in step 2 is the current action module name (e.g., dropbox.create_a_post), TRIGGER FIELDS in step 6 is the current trigger configurations (e.g., {"hashtag": "ifttt"}), and ACTION FIELDS in step 6 is the current action configurations (e.g., {"title": "post title", "body": "post body"}). When Taifu creates an applet, the values for the variable parts of the protocol steps are dynamically decided based on the chosen pair of trigger and action events.

Table 2: The Protocol Steps

<table>
<thead>
<tr>
<th>Protocol Step</th>
<th>Request Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Get Trigger Configurations</td>
<td><code>{ &quot;query&quot;: &quot;mutation{applet_feedback_by_user:\applet_feedback_by_user\can_push_disable\published\archived\service_name\channels\underlying_applet}\queries, \actions:{\channel_id: &lt;ACTION CID&gt;, \step_identifier:&lt;Y&gt;, \fields: &lt;ACTION FIELDS&gt;}){&quot;normalized_applet\applet\errors }&quot;}</code></td>
</tr>
<tr>
<td>2 Get Action Options</td>
<td><code>{ &quot;query&quot;: &quot;query\{applet\{\fields: &lt;TRIGGER FIELDS&gt;\}\}&quot;}</code></td>
</tr>
<tr>
<td>3 Get Trigger Field Options</td>
<td><code>{ &quot;query&quot;: &quot;query\{applet\{\fields: &lt;TRIGGER FIELDS&gt;\}\}&quot;}</code></td>
</tr>
<tr>
<td>4 Get Action Field Options</td>
<td><code>{ &quot;query&quot;: &quot;query\{applet\{\fields: &lt;TRIGGER FIELDS&gt;\}\}&quot;}</code></td>
</tr>
<tr>
<td>5 Send Event Configurations</td>
<td><code>{ &quot;query&quot;: &quot;mutation\{applet\{\fields: &lt;TRIGGER FIELDS&gt;\}\}&quot;}</code></td>
</tr>
<tr>
<td>6 Send Event Configurations</td>
<td><code>{ &quot;query&quot;: &quot;mutation\{applet\{\fields: &lt;TRIGGER FIELDS&gt;\}\}&quot;}</code></td>
</tr>
<tr>
<td>7 Send Applet Creation</td>
<td><code>{ &quot;query&quot;: &quot;mutation\{applet\{\fields: &lt;TRIGGER FIELDS&gt;\}\}&quot;}</code></td>
</tr>
</tbody>
</table>

Table 3: The Event Configuration Field Types

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Field Type given by IFTTT</th>
</tr>
</thead>
<tbody>
<tr>
<td>text field</td>
<td>text_field, text, text_entry, location, location_radius, location.enter_and_exit, location.enter, location_exit</td>
</tr>
<tr>
<td>option field</td>
<td>collection_select, time_select, double_collection_select, checkbox, multi_minute_select, date_time, no_year_select</td>
</tr>
</tbody>
</table>

Table 4: Examples of Identified Clusters and Defined Values

<table>
<thead>
<tr>
<th>Clusters</th>
<th>Cleared Input Field Label</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>folder</td>
<td>enter_folder_path, folder_path, drive_folder_path</td>
<td>myfolder</td>
</tr>
<tr>
<td>duration</td>
<td>cook_time, cooking_time, cooking_time_exceeds</td>
<td>1, 2, 3, 4, 5</td>
</tr>
<tr>
<td>keyword</td>
<td>word_phrase, apply_label, tag_add</td>
<td>&quot;ifttt&quot;</td>
</tr>
<tr>
<td>desc</td>
<td>message, photo_description, item_description</td>
<td>&quot;event desc&quot;</td>
</tr>
<tr>
<td>humidity</td>
<td>humidity_time, humidity_forecast, humidity</td>
<td>20, 40, 60, 80</td>
</tr>
</tbody>
</table>

Some variables such as APPLET TITLE and TRIGGER CID in step 7 could be filled with a particular value in previous responses received from the server. In contrast, others such as TRIGGER FIELDS and ACTION FIELDS expect values that are erratic and specific to trigger/action events. Next, we discuss how we handle them.

4.3.2 Constructing Configuration Data Corpus. Since the event configurations are specific to the trigger/action events, we seek to build a large-scale configure data corpus to accommodate them. We first collect and analyze the event configurations from all events provided by all services that are supported by IFTTT. Our analysis finds
Table 5: Taifu’s Mutations

<table>
<thead>
<tr>
<th>Field</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageURL</td>
<td>VideoURL, FacetURL, PostURL, WebURL</td>
</tr>
<tr>
<td>Permissions</td>
<td>Private Keywords: Only me, Personal, Private</td>
</tr>
<tr>
<td>File types</td>
<td>photo, text, video</td>
</tr>
</tbody>
</table>

![Figure 6: The Generated Applet in our Running Example](image)

Figure 6: The Protocol Steps for our Running Example

that out of the 4,433 events provided by IFTTT, 3,899 include event configurations. They include thirteen types of input fields as shown in Table 3. Among all events, 69.4% have collection_select type, 27.4% have text_field type, and the remaining 3.2% include other eleven types. Since the option type and the text type are the vast majority, we group all types into two categories, i.e., text fields and option fields, as shown in Table 3. An option field (e.g., Figure 4a) has to be selected from a pre-defined list given by IFTTT, and a text field (e.g., Figure 4b) requires a string the user types in. An event configuration may require either or both types of fields (Figure 4).

Providing a valid input for an option field is straightforward since Taifu can select an option from the received option list. In contrast, providing a valid input to the text fields is challenging. Some text fields expect integers (specified by IFTTT through their documentation [26]) that could be fetched from trigger data (discussed in Section 4.3.3), while remaining ones require taking into consideration the "semantic" information as they may take diverse types of values, e.g., filename, temperature and hours. As an example, the trigger "temperature increased more than Celsius x" requires meaningful input values—an integer of 0-35 for a room temperature, and a string is likely to fail the input validation. To handle this, we resort to natural language processing techniques. We crawl all the input field labels (e.g., "Drive folder path", "Which humidity threshold?" and "What word or phrase?") available in IFTTT, and uses the existing tools and techniques including RxNLP [19], k-means++ [1] and word clustering to group similar labels. This gives us 35 clusters including name, description, folder, color, duration, temperature, brightness, etc. We then manually define the possible values for each cluster. As a demonstration, Table 4 lists part of our clusters.

Figure 5 shows the executed protocol steps when creating the applet in our running example. The protocol steps (3) and (4) are not used, since all the fields are text fields in both of the trigger and action configurations. The request body in step (6) includes the event configurations for the applet. The trigger has only one configuration field (hashtag). Taifu has assigned the value “#ifttt” to it, since its label is classified into the cluster of keyword (row 4 in Table 4). The action has two text fields, and Taifu assigns trigger ingredients (CaptionNotHashtag, ImageSource, Caption, From, Link and CreatedAt) to them. Figure 6 shows the generated applet for our running example.

4.3.3 Generating Applets. With the extracted protocol steps and configuration data corpus, Taifu could proceed with generating applets. For each pair of trigger and action events, Taifu first obtains their event configuration requirements by executing the protocols steps (1) and (2). If the trigger_fields and/or action_fields of an event are empty, it does not require configurations; otherwise, they specify the number of fields and field types for the event. Accordingly, Taifu fills in values for the fields based on their types (option fields or text fields). This involves the execution of the protocol steps (3) and (4) when the field type is an option. If the field is a text field, Taifu queries the value from the corpus. Further, the action_fields can be assigned with the trigger’s ingredients. To this end, Taifu uses the protocol step (5) to obtain the ingredients and update the action configurations. Finally, Taifu executes the protocol steps (6) and (7) to verify the event configurations and create the applet. Consequently, for all of the event pairs, Taifu generates and outputs a set of applets.

4.4 Applet Execution and Behavior Monitoring

Taifu’s module of applet executor executes the generated applets to trigger privacy-related behaviors of each involved party, and the behavior monitor records them for analysis.

4.4.1 Executing Generated Applets. To trigger the execution of an applet, the trigger events need to be initiated at the trigger client.
We categorize trigger events into three types based on the extent of user interaction, i.e., interaction events, IoT device events and public events. The interaction events are generated by user interaction with the trigger client (e.g., to upload a photo to social network, or to turn on a smart bulb). The IoT device events are generated by a physical IoT device without user’s control (e.g., the temperature reaching a threshold). The public events occur at external parties without involving the user (e.g., an article being published in a news website and the weather forecast being updated).

We exclude the public events out of Taifu’s testing, as they involve only publicly accessible data and are irrelevant to the privacy properties. Regarding the other two types, we attempt to generate the events in a simulative way so that our testing process remains automatic. To this end, we sniff the HTTP(S) requests sent by the trigger client to the trigger service using our instrumentation infrastructure built for the creation protocol extraction (see Section 4.3.1). We take these requests as the seeds and generate more requests via mutation. They are then sent to the trigger service to simulate the event occurrences. Our mutation is focused on the privacy configuration, among the meta data of the triggers such as imageURLs, fileURLs, and webURLs. Table 5 summarizes the values used by Taifu’s mutation.

To execute the applet in our running example, Taifu uploads a new photo with hashtag #IoT at Facebook. Then Taifu mutates its privacy configurations (“only me” and “shared with group”), and also its data type (e.g., to upload a video or text file instead of a photo) to probe the behavior change in each party.

4.4.2 Monitoring Behaviors. While an applet is executed, information related to the applet’s execution from the involved services, such as policy alterations at each service and the data flow related to the trigger event (as shown in Table 6), can be used to determine privacy violations. Collecting such information from an online service is a non-trivial task though (see challenge 3). To address this, we resort to two sources, i.e., the web clients of online services and the applet execution history from the IFTTT service.

First, we exploit the fact that the web clients always display the latest updates or evidence of the latest actions. Hence, we build a web scraper to crawl the web client of each involved service to extract the affected data, including altered or newly inserted HTML elements in the page resulted from the applet’s execution. To achieve this, it requests cookies and access tokens from the authentication (step 3 in Figure 2) to log into the web service, and then crawls the web page twice, i.e., before and after the applet is executed. Taifu compares the two pages using the htmlDiff library [12] to obtain the alterations. For scalability, the comparison is based on the HTML tag attributes rather than words. In this way, Taifu identifies the altered (mainly added or deleted) text and URLs. As an example, the rows 2 and 3 of Table 6 show the text data (at Column 4) and URL data (at Column 6) extracted from the web clients for the applet in Figure 6.

The second source is the execution history, which is accessible via REST API calls provided by IFTTT for its mobile client. After initiating a trigger event, Taifu keeps polling the APIs to find information regarding the applet. Figure 7 shows the execution history obtained by Taifu after the applet in our running example (see Figure 6) is executed. From the history, Taifu spots also the textual data and URLs. The information extracted from Figure 7 is listed in row 4 of Table 6.

4.5 Privacy Checking

Taifu’s privacy checker applies a set of pre-defined rules on the execution data to determine whether the applet execution violates user privacy as defined in Section 3.3.

4.5.1 Inferring Access Control Policies. The violations V1 and V2 are spotted based on checking the actual access permissions against the desired policies (discussed soon in Section 4.5.2). This thus requires inferring the access control policies, as no participant in real-world Taip’s publishes its policies. To fit into the traditional access control model [38], we treat the monitored text and URL data as the objects, and the data owner (i.e., user), the data controller (i.e., each party in the TAIP) and the external entities as the subjects. We introduce continuousness to the privacy policy as the access throughout the whole life cycle of an object, namely, creation and deletion, should be considered. We take into consideration two privacy levels, i.e., private and public. With these elements defined, the policies can be expressed in the access control policy matrix (ACM).

To find out the privacy level of the textual data, Taifu performs a keyword search on the HTML tag attributes and text contents, for indicators such as only me, private, everyone and public. When there is no keyword found, Taifu assigns the default access permissions which are manually crafted by examining the policy of each online service. To find out the privacy level of the URLs, Taifu visits them without logging in with any account. If the URL does not require authentication to view the content, Taifu infers its privacy as public and otherwise as private. For example, the Column 5 and Column 7 of Table 6 show the inferred privacy levels of the textual and URL data related to the execution of the applet of our running example (see Figure 6). Table 7 lists the policies inferred by Taifu regarding our running example.

4.5.2 Checking for Violations. Among the five types of violations, V4 is checked in the applet creation stage, by comparing the privacy configuration options obtained from the event configuration interface (e.g., the “keep private?” option in Figure 4.e) with those from the trigger service and the action service. V5 is checked by comparing the ingredients provided for the action configuration with the trigger data extracted by the trigger-action service as recorded in the execution log. For violations V1-V3 that involve the temporal aspect of applets - after creation and deletion, we borrow the idea of checking the temporal logic formulas. We use finite state Machines to specify the applet execution, so that pre-defined rules could be applied for the checking. The state machine in Figure 8 is defined.
Table 6: Data Extracted by the Behavior Monitor and their Privacy as Inferred by Taifu for the Applet in Figure 6

<table>
<thead>
<tr>
<th>Monitored Entity</th>
<th>Example</th>
<th>Default Pr. Level</th>
<th>Extracted Textual Data</th>
<th>Inferred Text Privacy</th>
<th>Extracted URL Data</th>
<th>Inferred URL Privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger Client</td>
<td>Facebook</td>
<td>public</td>
<td>#ifttt secret message</td>
<td>private</td>
<td><a href="https://www.facebook.com/photo/bid=1348296398849004">https://www.facebook.com/photo/bid=1348296398849004</a> &amp;set=a.17145658984510</td>
<td>private</td>
</tr>
<tr>
<td>Exec. History</td>
<td>Feed</td>
<td>private</td>
<td>View posted on Facebook</td>
<td>(&quot;#ifttt secret message/&quot; + view photo)</td>
<td>private</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Inferred Privacy Policies in Each Participant

The table shows the inferred privacy policies for the execution of the applet shown in Figure 6. The dashes mean the data become unavailable once deleted.

<table>
<thead>
<tr>
<th>Trigger Service</th>
<th>Action Service (Blogger)</th>
<th>Create (After the Object is Created at Trigger Service)</th>
<th>Delete (After the Object is Deleted at the Trigger Service)</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigger &amp; applet executed</td>
<td>simulator</td>
<td>any access operation</td>
<td>s', s' != ACM1</td>
</tr>
<tr>
<td>trigger &amp; applet executed</td>
<td>simulator</td>
<td>any access operation</td>
<td>s', s' != ACM2</td>
</tr>
</tbody>
</table>

Figures and Discussion

- Figure 8: The State Machine for Checking V1 and V2
- Figure 9: State Transition Diagram of the User Expected Applet Execution Behavior

For checking policy compliance related violations V1 and V2, and that in Figure 9 is for checking data flow related violation V3. Based on the state machines, Taifu attempts to identify privacy violations during testing, using the rules defined in Figure 8 and Figure 9.

Discussion. It is possible to detect V1 and V2 with a static approach, for example, by a differential analysis on the access control policies of each party. Nonetheless, the policies are mostly unavailable to the analysts, so they may have to manually examine the applet creation and execution process to understand the implicit policies. Taifu’s dynamic approach attempts to automate this process, so that the actual applet creation and execution could help identify precise information to reduce false positives.

5 EVALUATION

We implement Taifu and evaluate it on IFTTT-based TAIPs. It is implemented in Python, and Table 8 shows the implementation details. Taifu’s source code and our detailed experimental results can be accessed online [41]. In the remaining of this section, we present the evaluation on the performance of Taifu. We target to answer the following three research questions.

- **RQ1:** Is Taifu accurate in detecting the five types of privacy weakness?
- **RQ2:** Can Taifu effectively detect privacy weaknesses from real-world (IFTTT-based) TAIPs?
- **RQ3:** To what extent can the TAIP testing process be automated?

5.1 RQ1: Accuracy Evaluation

Before conducting a large-scale testing, we first study the accuracy of Taifu in its detection tasks. The main challenge is that there is a lack of available benchmark in the literature. We thus resort to manual effort to construct one as the ground truth. To be representative, we select the most popular four trigger services including Instagram, Facebook, Twitter, and Android Photos, and from each of them, we select one trigger that involves sensitive information. We then create applets to incorporate the four triggers with 36 popular action services, and manually check the violations. The detailed configurations of these applets and the expected results are given in our technical report [41]. In our manual investigation, we find all types of violations except V3. This implies V3 may be rare in the wild, and later our larger-scale experiments confirm this finding (to discuss in Section 5.2).

We apply Taifu to create and test the applets in our benchmark. It manages to automatically create 127 applets and execute 116 of them, as listed in Table 9. We leave the investigation of automation failure in RQ3, and focus on the detection performance here. Table 10-12 list Taifu’s detection results of each violation (except V3). For V1 and V2 (Table 10), the detection through the IFTTT
service achieves the highest accuracy (100% as both precision and recall). This could be attributed to the cleanliness of the extracted behavior data from the IFTTT execution history, which is in a well structured JSON format. For the detection through the action service, the recall rates are high, but the false positive rates are not negligible. The imprecision is mainly caused by the noise in the extracted behavior data from web clients, for example, the deviation on advertisements when Taifu refreshes the page. For both V4 (Table 11) and V5 (Table 12), Taifu achieves a high precision (100% in both) and recall (100% in V4 and 92.5% in V5).

Finding #1: The experiment on our benchmark shows that Taifu achieves a high detection rate for almost all types of violations, so it is promising to be applied to real-world TAIPs. Although a relatively high FP rate is acceptable for detection tool, its false positive rate (~0.2) is not negligible, and confirmation thus should be conducted when it is used in practice.

5.2 RQ2: Effectiveness of Applying Taifu on Real-world TAIPs

After the benchmarking, we apply Taifu to more services. As of October 2020, IFTTT supports 701 services of 46 categories (defined by the IFTTT applet store). Since the account creation (for authentication) requires non-trivial manual effort, we are not able to test on all services. We therefore select the most popular ones based on the applet installation count at the IFTTT web client. We crawl the applets with more than 1,000 installations, and then extract their trigger and action services. Through this, we obtain 105 services covering 39 of the 46 categories.

Taifu manages to connect all of the 105 services. Figure 10 shows the amounts of each authentication type (cf. Section 4.2). Figure 11 shows the distribution of time taken for connecting a service with IFTTT. On average, it takes 46.6 seconds to connect a service. A vast majority (88 out of 105 services) take less than 100 seconds, and those taking longer are due to additional authentication methods like CAPTCHA are applied such that analyst interaction is needed.

Among the 105 services, only 33 have a web client, which is essential for behavior monitoring (see Section 4.4.2). We add another 19 services that support only a mobile client to increase the number of services to test, and in the experiments, we manually collect their behavior data. We thus obtain 52 services. From them, Taifu successfully generates 407 applets within 25 minutes. Taifu manages to execute 283 out of the generated applets (failures to be discussed in RQ3). During the execution and behavior monitoring, authentication with the action services takes similar time to that in the connection, and the execution and behavior data capturing take only a few seconds. Table 13 lists the detection results. Among the 283 applets, 194 lead to V1 in 14 services, 90 V2 in 12 services, 15 V3 in 15 services, 218 V4 in 21 services, and 73 V5 in 19 services. Compared to other four types, V3 is rare in the wild. This confirms our finding on the benchmark (see Section 5.1). Our manual confirmation finds the 15 weaknesses of V3 are all true alerts.

Finding #2: Taifu is able to effectively detect violations from the real-world TAIPs within reasonable time periods. We also find that in the IFTTT-based TAIPs, V1 and V4 are common among services. V2 and V5 are less common, but access revocation and least privilege enforcement are essential when sharing data with third parties. V3 is relatively rare but still exists in some services in real-world TAIPs.

Responsible disclosure. We have reported our findings to the service providers including IFTTT and the third-party services. Some of them, such as Facebook and Dropbox, have acknowledged our reports. We also have kept our findings confidential for more than 90 days before we reported them in this paper.

5.3 RQ3: Automation Failures

During our experiments, whenever there is any automation failure, we resort to manual effort to explore the cause. In this section, we report and discuss them. Through this, we aim to spot the sites for future improvement, and also provide insights for future research on automated testing in TAIPs.

Authentication and Authorization. Taifu manages to identify and provide credentials for all the web clients. Nevertheless, it sometimes encounters run-time exceptions (e.g., MoveTargetOutOfBound-
### Table 10: The detection results of Taifu on V1 and V2

<table>
<thead>
<tr>
<th>Trigger Service - Trigger</th>
<th>Access at Trigger</th>
<th>Access of Action Service</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instagram - Any new photo by you (we upload private photo)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Facebook - New photo post by you with hashtag (we post private photo with text)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Twitter - New tweet by you (we post private tweet)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Android Photo - Any new photo by you (by default private)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

| Total | 0 | 4 | 0 | 0 | 4 | 0 | 4 | 0 | 4 | 0 | 4 | 0 | 4 | 0 |

**False Positive Rate = TP/(TP+FP)**

**Recall = TP/(TP+FN)**

**Precision = TP/(TP+FP)**

**F-measure = 2pr/(p+r)**

1 In our benchmark, all applets of a trigger service use the same trigger. Therefore, these applets are either positive or negative simultaneously.

### Table 11: The detection results of Taifu on V4

<table>
<thead>
<tr>
<th>Trigger Service of the Applets</th>
<th>I: Instagram</th>
<th>F: Facebook</th>
<th>T: Twitter</th>
<th>A: Android Photos</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Violated? Yes (all TPs)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Yes (all TNs)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>No (all TNs)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Total Applets</td>
<td>32</td>
<td>32</td>
<td>31</td>
<td>32</td>
<td>127</td>
</tr>
</tbody>
</table>

### Table 12: The detection results of Taifu on V5

<table>
<thead>
<tr>
<th>Trigger Service of Applet</th>
<th>V5 TP</th>
<th>FP</th>
<th>TN</th>
<th>FN</th>
<th>Executed Applets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instagram</td>
<td>16</td>
<td>0</td>
<td>13</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Facebook</td>
<td>17</td>
<td>0</td>
<td>12</td>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>Twitter</td>
<td>0</td>
<td>0</td>
<td>28</td>
<td>0</td>
<td>28</td>
</tr>
<tr>
<td>Android Photos</td>
<td>0</td>
<td>0</td>
<td>22</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>Total</td>
<td>37</td>
<td>0</td>
<td>76</td>
<td>3</td>
<td>116</td>
</tr>
</tbody>
</table>

**FP Rate = FP/(FP+TN)**

**Recall = TP/(TP+FN)**

**Precision = TP/(TP+FP)**

**F-measure = 2pr/(p+r)**

### Figure 12: Summary of Failures in Applet Execution and Behavior Monitoring for the 105 Services used in RQ2.

![Summary of Failures in Applet Execution and Behavior Monitoring](image)

login page or server, 26 associate third-party events as triggers, 30 provide only mobile clients, 17 require IoT devices that are currently unavailable, and 2 require premium accounts or involve payment for the trigger.

### 6. Threats to Taifu’s Validity

Taifu focuses on the privacy issues arising when cross-party data flows are pervasive in real-world TAIPs. To the best of our knowledge, this is the first work that uses the applets as test cases to explore the behaviors of the TAIP participants. However, current work of Taifu carries several limitations that could be addressed in future work.

First, although our most effort has been put into automating the testing process, there are still a few factors that may lead to automation failures, as we discuss in Section 5.3. Analyst interaction sometimes is required to remove the hurdles like CAPTCHA in the authentication, which are specifically deployed by the service providers to obstruct robots. Some trigger events may be uncontrollable by the analysts, such that automatic testing would also fail. Continuous improvement on our approach is hence needed to enhance the automation. Second, Taifu’s applet generation relies on NLP and clustering techniques to produce “valid” configurations, so that non-trivial test cases can be generated. As shown in Table 9, there is still a failure rate around 10%. This may be because our datasets and corpus are small-scale. A richer knowledge set could be applied to improve this component. Third, Taifu achieves a sException in Selenium), due to CAPTCHA, additional fields involved other than credentials (e.g., blog url), or multi-factor authentication (e.g., a one-time password sent to email or mobile). Taifu requires human interaction to overcome them. Overall, 18.1% of the services require manual effort, including 7.6% button clicks, 3.8% CAPTCHA, 3.8% multi-factor authentication, and 2.9% additional information to input (e.g., the serial number of an IoT device).

**Applet Generation.** Most failures in applet generation are caused by incorrect event configurations, especially in those that require text inputs. We find that they get incorrect values due to the misclassification in the clustering (i.e., imprecision in NLP) or the lack of relevant values in the corpus (see Section 4.3). For example, when clustering the field name “to” of the action Send an email, Taifu misclassifies it as a location and thus provides a physical address. Because of such errors, IFTTT drops the generation request. If such errors are not detected at generated time, they may lead to run-time errors.

**Applet Execution and Behavior Monitoring.** The reasons for failures in applet execution and behavior monitoring are shown in Figure 12. Among those failed services, 6 have errors in their web...
Table 13: Taifu’s Results on the 407 TAIPs

<table>
<thead>
<tr>
<th>Generated</th>
<th>Executed</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>V4</th>
<th>V5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word-press</td>
<td>31</td>
<td>28</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Android Photos</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Facebook</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Drop-box</td>
<td>31</td>
<td>28</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>One-Drive</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Fitbit1</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Withings</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Fitbit2</td>
<td>31</td>
<td>28</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Google Sheets</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Pintrest</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Four-square</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Instagram</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Twitter</td>
<td>32</td>
<td>29</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>31</td>
<td>28</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

Taifu is related to the security analysis and automatic testing of IoT platforms. In this section, we summarize existing studies in these two areas.

7.1 Security Analysis of IoT Platforms

Taifu is not the first on the security and privacy of trigger-action platform. Surbatovich et al. [40] have analyzed IFTTT applets by labelling the events with secrecy and integrity attributes tracking their propagation for conflicts. Bastys et al. [4] propose an information flow tracking framework to detect malicious applets. Schuster et al. [39] propose to control situational access when giving authorization to a web service to access information. AutoTap [45] and Bu et al [5] apply formal methods to detect violations in the scenarios that TAP programs are co-installed, which inspire Taifu’s approach of privacy checking (cf. Section 4.5). In comparison with these studies, Taifu’s focus is on detecting violations against the data access control policy in the cross-participant integrations.

Some studies also have been conducted to analyze security and privacy of the application platforms (e.g., Microsoft Azure IoT Suite and SmartThings) that connect smart devices and mobile apps for easy control through either local area network or cloud. Fernandes et al. [15] perform an empirical analysis of SmartThings, revealing platform design flaws. FlowFence [16] enforces user declared data flow patterns on sensitive data of smart apps. ContexIoT [29] provides control over sensitive actions by smart apps on IoT device. These solutions do not target to protect user privacy of in the TAIPs where online services are integrated with a trigger-action platform, and cross-participant data flows are involved. Several studies [23, 42] have discussed on the unauthorized data access in IoT application platforms. In Taifu, we target the privacy issues arising after the user has authorized service connections with trigger-action service. It demonstrates that due to the weaknesses in the cross-party integration, the user privacy may be violated.

Several other studies [6, 46] have proposed vulnerability identification approaches through traffic and static code analysis. Taifu’s approach of creation protocol extraction and applet generation (cf. Section 4.3) is inspired by them.

7.2 Testing of IoT Platforms

Celik et al. [7] propose a model checking tool for IoT apps (e.g., Smarthings smart apps) to verify safety and security properties. IotGuard [9] tests the safety and security of the actions in IoT environment at runtime. IoTFuzzer [10] discovers memory corruptions in IoT devices by fuzzing through control app. In addition, some studies [48, 49] propose to fuzz the online services to find server-side vulnerabilities using their mobile client apps. Taifu’s approach of creation protocol extraction and applet generation (cf. Section 4.3) is inspired by them. In a summary, there is lack of work on privacy issues arising when multiple parties are integrated through the TAIP. Further, the existing approaches are not directly applicable. Hence, we propose Taifu to address this gap.

8 CONCLUSION

Taifu aims to automatically identify privacy weaknesses from the TAIPs. It is based on the insight that the applets can be used as test cases to test the TAIPs. To this end, Taifu uses a two-phase approach to automate the testing process. It generates applets for 407 IFTTT-based TAIPs by synthesizing the trigger/action types, and manages to identify a large number of privacy violations. Taifu offers a new perspective that programs can be used as test cases to identify faults from their execution host. We remark that our work is a preliminary work in this direction, and more future studies are desirable to cope with the challenges we report. For example, the TAIP is a blackbox to the analysts, so how to efficiently trigger the functionalities and data flows needs further exploration.

ACKNOWLEDGMENTS

We thank our shepherd Tuba Yavuz and the anonymous reviewers for their insightful comments to improve this manuscript. This work is supported by the University of Queensland under the NSRSG grant 4018264-617225 and the GSP Seed Funding, the National Natural Science Foundation of China (No.61802168), the Leading-edge Technology Program of Jiangsu Natural Science Foundation (No. BK20202001), and the National Research Foundation (NRF) of Singapore under its NSOE DeST-SCI programme (No. NSOE_DeST-SCI2019-0006).