DDIFT: Decentralized Dynamic Information Flow Tracking for IoT Privacy and Security

Nikolaos Sapountzis
University of Florida
nsapountzis@ufl.edu

Ruimin Sun
University of Florida
gracesrm@ufl.edu

Daniela Oliveira
University of Florida
daniela@ufl.edu

***Abstract*—By 2018, it is no secret to the global networking community: Internet of Things (IoT) devices, usually controlled by IoT applications and applets, have dominated human lives. It has been shown that popular applet platforms (including If This Then That (IFTTT)) are susceptible to attacks that try to exfiltrate private photos, leak user location, etc. As new attacks might show up very frequently, tracking them fast and in an efficient and scalable manner is a daunting task due to the limited (e.g., memory, energy) resources at the IoT/mobile device and the large network size. Towards that direction, in this paper we propose a decentralized Dynamic Information Flow Tracking (DDIFT) framework that overcomes these challenges, better adapts to the IoT context, and further, is able to illuminate IoT applet attacks. In doing so, we leverage the synergy between: (i) a *dynamic information flow tracking* module that considers the application of tags with different types along with provenance information and runs in the mobile device at a fast timescale, (ii) a *forensics analysis* module running in the cloud at a slow timescale, (iii) *distributed optimization* to optimize various functionalities of the above modules as well as their interaction. We show that our framework is able to detect IoT applet attacks with higher accuracy (on average 81% improvement for different URL upload attack scenarios) and decreases resource wastage (on average 71% less memory usage under different integrity attack scenarios) compared to traditional DIFT, opening new horizons for IoT privacy and security.**

I. Introduction

By 2018 the extensive usage of Internet-of-Things (IoT) and their dominance in human life is a reality. Home applications supporting smart thermostats, smart locks, sleeping monitoring alternate the way we interact with our living spaces. In that context, mobile phones are usually the interface to control and manage such devices e.g., through *applications* or through *applets* that are reactive applications including some triggers (e.g., new picture in the camera roll), some actions (e.g., upload the new picture to google drive) and a piece of filter code to dictate which actions should run based on the trigger data coming in. Popular platforms to develop such applets include IFTTT, Zapier and Microsoft Flow.

Nevertheless, these platforms usually open up for various security and privacy concerns. For instance, consider the following attacker model: a malicious applet maker may develop an applet (or, alter the filter code of an applet) open to the public, that attempts to leak sensitive information (e.g., user location or even information regarding when children are home back from school) to a proxy server. In that context, some recent works propose *static information-flow tracking* frameworks to track such applets [1], [2]. There, some sensitive inputs, including e.g., user location or the device's model, are "marked" as sensitive and "flagged" if they are led to a sensitive sink, e.g., to Internet. Nevertheless, static analyses are not always efficient as they often miss dynamic code evaluation, dynamic typing, and dynamic object modification, nor they are always feasible as they require access to the source code. In that context, *dynamic information-flow tracking (DIFT)* methods emerge. In general, DIFT, also called Dynamic Taint Analysis (DTA), is used in a plethora of scenarios, to keep track of the information as it flows through a program's or system's execution: some inputs or data get tainted and then these taint marks (often called *tags*) propagate at the instruction or application level.

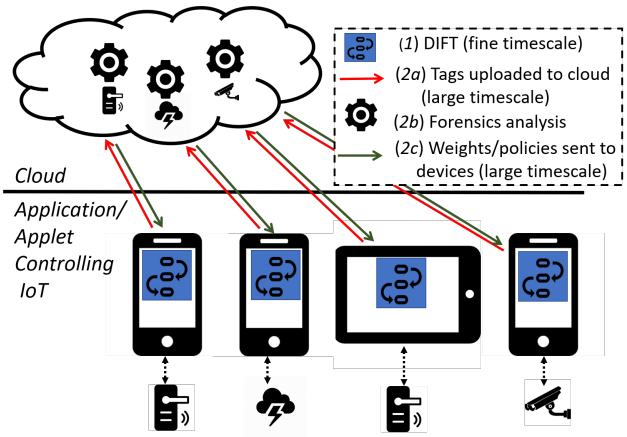

Fig. 1. DDIFT Architecture.

Due to promising flexibility that DIFT offers, it has been considered for various privacy and security frameworks; not only within the IoT applet context [3], [1], but also computer

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-1891562-49-5
http://dx.doi.org/10.14722/ndss.2018.23xxx
www.ndss-symposium.org

and cellular environments [4], [5], [6], [2]. Specifically, in [3], the proposed DIFT keeps a single float number, usually called *label*, per taintable object, namely per variable, to dictate how secure it is in that applet. However, keeping track of the information-flow through a single number per taintable object might not be sufficient e.g., when a reverse engineer wants to retrieve what other objects a malicious activity "touched", or the history of an object "touched" by a malicious process, or other provenance-related information. Additionally, *all* control and address dependencies, usually referred as *indirect flows*, are suggested to be propagated. However, it's clear from other DIFT works that this leads to *overtainting*, an undesirable phenomenon where all taintable objects become quickly tainted and little information can be gained about the information flow [7], [8], [9], [10]. Note that in the IoT context, the mobile devices, running the IoT applets, usually suffer from limited (e.g., memory, battery) resources, making things more challenging. Finally, the opportunity to perform forensics analysis and learn attack patterns e.g. by inspecting the same applet in different devices has not been considered yet as it is non-scalable to real systems.

To that end, in this paper we revisit all these challenges for IoT privacy and security. Our main contributions can be summarized as follows:

(1) *We propose a novel decentralized DIFT framework, shown in Fig. 1, that improves IoT security and privacy, and works in two steps, each running at a different element and timescale*. (Section IV)

(2) *(first step - fast timescale) We design a DIFT system that tracks all running applets at the mobile device level and adapts to the IoT context and constraints* (1a, Fig. 1). Specifically, we consider different tag types (e.g., network, social media, string tags etc.) each keeping provenance-related information (e.g., for network tags we keep a hashmap with the IP source/destination), and we introduce the *tag confluence* (i.e., when two tags come together) to illuminate the hallmarks of several IoT attacks. In doing so, we formulate a weighted objective function that attempts to (i) best allocate the limited (e.g., memory or battery) resources to tags with respect to some *weights* (dictated by the cloud, see below) and (ii) decide if the tags coming from an indirect flow are worth propagating, using distributed optimization (e.g., at the level of minute that applets are invoked). (Section V)

(3) *(second step - slow timescale) We propose a forensics analysis module at the cloud, that monitors the produced tags from all devices, and sends back to the devices information related to their local DIFT operation (2a-2c, Fig. 1).* For example, the cloud periodically derives and sends to the devices the resource allocation *weights* that prioritize the suspicious tags, suspicious tag types, or suspicious combinations (e.g., at the level of day, that the applet dynamics might change or that a new tag-signature threat found). (Section VI)

(4) Performance evaluation showed that *our framework detects IoT attacks with higher accuracy (on average* 81% *higher for different URL upload attacks) and decreases resource wastage (on average* 71% *less memory usage under different integrity attacks) compared to traditional DIFT.* (Section VII)

II. Background: DIFT and IoT Applets

A. DIFT Background

Dynamic Information Flow Tracking (DIFT), also known as Dynamic Taint Analysis (DTA), is a promising technology for making systems transparent. It works by tagging systems' data with tags and then propagating these tags as the system runs, so as something can be learned about the flow of information.

There are two main types of information flows: direct and indirect. In direct flows, a value is copied from one location (e.g., memory location or variable) to another. To track this information flow one propagates the tag so that the destination location is tainted with the same tag as the source location, e.g., for $x = y$; the variable x should have the same tag as y. Note that in computation dependencies, tags must be combined. For example, after the computation for $x = y + z$; the tag for x should contain the union of the tags for y and z.

An indirect flow occurs when information dependent on the program input determines from where and to where information flows. There are two indirect flows types: *address* and *control* dependencies. In address dependencies the value of the address of an array, memory etc. affects the program execution. For example, for $x = y[address]$; if $address$ is tainted, its tag should propagate to x. In control dependencies a condition affects the program execution. E.g., in $y = 1$; $if(x == 0)$ $\{y = 2;\}$ the value of y depends on the value of x, and thus y should be tainted with the tag of x. Contrary to direct flows, propagating all indirect flows leads to overtainting (all taintable objects get tainted) and no information can be inferred for the flow of information [8], [9].

B. IoT Applet Background

There are several popular IoT applet platforms, including IFTTT (If This Then That), Zapier, and Microsoft Flow. This section talks about the applet architecture on the IFTTT platform. An IFTTT **applet** is a small reactive app that includes *triggers* (e.g., if I take a new pic) and *actions* (e.g., upload that pic to Google drive) from different third-party partner services such as Instagram or Dropbox [3]. Applets may contain filter code for personalization. If present, the filter code is invoked after a trigger has been fired and before an action is dispatched. **Filter code** contains JavaScript code snippets with APIs pertaining to the services the applet uses. Filter code can use the APIs to configure the output actions of the applets.

C. DIFT in IoT Applets

JSFlow [11] is a JavaScript DIFT-based interpreter that monitors the execution flow of an application or applet. Specifically, during execution, JSFlow tracks various information sources (e.g. the triggers) and sinks (e.g. the actions), and forms the policies through labeling the sources and sinks.

III. Attacker Model

Most of the IoT *applets* are susceptible to different types of security and privacy attacks. Our main attacker model consists of a malicious application or applet maker, that attempts to extract personal information either when developing the applet or at a later stage through the *filter code*.

For example, consider the applet "Automatically back up your new photos from the Camera roll to Google Drive", and an attack that sends the new pictures to a malicious server before backing them up on Google Drive through the filter code (URL upload attack) [12]. This applet consists of trigger "Any new photo" in the camera roll, action "Upload file from URL" at Google Drive, and filter code for action customization. Through the filter code, IFTTT provides access to the trigger ingredients of the photos service and the action fields of the Google Drive service. In particular, there are APIs that e.g., provide the public URL of a photo being uploaded to the IFTTT server from the trigger "Any new photo". Similarly, there are APIs for the action field "Upload file from URL" that allow uploading any file from a public URL to Google Drive. The *URL upload attack* consists of filter code that inputs public URL of a picture as parameter to the attacker's server, and then uploads it to the attackers server with e.g., IP 192.168.172.50. Then, the attacker's server can be configured as a proxy to provide the user's photo in the response to Google Drive's, so that the image is backed up as expected by the user, while the attacker has leaked the information he needed.

IV. DDIFT Overview

In this section we describe the proposed architecture of our system, consisted of two thrusts, as seen in Fig. 1.

Thrust 1 (DIFT): Mobile device that controls IoT through an applet. The mobile device controls an IoT device(s) through an applet(s). To keep track of the information-flow of the applet(s) execution we propose a DIFT system. Some works have been proposed already in that context for IoT applets [3], e.g. through the JSFLow tool [11]. Nevertheless, these DIFT systems have several limitations: (i) they have simplistic rules for tag propagation that usually lead to resource wastage, (ii) they propagate all indirect flows, by leading to the undesirable phenomenon of over-tainting that threatens system entropy, as in most DIFT systems, (iii) their tags lack of provenance-related information by limiting the actions one can do upon an attack detection, and (iv) they do not have access to the DIFT systems running at other mobile devices (e.g., to learn tag-related signatures for new threats of different applets) for scalability reasons. Our DIFT system, addresses all that shortcomings by formulating new cognitive rules for resource allocation and indirect flow propagation, the introduction of tag differentiation and provenance list per taintable object. These functionalities operate using some *weights* (e.g., to best allocate resources to tags) that the cloud derives after inspecting different mobile devices (e.g., at the order of day or week). We elaborate more in Section V.

Thrust 2 (Forensics Analysis): Cloud. This thrust considers the cloud that enables tags produced in the device to be uploaded to the cloud for heavy-weight security and forensic analysis in a slow timescale. Then, by inspecting the same applet running at different devices, and collecting information about its operation, e.g., it derives the optimal values of a few *weights* for the problems above-mentioned. We elaborate more in Section VI.

To the best of our knowledge, this decentralized DIFT (DDIFT), is the first work that decentralizes such challenging DIFT problems that require centralized knowledge to the mobile devices.

V. Device: running DIFT

In this section we elaborate on the details and some additional assumptions of our proposed DIFT system (Section V-A). Then, we show our weighted objective function for resource allocation and indirect flow propagation as well as the optimal solution of the emerging problem (Section V-B).

A. System assumptions and overview

Tag differentiation. As discussed previously, the DIFT system will insert various types of tags. We introduce the following types of tags: (i) *location*, for the objects that relate to the user location, (ii) *string*, for string objects, (iii) *url*, for objects keeping urls, (iv) *social networks*, and (v) *fitness data* for objects keeping data coming from social network or fitness applications, (vi) *device info* for objects related to the information of the device (e.g., model, firmware version, etc.), (vii) *network* for all objects that related to packets coming from the network, (viii) *random generator* for all objects that call functions returning a random number.

Most of the tag types should keep some provenance information through a data structure. Specifically, *social networks* and *fitness data* tags, should be associated with a simple data structure that keeps the name of the application, *network* should keep the source/destination IP and port. *String, url* tags hold their actual value, and *device info* information regarding the model, firmware version, etc. *Random generator* does not need any provenance information.

Provenance list. We assume that our taintable objects are the variables; however our framework applies to finer setups (e.g. when the taintable objects are the bytes with increased overhead). For each variable we keep a *provenance list* of tags accumulated during the system execution. The provenance list, through the set of tags it stores, keeps all information flow history for the lifecycle of a variable in the system.

The use of tag differentiation along with provenance list, enables the feature of identifying "tag-signatures" for several attacks through tag confluences i.e., when more than a tag come together in the same provenance list. For example, in Fig. 2 we see the provenance list of the variable $attack$ composed of: a *url*, a *string*, and a *network* tag. As we shall see also later, this is a clear sign of a url upload attack. Additionally, our system allows to find the provenance information of the objects involved in an attack (e.g., IP address of the malicious server) and it also allows to find which other objects have been touched by the malicious process (e.g.,

which objects contain the tag url #8001). Finally, this allows the cloud to find additional similarities for certain attacks by looking at the provenance lists, e.g., if many devices report the string $https://attacker.com?$ as a part of an attack, the cloud can feed that back to the devices, so the latter know that this tag is suspicious even before it "confluences" with url and network.

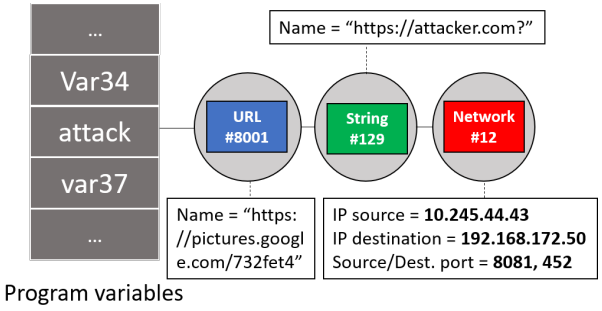

Fig. 2. Provenance list of the variable *attack* (related to url Upload Attack.)

Provenance list Size and Shadow Memory Due to the overhead, memory and battery limitations, the provenance list size should be kept small, denoted as M hereafter. We propose to let the value of M configurable as it directly dictates all the above factors (e.g., the larger the M the higher the latency for traversing the provenance lists and the more the battery consumption required) and could change based on the application scenario or any emergency e.g. due to a serious malware dictated by the cloud. For example, $M = 5$ means that each taintable object can keep up to 5 different tags in its provenance list. We also assume that the provenance list of tags for each variable will be stored in a shadow memory. This could be through a simple hashmap, as in [13].

B. Optimal Resource Allocation and Indirect Flow Propagation with Distributed Optimization

In this section, we consider two families of DIFT problems: (i) *resource allocation*, when e.g., a tag attempts to enter a list that is already full of tags (and the DIFT system has to decide which tags to schedule and which to drop), (ii) *indirect flow propagation*, i.e. when a control or address dependency attempts to propagate one ore more tags (and the DIFT has to decide whether it should schedule the newcoming tags or not). These are integer problems, and thus challenging to be solved. Alg. 1 optimally solves them through an analytical and generic framework leveraging distributed optimization, motivated by [14], [15].

Before proceeding, we want to introduce t that denotes the type of tag (e.g., t could be location, network, etc) and i an integer that differentiates the tags belonging to the same tag type (e.g., $i = 1, 2, 3$ etc.). Then, $n_{t,i}$ is the number of copies of the tag with ID $\{t, i\}$ on our system (namely, $n_{t,i}$ equals to the number of provenance lists that the tag $\{t, i\}$ exists). Also, λ_t is the weight of the tag type t and $\mu_{t,i}$ is the weight of the tag with ID $\{t, i\}$. Using distributed optimization, the

Algorithm 1 Optimal resource allocation and indirect flow decisioning.

Input. $\mathcal{C}$: the objective metric we attempt to optimize.
Output. $\Delta(n_{t,i})$: drop or schedule the tag.

1: Define the objective metric we attempt to optimize based on the per-tag metric value $c_{t,i}$, the tag type weights λ, and the tag weights μ.

$$\mathcal{C} = \sum_t \lambda_t \sum_i \mu_{t,i} \cdot c(n_{t,i}) \tag{1}$$

2: In order to decide about the potential propagation of a tag, the DIFT system should consider which decision offers the best gain for $\mathcal{C}$. To that end, we differentiate $\mathcal{C}$ with respect to $n_{t,i}$ (number of copies of the i-th tag belonging at the t-th type), we discretize and obtain:

$$\begin{aligned}\Delta(\mathcal{C}) &= \sum_t \lambda_t \sum_i \mu_{t,i} \frac{\partial c(n_{t,i})}{\partial n_{t,i}} \Delta(n_{t,i}) = \\ &= \sum_t \sum_i U_{t,i} \Delta(n_{t,i}),\end{aligned} \tag{2}$$

where:

$$U_{t,i} = \lambda_t \cdot \mu_{t,i} \cdot \frac{\partial c_{t,i}}{\partial n_{t,i}} \text{ is the utility of tag \{t,i\}} \tag{3}$$

$$\Delta(n_{t,i}) = \begin{cases} -1, \text{ if the tag \{t,i\} is dropped} \\ 0, \text{ if no action for the tag \{t,i\} is taken} \\ +1, \text{ if the tag \{t,i\} is scheduled} \end{cases} \tag{4}$$

3: For *resource allocation*: the DIFT should (i) schedule (i.e., keep or propagate) the tags in the order of decreasing $U_{t,i}$, and (ii) drop (i.e., delete or not propagate) the tags with the lowest $U_{t,i}$ (i.e., to ensure that the tags "carrying" more information are prioritized).

4: For *indirect flow* propagation decisioning: the DIFT should propagate a tag if this tag has $U_{t,i} > 0$ (i.e., the indirect flow propagation brings information to the DIFT).

values λ, μ (or, even the value of M) are dictated by the cloud in a slow timescale as we explain in Section VI. For example, the cloud might put a high value for $\lambda_{location}$ to boost the scheduling of location tags for a certain applet that is found to lead location, or boost the network tags coming from IP network 192.168.172.50.

Following Alg. 1, our **input** is $\mathcal{C}$, a generic objective function that can capture different privacy metrics and tradeoffs, and it can be found by summing e.g. some per-tag objectives over all tags, denoted as $c_{t,i}$ hereafter. For example, if we want to balance the copies of different tags and improve fairness, one can use the α-fair function from [16], [17]. Or, if we want to balance the tradeoff between over-tainting and under-tainting one might consider a penalty function for the former, and the α-fair function for the latter, weight them, and find Pareto optimal points, as we have done for other multi-criterion optimization problems in our previous work [17]. Our

output consists of the tags that will be dropped and the tags that will be scheduled when a scheduling decision has to be made in a provenance list in order to improve our considered privacy objective. To do so we introduce $\Delta(n_{t,i}) \in \{-1, 0, 1\}$: when $\Delta(n_{t,i}) = -1$ the tag with ID $\{t, i\}$ is dropped, when $\Delta(n_{t,i}) = 1$ it is scheduled, and when $\Delta(n_{t,i}) = 0$ no action is required. Specifically, after having defined our objective function (line 1, Alg .1), we differentiate it in order to obtain its gain over $n_{t,i}$ (line 2, Alg. 1). Specifically, the gain of our objective function is depicted in Eq. 4. Our aim is to maximize that gain at every scheduling decision, so we need to pick the values $\Delta(n_{t,i})$ for all tags in the corresponding list that optimize $\Delta(\mathcal{C})$. For *resource allocation* problems: the DIFT schedules the tags in the order of decreasing $U_{t,i}$, and (ii) drops the tags with the lowest $U_{t,i}$ to ensure that the tags "carrying" more information are prioritized. For *indirect flow* propagation decisioning the tag(s) are propagated if it has $U_{t,i} > 0$, to ensure e.g., that the propagation increases the overall gain.

Lemma 1. If $\mathcal{C}$ is concave (convex), Alg.1 corresponds to a distributed implementation of a gradient ascent (descent) algorithm that converges to a global optimal point [18].

VI. Cloud: running Forensics Analysis

The goal of this thurst, performing heavy forensics analysis is to to dictate the best values for the weighting parameters λ, μ, and develop additional policies that improve privacy and security, which will rely on a continuous analysis of a large volume of provenance tags produced by the inspected devices.

There are plenty of methods to update the parameters λ, μ that dictate the emphasis of a certain tag type (e.g., all *location* tags) or tag itself (e.g., the *string* tag with value *https://attacker.com?*), or their combination when they come together (e.g., the tag confluence *url, string, network*). For example, following Algorithm 1, one could use the subgradient of the objective at λ, μ to further improve the objective function. Then, the value of λ_t (similarly for $\mu_{t,i}$) would be:

$$\lambda = \lambda_{prev} + \zeta \cdot \sum_i \mu_{t,i} \cdot c(n_{t,i}), \tag{5}$$

where λ_{prev} is the previous value of λ, ζ is an appropriate step size that can be found e.g. with backtracking [18]. If $\mathcal{C}$ is concave (or, convex) on λ, μ our algorithm that also updates these values will eventually converge to the new global maximum (or, minimum) [18]. However, even if the function is not convex (convex), there are other methods to converge to a global optimum (e.g., for bi-convex functions see [17]).

Additionally, the cloud can build graphs representing the causal dependencies and interactions originated from the information flow in the system and represented by the tags. Specifically, it will map out global, long-term causality in systems and networks by constructing provenance graphs for system data and control flow. The whole-system snapshots of tags can be used to train a machine learning (ML) classifier to detect device specific atypical behaviors. Given the heterogeneity of IoT applets, *one-size-fits-all* ML-based detection solutions will likely generate a large number of false positives.

```
1. picURL = encodeURI('IosPhotos.newPicInCamRoll');
2. attack_string = ''www.attacker.com?'';
3. attack = do things//(e.g., connect to diff. ports);
4. attack = attack_string + picURL + attack;
5. GoogleDrive.ULFileFromUrlGoogleDr.setUrl.attack;
```

Fig. 3. *URL upload attack* through the applet "Automatically back up your new photos from the Camera roll to Google Drive" [12]. Line 3 makes the attack sophisticated: it attempts to congest the provenance list of the $attack$ variable, leaving no space for the important tags coming later.

In contrast, what is typical for that particular device will be leveraged to train the learning model, allowing the model to more accurately detect device/usage specific anomalies and how the device is being used (workload, user patterns, typical network traffic observed, etc.). Eventually, these will help on finding new treat tag-signatures and accordingly update λ, μ values for certain tag combinations.

VII. Evaluation

We study two attack examples under two real applets [12], [19] and evaluate the soundness of our proposed framework.

URL Upload Attacks (that also attempt to congest provenance lists). In Fig. 3 we illustrate an *URL upload attack* for an applet with title "Automatically back up your new photos from the Camera roll to Google Drive" [12]. As explained previously, the trigger of this applet is "Any new photo" and the action "Upload to Google Drive". The API encodeURI('IosPhotos.newPicInCamRoll') keeps the public URL of the user's photo on the IFTTT server (line 1). The attack consists of JavaScript filter code that passes the photo's public URL as parameter to the attacker's server (line 2-4). Specifically, the attacker creates a string $attack\_string$ that keeps the address of the attacker server (line 2). Then, the attacker uses the variable $attack$ to perform his sophisticated attack in two steps: (i) first in line 3 he attempts to do some dump activities in order to congest its provenance list (ii) in line 4 he creates the final string that corresponds to the url leading to the malicious server. Finally, in line 5 he concludes his attack by passing the intermediate URL to its own server.

We simulated 8 different scenarios of that attack in C++: by considering $M = 3, 4, 5, ..., 10$ and for each scenario we let the dump tags belonging to the same tag type of line 3, being uniformly distributed in $[0, M]$. We used the α-fair cost function as our objective in Alg. 1. We noticed that the accuracy of our algorithm on detecting the attack increases on average 81% compared to traditional DIFT [3], [13]. This happens since in traditional DIFT the tags that attempt to enter a congested list are immediately blocked (e.g., using a simple Last In First Out (LIFO) scheduling discipline [13]), thus the tags in line 4 will never be scheduled. Our method efficiently overcomes that problem as it follows. The scheduling decisioning is revisited within all (existing and newcoming) tags, everytime a list is congested: the gain of all tags will be measured, and using our analysis in Alg. 1 that gain will dictate which ones will be scheduled. Specifically, by considering the α-fair objective: (i) the gain of the tags of line

```
1. name = GoogleContacts.newContactAdded.Name
2. num = GoogleContacts.newContactAdded.PhoneNumber
3. chance = Math.random() + '' // returns [0,1]
4. if Integer.valueOf(chance) <= 0.5
5.    digit = Math.floor(Math.random()*10) + ''
6.    num = num.replace(num.charAt(digit),0)
7. GoogleSheets.appendToGoogleSpreadsheet(name + num)
```

Fig. 4. *Integrity attack altering phone numbers* through the applet "Back up Google Contacts to Google Drive Spreadsheet" [19]. Lines 3-6 make the attack sophisticated through indirect flows: propagating no indirect flow will miss the address dependency at line 6, while propagating all will over-taint several provenance lists due to the control dependency at line 4.

3 will be penalized, as they belong to the same tag type [17], and then (ii) the gain of *string, url* tags will be boosted in line 4 as they are part of a popular signature attack.

Integrity attacks through Indirect Flows. In Fig. 4 we illustrate an *Integrity attack altering phone numbers* through the applet "Back up Google Contacts to Google Drive Spreadsheet" which is used to back up the list of contact numbers into a Google Spreadsheet [19]. Specifically, the attacker using the variable $chance$ (line 3) flips a coin and with 50% chance (line 4) he selects a digit (line 5) of the phone number for the contact being backed up, and replaces it with 0 (line 6). This attack has two types of indirect flows: a control dependency at line 4 that attempts to propagate the tags of variable $chance$ to all variables encountered in the if-statement (should not be propagated as it does bring any useful information), and an address dependency at line 6 that dictates the address of the digit on the phone number (highlighting that the address of a digit is attempted to be modified).

Similarly to the URL upload attack, we simulated 8 scenarios of that integrity attack by using the α-fair cost function and assuming the cloud is monitoring other applets - three of them dealing with that attack. We found that on average our algorithm improves accuracy on detecting the attack 43% and it decreases the memory usage to 71% when compared to traditional algorithms. Traditional DIFT usually propagate no indirect flows (e.g., as in [13]); this would block the address dependency and it would not detect the attack. Other traditional DIFT works suggest to propagate all indirect flows (e.g., as in [3], [1]). In our case this would propagate also the meaningless control dependencies and congest the provenance lists, leaving no resources for the important tags coming later. Contrary to them, our algorithm considers one-by-one all tags coming from a potential indirect flow and *selectively* propagates them based on their gain. Using the α-fair cost function and the backtracking method for the λ, μ derivation at the cloud, the weights of the tags in the control dependency converge to 0 and thus they are not propagated. This decreases resource usage and leaves more space for the tags of the address dependency that illuminate the integrity attack.

VIII. Conclusion

In this paper we propose DDIFT, a decentralized framework for privacy and security within IoT applets. In doing so, we leverage the synergy between a dynamic information flow tracking module running in the mobile device at a fast timescale, and a forensics analysis module running in the cloud at a slow timescale. We show how our framework tackles the problems emerging in the IoT context e.g., how the system should allocate the limited (e.g., memory or energy) resources to improve privacy and security. Experiments showed that our framework improves accuracy (on average 81% improvement for different URL upload attacks) and decreases resource wastage (on average 71% less memory usage under different integrity attacks) compared to traditional DIFT.

References

[1] Z. B. Celik, L. Babun *et al.*, "Sensitive information tracking in commodity iot," in *USENIX Security Symposium*, 2018.

[2] C. Nandi and M. D. Ernst, "Automatic trigger generation for rule-based smart homes," in *ACM Workshop on Programming Languages and Analysis for Security*, 2016.

[3] I. Bastys, M. Balliu, and A. Sabelfeld, "If this then what?: Controlling flows in iot apps," in *ACM Conference on Computer and Communications Security*, 2018.

[4] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, 2014.

[5] B. Gu *et al.*, "D2taint: Differentiated and dynamic information flow tracking on smartphones for numerous data sources," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2013.

[6] A. M. Espinoza *et al.*, "V-difft: Vector-based dynamic information flow tracking with application to locating cryptographic keys for reverse engineering," in *IEEE International Conf. on Availability, Reliability and Security (ARES)*, 2016.

[7] M. Costa, J. Crowcroft *et al.*, "Vigilante: End-to-end containment of internet worms," in *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, 2005.

[8] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure program execution via dynamic information flow tracking," in *ACM Sigplan Notices*, 2004.

[9] J. R. Crandall and F. T. Chong, "Minos: Control data attack prevention orthogonal to memory model," in *Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture*, 2004.

[10] A. Slowinska and H. Bos, "Pointless tainting?: evaluating the practicality of pointer tainting," in *Proceedings of the 4th ACM European conference on Computer systems*, 2009.

[11] D. Hedin, A. Birgisson, L. Bello, and A. Sabelfeld, "Jsflow: Tracking information flow in javascript and its apis," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014.

[12] alexander via IFTTT, "Automatically back up your new ios photos to google drive," https://ifttt.com/applets/90254p-automatically-back-up-yournew-ios-photos-to-google-drive., 2018.

[13] M. N. Arefi, G. Alexander *et al.*, "Faros: Illuminating in-memory injection attacks via provenance-based whole-system dynamic information flow tracking," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2018.

[14] F. Bannour, S. Souihi, and A. Mellouk, "Distributed sdn control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tutorials*, 2018.

[15] P. Matzakos, T. Spyropoulos, and C. Bonnet, "Joint scheduling and buffer management policies for dtn applications of different traffic classes," *IEEE Transactions on Mobile Computing*, 2018.

[16] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, 2000.

[17] N. Sapountzis, T. Spyropoulos *et al.*, "Joint optimization of user association and dynamic tdd for ultra-dene networks," *IEEE International Conference on Computer Communications (INFOCOM)*, 2018.

[18] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[19] jayreddin via IFTTT, "Google contacts saved to google drive spreadsheet," https://ifttt.com/applets/nyRJVwYa-google-contacts-saved-to-googledrive-spreadsheet., 2018.