Poster: Data Minimization by Construction for Trigger-Action Applications

Downloaded from: https://research.chalmers.se, 2026-10-12 15:00 UTC

Citation for the original published paper (version of record):

Ahmadpanah, S., Hedin, D., Sabelfeld, A. (2023). Poster: Data Minimization by Construction for Trigger-Action Applications. CCS 2023 - Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security: 3522-3524. http://dx.doi.org/10.1145/3576915.3624376

N.B. When citing this work, cite the original published paper.

research.chalmers.se offers the possibility of retrieving research publications produced at Chalmers University of Technology. It covers all kind of research output: articles, dissertations, conference papers, reports etc. since 2004. research.chalmers.se is administrated and maintained by Chalmers Library

(article starts on next page)

Poster: Data Minimization by Construction for Trigger-Action Applications*

Mohammad M. Ahmadpanah
Chalmers University of Technology

Daniel Hedin
Chalmers University of Technology
Mälardalen University

Andrei Sabelfeld
Chalmers University of Technology

ABSTRACT

Trigger-Action Platforms (TAPs) enable applications to integrate various devices and services otherwise unconnected. Recent features of TAPs introduce additional sources of data such as *queries* in IFTTT. The current TAPs, like IFTTT, demand that trigger and query services transmit excessive amounts of user data to the TAP. To limit the data to what is actually necessary for the execution to comply with the principle of *data minimization*, input services should send no more than the necessary data. LazyTAP proposes a new paradigm of *data minimization by construction* in TAPs, introducing a novel perspective for data collection from input services. While the existing *push-all* approach of TAPs entails coarse-grained data over-approximation, LazyTAP *pulls* input data on-demand at the level of attributes, once accessed by the app execution. Thanks to the fine granularity provided by LazyTAP, multiple trigger and query services can be naturally minimized while the behavior of app executions is preserved. In addition, a great benefit of LazyTAP is being seamless for third-party app developers. By leveraging laziness, LazyTAP defers computation and proxies objects to load necessary remote data behind the scenes. Our evaluation study on app benchmarks shows that on average LazyTAP improves minimization by 95% over IFTTT and by 38% over minTAP, with a tolerable performance overhead. This poster goes into further details about LazyTAP and elaborates on its prototype implementation.

CCS CONCEPTS

• **Security and privacy → Web application security**.

KEYWORDS

Data Minimization, Trigger-Action Platforms, Lazy Computation

ACM Reference Format:
Mohammad M. Ahmadpanah, Daniel Hedin, and Andrei Sabelfeld. 2023. Poster: Data Minimization by Construction for Trigger-Action Applications. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), November 26–30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3576915.3624376

*This poster presentation is mostly based on our S&P'23 paper [2].

Permission to make digital or hard copies of part or all 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 third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CCS '23, November 26–30, 2023, Copenhagen, Denmark
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0050-7/23/11.
https://doi.org/10.1145/3576915.3624376

1 INTRODUCTION

Trigger-Action Platforms (TAPs) like IFTTT [7], Zapier [14], and Microsoft Power Automate [13] enable connecting otherwise unconnected devices and services. Services that manage users' data are among the ingredients of popular apps like "Every morning at 7am, send a Slack message with the first meeting of the day from Google Calendar" [11] or "When you turn your Samsung TV on after 5pm on Saturdays, randomly pick one of the personalized movie recommendations from Trakt" [12]. In these examples, the TAP app is fired by events from *trigger* services (Time and Samsung TV). Thereafter, the apps request additional inputs from *query* services (Google Calendar and Trakt), and send the outputs to *action* services (Slack and Notification).

Current TAPs like IFTTT employ a *push-all* approach for input data. When a new event is emitted by the trigger service, all input data attributes are pushed to the TAP, no matter if the data is going to be used in the app execution or not. This coarse-grained over-approximation is at odds with *data minimization*, a principle stipulating to limit the data to "what is necessary in relation to the purposes for which they are processed" [6]. This important principle is adopted by legal frameworks like the General Data Protection Regulation (GDPR) [6] and the California Privacy Rights Act (CPRA) [5]. The push-all approach exacerbates the privacy problem in the presence of multiple sources of input data. IFTTT allows multiple inputs via *queries* [8, 10], a recently introduced feature to develop apps with additional data sources.

In the meeting notification app (see Figure 1), the trigger service is Time, triggering the app at 7am every day. However, the sensitive input of the app is loaded by a query to Google Calendar. Even though the app needs information about only one meeting, the TAP unnecessarily fetches all attributes of all recent meetings. Moreover, even if the app only asks for a query conditionally on some input, IFTTT will always load the most recent 50 query events [9] regardless of the input and whether the data is necessary for the execution.

Similarly, in the movie recommender app (see Figure 2), all recommended movies will be sent from Trakt to the TAP, letting the TAP make a randomized pick, even though only one movie is recommended to the user. This example also illustrates the use of *nondeterminism* in apps.

Table 1 compares IFTTT's push-all approach with minTAP [4] and LazyTAP [2], two data minimization approaches for TAPs. minTAP assumes an ill-intended TAP trying to break data minimization by manipulating the apps to extend access to sensitive data and by gaining access attributes beyond necessary for the apps. The solution behind minTAP is preprocessing [3] data on the trigger service, to filter out any redundant data sent to the TAP. This solution needs a trusted client outside of the TAP that performs

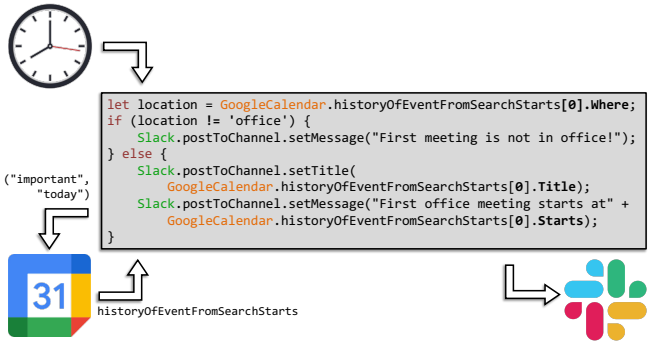

Figure 1: A meeting notification app [2].

attribute dependency analysis of the apps and ensures the integrity of installed apps.

The minimization approach of static minTAP is input-unaware in the sense that runtime values are not involved. Static minTAP could be extended to analyze query attributes in the app code as well. In a hypothetical extension of static minTAP, all the input attributes existing in the app code are identified as needed. For the meeting notification app, sensitive meeting details are disclosed regardless of whether they satisfy input conditions (e.g., the meeting location is the office). Similarly, in the movie recommender app, the static approach does not access the random index value at run time and reveals the entire list of recommended movies instead of only the necessary item.

To identify the necessary attributes, dynamic minTAP conducts a pre-run of the app code within a sandbox environment on the trigger service. Unlike static minTAP, this approach is input-sensitive, resulting in more precise minimization. Additionally, the trigger service can even choose to skip events entirely if the corresponding actions will be skipped in the execution.

The pre-run approach of dynamic minTAP imposes fundamental limitations to support for queries and nondeterminism. The trigger service does not (and should not) have access to query data, meaning that the preprocessor does not have the required information to execute the app. Moreover, the pre-run cannot guarantee the exact same behavior in runtime when nondeterminism exists in the app.

2 LAZYTAP

LazyTAP [2] proposes a new paradigm of *data minimization by construction* in TAPs, introducing a novel perspective for data collection from input services. While the existing *push-all* approach of TAPs entails coarse-grained data over-approximation, LazyTAP

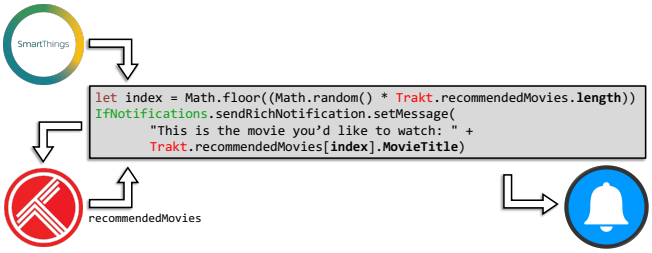

Figure 2: A movie recommender app [2].

pulls input data on-demand at the level of attributes, once accessed by the app execution. Thanks to the fine granularity, LazyTAP is able to send only the required attributes of at most one calendar event in the meeting notification app, and only at most one movie recommendation in the recommender app.

LazyTAP assumes that the TAP is willing to support the principle of data minimization for the sake of its users and in the light of the legal frameworks. The TAP being involved in data minimization opens up the opportunity of pulling data on-demand. Assuming that the TAP is untrustworthy and might want to collect more data than required [4] necessitates preprocessing [3] of the data before it is sent to the TAP. A willing-to-minimize TAP, on the other hand, relaxes the trust assumption, enabling the TAP and input services to cooperate with each other to attain a higher level of data minimization.

In contrast to minTAP, LazyTAP does not require the trigger service to run the app and no trusted client is involved. LazyTAP achieves input-sensitive minimization by fetching input data on the fly for a given run (see Table 1). Further, LazyTAP supports queries and is robust with respect to nondeterminism while preserving the behavior of the app in question. This implies sending only the meeting attributes used in the notification in the first app and precisely the randomly picked recommended movie in the second app. Notably, LazyTAP's commitment to enhancing data minimization makes it an appealing option for future adoption by TAPs, particularly when compared to solutions addressing untrustworthy TAPs.

LazyTAP keeps the runtime seamless for app developers. Under the current TAPs, trigger and query data is received and stored in an object tree which is processed by app code. In contrast, LazyTAP creates so-called *remote objects* for trigger and query data that look to app code like local objects but are in fact populated by network requests to trigger and query services at runtime, and only as much as needed by the given app execution. The novel

TAP	Minimization wrt	Minimization guarantees
IFTTT	None	Push all, no minimization guarantees
Static minTAP	Ill-intended TAP	Input-unaware minimization
Dynamic minTAP		Input-sensitive minimization wrt trigger input, No attributes when skip/timeout, No support for queries or nondeterminism
LazyTAP	TAP willing to minimize	Input-sensitive minimization wrt trigger and query inputs

Table 1: Comparison of TAPs [2].

architecture of LazyTAP brings on-demand computation in third-party JavaScript apps. Deferred computation and proxy objects empower the runtime to load necessary remote data behind the scenes once accessed in the execution.

Migrating to a fetch-on-demand paradigm on the TAP implies changes to the trigger and query services for compatibility reasons. These changes are straightforward and can be shimmed on the services themselves, or using third parties. Shimming on the services does not change the trust assumption on the services, already holding user data, and is the natural choice as services must implement a compatibility layer [9] with IFTTT regardless.

LazyTAP's prototype is implemented in a setting based on IFTTT, showing that on average minimization is improved by 95% over IFTTT and by 38% over minTAP on app benchmarks, while incurring a tolerable performance overhead. The source code and benchmarks are available [1].

3 CONTRIBUTIONS

In this poster, we go into further details about LazyTAP [2] and explain the paradigm for data minimization by construction in TAPs. We elaborate on the prototype implementation of LazyTAP that leverages laziness behind the scenes in the form of deferred computations and proxy objects accessing data attributes on a by-need basis, seamless for app developers.

Acknowledgments. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the Swedish Foundation for Strategic Research (SSF), and the Swedish Research Council (VR).

REFERENCES

[1] Mohammad M. Ahmadpanah, Daniel Hedin, and Andrei Sabelfeld. 2023. LazyTAP implementation and benchmarks. https://www.cse.chalmers.se/research/group/security/lazytap/.
[2] Mohammad M. Ahmadpanah, Daniel Hedin, and Andrei Sabelfeld. 2023. LazyTAP: On-Demand Data Minimization for Trigger-Action Applications. In *S&P*. IEEE, 3079–3097.
[3] Thibaud Antignac, David Sands, and Gerardo Schneider. 2017. Data Minimisation: A Language-Based Approach. In *SEC*.
[4] Yunang Chen, Mohannad Alhanahnah, Andrei Sabelfeld, Rahul Chatterjee, and Earlence Fernandes. 2022. Practical Data Access Minimization in Trigger-Action Platforms. In *USENIX Security*.
[5] CPRA 2020. California Privacy Rights Act (CPRA). https://oag.ca.gov/privacy/.
[6] GDPR 2018. General Data Protection Regulation (GDPR). Art. 5 Principles relating to processing of personal data. https://gdpr-info.eu/art-5-gdpr/.
[7] IFTTT 2023. If This Then That. https://ifttt.com.
[8] IFTTT. 2023. IFTTT's Glossary: Query. https://platform.ifttt.com/docs/glossary.
[9] IFTTT. 2023. Service API requirements. https://platform.ifttt.com/docs/api_reference.
[10] IFTTT. 2023. The art of the query. https://ifttt.com/developer_blog/the-art-of-the-query.
[11] IFTTT app 2023. Get a morning reminder about your first meeting daily. https://ifttt.com/connections/WHQ7AjWP.
[12] IFTTT app 2023. Saturday movie night recommendations with Samsung SmartThings and Trackt. https://ifttt.com/applets/jUy5if7H.
[13] Microsoft Power Automate 2023. https://powerautomate.microsoft.com.
[14] Zapier 2023. https://zapier.com.