Poster: Toward Secure & Serverless Trigger-Action Platforms

Pubali Datta[1], Tristan Morris[2], Hayawardh Vijayakumar[2], Michael Grace[2], Adam Bates[1], Amir Rahmati[2,3]
[1]*University of Illinois, Urbana-Champaign*, [2]*Samsung Research America*, [3]*Stony Brook University*
{pdatta2,batesa}@illinois.edu, {t.morris,h.vijayakuma,m1.grace,amir.rahmati}@samsung.com, amir@cs.stonybrook.edu

The automation rules in popular trigger-action platforms (TAP) (e.g.,IFTTT [1]) are usually simple event-driven workflows, in form of *"if [TRIGGER], then [ACTION]"*. Users compose these rules simply by stitching APIs exposed by various service providers connected to the trigger-action platform. While TAPs enable users to easily integrate various services together and cover simple cases that a regular user may grapple with, they suffer from limitations that make them inadequate for complex or security-sensitive settings: First, users of a TAP neither have control over, nor knowledge of, how their data are accessed or used by the workflows. This is especially problematic as previous work has shown these systems to be overprivileged [2]. Users must trust the workflow to execute according to its advertised behavior and not abuse the overprivilege. Second, existing platforms offer limited ability to customize or create complex workflows. Users are limited to predefined, single-stage workflows that they cannot extend in arbitrary ways. These limitations make current platforms unsuitable to deploy automation workflows in security sensitive environments. In this work, we propose the design of *Dromos*, a platform to overcome these shortcomings.

Design Goals. Dromos is designed to enable the following functionalities. (1) *Customizability:* Dromos should provide customizability by enabling creation of custom blackbox functions and should support complex workflows. (2) *Control over Data:* Dromos should provide user with fine-grained control over permissions for their workflows. (3) *Security:* Dromos should provide security guarantees on how user data is being used on the platform. (4) *Usability:* Dromos should maintain current TAP support for rapid and easy development of trigger-action workflows and easy integration to third party services to retain the ease-of-use. (5) *Scalability:* The trigger-actions workflows developed should scale to thousands of users simultaneously.

Design Sketch. Dromos consists of the following components designed to meet the described goals.

Function repository: A database of triggers, actions, and small functions which states their input, output and expected behavior (handling of input-data) which can be used in creating trigger-action workflows. Much like the service-developers in TAP, in Dromos function developers can create functions (sometimes opaque to the platform) performing specific tasks that can be stored in the function repository and published to be used in any workflow by workflow creators.

Workflow creation: Dromos supports creation of complex workflows, where workflow creators can use existing functions from repository or custom functions and stitch them into a workflow using the workflow creation mechanism. Workflow creation is followed by static high level flow tracking on a composed workflow and associating metadata of component functions stored in the repository to compute fine-grained permissions required to execute the workflow. Users are presented with these permissions at installation time of a workflow to make an informed decision on using it. The function repository and the workflow creation mechanism together achieves customizable and rapid development of workflows.

Execution Engine: The execution engine deploys the entire trigger-action workflow in a scalable way, enforces permission usage in the workflow, and provides secure isolation between workflows. It uses serverless execution framework in the background for execution. The serverless (Function-as-a-Service) frameworks (AWS Lambda, OpenFaaS, OpenWhisk etc) have recently gained popularity to execute event-driven tasks on demand. These platforms enable a function to be triggered by defined events or http requests and spin up a container with the appropriate runtime environment to execute the function. This contained execution of functions helps achieve isolation to provide security guarantees. Serverless frameworks automatically spawn multiple function instances to scale to multiple events or requests. This feature makes such frameworks perfect fit for Dromos to achieve scalability.

Implementation Challenges. We envision several challenges in implementing Dromos: (1) Generating readable fine-grained permissions required by a workflow from component function metadata and source code (when available). (2) Even with comprehensive list of permissions shown to the end-users, they might not accept all permissions (required for workflow execution) presented and Dromos may have to support the workflow execution with limited capabilities. (3) While serverless frameworks offer auto-scaling and contained execution, there are existing security concerns with them[3], [4]. For performance enhancement, many of these frameworks keep a pool of warm containers ready to serve repeated and parallel invocations of a function. This may cause cached data from one invocation to be leaked to future invocation of the same function. Also, functions can spawn arbitrary child processes inside containers leading to malicious behavior. Dromos solves such issues by adding security extensions to serverless frameworks. Dromos will implement a security shim around the function which dynamically profile the invoked function for suspicious behavior and provide proper sanitization and data-isolation between invocations. (4) Dromos provides additional security guarantees by allowing the workflow creators to define high level information flow policies to govern the communication between functions and by enforcing the policies dynamically during workflow execution, thus providing better control over data-usage.

We present the initial design of Dromos in this poster, that overcomes limitations in existing TAPs and brings in strong security guarantees about usage of data in the platform.

References

[1] "IFTTT," https://ifttt.com, 2017.

[2] E. Fernandes, A. Rahmati, J. Jung, and A. Prakash, "Decentralized action integrity for trigger-action iot platforms," in *NDSS*, 2018.

[3] A. Krug and G. Jones, "Hacking serverless runtimes," Blackhat 2017.

[4] "Gone in 60 Milliseconds," https://media.ccc.de/v/33c3-7865-gone_in_60_milliseconds, 2016.