SandTrap: Securing JavaScript-driven Trigger-Action Platforms

Andrei Sabelfeld @asabelfeld

CHALMERS

Joint work with M. Ahmadpanah, D. Hedin, M. Balliu, and E. Olsson

Trigger-Action Platforms (TAPs)

- “Managing users’ digital lives” by connecting
  - Smart homes, smartphones, cars, fitness armbands
  - Online services (Google, Dropbox,…)
  - Social networks (Facebook, Twitter,…)
- End-user programming
  - Users can create and publish apps
  - Most apps by third parties
- JavaScript-driven
  - IFTTT and Zapier (proprietary)
  - Node-RED (open-source)

IFTTT

Do more with the services you love

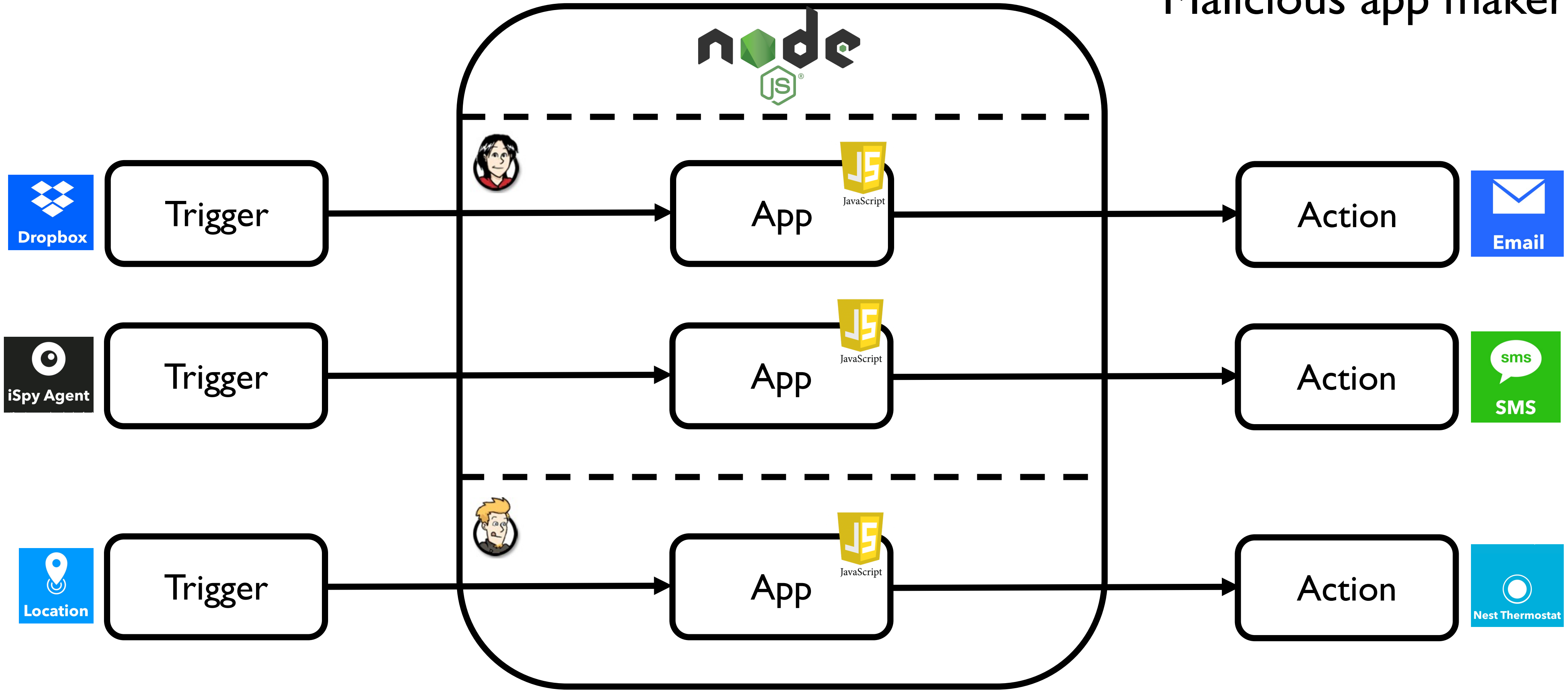
zapier

Node-RED

TAP architecture

Threat model:

Malicious app maker



- Zapier and Node-RED: single-tenant
- IFTTT: multi-tenant

Sandboxing apps in IFTTT and Zapier

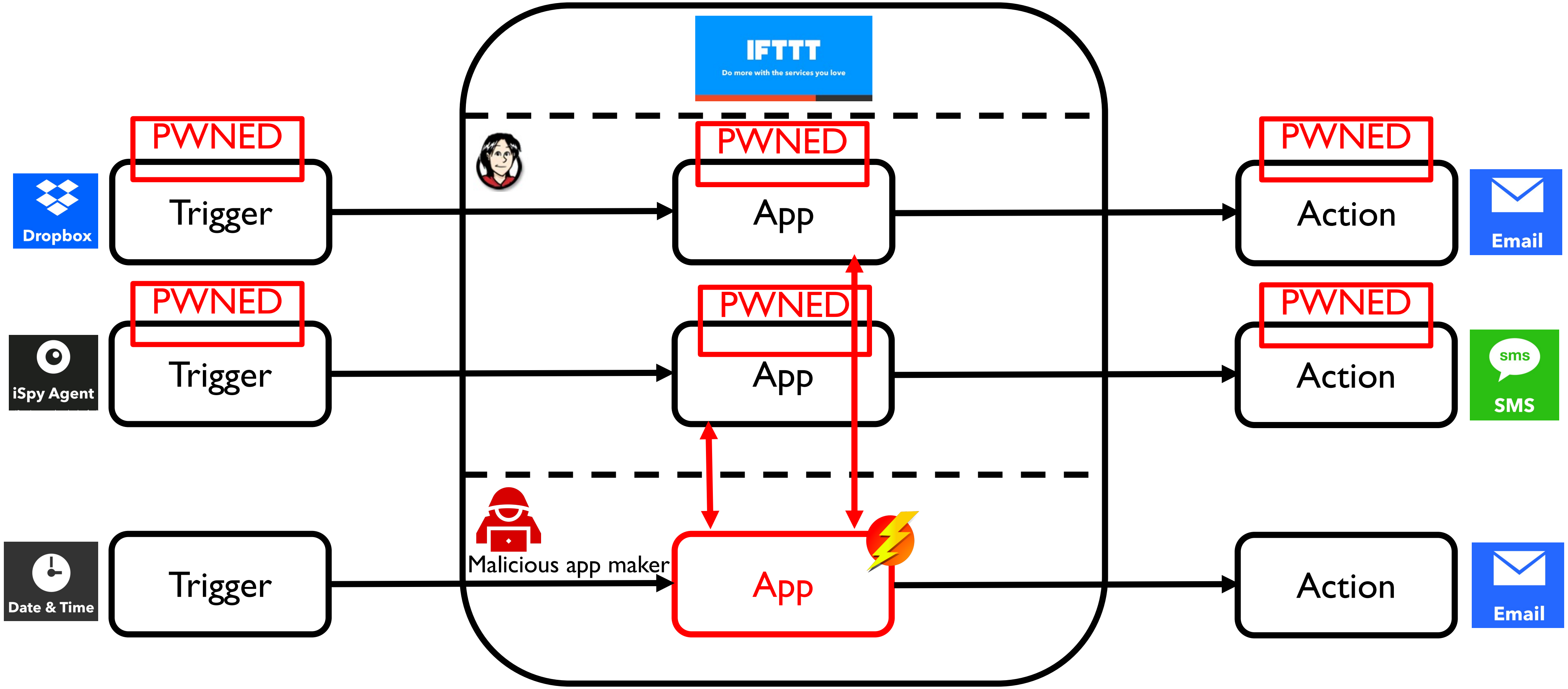
- JavaScript of the app runs inside AWS Lambda
- Node.js instances run in Amazon's version of Linux
- AWS Lambda's built-in sandbox at process level
- IFTTT:

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

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

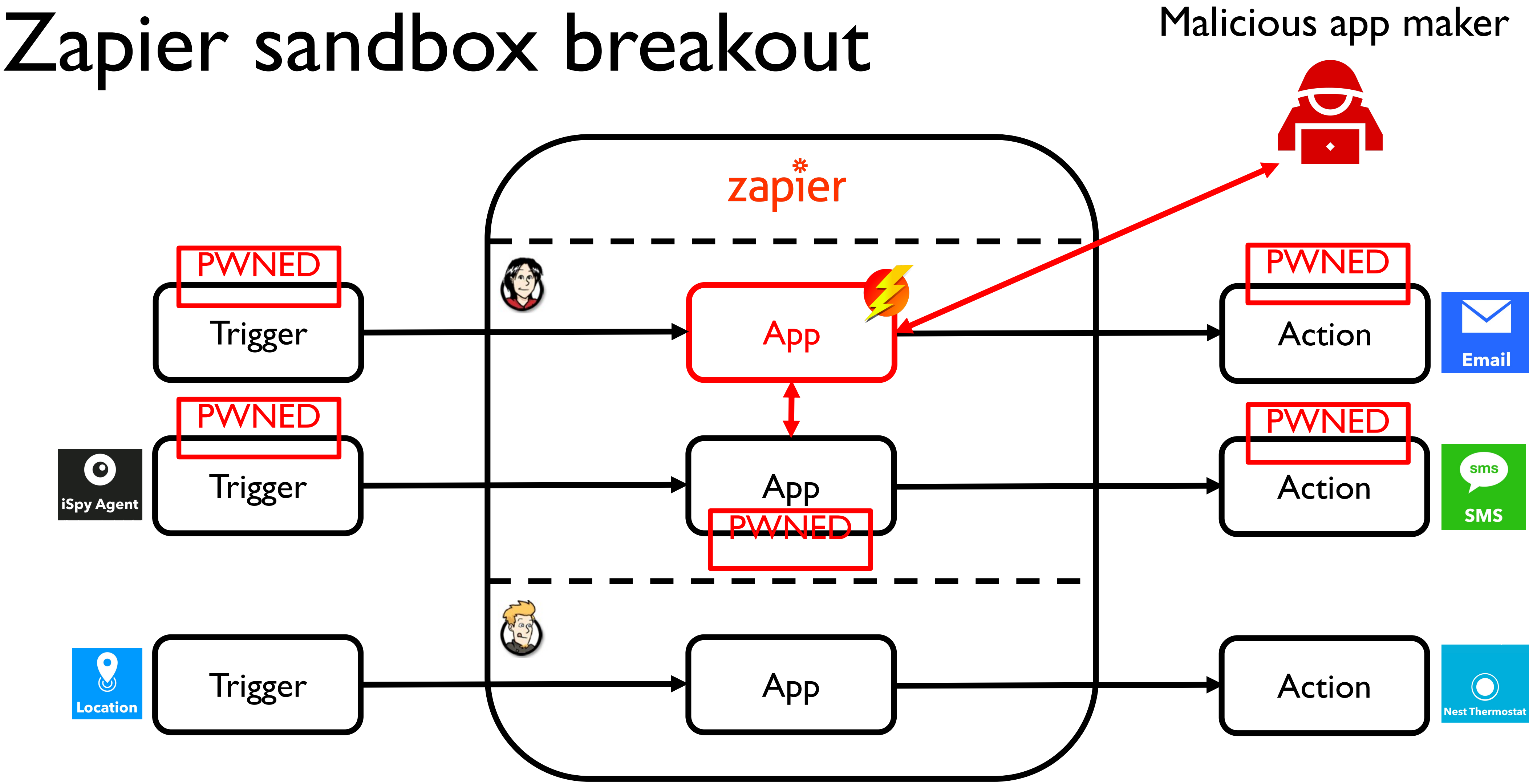
AWS
Lambda

IFTTT sandbox breakout



- Assumption: User installs a *benign* app from the app store
- Compromised: Trigger and action data of the benign app

Zapier sandbox breakout



- Assumption: User installs a malicious app that poses as benign in app store
- Compromised: Trigger and action data of other apps of the same user

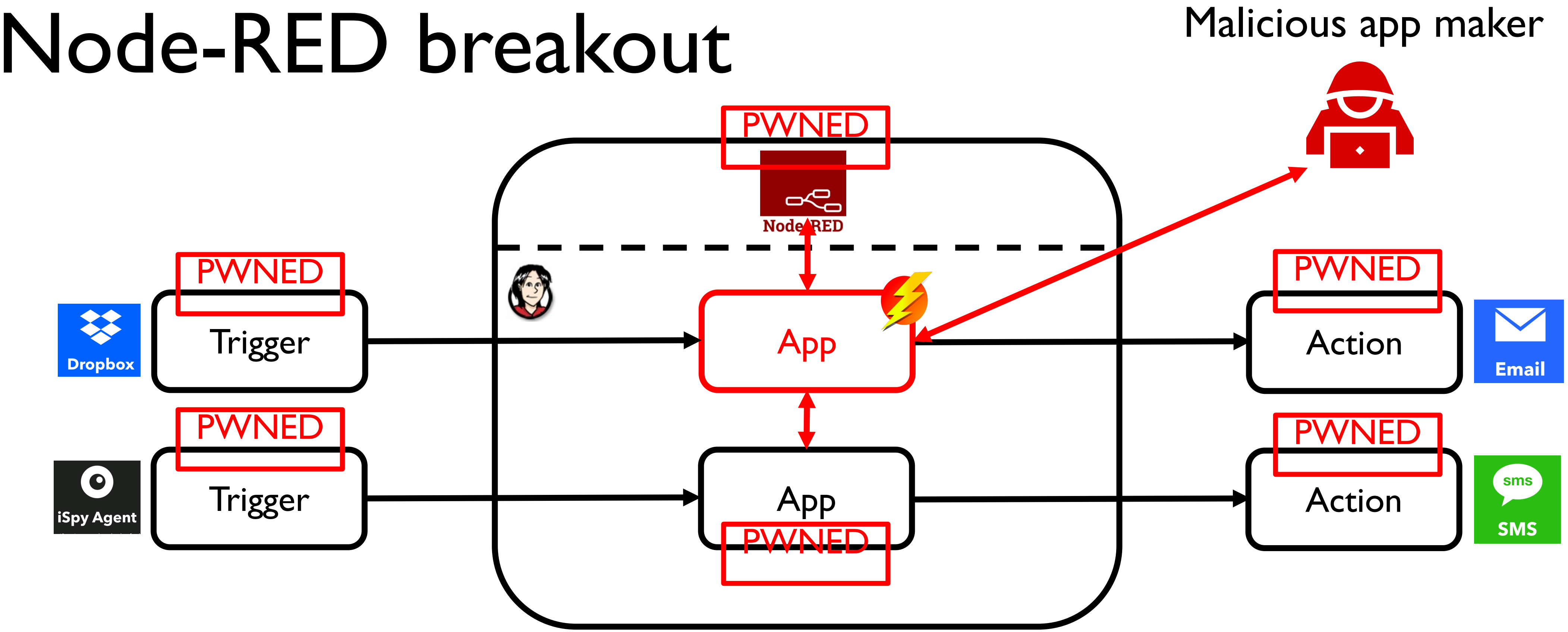
IFTTT breakout explained

- Prototype poisoning of `rapid.prototype.nextInvocation` in AWS Lambda runtime
  - Store trigger incoming data
- Evade security checks
  - Enable `require` via type declaration
  - Enable dynamic code evaluation
    - Manipulate function constructor
    - Pass `require` as parameter
- Use network capabilities of the app via `Email.sendMeEmail.setBody()`

```
declare var require : any;
var payload = ‘try { ...
 let rapid = require("/var/runtime/RAPIDClient.js");
 // prototype poisoning of rapid.prototype.
      nextInvocation
... }`;
var f = (() => {}).constructor.call(null,’require’,
     ’Dropbox’, ’Meta’, payload);
var result = f(require, Dropbox, Meta);
Email.sendMeEmail.setBody(result);
```

- IFTTT’s response
  - vm2 isolation 👍
  - Yet lacking fine-grained policies 🤔

Node-RED breakout



- Assumption: 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 have to
- Malicious Node-RED apps may abuse `child_process` to run arbitrary code

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)

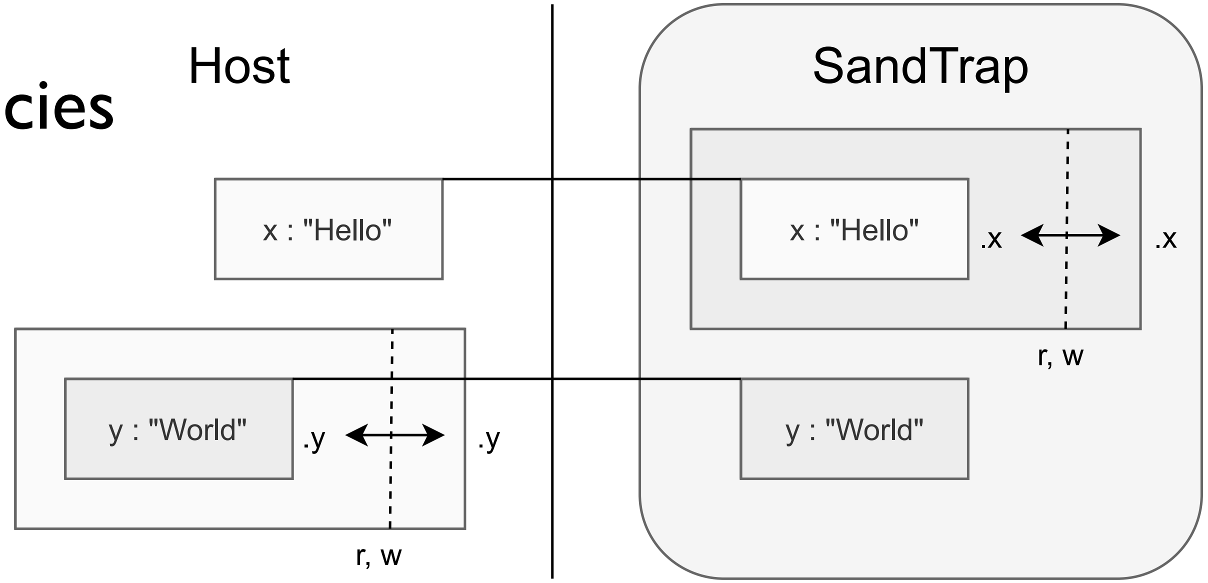
Need fine-grained access control at the level of APIs and their values

Baseline vs. advanced policies

- To aid developers, need
  - Baseline policies once and for all apps per platform
    - Set by platform
  - Advanced policies for specific apps
    - Set by platform but developers may suggest
    - ”Only use allowlisted URLs or emails”
  - Policy generation

SandTrap monitor

- Enforcing
  - read, write, call, construct policies
- Secure usage of modules
  - vs. `isolated-vm` and `Secure ECMAScript`
- Structural proxy-based
  - vs. `vm2`
- Allowlisting policies at four levels
  - module, API, value, context
- Policy generation
  - Execution mode



Baseline policies

IFTTT

- No modules, no APIs other than Trigger/Action
- Read-only `moment` API

zapier

- Read-only protection of Zapier runtime

Node-RED

- No modules, allowlisted calls on RED object

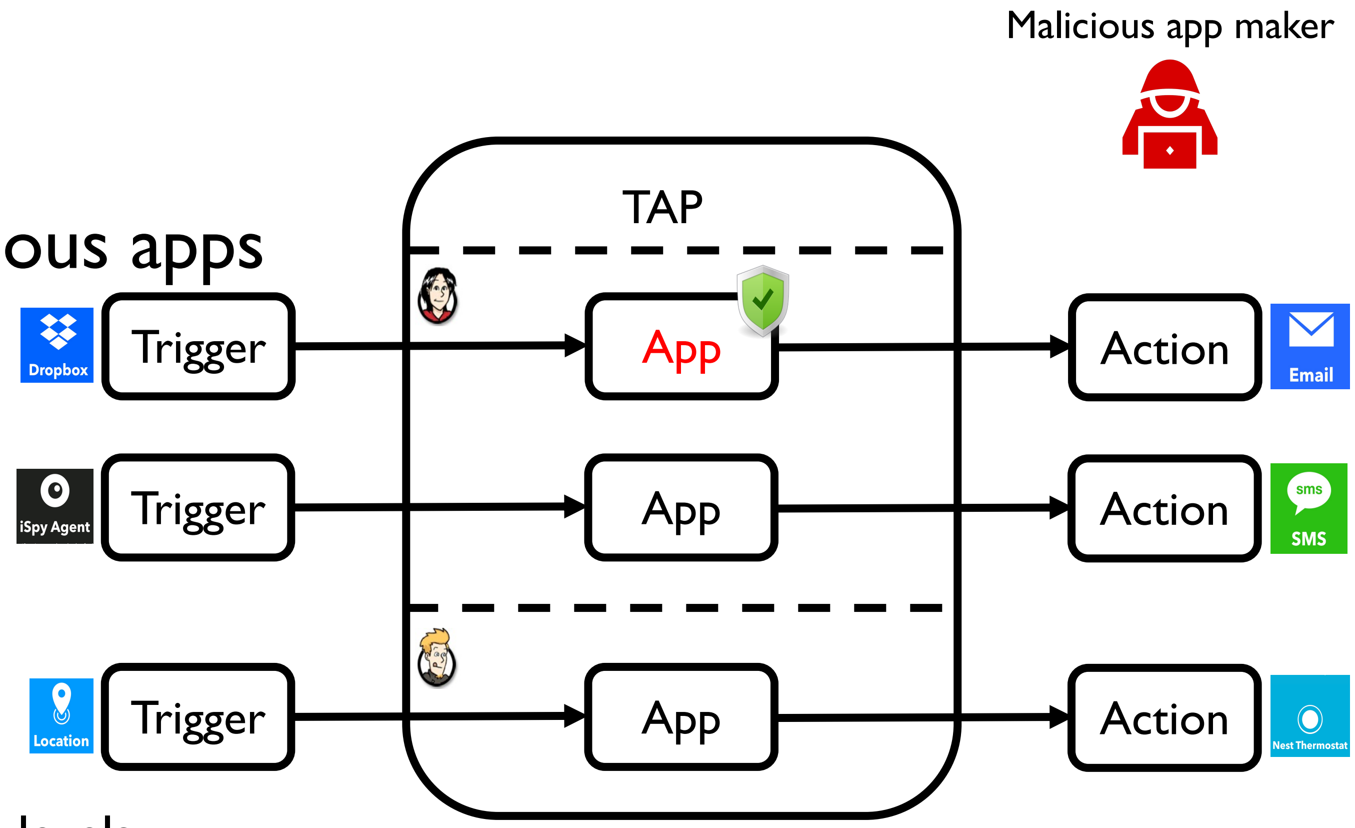
SandTrap benchmarking examples

Platform	Use case	Policy Granularity	Attacks prevented
IFTTT	Baseline	Module/API	Prototype poisoning
	Back up new iOS photos in Dropbox	Value	Leak photo URL
zapier	Baseline	Module/API	Prototype poisoning
	Create a watermarked image using Cloudinary	Value	Exfiltrate the photo
Node-RED	Baseline	Module/API	Run arbitrary code with `child_process`
	Water utility control	Context	Tamper with the tanks and pumps

Worst-case performance overhead under 5ms for most apps

SandTrap takeaways

- IFTTT, Zapier, and Node-RED vulnerable to attacks by malicious apps
  - Breakouts
  - Coordinated disclosure
  - Empirical studies
- SandTrap monitor
  - Policies
    - Baseline & advanced
    - Module-, API-, value-, and context-levels
  - Benchmarking on IFTTT, Zapier, and Node-RED
- Try at `https://github.com/sandtrap-monitor/sandtrap`


Malicious app maker
TAP
Dropbox
Trigger
App
Action
Email
iSpy Agent
Trigger
App
Action
sms
SMS
Location
Trigger
App
Action
Nest Thermostat