SandTrap: Securing JavaScript-driven Trigger-Action Platforms

Andrei Sabelfeld @asabelfeld

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)
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:

```javascript
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
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()**

```javascript
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

- No modules, no APIs other than Trigger/Action
- Read-only moment API
- Read-only protection of Zapier runtime
- No modules, allowlisted calls on RED object
## SandTrap benchmarking examples

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

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