Securing Software in the Presence of Third-Party Modules

Mohammad M. Ahmadpanah

Supervisor: Andrei Sabelfeld
Co-supervisor: Daniel Hedin

Examiner: David Sands
Discussion leader: Deian Stefan (UCSD)

CHALMERS
UNIVERSITY OF TECHNOLOGY

October 1, 2021

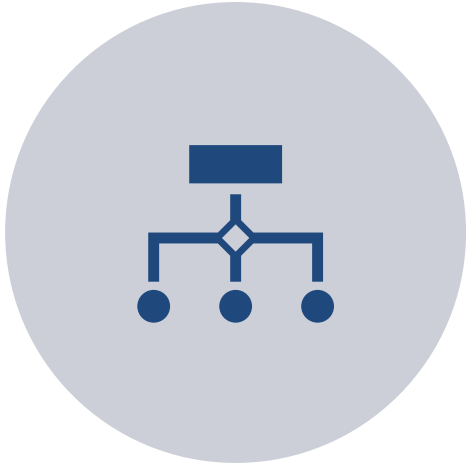
Modular programming

- Code modules
  - Designed and implemented *independently*
  - Often written by *third parties*

  - Security concerns such as:
    - Stealing confidential information
    - Tampering with sensitive data
    - Executing malicious code

Third-party modules: security policies

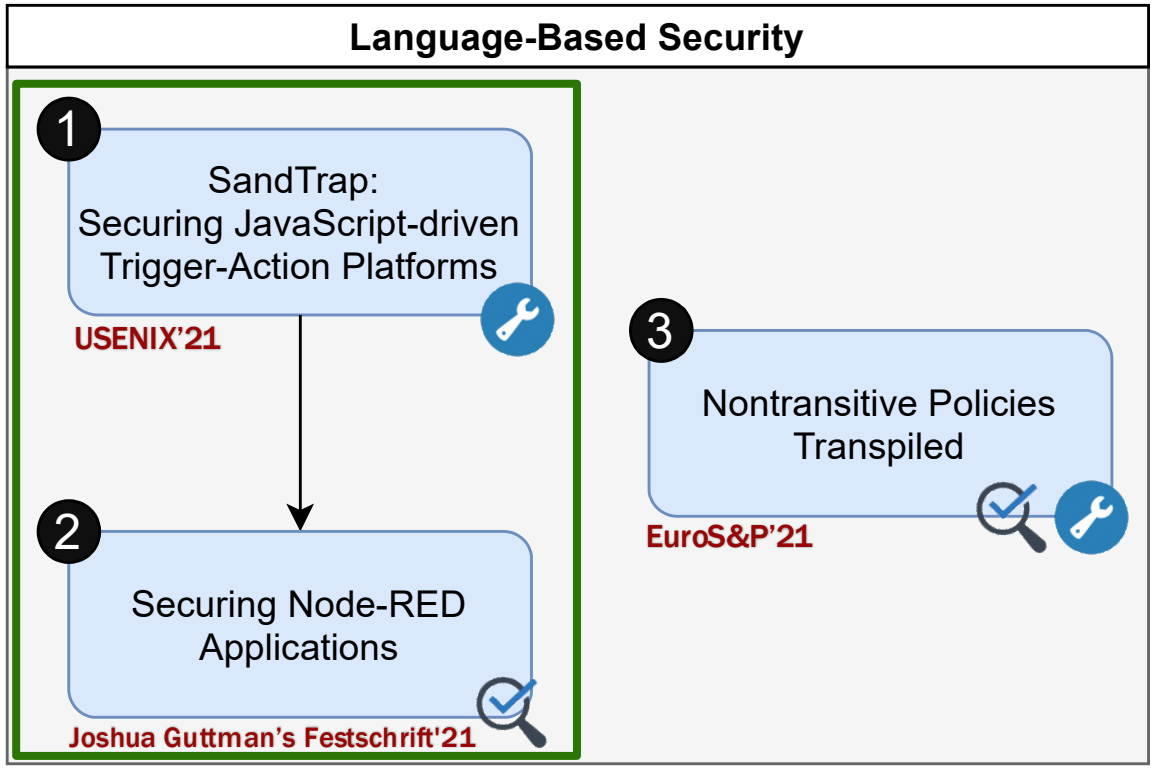
ACCESS CONTROL



INFORMATION-FLOW CONTROL

Papers at a glance

Access Control



A tool presented

Formal proof

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 (, ,…)

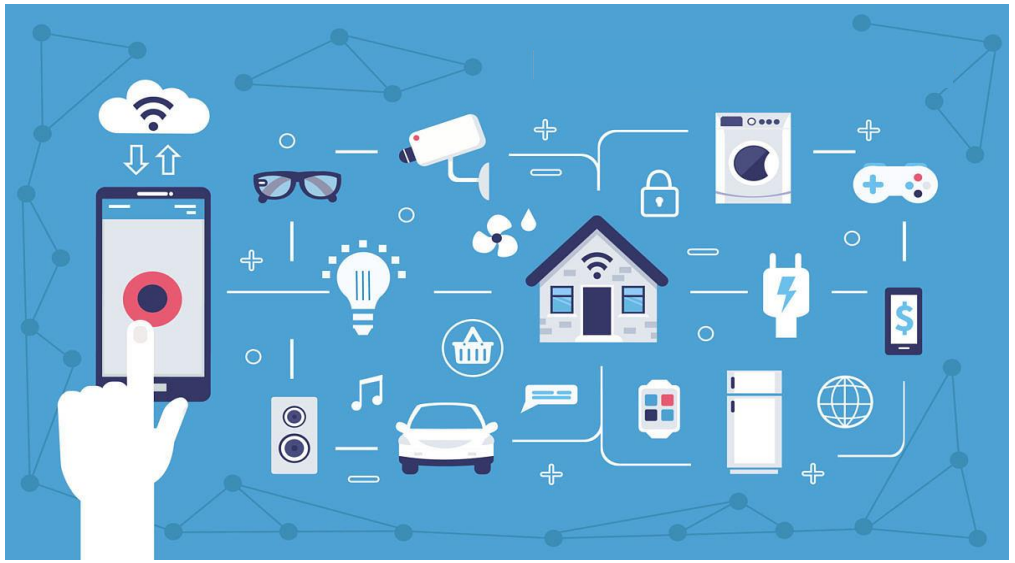


Image: © Irina Strelnikova / Adobe Stock

TAP: Examples

IFTTT

Get an email when your EZVIZ camera senses motion

EZVIZ

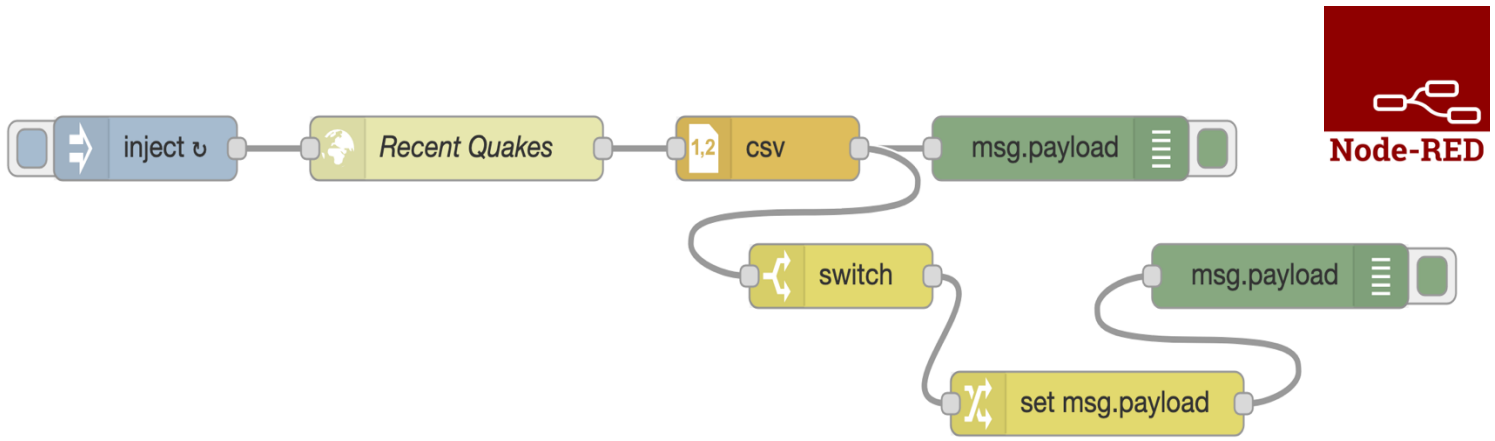
Connect

Save new Instagram photos to Dropbox

zapier

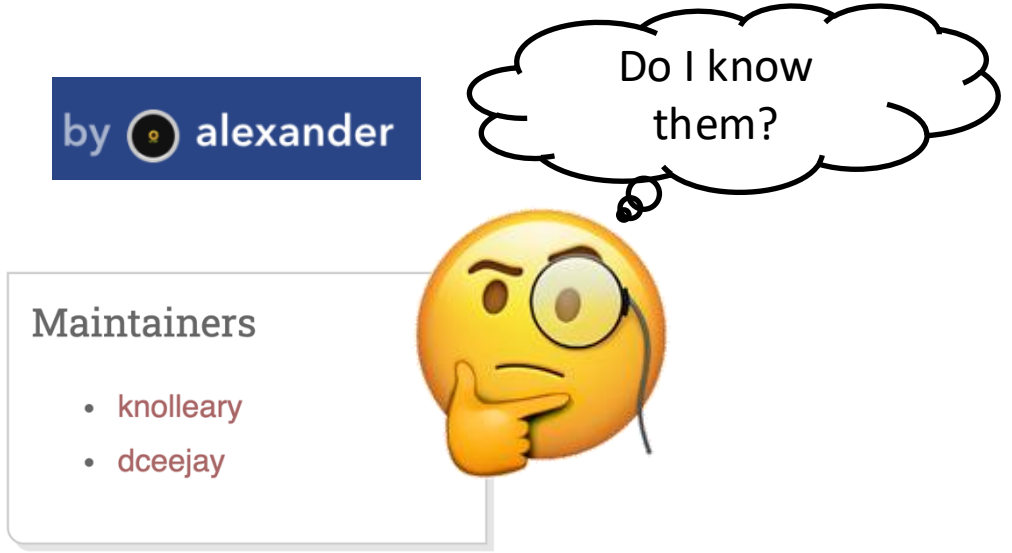
When this happens
Step 1: New Media Posted in My Account

Then do this
Step 2: Upload File



Trigger-Action Platform (cont.)

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



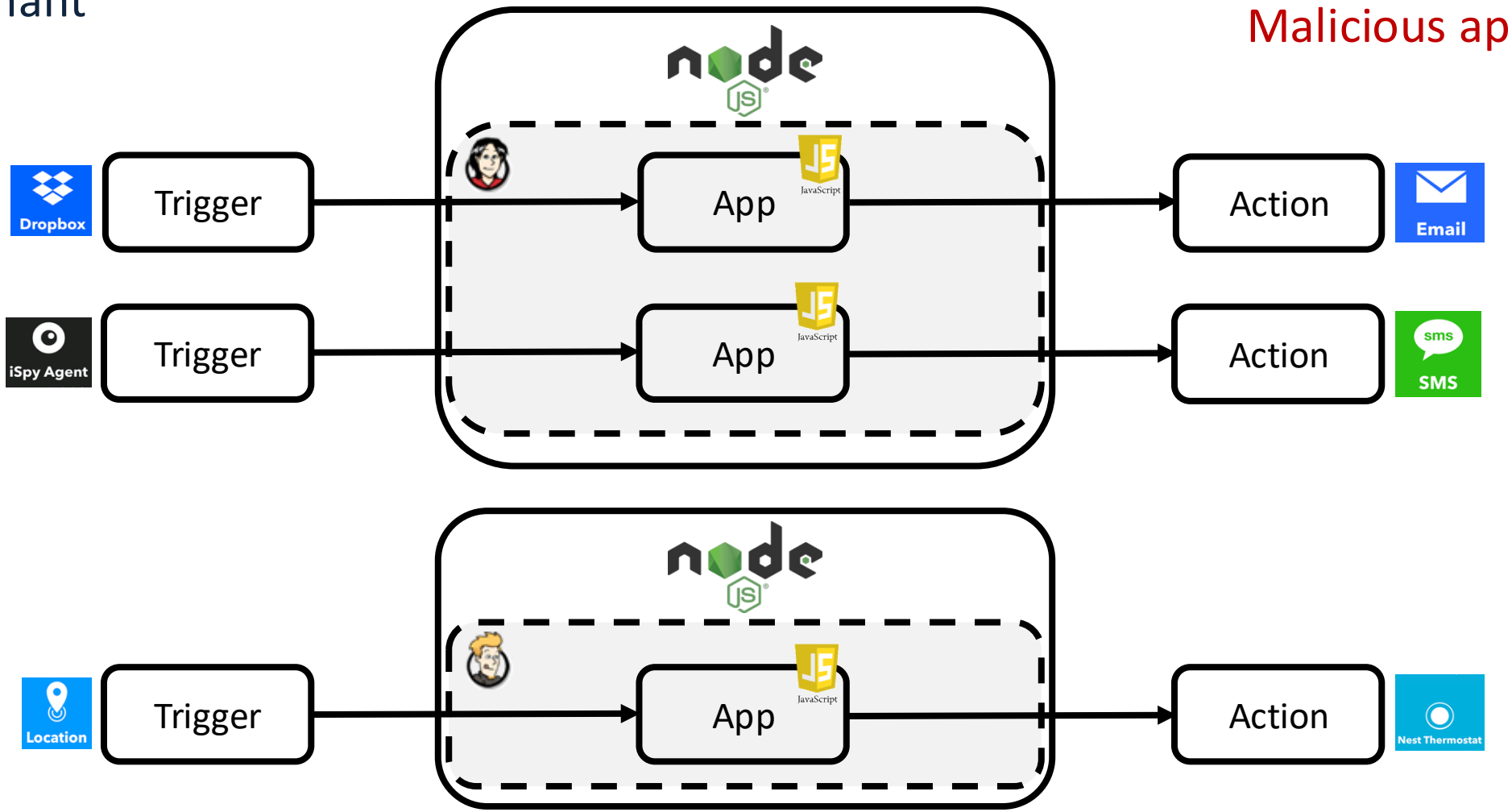
- Popular JavaScript-driven TAPs:
  - IFTTT and zapier (proprietary)
  - Node-RED (open-source)

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

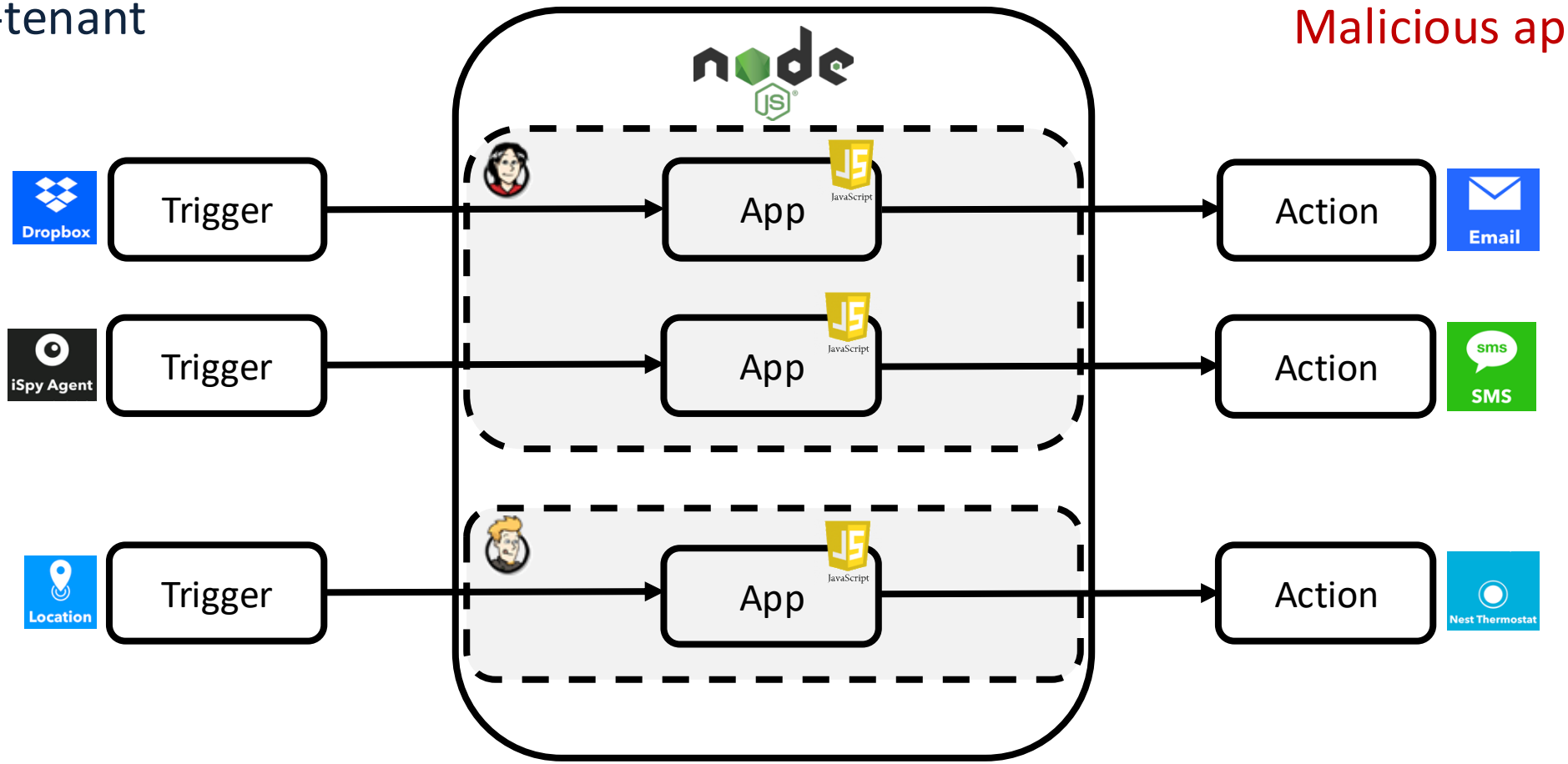
Threat model:
Malicious app maker



TAP architecture (cont.)

IFTTT:
multi-tenant

Threat model:
Malicious app maker



Sandboxing apps in IFTTT and Zapier

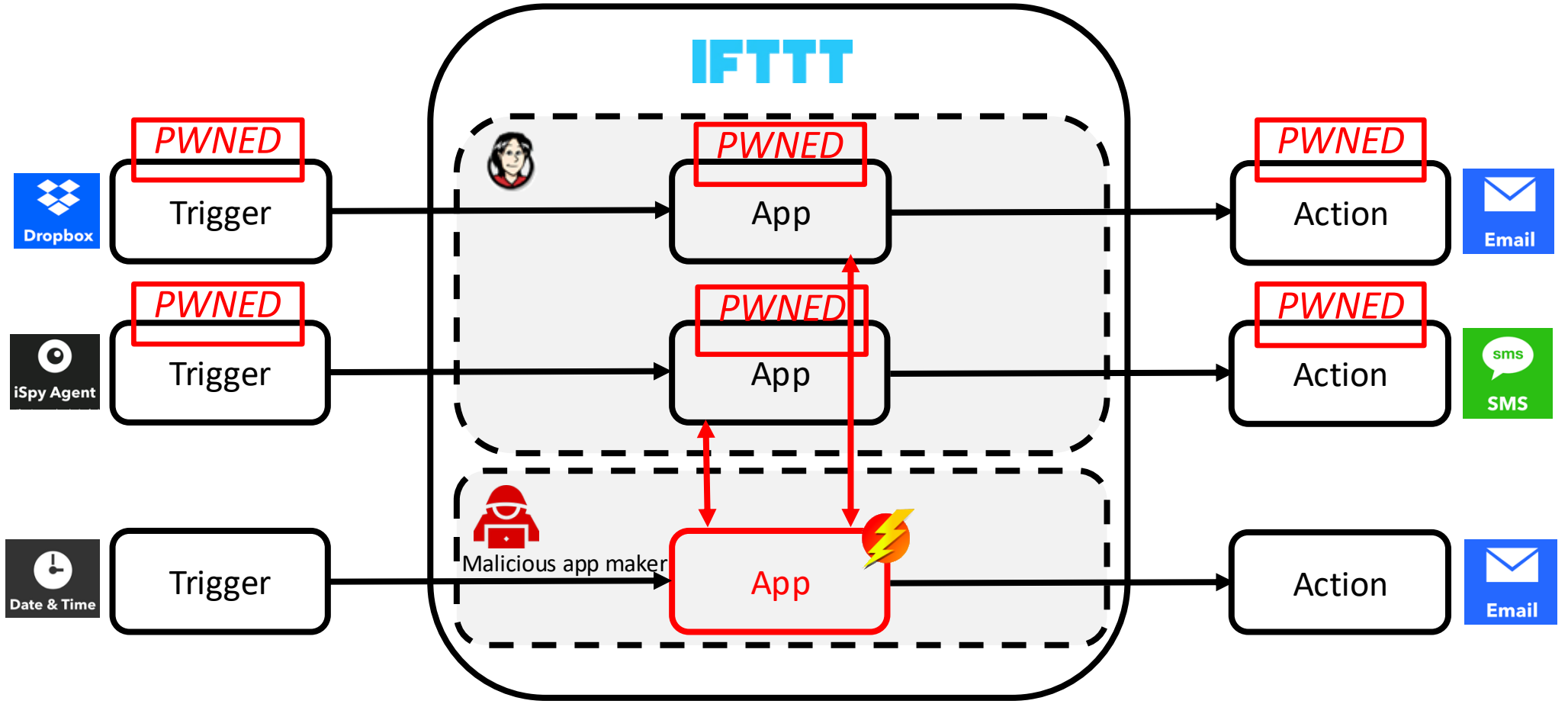
AWS Lambda

- 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

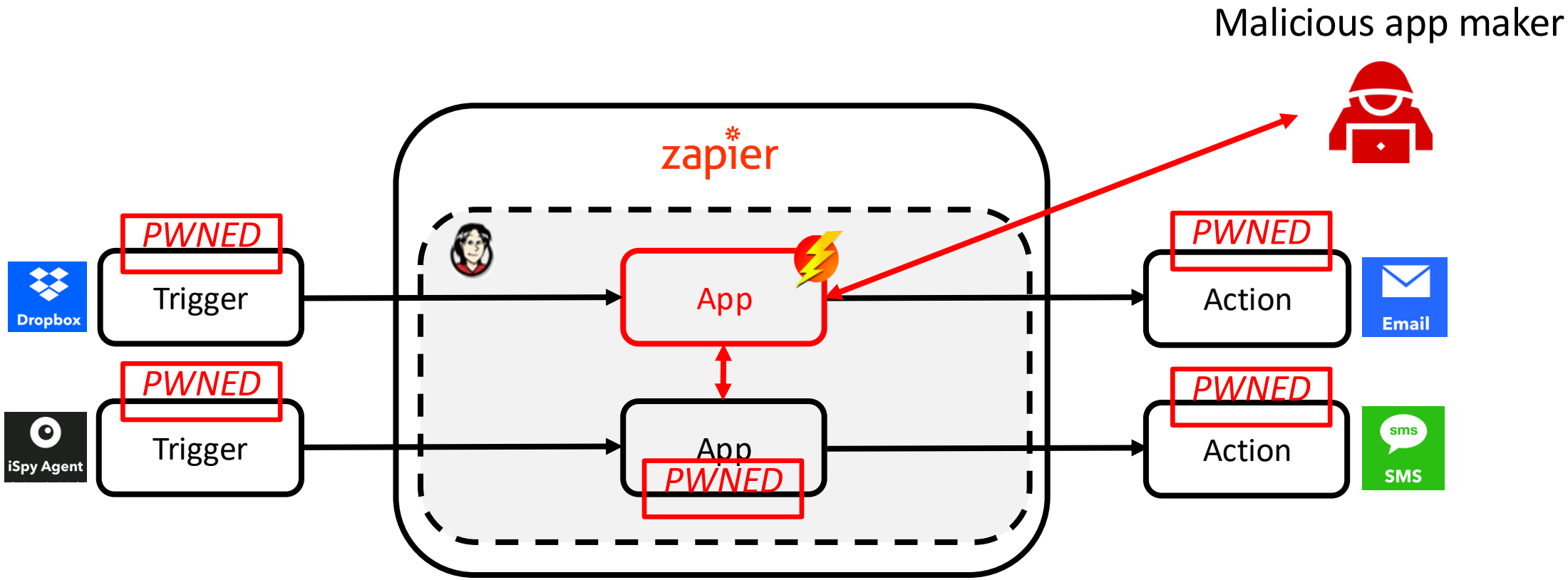
IFTTT sandbox breakout



User installs *benign* apps from the app store

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

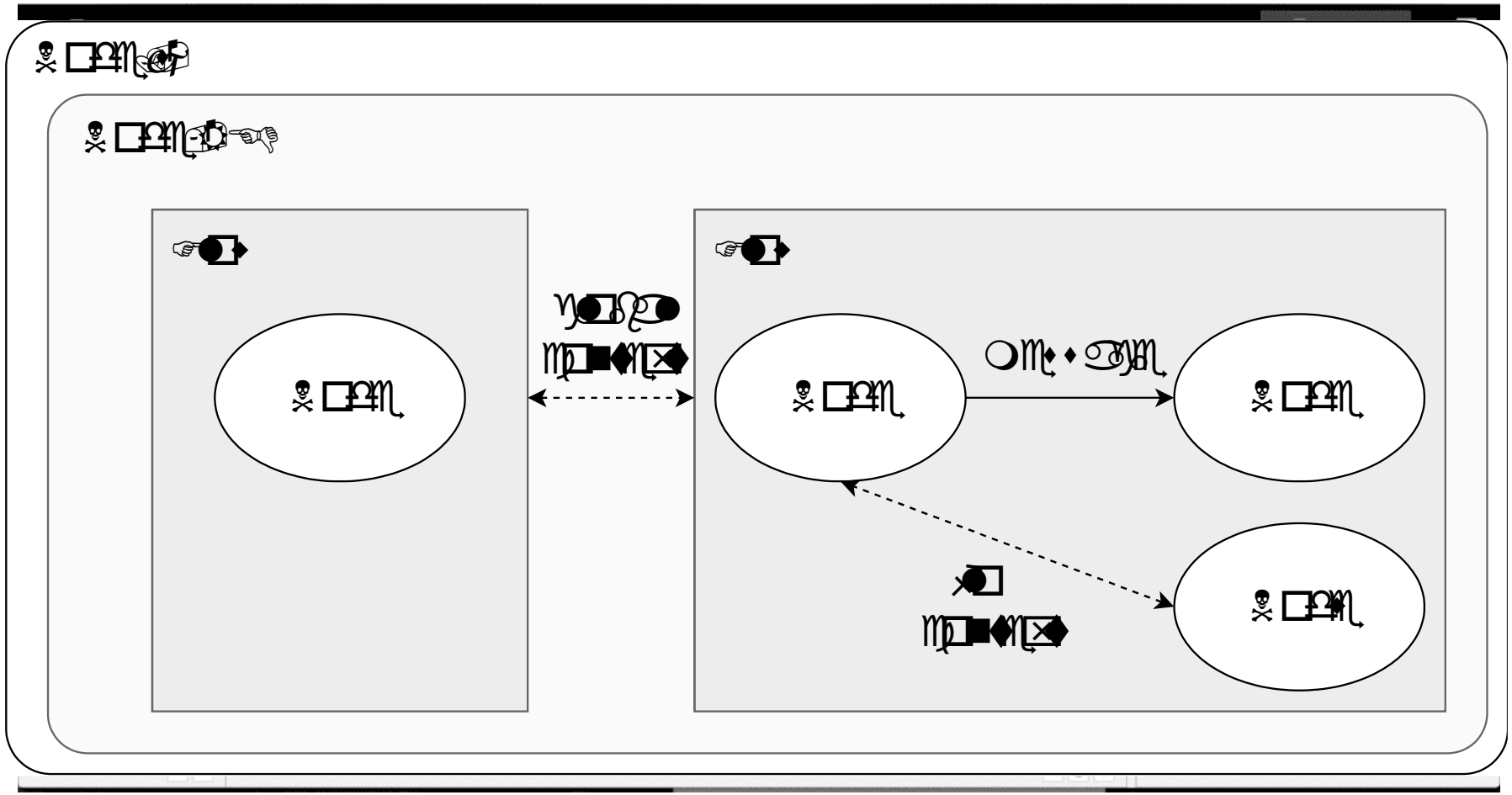
Zapier sandbox breakout



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

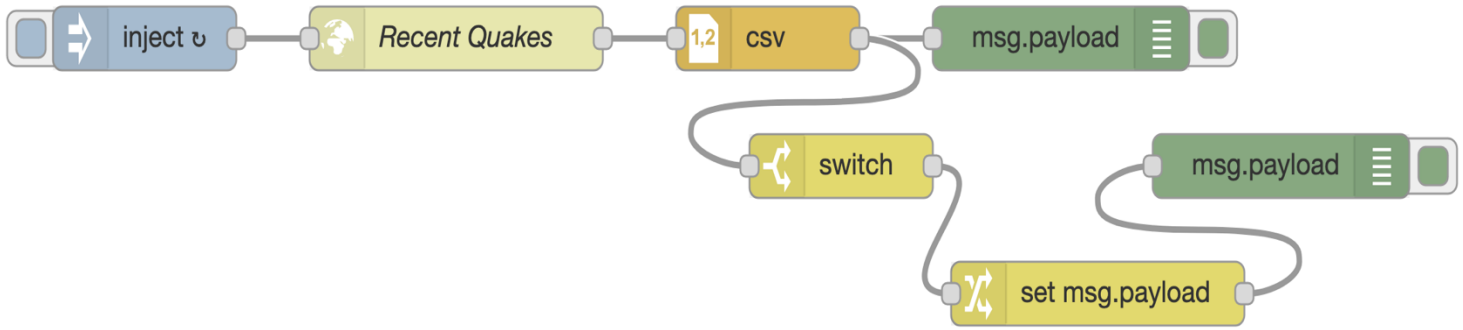
Compromised: Trigger and action data of other apps of the ***same*** user

Node-RED architecture



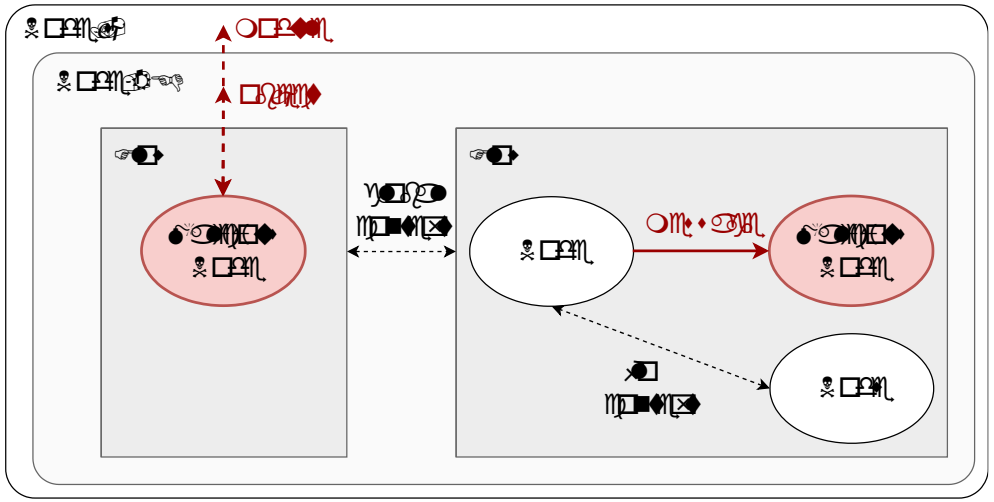
https://blog.techdesign.com/get-started-with-iot-visual-wiring-tool-node-red/

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



Security labeling:

- 408 node definitions and 642 flows
- **70.40**% of flows may violate **privacy**
- **76.46**% of flows may violate **integrity**

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:

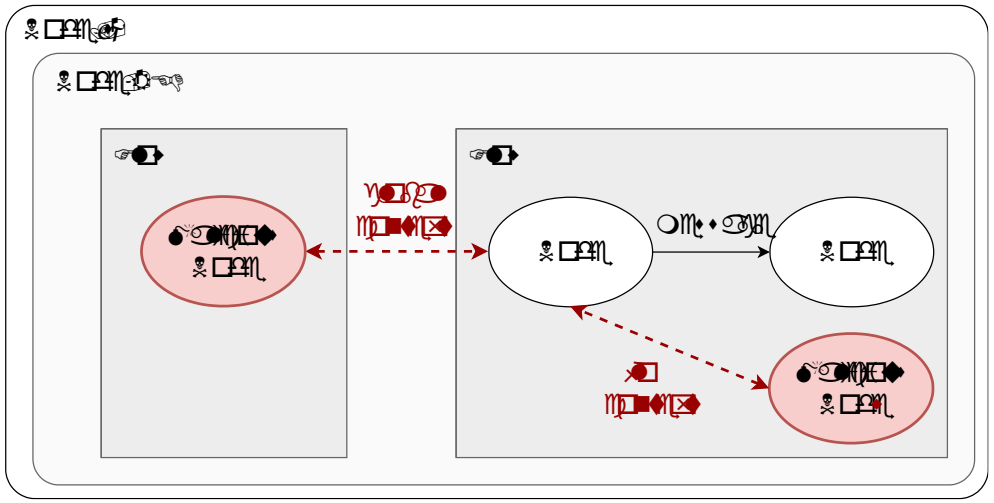
```
sendopts.to = node.name || msg.to;
```

  - 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.)



- **19.31**% of 1181 flows make use of **context**

Malicious node may:

- Exploit **inter-node** communication

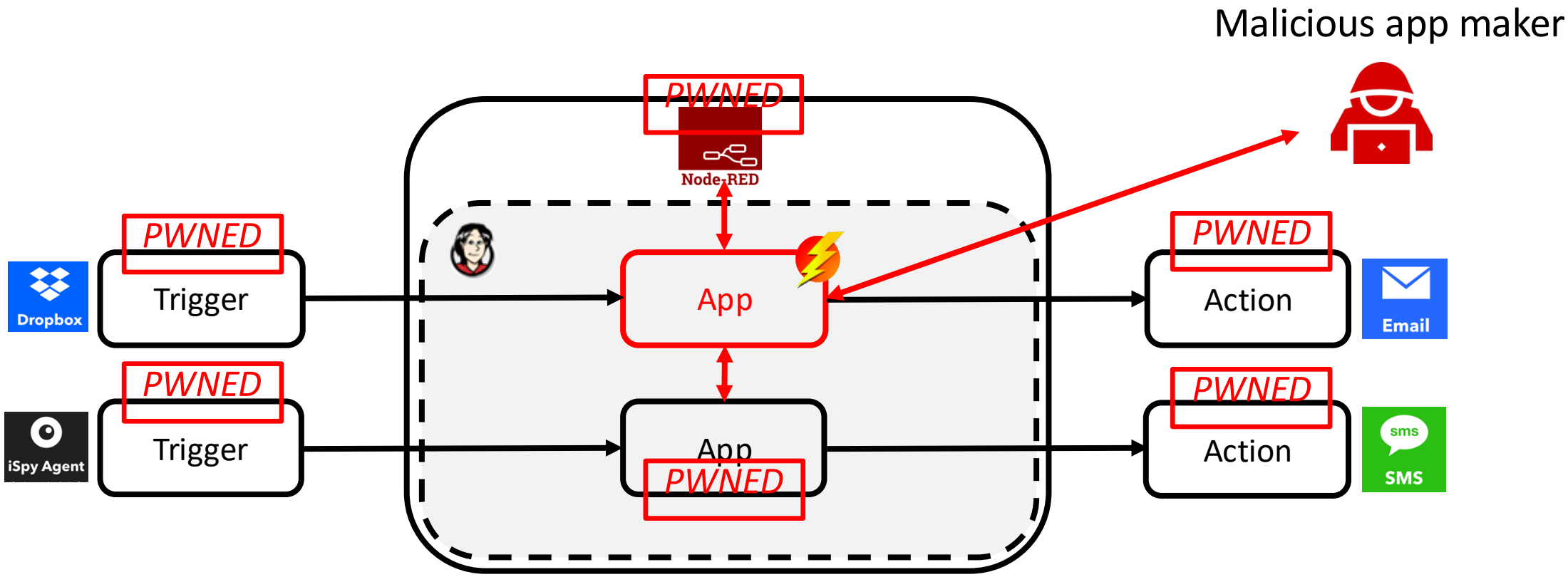
```
global.set("tankLevel", tankLevel);
…
var tankLevel = global.get("tankLevel");
if (tankLevel < 10) pump.stop(); else pump.start();
```

- 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: modeling [presented in Paper 2]

- 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.)

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

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

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

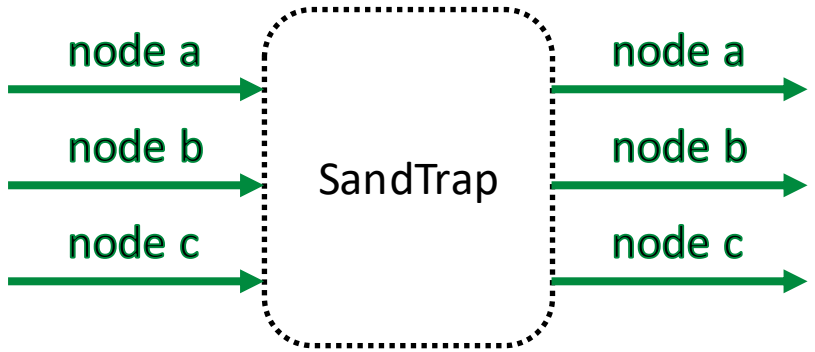
Malicious node attempting to send an email to attacker:

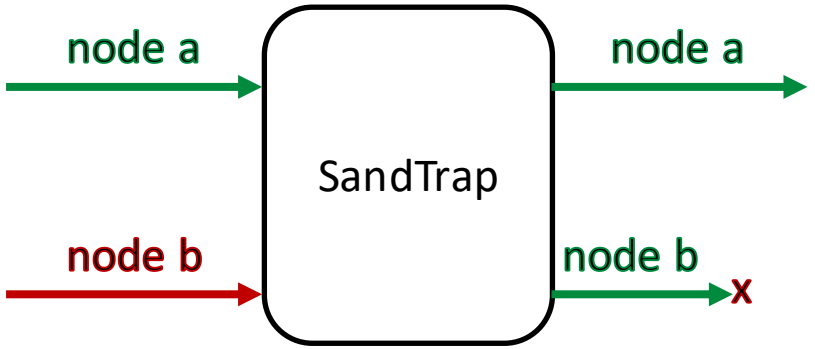
$$\texttt{sendMail} \in P_k \wedge \texttt{"me@attacker.com"} \notin V_k(\texttt{sendMail})$$

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

SandTrap: modeling (cont.)

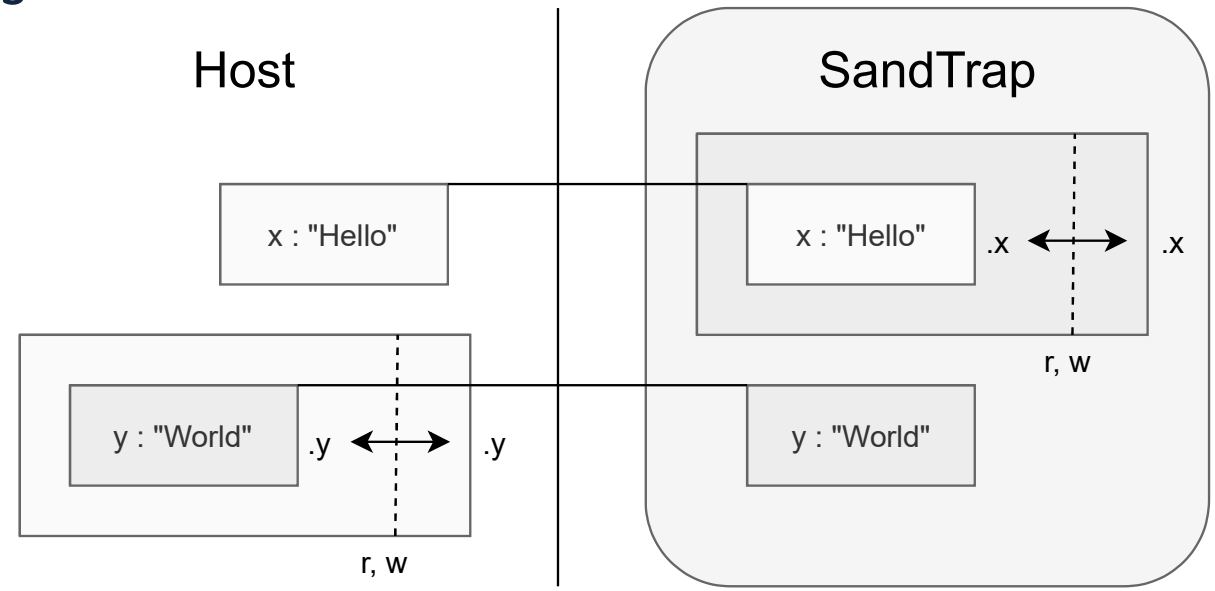
- Soundness
  - *Monitoring at node level enforces global security*
- Transparency
  - *No behavior modification other than raising security error*
  - *The monitor preserves the longest secure prefix of a given trace*



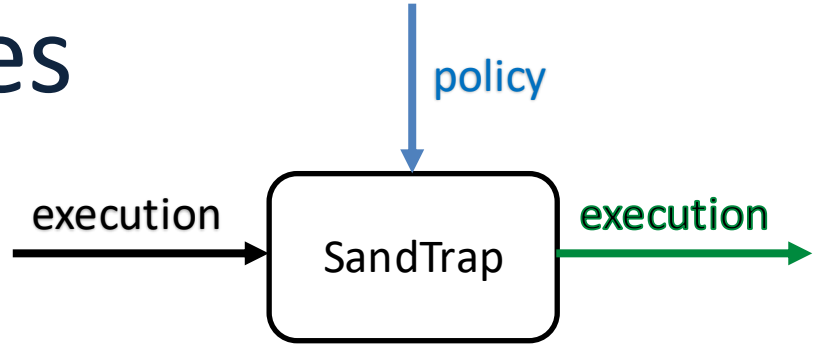


SandTrap: implementation

- 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



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 "},…}`

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…

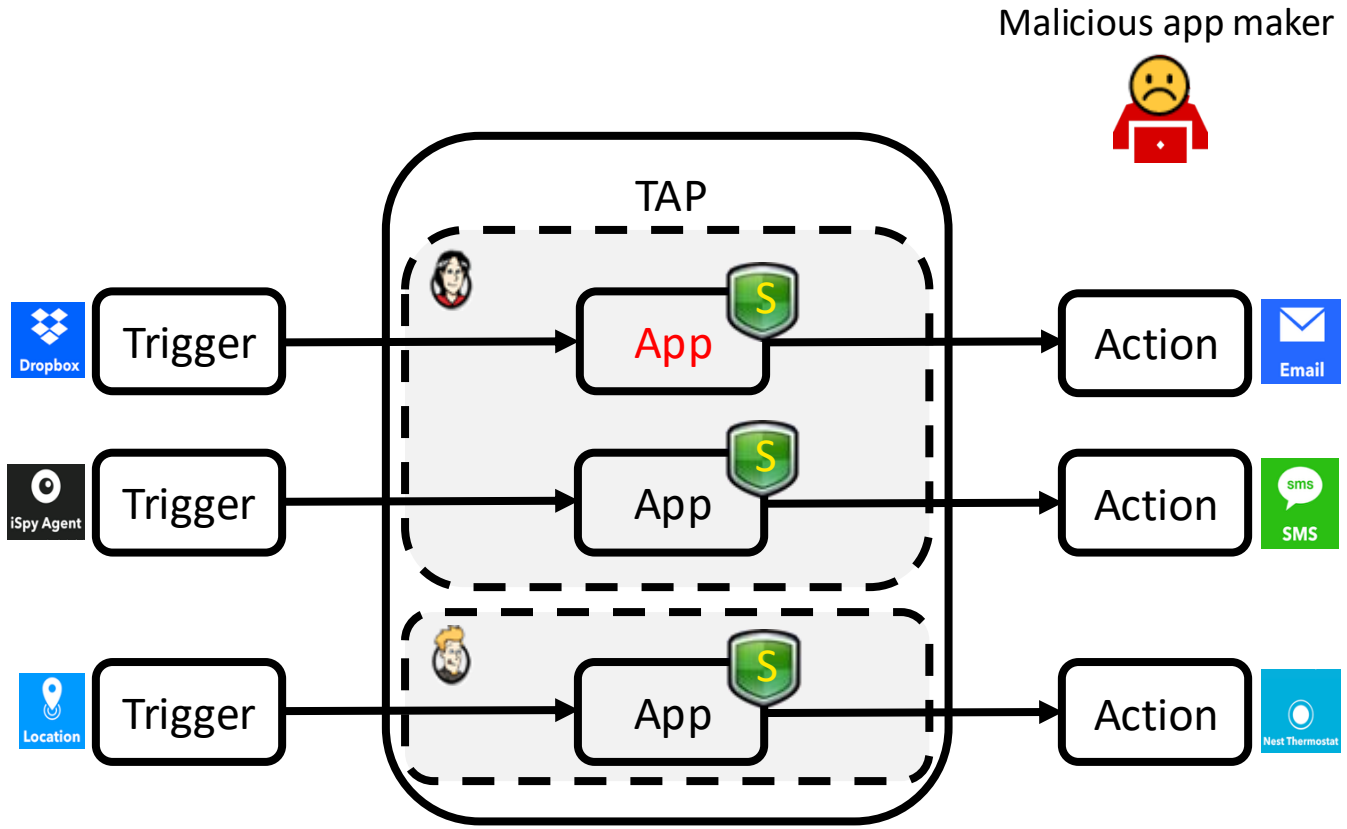
- Baseline policy: No modules, no APIs other than Trigger/Action
- Advanced policies: Fine-grained URL policies
- Overhead: <7ms
- Policy LoC (avg): 185

- 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

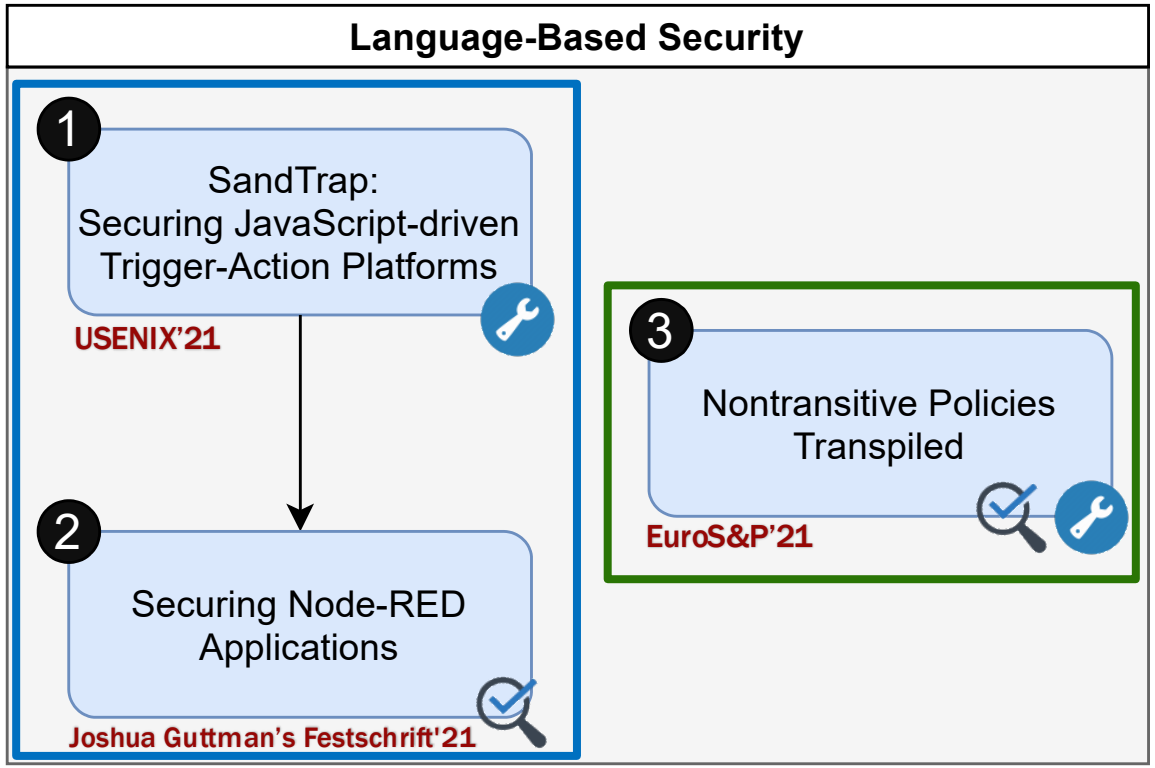
- Structural proxy-based monitor to enforce fine-grained policies for JavaScript
- Formal framework (for a core language)
  - Soundness and transparency



Try at `https://github.com/sandtrap-monitor/sandtrap`

Papers at a glance

Access Control



Information Flow Control

A tool presented

Formal proof

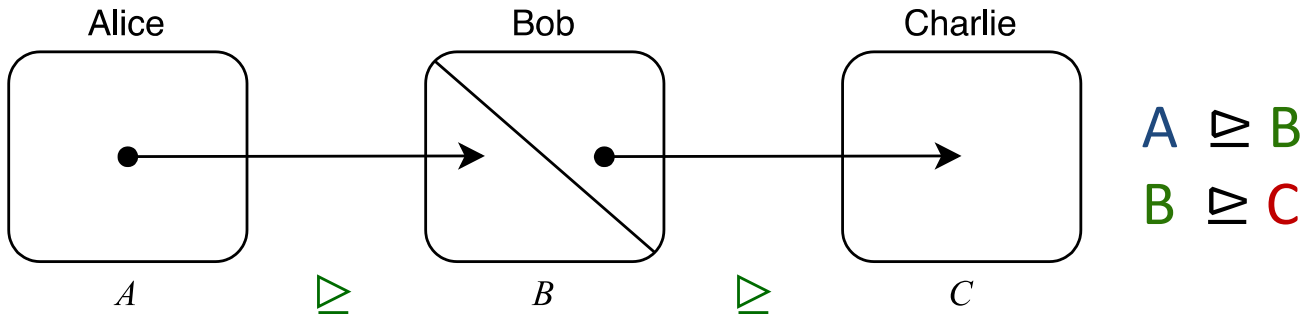
Nontransitive Noninterference (NTNI) [Paper 3]

Nontransitive Security Types for Coarse-grained Information Flow Control

Yi Lu
School of Computer Science
Queensland University of Technology
Brisbane, Australia
yt.lu@qut.edu.au

Chenyi Zhang
College of Information Science and Technology
Jinan University
Guangzhou, China
chenyi_zhang@jnu.edu.cn

CSF'20



Alice

```
data;
main(){
 Bob.good();
 Bob.receive(data);
 Bob.bad();
}
```

Bob

```
data1;
data2;
good(){Charlie.receive(data2)}
receive(x){data1 = x;}
bad(){Charlie.receive(data1)}
```

Charlie

```
data;
receive(x){data = x;}
```

Nontransitive types

$A \trianglerighteq B$

$B \trianglerighteq C$

$$canFlowTo(l) = \{l' | l' \trianglerighteq l\}$$

Alice.data	A
Bob.data1	B
Bob.data2	B
Charlie.data	C

	specified		inferred
$\{B\} \subseteq canFlow(C) = \{B, C\}$	C	`Charlie.data = Bob.data2`	{B}
$\{A\} \subseteq canFlow(B) = \{A, B\}$	B	`Bob.data1 = Alice.data`	{A}
$\{A, B\} \not\subseteq canFlow(C) = \{B, C\}$	C	`Charlie.data = Bob.data1`	{A,B}

NTNI reduces to TNI

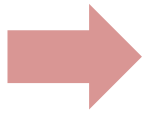
- Standard (transitive) information flow machinery **can** enforce nontransitive noninterference
- Two steps:
  - Program transformation
  - Lattice encoding
- The core idea: don't drop the lattice assumption

use ***power lattice*** in the *transformed program* and keep using TNI

Program transformation: running example

1) replace vars with internal *temp* vars
2) prepend *init* assignments (*source* vars)
3) append *final* assignments (*sink* vars)

```
1  // Bob.receive(data)
2  Bob.data1 := Alice.data;
3  // Bob.good()
4  Charlie.data := Bob.data2;
5  // Bob.bad()
6  Charlie.data := Bob.data1;
```



```
1  // init
2  Alice.data_temp := Alice.data;
3  Bob.data1_temp := Bob.data1;
4  Bob.data2_temp := Bob.data2;
5  Charlie.data_temp := Charlie.data;
6
7  Bob.data1_temp := Alice.data_temp;
8  Charlie.data_temp := Bob.data2_temp;
9  Charlie.data_temp := Bob.data1_temp;
10
11 // final
12 Alice.data_sink := Alice.data_temp;
13 Bob.data1_sink := Bob.data1_temp;
14 Bob.data2_sink := Bob.data2_temp;
15 Charlie.data_sink := Charlie.data_temp;
```

init

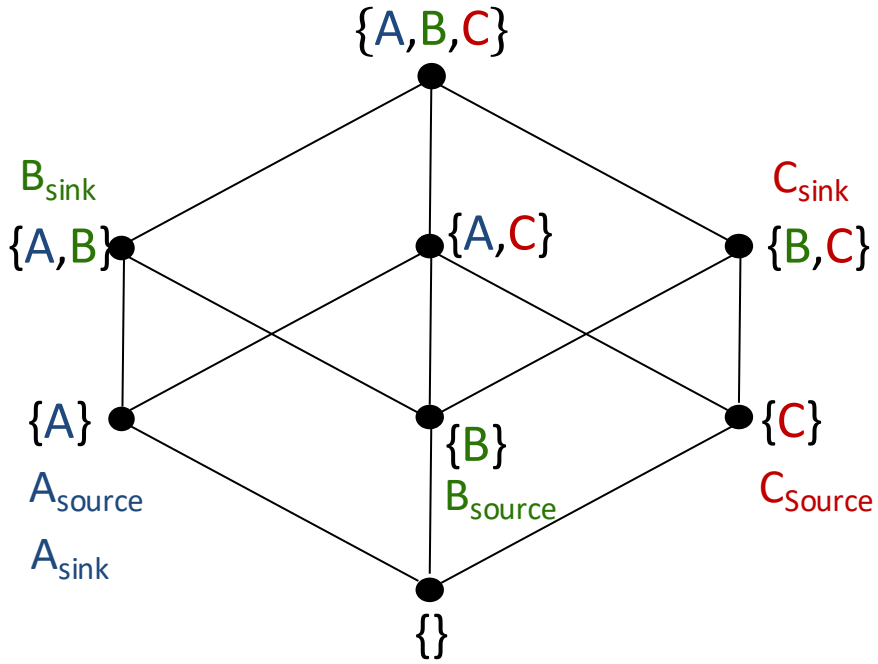
final

The transformed program is *semantically equivalent* to the original
(modulo renaming and having temp and final variables)

Lattice encoding: powerset lattice

$$A \trianglerighteq B$$
$$B \trianglerighteq C$$

$$l_{source} = \{l\}$$
$$l_{sink} = canFlowTo(l) = \{l' | l' \trianglerighteq l\}$$



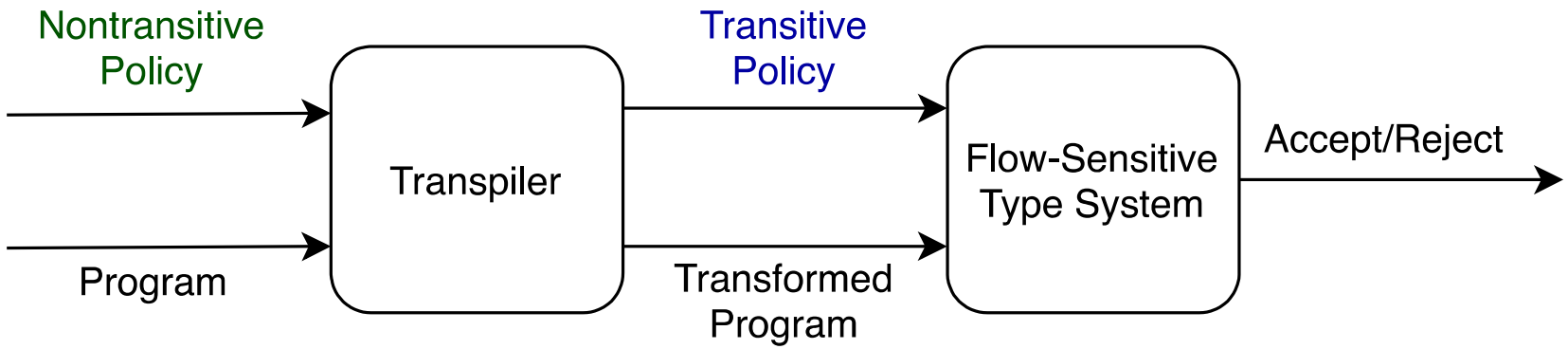
NTNI to TNI

Theorem 2 (*From* $NTNI_{TI}$ *to* TNI_{TI})**.** For any program c and any nontransitive security policy $\mathcal{N} = \langle L_{\mathcal{N}}, \trianglerighteq, \Gamma_{\mathcal{N}} \rangle$, there exist a semantically equivalent (modulo canonicaliztion) program c' and a transitive security policy $\mathcal{T} = \langle L_{\mathcal{T}}, \sqsubseteq, \Gamma_{\mathcal{T}} \rangle$, as specified in Definition 5, such that $NTNI_{TI}(\mathcal{N}, c) \iff TNI_{TI}(\mathcal{T}, c')$. Formally,

$$\forall \mathcal{N}.\, \forall c.\, \exists \mathcal{T}.\, \exists c'.\, c \simeq_C c' \wedge NTNI_{TI}(\mathcal{N}, c) \iff TNI_{TI}(\mathcal{T}, c').$$

What's next?

Nontransitive types to flow-sensitive types



- For the small calculus:
  - Flow-sensitive type system of [Hunt & Sands, POPL'06] is strictly ***more permissive*** than the specialized type system of [Lu & Zhang, CSF'20]
- For Java:
  - Case studies using JOANA information flow analyzer [Hammer & Snelting, 2020]

JOANA-based analysis

```
1  setLattice e<=A,e<=B,e<=C,A<=AB,A<=AC,B<=AB,
2     B<=BC,AB<=ABC,C<=AC,C<=BC,AC<=ABC,BC<=ABC
3  source Alice.data_source   A
4  sink   Alice.data_sink     A
5  source Bob.data1_source    B
6  sink   Bob.data1_sink      AB
7  source Bob.data2_source    B
8  sink   Bob.data2_sink      AB
9  source Charlie.data_source C
10 sink   Charlie.data_sink   BC
11 run    classical-ni
```

the powerset lattice

labeling

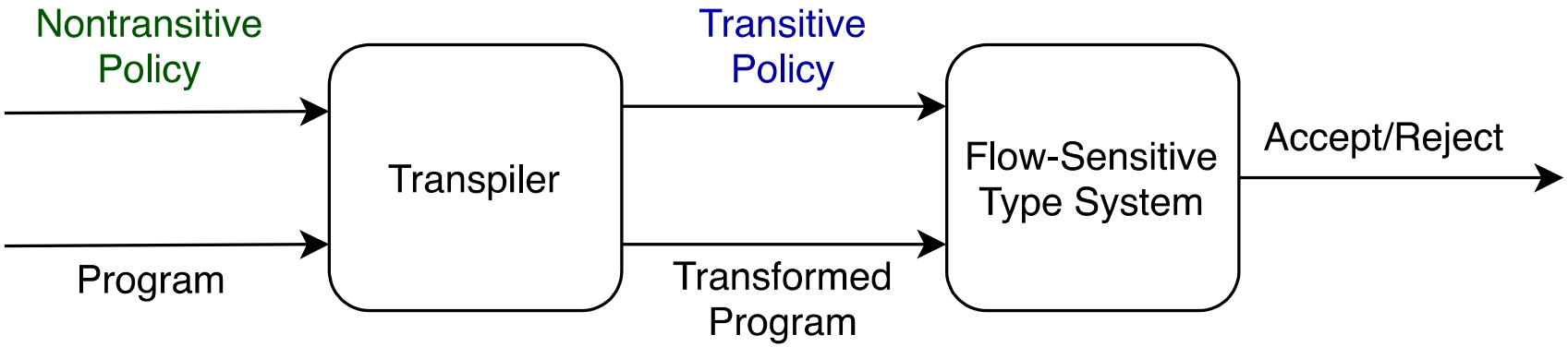
run the flow-sensitive analysis

JOANA

```
Illegal flow from
Alice.data_source to
Charlie.data_sink,
visible for BC
```

NTNI-to-TNI takeaways

- Inspired by Lu & Zhang work on nontransitive noninterference
- Our paper shows NTNI can be reduced to TNI, thus
  - reuse of the existing info flow machinery to enforce nontransitive policies



- Paper details: `https://www.cse.chalmers.se/research/group/security/ntni`

Included papers

Access Control

1 **SandTrap: Securing JavaScript-driven Trigger-Action Platforms**

Mohammad M. Ahmadpanah*, Daniel Hedin*,†, Musard Balliu‡, Lars Eric Olsson*, and Andrei Sabelfeld*

USENIX'21

2 **Securing Node-RED Applications**

Mohammad M. Ahmadpanah[1,✉], Musard Balliu[2], Daniel Hedin[1,3],
Lars Eric Olsson[1], and Andrei Sabelfeld[1]

Joshua Guttman's Festschrift'21

Information Flow Control

3 **Nontransitive Policies Transpiled**

Mohammad M. Ahmadpanah*, Aslan Askarov†, and Andrei Sabelfeld*

EuroS&P'21

Time for Discussion

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

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	

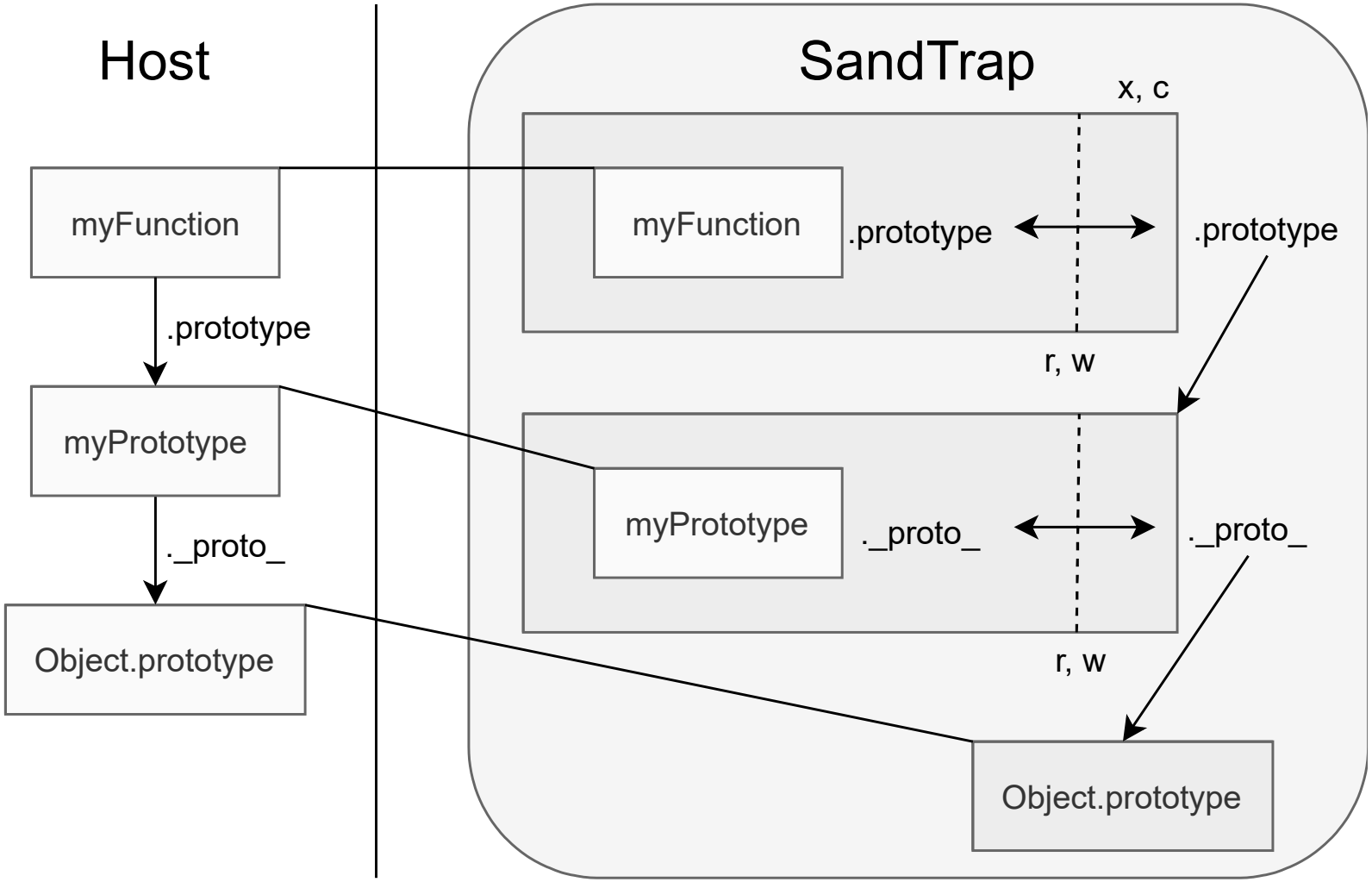
IFTTT breakout explained

- 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 🤔

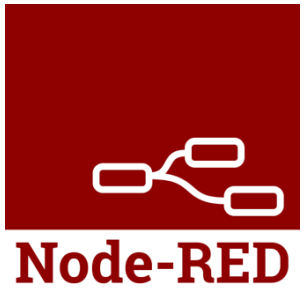
SandTrap implementation



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

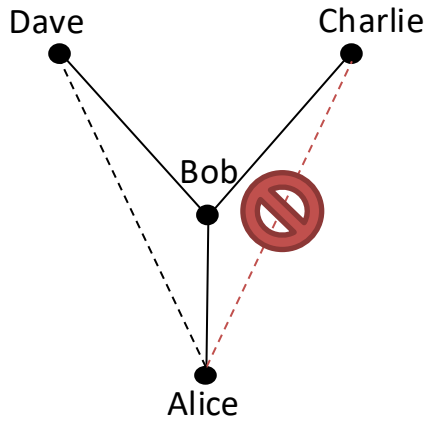
Tool	Isolation	Policy type	Policy generation	Full JavaScript and CJS support	Breakouts addressed	Local object views	Proxy control	Controlled cross-domain prototype modification	Fine-grained access control
vm2	vm + proxy membranes	Module mocking and API level JavaScript injection	×	✓	✓	×	×	×	×
JSand	SES + 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	✓	✓	✓	✓	✓	✓	✓

Nontransitive policies vs. tradition

The argument for transitivity of the flow relation

> "Since $A \rightarrow B$ implies permission to move a value $\mathbf{x}$ from an object in A to one in B, and $B \rightarrow C$ implies it is in turn permissible to move move $\mathbf{x}$ to an object in C, an inconsistency arises if $A \nrightarrow C$"

[D. Denning, *A lattice model for secure information flow,* 1976]



Nontransitive ≠ Intransitive
(confinement) (declassification)

Programs with I/O

- Same lattice encoding: powerset lattice
- Straightforward program transformation
  - `input(x,`l`)` $\mapsto$ `input(x,` $\boldsymbol{l_{source}}$`) = input(x,{`$\boldsymbol{l}$`})`
  - `output(x,`l`)` $\mapsto$ `output(x,` $\boldsymbol{l_{sink}}$`) = output(x,`$\boldsymbol{canFlowTo(l)}$`)`

- Similar reduction result for *progress-insensitive* notion of NTNI and TNI

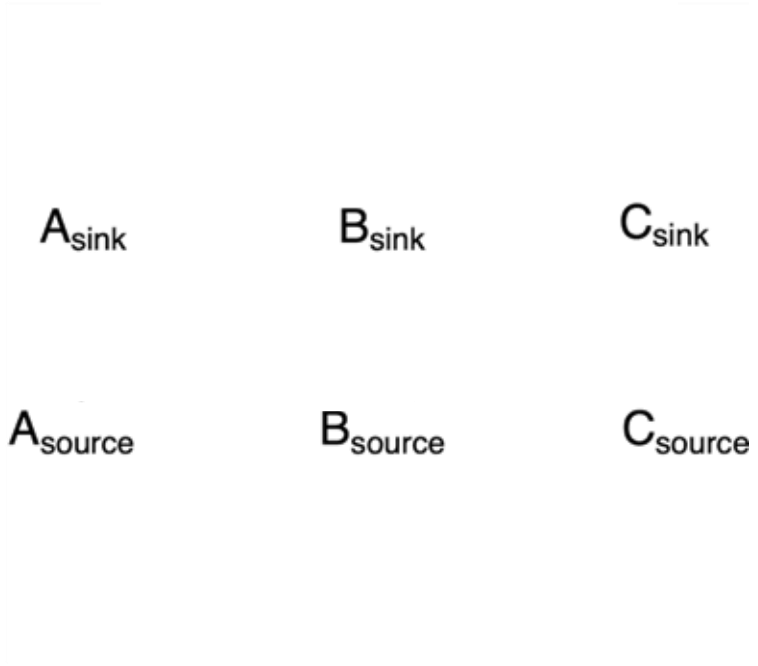
$$\forall \mathcal{N}.\, \forall c.\, \exists \mathcal{T}.\, \exists c'.\, c \simeq_T c' \wedge NTNI_{PI}(\mathcal{N}, c) \iff TNI_{PI}(\mathcal{T}, c')$$

- Similar flow-sensitive type system as the enforcement mechanism

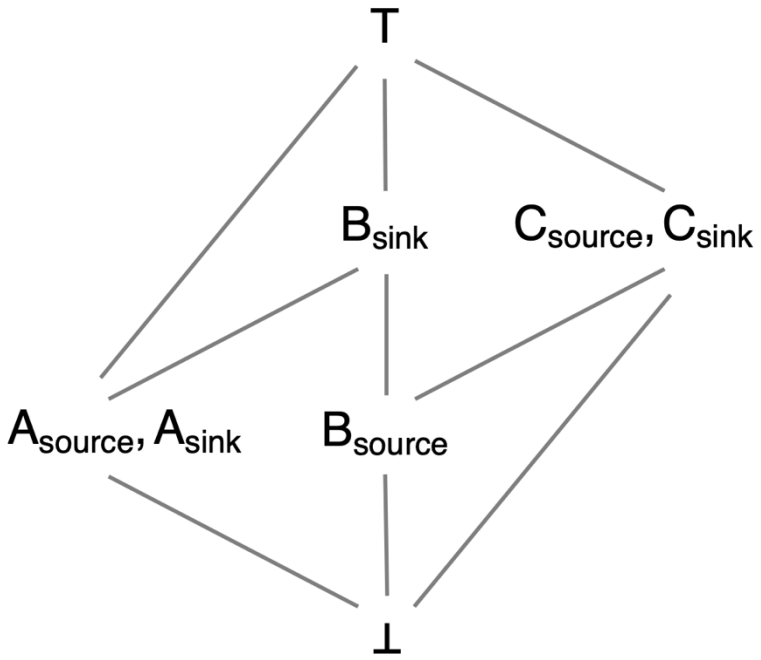
Alternatives to powerset lattice

$A \trianglerighteq B$

$B \trianglerighteq C$



Source-sink lattice
(via Dedekind-MacNeille completion algorithm)



Minimal lattice