Fixing the Dripping TAP:
Security and Privacy in Trigger-Action Platforms

Mohammad M. Ahmadpanah

CHALMERS
UNIVERSITY OF TECHNOLOGY

Dey 6, 1402
Amirkabir University of Technology

whoami

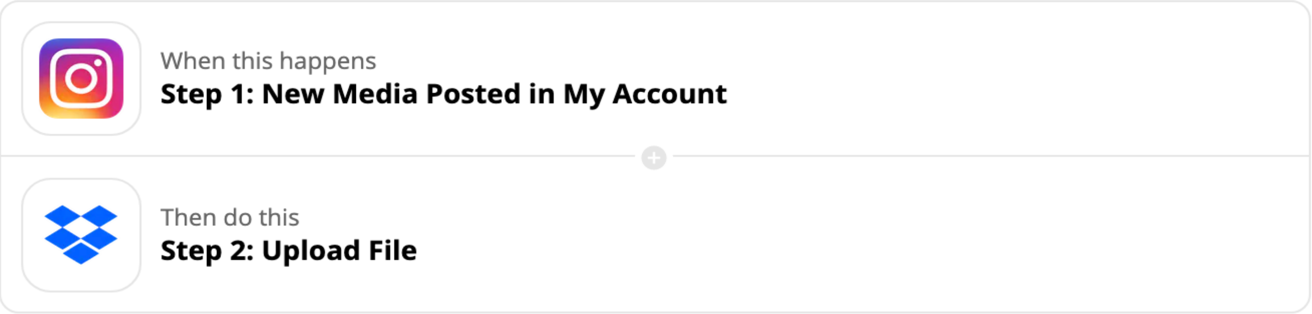
90-94	94-96	96-98	98-03
BSc Software	**MSc Info Sec**	**PhD (!) Software**	**PhD Info Sec**

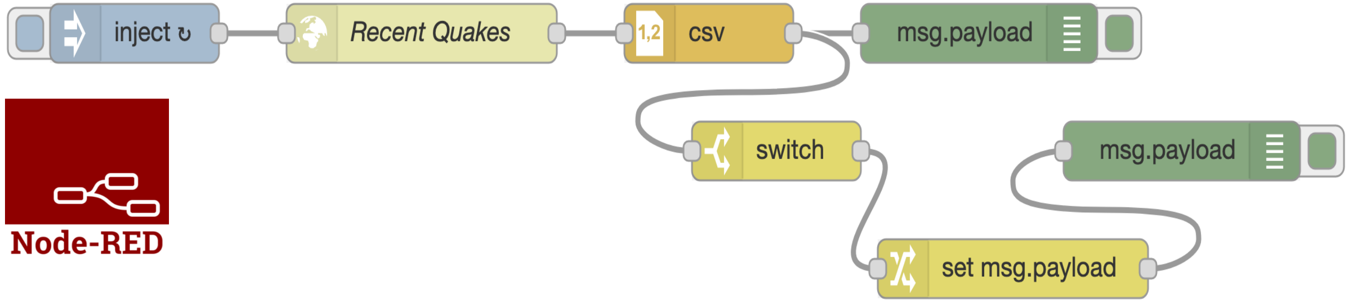
- LLM/GPT
- Deep/Machine Learning
- Blockchain
- Information Security
- Internet of Things
- Programming Languages

Trigger-Action Platform (TAP)

- Connecting otherwise unconnected services and devices
- **Trigger** event comes, app performs an **Action**

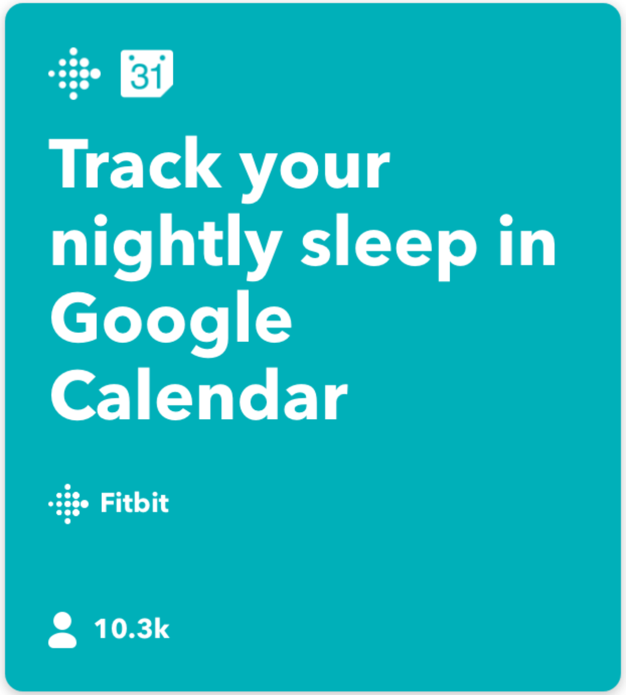
zapier





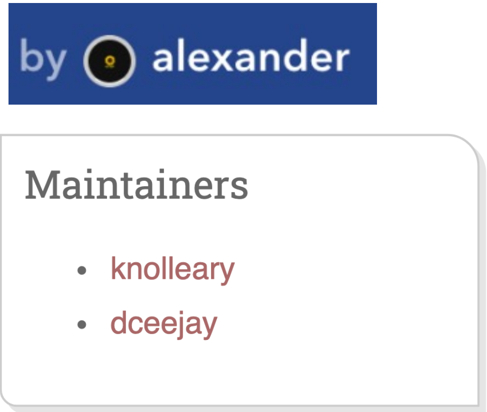
Node-RED

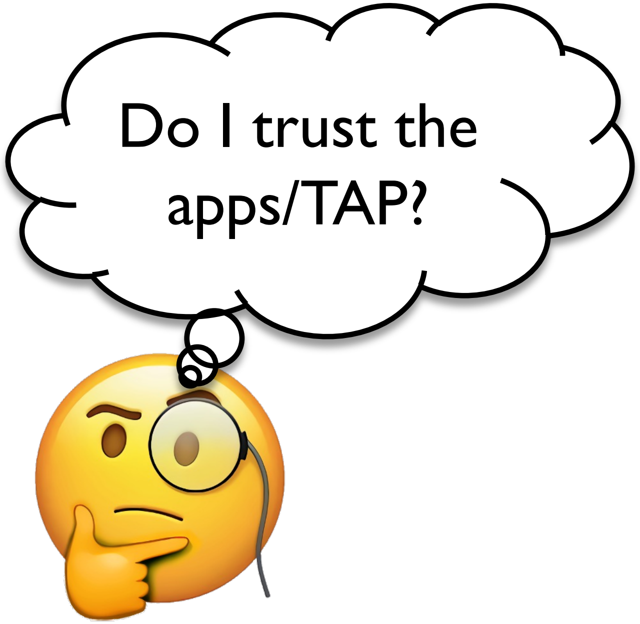
IFTTT



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)

IFTTT

>25M users

>1B apps per month

>800 partner services

Fixing the Dripping TAP

- Securing TAPs in the presence third-party apps

SandTrap: Securing JavaScript-driven Trigger-Action Platforms

Mohammad M. Ahmadpanah, Daniel Hedin, Musard Balliu, Eric Olsson, Andrei Sabelfeld

USENIX'21 (A*)

- Data privacy of TAP users

LazyTAP: On-Demand Data Minimization for Trigger-Action Applications

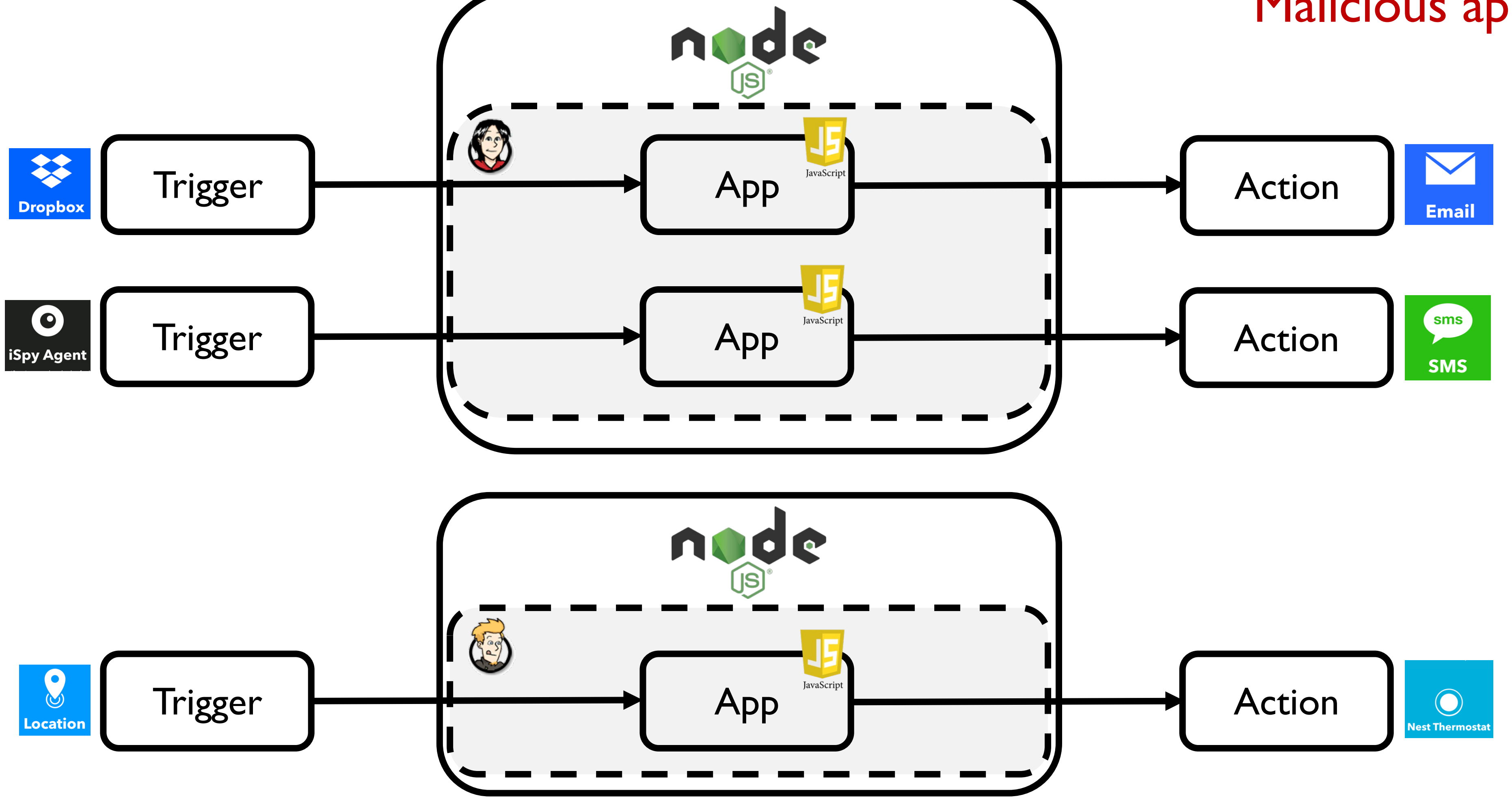
Mohammad M. Ahmadpanah, Daniel Hedin, Andrei Sabelfeld

S&P'23 (A*)

TAP architecture

Zapier and Node-RED:
single-tenant

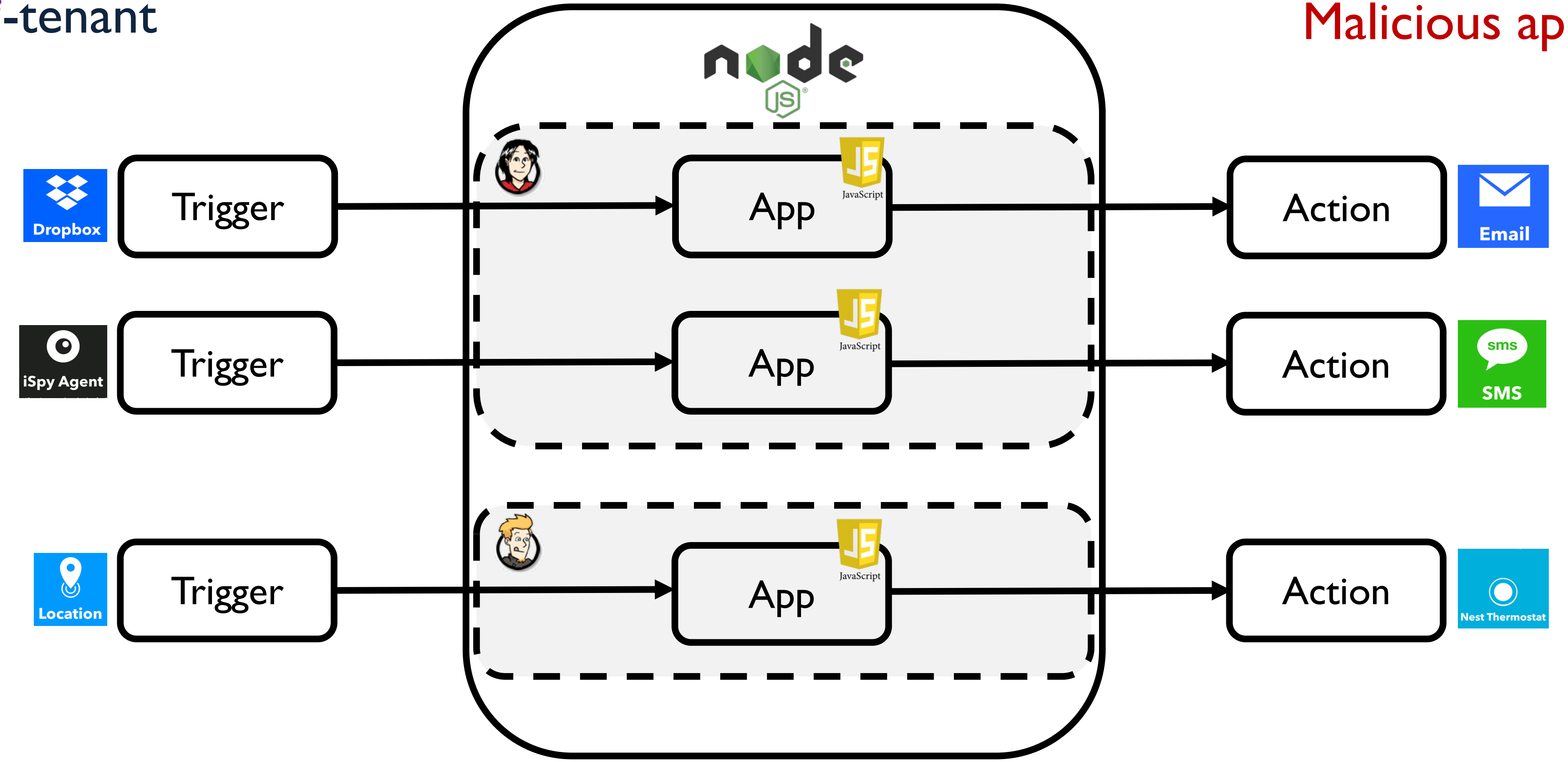
Threat model:
Malicious app maker



TAP architecture (cont.)

IFTTT:
multi-tenant

Threat model:
Malicious app maker



Sandboxing apps in IFTTT and Zapier

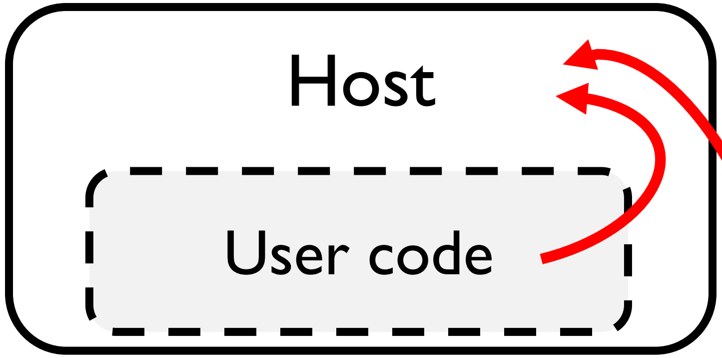
- JavaScript of the app runs inside AWS Lambda
- Node.js instances run in Amazon's version of Linux
- AWS Lambda's built-in sandbox at process level
- IFTTT: "App code is run in an **isolated** environment"

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

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

AWS Lambda

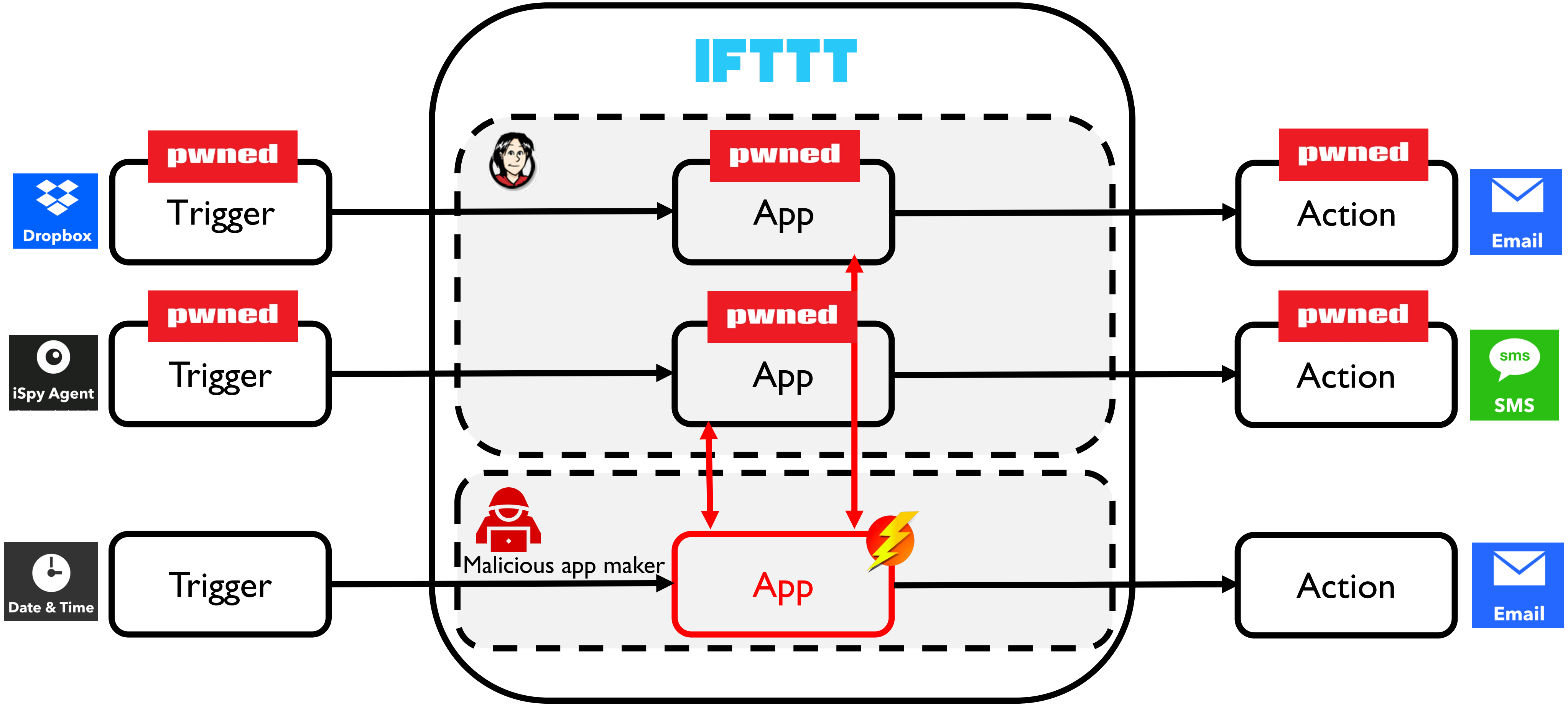
Sandbox breakout



- Using *prototype chain* in JS

```
function stack() {   new Error().stack;  stack();  }
try { stack(); }  catch (e) {
  e.constructor.constructor('return process')().mainModule
   .require('child_process').execSync('echo pwned!'); }
```

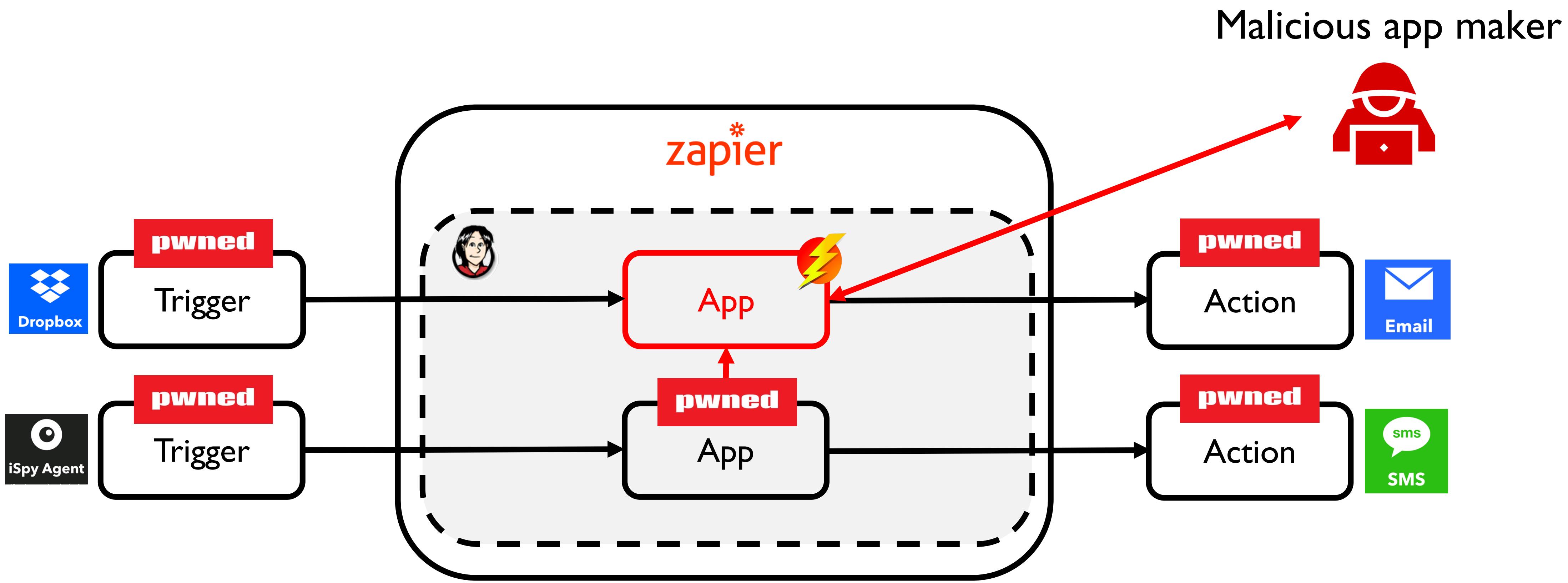
IFTTT sandbox breakout



User installs *benign* apps from the app store

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

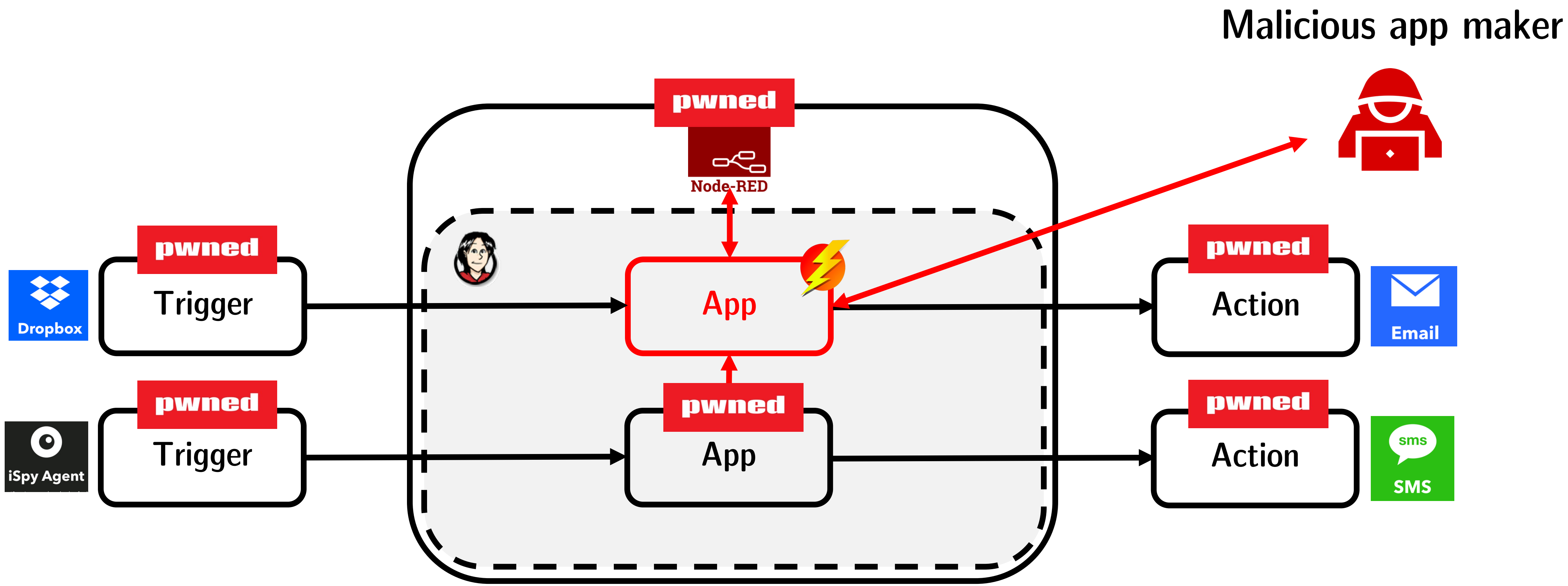
Zapier sandbox breakout



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 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***

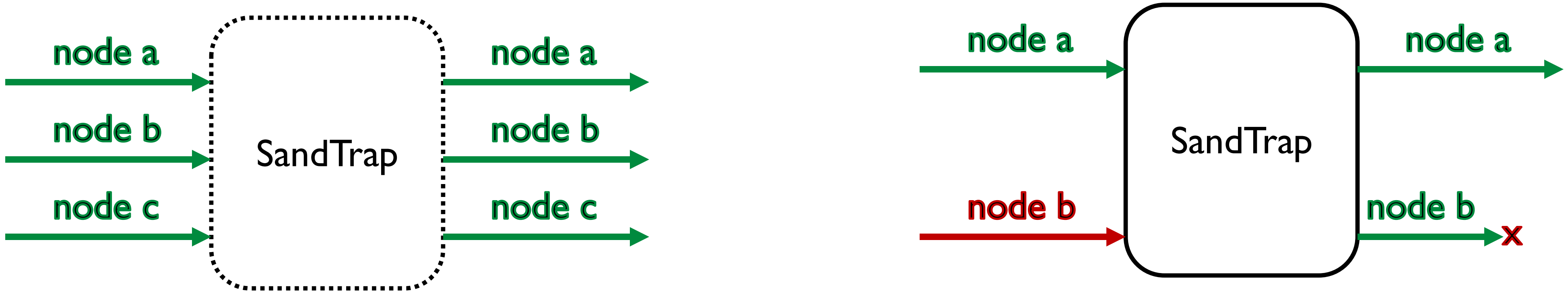
Need access control at **module-** and **context-**level

- 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**

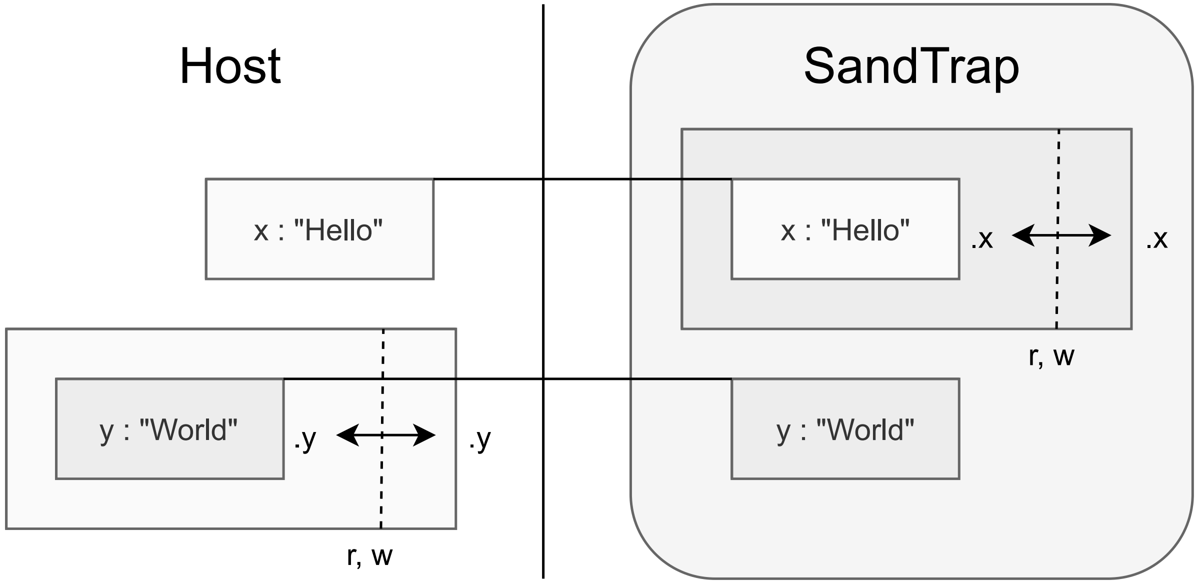
SandTrap: modeling

- 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



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"

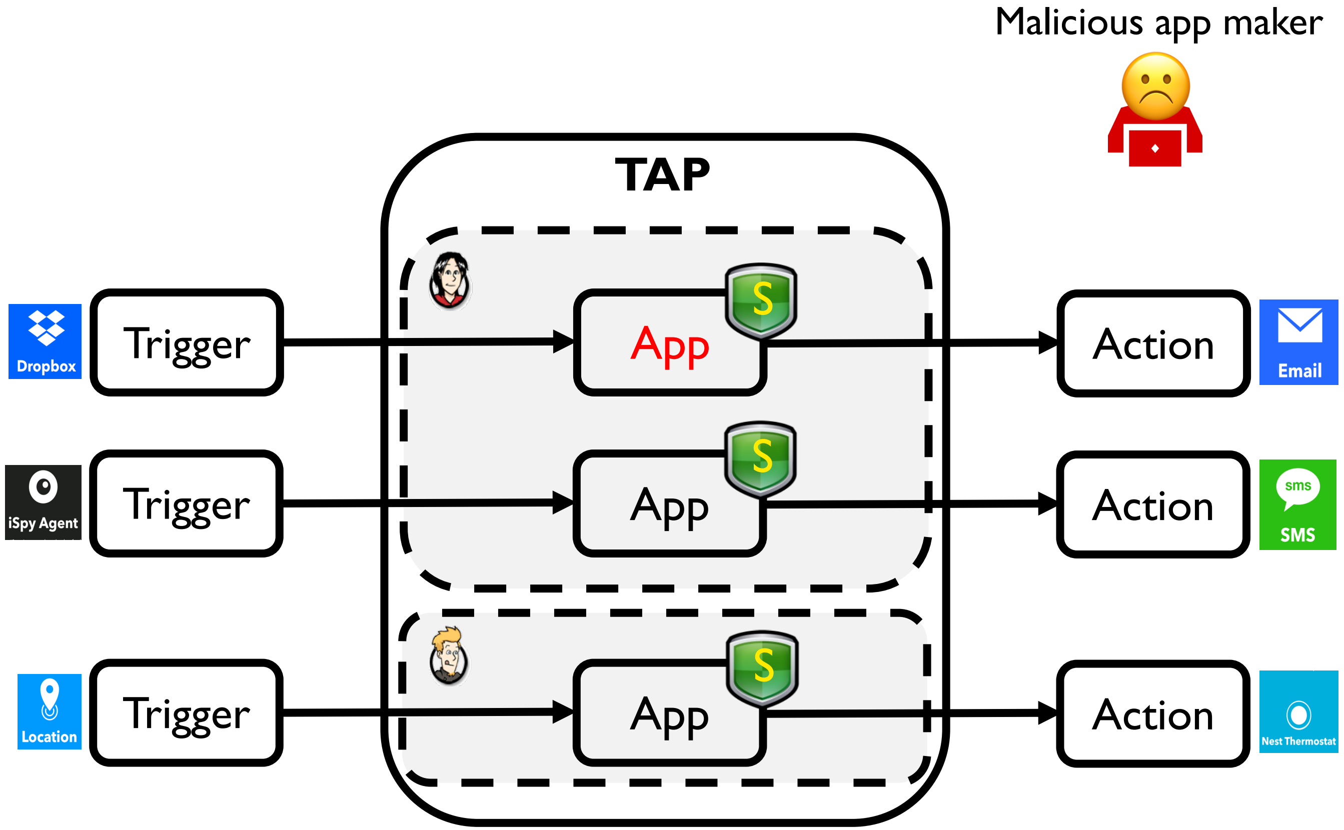
SandTrap benchmarking examples

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	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 monitor

– Structural proxy-based monitor to enforce fine-grained policies for JavaScript

– Formal framework (for a core language)
  - Soundness and transparency

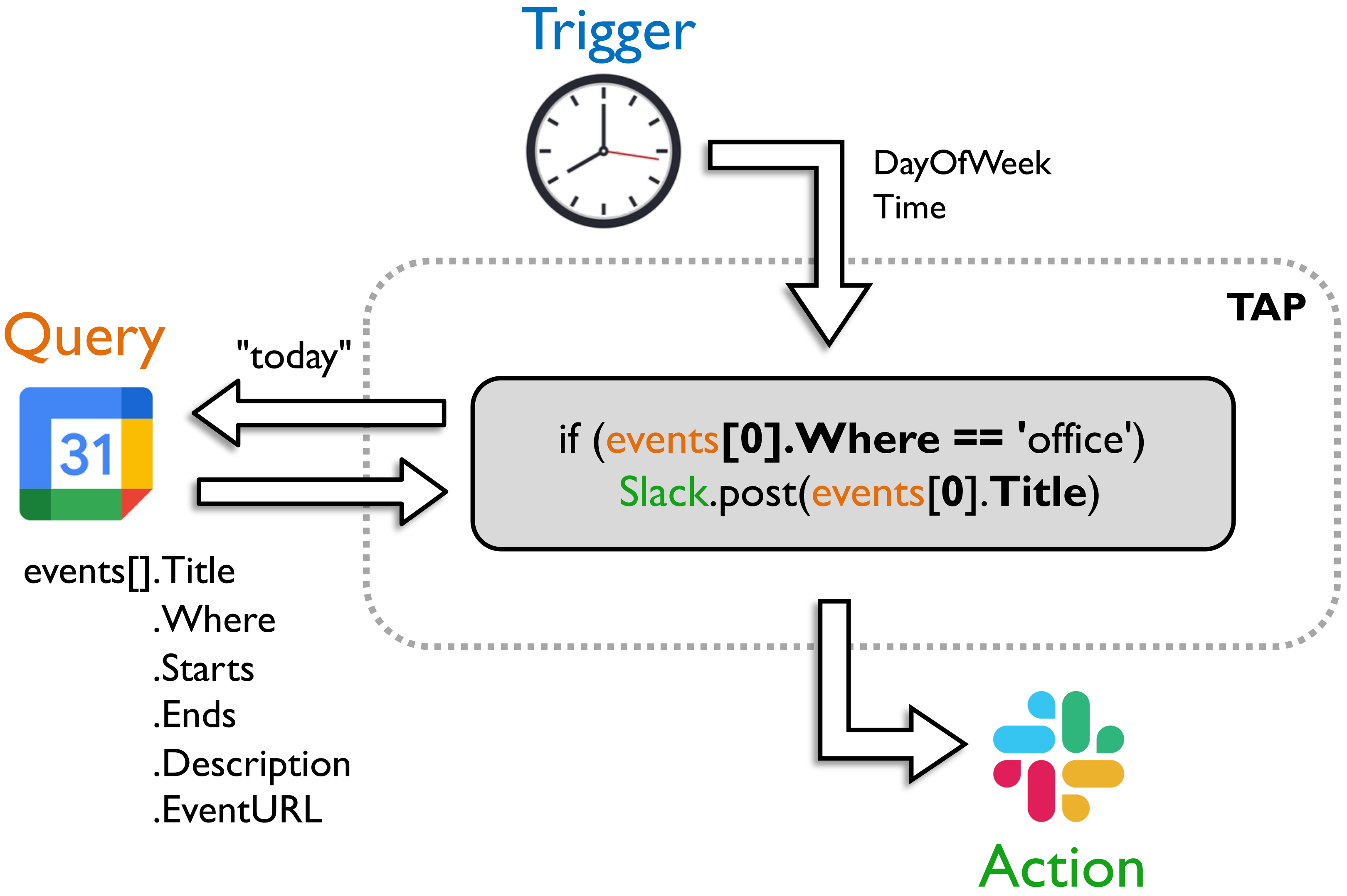


Try at https://github.com/sandtrap-monitor/sandtrap

TAPs with queries

- Additional data source with **Queries**
  - Recently introduced in IFTTT, allowing for complex apps
  - Accessing private data e.g., calendar events, watched movies, and locations

Push-all approach in TAPs



"Every morning, post the title of the first office meeting to Slack"

Push-all approach

All trigger/query data to TAP independent of the app code

at odds with ***data minimization***

Movie recommender

SmartThings

The movie title is picked **randomly** among user's recommended movies

```
let index = Math.floor((Math.random() * Trakt.recommendedMovies.length))
Notifications.setMessage(
    "Let's watch: " + Trakt.recommendedMovies[index].MovieTitle)
```

recommendedMovies[].MovieId
.MovieTitle
.MovieYear
.MovieDescription
...

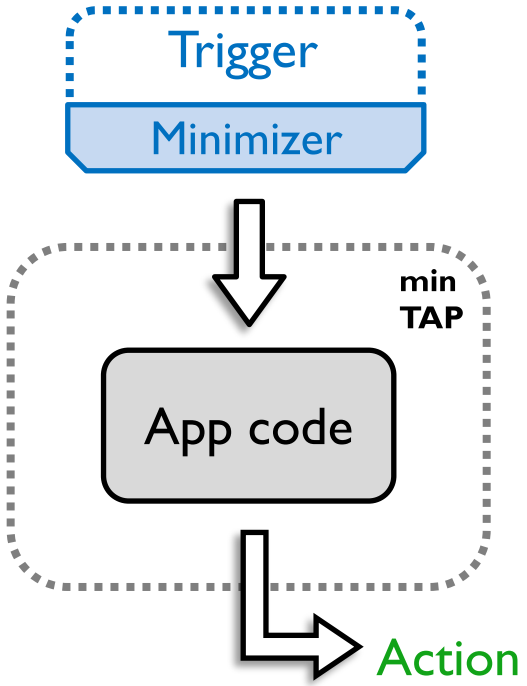
Data minimization

- "Only **necessary** data should be collected for the **specific purpose** the user consented"
- IFTTT's approach: Attribute-level **overprivilege**
  - **Push-all** approach
  - Input services should send (by default) the **50 most recent events**

Image: © https://www.cookieyes.com/blog/ccpa-vs-gdpr/

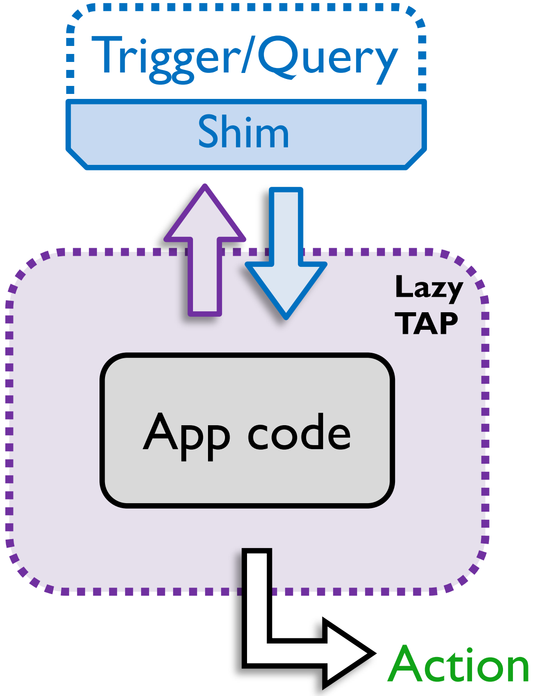
minTAP [USENIX'22]

- Minimization wrt ill-intended TAP
- **Preprocessing** approach
  - Minimizing attributes of **trigger** data
- Modes: **Static** and **Dynamic**
  - **Static**: All attributes in the app code
  - **Dynamic**: Pre-runs the app code on the service
- Trusted clients required
  - For minimization analysis and app integrity

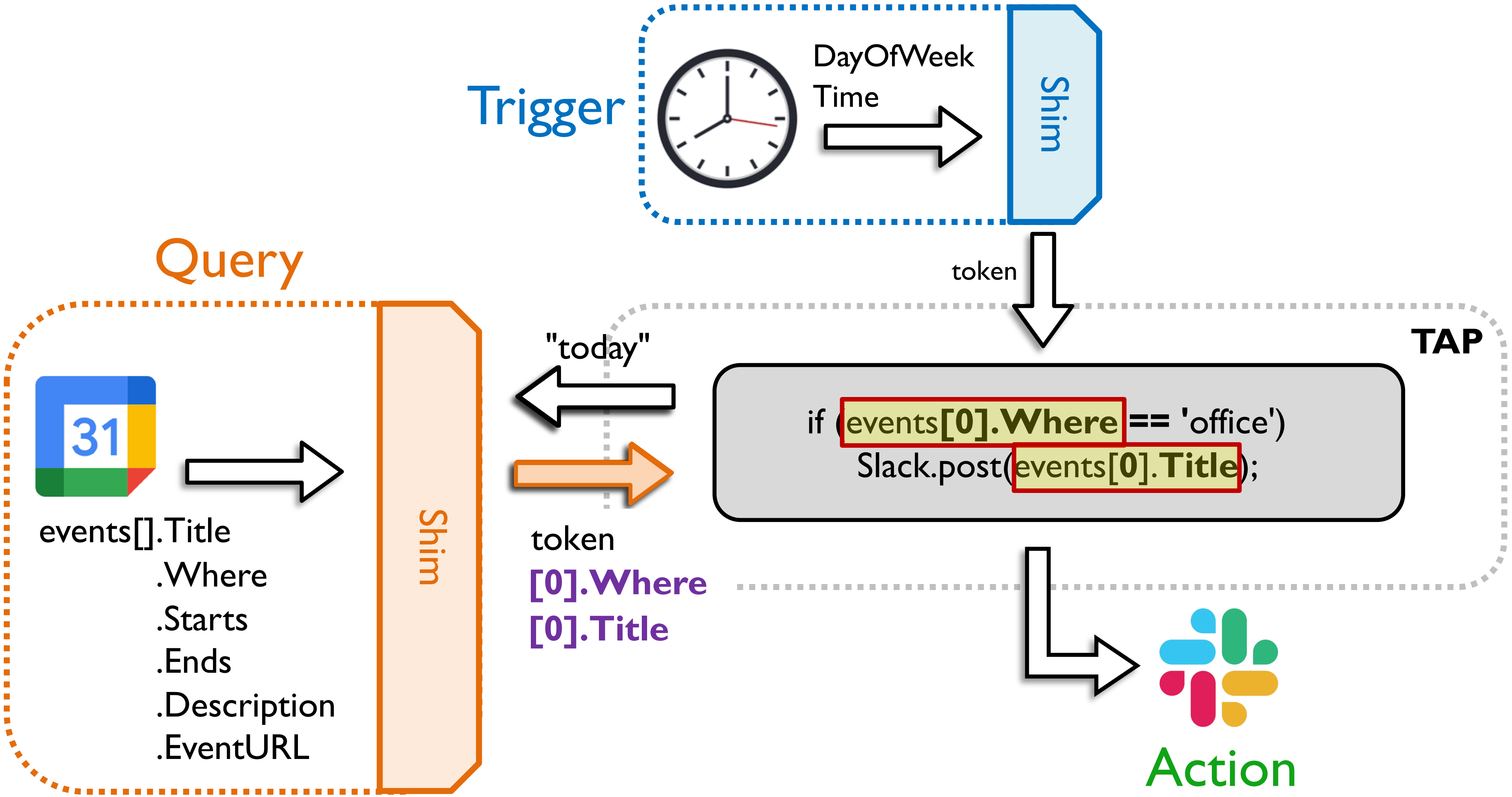


LazyTAP: data minimization by design

- Minimization wrt willing-to-minimize TAP
- **On-demand** approach
  - Pulling attributes of **trigger** and **query** data
  - Data source unification
- **Input-sensitive** and fine-grained
  - TAP: **Lazy runtime** supporting **fetch-on-access**
  - Trigger/Query services: **Shim** layers
    - Caching mechanism



LazyTAP: running example



Seamlessness for app developers

- App code remains as is
  - Using the same APIs
  - Supporting *nondeterminism* and *query chains*
- **Lazy runtime** for apps
  - **Remote proxied objects** for trigger and queries
  - Deferred query preparation and property access by **thunking**

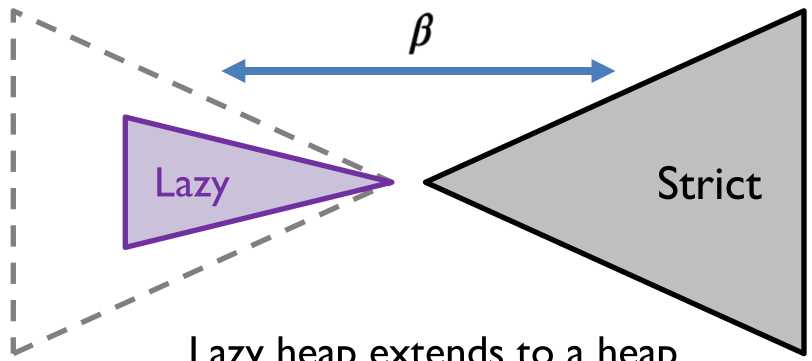
Modeling

- Core language: While language with objects

$$e ::= v \mid x \mid e \oplus e \mid f(e) \mid e[e] \mid \{\} \mid \boxed{T \mid Q(k, e) \mid A(m)} \mid \boxed{() \Rightarrow e}$$

- Modeling remote objects, lazy query, and deferred computation

Theorem: LazyTAP is correct and at least as precise as preprocessing minimization



Lazy heap extends to a heap isomorphic to strict heap

Evaluation

App Id	Distinctive pattern	Total attributes (IFTTT)	Static minTAP	**LazyTAP**
MeetNotif	Sensitive independent query	2 + (6 * CalendarLength)	2	1 | 2
MovieRec	Nondeterministic query, skip on time	3 + (7 * TraktLength)	TraktLength + 1	1
ParkFind	Conditional query chain, skip on queries	4 + (6 * CalendarLength) + (7 * YelpLength)	4	1 | 3 | 4

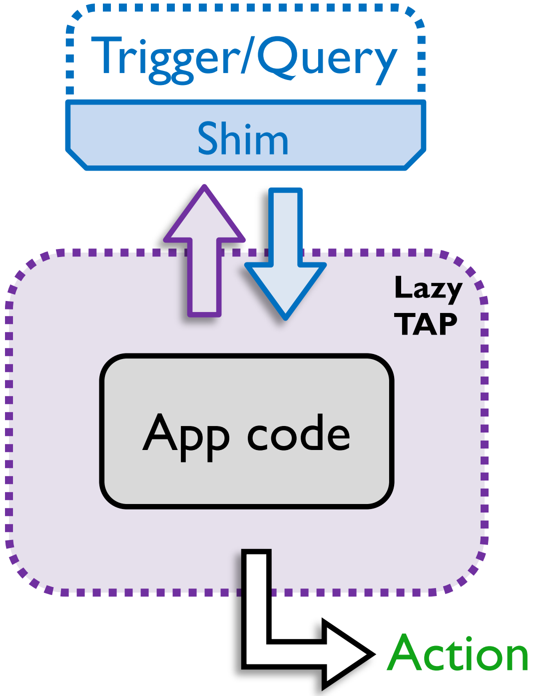
Minimization: 95% over IFTTT; 38% over static minTAP

LazyTAP in comparison

Approach	Minimization wrt	Minimization guarantees
IFTTT	None	**Push all**, no minimization guarantees
Static minTAP	Ill-intended TAP	**Input-unaware** minimization
Dynamic minTAP	Ill-intended TAP	**Input-sensitive** minimization No attributes when skip/timeout + **No support for queries**
LazyTAP	TAP willing to minimize	**Input-sensitive** minimization wrt **trigger and query** inputs (supporting *nondeterminism* and *query chains*)

LazyTAP takeaways

On-demand minimization by design:
- **Input-sensitive** and fine-grained
- Supporting **queries** and **nondeterminism**
- **Seamless** for app developers
- **Correctness** and **precision** formally proved
- Benchmarking:
  95% over IFTTT, **38%** over static minTAP

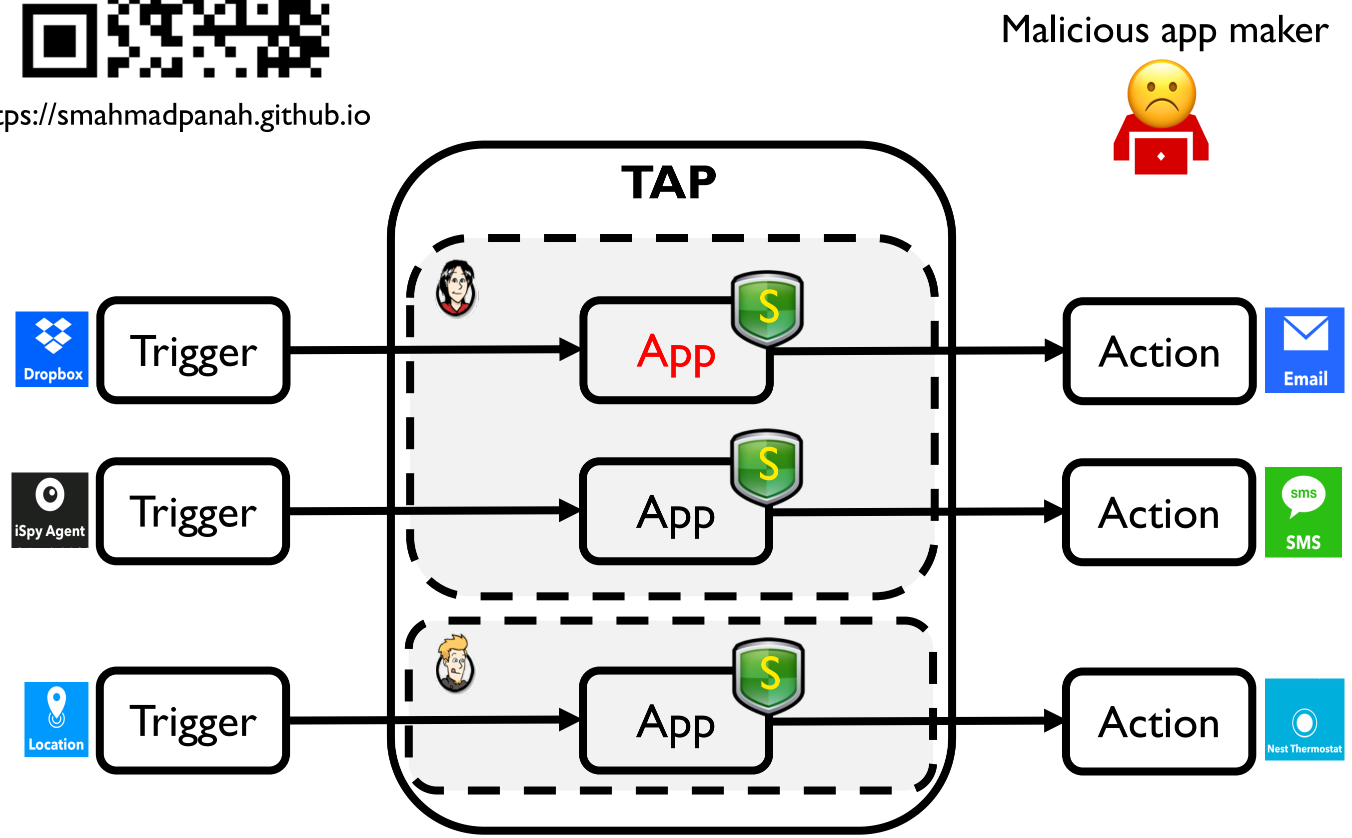


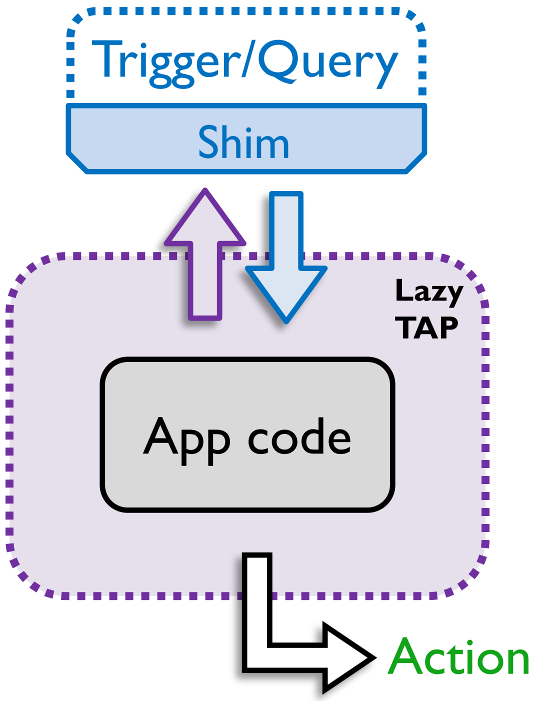
Lazy runtime by:
- Proxied **remote objects**
- Deferred computation by **thunking**

https://www.cse.chalmers.se/research/group/security/lazytap

The Dripping TAP Fixed!

https://smahmadpanah.github.io





Fine-grained access control enforcing isolation

On-demand data minimization