Fear and Logging in the Internet of Things

Qi Wang, Wajih Ul Hassan, Adam Bates, Carl Gunter

NDSS Symposium 2018
Feb 19, 2018
IDC: Internet Of Things To Be Valued $1.1 Trillion By 2021

December 10, 2017 - Written By Manny Reyes

The Internet of Things is expected to grow to a value of $1.1 trillion in 2021 based on consumer spending worldwide, according to a new report by the International Data Corporation, which also forecasts hardware to account for the biggest technology segment of IoT next year with a value of $239 billion, followed by services, software and connectivity. The report projects majority of spending will be on sensors, modules, security and infrastructure, with the manufacturing industry being projected to make the largest investment of $189 billion on IoT products and services in 2018 alone to support operations and asset management efforts. According to IDC’s forecast, overall spending on IoT is likely to hit $772.5 billion next year from the projected spending of $674 billion in 2017.
How to diagnose an incorrect behavior?

How to explain system behaviors?
IoT Logging

- Current logging mechanisms are device-centric
  - It is difficult to infer the causal dependencies between different events and data states.

  \[ \text{……} \]
  \textit{Motion was detected at 11:13 AM}

  \[ \text{……} \]
  \textit{Front door was unlocked at 11:13 AM}

  \[ \text{Light was turned on at 11:14 AM} \]

  Why the light was turned on?
Data Provenance

- Data provenance describes the *history of actions* taken on a data object from its creation up to the present.

……
Motion was detected at 11:13 AM
……
Front door was unlocked at 11:13 AM
Light was turned on at 11:14 AM

Light was turned on because motion was detected.
Our Framework: ProvThings

- A general framework for the **capture, management**, and **analysis** of data provenance in IoT platforms.
Provenance Collection

- ProvThings collects provenance metadata from different components in an IoT platform
  - IoT Apps
  - Device APIs (handlers)

- ProvThings uses automated program instrumentation to collect provenance metadata in a program
  - Minimally invasive to existing platforms
Instrumentation-based Collection

- ProvThings instruments IoT programs statically before a program is submitted for execution
  - Control flow and data flow analysis

- The instrumented code collects provenance metadata at runtime
- Data creations, data derivations and actions
preferences {
    input "lock", "capability.lock"
}
def installed() {
    subscribe(lock, "lock", eventHandler)
}
def eventHandler(evt){

    name = evt.name
    value = evt.value
    log.debug "Lock event: $name, $value"
    msg = "Lock event data:" + value
    httpPost("http://www.domain.com", msg)
}

After Instrumentation

```python
preferences {
    input "lock", "capability.lock"
}
def installed() {
    subscribe(lock, "lock", eventHandler)
}
def eventHandler(evt){
    def scope = [:]
    entryMethod(scope, "eventHandler", "evt", evt)
    def name = evt.name
    trackVarAssign(scope, "name", "evt")
    def value = evt.value
    trackVarAssign(scope, "value", "evt")
    log.debug "Lock event: $name, $value"
    trackCall(scope,"log.debug",["value", "name"], ["Lock event: $name, $value"])
    def msg = "Lock event data:" + value
    trackVarAssign(scope, "msg", "value")
    httpPost("http://www.domain.com", msg)
    trackSink(scope,"httpPost","msg", ["http://www.domain.com ",msg])
}
```
Selective Code Instrumentation

- To avoid collecting *unnecessary* provenance metadata, ProvThings performs *source-sink* based instrumentation
  - **Source**: a security sensitive data object, e.g., the state of a lock
  - **Sink**: a security sensitive method/action, e.g., the unlock command

![Program slicing diagram]
preferences {
    input "lock", "capability.lock"
}
def installed() {
    subscribe(lock, "lock", eventHandler)
}
def eventHandler(evt){

    def value = evt.value

    def msg = "Lock event data:" + value
    httpPost("http://www.domain.com", msg)
}

Provenance Management

• Process provenance metadata collected from different components
  • Aggregation, merging and filtering

• Convert metadata into a *unified* IoT provenance model and build provenance graphs

• Provide modular support to store provenance graphs into different storage backends
Provenance Analysis

• Query APIs
  • Forward and backward dependency analysis

• The policy engine allows specification of policies in the form of graph pattern
  • a sequence of activities

• The policy monitor queries the provenance graphs every time new records are added
  • Policy enforcement
Implementation

• We prototype ProvThings on Samsung SmartThings platform
Evaluation

• **Datasets**
  • SmartApps of 26 possible IoT attacks\(^1\) for effectiveness
  • 236 commodity SmartApps for performance

• **Effectiveness**: ProvThings was able to accurately reconstruct all the 26 tested IoT attacks!!

• **Instrumentation overhead**: 34 ms for SmartApps and 27 ms for a Device Handlers. (*Note: one time cost!*)

• **Storage overhead**: Just 260 KB of storage for daily use!!

\(^1\) ContexIoT, Jia et al. *NDSS’ 17*
• **End-to-end latency**
  • Tested on both virtual and physical devices
    • 20.6% latency on virtual devices
    • 4.5%~5.3% latency on physical devices

Breakdown of end-to-end event handling latency overhead
Case Study: Information Leakage

- **LockManager** is an app that updates or deletes lock pin codes.
- **FaceDoor** is an app that unlocks a door via face recognition using the front door camera.
Investigation

- **LockManager**
  - **subtype: INPUT_USER**
  - **FunctionName: updatePincode**
  - **CommandName: setCode**
  - **Event: codeReport**
    - **value: 'pincode:1234'**
    - **FunctionName: spyHandler**
  - **name: state.data**
    - **subtype: memory**
    - **FunctionName: httpPost**
      - **uri: 'http://attacker.appspot.com'**

- **FaceDoor**
  - **FunctionName: installed**
  - **FunctionName: initialize**
  - **FunctionName: schedule**
    - **arg: '0 0 0 * * ?'**
  - **FunctionName: sendData**
  - **subtype: EVENT_TIMER**

**Backward analysis**
Summary

• ProvThings is a general and practical framework for the capture, management and analysis of data provenance in IoT

• ProvThings is a first step towards providing solutions for different IoT stakeholders
  • System diagnosis, debugging, monitoring, investigation and access control
Questions?

Thank you for your time!

qiwang11@illinois.edu
Backup slides
Consumer Application

- For typical consumers, we provide *WhyThis?* to explain unseen sequence of activities and allows them to allow or deny such activities.

The open command of Window is informed by SmokeMonitor which used a smoke detected event generated by SmartLight.
Threat Model & Assumptions

- Malicious API-level attacks
  - Malicious apps
  - Device vulnerability
  - Proximity
- Accidental app misconfigurations

- Assumptions
  - We assume the devices are not compromised
  - We assume the entity responsible for executing the IoT’s central management logic is not compromised
    - SmartThings cloud
User Scenarios

• ProvThings provides different frontends to meet the needs of a variety of stakeholders in the IoT ecosystem
  • Professionals could use the query APIs to investigate abnormal behaviors in their customers’ homes

• Techies could use the policy engine to create customized policies for their smart homes

• Typical users could use the consumer app to understand and react to peculiar events that happening in their smart homes
Policy Engine

- Policy format

```plaintext
pattern: {
    check: exist | not exist
    action: notify | allow | deny
}
```

- Policy example

```plaintext
pattern:
    MATCH (a:DEVICE_CMD {name: "setCode"}) WasOriginatedFrom
        (b:INPUT_HTTP {name: "HTTP Request"}),
        (c:DEVICE {name: "Front Door Lock"})
    WHERE a.agentid = c.id
    RETURN a
}
check: exist
action: notify
```
# IoT Provenance Model

<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
<th>PORV Model</th>
<th>Subtype</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>An application in a IoT platform</td>
<td>Agent</td>
<td>APP</td>
</tr>
<tr>
<td>Device</td>
<td>A smart device in a platform.</td>
<td>Agent</td>
<td>DEVICE</td>
</tr>
<tr>
<td>Device command</td>
<td>A action supported by a device.</td>
<td>Activity</td>
<td>DEVICE_CMD</td>
</tr>
<tr>
<td>Device event</td>
<td>An object that represents a state change on a device.</td>
<td>Entity</td>
<td>EVENTDEVICE</td>
</tr>
</tbody>
</table>
Policy for the example

```plaintext
pattern: {
    MATCH (a:SINK)-[:Used]->(b:Entity),
    (c:APP_IOT {name:"FaceDoor"})
    WHERE a.agentid=c.id and
        (a.uri<>"http://trust.me" || b.taint <> "ImageCapture")
    RETURN a
}

check: exist
action: notify
```
Fake Device Events

- **SmokeMonitor** is an app which will open the Window and sound the Alarm if smoke is detected by the Smoke Detector.

- **SmartLight** is a malicious app which will raise fake physical device events for Smoke Detector.
Remote Command

- **WhenEveryoneIsAway** is an app that sets the mode of a home to Away when everyone has left home.

- **LockItWhenILeave** is an app that locks the door when the mode is set to Away. However, when installed, the app will query a malicious domain to get an attack command and time. It waits until everyone is away to execute the attack command.
IoT Platforms

IoT APP

Device Abstraction

Heterogeneous Devices
preferences {
  input "lock", "capability.lock"
}
def installed() {
  subscribe(lock, "lock", eventHandler)
}
def eventHandler(evt){
  def name = evt.name
  def value = evt.value
  log.debug "Lock event: $name, $value"
  def msg = "Lock event data:" + value
  httpPost("http://www.domain.com", msg)
}