Two Perspectives on IoT Security at the Edge: Standards and Runtime Enforcement

Grace A. Lewis
Why is IoT Security a Big Deal?

Largest DDoS attack ever launched on service provider Dyn using an IoT botnet — Mirai searched the internet for vulnerable IoT devices

St. Jude Medical’s implantable cardiac devices had vulnerabilities — hacker could deplete the battery or administer incorrect pacing or shocks

Owlet WiFi baby heart monitor — ad-hoc Wi-Fi network linking the base station to the sensor device was unencrypted and did not require any authentication to access

TrendNet Webcam let anyone who obtained a camera’s IP address look through it

Team of researchers took total control of a Jeep SUV using the vehicle’s CAN bus

“Insecure and highly vulnerable IoT devices can leave an indeterminate number of critical systems and data around the world at risk.”

Source: https://www.iotforall.com/5-worst-iot-hacking-vulnerabilities/
IoT Security Challenges

Not all IoT device manufacturers follow secure development practices

Not all IoT devices allow software updates → security software is obsolete

The current IoT device market is very dynamic → no security patches

Lack of IoT security standards

Additional challenges at the edge

• Disconnected, intermittent, and limited network connectivity

• Threats of sabotage, capture, and impersonation
The Standards Perspective: Authentication and Authorization for IoT Devices in Edge Environments
IoT Security at the Edge

IoT devices are increasingly being used in disadvantaged edge environments

However ... most existing IoT security approaches are targeted at home and industrial environments with a very different threat model

Solution: Evaluate and Extend Authentication and Authorization for Constrained Environments (ACE) for use in edge environments

- IETF draft standard based on OAuth 2.0
- Adapts OAuth to limitations of IoT devices (e.g., no user intervention, limited resources)
Authentication and Authorization for Constrained Environments Framework (ACE)

Client (C) → Authorization Server (AS) → Resource Server (RS)

1. Token Request
2. Access Token + Access Information
3. Token + Request
4. Protected Resource

Uses alternative technologies that require less resources:
- HTTP → CoAP
- JSON → CBOR
- JOSE → COSE

Eliminates need for consistent online access to an AS through self-contained, proof-of-possession (PoP) tokens associated to cryptographic keys.

Tokens can only be decrypted by the RS.

PoP tokens allow clients to prove a token was issued to them.

Identified limitations:
- Bootstrapping is intentionally out of scope
- Token revocation is not addressed (other than time-based expiration)
Extensions to ACE – Bootstrapping

Bootstrapping

• Grant clients and RSs access to AS services
• Exchange credentials to secure communications and tokens

Assumptions

• AS in the field may be disconnected from the enterprise
• IoT devices (RS) may have no input/output mechanism

Approach: Use credentials in a printed QR code associated to IoT devices to link them through a pairing procedure
Extensions to ACE – Bootstrapping (Pairing Process)
Extensions to ACE – Token Revocation

If a Client or RS is compromised

- Clients will have access to resources until their token expires
- RS can report fake information to clients with valid tokens for them

Short expiration times are not appropriate as Client or RS may be out of range of AS for long periods of time

Approach: Opportunistic introspection requests from Clients and RSs to AS

- Whenever they are in range of each other
Extensions to ACE – Token Revocation

DTLS Session

C-ID

Loop

For All Active Tokens

Introspection Request (Encrypted Token)

RS-ID = GetKeyID(Encrypted Token)

PSK_{RS} = GetKey(RS-ID)

Token = Decrypt(Encrypted Token, PSK_{RS})

Validate Token (Token)

If Invalid

Remove Token (Encrypted Token)
High-Level Architecture

Client (Ubuntu)
- IoT Client
  - 6lbr
  - Credentials File
  - Token File

Authorization Host (Ubuntu)
- Authorization Server
  - 6lbr
  - ACE AS DB (PostgreSQL)

Legend
- System Boundary
- Custom Runtime Component
- Read/Write
- 3rd Party Runtime Component
- CoAP over DTLS
- CoAP without DTLS

IoT Device (Contiki)
- Resource Server
  - Token File
  - Credentials File

Open Questions
- How to pair IoT Client to Authorization Server?
- Define real scenarios
- How to pair/register Resource Server to Auth Server?
- What credentials to use?

Decisions:
- Not use introspection
- Token: CWT with COSE, containing public key of client
- CWT already has scopes in token
- Auth Code Grant type (Oauth 2)
- Profile: mutual authentication?
- Use DTLS
- Use as much of ACE Constrained as it is possible
- CoAP without DTLS
Threat Modeling using STRIDE – Pairing Procedure and Resource Access

## Threat Model – Pairing Procedure

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Mitigation</th>
</tr>
</thead>
</table>
| **Spoofing**      | Node impersonation                | • RS (IoT device) can only be paired once without rebooting  
                   |                                                  | • Rebooting and access to QR code should be physically protected                                     |
| **Information Disclosure** | Data flow sniffing  | • PSK in QR code is used to encrypt communication using DTLS                                           |
| **Tampering**     | Data flow tampering               | • PSK in QR code is used to encrypt communication using DTLS                                           |
| **Information Disclosure** | Unauthorized credential store access | • In RS, no stored credentials exist before pairing procedure  
                   |                                                  | • QR code should be physically protected from view in the field                                     |
| **Elevation of Privilege** | Execution of unauthorized code | • Pairing protocol is simple and sanitizes input to avoid arbitrary code execution                   |
### Threat Model – Resource Access

<table>
<thead>
<tr>
<th>Category</th>
<th>Threat</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Node impersonation</td>
<td>• ACE is used along with DTLS and PoP keys to ensure authentication&lt;br&gt;• Token revocation can be used if node has been compromised</td>
</tr>
<tr>
<td>Information disclosure</td>
<td>Data flow sniffing</td>
<td>• DTLS connections provide integrity protection</td>
</tr>
<tr>
<td>Tampering</td>
<td>Data flow tampering</td>
<td>• DTLS connections provide traffic encryption</td>
</tr>
<tr>
<td>Information disclosure</td>
<td>Unauthorized credential store access</td>
<td>• AS stores can be encrypted&lt;br&gt;• RS stores would need hardware solutions to be better protected</td>
</tr>
<tr>
<td>Elevation of privilege</td>
<td>Execution of unauthorized code</td>
<td>• Resource access protocol is simple and sanitizes input to avoid arbitrary code execution</td>
</tr>
<tr>
<td>Elevation of privilege</td>
<td>Elevation using impersonation</td>
<td>• Because OAuth assumes AS is trusted, there is no good mitigation for AS issuing tokens to itself.&lt;br&gt;• Tokens are separate for each RS, to avoid that a compromised RS impersonates as a client to other RSs</td>
</tr>
</tbody>
</table>
Vulnerability Analysis – Attack Graph
### Resource Consumption

<table>
<thead>
<tr>
<th>Platform</th>
<th>Min Binary (bytes)</th>
<th>ACE Binary (bytes)</th>
<th>Increase (%)</th>
<th>ACE RAM Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raspberry Pi</td>
<td>995,783</td>
<td>1,452,153</td>
<td>45.83</td>
<td>2.3-3.5 MB</td>
</tr>
<tr>
<td>TI CC2538</td>
<td>286,660</td>
<td>354,432</td>
<td>23.64</td>
<td>&lt; 32 KB</td>
</tr>
</tbody>
</table>

Evaluated the constrained RS implementation against the constraints of a Class 2 device (~50KB for data and ~250KB for code)

Size of a sample unencrypted CBOR Web Token (CWT) = 74 bytes
Size of a sample COSE encrypted CWT = 113 bytes

Pairing request using DTLS = ~300 ms
Resource access request posting a CWT (no DTLS, but includes decryption) = ~700 ms
Summary and Next Steps

Code available on Github

Next Steps
  • Creating a mesh of devices over an IEEE 802.15.4 wireless network and adding routing between peers to provide better guarantees for periodic introspection requests and better understand resiliency and scalability
  • Further optimizing the constrained RS implementation for a Class 1 device
However …

Although appropriate and relevant, using ACE for IoT security requires the fabrication of ACE-compliant IoT devices.

As is common with standards

• There is rarely a single standard
• Many standards come and go
• Often not enough financial and technical incentives for manufacturers to commit to a standard
The term "KalKi" is of Sanskrit origin and derived from the Sanskrit word "Kala," which means destroyer of filth or malice and bringer of purity, truth and trust.
Motivation

Problem
Integration of IoT devices into systems with high security requirements is difficult due to issues such as reported vulnerabilities and untrusted supply chains.

At the same time, organizations recognize the rapid pace at which the IoT commercial marketplace is evolving and its urgency to embrace commodity technologies to “keep up with the market”

Solution
Move part of security enforcement to the network to enable the integration of IoT devices into DoD systems, even if the IoT devices are not fully trusted or configurable, by creating an IoT security infrastructure that is provably resilient to a collection of prescribed threats.
The “Software-Defined” Aspect

Use software-defined networking (SDN) and network function virtualization (NFV) to create a highly dynamic IoT security framework

1. Each IoT device, D, senses/controls a set of environment variables, EV.
2. Network traffic to/from each device is tunneled through μmboxes that implement the desired network defense for the device’s current security state
   - μmbox[SS₁] = Firewall
   - μmbox[SS₂] = IPS, ...
3. IoT controller maintains a shared statespace composed of {EV} and security state (SS) for each device
   - SS = {Normal, Suspicious, Attack}
4. Changes in the shared statespace are evaluated by policies and may result in the deployment of new μmboxes.
The “High Assurance” Aspect

Use überSpark (a framework for building secure software stacks) to incrementally develop and verify security properties of elements of the software-defined IoT security infrastructure.

**Control Node Properties**
- Policy data integrity
- µmbox image storage integrity

**Device Node Properties**
- Attestation
- Authenticated channel of communication with the IoT Controller

**Data Node Properties**
- Isolation between µmboxes of different trust levels: trusted, untrusted, and verified
- µmbox deploy-time integrity
KalKi Architecture
Current Prototype Architecture
IoT Data Policy Integrity via Data Capsules

FUNCy Views (Secure) System Architecture: Hardware-assisted, Low-latency, Low-TCB, Legacy Code Compartmentalization on x86 platforms

IoT policy data capsule guarantees that
- policy data can only be modified by prescribed policy interfaces
- policy interfaces only available to authorized clients
Policy Model

A policy is
- the set of conditions that indicate a change in the security state of an IoT device, and
- the set of actions taken when the conditions are met.
Dashboard

Main functionality
- IoT device configuration
- System status visualization
- Visualization of system-level property guarantees
# Threat Model

<table>
<thead>
<tr>
<th>#</th>
<th>Threat</th>
<th>In Scope?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Attacker finds a way to deploy the wrong µmbox for an IoT device given its security posture at a point in time</td>
<td>Y</td>
</tr>
<tr>
<td>2</td>
<td>Attacker loads malicious firmware/software on the IoT device</td>
<td>Y</td>
</tr>
<tr>
<td>3</td>
<td>Attacker finds a way to circumvent µmbox</td>
<td>Y</td>
</tr>
<tr>
<td>4</td>
<td>Attack from a µmbox to another</td>
<td>Y</td>
</tr>
<tr>
<td>5</td>
<td>Attacker compromises software running inside a µmbox</td>
<td>Y</td>
</tr>
<tr>
<td>6</td>
<td>Attacker identifies combination of inputs that can cause the FSM (internal policy representation) to lead to undesirable results</td>
<td>N</td>
</tr>
<tr>
<td>7</td>
<td>Attacker compromises communication between µmbox and IoT device</td>
<td>Y</td>
</tr>
<tr>
<td>8</td>
<td>Attacker compromises communication between µmbox and IoT controller</td>
<td>Y</td>
</tr>
</tbody>
</table>
What’s Next

Extend architecture and prototype of the IoT Security Framework

Implement additional security properties

Add features for easier µmbox creation and IoT device integration

Participate in exercises

Transition IoT Security Framework and lessons learned
Contact Information

Grace A. Lewis
Principal Researcher
Email: glewis@sei.cmu.edu
http://www.sei.cmu.edu/staff/glewis

Team Members (AAIoT)
Grace A. Lewis (PI)
Sebastián Echeverría
Dan Klinedinst (now at ShieldAI)
Ludwig Seitz (SICS)

Team Members (KalKi)
Grace A. Lewis (PI)
Sebastián Echeverría
Chris Grabowski
Craig Mazzotta
Matt McCormack (CMU/ECE)
Vyas Sekar (CMU/ECE)
Amit Vasudevan
Backup Slides
Edge Computing

Idea is to push applications, data and computing power to the edge of the Internet, in close proximity to mobile devices, sensors, and end users.

An early example is Akamai with servers around the world to distribute web site content from locations close to the user.
Cyber-Foraging

Mechanism by which mobile devices can discover and use edge servers for

- Computation Offload
  - Offload of expensive computation to edge servers in order to increase computational capability, reduce latency and bandwidth usage, and extend battery life

- Data Staging
  - Improve data transfers between mobile devices and the cloud by temporarily staging data in transit on edge servers
Tactical Cloudlets

Forward-deployed, discoverable, virtual machine (VM) based computing nodes that can be hosted on vehicles or other platforms to provide

- infrastructure to offload computation
- forward-data-staging for a mission
- data filtering to remove unnecessary data from streams intended for mobile users
- collection points for data heading for enterprise repositories
Tactical Cloudlet Features

- **Standard Packaging of Service VMs** to support ease of provisioning, scalability, and reuse.
- **Pre-Provisioned Cloudlets with App Store** to support disconnected operations.
- **Cloudlet Management Component** provides console-like capabilities and credential management.
- **Secure Key Generation and Exchange** for establishing trust between cloudlets and mobile clients.
- **Service VM Migration** enables mobility and reintegration to a fixed infrastructure.
Tactical Cloudlets: Authentication and Authorization

1. Bootstrapping
   - Generation of Cloudlet Credentials using IBE (Identity-Based Encryption)
   - Setup of RADIUS Server with Cloudlet Credentials
2. User connects mobile device to the cloudlet, and upon visual confirmation the admin starts the pairing process
3. Mobile Device connects to router, validates server credentials, and authenticates with RADIUS server
   - Wi-Fi Authentication
     - RADIUS Server implements Wi-Fi WPA2-Enterprise 802.1X EAP-TTLS with PAP
     - Device receives cloudlet credentials and validates
     - Devices sends its credentials for validation
4. Communication between the mobile device and the cloudlet is encrypted at the transport and message level
   - API Requests
     - Device exchanges encrypted messages with the cloudlet
     - Each exchange is validated against authorized device list

Device Credential Revocation
- Automatic due to timeout: Bootstrapping requires setting up mission duration
- Manual due to known loss or compromise: Cloudlet Manager component has revocation option
Extend Security to the Internet of Things (IoT) at the Edge

Problem

Integrating IoT devices into edge systems expands the attack surface of the system. Most existing IoT security approaches are targeted at home and industrial environments with a very different threat model.
## Extensions to ACE – Bootstrapping Alternatives

<table>
<thead>
<tr>
<th>When</th>
<th>Details</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>At manufacturing time</td>
<td>QR code has public key, private key embedded in hardware</td>
<td>Good for addressing device impersonation</td>
<td>IoT devices needs to support hardware-based credentials</td>
</tr>
<tr>
<td>Before deployment</td>
<td>QR code has credentials read by AS before deployment to the field</td>
<td>- Out-of-the-box deployment in the field</td>
<td>Loss of flexibility (more devices cannot be added in the field)</td>
</tr>
<tr>
<td>During deployment (fixed PSK)</td>
<td>QR code has PSK read by AS in the field and used for secure ACE connections</td>
<td>Pairing can be done in the field</td>
<td>- ACE PSK is available to anyone with access to QR code,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PSK remains constant between deployments</td>
<td>- PSK remains constant between deployments</td>
</tr>
<tr>
<td>During deployment (temporary PSK)</td>
<td>QR code has PSK used to secure exchange of second PSK, used for secure ACE connections</td>
<td>- Pairing can be done in the field</td>
<td>IoT device needs to support storing new credentials</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ACE PSK is different for each deployment</td>
<td></td>
</tr>
</tbody>
</table>
Architecture – Code and Libraries

ACE Authorization Server (Java)

Server

ACE-Java

COSE-Java

Scandium (DTLS)

CBOR-Java

Californium (COAP)

Java JRE

External Libraries

Legend

System or Subsystem Boundary
Custom Module
3rd Party Library

ACE Resource Server

RS Server and COSE parsing (C)

TinyDTLS (C)

Cn-chor (C)

Erbium REST COAP (C)

Contiki OS (6br fork)

External Libraries

Legend

System or Subsystem Boundary
Custom Module
3rd Party Library
Architecture – Device and Networks

Client (Laptop)
- IoT Client
  - Wi-Fi NIC
  - 802.15.4 RF Dongle (CC2531EMK)

Authorization Host (PC)
- Ethernet NIC
- Authorization Server
  - USB Port

IoT Device (CC2538EM)
- 802.15.4 NIC
- Resource Server

Legend:
- Hardware Device
- Custom Runtime Component
- Hardware Component

Connectors:
- Ethernet
- WiFi
- 802.15.4
- USB
- IPv6
## Classes of Constrained Devices

<table>
<thead>
<tr>
<th>Name, C</th>
<th>data size (e.g., RAM)</th>
<th>code size (e.g., Flash)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 0, C0</td>
<td>&lt;&lt; 10 KiB</td>
<td>&lt;&lt; 100 KiB</td>
</tr>
<tr>
<td>Class 1, C1</td>
<td>~ 10 KiB</td>
<td>~ 100 KiB</td>
</tr>
<tr>
<td>Class 2, C2</td>
<td>~ 50 KiB</td>
<td>~ 250 KiB</td>
</tr>
</tbody>
</table>

Table 1: Classes of Constrained Devices (KiB = 1024 bytes)

Source: IETF RFC-7228: Terminology for Constrained-Node Networks