How We Learned to Stop Worrying and Love Middleboxes

Keith Winstein

Stanford Computer Science Department
https://cs.stanford.edu/~keithw
How We Learned to Stop Worrying and Love Middleboxes Smart IoT Gateways

Keith Winstein

Stanford Computer Science Department
https://cs.stanford.edu/~keithw
Most communications applications are built on **one venerable abstraction**:
<table>
<thead>
<tr>
<th>Venerable abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>An end-to-end inviolable channel</strong> between two endpoints</td>
</tr>
</tbody>
</table>
Venerable abstraction

An end-to-end inviolable channel between two endpoints

- Bluetooth API provides it (app to device)
Venerable abstraction

An end-to-end inviolable channel between two endpoints

- Bluetooth API provides it (app to device)
- TCP provides it (app to app)
Venerable abstraction

An **end-to-end inviolable** channel between two endpoints

- Bluetooth API **provides it** (app to device)
- TCP **provides it** (app to app)
- TLS **enforces it** (no more Web caches)
A venerable abstraction

An *end-to-end inviolable* channel between two endpoints

- Bluetooth API *provides it* (app to device)
- TCP *provides it* (app to app)
- TLS *enforces it* (no more Web caches)
- QUIC and Mosh *enforce it even for control information* (no more accelerators)
The Internet owes much of its success to the view that the network should avoid meddling in endpoints’ affairs.

Traditional view:
“The endpoints are the principals.”

Here are five ways the Internet of Things can benefit from smarter gateways:
Idea #1: multiplexing and access control for Bluetooth

Challenges with Bluetooth API:

- Only one app can open stream to device
  - Can’t make: “is any device low on batteries?” app
- Access is all-or-nothing
- App must run on the gateway

Solution: Beetle, an OS Service for BLE

**Sharing**
Apps can share access to peripherals.

**Access Control**
Users can specify access policies on peripherals and apps.

**Communication flexibility**
Communication between peripherals, gateway and cloud apps.

**Backwards compatible**
No changes to existing peripherals or applications.
Solution: Beetle, an OS Service for BLE

Sharing
Apps can share access to peripherals.

Access Control
Users can specify access policies on peripherals and apps.

Communication flexibility
Communication between peripherals, gateway and cloud apps.

Backwards compatible
No changes to existing peripherals or applications.
Solution: Beetle, an OS Service for BLE

Sharing
Apps can share access to peripherals.

Access Control
Users can specify access policies on peripherals and apps.

Communication flexibility
Communication between peripherals, gateway and cloud apps.

Backwards compatible
No changes to existing peripherals or applications.
Solution: Beetle, an OS Service for BLE

Sharing
Apps can share access to peripherals.

Access Control
Users can specify access policies on peripherals and apps.

Communication flexibility
Communication between peripherals, gateway and cloud apps.

Backwards compatible
No changes to existing peripherals or applications.
Solution: Beetle, an OS Service for BLE

**Sharing**
Apps can share access to peripherals.

**Access Control**
Users can specify access policies on peripherals and apps.

**Communication flexibility**
Communication between peripherals, gateway and cloud apps.

**Backwards compatible**
No changes to existing peripherals or applications.
Solution: Beetle, an OS Service for BLE

Sharing
Apps can share access to peripherals.

Access Control
Users can specify access policies on peripherals and apps.

Communication flexibility
Communication between peripherals, gateway and cloud apps.

Backwards compatible
No changes to existing peripherals or applications.
Solution: Beetle, an OS Service for BLE Sharing
Apps can share access to peripherals.

Access Control
Users can specify access policies on peripherals and apps.

Communication flexibility
Communication between peripherals, gateway and cloud apps.

Backwards compatible
No changes to existing peripherals or applications.

Kernel Bluetooth Driver
Local apps
Virtual Devices
iForgotTheLights
Beetle: Key insight

Bluetooth application protocol amenable to multiplexing:

- Unified data model
- Standardized data types
- Transactions are meaningful to applications

Interposing on application layer can provide

- Sharing by multiplexing transactions from different clients
- Flexible communication by allowing transactions over any link
- Access control by mapping handle space
Idea #2: comprehensible, user-controllable access control

More challenges with Bluetooth API:

▶ Every app does its own access control.

▶ No common notion of identity.

▶ No common language for what’s permitted.
Bark, fine-grained access control at gateways

Figure 1: Enforcing fine-grained access control at gateways.

Figure 2: Beetle access control policies have simple English interpretations. ((who), [what], {where}, (when), [how])

James Hong, Amit Levy, and Philip Levis. Demo: Building Comprehensible Access Control for the Internet of Things with Beetle (MobiSys ’16)
Idea #3: who watches the IoT?

- Web browsers and smartphones allow user to install a CA cert.
  - Allows “auditors”: IDS, virus scan, curious researcher, Underwriters Laboratories . . .

- Today’s IoT devices don’t allow user-installed CA cert. Device talks to manufacturer with end-to-end encryption.

- Our view: users should be able to listen in on what their own devices are saying about them.

- But auditor only needs read-only access.
How can we audit TLS communication between our IoT devices and the cloud?

Secure Devices:
- No man-in-the-middle
- Signed firmware

*Nest used for illustrative purposes only.
Solution: TLS-RaR

A standard TLS connection...

- Begin TCP Connection
- Enter TLS Session
- Handshake
- AES-GCM
- Encrypted Session
Solution: TLS-RaR
Use standard TLS features to **Rotate** keys,

Begin TCP Connection
Enter TLS Session

Handshake

AES-GCM

Epoch 0

Rotate Keys
Reconnect, Renegotiate, Resume or KeyUpdate

AES-GCM

Epoch 1
Solution: TLS-RaR

Use standard TLS features to **Rotate** keys, and then securely **Release** the previous keys to auditing devices.
Nice Properties

- End-to-end integrity is preserved! (Unlike MITM)
  - Guaranteed tamper-proof communication.
Nice Properties

• **End-to-end integrity is preserved! (Unlike MITM)**
  - Guaranteed tamper-proof communication.

• Audit box's decryption yields the same stream of data as endpoints' `SSL_read()` calls, but delayed
  - Audit matches what was received
Nice Properties

• End-to-end integrity is preserved! (Unlike MITM)
  ➔ Guaranteed tamper-proof communication.

• Audit box's decryption yields the same stream of data as endpoints' SSL_read() calls, but delayed
  ➔ Audit matches what was received

• Format of TLS on the wire is not changed
  ➔ Easy to reason about security of the protocol
Nice Properties

- **End-to-end integrity is preserved! (Unlike MITM)**
  - Guaranteed tamper-proof communication.

- Audit box's decryption yields the same stream of data as endpoints' `SSL_read()` calls, but delayed
  - Audit matches what was received

- Format of TLS on the wire is not changed
  - Easy to reason about security of the protocol

- For some existing servers no change is necessary
  - Really easy to adopt
Nice Properties

• End-to-end integrity is preserved! (Unlike MITM)
  ➔ Guaranteed tamper-proof communication.

• Audit box's decryption yields the same stream of data as endpoints' SSL_read() calls, but delayed
  ➔ Audit matches what was received

• Format of TLS on the wire is not changed
  ➔ Easy to reason about security of the protocol

• For some existing servers no change is necessary
  ➔ Really easy to adopt

• Minimal change to OpenSSL on the device
  ➔ Easy to reason about security of the implementation
  ➔ Easy to adopt
Nice Properties

- **End-to-end integrity is preserved! (Unlike MITM)**
  - Guaranteed tamper-proof communication.

- Audit box's decryption yields the same stream of data as endpoints' `SSL_read()` calls, but delayed
  - Audit matches what was received

- Format of TLS on the wire is not changed
  - Easy to reason about security of the protocol

- For some existing servers no change is necessary
  - Really easy to adopt

- Minimal change to OpenSSL on the device
  - Easy to reason about security of the implementation
  - Easy to adopt
Idea #4: Who are my devices talking to?

TrafficMon: Home Gateway Monitoring System
Currently running on DD-WRT, a Linux based open source firmware

• Capture
  • Capture packets to and from each home device, up to 18,000 packets per second
  • Based on libpcap, a system-independent interface for user-level packet capture

• Extract
  • Extract header information such as source, destination, port, packet length

• Log
  • Logs are exported as text files to a USB drive connected to the gateway
  • Also accessible via DD-WRT’s web interface
### User Interface

**Traffic History**

**Pick a client to view:** [LENOVO-PC]   
**Pick protocols to view:** [Go!]

- Grouped by top-level domains
- Sorted by amount of data sent/received

<table>
<thead>
<tr>
<th>Host</th>
<th>IP</th>
<th>Total</th>
<th>HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>com</td>
<td></td>
<td>758B, 922B</td>
<td>758B, 922B</td>
</tr>
<tr>
<td>akamaitechnlogies</td>
<td>198.189.255.222</td>
<td>379B, 461B</td>
<td>379B, 461B</td>
</tr>
<tr>
<td>a198-189-255-222.de</td>
<td>198.189.255.201</td>
<td>379B, 461B</td>
<td>379B, 461B</td>
</tr>
<tr>
<td>net</td>
<td></td>
<td>6KB, -</td>
<td>-</td>
</tr>
<tr>
<td>mcast</td>
<td></td>
<td>6KB, -</td>
<td>-</td>
</tr>
<tr>
<td>Other</td>
<td></td>
<td>11KB, -</td>
<td>-</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td>11KB, -</td>
<td>-</td>
</tr>
</tbody>
</table>
Idea #5: Dealing with developing-world networks

Problems in the developing world:

- Medical devices work poorly across African cellular networks

  - Current solution: staffer collects a week’s data on a USB drive, takes motorcycle to hilltop with better coverage, uploads from there.

Greg Hill, Yu Yan, and Keith Winstein.
Our solution: a smart gateway that breaks the abstraction

- Collect all the (encrypted) data on a gateway.

- Strategy 1: Use better congestion control and error-correction.

- Strategy 2: Offer **anybody** $1 per GB for successfully getting that chunk to the cloud. Doesn’t matter how.
  - More motorcyclists?
  - Wi-Fi network?
  - New cell tower?
Congestion control and error correction for fragile networks

- Frank (Yu) Yan has assembled the Pantheon of Congestion Control
  - Every relevant congestion-control scheme
  - In one place, with one interface
  - Linux CUBIC, QUIC CUBIC, LEBAT, PCC, Verus, SCReAM, WebRTC GCC, Sprout, Koho...
  - CI tests and perf measurements for all schemes
  - Anybody can submit a pull request to add more

- Greg Hill is building the Observatory of Congestion Control
  - Goal: run the entire pantheon, every day, from 10+ developing-world cellular networks
  - Intent: create a platform for anybody to do developing-world wireless networking research
    - (Including automatic protocol-synthesis tools)
How We Learned to Stop Worrying and Love Gateways

1. **Beetle**: multiplexing and access control for BLE IoT devices [Levy/Hong/Riliskis/Levis/Winstein]

2. **Bark**: sane access-control policies for the IoT [Hong/Levy/Levis]

3. **TLS-RaR**: read-only audits of IoT communications [Wilson/Corrigan-Gibbs/Wahby/Boneh/Levis/Winstein]

4. **TrafficMon**: who are my devices talking to? [Rong/Levis]

5. **RAIL**: developing-world networks [Hill/Yan/Winstein]