IoT Security in Best Practice: Azure Sphere “Gen 1” on the MediaTek MT3620

Galen Hunt
Distinguished Engineer
Microsoft Azure Sphere

https://aka.ms/19bestpractices

@galen_hunt
/in/galenh
Outline

Part I. The Problem
Part II. The Product
Part III. The Practices
Part IV. The Proof
Part I. The Problem
Microcontrollers (MCUs)
low-cost, single chip computers

9 BILLION new MCU devices
built and deployed every year
The “Embedded” World.
1. **Customer experience:**
Deliver consistent quality, “the perfect pour every time”.

2. **Operational efficiency:**
Download recipes directly to machines

3. **Cost savings:**
Reduce unnecessary maintenance truck rolls
## Connecting Devices into an IoT Estate

### Azure IoT priority verticals
<table>
<thead>
<tr>
<th>Manufacturing</th>
<th>Retail</th>
<th>Agriculture</th>
<th>Energy</th>
<th>Smart Cities</th>
<th>Healthcare</th>
<th>Transportation</th>
</tr>
</thead>
</table>

### Azure IoT Solutions
- Azure IoT Central (SaaS)
- Azure IoT Reference Architecture & Accelerators (PaaS)
- Dynamics Connected Field Service (SaaS)

### Azure Services for IoT
- Azure IoT Hub
- Azure IoT Hub Device Provisioning Service
- Azure Digital Twins
- Azure Time Series Insights
- Azure Maps
- Azure Stream Analytics
- Azure Cosmos DB
- Azure AI
- Azure Cognitive Services
- Azure ML
- Azure Logic Apps
- Azure Active Directory
- Azure Monitor
- Azure DevOps
- Power BI
- Azure Data Share
- Azure Spatial Anchors

### IoT & Edge Device Support
- Azure Sphere
- Azure IoT Device SDK
- Azure IoT Edge
- Data Box Edge
- Windows IoT
- Azure Certified for IoT—Device Catalog
- Azure Stream Analytics
- Azure Storage
- Azure ML
- Azure SQL
- Azure Functions
- Azure Cognitive Services
Part II. The Product
The Azure Sphere mission is to provide an end-to-end security platform for embedded devices.

**Azure Sphere Certified Chips**
defense-in-depth chips with built-in Microsoft Pluton hardware root of trust

**Azure Sphere OS**
multi-layer, defense-in-depth operating system with robust application isolation

**Azure Sphere Security Service**
cloud service guarding every Azure Sphere device; brokers trust, detects emerging threats, and renews device security

**Azure Sphere Servicing**
ongoing security improvements from Microsoft protect against a continuously evolving threat environment
### MT3620 Azure Sphere Chip Architecture

<table>
<thead>
<tr>
<th>CPUs</th>
<th>ARM Cortex A7 (500MHz) + 2 x Cortex M4 (200MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>4MB</td>
</tr>
<tr>
<td>Flash</td>
<td>16MB (8MB Runtime Firmware + 8MB Backup Firmware)</td>
</tr>
<tr>
<td>Connectivity</td>
<td>WiFi 802.11 b/g/n, dual band: 2.4GHz, 5GHz</td>
</tr>
<tr>
<td>Microsoft Security</td>
<td>Pluton Security Subsystem, Firewalls, AES-256, SHA-2, ECC, RSA2K, e-Fused private and public keys, attestation, ...</td>
</tr>
<tr>
<td>I/O</td>
<td></td>
</tr>
<tr>
<td>GPIO</td>
<td>24, 4 configurable as PWM</td>
</tr>
<tr>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>I2C</td>
<td>6 configurable</td>
</tr>
<tr>
<td>UART</td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td>8 Channels, 12bit SAR, 2M sample/sec</td>
</tr>
<tr>
<td>I2S/TDM</td>
<td>I2S (2 interfaces) or TDM (4 channels)</td>
</tr>
<tr>
<td>Package</td>
<td>DR-QFN 164</td>
</tr>
</tbody>
</table>

**Diagram:**
- **Microsoft Pluton Security Subsystem**
- **Wi-Fi 802.11**
- **ARM Cortex-A7**
  - 64KB I / 32KB D L1
  - 128KB L2
  - FPU, NEON (SIMD)
- **ARM Cortex-M4**
  - 256KB SRAM TCM
  - DMA, I/O
- **Multiplexed MCU I/O**
  - GPIO
  - PWM
  - TDM
  - I2S
  - UART
  - I2C
  - SPI
  - ADC
Azure Sphere OS Architecture

Secure World

- Pluton Core
- Pluton Fabric
- Pluton Runtime
- Security Monitor

Normal World

- On-chip Cloud Services
- Custom Linux Kernel
- Executive Containers
- Real-Time Containers

OS Layer 4

OS Layer 3

OS Layer 2

OS Layer 1

OS Layer 0

Hardware

Cortex-A7

Cortex-M3s
Pluton Architecture

Pluton Security Subsystem

Secure RAM
Secure ROM
Secure Fuse Array

Memory Interface
Security Processor CPU
Environmental Sensors
Fuse Controller

Control Registers
Public Key Engine
AES Engine
SHA Engine
Random Number Generator

Public Key RAM

Cryptographic Operation Engine

Entropy Source

Secure ROM
Secure Fuse Array

Sparse Interconnect

Pluton Memory Interface
MPU

Secure Processor
CPU

Environmental Sensors
Fuse Controller

AES Engine
SHA Engine
Random Number Generator

Entropy Source

Secure ROM
Secure Fuse Array

Sparse Interconnect

Secure Processor
CPU

Environmental Sensors
Fuse Controller

AES Engine
SHA Engine
Random Number Generator

Entropy Source

Secure ROM
Secure Fuse Array

Sparse Interconnect
**PROTECTS** your devices and your customers with **certificate-based authentication** of all communication

**DETECTS** emerging security threats through **automated processing of on-device failures**

**RESPONDS** to threats with fully automated on-device OS updates

**ALLOWS** for easy deployment of software **app updates** to Azure Sphere powered devices
## Azure Sphere Protocols

<table>
<thead>
<tr>
<th>Technology</th>
<th>Purpose</th>
<th>Protocol</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server authentication</td>
<td>Verify remote server identity.</td>
<td>TLS</td>
<td>Verifies Microsoft’s identity. Certificate chain is put on chips during manufacturing.</td>
</tr>
<tr>
<td>Secure Boot</td>
<td>Verify software executed is genuine.</td>
<td>ECDSA using ECC public keys on device</td>
<td>Uses chain of trust. First public key burned into fuses on device.</td>
</tr>
<tr>
<td>Measured Boot/Remote Attestation</td>
<td>Proves to the Azure Sphere Security Service that the chip is genuine and running trusted software.</td>
<td>Custom remote attestation protocol</td>
<td>Depends on ECC public/private key pair generated within Pluton and burned into fuses. Only private key on-device used by Azure Sphere.</td>
</tr>
<tr>
<td>Device authentication</td>
<td>Proves to any service on the internet that the Azure Sphere device completed attestation successfully.</td>
<td>TLS</td>
<td>Generates a special, short-lived device certificate, via remote attestation, which is used for TLS device authentication</td>
</tr>
</tbody>
</table>
Part III. The Practices
1. Treat ROM as non-updatable software and minimize its size.

2. Never expose private device keys to software.

3. In IoT, choose ECC, not RSA, for device-specific keys.

4. Use Secure Boot everywhere and always.

5. Use silicon-based Measured Boot to attest remotely that Secure Boot completed successfully.

6. Do not use (or parse) certificates in the Trusted Computing Base (TCB).

+ Azure Sphere minimizes the amount of ROM code and includes countermeasures that make it more difficult to skip critical code paths.

+ Azure Sphere chips embed keys in silicon and use elliptic-curve cryptography (ECC) public/private key pairs to implement Measured Boot and Secure Boot.

+ Azure Sphere uses ECC keys. They are more cost-effective for greater security.

+ Azure Sphere helps protect the boot process by using ECC to power Secure Boot on every piece of software that runs on the device.

+ Azure Sphere chips with silicon-based Measured Boot ensure only three possible outcomes: a successful attestation, an attestation that requires a software update, and a failed attestation attempt.

+ There's no need for additional KPI to prove certificates if you generate keys on the device and collect those keys during the chip manufacturing process. That's what Azure Sphere does.
Use silicon-based Measured Boot to attest remotely that Secure Boot completed successfully.

Azure Sphere chips perform silicon-based Measured Boot on start-up. This ensures only three possible outcomes when authenticating to the cloud: a successful attestation, an attestation that requires a software update, or a failed attestation attempt.
7. Handle server certificate expiration gracefully.

8. Connectivity is optional.

9. Make it harder to build botnets out of zero-day vulnerabilities.

10. Use a policy of “deny by default” and enforce it in silicon.

11. Eliminate the concept of users on IoT devices.

12. Physically separate real-time execution from internet communication.

- With Azure Sphere, devices manage server certificates before they connect for attestation—no matter how long they’ve been offline.
- With Azure Sphere, devices continue to operate even when they’re not connected. Secure Boot does not use certificates, so you don’t need to keep them connected just to keep them running.
- Azure Sphere addresses network firewall permissions during application development—so applications won’t modify firewalls at runtime.
- Azure Sphere chips ground resource access control mechanisms in silicon. Every resource that is accessible from software is capable of silicon-based lockdown.
- User accounts on devices introduce new attack surfaces. The Azure Sphere operating system does not have user accounts, logins, or their associated passwords.
- Azure Sphere chips contain two different cores. Separating execution domains into different physical cores is the best way to guarantee that one core cannot interfere with another.
9. Make it harder to build botnets out of zero-day vulnerabilities.

Network firewalls, which name valid cloud targets, are programmed by application manifests.

The manifests are created during application development—so applications can’t modify firewalls at runtime.
13. Divide code into user-mode and kernel-mode code.

14. Ensure all software is updatable.

15. Make software update-fault tolerant.

16. Isolate applications to make updates easier.

17. Do not allow the system to dynamically change code execution.

18. Defend against downgrade attacks.

Because Azure Sphere uses a Cortex-A for its Linux-kernel-based operating system, it supports virtualized address spaces, an isolated kernel, and hardware-isolated applications.

With Azure Sphere, every piece of software, including the bootloader, can be updated remotely.

Azure Sphere uses several techniques to ensure that software updates succeed and are fault-tolerant.

Azure Sphere helps reconcile dependencies between OS and applications so it’s easier to update applications more frequently.

Dynamic code execution at runtime introduces attack surfaces that are difficult to secure. Azure Sphere disables these attack surfaces, so that attackers cannot exploit them.

Azure Sphere is built so that it can stop trusting—and running—all previous versions of the operating system.
Make software update-fault tolerant.

Azure Sphere keeps Last Known Good images for failback, uses a separate TCB for update and recovery code, and uses on-device erasure coding to correct local storage corruption.
19. Use tools and processes to make software more secure

Example techniques we use to make software development more secure

- **Automated common vulnerabilities and exposures (CVE) checks**
  - Azure Sphere's build system checks for CVEs in the operating system build process, so you don’t need to manually check whether a CVE is filed.

- **Software fuzz testing**
  - Azure Sphere integrates several different fuzz testing tools into its software development processes to look for and find bugs in data processing and parsing before the software ships to customers.

- **Static analysis**
  - Azure Sphere runs several types of static analysis tools, so you can more easily see code patterns that may indicate vulnerabilities.

- **Red team exercises**
  - Azure Sphere regularly hosts red team exercises against both the operating system and the Azure Sphere Security Service.
Part IV. The Proof
What does one of our red team exercises look like?
2020 Azure Sphere Security Research Challenge

Three-month security challenge with the world’s best researchers and red teams

June 1 to August 31, 2020

Enable researchers to find high impact security vulnerabilities

Validate our security promise with the best in their field

70 of the most talented individual researchers & security vendors from over 21 countries

Dev kits, kernel source code, direct line to OS Security Team, weekly office hours, email support

McAfee, Cisco Talos, FireEye, Avira, ESET, F-Secure, Zscaler, etc.

Bounties of up to $100k for ability to execute code on Secure World & Pluton
What it takes to defend against the best

Learnings from the challenge and 70 hackers

Submission breakdown:
- Total submissions: 40
- Led to improvements: 30
- Non-issues: 10

Total bounty awards:
- $374,300
  - Largest: $48,000
  - Smallest: $3,300

How we mitigated:

- Fixed in less than a week:
  Pivot point was in cloud infrastructure. One fix in our cloud, rendered full attack chain unable to execute

- Fixed remaining issues in less than 30 days:
  - Potential vulnerabilities each fixed with next Azure Sphere OS release after disclosure
  - Linux Kernel updated publicly

• McAfee ATR put together attack chain with half a dozen vulnerabilities (source)
• 0-day in Linux Kernel found by McAfee ATR & Cisco Talos
• Even after getting kernel root access, hackers were still unable to compromise Secure World and Pluton
The attacker’s approach:  

**What persistent hacking really looks like** *(source)*

**Became expert in product**
- Analyzed all publicly available documentation
- Leveraged two-year old YouTube talk from team member
- Used tools such as IDA Pro and DNSpy to understand system
- Code reversing (C/C++), reading ARM assembly, decompiler output
- Divide and conquer: Split into two teams to pursue different paths

**Attacked every possible surface**
- Network stack
- Rogue application for sandbox escape
- Weaknesses in signature verification
- Target communication between development board and host-pc
- Drivers handling GPIO, SPI, I2C, etc.
- Communication between cores
- WiFi core/module

**Went as deep as possible**
- Used recovery mechanism to look at recovery file
- Analyzed .bin files and image manifest
- Imported raw blobs (security monitor, Pluton, etc.) into IDA Pro

**Built rogue applications**
- Packaged a custom application with Unbridled Libc
- Got familiar with Userland
- Looked at the ASXIPFS code
- Patched ASXIPFS archive to add a Symlink

Let’s secure the future.

SECURED FROM THE SILICON UP