Microsoft Azure

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

Galen Hunt
Distinguished Engineer
Microsoft Azure Sphere

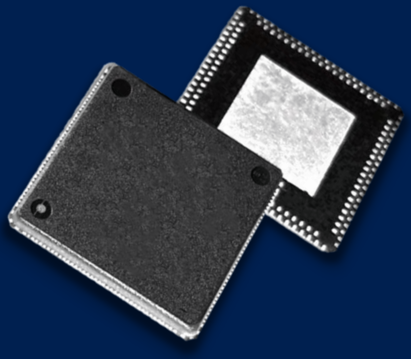
https://aka.ms/19bestpractices

@galen_hunt
/in/galenh

Outline

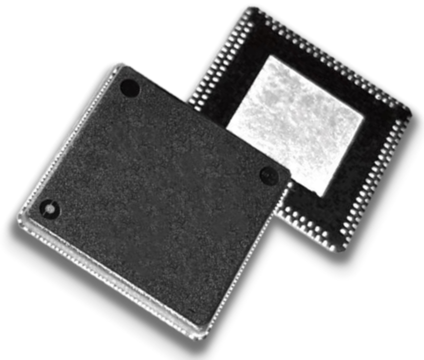
Part I. The Problem

Microcontrollers (MCUs)
low-cost, single chip computers



9 BILLION new MCU devices built and deployed every year

The “Embedded” World.



STARBUCKS

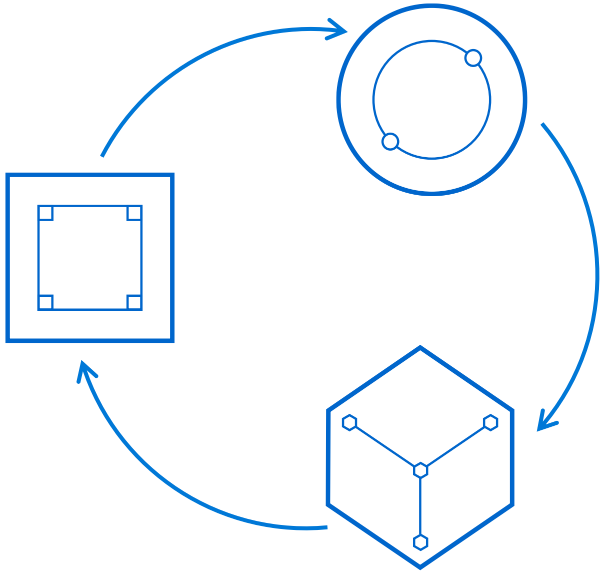
1. **Customer experience:** Deliver consistent quality, “the perfect pour every time”.
2. **Operational efficiency:** Download recipes directly to machines
3. **Cost savings:** Reduce unnecessary maintenance truck rolls

Connecting Devices into an IoT Estate

Azure IoT priority verticals	Manufacturing, Retail, Agriculture, Energy, Smart Cities, Healthcare, Transportation			
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**

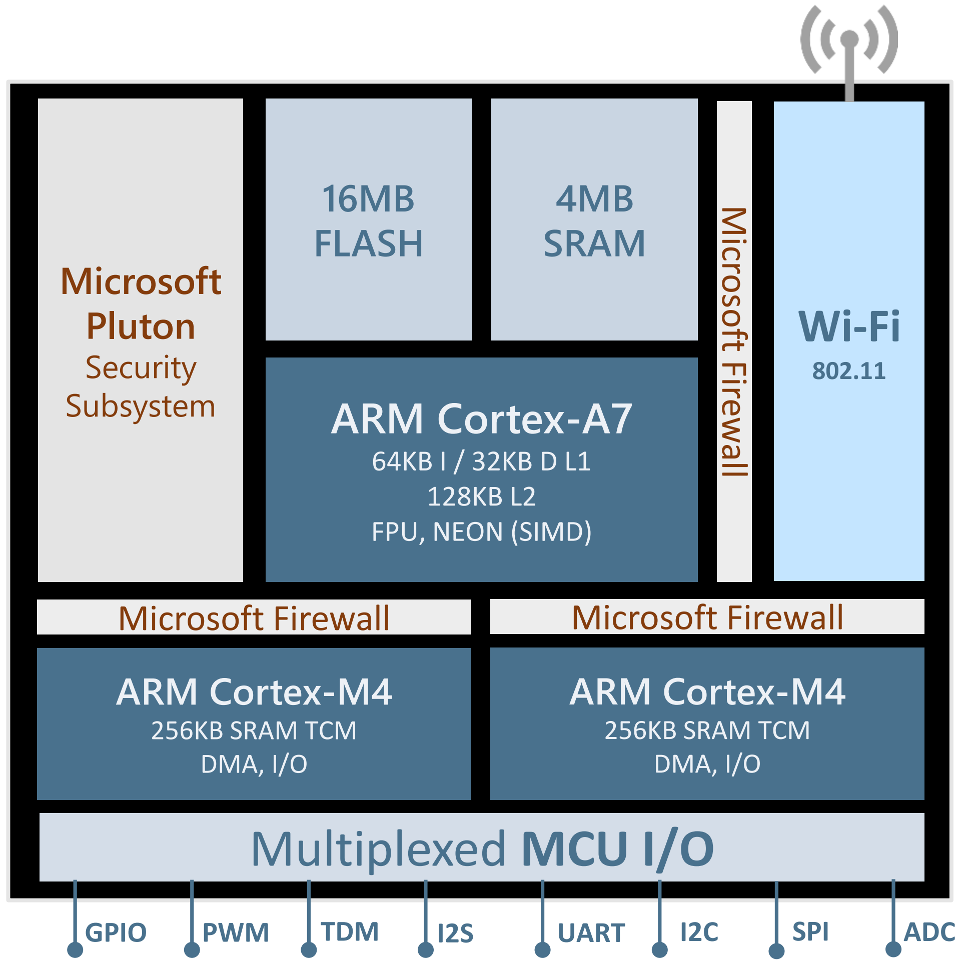
Azure Sphere OS
multi-layer, defense-in-depth operating system with **robust application isolation**

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

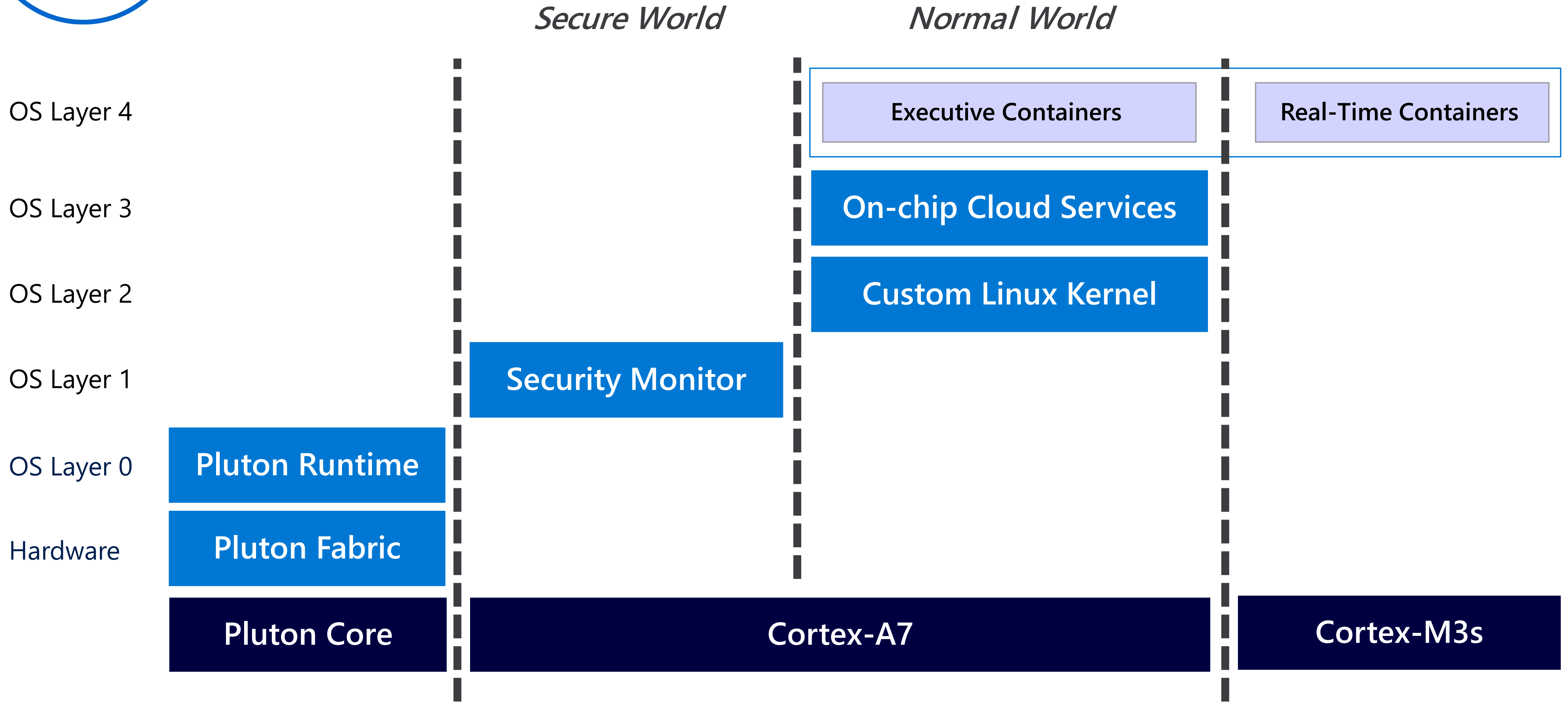
Azure Sphere Servicing
ongoing security improvements from Microsoft protect against a continuously evolving threat environment

MT3620 Azure Sphere Chip Architecture

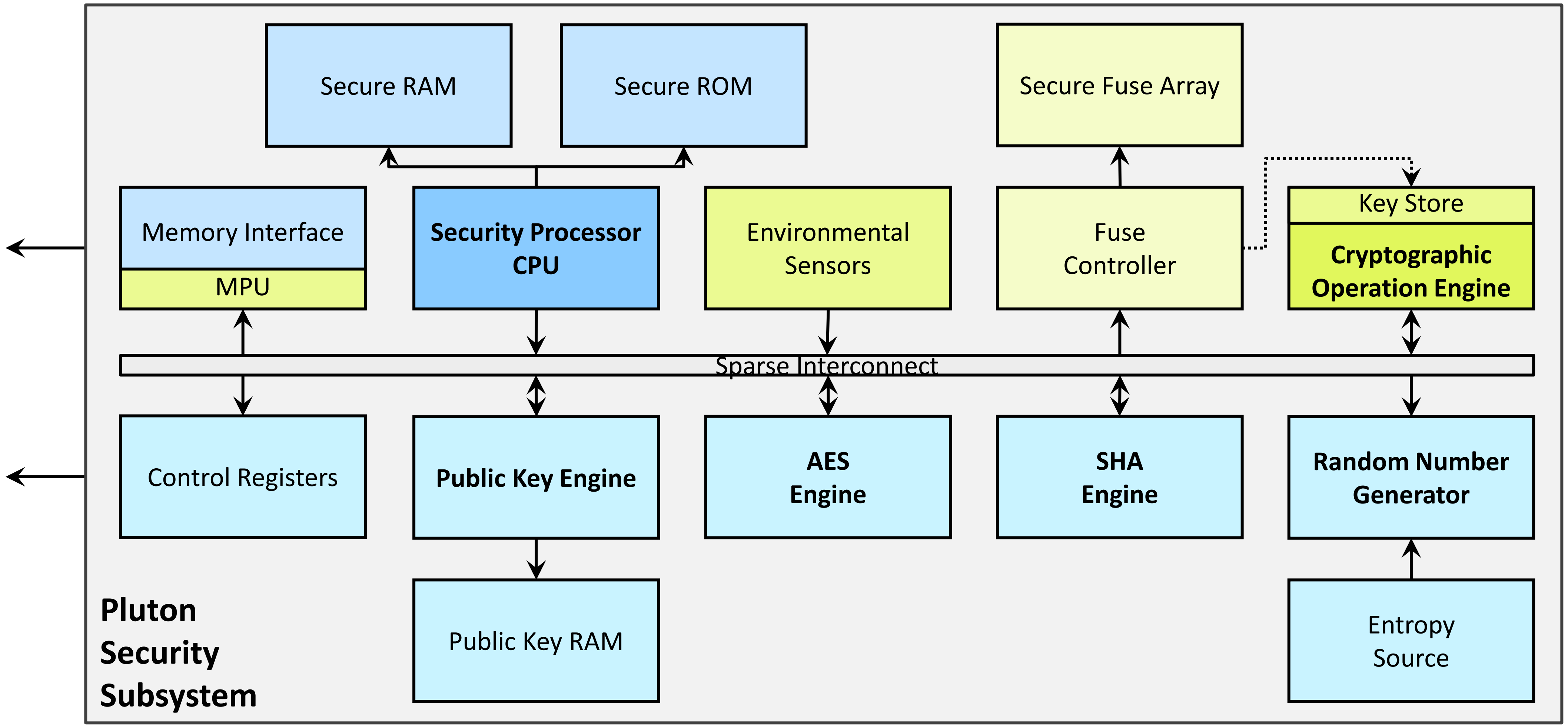
CPUs		**ARM Cortex A7 (500MHz) +** **2 x Cortex M4 (200MHz)**	
RAM		4MB	
Flash		16MB (8MB Runtime Firmware + 8MB Backup Firmware)	
Connectivity		WiFi 802.11 b/g/n, dual band: 2.4GHz, 5GHz	
Microsoft Security		**Pluton Security Subsystem**, Firewalls, AES-256, SHA-2, ECC, RSA2K, e-Fused private and public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


Microsoft Pluton Security Subsystem
16MB FLASH
4MB SRAM
Microsoft Firewall
Wi-Fi 802.11
ARM Cortex-A7 64KB I / 32KB D L1 128KB L2 FPU, NEON (SIMD)
Microsoft Firewall
Microsoft Firewall
ARM Cortex-M4 256KB SRAM TCM DMA, I/O
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



Pluton Architecture



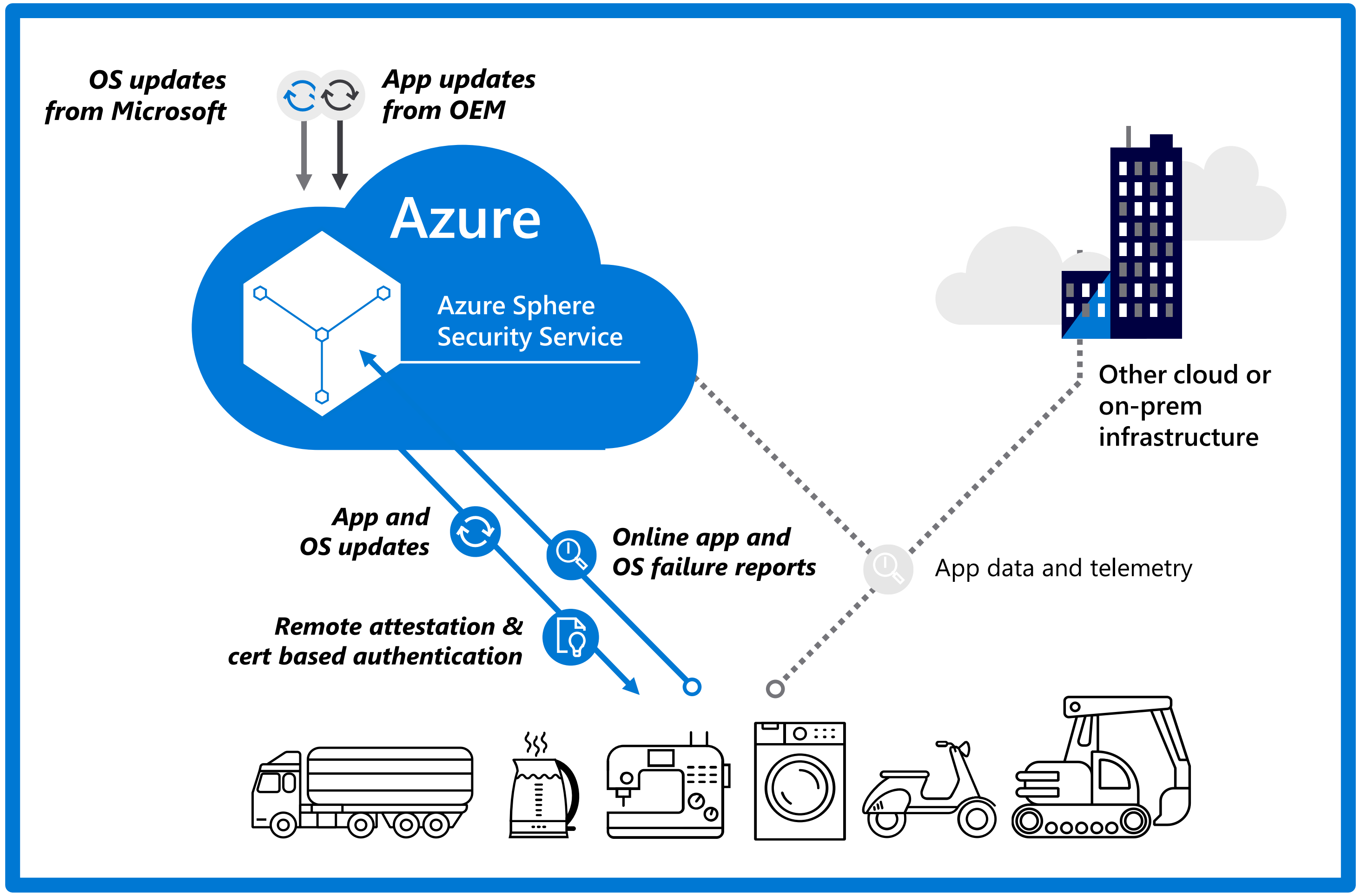
Azure Sphere Security Service Architecture

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


OS updates from Microsoft
App updates from OEM
Azure
Azure Sphere Security Service
Other cloud or on-prem infrastructure
App and OS updates
Online app and OS failure reports
App data and telemetry
Remote attestation & cert based authentication

Azure Sphere Protocols

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

Part III. The Practices

1.

Treat ROM as non-updatable software and minimize its size.

+

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

2.

Never expose private device keys to software.

+

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

3.

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

+

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

4.

Use Secure Boot everywhere and always.

+

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

5.

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

+

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.

6.

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

+

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.

5.

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.

+

With Azure Sphere, devices manage server certificates before they connect for attestation—no matter how long they've been offline.

8.

Connectivity is optional.

+

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.

9.

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

+

Azure Sphere addresses network firewall permissions during application development—so applications won't modify firewalls at runtime.

10.

Use a policy of "deny by default" and enforce it in silicon.

+

Azure Sphere chips ground resource access control mechanisms in silicon. Every resource that is accessible from software is capable of silicon-based lockdown.

11.

Eliminate the concept of users on IoT devices.

+

User accounts on devices introduce new attack surfaces. The Azure Sphere operating system does not have user accounts, logins, or their associated passwords.

12.

Physically separate real-time execution from internet communication.

+

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.

+

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.

14. Ensure all software is updatable.

+

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

15. Make software update-fault tolerant.

+

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

16. Isolate applications to make update easier.

+

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

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

+

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.

18. Defend against downgrade attacks.

+

Azure Sphere is built so that it can stop trusting—and running—all previous versions of the operating system.

15.

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

+

Writing software is difficult. It will always have unknown bugs.

Our goal with Azure Sphere is that customers do not need to reinvent security features for their own IoT products.

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.

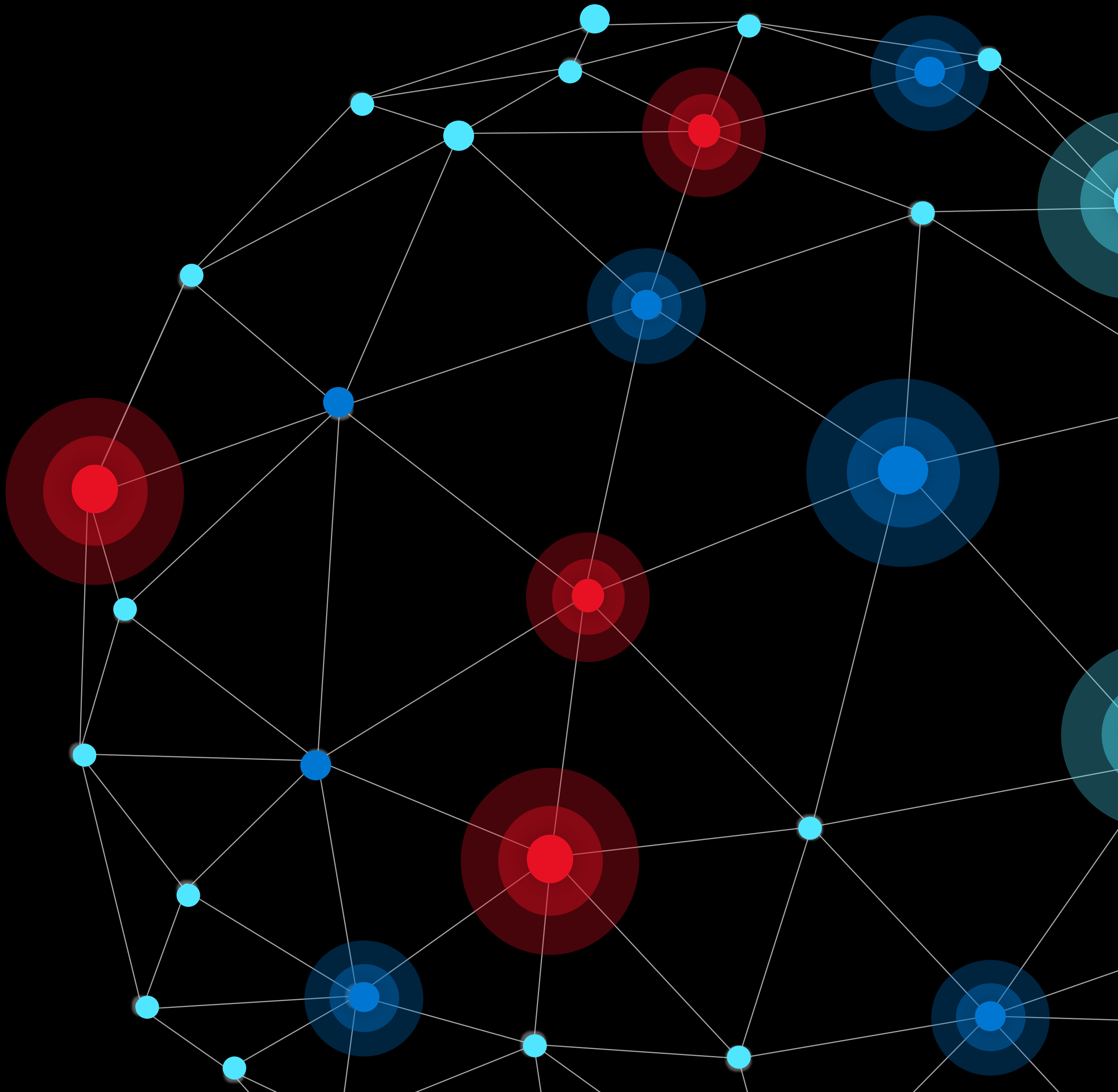
✓

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
- 70 of the most talented individual researchers & security vendors from over 21 countries
- McAfee, Cisco Talos, FireEye, Avira, ESET, F-Secure, Zscaler, etc.
- Validate our security promise with the best in their field
- Dev kits, kernel source code, direct line to OS Security Team, weekly office hours, email support
- 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**

- 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

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

The attacker's approach: McAfee

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

https://www.mcafee.com/enterprise/en-us/assets/white-papers/wp-prisoner-of-azure-kaban.pdf

Let’s secure the future.

SECURED FROM THE SILICON UP

@galen_hunt

/in/galenh