Overview

- About Me
- Intro to Zephyr
- What We’re Trying To Protect From
- Secure Development Lifecycle
- TinyCrypt
- Possible Future Directions
About Me

- **Constanza Heath, Security Researcher, Intel Open Source Technology Center**
- 1998 Time Warner Cable’s Information Technology Dept.
- 2004 Hewlett Packard’s Imaging Product Group, All-in-One Division, and Photosmart Printer Division.
- Joined Intel in 2010.
- Previously worked on Moblin, MeeGo and Tizen operating systems.
- Career up and down the software stack, making security a natural discipline.
- Design core security features, review architectures, and implement secure software solutions.
- Currently focused on IoT security and enablement and Cloud OSs.
What is Zephyr?

- Microcontroller operating system
- Very small memory footprint (will run in 8k)
- Open Source under Apache* 2.0 license, hosted by Linux* Foundation
- Supports multiple architectures
Why Zephyr Project?

- Strategic Investment
- Best-of-Breed RTOS
- True Open Source
- Permissively Licensed
- Established Code Base
- Secure
Zephyr Overview

- Provide an OS that runs best on MCUs for wearable and IoT devices, where the cost of the silicon is minimal
- Highly Configurable, Highly Modular
- Kernel mode only
- Two Modes:
  - Nanokernel: Limited functionality targeting small footprint (below 10k)
  - Microkernel (superset of nanokernel): with additional functionality and features
- No user-space and no dynamic runtimes
- Memory and Resources are typically statically allocated
- Cross architecture (IA32, ARM*, ARC, others under discussion)

---

**Applications**

3rd Party Libraries

- C APIs
- CoAP, MQTT*, HTTP*, LWM2M*
- DTLS, TLS, ..

**Device Drivers**

- IPv4*, IPv6
- 6LowPAN
- Device Management

**Power Management**

Microkernel

Nanokernel

**Platform**

- UART
- SPI
- GPIO
- PC
- Radios

* planned
Zephyr Nanokernel Overview

- A high-performance, multi-threaded execution environment with a basic set of kernel features
- Ideal for systems with sparse memory (the kernel itself requires as little as 2 KB!) or only simple multi-threading requirements (such as a set of interrupt handlers and a single background task)
- Examples of such systems include:
  - embedded sensor hubs
  - environmental sensors
  - simple LED wearables
  - store inventory tags
Zephyr Microkernel Overview

- Supplements the capabilities of the nanokernel to provide a richer set of kernel features
- Suitable for systems with
  - heftier memory (50 to 900 KB)
  - multiple communication devices (like Wi-Fi and Bluetooth® Low Energy)
  - and multiple data processing tasks
- Examples of such systems include:
  - Fitness wearables
  - Smart watches
  - IoT wireless gateways
Supported Platforms

- Arduino* 101
- Intel® Quark™ D2000 CRB
- 2nd Generation Intel® Galileo
- FRDM-K64F
- Arduino Due

► More platforms and boards to follow ◀
Anas Nashif - Zephyr™ Project Overview Tuesday, April 5, 2016
Security

- Standardized building block and robust communication stacks
- Cryptographic library based on TinyCrypt
- Static and single binary applications, Single address space, No loadable modules
- Planned security features:
  - Device Management and Updates
  - APIs to support vendor specific Crypto implementations (SW/HW)
  - Secure Key Storage
What’s Out There?
Not Your Mother’s Hacker
Security Hackers Ecosystem

- National Interest
  - Spy
- Personal Gain
  - Thief
  - Trespasser
  - Expert
  - Specialist
- Personal Fame
  - Author
- Curiosity
  - Vandal
  - Script-Kiddy

The World Today

Tools created by experts now used by less-skilled attackers and criminals

Fastest growing segment
"Two newly-discovered flaws light fire under IoT security" - ZDNet

"'Internet of Things’ security is hilariously broken and getting worse" – Ars Technica

Tentler has discussed the lack of security on webcams and other devices connected to the Internet at multiple conferences, and occasionally he posts interesting results from Shodan to his Twitter feed. The popularity of the devices and their lack of security is creating a burgeoning problem, he said in an e-mail interview.

“It says neither consumers nor vendors care about security, and it’s going to be an amazing apocalypse,” Tentler said.

"Princeton researches find security flaws in IoT devices” - Engadget

"Internet of Things Security Problem Just Keeps Getting Bigger” - eWeek

Just in the last few Months!
New Zephyr™ Project Features

- Take new features through Security Risk Assessment
  - Security Risk Assessment determines what security review takes place
  - Security features get extra scrutiny

- Review based on:
  - Initial touch feature review
  - Use of cryptography
  - Compliance with Threat Model
  - Incorporation into Security Architecture
Zephyr™ Project Code Review

- Why do Code Review?
  - No one writes perfect code
  - Ensure code complies with rest of communities expectations
- All code goes through review in Gerrit
  - Ensure appropriate string/memory function use
  - Validate user input
  - Readability & Maintainability
  - Correct Functionality
  - Security review of all security relevant code
Zephyr™ Project Static Code Analysis

- Why Static Code Analyze Zephyr?
  - Developers aren’t perfect at code review either
  - Static Analysis (despite significant false positives) identifies low-hanging functional and security defects
  - It also finds some complex and hard to identify issues

- All code goes through static code analysis
  - Fix all identified significant issues
  - Considering utilizing https://scan.coverity.com/ as project solution
Why Survivability?
- Projects Upstream of Zephyr will have security vulnerabilities discovered
- Ensure Zephyr knows how to respond when a security issue is found

Team ready to respond to security issues
- Project Practices “Responsible Disclosure”
TinyCrypt
Zephyr™ Project Crypto Needs

- Cryptographic protections have become standard for almost all forms of communication
  - If Zephyr is going to talk to things, it needs to talk crypto
- Complication: Standard cryptographic libraries like OpenSSL are orders of magnitude larger than Zephyr target platforms.
  - What to do?
EUREKA!

Let’s see what’s available!
Cryptographic Experts in Intel Labs had been looking at the problem

They had an internal codebase they had already been working on which met Zephyr’s needs

Zephyr knew this codebase would be useful for other projects

TinyCrypt was made its own project

Upstream Maintainer: Constanza Heath

https://github.com/01org/tinycrypt

Goals:

- Minimize code size of each crypto primitive
- Minimize dependencies among primitive implementations
- Multiple architecture
TinyCrypt Provides

- SHA-256
  - Hash function
  - Defined in NIST FIPS PUB 180-4
- HMAC-SHA256
  - Message authentication code
  - Defined in RFC 2104
  - Requires SHA-256
- HMAC-PRNG
  - Pseudo-random number generator
  - Defined in NIST SP 800-90A
  - Requires SHA-256 & HMAC-SHA256
- AES-128
  - Block cipher
  - Defined in NIST FIPS PUB 197
- AES-CBC
  - AES mode of operation
  - Defined in NIST SP 800-38A
  - Requires AES-128
- AES-CTR
  - AES mode of operation
  - Defined in NIST SP 800-38A
  - Requires AES-128
- AES-CMAC
  - Message authentication code
  - Defined in NIST SP 800-38B
  - Requires AES-128
- AES-CCM
  - Authenticated encryption
  - Defined in NIST SP 800-38C
  - Requires AES-128
- ECC-DH
  - Key Exchange
  - Defined in RFC 6090
  - Requires ECC auxiliary functions
- ECC-DSA
  - Digital Signature
  - Defined in RFC 6090
  - Requires ECC auxiliary functions

Can be utilized by TLS libraries
Looking Forward
Future Directions

- Memory Protections
- Device Management and Updates
- APIs to support vendor specific Crypto implementations (SW/HW)
- Secure Key Storage
- Your Ideas?
Summary

- Zephyr Project is...
  - Being proactive about security
  - Looking at the environment that we are deploying into for security concerns
  - Ensuring software is being constructed correctly from a security perspective
  - Including a new, novel cryptographic library in TinyCrypt
  - Looking for feedback on what’s useful for future directions of security
Visit Us!
Booth #108
Zephyr™ Project Demos from our member companies

Visit Us!
Booth #108
Zephyr™ Project Demos from our member companies

www.zephyrproject.org

<table>
<thead>
<tr>
<th>Other Zephyr™ Project Sessions @ ELC/OpenIoT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zephyr™ Project Security</td>
</tr>
<tr>
<td>Zephyr™ Project Overview</td>
</tr>
<tr>
<td>Power Management in Zephyr™ RTOS</td>
</tr>
</tbody>
</table>
www.zephyrproject.org
Zephyr Kernel – Key Features

Multi-threading services, including both priority-based, non-preemptive fibers and priority-based, preemptive tasks (with optional round robin time-slicing).

Interrupt services, including both compile-time and run-time registration of interrupt handlers, which can be written in C or assembly language.

Inter-thread synchronization services, including binary semaphores, counting semaphores, and mutex semaphores.

Inter-thread data passing services, including basic message queues, enhanced message queues, and byte streams.

Memory allocation services, including dynamic allocation and freeing of fixed-size or variable-size memory blocks.
Zephyr Kernel – Key Features

- Power management services, including tick-less idle and an advanced idling infrastructure.
- Highly configurable, allowing an application to incorporate only the capabilities it needs, and to specify their quantity and size.
- Zephyr requires all system resources to be defined at compile-time to reduce code size and increase performance.
- Provides minimal run-time error checking to reduce code size and increase performance. An optional error checking infrastructure is provided that can assist in debugging during application development.
- Library based RTOS ("kernel-less")
Best-of-Breed Tools

- Kbuild - The build system of the Linux kernel
- Kconfig - The configuration system of the Linux kernel
- Rich and Powerful SDK developed specifically for Zephyr and powered by the Yocto project:
  - 5 different cross-compilers for all supported architectures and platforms
  - Support for baremetal c library, based on newlib
  - Host tools needed for debugging and downloading images (flashing) into target platforms
- Builds natively on Linux*, MacOS* and Microsoft* Windows
- Building on all various operating systems using Docker containers