Making RISC-V the Most Secure Platform Ever

Cesare Garlati, Hex Five Security

RISC-V Summit – December 4, 2018
The Big Hack

How [REDACTED] used a tiny chip to infiltrate America's top companies
Researchers Detail Two New Attacks on TPM Chips

By Catalin Cimpanu

Some PC owners may need to apply motherboard firmware updates in the near future to address two attacks on TPM chips detailed earlier this month by four researchers from the National Security Research Institute of South Korea.
SgxPectre attack allows to reveal the content of the SGX enclave

March 5, 2018  By Pierluigi Paganini

A group of researchers from the Ohio State University has discovered a new variation of the Spectre attack named SgxPectre that allows to reveal the content of the SGX enclave.
FreeRTOS TCP/IP Stack Vulnerabilities Put A Wide Range of Devices at Risk of Compromise: From Smart Homes to Critical Infrastructure Systems

Researchers: Ori Karliner (@oriHCX)

Relevant Operating Systems: FreeRTOS up to V10.0.1 (with FreeRTOS+TCP), AWS FreeRTOS up to V1.3.1, WHIS OpenRTOS and SafeRTOS (With WHIS Connect middleware TCP/IP components).

CVE List:

<table>
<thead>
<tr>
<th>CVE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2018-16522</td>
<td>Remote code execution</td>
</tr>
</tbody>
</table>
Brain implants used to treat Parkinson's can be hacked and used to control people, scientists warn

By Natasha Bemal
31 October 2018 • 11:33AM

Brain implants used to treat Parkinson's disease could be hacked by cyber attackers and used to control people, scientists have claimed.
Making RISC–V the Most Secure Platform Ever

Cesare Garlati, Hex Five Security

RISC–V Summit – December 4, 2018
Traditional computing is not safe nor secure

All programs share the same access to code and data
- Compiler & linker combine code blocks into monolithic binaries
- Functional blocks communicate via shared memory – stack, heap, buffers
- Any bit of code can access any bit of data – can’t hide secrets
- One faulty / malicious instruction can stall / crash the whole system – DOS
- MMUs that run 17M lines of “rich OS” can’t be trusted either – see enclaves

System engineering – uncomfortable commercial reality
- No company can afford the cost and the time to develop a full sw stack in house
- Open source software, 3rd party libraries, “untouchable” legacy code are the norm
- The design complexity of security technologies often results in them not being used

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler &amp; Linker</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|                  | stack          |              |
|------------------|----------------|
|                  | ↓              |

|                  | heap           |              |
|------------------|----------------|
|                  | ↑              |

<table>
<thead>
<tr>
<th></th>
<th>uninitialized data (bss)</th>
<th>initialized data</th>
<th>text</th>
</tr>
</thead>
</table>
(a) Industry Average: "about 15 - 50 errors per 1000 lines of delivered code." He further says this is usually representative of code that has some level of structured programming behind it, but probably includes a mix of coding techniques.

(b) Microsoft Applications: "about 10 - 20 defects per 1000 lines of code during in-house testing, and 0.5 defect per KLOC (KLOC IS CALLED AS 1000 lines of code) in released product (Moore 1992)." He attributes this to a combination of code-reading techniques and independent testing (discussed further in another chapter of his book).

(c) "Harlan Mills pioneered ‘cleanroom development’, a technique that has been able to achieve rates as low as 3 defects per 1000 lines of code during in-house testing and 0.1 defect per 1000 lines of code in released product (Cobb and Mills 1990).

17,019,619 * 10^-4 = 1,701 disasters waiting to happen
A Better Way – Multidomain Trusted Execution Environment

Functional Code Blocks

- Network Stack
- Root of Trust
- Crypto Libraries
- User Code

Compiler & Linker

<table>
<thead>
<tr>
<th>high address</th>
<th>stack</th>
<th></th>
<th>stack</th>
<th></th>
<th>stack</th>
<th></th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stack</td>
<td></td>
<td>stack</td>
<td></td>
<td>stack</td>
<td></td>
<td>stack</td>
</tr>
<tr>
<td>heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>heap</td>
<td></td>
<td>heap</td>
<td></td>
<td>heap</td>
<td></td>
<td>heap</td>
</tr>
<tr>
<td>uninitialized data (bss)</td>
<td></td>
<td>uninitialized data (bss)</td>
<td></td>
<td>uninitialized data (bss)</td>
<td></td>
<td>uninitialized data (bss)</td>
<td></td>
</tr>
<tr>
<td>initialized data</td>
<td></td>
<td>initialized data</td>
<td></td>
<td>initialized data</td>
<td></td>
<td>initialized data</td>
<td></td>
</tr>
<tr>
<td>text</td>
<td></td>
<td>text</td>
<td></td>
<td>text</td>
<td></td>
<td>text</td>
<td></td>
</tr>
</tbody>
</table>

Secure InterZone™ Communications – no shared memory
Secure MultiZone™ nanoKernel – boot room

low address

text

Network Stack  Root of Trust  Crypto Libraries  Operating System
RISC-V Security 101

Privilege Levels (rings) – Control and Status Registers (CSRs)

- Machine – always present
- Supervisor – Linux
- Reserved (Hypervisor) – work in progress
- User / Application – i.e. SiFive E21/E31/E51, Andes N25 / NX25
- Trusted Execution Environment runs at highest privilege level
- Note: Interrupts always M mode (unless “N” implemented)

<table>
<thead>
<tr>
<th>Number of levels</th>
<th>Supported Modes</th>
<th>Intended Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M</td>
<td>Simple embedded systems</td>
</tr>
<tr>
<td>2</td>
<td>M,U</td>
<td>Secure embedded systems</td>
</tr>
<tr>
<td>3</td>
<td>M,S,U</td>
<td>Unix-like operating systems</td>
</tr>
</tbody>
</table>
RISC-V Security 101

Physical Memory Attributes / Physical Memory Protection

- Hardware enforced – 4 ranges * 4 config reg (if implemented)
- Policy R/W/X => synchronous exception mechanism (trap)
- Overlapping OK, ranges can be locked down
- Top of range (TOR) or naturally aligned power of two (NAPOT)
- Trusted Execution Environment manages PMP context at runtime
- Note: enforced per core – no ISA spec for multi-core / platform

<table>
<thead>
<tr>
<th>A</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OFF</td>
<td>Null region (disabled)</td>
</tr>
<tr>
<td>1</td>
<td>TOR</td>
<td>Top of range</td>
</tr>
<tr>
<td>2</td>
<td>NA4</td>
<td>Naturally aligned four-byte region</td>
</tr>
<tr>
<td>3</td>
<td>NAPOT</td>
<td>Naturally aligned power-of-two region, ≥8 bytes</td>
</tr>
</tbody>
</table>
User–mode Interrupts and exception handling (“N”)
- Outer ring delegates designated interrupts to user–level
- Hardware transfers control directly to U–mode – but no security ctx
- Intended for secure embedded systems with only M + U
- Optional extension – no known commercial implementations yet
- Trusted Execution Environment provides “N” in sw via trap & emulate

<table>
<thead>
<tr>
<th>Number</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>ustatus</td>
<td>User status register.</td>
</tr>
<tr>
<td>0x004</td>
<td>uie</td>
<td>User interrupt-enable register.</td>
</tr>
<tr>
<td>0x005</td>
<td>utvec</td>
<td>User trap handler base address.</td>
</tr>
<tr>
<td>0x040</td>
<td>uscratch</td>
<td>Scratch register for user trap handlers.</td>
</tr>
<tr>
<td>0x041</td>
<td>uepc</td>
<td>User exception program counter.</td>
</tr>
<tr>
<td>0x042</td>
<td>ucause</td>
<td>User trap cause.</td>
</tr>
<tr>
<td>0x043</td>
<td>utval</td>
<td>User bad address or instruction.</td>
</tr>
<tr>
<td>0x044</td>
<td>uip</td>
<td>User interrupt pending.</td>
</tr>
</tbody>
</table>
How all it comes together: 
MultiZone™ Trusted Execution Environment

“MultiZone it’s like Docker at the chip level – just way better”

- Unlimited number of equally secure zones – ram, rom, i/o
- Hardware-enforced, Software-enabled, Policy-driven RWX
- Supports fencing and user mode interrupts – even without “N”
- Formally verifiable nanoKernel completely written in assembly
- Consumes <1% of cpu and <1KB of ram, fits every boot rom
- Does not require changes to existing code – elf / hex sources
- Built from the ground up for RISC-V – no clunky legacy porting
- Completely based on RISC-V standard extensions
- Free and open source API under Apache v2 available on GitHub
MultiZone™ Trusted Execution Environment

Enable **proper** implementation of all typical security components for RISC-V systems

- OOB pre-certified implementations – wrap traditional APIs with InterZone™ Messages
- Open standards interoperable solution – OEMs focus on differentiation
- Low friction path to market – Hex Five in every RISC-V boot ROM, SDK and custom design IP

“the design complexity associated with properly implementing [these] security technologies often results in them not being used at all” - Brandon Lewis, Editor-in-Chief, Embedded Computing Design
# Copyright(C) 2018 Hex Five Security, Inc. - All Rights Reserved

# Kernel

tick = 10 # ms
rtc = 32768 # Hz
sched = PREEMPT # PREEMPT | COOP

# Zone 1
mz1_fence = FENCE
mz11_base = 0x40410000; mz11_size = 64K; mz11_rwx = RX # FLASH
mz12_base = 0x80001000; mz12_size =  4K; mz12_rwx = RW # RAM
mz13_base = 0x20000000; mz13_size =  32; mz13_rwx = RW # UART

# Zone 2
mz2_irq = 11, 21, 22 # BTN0 BTN1 BTN2
mz21_base = 0x40420000; mz21_size = 64K; mz21_rwx = RX # FLASH
mz22_base = 0x80002000; mz22_size =  4K; mz22_rwx = RW # RAM
mz23_base = 0x8000FF8; mz23_size =  8; mz23_rwx = RO # RTC
mz24_base = 0x20005000; mz24_size =  64; mz24_rwx = RW # PWM
mz25_base = 0x20002000; mz25_size =  64; mz25_rwx = RW # GPIO
mz26_base = 0x0C000000; mz26_size =  4M; mz26_rwx = RW # PLIC

# Zone 3
mz3_irq = 23 # BTN3
mz31_base = 0x40430000; mz31_size = 64K; mz31_rwx = RX # FLASH
mz32_base = 0x80003000; mz32_size =  4K; mz32_rwx = RW # RAM
mz33_base = 0x8000FF8; mz33_size =  8; mz33_rwx = RO # RTC
mz34_base = 0x20002000; mz34_size =  64; mz34_rwx = RW # GPIO
MultiZone™ Free and Open API

/* Copyright 2018 Hex Five Security, Inc.
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or imp!
See the License for the specific language governing permissions and
limitations under the License. */

#include "plic.h"

void button_0_handler(void) __attribute__((interrupt("user")));
void button_0_handler(void){
  plic_source int_num = PLIC_claim_interrupt(&g_plic);  // claim
  LED1_GRN_ON; LED1_RED_OFF; LED1_BLU_OFF;
  GPIO_REG(GPIO_RISE_IE) |= (1<<BUTTON_0_OFFSET);  // clear gpio irq
  PLIC_complete_interrupt(&g_plic, int_num);        // complete
  ECALL_SEND((int[]){201,0,0,0});
}

/*configures Button0 as a global gpio irq*/
void b0_irq_init() {
  // disable hw io function
  GPIO_REG(GPIO_IOF_EN) &= ~(1 << BUTTON_0_OFFSET);
  // set to input
  GPIO_REG(GPIO_INPUT_EN) |= (1<<BUTTON_0_OFFSET);
  GPIO_REG(GPIO_PULLUP_EN) |= (1<<BUTTON_0_OFFSET);
  // set to interrupt on rising edge
  GPIO_REG(GPIO_RISE_IE) |= (1<<BUTTON_0_OFFSET);

  ECALL_IRQ_VECT(11, button_0_handler);
}
MultiZone™ Reference Application

Zone 1
Terminal Console
Ilm + dlm + uart

Zone 2
irq demo app
Ilm + dlm + plic + gpio

Zone 3
real time ctrl app
Ilm + dlm + spi + gpio + rtc

InterZone™ Secure Communications
MultiZone™ nanoKernel

UART
SPI / USB

HEX-Five Security
Open Source License – Apache v2

Hex-Five multizone api is licensed under the Apache License 2.0

A permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.
Arms race: SiFive, Hex Five build code safe houses for RISC-V chips

Those developing custom CPUs can now tap a TrustZone-ish trusted execution environment

By Thomas Claburn in San Francisco 10 Sep 2018 at 20:08

Hex Five adds MultiZone Security to the Andes RISC-V Cores on GOWIN FPGAs

The first RISC-V Trusted Execution Environment Enabling Robust Security for Andes N(X)25 RISC-V Cores on the GOWIN GW-2A Family of FPGAs

November 09, 2018 11:00 ET | Source: Andes Technology Corporation

San Jose, CA, Nov. 09, 2018 (GLOBE NEWSWIRE) -- Hex Five Security, Inc. the creator of MultiZone™ Security, Andes Technology Corporation and GOWIN Semiconductor Corp announce a collaboration to enable MultiZone™ Security, the first Trusted Execution Environment for RISC-V on the Andes N(X)25 RISC-V Cores, which is part of 25-series, with the GOWIN GW-2A Family of FPGAs.

Hex Five’s patent pending technology provides policy-based hardware-enforced separation for an unlimited number of security zones, with full control over data, code, interrupts and peripherals. MultiZone’s™ Configurator takes fully compiled and linked customer code and merges it with Hex Five’s nanoKernel to enable rapid adoption without any changes to hardware or customer code bases.

Andes AX25, AX25F, and AX25A are versatile CPU cores, compliant in RISC-V ISA that deliver 3.5...
Takeaways

- Embedded systems – with or without MMU – are inherently not secure as all code can access all data and peripherals.
- The RISC-V ISA defines some security building blocks including privileged modes, physical memory protection and user-mode interrupts.
- However, the design complexity associated with properly implementing these technologies often results in them not being used at all.

1. MultiZone™ Security provides an unlimited number of equally secure execution environments.
2. MultiZone™ Security provides hardware-enforced policy-defined separation for code, data and I/O.
3. MultiZone™ Security doesn’t require additional cores, specialized IP or changes to existing applications.
Hex Five Security, Inc. is the creator of MultiZone™, the first Trusted Execution Environment for RISC-V. Hex Five patent pending technology provides policy-based hardware-enforced separation for an unlimited number of security domains, with full control over data, code and peripherals. Contrary to traditional solutions, Hex Five MultiZone™ requires no additional cores, specialized hardware or changes to existing software. Open source libraries, third party binaries and legacy code can be configured in minutes to achieve unprecedented levels of safety and security.
Hex Five - MultiZone™ Security

Hex Five Security, Inc. is the creator of MultiZone™ Security, the first Trusted Execution Environment (TEE) for RISC-V. Hex Five’s patent pending technology provides policy-based hardware enforced separation for an unlimited number of security domains, with full control over data, code and peripherals.

What is MultiZone™ Security?

MultiZone™ Security is the first Trusted Execution Environment for RISC-V – it enables development of a simple, policy-based security environment for RISC-V that supports rich operating systems through to bare metal code. MultiZone™ Security consists of the following components:

- **MultiZone™ Nonkernel** - lightweight, verifiable, bare metal kernel providing policy-driven hardware-enforced separation of ram, rom, i/o and interrupts.
- **InterZone™ Messenger** - communications infrastructure to exchange secure messages across zones on a no-shared memory basis.
- **MultiZone™ Configurator** - combines fully linked zone executables with policies and kernel to generate the signed firmware image.
- **MultiZone™ Signed Boot** - 2-stage signed boot loader to verify integrity and authenticity of the firmware image (sha-256 / ECC).

How does MultiZone™ Security Integrate with Existing Development Environment?

MultiZone™ Security integrates seamlessly into your existing IDE such as Eclipse or command line based toolset. Application blocks are written, compiled and linked separately for each zone producing a set of elf or hex file. MultiZone™ Policies are set to achieve the desired ram, rom, i/o and interrupt isolation for each zone – RWK, with granularity down to 4 bytes.

Finally the MultiZone™ Configurator is invoked to combine zone elf/hex files with the nanokernel and bootloader into a signed firmware image. The full system can be written, compiled and debugged with your existing GNU or Eclipse toolset.

Features

- Preemptive real time scheduler: round robin / cooperative, configurable time tick, cpu overhead < 1%
- Formally verifiable, completely written in assembly, self-contained - no 3rd party library dependencies
- Unlimited number of isolated Trusted Execution Environments (zones) - hardware-enforced, policy-defined
- Up to 32 memory-mapped resources per zone – i.e. flash, ram, i/o, uart, gpio, timers, etc.
- Any combination of top-of-range and naturally aligned configuration – minimum granularity 4 bytes
- Any combination of read, write, execute policy – resource overlapping allowed although not recommended
- Built-in support for fencing configurations on a per-zone basis – i.e. cache / pipeline / instruction / load / store
- Full support for PLIC and CLIC interrupts – fully configurable zone / interrupt mapping
- Full support for secure user-mode interrupt handlers – even without ‘N’ extensions
- Full support for low-latency vectored interrupts, preemptable interrupts, and Wait For Interrupt - suspend mode
- Built in trap & emulate for most protected instructions – i.e. CSR read only
- Secure inter-zone communications infrastructure based on messaging - no shared memory / buffers
- C library wrapper for protected mode execution – via ECALL exception handling mechanism
- Signed boot suitable for 2-stage boot room and/or public key / root of trust / PUF – SHA-256 / ECC
- Command line configurator utility compatible with any operating system capable of running Java 1.8

Development Environments

- Eclipse IDE including MCU and GNU Toolchain plugins and OpenOCD / JTAG / GDB live debugging
- SiFive FreedomStudio IDE including MCU and GNU Toolchain plugins and OpenOCD / JTAG / GDB live debugging
- SiFive Freedom E SDK – command line based
- Linux and Windows command line tools (make, gcc, gdb, etc.) – native Linux, Java 1.8 required for Windows
- Built-in Board Support Packages for SiFive E31 and E21 bitstreamers (Xilinx ARTY FPGA)

System Requirements

- 32 bit or 64 bit RISC-V ISA with ’S’ or ‘U’ extensions
- Physical Memory Protection compliant with Ver. 1.10
- 4GB FLASH and 1KB RAM

Hex Five Security is a proud member of the RISC-V Foundation