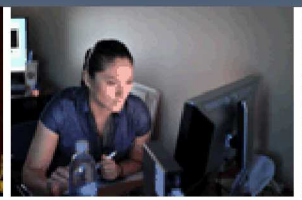
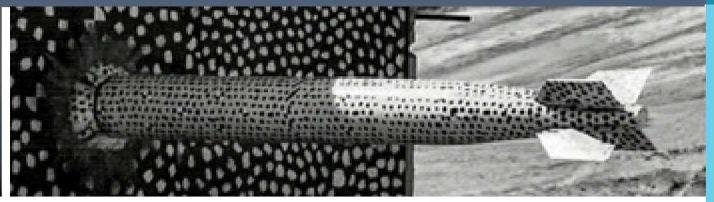
SAND2019-10336PE

Firmware Emulation for Embedded Systems





Presented by

Sri Cherukuri

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Embedded Systems

- Physical devices can be difficult to test thoroughly for faults
  - Hardware required for testing
  - Difficult to replicate or repeat quickly for multiple devices

Emulation

- Mimic device or chip architecture/memory structure to run programs on virtual hardware
- Testing is dependent on sufficient processing power and memory rather than availability of the physical system
- Difficult to detect which device failed where in a system crash

Difficulties in Emulation

- Emulating specific architectures requires heavy understanding of their inner workings
  - Not all devices of interest have documentation made available to emulated properly
- Each device must be implemented accordingly for the system to be emulated
- Difficult to tell if a fault occurred due to emulation fault or actual device fault

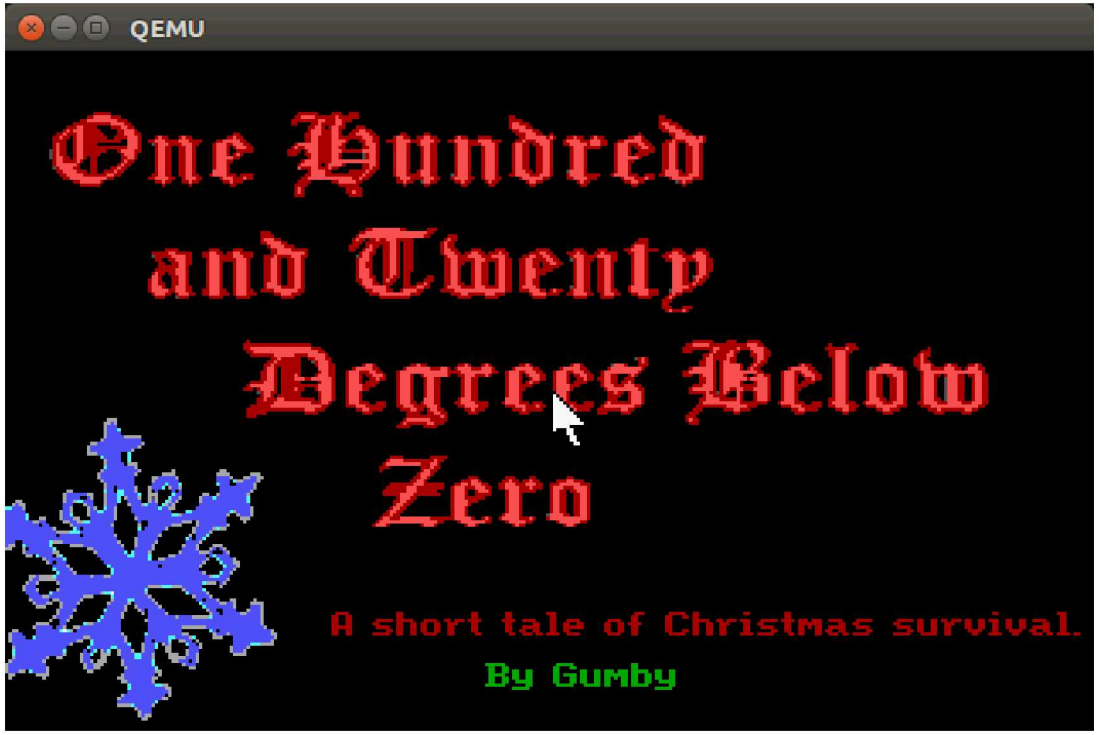
Investigation in Device Removal

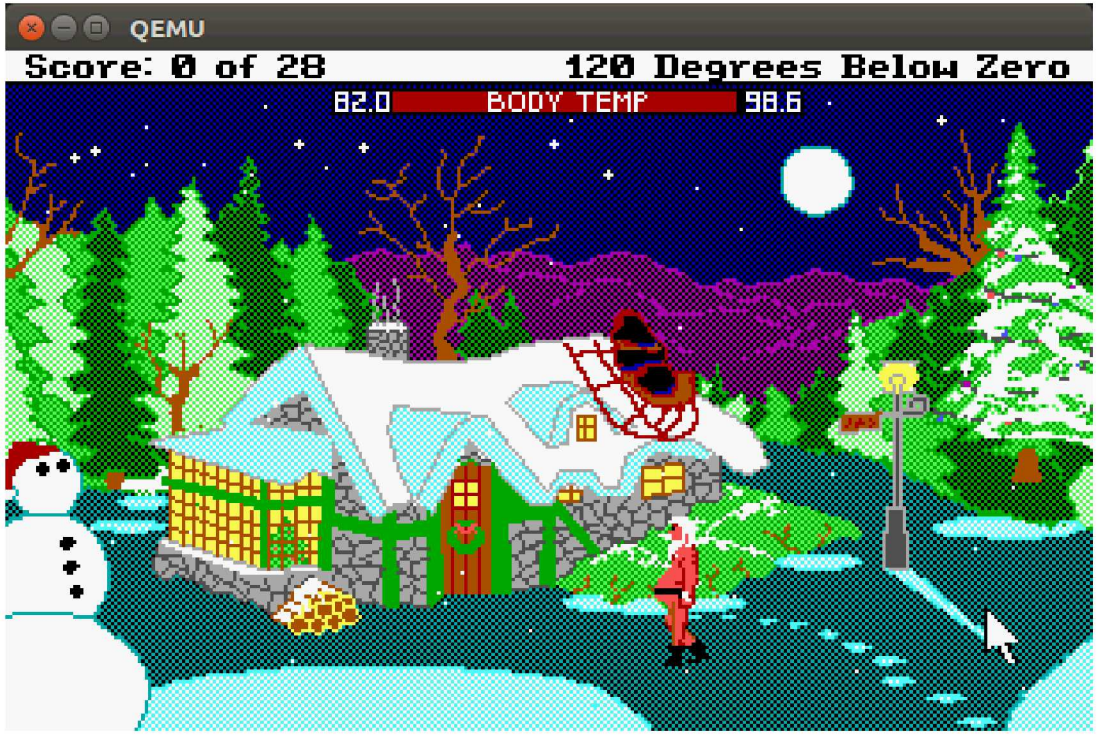
- Use known firmware and remove devices for analysis
- Use break locations in runtime to identify removed devices and their involvement in program
  - Essentially creating problems to solve one at a time rather than have several from different devices all at once
- Find patterns device failures in system failures across known and unknown systems
  - Error found when removing a device in a known emulation could allow us to identify similar errors in other devices

Methods

- QEMU
  - open-source emulator
  - Converts binary of source instructions to target instructions (ex: 32-bit ARM binary to x64 Linux)
- Apply to known architecture (ARM)
  - Use firmware and images from the QEMU advent calendar as a testing corpus

QEMU Advent Calendar





QEMU Advent Calendar

Attempted Implementation (pruning after initialization)

```
dev = qdev_create(NULL, "versatile_pci");
busdev = SYS_BUS_DEVICE(dev);
qdev_init_nofail(dev);
sysbus_mmio_map(busdev, 0, 0x10001000); /* PCI controller regs */
sysbus_mmio_map(busdev, 1, 0x41000000); /* PCI self-config */
sysbus_mmio_map(busdev, 2, 0x42000000); /* PCI config */
sysbus_mmio_map(busdev, 3, 0x43000000); /* PCI I/O */
sysbus_mmio_map(busdev, 4, 0x44000000); /* PCI memory window 1 */
sysbus_mmio_map(busdev, 5, 0x50000000); /* PCI memory window 2 */
sysbus_mmio_map(busdev, 6, 0x60000000); /* PCI memory window 3 */
sysbus_connect_irq(busdev, 0, sic[27]);
sysbus_connect_irq(busdev, 1, sic[28]);
sysbus_connect_irq(busdev, 2, sic[29]);
sysbus_connect_irq(busdev, 3, sic[30]);
pci_bus = (PCIBus *)qdev_get_child_bus(dev, "pci");

for(n = 0; n < nb_nics; n++) {
    nd = &nd_table[n];

    if (!done_smc && (!nd->model || strcmp(nd->model, "smc91c111") == 0)) {
        smc91c111_init(nd, 0x10010000, sic[25]);
        done_smc = 1;
    } else {
        pci_nic_init_nofail(nd, pci_bus, "rtl8139", NULL);
    }
}
if (machine_usb(machine)) {
    pci_create_simple(pci_bus, -1, "pci-ohci");
}
n = drive_get_max_bus(IF_SCSI);
while (n >= 0) {
    lsi53c895a_create(pci_bus);
    n--;
}

pl011_create(0x101f1000, pic[12], serial_hd(0));
pl011_create(0x101f2000, pic[13], serial_hd(1));
pl011_create(0x101f3000, pic[14], serial_hd(2));
pl011_create(0x10009000, sic[6], serial_hd(3));

sysbus_create_simple("pl080", 0x10130000, pic[17]);
sysbus_create_simple("sp804", 0x101e2000, pic[4]);
sysbus_create_simple("sp804", 0x101e3000, pic[5]);

sysbus_create_simple("pl061", 0x101e4000, pic[6]);
sysbus_create_simple("pl061", 0x101e5000, pic[7]);
sysbus_create_simple("pl061", 0x101e6000, pic[8]);
sysbus_create_simple("pl061", 0x101e7000, pic[9]);
```

```
Bus: System
  Dev Type: cfi.pflash01        Path: /machine/unattached/device[29]
  Dev Type: pl041       Path: /machine/unattached/device[28]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: versatile_i2c       Path: /machine/unattached/device[26]
    Bus: i2c-bus
      Dev Type: ds1338  Path: /machine/unattached/device[27]
  Dev Type: pl031       Path: /machine/unattached/device[25]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl181       Path: /machine/unattached/device[23]
    IRQ : (unnamed)     #In: 0  #Out:2
    IRQ : sysbus-irq    #In: 0  #Out:2
  Dev Type: pl181       Path: /machine/unattached/device[21]
    IRQ : (unnamed)     #In: 0  #Out:2
    IRQ : sysbus-irq    #In: 0  #Out:2
  Dev Type: pl110_versatile     Path: /machine/unattached/device[20]
    IRQ : (unnamed)     #In: 1  #Out:0
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl061       Path: /machine/unattached/device[19]
    IRQ : (unnamed)     #In: 8  #Out:8
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl061       Path: /machine/unattached/device[18]
    IRQ : (unnamed)     #In: 8  #Out:8
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl061       Path: /machine/unattached/device[17]
    IRQ : (unnamed)     #In: 8  #Out:8
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl061       Path: /machine/unattached/device[16]
    IRQ : (unnamed)     #In: 8  #Out:8
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: sp804       Path: /machine/unattached/device[15]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: sp804       Path: /machine/unattached/device[14]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl080       Path: /machine/unattached/device[13]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl011       Path: /machine/unattached/device[12]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl011       Path: /machine/unattached/device[11]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl011       Path: /machine/unattached/device[10]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl011       Path: /machine/unattached/device[9]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: versatile_pci       Path: /machine/unattached/device[6]
    IRQ : sysbus-irq    #In: 0  #Out:4
    Bus: PCI
      Dev Type: lsi53c895a      Path: /machine/unattached/device[8]
        Bus: SCSI
          Dev Type: scsi-disk   Path: /machine/unattached/device[8]/scsi.0/legacy[2]
      Dev Type: versatile_pci_host      Path: /machine/unattached/device[7]
  Dev Type: pl050_mouse Path: /machine/unattached/device[5]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: pl050_keyboard      Path: /machine/unattached/device[4]
    IRQ : sysbus-irq    #In: 0  #Out:1
  Dev Type: versatilepb_sic     Path: /machine/unattached/device[3]
    IRQ : sysbus-irq    #In: 0  #Out:32
    IRQ : (unnamed)     #In: 32 #Out:0
  Dev Type: pl190       Path: /machine/unattached/device[2]
    IRQ : sysbus-irq    #In: 0  #Out:2
```

Potential Alternative (start from scratch)

- Start with base avatar CPU in question
  - Manually implement processor board
  - Implement connected devices with desired memory mapping such that startup removal is easier to configure

Successful Implementation (blacklist initialization)

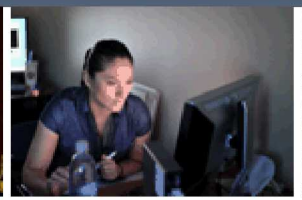
```
/* Memory map for Versatile/PB:  */
/* 0x10000000 System registers.  */
/* 0x10001000 PCI controller config registers.  */
/* 0x10002000 Serial bus interface.  */
/*  0x10003000 Secondary interrupt controller.  */
/* 0x10004000 AACI (audio).  */
/*  0x10005000 MMCI0.  */
/*  0x10006000 KMI0 (keyboard).  */
/*  0x10007000 KMI1 (mouse).  */
/* 0x10008000 Character LCD Interface.  */
/*  0x10009000 UART3.  */
/* 0x1000a000 Smart card 1.  */
/*  0x1000b000 MMCI1.  */
/*  0x10010000 Ethernet.  */
/* 0x10020000 USB.  */
/* 0x10100000 SSMC.  */
/* 0x10110000 MPMC.  */
/*  0x10120000 CLCD Controller.  */
/*  0x10130000 DMA Controller.  */
/*  0x10140000 Vectored interrupt controller.  */
/* 0x101d0000 AHB Monitor Interface.  */
/* 0x101e0000 System Controller.  */
/* 0x101e1000 Watchdog Interface.  */
/* 0x101e2000 Timer 0/1.  */
/* 0x101e3000 Timer 2/3.  */
/* 0x101e4000 GPIO port 0.  */
/* 0x101e5000 GPIO port 1.  */
/* 0x101e6000 GPIO port 2.  */
/* 0x101e7000 GPIO port 3.  */
/* 0x101e8000 RTC.  */
/* 0x101f0000 Smart card 0.  */
/*  0x101f1000 UART0.  */
/*  0x101f2000 UART1.  */
/*  0x101f3000 UART2.  */
/* 0x101f4000 SSPI.  */
/* 0x34000000 NOR Flash */
```

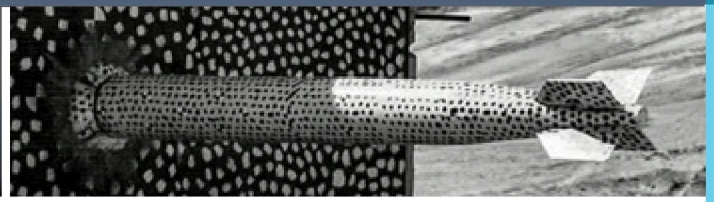
```
0x10000000      mapped
0x10140000      mapped
0x10003000      mapped
0x10006000      mapped
0x10007000      skipped
0x10001000      mapped
0x41000000      mapped
0x42000000      mapped
0x43000000      mapped
0x44000000      mapped
0x50000000      mapped
0x60000000      mapped
0x101f1000      mapped
0x101f2000      mapped
0x101f3000      mapped
0x10009000      mapped
0x10130000      mapped
0x101e2000      mapped
0x101e3000      mapped
0x101e4000      mapped
0x101e5000      mapped
0x101e6000      mapped
0x101e7000      mapped
0x10120000      mapped
0x10005000      mapped
0x1000b000      mapped
0x101e8000      mapped
0x10002000      mapped
audio: Could not init `oss' audio driver
0x10004000      mapped
0x34000000      mapped
```

Further Investigations Using Device Blacklist

- Find patterns in device failures
- Automate identification of minimum system requirements
  - Test generated blacklists to find which devices can be removed without system failure

Emulation and Binary Analysis





Presented By

Christopher Wright

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Contributions

- Systemization of Knowledge
  - Working examples with multiple tools
  - Summarizing Emulation Challenges
- Loop Analysis
  - LLVM using RetDec
  - Angr
  - Ghidra
- Escapes Analysis
  - Ghidra

Overview

- Background and Motivation
- Tools
- Decompilation
- Analysis
- Future Work

Why Emulation?

- System Migration/Testing
- Bug Finding
- Reverse Engineering
- Verification

Emulation Challenges

- Pre Emulation
- Emulation
  - Setup
  - Execution
- Post Emulation

Overview

- Background and Motivation
- Tools
- Decompilation
- Analysis
- Future Work

Tools

- QEMU
- SIMICS
- AVATAR
- PANDA
- RetDec
- angr
- IDA PRO
- GHIDRA

Tools

QEMU

- Machine Emulator (Hardware Virtulization)
  - CPU Emulator
  - Emulated Devices(VGA, Hard Disk)
  - Generic Devices (Network Devices)
  - Machine Descriptions
  - Debugger
  - User Interface

Tools

RetDec – Retargetable Decompiler

- Current Support:
  - 32-bit: Intel x86, ARM, MIPS, PIC32, PowerPC
  - 64-bit: x86-64
  - ELF, PE, Mach-O, COFF, AR, Intel HEX, raw
- LLVM-IR

Tools

angr – Because binary analysis makes you angry

- Python Framework
- Static and Dynamic symbolic analysis
  - Control flow graph
  - ROP chains
  - Binary hardening
- Uses
  - CLE, archinfo, PyVex, Claripy
  - angr analysis suite
    - BB level

Tools

Ghidra – NSA Reverse Engineering Tool

- IDA Pro competitor
- Partially open-sourced, >1M code lines for the framework
- Disassembly, assembly, decompilation, scripting, graphing, etc.
- Java or Python (Jython)
- Intermediate code is PCODE
  - Simple, very limited number of instructions
  - Relatively easy to add more architectures to the lifter

Overview

- Background and Motivation
- Tools
- Decompilation
- Analysis
- Future Work

Analysis

- Using LLVM
  - Relies on RetDec
- Angr
  - Static analysis
  - Dynamic
- Using Ghidra
  - Uses Ghidra decompiler
  - Can plugin IDA Pro if you have access

Analysis (LLVM)

- Decompile with RetDec
- Function pass
  - Find loops
  - Get condition instruction
  - Check if infinite loop
    - Essentially get def-use chains for condition instruction, modified analysis on any instruction in the loop that can taint any of those instructions
- Failures
  - Only works if RetDec works in correct predictable manner

Analysis (angr)

- Use angr built-in disassembler (Uses Capstone)
- BB level analysis
  - Harder to do instruction level analysis
- Finding loops was simple
  - I failed to integrate dynamic analysis/symbolic execution after finding loops
  - Requires providing initial state or running from the beginning of an executable
    - Would not work well for a larger firmware

Analysis (Ghidra)

- Loop Analysis
  - PCODE analysis as well as assembly instruction level analysis
    - Small issues with either, though both worked well enough is our use cases
  - Simplified Loop Finding
    - 2 Approaches:
      - Find conditional branching instructions and if target address was smaller, it was end of a loop
      - BFS on the instructions
  - After Loop found
    - Convert loop to PCODE, do a backward analysis from the branching condition on loop, check modification of any variable in active set

Analysis (Ghidra)

- Escapes Analysis
  - Function analysis
  - Check each instruction in the function
    - If it is Persistent and Address Tied, then it escapes the function
      - Not perfect either, but gets most cases
      - If we only care about persistent, all functions return a persistent value
      - If we only care about address tied, it will use Ghidras Unique() values that are really just locals
      - Requiring both gives most of what we want

Overview

- Background and Motivation
- Tools
- Decompilation
- Analysis
- Future Work

Future Work

- Finish Systemization of Knowledge Paper
- Combine Loop and Escapes Analysis and add auto-patching
- Add more analyses
- Emulation platforms