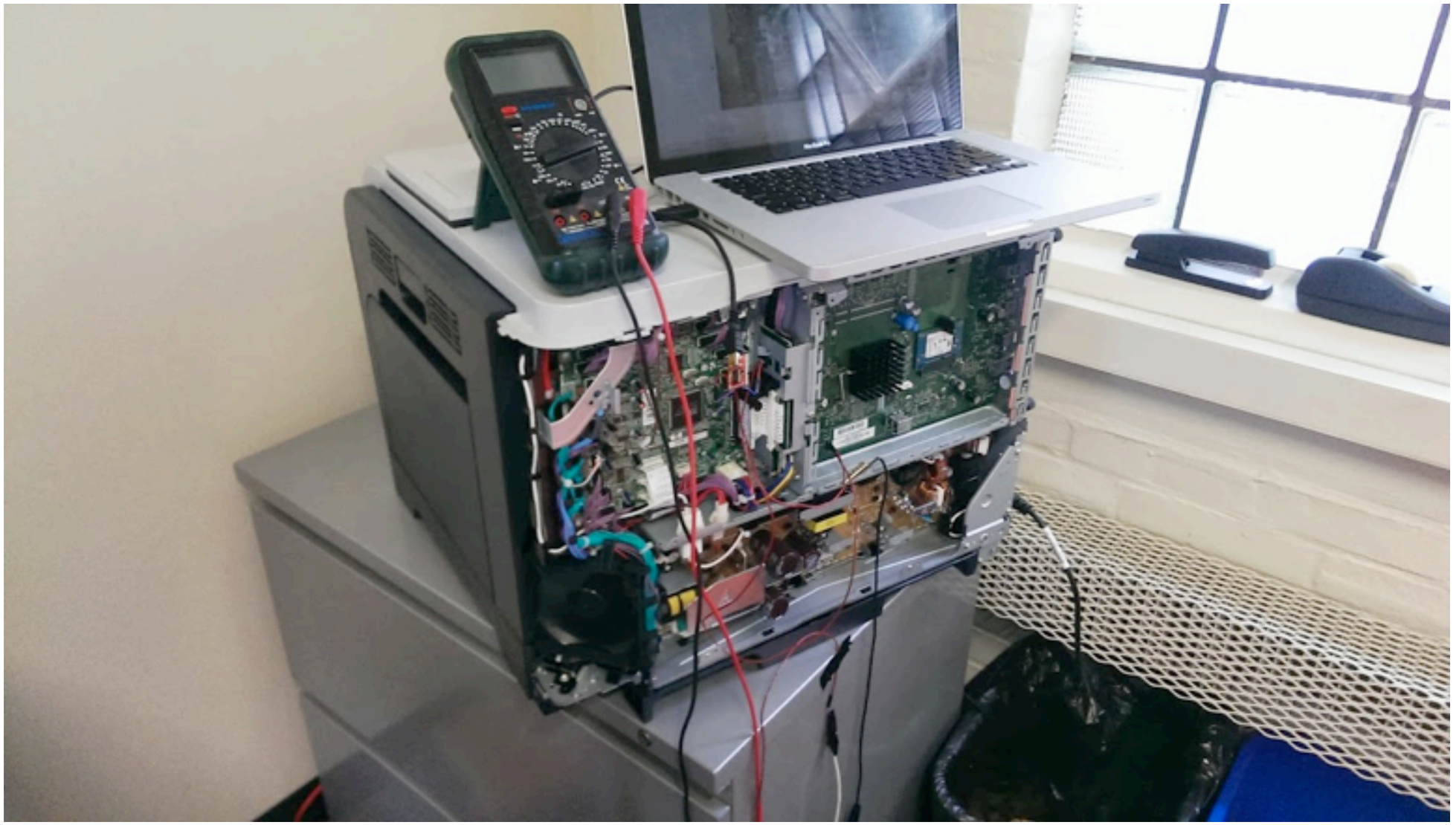
Rehosting Embedded Devices



BeaverWorks Lightning Talk
7/15/2013

Embedded Security

- Embedded devices: everywhere
- Their security: terrible (probably)
- Current state of the art: extract firmware, do (mostly) manual static analysis

Rehosting

- Want to do dynamic analysis: run embedded firmware in PANDA
- QEMU has great CPU support :)
- QEMU has poor peripheral support :(
- Goal: tools to assist creation of embedded device peripheral models in QEMU

Trail of Breadcrumbs

- Inspired by Jin & Orso's BugRedux
- Idea: use device output (serial port) to infer "correct" path through program
  - **NOTE**: Static analysis required!
- Try to solve for unknowns (device inputs) and get an execution that fits path

Breadcrumbs

```
seg002:8005AAD0 70 40 2D E9     STMFD   SP!, {R4-R6,LR}
seg002:8005AAD4 00 50 A0 E1     MOV     R5, R0
seg002:8005AAD8 74 31 9F E5     LDR     R3, =0xBF030100
seg002:8005AADC A0 10 95 E5     LDR     R1, [R5,#0xA0]
seg002:8005AAE0 68 01 9F E5     LDR     R0, =aRamSize0x08x ; "RAM Size=0x%08x\r\n"
seg002:8005AAE4 00 20 93 E5     LDR     R2, [R3]
seg002:8005AAE8 FF EC A0 E3     MOV     LR, #0xFF00
seg002:8005AAEC FF 30 8E E3     ORR     R3, LR, #0xFF
seg002:8005AAF0 03 60 02 E0     AND     R6, R2, R3
seg002:8005AAF4 D8 16 00 EB     BL      DbgPrintf
seg002:8005AAF8 4C 01 9F E5     LDR     R0, =aAsicid0x08x ; "asicID=0x%08x\r\n"
seg002:8005AAFC 06 10 A0 E1     MOV     R1, R6
seg002:8005AB00 D5 16 00 EB     BL      DbgPrintf
```

```
WindowsCE on ARM

ARM Cortex-A8 CPU
Stack Pointer: 0x00007ffc
Windows CE Kernel for ARM (Thumb Enabled) Built on Jan 26 2012 at 21:54:55
ProcessorType=0c08  Revision=1 CpuId=0x412fc081
OEMAddressTable = 8005923c
OEMInit (db)
Trace system activeRAM Size=0x40cbe000
asicID=0x00002600
Jedi Memory Pool: Size=0x2D000000 Start=0x12D88000
```

Breadcrumbs

```
seg002:8005AAD0 70 40 2D E9     STMFD   SP!, {R4-R6,LR}
seg002:8005AAD4 00 50 A0 E1     MOV     R5, R0
seg002:8005AAD8 74 31 9F E5     LDR     R3, =0xBF030100
seg002:8005AADC A0 10 95 E5     LDR     R1, [R5,#0xA0]
seg002:8005AAE0 68 01 9F E5     LDR     R0, =aRamSize0x08x ; "RAM Size=0x%08x\r\n"
seg002:8005AAE4 00 20 93 E5     LDR     R2, [R3]
seg002:8005AAE8 FF EC A0 E3     MOV     LR, #0xFF00
seg002:8005AAEC FF 30 8E E3     ORR     R3, LR, #0xFF
seg002:8005AAF0 03 60 02 E0     AND     R6, R2, R3
seg002:8005AAF4 D8 16 00 EB     BL      DbgPrintf
seg002:8005AAF8 4C 01 9F E5     LDR     R0, =aAsicid0x08x ; "asicID=0x%08x\r\n"
seg002:8005AAFC 06 10 A0 E1     MOV     R1, R6
seg002:8005AB00 D5 16 00 EB     BL      DbgPrintf
```

```
WindowsCE on ARM

ARM Cortex-A8 CPU
Stack Pointer: 0x00007ffc
Windows CE Kernel for ARM (Thumb Enabled) Built on Jan 26 2012 at 21:54:55
ProcessorType=0c08  Revision=1 CpuId=0x412fc081
OEMAddressTable = 8005923c
OEMInit (db)
Trace system activeRAM Size=0x40cbe000
asicID=0x00002600
Jedi Memory Pool: Size=0x2D000000 Start=0x12D88000
```

Breadcrumbs

```
seg002:8005AAD0 70 40 2D E9     STMFD   SP!, {R4-R6,LR}
seg002:8005AAD4 00 50 A0 E1     MOV     R5, R0
seg002:8005AAD8 74 31 9F E5     LDR     R3, =0xBF030100
seg002:8005AADC A0 10 95 E5     LDR     R1, [R5,#0xA0]
seg002:8005AAE0 68 01 9F E5     LDR     R0, =aRamSize0x08x ; "RAM Size=0x%08x\r\n"
seg002:8005AAE4 00 20 93 E5     LDR     R2, [R3]
seg002:8005AAE8 FF EC A0 E3     MOV     LR, #0xFF00
seg002:8005AAEC FF 30 8E E3     ORR     R3, LR, #0xFF
seg002:8005AAF0 03 60 02 E0     AND     R6, R2, R3
seg002:8005AAF4 D8 16 00 EB     BL      DbgPrintf
seg002:8005AAF8 4C 01 9F E5     LDR     R0, =aAsicid0x08x ; "asicID=0x%08x\r\n"
seg002:8005AAFC 06 10 A0 E1     MOV     R1, R6
seg002:8005AB00 D5 16 00 EB     BL      DbgPrintf
```

```
WindowsCE on ARM

ARM Cortex-A8 CPU
Stack Pointer: 0x00007ffc
Windows CE Kernel for ARM (Thumb Enabled) Built on Jan 26 2012 at 21:54:55
ProcessorType=0c08  Revision=1 CpuId=0x412fc081
OEMAddressTable = 8005923c
OEMInit (db)
Trace system activeRAM Size=0x40cbe000
asicID=0x00002600
Jedi Memory Pool: Size=0x2D000000 Start=0x12D88000
```

Device Memory

1. Execute concretely and gather a trace
2. Symbolically re-execute along the same path, making device memory symbolic
3. At first node in the path to the next breadcrumb we stop and ask solver for sat. assignment

Example

```
void OutputDebugChar(int c) {
    // Wait until serial port is ready
    while (ReadDword(0xbd370404) & 0x4 == 0) {
        // busy loop
    }
    WriteDword(0xbd370414, c);
}
```

Symbolic Input

Goal

```
IN:
0x0005cb08:  ldr   r0, [pc, #44]
0x0005cb0c:  bl    0x5cc24
----------------
IN:
0x0005cc24:  ldr   r0, [r0]
0x0005cc28:  bx    lr
----------------
IN:
0x0005cb10:  tst   r0, #4   ; 0x4
0x0005cb14:  beq   0x5cb08
```

```
============ tcg-llvm-tb-40-5cb10 ============
(Eq false
    (Eq 0
        (And w32 (ReadLSB w32 0 device memory @0xbd370404.0)
                 4)))
True:
device memory @0xbd370404.0: 04000000
```

Automatically Generated QEMU Devices

```
{
    "dev_0xbd370404" : {
        "description": "Automatically generated device at 0xbd370404",
        "base": 3174499328,
        "memory": [
            {"address": 3174499332, "values": [4], "size": 4}
        ]
    }
}
```

Problems on the Horizon

- Static values in memory are not enough
- Interrupts (timers, particularly)
- Complicated devices (HDDs, NICs)

Interrupts

- May not even know how to map interrupt –> handler
- How do we know what interrupts are necessary, and when? Try them all when stuck?

Complicated Devices

- Possibility: find a similar device already implemented by QEMU and make an adaptor
- Any way to do this automatically?