Embedded systems are hard to analyze

- On standard platforms, tools can take advantage of standard I/O interface
- Analysis tools like fuzzing and dynamic RE “just work”
- Embedded systems may have nonstandard (or unique) I/O interfaces
Testing firmware

Emulator

Hardware

Firmware

read(0x1000)
Firmware rehosting
A targeted approach to rehosting

- Key Insight: Firmware implicitly encodes expected hardware behavior
- What values need to be read from the device to read the boot point?
- Goal: Generate HW device that guides firmware towards the boot point
Jetset: targeted firmware rehosting
Searching for a boot path

- Jetset uses a guided-DFS to find a path to the boot point
Example: initializing UART and USB

USB:
    ...

UART:
    ...

FINISH_BOOT:
    ...

call print_boot_msg
Example: initializing UART and USB

USB:

mov ebx, [0x1000]; usb_present
cmp ebx, 0;
je UART;
call init_usb

UART:

mov ebx, [0x2000]; uart_present
cmp ebx, 0
je FINISH_BOOT

call init_uart

FINISH_BOOT:

mov ebx, [0x400000]; all_ok
cmp ebx, 0
je FAIL

call print_boot_msg
Example: initializing UART and USB

USB:
  mov ebx, [0x1000]; usb_present  
cmp ebx, 0;  
je UART;  
call init_usb

UART:
  mov ebx, [0x2000]; uart_present  
cmp ebx, 0  
je FINISH_BOOT  
call init_uart

FINISH_BOOT:
  mov ebx, [0x400000]; all_ok  
cmp ebx, 0  
je FAIL  
call print_boot_msg

Backtrack!
Jetset: targeted firmware rehosting
Generating device models

- Use SMT solver to generate satisfying trace that replays successful path
- Replies last value after reaching the boot point
- Generated device works on unmodified QEMU instance
Example: Generating device models for UART and USB

USB:

```
    mov ebx, [0x1000]; usb_present
    cmp ebx, 0;
    je UART;
    call init_usb
```

UART:

```
    mov ebx, [0x2000]; uart_present
    cmp ebx, 0
    je FINISH_BOOT
    call init_uart
```

FINISH_BOOT:

```
    mov ebx, [0x400000]; all_ok
    cmp ebx, 0
    je FAIL
    call print_boot_msg
```

Traces:

- 0x1000: 0x0
- 0x2000: 0x1
- 0x2004: 0x2 0x4 0x0f ...
- 0x2008: 0x33 ...
- 0x1000: 0x0
Jetset: targeted firmware rehosting

- Firmware binary
- Entry point
- Memory map

Jetset

Device inference → Device synthesis

QEMU

Synthetic devices
Evaluation: generating device models for realistic firmware

- 13 firmware targets (4 original, 9 from previous work)
- 3 different architectures (ARM, i386, m68k-coldfire)
- 4 operating systems: Linux, VRTX, RIOT, Arduino (+ 4 bare metal)
- Average synthesis time ~14 minutes (one subject took 2 hours 34 minutes)
Evaluation: using generated models for dynamic analysis

- We used Jetset’s generated models to fuzz CMU-900 and RPi2
- Found (not remotely exploitable) privilege escalation bug in the CMU-900
- Fuzzed syscall handlers of RPi2, to check that emulation had correct behavior
Summary

- Jetset uses directed symbolic execution to generate emulators for firmware
- Technique tested against several architectures and operating systems
- We used Jetset to find a bug in an otherwise untestable piece of firmware

https://jetset.aerosec.org