Grey Box Fuzzing the IoT

Program Monitoring for Memory and Power Constraint Embedded Devices

Kevin Läufer <laeufer@berkeley.edu>

Problems in Embedded Programming and Testing

- No memory protection:
  - all processes can access all memory
  - hard to detect and contain failures
- Still predominantly programmed in C
- No logging, no automatic stack traces
- Lacks tools for automated testing

Fuzzing Embedded Devices

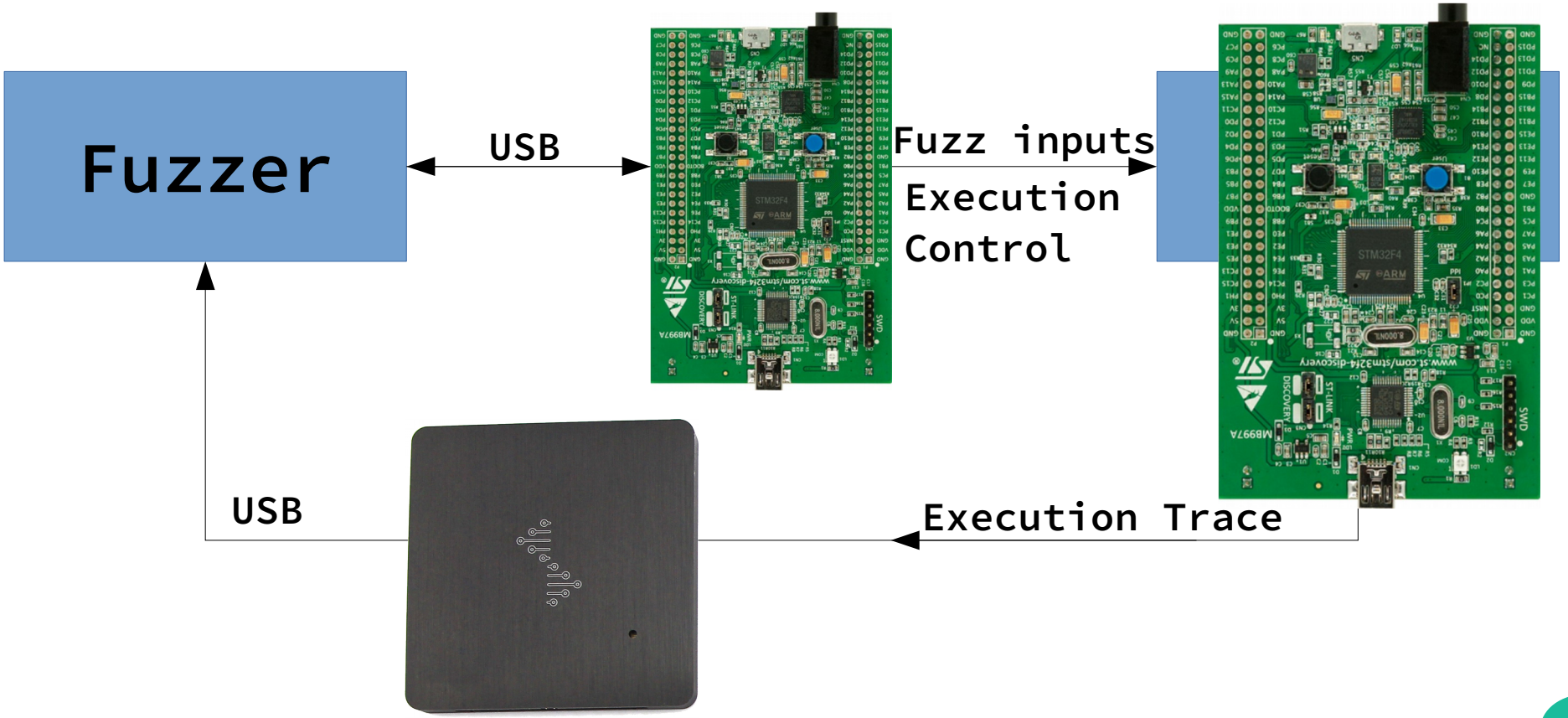
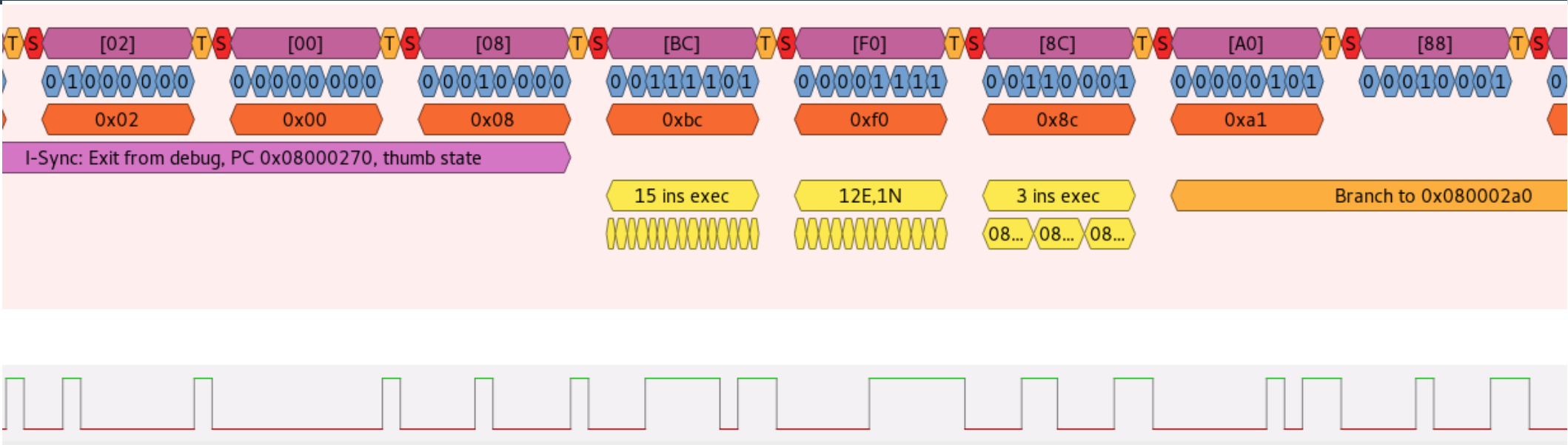


Image Source: seeedstudio.com

Image Source: STMicroelectronics

Fuzzing Embedded Devices



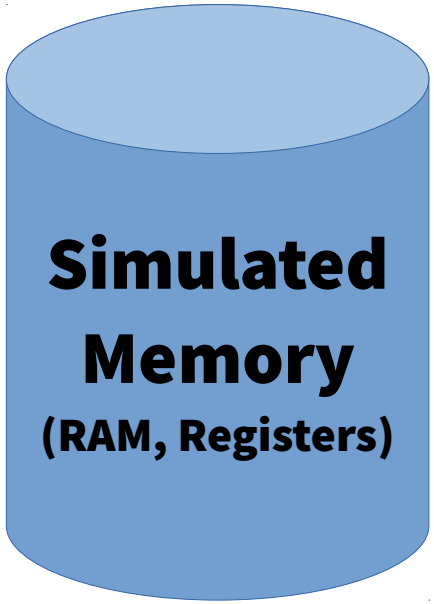
Trace Analysis Goals

- Memcheck: V(alid) bits
  - find use of uninitialized value
  - find double free, use after free
- Dynamic points to analysis:
  - detect buffer overflows

What’s new?

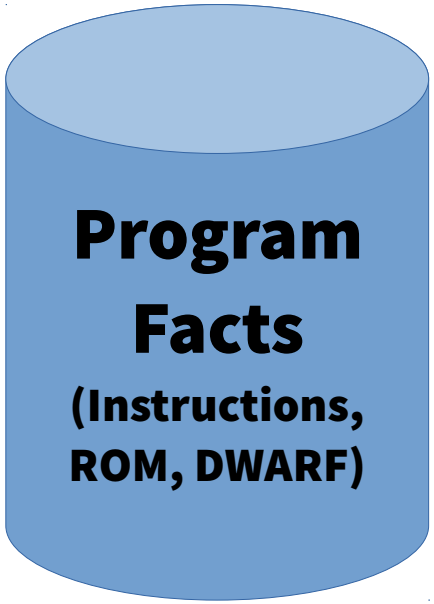
- Execution Engine based on radare2 intermediate language: ESIL
- Standardized Analysis Tool interface
- Meta data storage

Micro Controller Trace Analysis (UCTA)



Custom Analysis Tools

Execution
Engine



ESIL

- Stack based intermediate language
- Emitted by radare2
- radare2 emits wrong code for some ARM instructions
- Makes some analysis steps harder because intermediate values are saved to the stack
- Better separation of concerns

ESIL execution

```
for token in esil.split(','):
    if token in self.esil_commands:
        self.esil_commands[token](token, stack)
    else:
        stack.append((token, {'src': 'literal'}))
```

ESIL example

```
strb r4, [r3, 1]

r4,r3,1,+,=[1]

*(char*)(r3 + 1) = r4
```

ESIL bug

```
strb r4, [r3, 1]!
```

```
r4,r3,1,+,=[1],1,r3,+=
```

Analysis Tools Interface

```
class AnalysisTool:
    def __init__(self):
        self.step = SimulationStep.default()
    def next_step(self, step):
        self.step = step
    def on_store(self, addr, value):
        pass
    def on_load(self, addr, value):
        pass
    def on_assign_reg(self, reg, value):
        pass
    def on_compare(self, a, b, result):
        pass
    def on_binary_op(self, a, b, result):
        pass
    def on_unary_op(self, a, result):
        pass
    def on_load_from_reg(self, value, reg):
        pass
    def on_store_to_reg(self, value, reg):
        pass
```

Register Taint Analysis

```
class RegisterTainter(AnalysisTool):
    def on_load(self, addr, value):
        value[1]['regs'] = []
    def on_load_from_reg(self, value, reg):
        if isinstance(reg, tuple):
            reg = reg[0]
        value[1]['regs'] = [reg]
    def on_binary_op(self, a, b, result):
        a_regs = a[1]['regs'] if 'regs' in a[1] else []
        b_regs = b[1]['regs'] if 'regs' in b[1] else []
        result[1]['regs'] = a_regs + b_regs
```

Return Address Overwrite Analysis

```
class ReturnAddressOverwriteCheck(AnalysisTool):
    def __init__(self):
        self.return_addr_locs = {}
    def on_store(self, addr, value):
        if addr[0] in self.return_addr_locs:
            msg = "Return address overwriten with 0x{:08x} @ pc=0x{:08x}".format(value[0], self.step.instr['offset'])
            ii = self.return_addr_locs[addr[0]]
            msg += "; originally saved at instr_count={} pc=0x{:08x}".format(ii.instr_count, ii.instr['offset'])
            raise Exception(msg)
        elif 'lr' in value[1]['regs']:
            self.return_addr_locs[addr[0]] = self.step
    def on_load(self, addr, value):
        if addr[0] in self.return_addr_locs:
            del self.return_addr_locs[addr[0]]
```

```
Exception: Return address overwritten with
0x000000a4 @ pc=0x08000616; originally saved at
instr_count=11372 pc=0x080001a0
```

Dynamic Points to Analysis for Arrays on Stack: Idea Sketch

- Use debug information to determine location of arrays on stack frame
- Address always originates from Stack Pointer
- Tag address with array name and bounds once it is derived from the Stack pointer
- Throw exception if address is used to access memory outside array bounds

Buggy Function

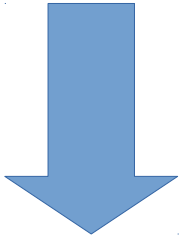
```
void buggy_function(const uint8_t* packet) {
   char buffer[8];
   const uint8_t length = packet[0];
   std::memcpy(buffer, packet + 1, length);
}
```

DWARF Debug Information

```
{ 'name': 'buggy_function',
  'file': '/home/kevin/d/ucta/program_under_test/main.cpp', 'line': 3,
  'lowpc': 0x080001a0, 'highpc': 0x080001c8,
  'return': uint8_t,
  'params': [
    { 'name': 'packet', 'location': {'mem': 'reg0'},
      'type': {'name': 'array', 'length': 8, 'base': uint8_t } },
  ],
  'vars': [
    { 'name': 'buffer', 'location': {'mem': 'stack', 'offset': -16},
      'type': {'name': 'array', 'length': 8,
               'base': unsigned_char_t } },
  ]
}
```

Address Loaded to Stack Pointer

```
0x080001a8 mov r0, sp
```



```
r0 <= ('buffer', 268438432, 268438440)
 @ pc=0x080001a8
```

Address Decrement and Moved to r3

```
0x0800060a subs r3, r0, 1
```

Load from address and pre increment

```
0x08000616 strb r4, [r3, 1]!
```

```
r3 points to ('buffer', 268438432, 268438440)
 → check if r3 >= 268438432 and r3 < 268438440
   When performing the store
```

Working

- Simulation engine based on ESIL
- Simulation can store meta data
- Flexible and composable analysis passes
- Return address overwrite analysis provides more information

Still Missing

- Load DWARF debug information from file
- Interrupt handling
- Implement dynamic points to analysis
- V(alid) bits for memcheck implementation
- Test with larger code base