Femto-Containers

Lightweight DevOps-style Virtual Machines on RIOT

Koen Zandberg
Overview

● How to maintain your deployment
● Virtual Machine Solutions
● The Linux solution: eBPF
● Femto-Containers
● Example: thread counter
● Limitations and Conclusion
How to maintain your deployment?

- Deploying IoT nodes at scale challenging.
- How about maintaining them in the field?
Current Issues

One of the devices in the field shows odd behaviour, can we debug this?

A customer needs modified behaviour on the deployed nodes.

A third party wants to run code on our devices.
Categories of Solutions

• Traditional solution: firmware updates
  – Simple, but has downsides
  – Maintaining and deploying another firmware version is costly

• Alternative solutions? Modular updates
  – Dynamic linking
  – Virtual Machines
  – ...
Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Limitations and Conclusion
Virtual Machine Solutions

- Python
- Javascript
- WebAssembly
- MicroEJ

- (And others)
Virtual Machine Solutions

Downsides: bulky to add for simple applications [1]

Measured on the nRF52840dk, Hosting engine only (RIOT 2021.4 release)

Sneak Peek

• Femto-containers:
  – Much smaller VMs!
  – Based on eBPF
  – Hosting engine:
    • 4.7 KiB ROM
    • 664 B of RAM

Measured [1] on the nRF52840dk (RIOT 2021.4 release)

Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Limitations and Conclusion
The Linux solution: eBPF

Event-driven in-kernel sandbox:
- Tracing
- Profiling
- Monitoring
- Network Protocol parsers
The Linux solution: eBPF

- in-kernel Virtual Machine:
  - 64 bit RISC architecture
  - Register based
  - 512 byte stack
- Allows for verification of loaded applications:
  - Application must halt
Overview

● How to maintain your deployment
● Virtual Machine Solutions
● The Linux solution: eBPF
● Femto-Containers
● Example: thread counter
● Limitations and Conclusion
Femto-Containers

- Simple virtual machine
- Hardware independent
- Short-lived, Event driven
- Integration with RIOT
- Based on Linux eBPF
- Minimal footprint
Femto-Containers

Why eBPF?

✓ 512 Byte stack
✓ Limited instruction set
✓ Secure by design

✗ 64 bit architecture
Femto-Containers: Isolation

- Sandboxed from the host
  - Pre-flight checks
  - Memory access guards
Femto-Containers: Events

Event triggered:
- Network
- USB
- System events
- Timers

Adding hooks is cheap
Femto-Containers: OS Interaction

- Context and return value
  - Packet and Allow/Reject
- Bindings
  - Calls to OS, e.g. saul_read
- Value store
  - Store simple values
Femto-Containers: Caveats

- Slow down
  - Virtual machine overhead
- Instruction set limitations
  - No indirect jumps
- Security and isolation
  - Basic security measures only
  - No formal verification (yet)

<table>
<thead>
<tr>
<th></th>
<th>App size</th>
<th>Startup time</th>
<th>Run time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native C</td>
<td>74 B</td>
<td>-</td>
<td>27 μs</td>
</tr>
<tr>
<td>WASM3</td>
<td>322 B</td>
<td>17 096 μs</td>
<td>980 μs</td>
</tr>
<tr>
<td>Femto-Containers</td>
<td>456 B</td>
<td>1 μs</td>
<td>2133 μs</td>
</tr>
<tr>
<td>JerryScript</td>
<td>593 B</td>
<td>5589 μs</td>
<td>14 726 μs</td>
</tr>
<tr>
<td>MicroPython</td>
<td>497 B</td>
<td>21 907 μs</td>
<td>16 325 μs</td>
</tr>
</tbody>
</table>

Fletcher32 startup time and run time [1]
Overview

- How to maintain your deployment
- Virtual Machine Solutions
- The Linux solution: eBPF
- Femto-Containers
- Example: thread counter
- Conclusion
Example

• Thread counter
  – Maintain thread run counters
  – Hooks into the scheduler
  – Store counters in the value store
Workflow

1) Write code
2) Compile
3) Transfer
4) Run

Code: C, C++, Rust
Compiler: LLVM
Bytecode
Transfer
Execution: VM Interpreter
Operating System
Sandbox
Maintainer PC
IoT Device
Workflow

- Write our code
  - C
  - Rust
  - TinyGo?
- Store the run count for each thread

```c
#include <stdint.h>
#include "bpf/bpfapi/helpers.h"

#define THREAD_START_KEY 0x0

typedef struct {
    uint64_t previous; /* previous thread */
    uint64_t next;    /* next thread */
} pid_ctx_t;

int pid_log(pid_ctx_t *ctx) {
    /* Zero pid means no next thread */
    if (ctx->next != 0) {
        uint32_t counter;
        uint32_t thread_key = THREAD_START_KEY +
                             ctx->next;
        bpf_fetch_global(thread_key, &counter);
        counter++;
        bpf_store_global(thread_key, counter);
    }
    return 0;
}
```
Workflow

● Compilation with LLVM
  - eBPF support
● RIOT bindings
Workflow

• Transfer the application:
  – CoAP
  – Bluetooth
  – Compile-in

• Independent of Femto-containers
Workflow

Start VM from RIOT

- Our code is compiled-in for simplicity
- RIOT executes the VM when switching threads
Workflow

• Run the code
• Query the value store counters

https://github.com/bergzand/RIOT/tree/wip/bpf/examples/rbpf_sched/
Overview

• How to maintain your deployment
• Virtual Machine Solutions
• The Linux solution: eBPF
• Femto-Containers
• Example: thread counter
• Conclusion
Conclusions

Rethink the cost of a VM on your RIOT device! Femto-Containers can provide:

• Customized behaviour
• Debugging
• Isolating code

With minimal impact on memory requirements.
Femto-Containers

Want to know more?

• Example:
  https://github.com/bergzand/RIOT/tree/wip/bpf/examples/rbpf_sched

• Tutorials:
  https://github.com/future-proof-iot/Femto-Container_tutorials

• Preprint:
  https://arxiv.org/abs/2106.12553
Thanks!