A Rust-based Runtime for the Internet of Things

Niklas Adolfsson
@niklasad1
2017-09-30
Who am I?

Niklas Adolfsson
Embedded Software Engineer, Cybercom
MSc. Computer Science
Software security, embedded systems, programming languages
What is this talk about?

My experience using Rust to build Bluetooth Low Energy firmware
What is Internet-of-Things?
Internet-of-Things characteristics

Low-end devices (low power, low cost)
Battery powered

Microcontroller characteristics:
- 1 CPU, tens of MHz
- Tens of kB RAM
- Hundreds of kB Flash
- Lack of Memory Manage Unit (MMU)
- Equipped with radio chips

nRF51-DK
Internet-of-Things - Bluetooth Low Energy

Low-end devices, low power

< 100 meters range

Two types of packets:
- Advertisements (31 bytes)
- Data packets (255 bytes)

<table>
<thead>
<tr>
<th>Application</th>
<th>Host Controller Interface (HCl)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GAP</td>
<td></td>
</tr>
<tr>
<td>SM</td>
<td>L2CAP</td>
</tr>
<tr>
<td></td>
<td>Link Layer (LL)</td>
</tr>
<tr>
<td></td>
<td>Physical Layer (PH)</td>
</tr>
<tr>
<td></td>
<td>GATT</td>
</tr>
<tr>
<td></td>
<td>ATT</td>
</tr>
</tbody>
</table>

Physical Layer (PH)
Link Layer (LL)
Host Controller Interface (HCI)
Link Layer (LL)
Physical Layer (PH)
Internet-of-Things characteristics - cont’d

Reliability important (limited access)

Security & privacy

Suitable programming languages:
  ● Low runtime overhead
  ● Fine-grained memory control
  ● Deterministic behavior

C/C++

nRF51-DK
Increasing complexity

Need a runtime

IoT operating systems:

- Trade-off memory & power over safety
- No memory isolation?
Faulty application - Will this segfault?

- Application
- Kernel
- Buffer overflow
- Same address space
No, it is WORSE than a segfault!!!
So, it's not a surprise that IoT is insecure

A buffer overflow flaw has been discovered by security researchers (on the IoT-focused security agency Senrio) in an open-source software program improvement library that's extensively utilized by main producers of the Internet-of-Thing devices.

The buffer overflow vulnerability (CVE-2017-9765), which is named "Devil's Ivy" permits a distant attacker to crash the SOAP (Simple Object Access Protocol) WebServices daemon and make it doable to execute arbitrary code on the affected devices.

"The affect of Devil's Ivy goes far past Axis. It lies deep within the communication layer, in an open third-party toolkit known as gSOAP (Simple Object Access Protocol). gSOAP is a extensively used internet companies toolkit, and builders all over the world use gSOAP as half of a software program stack to allow devices of every kind to speak to the web. Software or gadget producers who depend on gSOAP to assist their companies are affected by Devil's Ivy, although the extent to which such devices could also be exploited can't be decided at the moment. Based on our analysis, servers are extra possible to be exploited. But shoppers could be vulnerable as properly, in the event that they obtain a SOAP message from a malicious server. " 
Why Rust?

- Memory safety and type-safety
- Fine-grained memory control
- Low runtime overhead
- Reduce the number of vulnerable IoT devices
- Why not write an IoT OS in Rust? :)
Tock

IoT operating system
Reliability and Security
Research project
ARM-Cortex M Microcontrollers
Tock - Architecture

Memory Isolation

User-space processes

“Microkernel-ish”:

- Core kernel
- Capsules (relied on type-system not separate processes)

Figure from www.tockos.org
nRF51-DK

nRF51, ARM cortex M0
16 MHz CPU,
32kB RAM
256kB Flash
Radio 2.4GHz (Bluetooth Low Energy)
AES, TRNG, Temperature Sensor
What have we done?

<table>
<thead>
<tr>
<th>Packet</th>
<th>TockOS</th>
<th>Encrypt Temperature &amp; Random bytes</th>
</tr>
</thead>
</table>

- Temperature Sensor
- AES encryption
- TRNG
- Radio

nRF51
Bluetooth Low Energy - Driver

User-space Process
- main program

BLE library
- ble_adv_data
- ble_adv_clear_data
- ble_adv_set_txpower
- ble_adv_set_interval
- ble_adv_start
- ble_adv_scan
- ble_adv_stop
- ble_adv_set_address

Tock Kernel
- BLE Capsules
- BLE HiL
- BLE Radio
- Hardware

System Call
User-space process in C

```c
#include <stdio.h>
#include <ble.h>

/*
 * BLE Demo Application
 * 1. Configures transmitting power, advertisement interval & advertisement address
 * 2. Configures advertisement data
 * 3. Start advertisement and run forever
 */

int main(void)
{
    unsigned char name[] = "TockOS";
    unsigned char addr[] = {0x1, 0x2, 0x3, 0x4, 0x5, 0x6};

    ble_adv_set_txpower(ODBM);
    ble_adv_set_interval(TEN_MS);
    ble_adv_set_address(addr, sizeof(addr));
    ble_adv_data(BLE_HS_ADV_TYPE_COMP_NAME, name, sizeof(name) - 1);
    ble_adv_start(CONN_NON);

    return 0;
}
```
Bluetooth Low Energy - Capsule
fn set_adv_data(&self, ad_type: usize) -> ReturnCode {
    let mut return_code = ReturnCode::ESIZE;
    for cntr in self.app.iter() {
        cntr.enter(|app, _| {
            app.app_write.as_ref().map(|slice| {
                let len = slice.len();
                // Each AD TYP consists of TYPE (1 byte), LENGTH (1 byte) and
                // PAYLOAD (0 - 31 bytes)
                // This is why we add 2 to start the payload at the correct position.
                let i = self.offset.get() + len + 2;
                if i <= 31 {
                    self.kernel_tx.take().map(|data| {
                        for (out, inp) in data.iter_mut().zip(slice.as_ref()[0..len].iter()) {
                            *out = *inp;
                        }
                    }).
                    .set_advertisement_data(ad_type, data, len, self.offset.get() + 8);
                    self.kernel_tx.replace(tmp);
                    self.offset.set(i);
                    return_code = ReturnCode::SUCCESS;
                }
            });
        });
        return_code
    }
}
Bluetooth Low Energy - Hardware Module
pub struct Radio {
    // pointer to struct of memory mapped I/O
    regs: *const peripheral_registers::RADIO_REGS,
}

fn radio_on(&self) {
    // deference and write to raw memory
    let regs = unsafe { &*self.regs };
    // reset and enable power
    regs.POWER.set(0);
    regs.POWER.set(1);
}
Benchmarks

Evaluate our drivers

Comparison with state-of-the-art IoT operating systems:

- Apache mynewt
- ARM mbed
- Zephyr
10 seconds advertisement

Advertisement configuration:
- 150 ms interval
- Transmitting power 0 dBm
- Payload size 22 bytes

The bars illustrate different power consumption

Turning off power hungry peripherals
This sounds great right, but how has the journey been?
Fail, pick yourself up and fail again.

- Charlie Day
Learning an Operating System with limited documentation is hard

How do the system calls work?

How to pass a buffer to the kernel?

How to use raw bytes in the kernel? (nested closures)

IRC-channel a big help
No debugging symbols

```
00000000
  dc:  e92d 47f0  movw  r8, #4868    ; 0x1304
  e0:  f241 3804  movw  r8, #4868    ; 0x1304
  e4:  4682      mov    sl, r0      
  e6:  f44f 3080  mov    r0, #65536  ; 0x10000
  ea:  2700      movs   r7, #0      
  ec:  f2c4 0801  movt   r8, #16385 ; 0x4001
  f0:  f8c8 0004  str    r0, [r8, #4]
  f4:  f8d8 9200  ldr    r9, [r8, #512] ; 0x200
```
No printouts

Issue: not possible to print messages with panic. #295

frenich wants to merge 10 commits into heleena-project:master from frenich:nrf51/panic_fix

frenich commented on Feb 21

Panic for NRF51DK was missing feature to print panic messages. Implemented the panic message in a similar way as for the other boards.

Issue: not possible to print messages with panic. Solution: implement...
What it is my experience using Rust?

Learning curve is rather steep (ownership paradigm, interior mutability, etc)

The compiler is your friend and educates you to write good code

Made me a better programmer

Crashes happens very rarely (don’t do unwrap on Options)

Rust IRC-channels are very useful (keep it up)
What I want to see in the future

Rust in safety critical applications (medical devices, autonomous vehicles and etc)

Convince embedded community to adopt Rust (C++ has been struggled with this as well)

Full-fledged IDEs with integrated debugger
Thanks for your attention

Contribute to Tock:

- Buy a hail board, [https://www.tockos.org/hardware/hail](https://www.tockos.org/hardware/hail)
- Buy nRF51-DK, nRF52-DK
- Port a new processor (e.g STM chip)
- [https://github.com/helena-project/tock](https://github.com/helena-project/tock)

Slides inspired by:

Amit Levy, Fredrik Nilsson, Alejandro Russo and many others