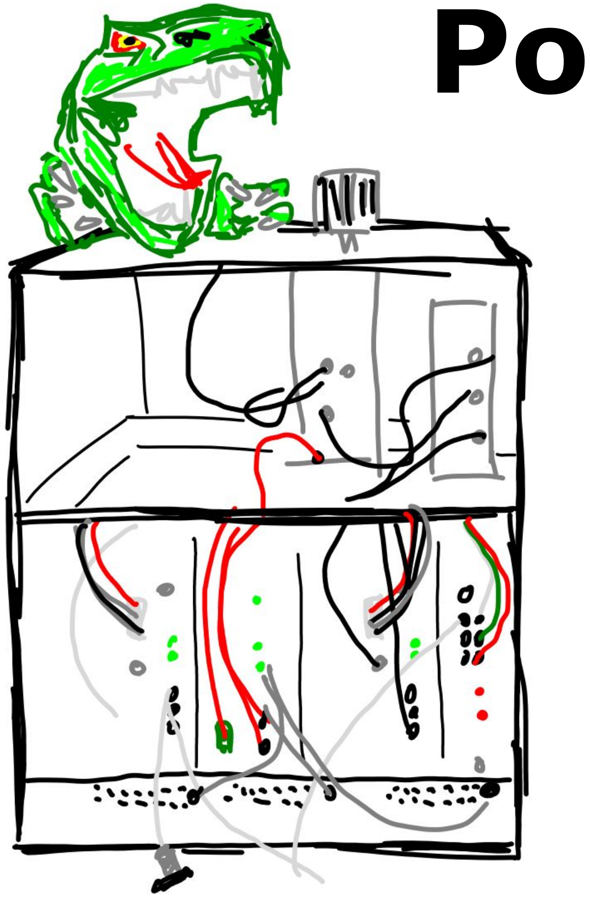
Dinosaur Resurrection

PowerPC Binary Patching for Base Station Analysis



Uwe Müller, Eicke Hauck, Timm Welz, Jiska Classen, Matthias Hollick
Secure Mobile Networking Lab - SEEMOO
Technische Universität Darmstadt, Germany

Motivation

What is TETRA?

Just the same as GSM but for emergency communication in Europe.

Stronger encryption than GSM :)

SIM-based authentication

Walkie-talkie mode (DMO) and base station mode (TMO)

Separate from other mobile infrastructure

Group calls

Voice + text messages

What is PowerPC?

A dating^Wdated computing architecture.

PowerPC-based TETRA Base Station

Never gets old!!!1!

1995

1991

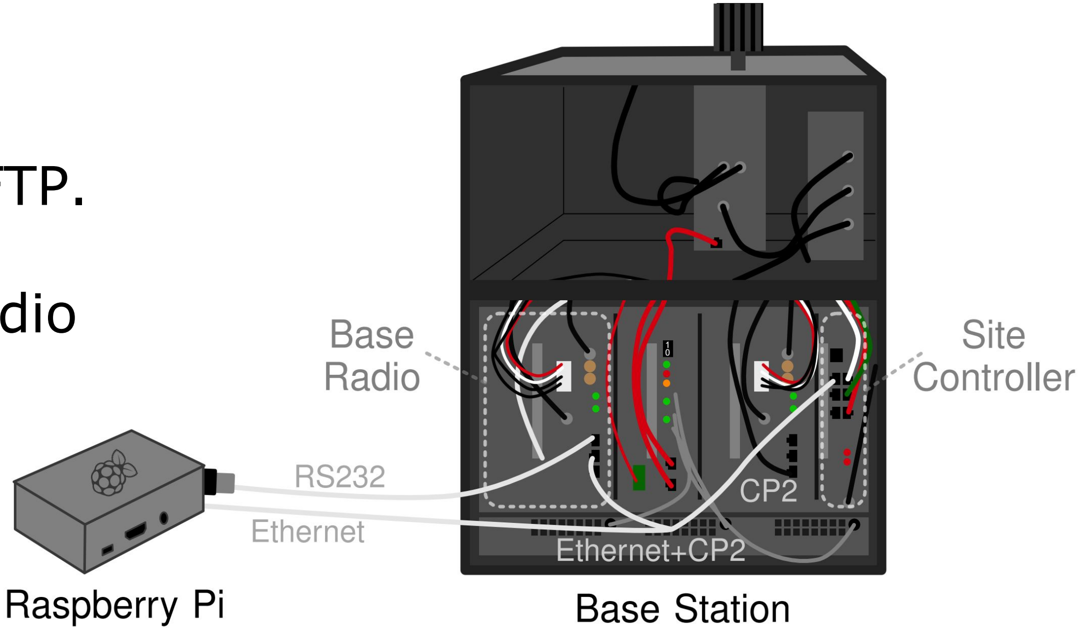
TETRA Base Station Setup for Testing

Safety measures

- Put everything into an EMF-shielded tent.
- Add a huuuuge dummy load.
- Configure an invalid frequency.
- Only analyze and fuzz local interfaces.

Firmware flashing and control

- Site controller usually offers firmware via TFTP.
- Raspberry Pi replaces TFTP controller.
- Also connect to serial console of the base radio (bootloader and crash output, local shell).



Static Firmware Analysis

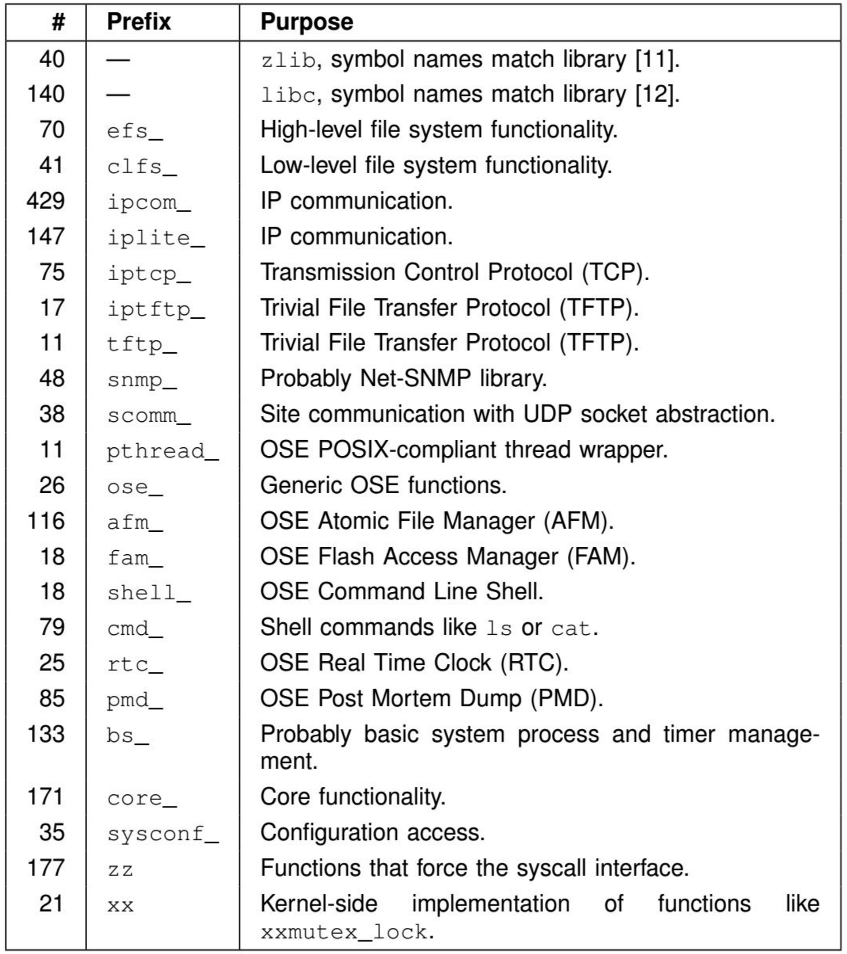
Firmware Format

- Base station runs an Enea POLO Bootloader.
- Bootloader gets ELF via TFTP from site controller.
- The ELF can be compressed with gzip.
- The ELF contains symbols! 🎉 🥳 🥂

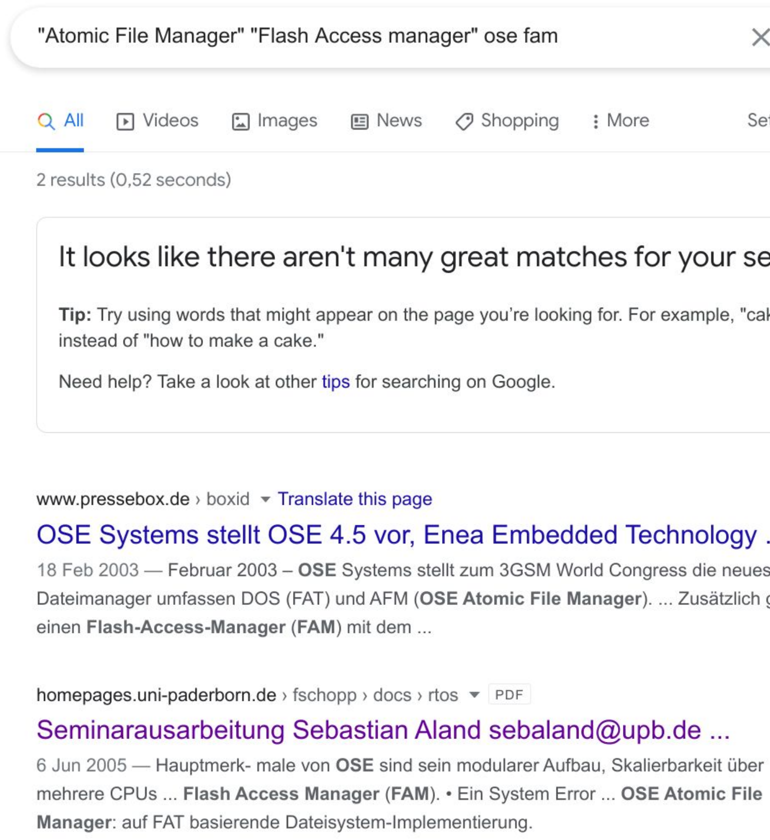
Reverse Engineering <3

Function Name and Library Analysis

- Operating System Embedded (OSE) 4.5.2, developed by Enea AB.
- IPCOM network stack by Interpeak AB.
- MPC8260ADS SoC featuring a big-endian PowerPC CPU.
- Compile dates back from 2006/2007.



#	Prefix	Purpose
40	—	`zlib`, symbol names match library [11].
140	—	`libc`, symbol names match library [12].
70	`efs_`	High-level file system functionality.
41	`clfs_`	Low-level file system functionality.
429	`ipcom_`	IP communication.
147	`iplite_`	IP communication.
75	`iptcp_`	Transmission Control Protocol (TCP).
17	`iptftp_`	Trivial File Transfer Protocol (TFTP).
11	`tftp_`	Trivial File Transfer Protocol (TFTP).
48	`snmp_`	Probably Net-SNMP library.
38	`scomm_`	Site communication with UDP socket abstraction.
11	`pthread_`	OSE POSIX-compliant thread wrapper.
26	`ose_`	Generic OSE functions.
116	`afm_`	OSE Atomic File Manager (AFM).
18	`fam_`	OSE Flash Access Manager (FAM).
18	`shell_`	OSE Command Line Shell.
79	`cmd_`	Shell commands like `ls` or `cat`.
25	`rtc_`	OSE Real Time Clock (RTC).
85	`pmd_`	OSE Post Mortem Dump (PMD).
133	`bs_`	Probably basic system process and timer management.
171	`core_`	Core functionality.
35	`sysconf_`	Configuration access.
177	`zz`	Functions that force the syscall interface.
21	`xx`	Kernel-side implementation of functions like `xxmutex_lock`.



"Atomic File Manager" "Flash Access manager" ose fam

2 results (0,52 seconds)

It looks like there aren't many great matches for your search

Tip: Try using words that might appear on the page you're looking for. For example, "cake recipes" instead of "how to make a cake."

Need help? Take a look at other tips for searching on Google.

www.pressebox.de › boxid ▾ Translate this page

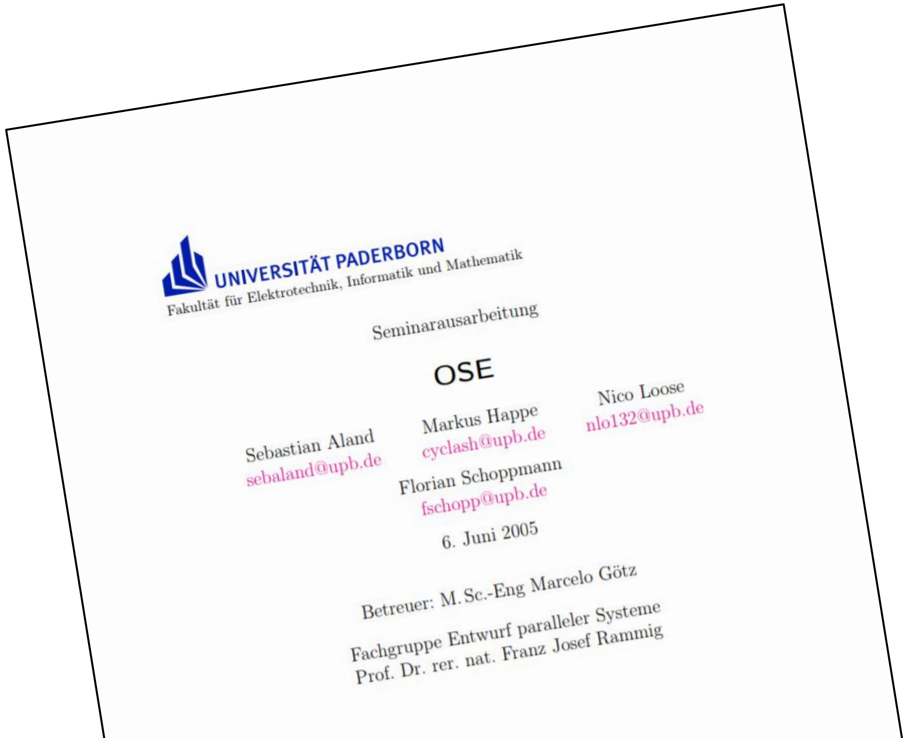
OSE Systems stellt OSE 4.5 vor, Enea Embedded Technology ...

18 Feb 2003 — Februar 2003 – **OSE** Systems stellt zum 3GSM World Congress die neueste ... Dateimanager umfassen DOS (FAT) und AFM (**OSE Atomic File Manager**). ... Zusätzlich gibt es einen **Flash-Access-Manager** (**FAM**) mit dem ...

homepages.uni-paderborn.de › fschopp › docs › rtos ▾ PDF

Seminarausarbeitung Sebastian Aland sebaland@upb.de ...

6 Jun 2005 — Hauptmerk- male von **OSE** sind sein modularer Aufbau, Skalierbarkeit über mehrere CPUs ... **Flash Access Manager** (**FAM**). • Ein System Error ... **OSE Atomic File Manager**: auf FAT basierende Dateisystem-Implementierung.



UNIVERSITÄT PADERBORN
Fakultät für Elektrotechnik, Informatik und Mathematik

Seminarausarbeitung

OSE

Sebastian Aland
sebaland@upb.de

Markus Happe
cyclash@upb.de

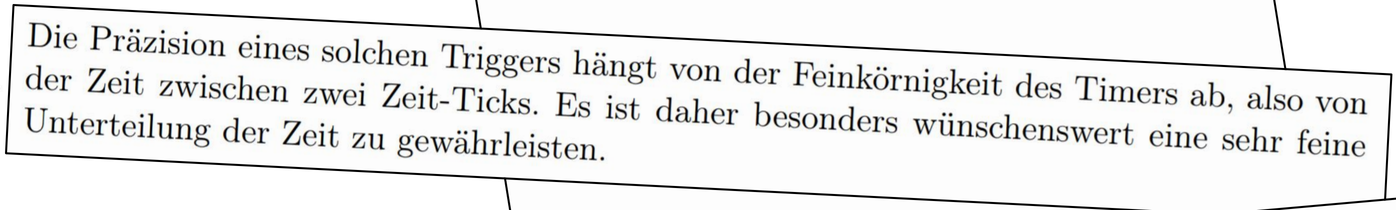
Nico Loose
nlo132@upb.de

Florian Schoppmann
fschopp@upb.de

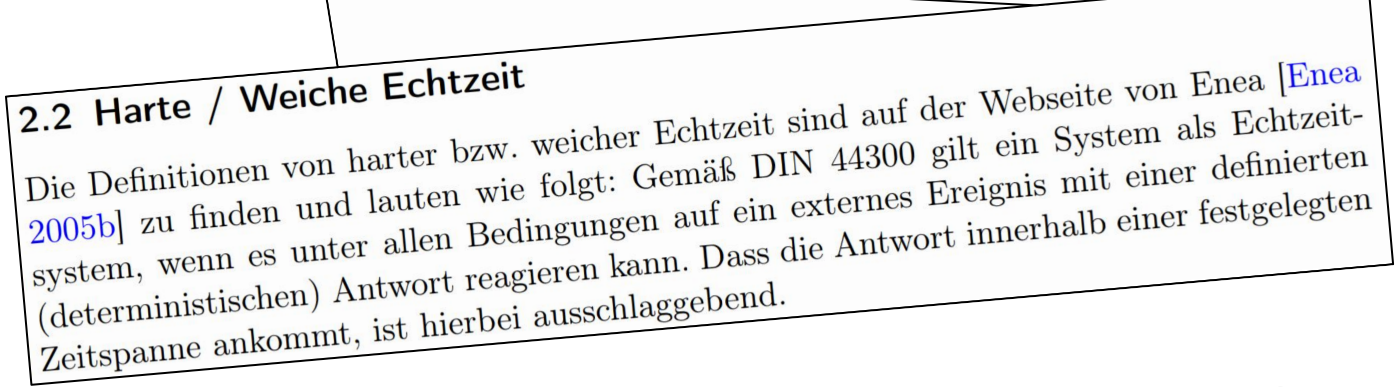
6. Juni 2005

Betreuer: M. Sc.-Eng Marcelo Götz

Fachgruppe Entwurf paralleler Systeme
Prof. Dr. rer. nat. Franz Josef Rammig



Die Präzision eines solchen Triggers hängt von der Feinkörnigkeit des Timers ab, also von der Zeit zwischen zwei Zeit-Ticks. Es ist daher besonders wünschenswert eine sehr feine Unterteilung der Zeit zu gewährleisten.



2.2 Harte / Weiche Echtzeit

Die Definitionen von harter bzw. weicher Echtzeit sind auf der Webseite von Enea [Enea 2005b] zu finden und lauten wie folgt: Gemäß DIN 44300 gilt ein System als Echtzeitsystem, wenn es unter allen Bedingungen auf ein externes Ereignis mit einer definierten (deterministischen) Antwort reagieren kann. Dass die Antwort innerhalb einer festgelegten Zeitspanne ankommt, ist hierbei ausschlaggebend.

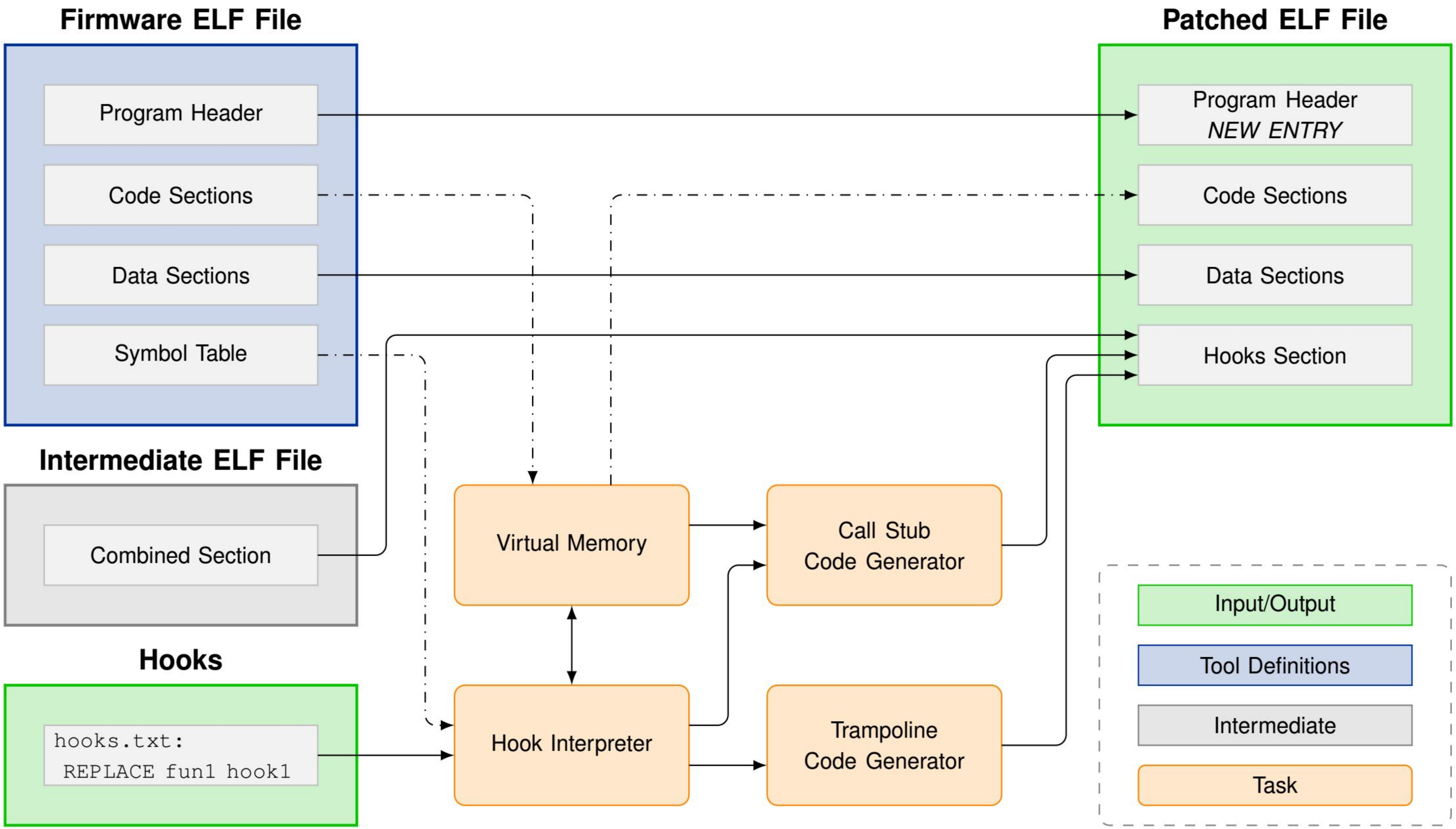
PowerPC Binary Patcher

Let’s patch the firmware using C!

PowerPC Assembler Example

```
stwu r1, -0x10(r1) ; r1 is the stack pointer, make room
     ; Replace this with branch to hook
mflr r0            ; Move contents of link register to r0
stw  r0, 0x10(r1)  ; Push r0 onto stack
; Function code
lwz  r0, 0x10(r1)  ; Load r0 from stack
mtlr r0            ; Move contents of r0 to link register
addi r1, r1, 0x10  ; Restore old stack pointer
blr                ; Branch link return
```

- Each function in our target binary starts with the same two position-independent instructions.
- Replace these with a jump to the actual hook.
- Hooks can be added to the beginning (`PRECALL`), end (`POSTCALL`), or replace a function (`REPLACE`).



Firmware ELF File

Program Header

Code Sections

Data Sections

Symbol Table

Intermediate ELF File

Combined Section

Hooks

```
hooks.txt:
 REPLACE fun1 hook1
```

Virtual Memory

Hook Interpreter

Call Stub
Code Generator

Trampoline
Code Generator

Patched ELF File

Program Header
NEW ENTRY

Code Sections

Data Sections

Hooks Section

Input/Output

Tool Definitions

Intermediate

Task

Demo: Blinking LEDs

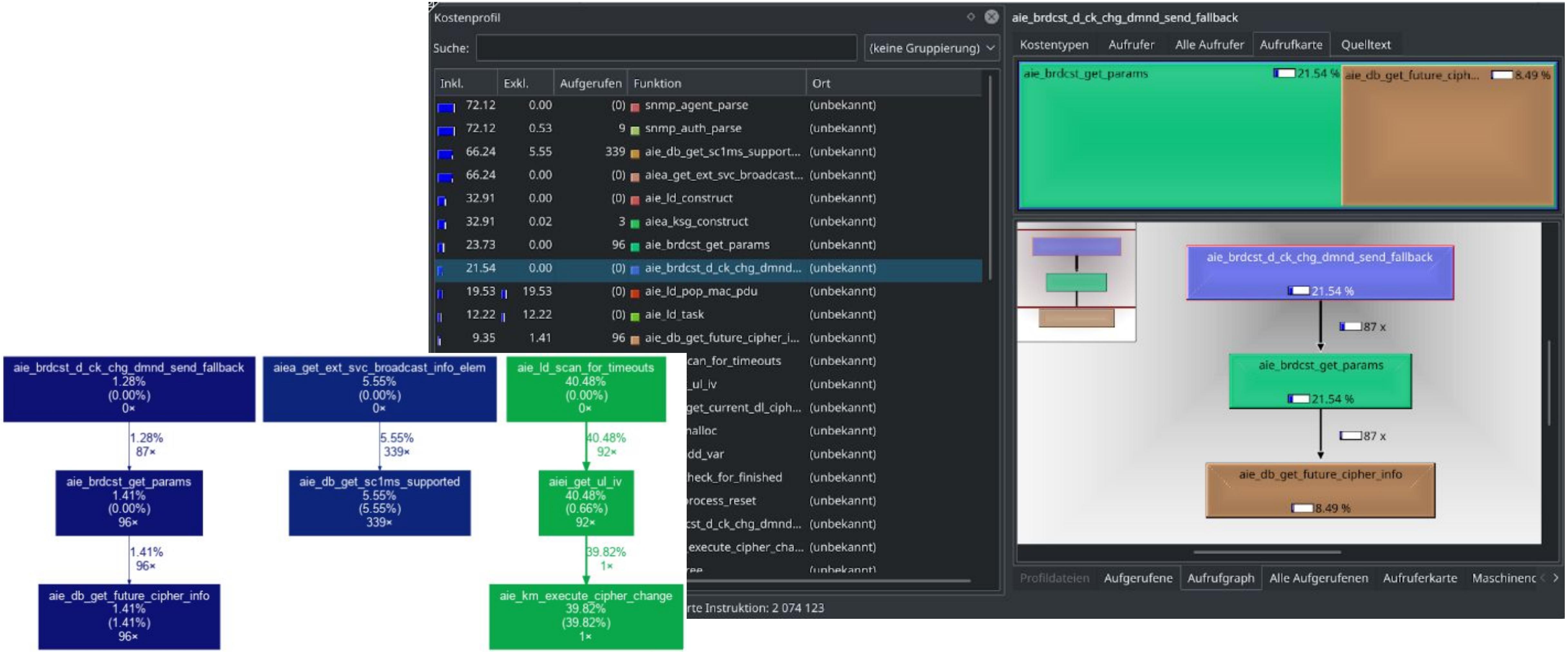
Dynamic Firmware Analysis

Call Traces

```
uint32_t calltrace() {
  uint32_t pid = 0;
  if (ose_ready) {
    pid = current_process();
  }
  calltrace_log_enter(pid);
  // Get cycle count from CPU registers for time measurement
  uint64_t begin =  cpu_cycle_count();
  // Call original function w/o knowing anything about it
  uint32_t ret   =  orig_call();
  // Get cycle count again for duration
  uint64_t end   =  cpu_cycle_count();
  calltrace_log_leave(end - begin);
  return ret;
}
```

- Replace all functions matching a regular expression with a call trace instrumentation.
- Log time (execution time and function order) and currently active thread.
- Conversion to Callgrind format, shows time spent in each function.

Callgrind Interpretation



Kostenprofil

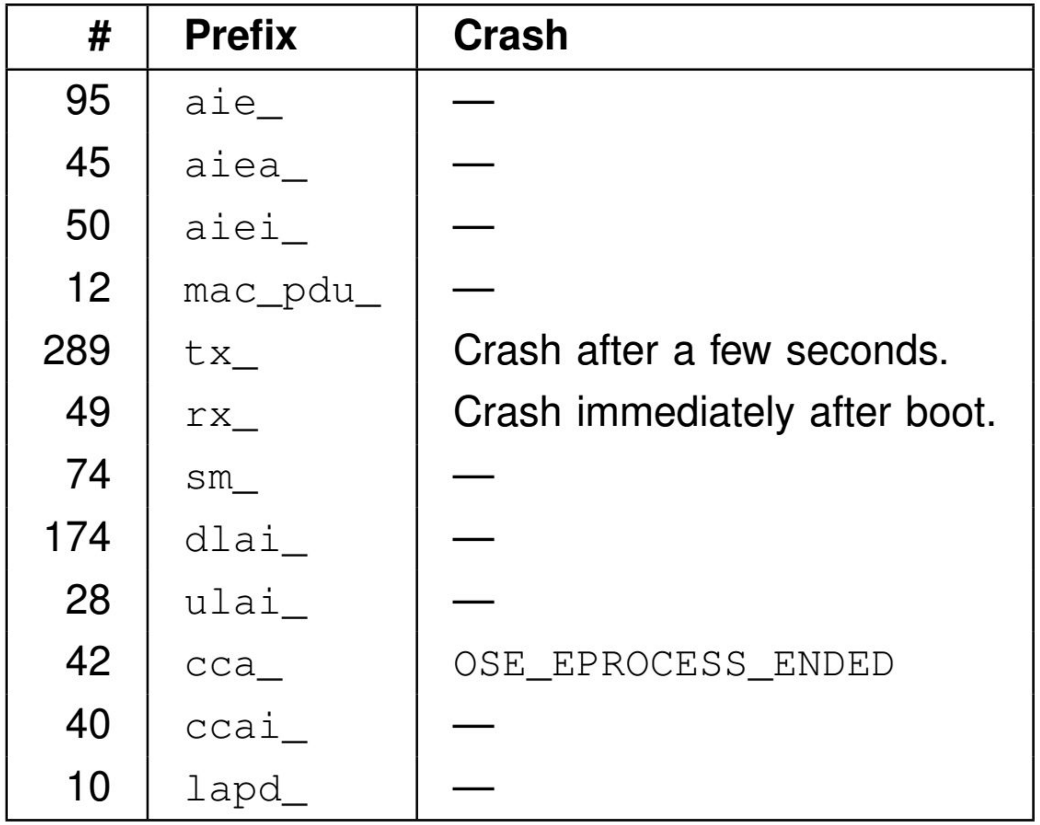
Suche: (keine Gruppierung)

Inkl.	Exkl.	Aufgerufen	Funktion	Ort
72.12	0.00	(0)	snmp_agent_parse	(unbekannt)
72.12	0.53	9	snmp_auth_parse	(unbekannt)
66.24	5.55	339	aie_db_get_sc1ms_support...	(unbekannt)
66.24	0.00	(0)	aiea_get_ext_svc_broadcast...	(unbekannt)
32.91	0.00	(0)	aie_ld_construct	(unbekannt)
32.91	0.02	3	aiea_ksg_construct	(unbekannt)
23.73	0.00	96	aie_brdcst_get_params	(unbekannt)
21.54	0.00	(0)	aie_brdcst_d_ck_chg_dmnd...	(unbekannt)
19.53	19.53	(0)	aie_ld_pop_mac_pdu	(unbekannt)
12.22	12.22	(0)	aie_ld_task	(unbekannt)
9.35	1.41	96	aie_db_get_future_cipher_i...	(unbekannt)
			can_for_timeouts	(unbekannt)
			_ul_iv	(unbekannt)
			get_current_dl_ciph...	(unbekannt)
			nalloc	(unbekannt)
			dd_var	(unbekannt)
			heck_for_finished	(unbekannt)
			rocess_reset	(unbekannt)
			cst_d_ck_chg_dmnd...	(unbekannt)
			execute_cipher_cha...	(unbekannt)
			ree	(unbekannt)

rte Instruktion: 2 074 123

Interrupt-related Hooks

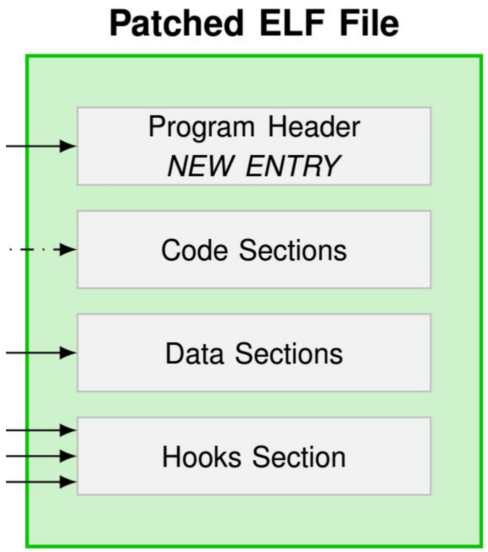
- Call traces perform very smooth within most libraries.
- If functions are related to hardware interrupts, certain PowerPC instructions cannot be executed.
- This leads to crashes within some libraries.



#	Prefix	Crash
95	aie_	—
45	aiea_	—
50	aiei_	—
12	mac_pdu_	—
289	tx_	Crash after a few seconds.
49	rx_	Crash immediately after boot.
74	sm_	—
174	dlai_	—
28	ulai_	—
42	cca_	OSE_EPROCESS_ENDED
40	ccai_	—
10	lapd_	—

Patching without Reboots

```
405360c405360
< 1cd5c4: 94 21 ff 18  stwu r1,-232(r1) ; orig. instruction
---
> 1cd5c4: 48 02 dc 2e  ba   2dc2c        ; jump to trampoline
586913a586914,586934
>  2dc1c: 94 21 ff 18  stwu r1,-232(r1) ; new hook code
>  2dc20: 48 1c d5 ca  ba   1cd5c8 <printf+0x4>
> ; further instructions that are added...
```



- Hooks section always ends up at the same address within the patched ELF.
- Comparison based on objdump output is straightforward :)
- We can use this to patch the firmware at runtime.
- Sufficiently stable for most use cases :D

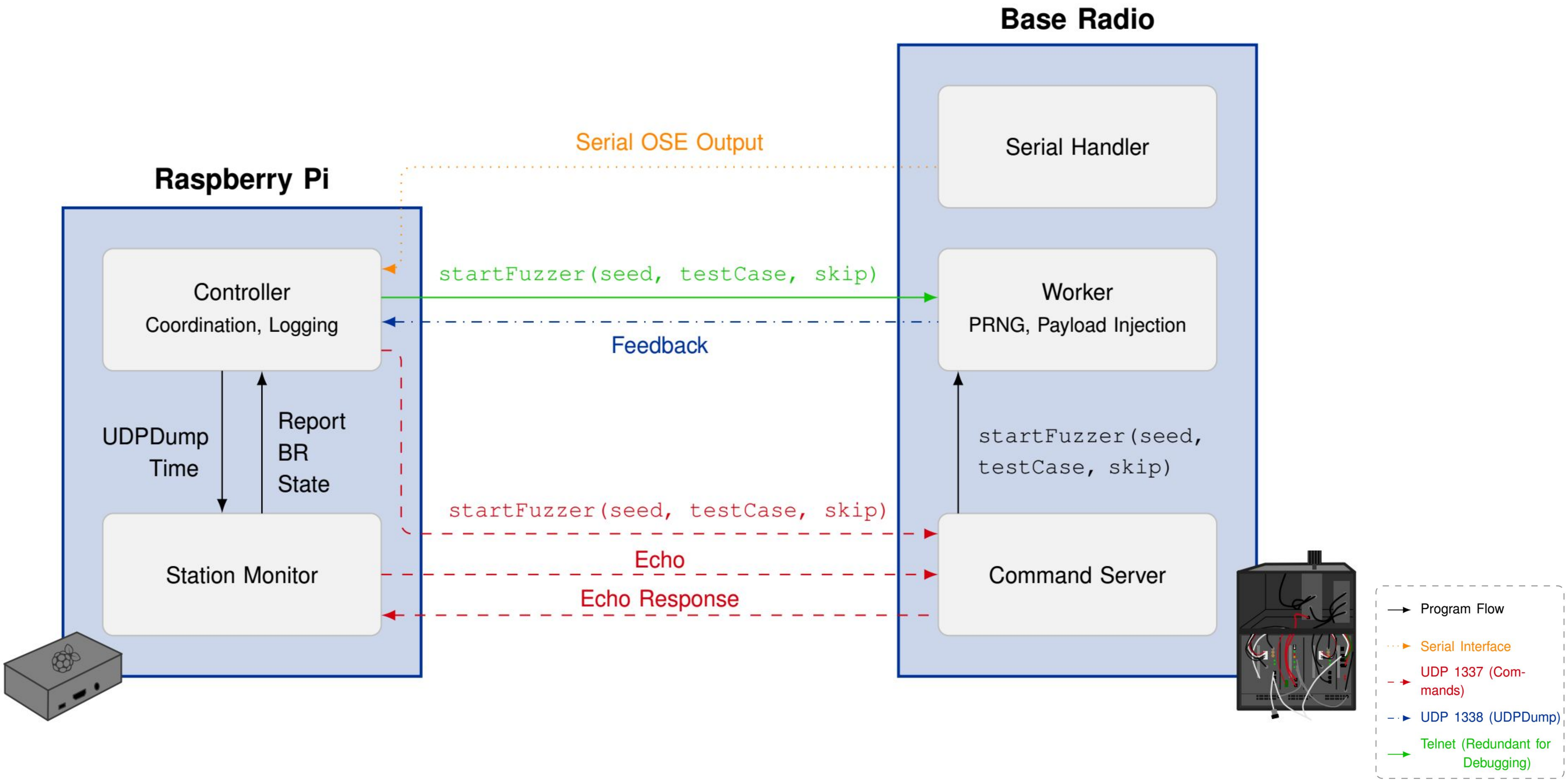
Calling Functions During Runtime

- The previous approach still requires firmware recompilation.
- We can add a simple handler that allows calling functions with arguments directly from the serial command line interface.

```
void* execute_address(int argc, void* addr, void** args) {
  if (argc == 0) {
      return ((void* (*)(void))addr)();
  } else {
      return ((void* (*)(void*, ...))addr)(args[0],
          args[1], args[2], args[3], args[4],
          args[5], args[6], args[7]);
  }
}
```

```
CSS: exaddr 0x1cd5c4 -p 3 %s "Hello %s %d" %s "World" %d 42
Hello World 42
```

Fuzzing with Hyphuzz



Base Radio

Serial Handler

Worker

PRNG, Payload Injection

Command Server

startFuzzer(seed, testCase, skip)

Raspberry Pi

Controller

Coordination, Logging

Station Monitor

UDPDump Time

Report BR State

Serial OSE Output

startFuzzer(seed, testCase, skip)

Feedback

startFuzzer(seed, testCase, skip)

Echo

Echo Response

Program Flow

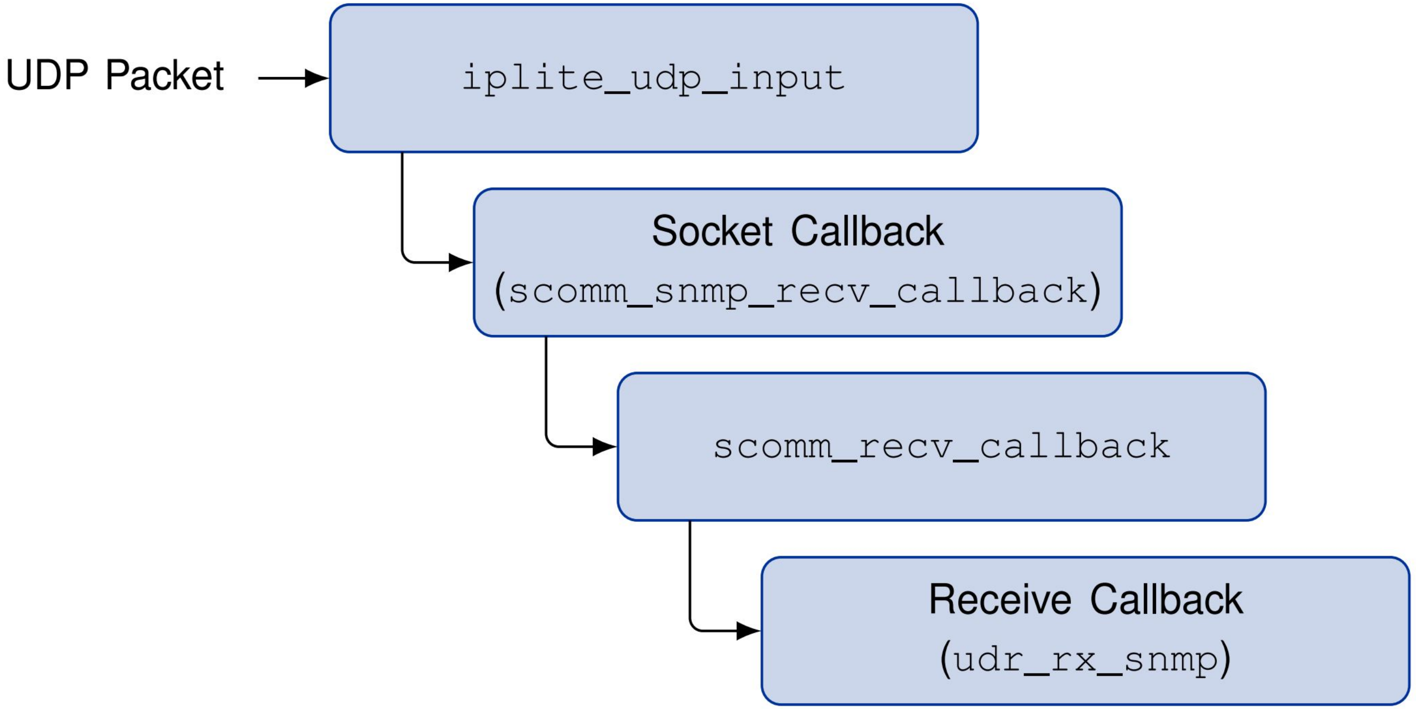
Serial Interface

UDP 1337 (Commands)

UDP 1338 (UDPDump)

Telnet (Redundant for Debugging)

Fuzzing the IPCOM Network Stack



UDP Packet

iplite_udp_input

Socket Callback
(scomm_snmp_recv_callback)

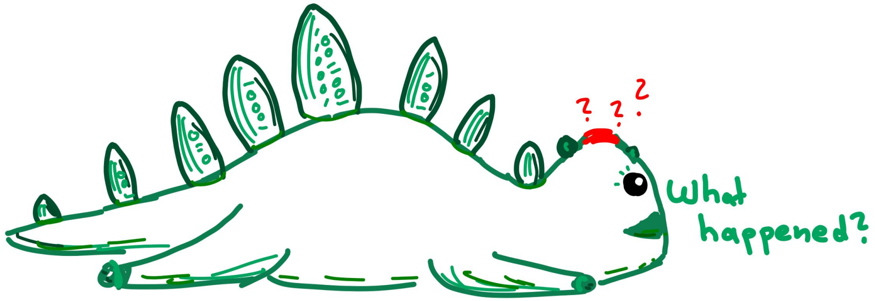
scomm_recv_callback

Receive Callback
(udr_rx_snmp)

```
void scomm_snmp_recv_callback(uint32_t *rxSocketHandlePtr,
     struct NetPacketInfo *netPacketInfo, int
     packetStatus) {
       scomm_recv_callback(netPacketInfo, packetStatus,
              161, *rxSocketHandlePtr);
return;
}
```

OSE Error Handlers and Crash Types

#	Error Type	Caller
158	OSE_EILLEGAL_PROCESS_ID	OSE_SEND_W_S
33	OSE_ENOT_SIG_OWNER	OSE_SEND
8	OSE_ENOT_SIG_OWNER	OSE_SIGSIZE
4	OSE_EPROCESS_ENDED	<UNKNOWN>
3	OSE_EILLEGAL_SYSTEMCALL	OSE_WAIT_SEM



```
[ERROR HANDLER INVOKED]  fatal:YES  error:
                         OSE_ESUPERV_STACK_OVERFLOW(0x0102)
                         caller:<UNKNOWN>(0x00)
        [ERROR DETAILS]  user:NO  code:0x080000102
                         subcode:0x0aebd60
[PROCESS CONTROL BLOCK]  name:fuzzer_thread type:OS_BG_PROC
                         (64) status:<UNKNOWN>(0) priority:0
                [STACK]  top:0x0aec55f limit:0x0aebd60
         [CALLING CODE]  n/a:0
            [REGISTERS]  R0=3718B74E R1=00AEBD38 R2=002877CC
                         ...
               [ACTION]  Writting post mortem debuger info
               [ACTION]  Resetting BR
```

- Some crashes do not result in an error. Hard to analyze without emulation etc.
- Other crashes result in crash logs sent to the serial console :)

Fuzzing Overhead

Activity	CPU Cycles	Overhead
Target Call	117 207	—
Input Generation	11 084	9.5 %
Feedback	1318	1.1 %
Cleanup	1278	1.1 %
Total Overhead	13 680	11.7 %

Q&A

Twitter: @naehrdine, @seemoolab

✉ jiska@bluetooth.lol

https://github.com/seemoo-lab/powerpc-ose