Inside the AMD Microcode ROM - (Ab)Using AMD Microcode for fun and security

35th Chaos Communication Congress, Leipzig

December 27, 2018

Benjamin Kollenda, Philipp Koppe, Marc Fyrbiak, Christian Kison, Christof Paar, Thorsten Holz

Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum
`<firstname.lastname>@rub.de`

emproof
www.emproof.de

Outline

- Crash course: Micro-architecture basics and Microcode
- Reconstructing the Microcode ROM
- Application examples
- Framework overview

Outline

RUHR UNIVERSITÄT BOCHUM RUB

- Crash course: Micro-architecture basics and Microcode
- Reconstructing the Microcode ROM
- Application examples
- Framework overview

Microcode Essentials

- Firmware for the processor

Microcode Essentials

- Firmware for the processor
    - Fix CPU bugs

Microcode Essentials

- Firmware for the processor
    - Fix CPU bugs
    - **Instruction decoding**

Microcode Essentials

- Firmware for the processor
  - Fix CPU bugs
  - **Instruction decoding**
  - Exception handling

Microcode Essentials

- Firmware for the processor
  - Fix CPU bugs
  - **Instruction decoding**
  - Exception handling
  - Power Management

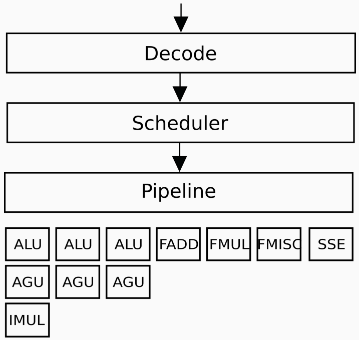
Microcode Essentials

- Firmware for the processor
  - Fix CPU bugs
  - **Instruction decoding**
  - Exception handling
  - Power Management
  - Complex features (Intel SGX)

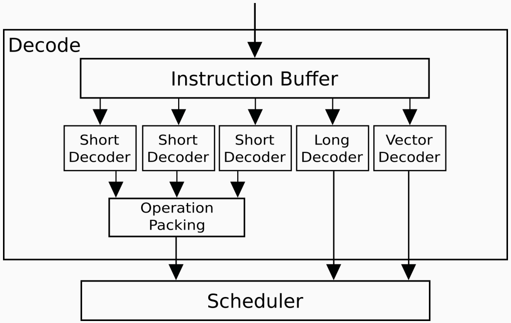
Microcode Essentials

- Firmware for the processor
    - Fix CPU bugs
    - **Instruction decoding**
    - Exception handling
    - Power Management
    - Complex features (Intel SGX)
- **Update capabilities**

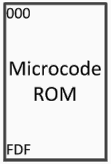
x86 Instruction Decoding



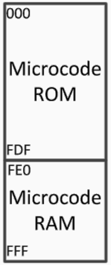
x86 Instruction Decoding



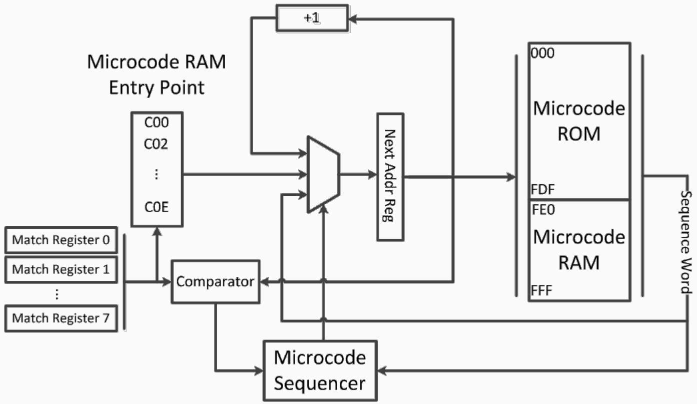
Microcode Engine (Vector Decoder)



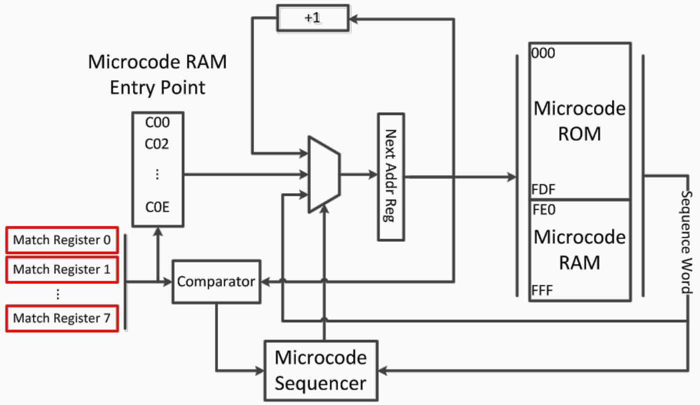
Microcode Engine (Vector Decoder)



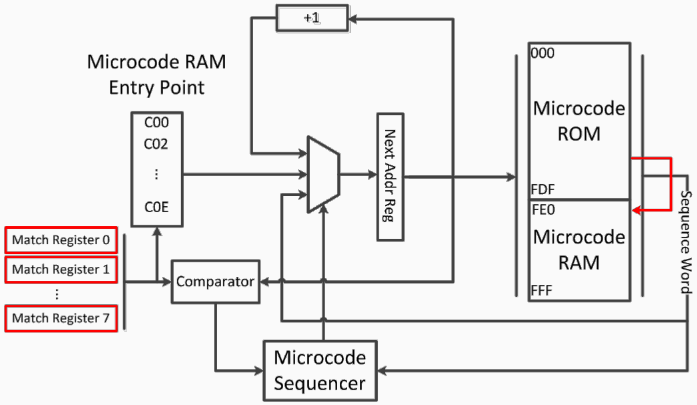
Microcode Engine (Vector Decoder)



Microcode Engine (Vector Decoder)



Microcode Engine (Vector Decoder)



Microcode Updates

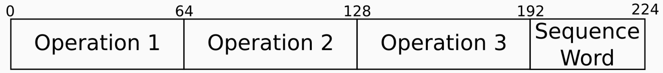
- Updates are loaded by BIOS or kernel

Microcode Updates

- Updates are loaded by BIOS or kernel
- Header followed by multiple triads

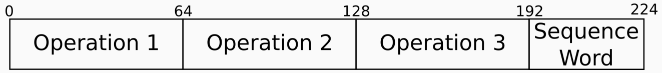
Microcode Updates

- Updates are loaded by BIOS or kernel
- Header followed by multiple triads
- Triad structure:



Microcode Updates

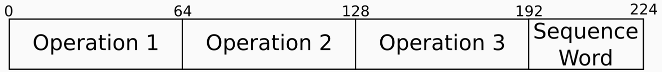
- Updates are loaded by BIOS or kernel
- Header followed by multiple triads
- Triad structure:



- Updates protected by weak authentication

Microcode Updates

- Updates are loaded by BIOS or kernel
- Header followed by multiple triads
- Triad structure:



- Updates protected by weak authentication
- Only one update may be loaded at a time

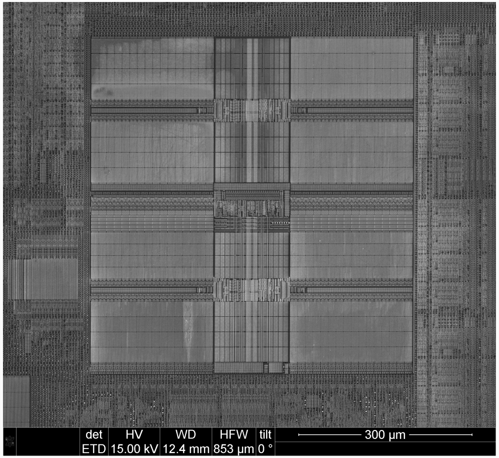
Microcode RTL

```
sub eax, edx
sub.C t56q, rcx, 0x100
jcc ECF, 1
.sw_next // implied sequence word if omitted

ld t1d, [eax]
st [edx], t1d
mov eax, eax
.sw_complete

mov eax, 1
sub.Q rax, rcx
add.EP t56d, eax, ecx
.sw_branch 0xF01
```

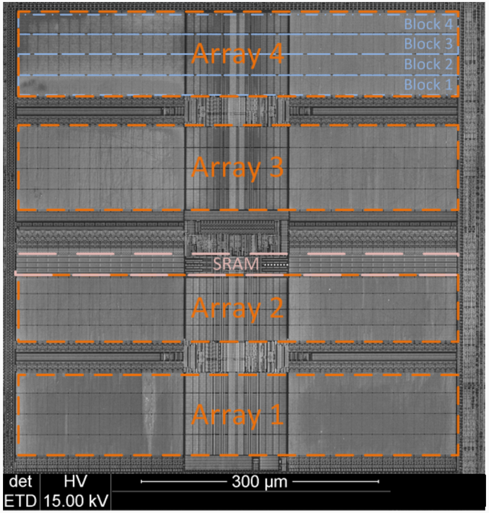
Outline

- Crash course: Micro-architecture basics and Microcode
- Reconstructing the Microcode ROM
- Application examples
- Framework overview

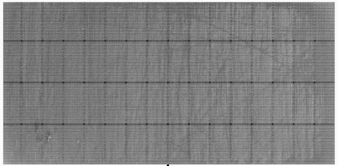
Hardware Analysis



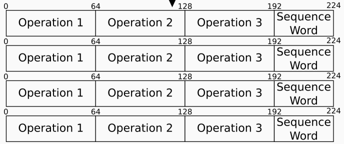
Hardware Analysis



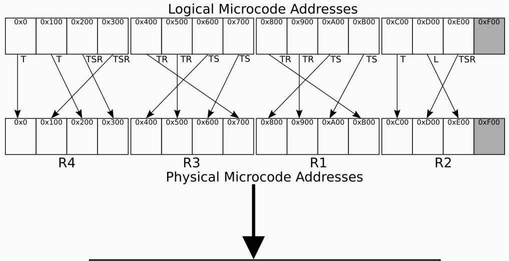
ROM - Recovery Process Overview



```
010101001010010010010
111101000001100010000
110010001010010001100
001001010110000001110
010001110100100100010
001010100100110100001
100001010110011000100
```



ROM - Recovery Process Overview



```
mov t1d, eax
xor.Z t1d, 0x23
jcc nEZF, 0x40
.sw_branch 0x96
xor.Z t1d, 0x64
jcc EZF, 0x30
xor eax, eax
```

ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs

ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs
- Updates yield only two pairs

ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs
- Updates yield only two pairs
- Generate mappings by matching the semantics of triads between ROM dump and address based execution

ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs
- Updates yield only two pairs
- Generate mappings by matching the semantics of triads between ROM dump and address based execution
- Implement microcode emulator to extract semantics

ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs
- Updates yield only two pairs
- Generate mappings by matching the semantics of triads between ROM dump and address based execution
- Implement microcode emulator to extract semantics
  - Works on triad level

ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs
- Updates yield only two pairs
- Generate mappings by matching the semantics of triads between ROM dump and address based execution
- Implement microcode emulator to extract semantics
  - Works on triad level
  - Determines output state based on given input (x86 and microcode registers)

ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs
- Updates yield only two pairs
- Generate mappings by matching the semantics of triads between ROM dump and address based execution
- Implement microcode emulator to extract semantics
    - Works on triad level
    - Determines output state based on given input (x86 and microcode registers)
    - Supports known arithmetic operations

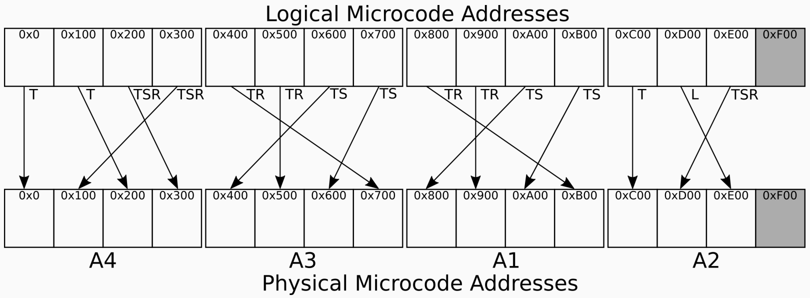
ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs
- Updates yield only two pairs
- Generate mappings by matching the semantics of triads between ROM dump and address based execution
- Implement microcode emulator to extract semantics
    - Works on triad level
    - Determines output state based on given input (x86 and microcode registers)
    - Supports known arithmetic operations
    - Whitelist of no-op operations

ROM - Mapping Recovery Details

- Mapping recovery requires physical-virtual address pairs
- Updates yield only two pairs
- Generate mappings by matching the semantics of triads between ROM dump and address based execution
- Implement microcode emulator to extract semantics
    - Works on triad level
    - Determines output state based on given input (x86 and microcode registers)
    - Supports known arithmetic operations
    - Whitelist of no-op operations
- Emulation yielded 54 additional address pairs

ROM - Mapping Result



ROM - Implementation of Instructions

- SHRD - 0xACA
- RDTSC - 0x318
- WRMSR - 0x6A9

Outline

Microcode applications

- Configurable `rdtsc` precision

Microcode applications

- Configurable `rdtsc` precision
- **Microcode assisted Address Sanitizer**

Microcode applications

- Configurable `rdtsc` precision
- **Microcode assisted Address Sanitizer**
- Microcode instruction set randomization

Microcode applications

- Configurable `rdtsc` precision
- **Microcode assisted Address Sanitizer**
- Microcode instruction set randomization
- Microcode-assisted instrumentation

Microcode applications

- Configurable `rdtsc` precision
- **Microcode assisted Address Sanitizer**
- Microcode instruction set randomization
- Microcode-assisted instrumentation
- Authenticated microcode updates

Microcode applications

- Configurable `rdtsc` precision
- **Microcode assisted Address Sanitizer**
- Microcode instruction set randomization
- Microcode-assisted instrumentation
- Authenticated microcode updates
- Enclave-like execution environment

HWASAN Background

- Address Sanitizer: software instrumentation to detected invalid memory accesses

HWASAN Background

- Address Sanitizer: software instrumentation to detected invalid memory accesses
- Authors proposed HWASAN - hardware assisted ASAN

HWASAN Background

- Address Sanitizer: software instrumentation to detected invalid memory accesses
- Authors proposed HWASAN - hardware assisted ASAN
- New instruction performs ASAN checks, raises fault if invalid

HWASAN Background

- Address Sanitizer: software instrumentation to detected invalid memory accesses
- Authors proposed HWASAN - hardware assisted ASAN
- New instruction performs ASAN checks, raises fault if invalid

```
CheckAddressAndCrashIfBad(Addr, kSize) {
  ShadowAddr = (Addr >> 3) + kOffset;
  if (kSize < 8) {
    Shadow = LoadByte(ShadowAddr);
    if (Shadow && Shadow <= (Addr & 7) + kSize - 1)
      ReportBug(Addr);
  } else {
    Shadow = LoadNBytes(ShadowAddr, kSize / 8);
    if (Shadow) ReportBug(Addr);
  }
}
```

HWASAN Background

- Address Sanitizer: software instrumentation to detected invalid memory accesses
- Authors proposed HWASAN - hardware assisted ASAN
- New instruction performs ASAN checks, raises fault if invalid
- Advantages:

HWASAN Background

- Address Sanitizer: software instrumentation to detected invalid memory accesses
- Authors proposed HWASAN - hardware assisted ASAN
- New instruction performs ASAN checks, raises fault if invalid
- Advantages:
    - Better performance

HWASAN Background

- Address Sanitizer: software instrumentation to detected invalid memory accesses
- Authors proposed HWASAN - hardware assisted ASAN
- New instruction performs ASAN checks, raises fault if invalid
- Advantages:
  - Better performance
  - More compact code

HWASAN Background

- Address Sanitizer: software instrumentation to detected invalid memory accesses
- Authors proposed HWASAN - hardware assisted ASAN
- New instruction performs ASAN checks, raises fault if invalid
- Advantages:
  - Better performance
  - More compact code
  - Runtime configuration

HWASAN Implementation

- Implement HWASAN by replacing `bound`

HWASAN Implementation

- Implement HWASAN by replacing `bound`
  - New interface: `bound reg, [size]`

HWASAN Implementation

- Implement HWASAN by replacing `bound`
  - New interface: `bound reg, [size]`
  - No-op for successful check

HWASAN Implementation

- Implement HWASAN by replacing `bound`
  - New interface: `bound reg, [size]`
  - No-op for successful check
  - Configurable action taken for invalid access

HWASAN Implementation

- Implement HWASAN by replacing `bound`
  - New interface: `bound reg, [size]`
  - No-op for successful check
  - Configurable action taken for invalid access
- Single Instruction error check

HWASAN Implementation

- Implement HWASAN by replacing `bound`
  - New interface: `bound reg, [size]`
  - No-op for successful check
  - Configurable action taken for invalid access
- Single Instruction error check
- No x86 registers needed

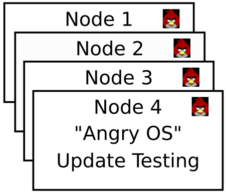
HWASAN Implementation

- Implement HWASAN by replacing `bound`
  - New interface: `bound reg, [size]`
  - No-op for successful check
  - Configurable action taken for invalid access
- Single Instruction error check
- No x86 registers needed
- Micro benchmark shows performance advantage (106 vs. 129 cycles)

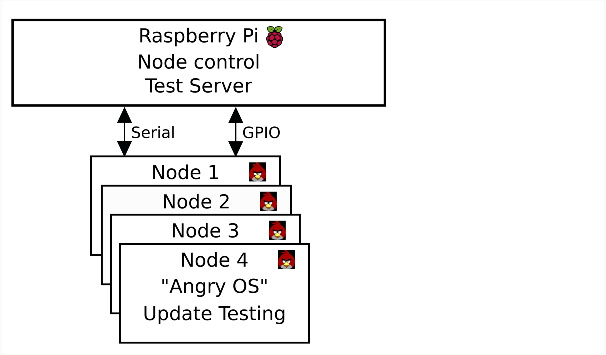
Outline

- Crash course: Micro-architecture basics and Microcode
- Reconstructing the Microcode ROM
- Application examples
- Framework overview

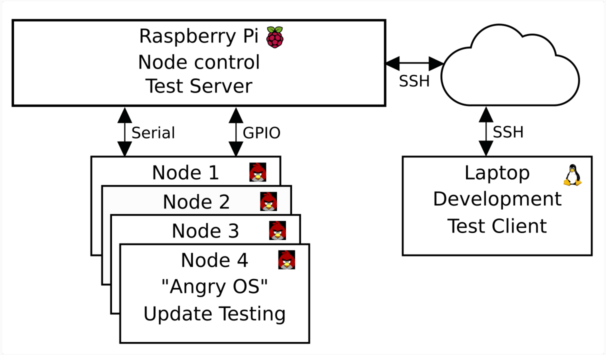
Hardware



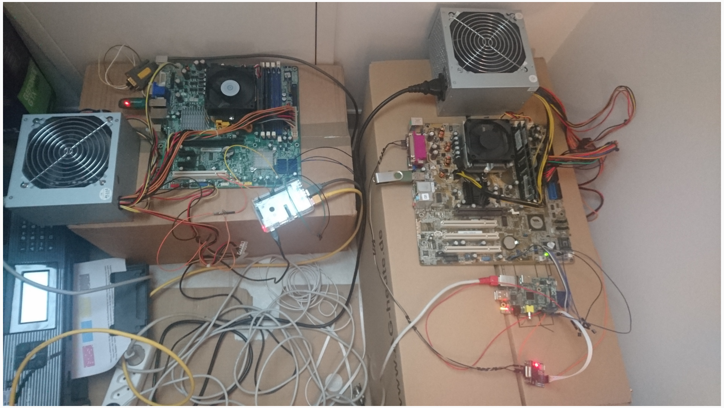
Hardware


Raspberry Pi
Node control
Test Server
Serial
GPIO
Node 1
Node 2
Node 3
Node 4
"Angry OS"
Update Testing

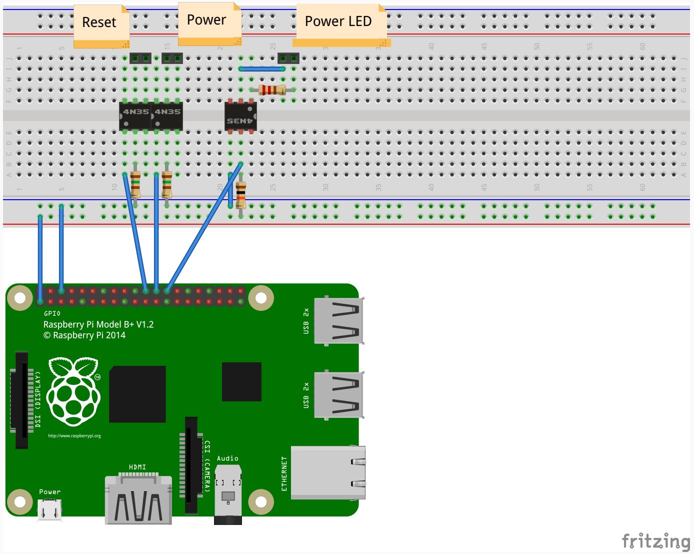
Hardware


Raspberry Pi
Node control
Test Server
SSH
Serial
GPIO
SSH
Node 1
Node 2
Node 3
Node 4
"Angry OS"
Update Testing
Laptop
Development
Test Client

Hardware



Hardware


Reset
Power
Power LED
4N35
4N35
4N35
GPIO
Raspberry Pi Model B+ V1.2
© Raspberry Pi 2014
DSI (DISPLAY)
http://www.raspberrypi.org
HDMI
CSI (CAMERA)
Audio
Power
ETHERNET
USB 2x
USB 2x
fritzing

Angry OS

- Minimal custom operating system

Angry OS

- Minimal custom operating system
- Control 100% of executed instructions

Angry OS

- Minimal custom operating system
- Control 100% of executed instructions
- Listens for commands on the serial port

Angry OS

- Minimal custom operating system
- Control 100% of executed instructions
- Listens for commands on the serial port
- Apply updates, run streamed test code, error reporting

Framework

- Microcode assembler and verbose disassembler

Framework

- Microcode assembler and verbose disassembler
- x86 assembler to write test code

Framework

- Microcode assembler and verbose disassembler
- x86 assembler to write test code
- Disassemble existing updates and ROM contents after extraction

Framework

- Microcode assembler and verbose disassembler
- x86 assembler to write test code
- Disassemble existing updates and ROM contents after extraction
- Create new updates, loadable by Linux update driver

Framework

- Microcode assembler and verbose disassembler
- x86 assembler to write test code
- Disassemble existing updates and ROM contents after extraction
- Create new updates, loadable by Linux update driver
- Control Angry OS node via serial and GPIO

Framework

- Microcode assembler and verbose disassembler
- x86 assembler to write test code
- Disassemble existing updates and ROM contents after extraction
- Create new updates, loadable by Linux update driver
- Control Angry OS node via serial and GPIO
- Remote execution wrapper

Conclusion

- Reversing of the ROM opens up many more possibilities
- Lots left to do, if you want to help, contact us!
- Framework, Angry OS, example programs and more available on Github

`https://github.com/RUB-SysSec/Microcode`

Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum

emproof
www.emproof.de