Machine Code Verification of a Tiny ARM Hypervisor

Hamed Nemati

Joint work with: Mads Dam, Roberto Guanciale

KTH Royal Institute of Technology - Stockholm

TrustED November 2013
Motivation

Photo courtesy www.bravr.com
Physical independence!
Deploying same platform
Isolation is required
Virtualization could give us all!
Virtualization could give us all!

It should be verified.
Verification Strategy

- Ideal setting as a Top Level Specification

Diagram:
- **Ideal setting**
  - Untrusted App.
  - OS Kernel
  - ARMv7

- **Ideal setting**
  - Trusted App.
  - OS Kernel
  - ARMv7

Timer
Verification Strategy

- Real world: the hypervisor with two guests on top
Verification Strategy

**Task!** Check if these two worlds are behaving in the same way
Real world traces
Verification Strategy

- Ideal setting traces

1 → 2 → 3
5 → 6 → 7
10 → 11 → 12
15 → 16
Verification Strategy

What we want to achieve

[Diagram showing a comparison between real and ideal settings, with nodes and arrows representing communication and processes.]
Verification Strategy

- Safety of user transitions

- Processor configuration property, proved in HOL4
Verification Strategy

- Safety of mode switches

- Processor configuration property, proved in HOL4
Verification Strategy

- Safety of handlers code

- Using machine code verification.
Actual code verification

Handlers’ Contract is in the form of Hoare triples.

- **Example:**

  - **Precondition:**
    \(~(\text{mem:}u32[R\_EBP:u32-8:u32, \text{e\_little}]:u32 \leq 0x10) \& \ (R\_EAX \times R\_EAX \leq \text{xparam})\) & ...

  - **Body:**
    \text{addr 0x54 @asm "mov -0x4(%ebp),%eax"}
    \text{label pc\_0x54}
    \text{R\_EAX:u32 = mem:}u32[R\_EBP:u32 + -4:u32, \text{e\_little}]:u32
    ...

  - **Postcondition:**
    \((R\_EAX \times R\_EAX \leq \text{xparam}) \& \ (R\_EAX+1:u32) \times (R\_EAX+1:u32) > \text{xparam}) \& ...
Actual code verification

Handlers’ Contract is in the form of Hoare triples

Example:

Precondition

\neg (\text{mem:}\text{u32}[\text{R}_{EAX}:\text{u32}-8:\text{u32}, \text{e_little}:\text{u32}] \leq 0x10) \& (\text{R}_{EAX} \cdot \text{R}_{EAX} \leq \text{xparam}) \& \ldots

Body

\text{addr} 0x54 \ @ \text{asm} \ "\text{mov} -0x4(\%ebp),\%eax" \ label \ pc_0x54
\text{R}_{EAX}:\text{u32} = \text{mem:}\text{u32}[\text{R}_{EAX}:\text{u32} + -4:u32, \text{e_little}:\text{u32}]
\ldots

Postcondition

(\text{R}_{EAX} \cdot \text{R}_{EAX} \leq \text{xparam}) \& (\text{R}_{EAX} + 1:u32) \cdot (\text{R}_{EAX} + 1:u32) > \text{xparam}) \& \ldots

The validation task is simplified by:

1. No self-modifying code
2. Loop-free code for handlers
3. Structured loop for boot strap
4. No nested interrupts
Actual code verification

Handlers’ Contract is in the form of Hoare triples.

- Prove that \( \{ P \} \text{handler} \{ Q \} \) is a valid Hoare-triple
  1. compute the weakest precondition (WP) of handler, Q
  2. check that \( P \Rightarrow WP \)
  3. automate as much as possible

- Automation is done using a SMT solver.
- Requires \( (P \Rightarrow WP) \equiv \forall \bar{x}. R \), where \( R \) is quantifier free.
Actual code verification

Handlers’ Contract is in the form of Hoare triples.

- Prove that $\{P\}handler\{Q\}$ is a valid Hoare-triple
  1. compute the weakest precondition (WP) of $\text{handler}$, $Q$
  2. check that $P \Rightarrow WP$
  3. automate as much as possible

- Automation is done using a SMT solver.
- Requires $(P \Rightarrow WP) \equiv \forall \vec{x}. R$, where $R$ is quantifier free.
- Exploit BAP to compute the weakest precondition
**Binary Analysis Platform (BAP)**

- Framework providing several binary program analysis techniques.
- BAP back-end can extract CFG and PDG, to perform symbolic execution and to compute WP.
- Back-end utilities reason on BAP Intermediate Language (BIL).

First step is converting the code to “BIL”

```plaintext
label name:
R0: u32 = 0x1: u32 + PC + mem[R2]: u32
cond: bool = R0 > R1
mem? u32 = mem with [R3]: u32 = 5: u32
cjmp cond, "name1", "name2"
jmp "name"
```

- A BIL program is a sequence of statements.

```
program ::= var := exp | jmp(exp) | cjmp(exp, exp, exp) |
           assert(exp) | halt(exp) | label(string)
```
A new front-end for BAP

- BAP front-end does not support ARMv7 binaries.
- The new front-end for ARMv7 is called Lifter.
- It’s built on top of the HOL4 `arm_steps` library.
- Steps:
  1. HOL4 ARM states ⇒ BIL states,
  2. ARM instructions ⇒ BIL fragments.
Lifting instructions

Given an instruction:

\[(c_1, t_1) \ldots (c_n, t_n)\] = \(\text{arm\_steps} \ (\ldots \text{state} \ldots, \text{inst})\)

- \(c_i\): condition.
- \(t_i\): state transition function.
Lifting instructions

Given an instruction:

- $[(c_1, t_1), \ldots, (c_n, t_n)] = \text{arm\_steps} (\ldots\text{state}\ldots, \text{inst})$
  - $c_i$: condition.
  - $t_i$: state transition function.

Symbolic evaluation

(regs, psrs, coregs, mem)

Machine registers

Co-processor registers

System memory

Machine flags (cpsr, psr{svc,abort, undef, irq})
Given an instruction:

\[ [(c_1, t_1), \ldots, (c_n, t_n)] = \text{arm_steps} (\ldots \text{state} \ldots, \text{inst}) \]

- \( c_i \): condition.
- \( t_i \): state transition function.

**Symbolic evaluation**

Goal: HOL4 ARM state terms \( \Rightarrow \) BIL expressions
Lifting expressions

Soundness of the lifter depends on the correct transformation of HOL4
terms to the BIL expressions.

- Our certifying lifter is constructed by:
  1. Formal model of BIL expressions in HOL4,
  2. Theorems showing the equivalence of HOL4 terms vs. composed BIL expressions,
     \[ \text{sideCondition} \implies (\alpha = \beta) \]
  3. ML module to convert the HOL4 terms into BIL counterparts, and
  4. Serialization mechanism.

- To guarantee the soundness of conversion, the \textit{liftExp} module
  “certifies” its output:

\[ \text{liftExp}(\text{exp}) = (\text{exp}', \vdash \text{exp} = \text{exp}') \]
Effective application of the verification strategy required the implementation of several tools.

Our tool chain consists of following tools:

- Indirect jumps resolver
- Integration of HOL4/BAP and GDB
- Optimization of the weakest precondition algorithm of BAP
Simple iterative algorithm to resolve indirect jumps

Verifying a Hoare triple by computing the weakest precondition depends on knowing the CFG.

- This needs to resolve the indirect jumps.
- Two sources of indirect jumps:
  - Function pointers: e.g. C-handler addresses
  - Function exit points: pop pc, b lr
Integration of HOL4/BAP and GDB

Why is it useful?
To extract:
- Data types,
- Location, alignment and size of data-structure fields,
- Content of static memory used to represent constant values.
Optimization of BAP WP computation algorithm

The size weakest precondition can be exponential.

- WP of sequentially composed conditionals can be optimized.

Example:

- WP of $R_1 > 10$ for the addeq $R_1 \#7;\text{muleq } R_1 \#2$

What has been generated before optimization:

$$Z \Rightarrow (Z \Rightarrow ((R_1 + 7) \cdot 2 > 10) \land \neg Z \Rightarrow ((R_1 + 7) > 10))$$

$$\land \neg Z \Rightarrow (Z \Rightarrow (R_1 \cdot 2 > 10) \land \neg Z \Rightarrow (R_1 > 10))$$

What we got after optimization:

$$(Z \Rightarrow ((R_1 + 7) \cdot 2 > 10)) \land (\neg Z \Rightarrow (R_1 > 10))$$
Optimization of BAP WP computation algorithm

The size weakest precondition can be exponential.

- WP of sequentially composed conditionals can be optimized.

Example:

- WP of $R_1 > 10$ for the `addeq R1 #7;muleq R1 #2`

What has been generated before optimization:

$$Z \Rightarrow (Z \Rightarrow ((R1 + 7) \ast 2 > 10) \land \neg Z \Rightarrow ((R1 + 7) > 10))$$

$$\neg Z \Rightarrow (Z \Rightarrow (R1 \ast 2 > 10) \land \neg Z \Rightarrow (R1 > 10))$$

What we got after optimization:

$$(Z \Rightarrow ((R1 + 7) \ast 2 > 10)) \land (\neg Z \Rightarrow (R1 > 10))$$
What has been presented:

- Verification of a tiny ARMv7 hypervisor.
- 3k ARM instructions $\Rightarrow$ 7k lines of BIL in about an hour.
- Automation is done using HOL4, the BAP utilities and STP solvers.
Concluding remarks

-ranking

Three main constraints:

- The MMU setup is restricted to identity virtual-to-physical address translation,
- The kernel does not allow preemption,
- Only one physical core is used.
Three main constraints:

- The MMU setup is restricted to identity virtual-to-physical address translation,
- The kernel does not allow preemption,
- Only one physical core is used.

Ongoing:

- Dynamic MMU for HOL4 ARM model to relax the first constraint.
Thanks for your attention!