A system for testing specifications of CPU semantics

or,
What I did on my summer vacation

Lindsey Kuper
The TSL testing problem
The TSL testing problem

Lim, J, and Reps, T., "A System for Generating Static Analyzers from Machine Instructions", CC '08
The TSL testing problem

- **TSL** (Transformer Specification Language) lets us **generate static analyzers from specifications**. Great!

Lim, J, and Reps, T., "A System for Generating Static Analyzers from Machine Instructions", CC '08
The TSL testing problem

• **TSL** (Transformer Specification Language) lets us generate *static analyzers from specifications*. Great!
The TSL testing problem

- **TSL** (Transformer Specification Language) lets us generate static analyzers from specifications. Great!
The TSL testing problem

- TSL (Transformer Specification Language) lets us generate static analyzers from specifications. Great!
The TSL testing problem

- TSL (Transformer Specification Language) lets us generate static analyzers from specifications. Great!
The TSL testing problem

- **TSL** (Transformer Specification Language) lets us generate static analyzers from specifications. Great!
The TSL testing problem

- **TSL** (Transformer Specification Language) lets us generate static analyzers from specifications. Great!
The TSL testing problem

- **TSL** (Transformer Specification Language) lets us **generate static analyzers from specifications**. Great!
The TSL testing problem

- TSL (Transformer Specification Language) lets us generate static analyzers from specifications. Great!

- But how do we know if the generated analysis engines (multiplicatively many!) are correct?
Our approach
Our approach

ISA spec
(IA32, PPC, ...)
written in TSL language

TSL system

Analysis spec
aka "interpretation"

Analysis engine
Our approach

- Narrow the focus to testing just the ISA specs.
Our approach

- Narrow the focus to testing just the ISA specs.
Our approach

- Narrow the focus to testing just the ISA specs.
- Can we really *isolate* an ISA spec? We can come close by using EMUL, the “simplest” interpretation.
Our approach

- Narrow the focus to testing just the ISA specs.
- Can we really isolate an ISA spec? We can come close by using EMUL, the “simplest” interpretation.
Our approach

- Narrow the focus to testing just the ISA specs.
- Can we really isolate an ISA spec? We can come close by using EMUL, the “simplest” interpretation.
Our approach

- Narrow the focus to testing just the ISA specs.
- Can we really *isolate* an ISA spec? We can come close by using EMUL, the “simplest” interpretation.
- And for now, start with IA32.
Goal for the summer
Goal for the summer

- Find out how complete and precise our IA32 TSL specification is...
Goal for the summer

• Find out how complete and precise our IA32 TSL specification is...

• ...by generating an IA32 emulator, then comparing the emulator to the real processor.
Goal for the summer

• Find out how complete and precise our IA32 TSL specification is...
  • ...by generating an IA32 emulator, then comparing the emulator to the real processor.
  • If resulting states differ on the same inputs, the spec was (probably) buggy.
Goal for the summer

• Find out how complete and precise our IA32 TSL specification is...
  • ...by generating an IA32 emulator, then comparing the emulator to the real processor.
  • If resulting states differ on the same inputs, the spec was (probably) buggy.
• We already have all the pieces: IA32 spec, EMUL, and a third-party tool for testing CPU emulators. This will be easy, right?!
How to test a CPU emulator
How to test a CPU emulator

EmuFuzzer’s design

Emulated environment

Randomly chosen state

 interpret instruction

$E_0$

"Concretize"

$E_I$

$E_I' = \equiv$

Physical environment

add eax, 4

 Execute instruction

$P_0$

$P_I$

"Abstract"

But wait!

We don’t only want to test emulators.

EmuFuzzer’s design

Emulated environment

Randomly chosen state

Interpret instruction

? =

Physical environment

Execute instruction

add eax, 4


Friday, August 20, 2010
How to test a CPU emulator

EmuFuzzer’s design

Emulated environment

Randomly chosen state

E₀

"Concretize"

Interpret instruction

E₁

? ≡ E₁′

Physical environment

P₀

add eax, 4

Execute instruction

P₁

α

"Abstract"

How to test a CPU emulator
How to test any analysis engine
How to test any analysis engine

Our design

Abstract environment

Physical environment

Randomly (or deliberately) chosen state

add eax, 4

Execute instruction
Look-thru memory
Look-thru memory

"Hey, what did you say eax was again?"

E₀

α
does nothing
at first!

P₀

"eax = 2"
Look-thru memory

• The ability to *lazily* instantiate the emulator's state (memory and registers) from that of the process as each instruction is being emulated.
TSL validator main loop
TSL validator main loop

- Halt test program $P$ at instruction $i$
- Repeat for instruction $i + 1$
- Clear emulator's memory
- ?
- Try to single-step $P$ on instruction $i$
- Write fault? Unprotect and try again
- Emulate $i$ using look-thru memory
- Write-protect all of $P$'s memory pages
TSL validator main loop

- Halt test program \( P \) at instruction \( i \)
  - Repeat for instruction \( i + 1 \)

- Clear emulator's memory

- Write fault? Unprotect and try again

- Emulate \( i \) using look-thru memory
  - Write-protect all of \( P \)'s memory pages

- Try to single-step \( P \) on instruction \( i \)
TSL validator main loop

1. Halt test program \( P \) at instruction \( i \)
2. Repeat for instruction \( i + 1 \)
3. Clear emulator's memory
4. ?
5. Write-protect all of \( P \)'s memory pages
6. Try to single-step \( P \) on instruction \( i \)
7. Write fault? Unprotect and try again

Diagram:

- **Abstract environment**
  - \( E_0 \) → \( \alpha \) (\textit{Abstract})
  - \( E_1 \)
  - \( E_1' \)

- **Physical environment**
  - \( P_0 \) → \( \alpha \) (\textit{Abstract})
  - \( p_1 \)
  - \( \text{add eax, 4} \)
  - \( \text{Execute instruction} \)
TSL validator main loop

1. Halt test program $P$ at instruction $i$
   - Repeat for instruction $i + 1$

2. Clear emulator's memory

3. Write-protect all of $P$'s memory pages

4. Emulate $i$ using look-thru memory

5. Try to single-step $P$ on instruction $i$
   - Write fault? Unprotect and try again

6. Abstract environment
   - $E_0$ -> $E_1$ -> $E_1'$
   - Interpret instruction

7. Physical environment
   - $P_0$
   - Randomly (or deliberately) chosen state
   - $P_1$
   - Execute instruction
   - $add eax, 4$
TSL validator main loop

1. Halt test program $P$ at instruction $i$
2. Repeat for instruction $i + 1$
3. Clear emulator's memory
4. Write-protect all of $P$'s memory pages
5. Try to single-step $P$ on instruction $i$
6. Write fault? Unprotect and try again

Abstract environment:
- $E_0$
- $E_i$
- $E_i'$

Physical environment:
- $P_0$
- $P_i$

Randomly (or deliberately) chosen state

add eax, 4
 Execute instruction
TSL validator main loop

1. Clear emulator's memory
2. Repeat for instruction i + 1
3. Halt test program P at instruction i
4. Emulate i using look-thru memory
5. Write-protect all of P's memory pages
6. Try to single-step P on instruction i
7. Write fault? Unprotect and try again

Abstract environment:
- $E_0$ (Abstract)
- $E_1$, $E_1'$

Physical environment:
- $P_0$ (Randomly or deliberately chosen state)
- add eax, 4
- $P_1$ (Abstract)
- Execute instruction
TSL validator main loop

1. Halt test program $P$ at instruction $i$
2. Repeat for instruction $i + 1$
3. Clear emulator's memory
4. State comparison is tricky
5. Emulate $i$ using look-thru memory
6. Write-protect all of $P$'s memory pages
7. Try to single-step $P$ on instruction $i$
8. Write fault? Unprotect and try again

Abstract environment:
- $E_0$ (Abstract)
- $E_1$
- $E_1'$
- $P_0$
- $P_1$

Physical environment:
- Randomly (or deliberately) chosen state
- $\alpha$
- Add eax, 4
- Execute instruction

Friday, August 20, 2010
TSL validator main loop

Halt test program $P$ at instruction $i$

Repeat for instruction $i + 1$

Clear emulator's memory

State comparison is tricky

Emulate $i$ using look-thru memory

Write-protect all of $P$'s memory pages

Try to single-step $P$ on instruction $i$

Write fault? Unprotect and try again
TSL validator main loop

- Halt test program $P$ at instruction $i$
- Repeat for instruction $i+1$
- Clear emulator's memory
- ?
- State comparison is tricky
- Emulate $i$ using look-thru memory
- Write-protect all of $P$'s memory pages
- Try to single-step $P$ on instruction $i$
- Write fault? Unprotect and try again
Future work:
short-term stuff
Future work: short-term stuff

- The hard part of state comparison: identify changed locations on the real process side, and compare them with corresponding locations on the emulator side.
Future work: short-term stuff

- The hard part of state comparison: identify changed locations on the real process side, and compare them with corresponding locations on the emulator side.

- Better logging and reporting: eventually, we’d like to have a “dashboard”.
Future work: short-term stuff

- **The hard part of state comparison**: identify changed locations on the real process side, and compare them with corresponding locations on the emulator side.

- Better logging and reporting: eventually, we’d like to have a “dashboard”.

- How will we deal with test programs that “misbehave”? 
Future work:
long-term stuff
Future work: long-term stuff
Future work: long-term stuff

- Support for more ISAs. (x64, at least!)
Future work: long-term stuff

- Support for more ISAs. (x64, at least!)
- **Support for abstract interpretations**, not just EMUL.
Future work: long-term stuff

- Support for more ISAs. (x64, at least!)
- **Support for abstract interpretations**, not just EMUL.
- Find ways to choose which inputs to test that will be most likely to turn up bugs in a specification.
What I learned
What I learned

- Emulators, debuggers, and static analyzers are not made of magic
What I learned

• Emulators, debuggers, and static analyzers are not made of magic

• First real systems programming experience: didn’t quite cross the kernel space boundary, but came right up next to it
What I learned

- Emulators, debuggers, and static analyzers are not made of magic
- First real systems programming experience: didn’t quite cross the kernel space boundary, but came right up next to it
- A metric for how much I can accomplish in 13 weeks
What I learned

- Emulators, debuggers, and static analyzers are not made of magic
- First real systems programming experience: didn’t quite cross the kernel space boundary, but came right up next to it
- A metric for how much I can accomplish in 13 weeks
- Finally convinced that OOP is good for something
Thank you!

Questions?
(exit)