Ben Greenman December 10, 2015
Secure Compilation to Protected Module Architectures

Outlines a fully-abstract compilation scheme from a source language with objects, types, and exceptions to a corresponding bytecode [1]. The main technical content is an algorithm for disproving contextual equivalence of source programs using their bytecode.

Strengths

- The object-oriented source language was very interesting, because it allows generating **new** private regious at runtime.
- Made good use of related work (though, Assumption 1 requires proof for a Java-like language). Theorem 5.9 stated alone would be a nice result, but it was more entertaining to see it used in context.

Weaknesses

- I am not convinced that contrapositive of Theorem 5.9 holds in this setting. The proof seems to say: "our attacker could not distinguish the target components, therefore no attacker could distinguish them." I would prefer a direct proof that target contextual equivalence implies source contextual equivalence.
- As the paper put more things in the protected code segment, I became less convinced that the attack model was interesting. How large is the protected segment? How realistic is it to assume no attacker can corrupt the protected memory?
- Performance is obviously bad—I wish the authors would justify a target overhead. Maybe 10x slowdown is acceptable for applications with strict security requirements.

  It was also frustrating to see arguments using asymptotic complexity in Section 3. At the hardware level, speed of memory is really all that matters.

References

[1] Marco Patrignani, Pieter Agten, Raoul Strackx, Bart Jacobs, Dave Clarke, and Frank Piessens. Secure compilation to protected module architectures. In *TOPLAS*, 2015.