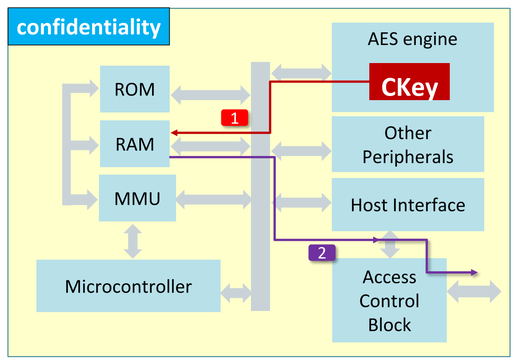
TOWARDS SCALABLE SOC SECURITY VALIDATION

Sujit Kumar Muduli
Department of Computer Science and Engineering, IIT Kanpur

Introduction

- Confidentiality
  - secret information should not flow to untrusted region
- Integrity
  - no information flow from untrusted region to secure location



[Subramanyan, P., Arora, D., DATE'14])

Integrity is a *dual* of confidentiality

Confidentiality and integrity are 2-safety properties

- Properties refuted by observing ***two*** finite traces
- A ***trace*** is sequence of execution states, $t = s_0 s_1 \dots s_n$

- 2-safety property is from the class of Hyperproperties

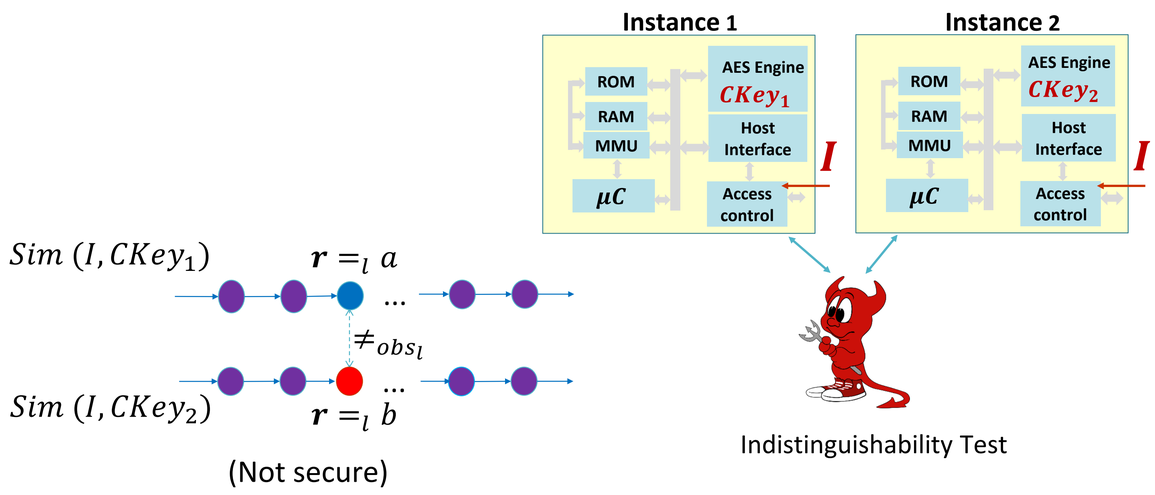
Problem Statement

Proving confidentiality and integrity

show system leaks no secret information

or

show execution traces are indistinguishable to untrusted entity



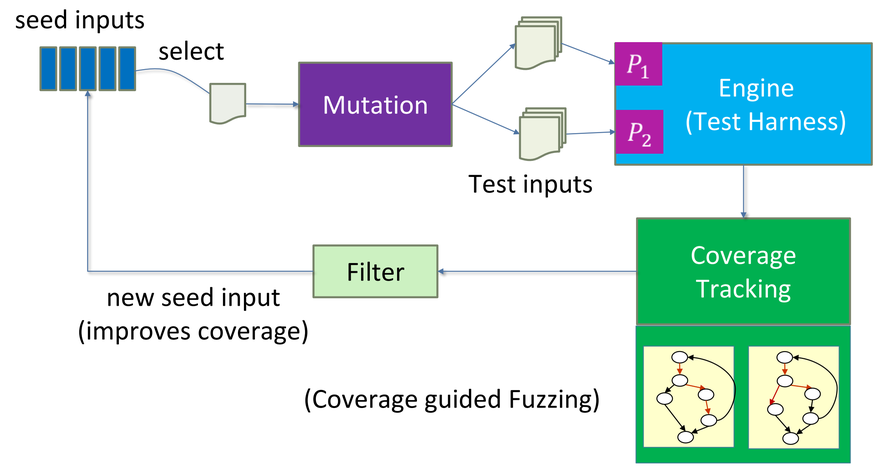
Goal :

- automated technique for finding 2-safety hyperproperty violations
- language for specifying security properties

Prior Art

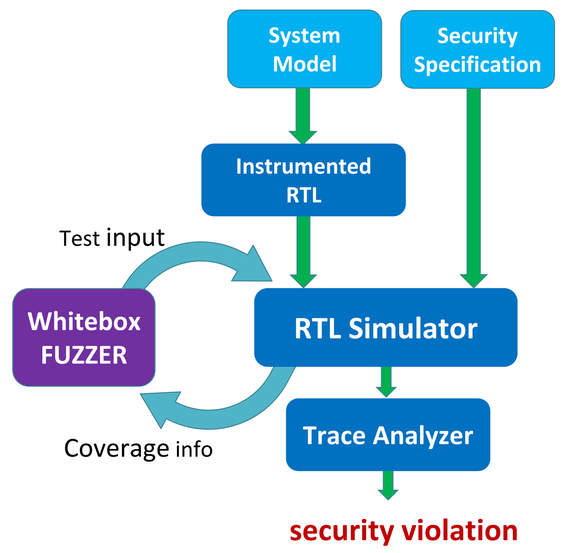
SPECIFICATION	Non-interference Observational Determinism HyperLTL HyperCTL*	❑Goguen, J. A., & Meseguer, J. Security policies and security models. ❑Zdancewic, S., et al. Observational determinism for concurrent program. ❑HyperLTL and HyperCTL* are extension of LTL and CTL* [CFK+14]
VERIFICATION	Information Flow Tracking (GLIFT, HDL LIFT)	❑Hardware design is converted to a **GLIFT** logic for verification [HOI+11] ❑SecVerilog, Caiosson and Sapper uses information flow type systems at HDL **GLIFT : much overhead** **HDL LIFT : doesn't guarantee no violations in runtime**
	Model Checking	❑Barthe, et al. "Secure information flow by self-composition." ❑Terauchi, T., & Aiken, A. "Secure information flow as a safety problem." ❑Sousa, M., & Dillig, I."Cartesian hoare logic for verifying k-safety properties." ❑Subramanyan, P., et al."Verifying information flow using symbolic execution." **computationally expensive ➔ not scalable**
FUZZING	American Fuzzy Lop (afl)	❑Fuzzing is used for finding software vulnerabilities ❑AFL uses genetic algorithm to discover new test inputs [Zal14] **can't be used for 2-safety violation SoCs** **scalable**

Technical Challenges



- Specification Language
- Mutation Algorithm
  - Need new algorithms to generate inputs that trigger 2-safety violations
- Test harness
  - Modify test harness engine to work with two system instances
- Coverage tracking
  - Mechanism to find new seed inputs
  - Ensures fuzzer does not revisit inputs
  - Need new coverage metrics to explore product state space

Proposed System Architecture



Hardware modeling

- **Verilog** implementation

Intermediate representation

- Instrument Verilog model to collect simulation metric

Fuzzer

- A variant of AFL to be used along with Hyperproperties

Conclusion

- Existing solutions for SoC security validation are not effective
- Fuzzing has the potential to be scalable.
- It has been successful in finding many software vulnerabilities.
- HYPERFUZZING leverages power of fuzzing to find security violations in SoCs.

References

1. Godefroid, Patrice, Michael Y. Levin, and David Molnar. "SAGE: whitebox fuzzing for security testing." Communications of the ACM 55.3 (2012): 40-44.
2. Bounimova, Ella, Patrice Godefroid, and David Molnar. "Billions and billions of constraints: Whitebox fuzz testing in production." Proceedings of the 2013 International Conference on Software Engineering. IEEE Press, 2013.
3. Clarkson, Michael R., and Fred B. Schneider. "Hyperproperties." Journal of Computer Security 18.6 (2010): 1157-1210.
4. Barthe, Gilles, Pedro R. D'Argenio, and Tamara Rezk. "Secure information flow by self-composition." Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.. IEEE, 2004.
5. Terauchi, Tachio, and Alex Aiken. "Secure information flow as a safety problem." International Static Analysis Symposium. Springer, Berlin, Heidelberg, 2005.