

# New Static Analysis Techniques to Detect Entropy Failure Vulnerabilities

---

Andrew Russell & Rushi Shah

December 12, 2018

The University of Texas at Austin

# Entropy Failures: A Historical Perspective

1. OpenSSL
2. FreeBSD

## How Could This Have Been Avoided?

- Audit code once, then use relational verification to prove you haven't introduced bugs with small changes to program.
- This differential approach works well with the way software is written (CI, etc.)

## Background/Initial Approach

*Statement*  $S$   $:=$   $A$   
                   $| S_1 ; S_2$   
                   $| \text{if } p \text{ then } S_1 \text{ else } S_2$   
                   $| \text{while } p \text{ } S$   
*Predicate*  $p$   $:=$   $\top$   $| \perp$   $| A$   $| \neg p$   $| p \odot p$   
*Operator*  $\odot$   $:=$   $\wedge$   $| \vee$

# Taint Analysis

- Definition, terminology, uses, etc.
- Sources are legitimate sources of entropy (/dev/random/)
- Sinks are things like cryptographic algorithms (KDF)
- But taint analysis is unsound when used on two versions of the program (overapproximation)

# Predicate Abstraction

- Finer grained version of taint analysis across versions of a program.
- Taint set of variable in program two should be a superset of taint set of variable in program one.

# Predicate Abstraction

- Finer grained version of taint analysis across versions of a program.
- Taint set of variable in program two should be a superset of taint set of variable in program one.
- Weaker version: if  $v_1$  is the taint set of variable  $v$  passed to sink in program one and  $v_2$  is the taint set of  $v$  in program two, then we would like

*`assert( $v_1 == v_2$ )`*



- Off-the-shelf state-of-the-art: CPAChecker

- Sequential Composition

$$S_1 ; S_2$$

- Sequential Composition

$$S_1 ; S_2$$

- Synchronized Composition

$$S_1 \otimes S_2$$

# Product Programs

- Sequential Composition

$$S_1 ; S_2$$

- Synchronized Composition

$$S_1 \otimes S_2$$

- Hybrid?

# Algorithm

# High level overview

1. Instrumentation
2. Product Program
3. Assertions + CPAChecker

- Replace sources with labelled constants
- Perform taint analysis on sources to generate environment  $\Gamma$  which marks statements involving tainted variables.

- For values that are tainted by more than one source (for example  $S_1 + S_2$ ) replace with one of two uninterpreted functions over the sources:

1. *preserving*( $s_1, s_2, \dots, s_n$ )
2. *nonPreserving*( $s_1, s_2, \dots, s_n$ )

- Preserving functions are  $+$ , XOR, etc.
- Non-preserving functions are left or right shift, etc.



# Sequential Product Program

# Naive Synchronized Product Program

# Heuristic-Optimized Synchronized Product Program





## Future Work



## Conclusion



# Acknowledgements

Prof. Hovav Shacham and Prof. Isil Dillig

