MODEL-BASED GREY-BOX FUZZING

“Fuzzing the Shall-Nots”

HCSS APRIL 31, 2019

DAVID GREVE
DAVID.GREVE@COLLINS.COM

Work sponsored in part by DARPA/AFRL Contract FA8750-16-C-0218.
The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.
Approved for Public Release, Distribution Unlimited

Collins Aerospace

TALK OVERVIEW

- **Model-Based Test Generation and Fuzzing**
- Testing –vs- Fuzzing
- Environmental Models
- Fuzzing Requirements Framework
- Fuzzing for Credit

MODEL-BASED TEST GENERATION

- Given:
  - A Model of the System (Requirements)
  - Simulink, SpeAR, DSL
  - Mathematical Description
- Objective:
  - Generate Tests that Satisfy Stringent Coverage Criteria
  - Multiple-Condition/Decision-Coverage (MC/DC)
- Methodology:
  - Express Testing Objectives as Logical Constraints
  - Generate Tests Using Constraint Solver

- Historically Labor Intensive Activity
- High-Coverage Tests Generated Automatically (from Requirements)

CREW ALERTING SYSTEM: PROBLEM

- The logic for displaying a CAS message driven by complex Boolean equations
- Each airplane program contains a thousand or more such equations and each need to be thoroughly tested
- Example:

```
ID: TENC_OIL_PRESS_SB1
Logic:
    TDT2S.SB1_PRESS_LOW OR
    TDT2S.SB1_PRESS_HIGH OR
    TDT500MS.(SB1_PRESS_LOW AND SB1_PRESS_HIGH);

Inhibit: LANDING
^**
```

- The complexity of CAS equations can be overwhelming:
  - Contain numerous logical conditions (not unusual for 10 or more to appear in an equation)
  - Reference other equations
  - Reference previous versions of variables, including the equation other test.
  - May be inhibited by other equations

“Formal Methods for Certification”, Lucas Wagner

Collins Aerospace

CREW ALERTING SYSTEM: IMPACT

Model Based Test Generation

- Constraint solver employed to generate tests that satisfy "MC/DC" coverage metric.
- Generated thousands of tests covering ~95% of equations under test.

Future:

- Test generator is scheduled for use on every program as standard work.

Table A-7 Verification of Verification Process Results

	Objective		Activity	Applicability by Software Level				Output		Control Category by Software Level			
	Description	Ref	Ref	A	B	C	D	Data Item	Ref	A	B	C	D
1	Test procedures are correct.	6.4.5.b	6.4.5	●	○	○		Software Verification Results	11.14	②	②	②	
2	Test results are correct and discrepancies explained.	6.4.5.c	6.4.5	●	○	○		Software Verification Results	11.14	②	②	②	
3	Test coverage of high-level requirements is achieved.	6.4.4.a	6.4.4.1	●	○	○	○	Software Verification Results	11.14	②	②	②	②
4	Test coverage of low-level requirements is achieved.	6.4.4.b	6.4.4.1	●	○	○		Software Verification Results	11.14	②	②	②	
5	Test coverage of software structure (modified condition/decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●				Software Verification Results	11.14	②			
6	Test coverage of software structure (decision coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●	●			Software Verification Results	11.14	②	②		
7	Test coverage of software structure (statement coverage) is achieved.	6.4.4.c	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d 6.4.4.3	●	●	○		Software Verification Results	11.14	②	②	②	
8	Test coverage of software structure (data coupling and control coupling) is achieved.	6.4.4.d	6.4.4.2.c 6.4.4.2.d 6.4.4.3	●	●	○		Software Verification Results	11.14	②	②	②	
9	Verification of additional code, that cannot be traced to Source Code, is achieved.	6.4.4.c	6.4.4.2.b	●				Software Verification Results	11.14	②			

"Formal Methods for Certification", Lucas Wagner

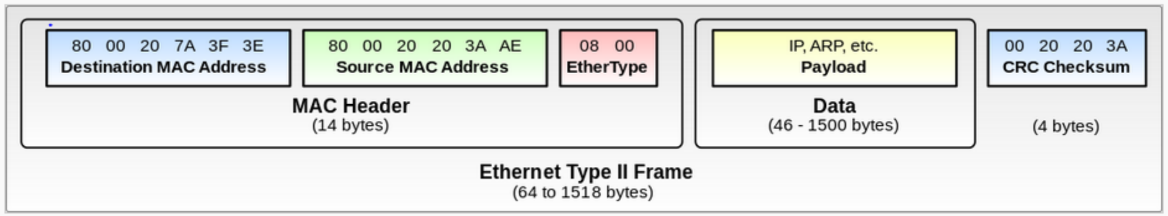
FUZZING (FUZZ TESTING)

- Robustness Testing
  - Apply Random, Invalid or Unexpected Inputs
- Monitor Health of System
  - Exceptions, Lock-Up, Memory Usage, Power Consumption, etc.
- Anomalous Behavior
  - May Reveal Exploitable Vulnerability
  - Record Inputs for Later Forensic Analysis
- Cyber Grand Challenge
  - Fuzzing Used Extensively for Automated Penetration Testing

> The original work was inspired by being logged on to a modem during a storm with lots of line noise. And the line noise was generating junk characters that seemingly was causing programs to crash. The noise suggested the term "fuzz".
>
> --Barton Miller, University of Wisconsin (1988)

SMART FUZZING

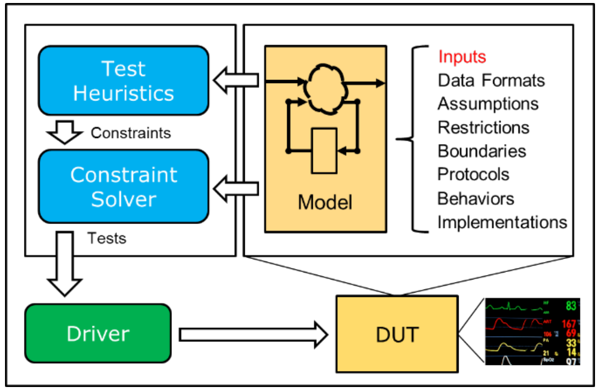
- Smart Fuzzing Frameworks
  - Sulley, Peach, scapy
- Format Specifications (Templates)
  - Random Inputs are “Constructed” by filling in blanks in Templates
- Enables Detection of Deeper Bugs
  - Passes CRC Check

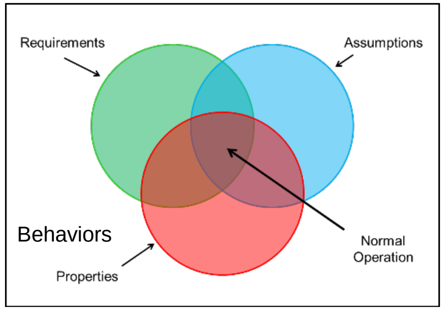


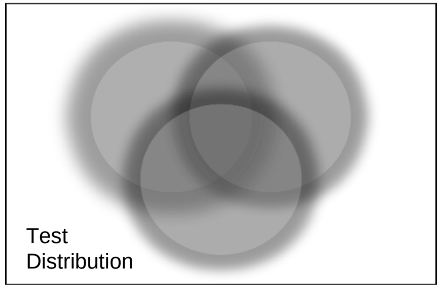
The most common Ethernet Frame format, type II

MODEL-BASED FUZZING

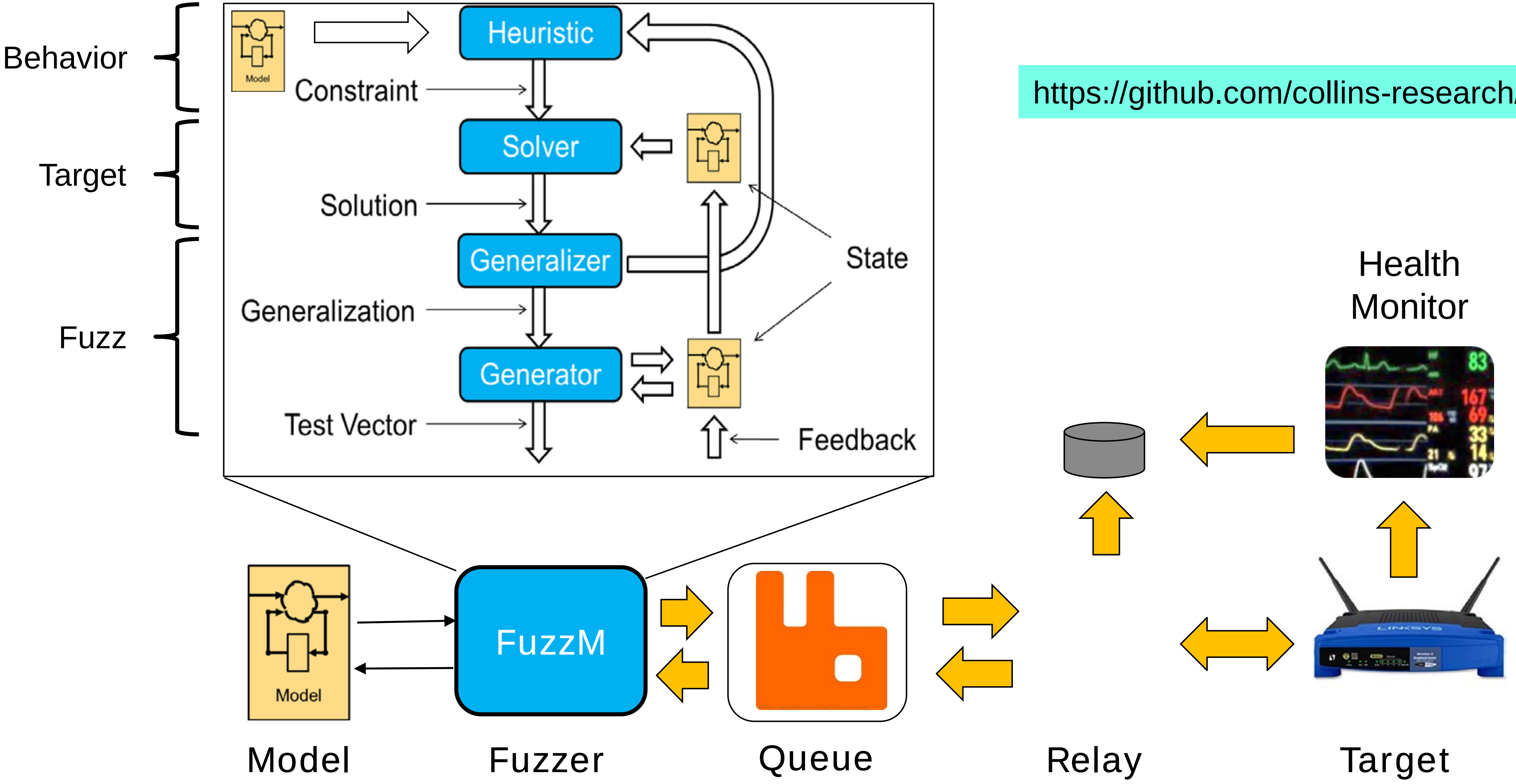
- *Model* Describes Fuzzing Target
  - Description Includes Behavior
    - Not Just Data Formats
  - Can Describe Stateful Behaviors
    - Fragment/Reassemble Message
- *Constraint Solver* Generates Tests
  - Tests are “Deduced”, not “Constructed”
  - Constraints capture “Interesting Behaviors”
- Constraint Solving + Fuzzing
  - Solver **Targets** Behaviors we **Know**
  - Fuzzer **Explores** Behaviors we **Don’t Know**







FUZZM COMPONENT ARCHITECTURE



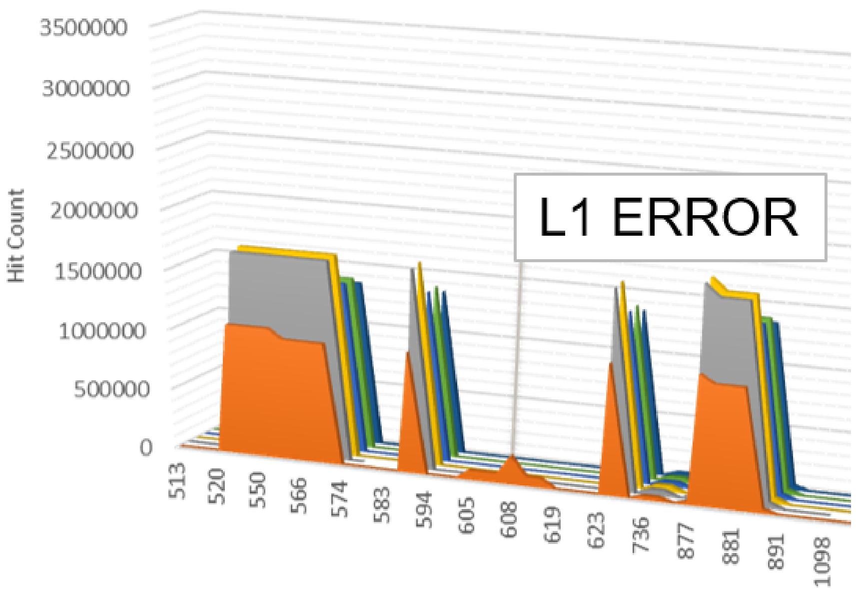
https://github.com/collins-research/FuzzM

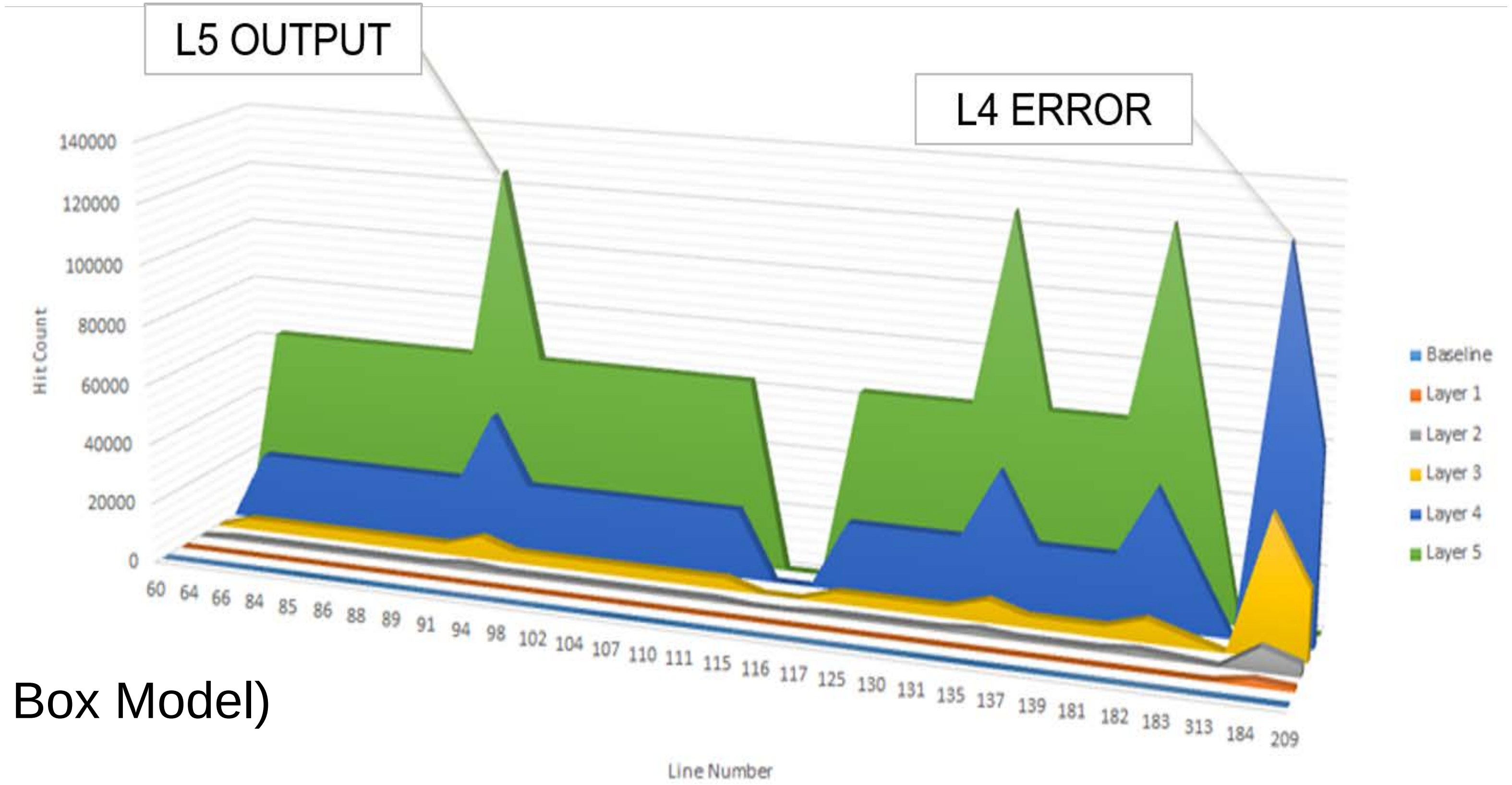
Collins Aerospace

LAYERED REQUIREMENTS MODEL

OSI Model				
Layer			Protocol data unit (PDU)	Function[3]
Host layers	7	Application	Data	High-level APIs, including resource sharing, remote file access
	6	Presentation		Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption
	5	Session		Managing communication sessions, i.e. continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes
	4	Transport	Segment, Datagram	Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing
Media layers	3	Network	Packet	Structuring and managing a multi-node network, including addressing, routing and traffic control
	2	Data link	Frame	Reliable transmission of data frames between two nodes connected by a physical layer
	1	Physical	Symbol	Transmission and reception of raw bit streams over a physical medium

LAYERED MODEL COVERAGE RESULTS





Baseline : No Requirements (Black Box Model)

…

Layer 5 : Complete Requirements Model

Collins Aerospace

FUZZER COVERAGE COMPARISONS

AFL Boofuzz Hongfuzz Radamsa FuzzM

Missed Coverage

Unique Coverage

Collins Aerospace

TALK OVERVIEW

- Model-Based Test Generation and Fuzzing
- **Testing –vs- Fuzzing**
- Environmental Models
- Fuzzing Requirements Framework
- Fuzzing for Credit

TESTING –VS- FUZZING

Testing

- Methodology
  - Apply (Crafted) Inputs
  - Measure Outputs
    - Compare against expected Oracle
- Abstraction
  - Underspecified Behavior
  - “Oracle Equality” Challenging

Fuzzing

- Methodology
  - Apply (Random) Inputs
  - Monitor Health
    - Compare against Nominal Behavior
- Relaxed Oracle
  - Makes Fuzzing “Easier”
- If Fuzzing Violates Assumptions
  - Behavior is Unspecified
  - “Testing” is not possible

TESTING –VS- FUZZING

Testing

- Keys to Success
  - Strong Controllability
  - Strong Observability
  - Precise Oracle

Fuzzing

- Challenges
  - Controllability
  - Observability
  - Oracle Precision (Health)

TESTING –VS- FUZZING

Testing

- Limited Test Suite
  - Certification Tests
    - Cost of Development
    - Cost of Maintenance
    - Cost of Traceability
  - Production/Acceptance Tests (HW)
    - Cost of Test Evaluation Time
- Testing Metrics
  - Proxy for Effectiveness
  - Trade Quality for Quantity

Fuzzing

- “Unlimited” Test Suite
  - Fuzz and Forget
  - Continuous Integration
  - Production Testing
    - Offers little or no value
    - Not Detecting Manufacturing Defects
  - Acceptance Tests (?)
- Fuzzing Metrics
  - No Standard Metrics
  - Trade Quantity for Quality (?)

TESTING –VS- FUZZING

Testing

- SHALL
  - Typifies “Safety Requirement”
  - Property
    - forall (x): good(x)
  - Test
    - good(x0)
    - some (x): good (x)

Fuzzing

- SHALL NOT
  - Typifies “Security Requirement”
  - Property
    - not exists (x): bad(x)
      - forall (x): not bad(x)
  - Test
    - some (x): not bad(x)
  - Fuzz
    - foralot (x) : not bad(x)

TALK OVERVIEW

- Model-Based Test Generation and Fuzzing
- Testing –vs- Fuzzing
- **Environmental Models**
- Fuzzing Requirements Framework
- Fuzzing for Credit

MODEL-BASED FUZZING

- How does it differ from model (requirements) based test generation?
- What constitutes a fuzzing model?
- How does it compare to existing MDB artifacts?

REQUIREMENTS, ASSUMPTIONS AND OPERATING ENVIRONMENT

- Requirement Specifications
  - **Typically Include Assumptions**
  - Embedment Manual
    - Where and How can this system be used?

- Assumptions Constrain the **Environment**
  - We Found a Bug .. Here is the Trace!
    - **"That Would Never Happen In-System"**
      - .. but what if it does?
  - Assumptions Restrict the **Threat Model**

FUZZING STRAINS ENVIRONMENTAL MODELS

- Basic (Random)
  - Env. Assumption : Variable Bounds
  - Fuzzing Objective : Boundary and Combinatorial Testing
- Safety (Murphy)
  - Env. Assumption : Operational Envelope
  - Fuzzing Objective : Robustness
- Security (Malicious)
  - Env. Assumption : Deployment Threats/Risks
  - Fuzzing Objective : Resiliency

THE BAD-GUY

- Quantification in 1st order Logic
  - Replace quantified variable
  - With a function (skolem)
    - Not just any function ..
    - The “bad-guy” function
  - If there is a problem input
    - this function will find it!

- The bad-guy function
  - Aware of the “model”
  - Aware of the desired property
  - Computes “worst possible” value

- If property is true for bad-guy
  - The property is true for all inputs

forall (x) : not bad(x)

```
(iff (list-equiv x y)
     (and (equal (len x) (len y))
          (forall (a) (equal (nth a x) (nth a y)))))
```

```
(local
 (defun list-equiv-bad-guy (x y)
   (if (and (consp x) (consp y))
       (if (not (equal (car x) (car y))) 0
         (1+ (list-equiv-bad-guy (cdr x) (cdr y))))
     1)))

(local
 (defthm list-equiv-reduction
   (iff (list-equiv x y)
        (and (equal (len x) (len y))
             (equal (nth (list-equiv-bad-guy x y) x)
                    (nth (list-equiv-bad-guy x y) y))))
   :hints (("Goal" :in-theory (enable nth)))))
```

“**FUZZING MODELS**” ARE “**ENVIRONMENTAL MODELS**”

- The Most **Formidable** Environmental Models
  - Include a Model of the **Target** System
    - The Protocol it Speaks
    - The Mode it is In
    - The Input it Expects
  - Knowledge of the Target
    - Enables Effective “Attacks”
    - Bad-Guy
  - Murphy and Malicious Models
    - Will Always Have This Flavor
  - Still: Not Simply Unconstrained

TALK OVERVIEW

- Model-Based Test Generation and Fuzzing
- Testing –vs- Fuzzing
- Environmental Models
- **Fuzzing Requirements Framework**
- Fuzzing for Credit

REQUIREMENTS SPECIFICATION IN SPEAR

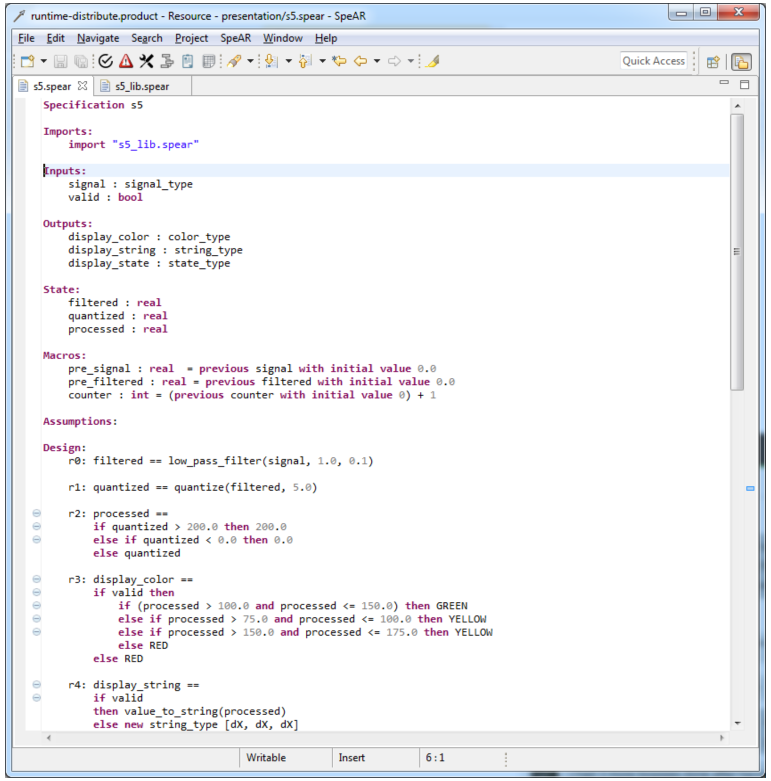
SpeAR =

Specification and **A**nalysis of **R**equirements

An Integrated development environment for formally specifying and rigorously analyzing requirements.

- Eclipse-based, Xtext language
- Formal methods driven analyses

- A specification language that's expressive as possible while still analyzable using state-of-the-art model checking tools.



https://github.com/lgwagner/SpeAR

SPEAR CORE CAPABILITIES

SPECIFICATION

Rich (as possible) specification language for formally describing how a system should operate.

- supports temporal predicates for describing event ordering
- type system that allows for efficient behavioral specification
- well-formedness checking
- supplemental static analyses

ANALYSES

A set of analyses to establish correctness, completeness, and consistency of requirements sets before actually building the system.

- logical entailment
- consistency and realizability
- traceability

FuzzM Integration

- UFC-Based Fuzzing Constraints
- Selectively Relaxed Assumptions

TALK OVERVIEW

- Model-Based Test Generation and Fuzzing
- Testing –vs- Fuzzing
- Environmental Models
- Fuzzing Requirements Framework
- **Fuzzing for Credit**

FUZZING IN THE LARGE

- Fuzzing Has Proven Effective
  - Finds Many Kinds of Issues
  - Implementation
    - Bugs in Corner Cases
  - **Requirements**
    - **Unintended/Emergent Behaviors**
    - Requirements (Assumption) Validation
  - Forces Consideration
    - Of Additional Use Cases
  - Fuzzing Can be “Cheap”
    - Fuzz and Forget

- Model-Based Fuzzing
  - Leverages, Extends MBD Paradigm
    - Constrained, Formidable Environmental Models
  - Automated Fuzz Test Generation
    - Targets Interesting Behaviors
  - Comparable to white-box fuzzing
    - Complete Requirements

FUTURE: FUZZING FOR CREDIT

- Emerging Security Certification Standards
  - Proposed ASISP amendment 14 CFR 25
  - Proposed EASA amendment 2019-01

- Measurements for Security
  - Effectiveness arguments often lack Rigor
  - Lacks Quantitative Measures

- **Fuzzing will Eventually be Part of the Assurance Story**
  - Safety
    - Robustness
  - Security
    - Resiliency
  - To Compete with Testing
    - Needs Rigor, Quantitative Measures

Fuzzing the Shall-Nots

TALK OVERVIEW

- Model-Based Test Generation and Fuzzing
- Testing –vs- Fuzzing
- Environmental Models
- Fuzzing Requirements Framework
- Fuzzing for Credit

Questions?