University of Victoria

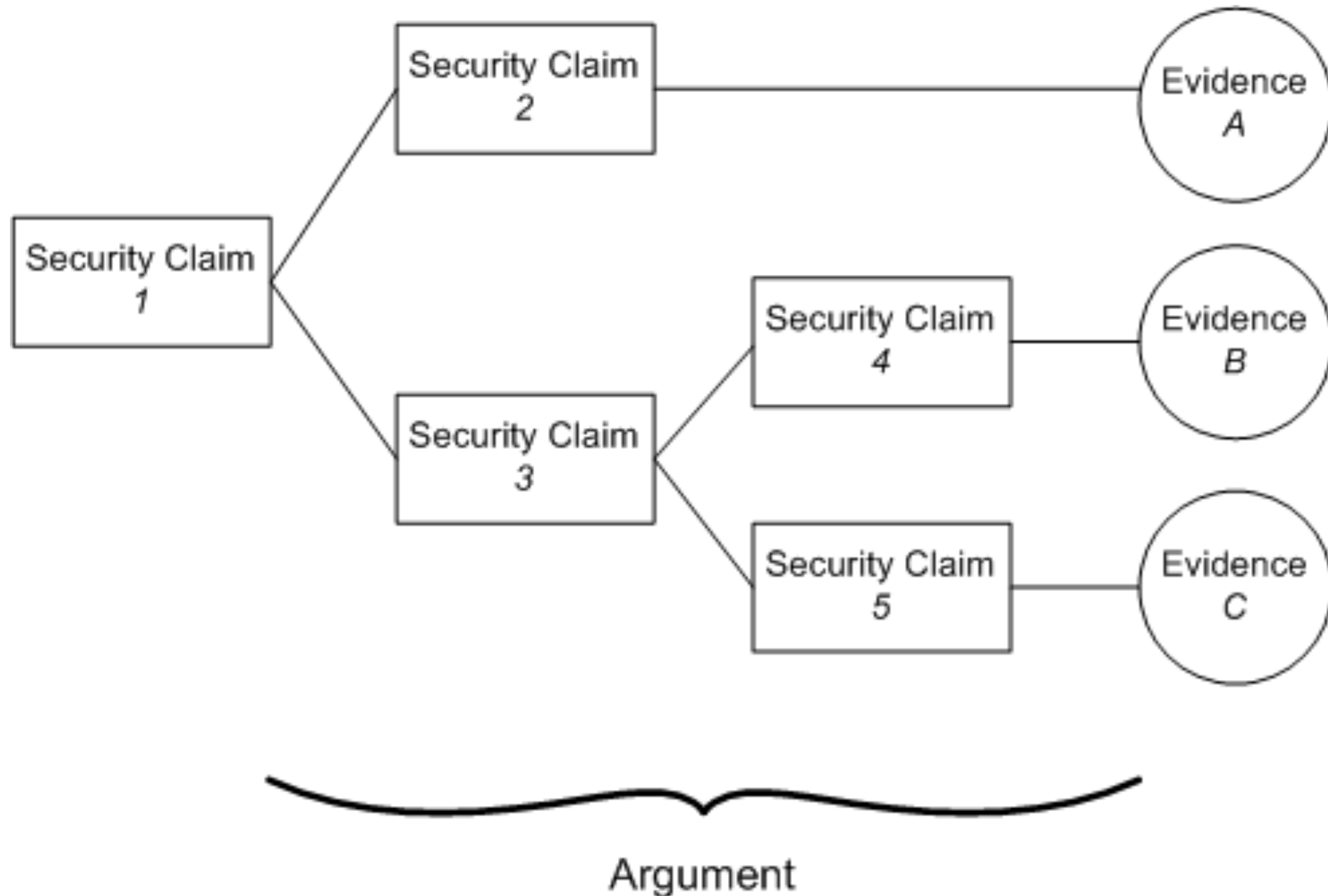
Engineering

Computer Science

Fuzz Testing for Creating Evidence in Security Assurance Cases

Jens Weber
University of Victoria, Canada

Security Assurance Cases

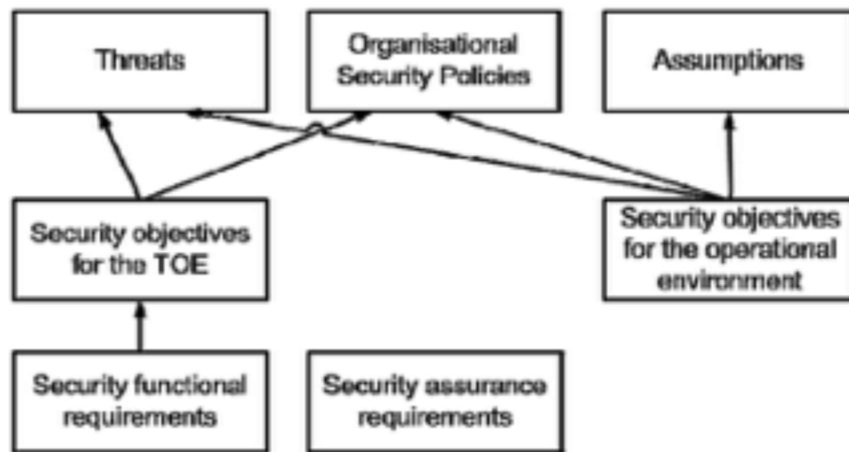


2 General Types of Security Claims

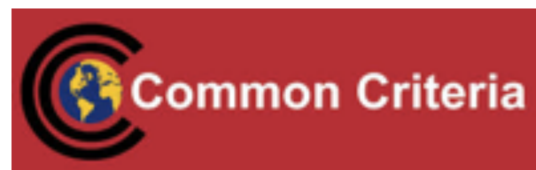
*(1) claims about the **requirements-based security properties** of a system and*

*(2) claims about the **absence of vulnerabilities** in the design or implementation that could be exploited to break the system's security model*

Claims about Requirements-based Security Properties



Common Criteria Evaluation Assurance Level (EAL)	Process rigor required for development of an IT product
EAL 1	Functionally tested.
EAL 2	Structurally tested.
EAL 3	Methodically tested and checked.
EAL 4	Methodically designed, tested and reviewed.
EAL 5	Semi-formally designed and tested.
EAL 6	Semi-formally verified, designed and tested.
EAL 7	Formally designed and tested.



2 General Types of Security Claims

(1) *claims about the* ***requirements-based security properties*** *of a system and*

(2) *claims about the* ***absence of vulnerabilities*** *in the design or implementation that could be exploited to break the system's security model*

"Indiana Jones Attack"

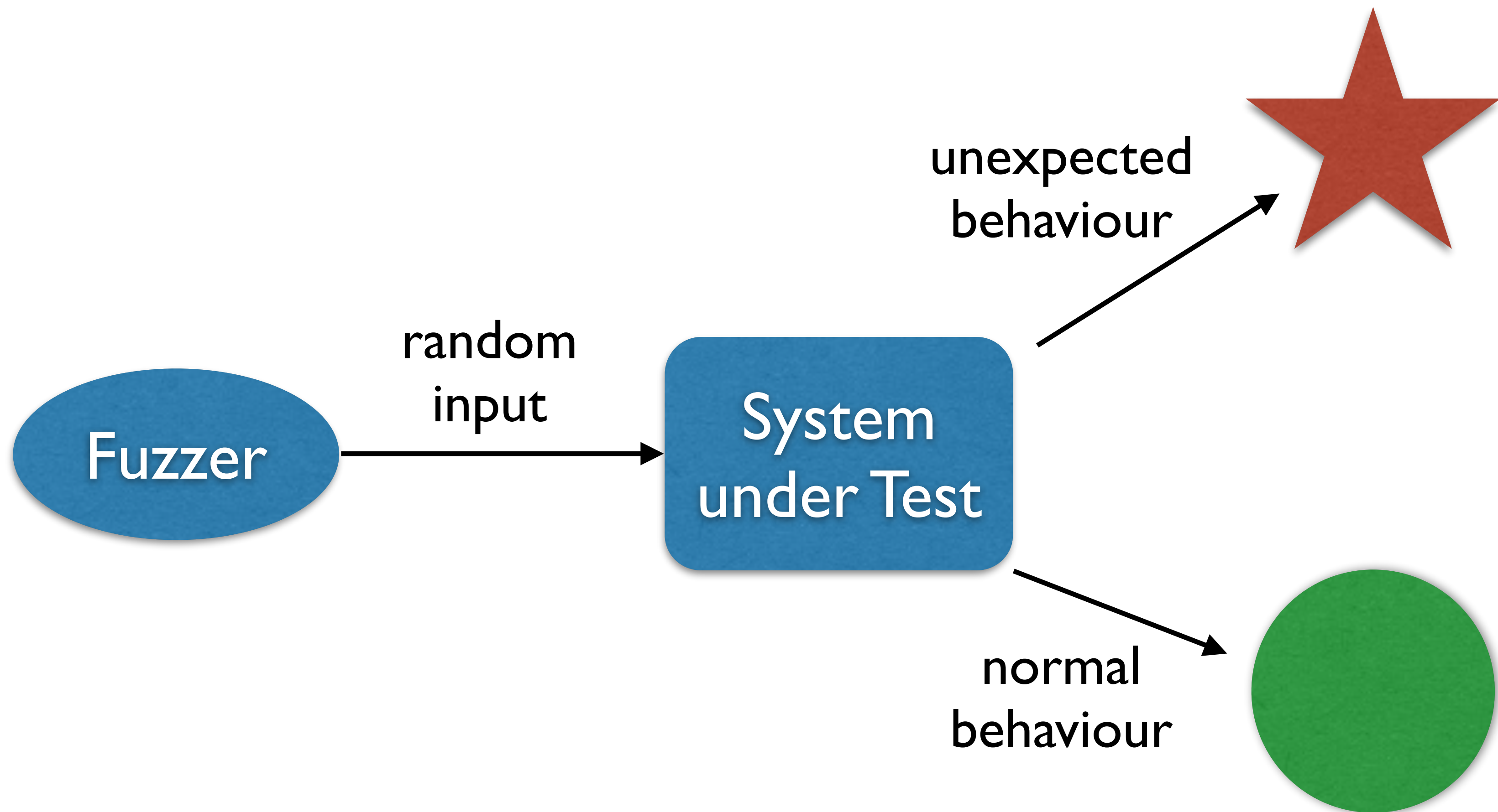
Weinstock, C. B., & Lipson, H. F. (2013).

Claims about Absence of Security Vulnerabilities

Testing efforts attempt to produce defeaters/rebuttals

- Penetration testing
- Random negative testing (Fuzz Testing)

Fuzz Testing



Fuzzing is a black-box testing technique that exposes the SUT to randomized input while observing its behaviour.

Evidence Template: Testing [Lipson & Weinstock]

Evidence	Results from running test of the system using attacks of type *A*
Type of Evidence	Technology & System => Test Results
Claim	The system is secure against attacks of type *A*.
Assumption	The tests can adequately exercise the code, and therefore the results are a reasonable measure of code quality.
Argument	Since the level of assurance needed isn't ultra-high, it is possible to run enough tests for long enough to make the results believable. The tests have good coverage of the software under test. The testing methods have been proven to work in other similar systems.

Weinstock, C. B., & Lipson, H. F. (2013). Evidence of Assurance: Laying the Foundation for a Credible Security Case. SEI - CMU Report.

Evidence Template: Testing [Lipson & Weinstock]

Evidence	Results from running test of the system using attacks of type *A*
Type of Evidence	Technology & System => Test Results
Claim	The system is secure against attacks of type *A*.
Assumption	The tests can adequately exercise the code, and therefore the results are a reasonable measure of code quality.
Argument	Since the level of assurance needed isn't ultra-high, it is possible to run enough tests for long enough to make the results believable. The tests have good coverage of the software under test. The testing methods have been proven to work in other similar systems.

Fuzz Testing does not generate specific evidence

Weinstock, C. B., & Lipson, H. F. (2013). Evidence of Assurance: Laying the Foundation for a Credible Security Case. SEI - CMU Report.

Evidence Template: Testing [Lipson & Weinstock]

Evidence	Results from running test of the system using attacks of type *A*
Type of Evidence	Technology & System => Test Results
Claim	The system is secure against attacks of type *A*.
Assumption	The tests can adequately exercise the code, and therefore the results are a reasonable measure of code quality.
Argument	Since the level of assurance needed isn't ultra-high, it is possible to run enough tests for long enough to make the results believable. The tests have good coverage of the software under test. The testing methods have been proven to work in other similar systems.

Random Testing results in shallow, poor code coverage

Weinstock, C. B., & Lipson, H. F. (2013). Evidence of Assurance: Laying the Foundation for a Credible Security Case. SEI - CMU Report.

Benefits and Limitations of Fuzzing

Fuzzing has been proven effective in finding security vulnerabilities (defeaters for security claims).

However, what evidence arises from unsuccessful fuzz testing?

Smart Fuzzing

Traditional Fuzzing is akin to firing a scatter gun

"Smart Fuzzing" uses some utility function to optimize the random data generation with respect to a predefined "*vulnerability pattern*"

Bekrar, S., Bekrar, C., Groz, R., & Mounier, L. (2011, March). Finding software vulnerabilities by smart fuzzing. In Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth Intl Conf. on (pp. 427-430). IEEE.

Two main issues

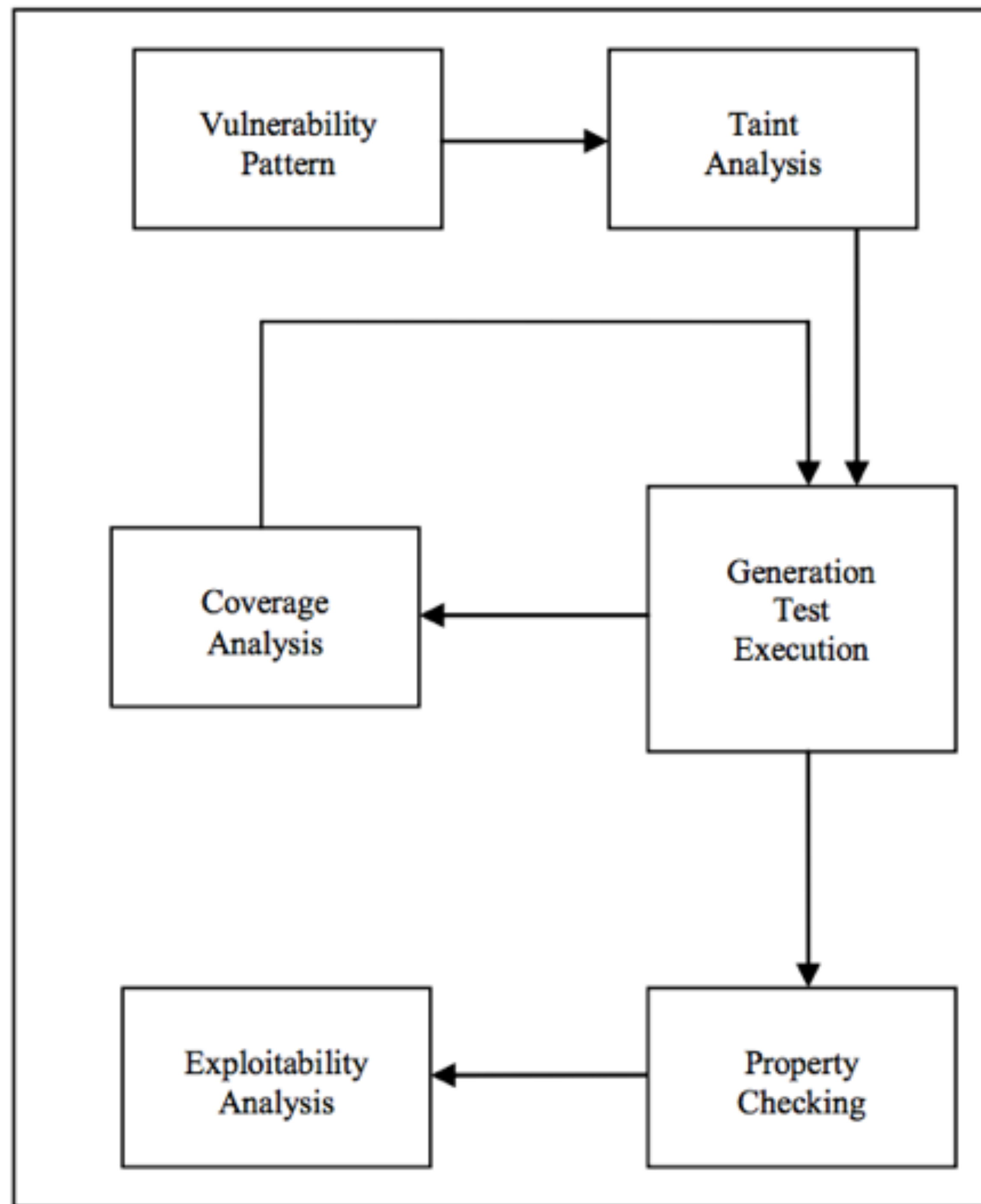
- How to find vulnerability patterns (targets)?
- How to guide the Fuzzer to target this code?

How to find vulnerable patterns

static code analysis and complexity metrics can be used to detect potential vulnerabilities, e.g., Buffer Overflows, Integer over/underruns

How to guide the Fuzzer to target vulnerabilities?

Taint Analysis



Bekrar, Sofia, et al. "A taint based approach for smart fuzzing." Software Testing, Verification and Validation (ICST), 2012 IEEE 5th Intl Conf. on., 2012.

Problems with Taint Analysis

Precision

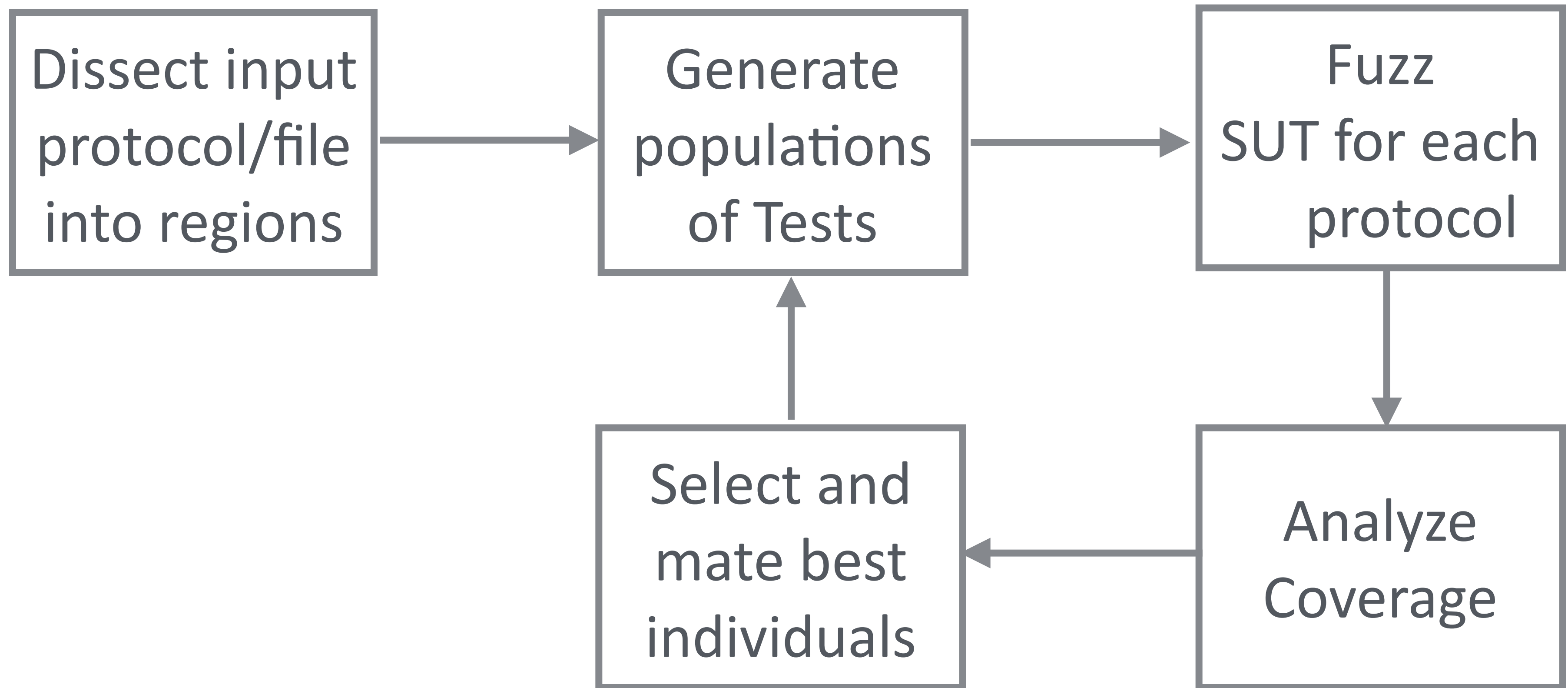
- Static taint analysis over approximates (many false positive dependencies)
- Dynamic taint miss (indirect) dependencies (control flow)

Tooling

- Taint analysis requires language-aware tooling
- Best suited for programs with several discrete inputs (as opposed to programs reading complex files / protocols)

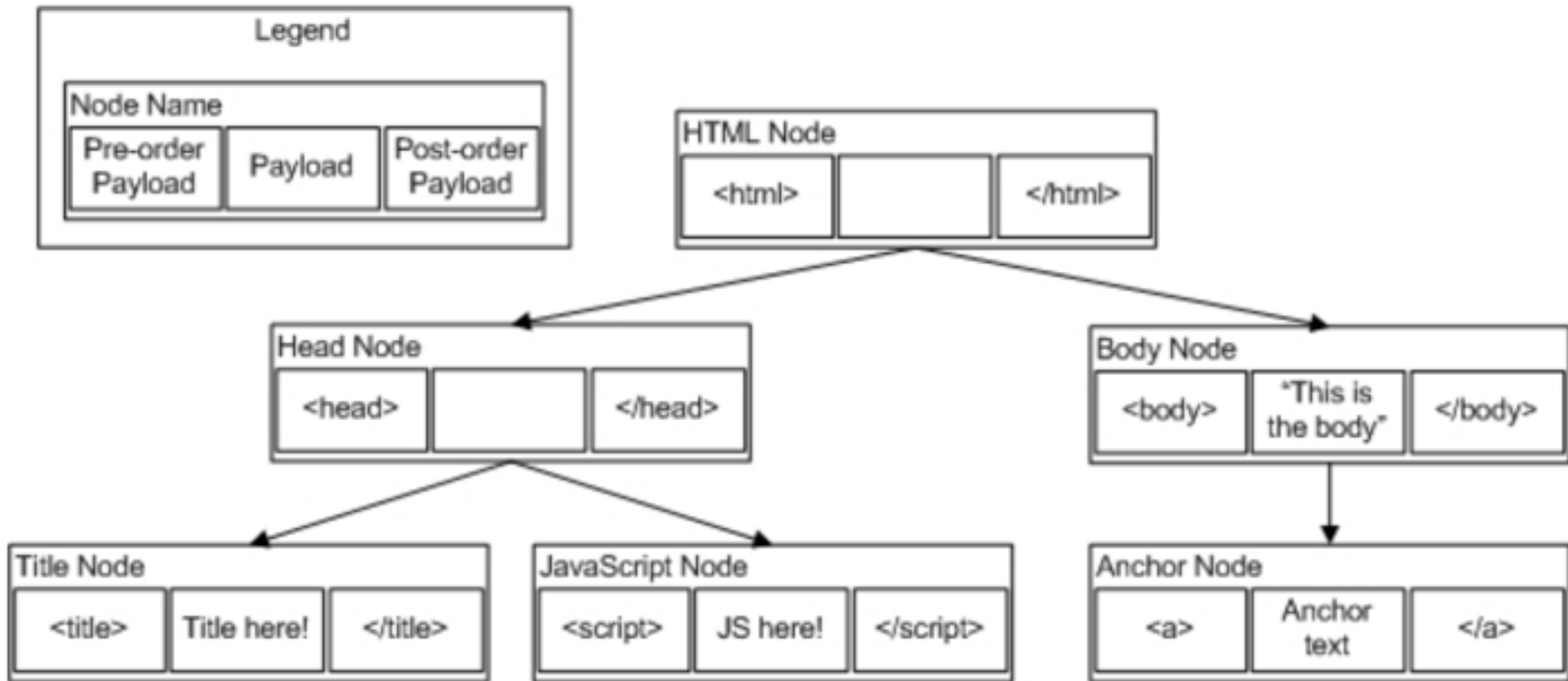
An alternative way to guide the Fuzzer

Evolutionary Fuzzing



Example: Fuzzing Web Browser

HTML Documents has many logical feature blocks



Infinitely many possible HTML documents, generated from protocol

Protocol "Block" for Anchor Nodes

```
# The following code (Python) will produce the anchor tag:
# <a alt="this will be fuzzed." href="127.0.0.1">So will this.</a>

s_static("<a alt=\"")
s_string("this will be fuzzed.")
s_static("\" href=\"")
s_string("127.0.0.1")
s_static("\">")

s_string("So will this.")

s_static("</a>")
```

Definition of Fuzzed Feature Blocks Evolutionary Algorithm

Example:

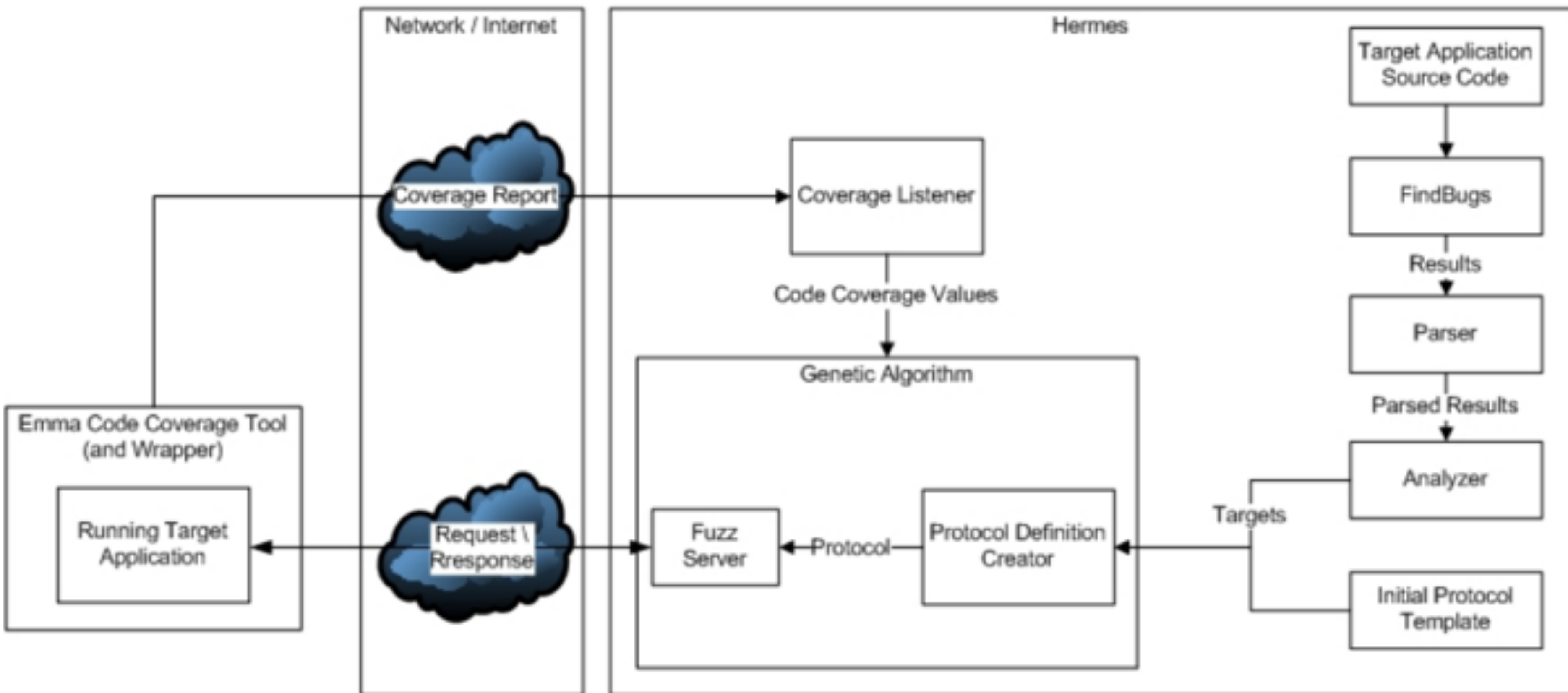
- Anchors,
- Images,
- Divs,
- IFrames,
- Objects,
- JavaScript, and
- Applets

(1, 0, 1, 1, 0, 0, 1)

```
    Author: HTMLTreeConstructor
    Description:
        Auto Generated Protocol Definition: Chromosome:
        [1, 0, 0, 0, 0, 0, 0]

    Source: PDHelpers.py
    Created: 2014-05-15 19:00:38.530000
'''

from sulley import *
import random

s_initialize("Protocol Definition")

s_static("<html>")
s_static("<head>")
s_static("<title>")
s_string("Sulley Says Hello!")
s_static("</title>")
s_static("</head>")
s_static("<body>")
```

```
# Beginning of block: body_block
if s_block_start("body_block"):
    s_string("body_block+assurance")

    # Beginning of block: body_block_a1_block
    if s_block_start("body_block_a1_block"):
        s_string("body_block_a1_block+assurance")

        # Begin <a> tag
        s_static("<a ")
        s_static("alt=\"")
        s_string("body_block_a1")
        s_static("\" ")
        s_static("href=\"127.0.0.1/")
        s_string("body_block_a1")
        s_static("\" ")
```

Experiment: Extend Sulley Framework with Evolutionary Protocol Generation



Genetic Algorithm Configuration

P(Crossover) = 0.5

P(Mutation) = 0.05

Number of Generations = 30

Individuals per Generation = 10

Selection Algorithm = Tournament Selection (size=3)

Target application: Crawler4J
https://github.com/yasserg/crawler4j

Experiment

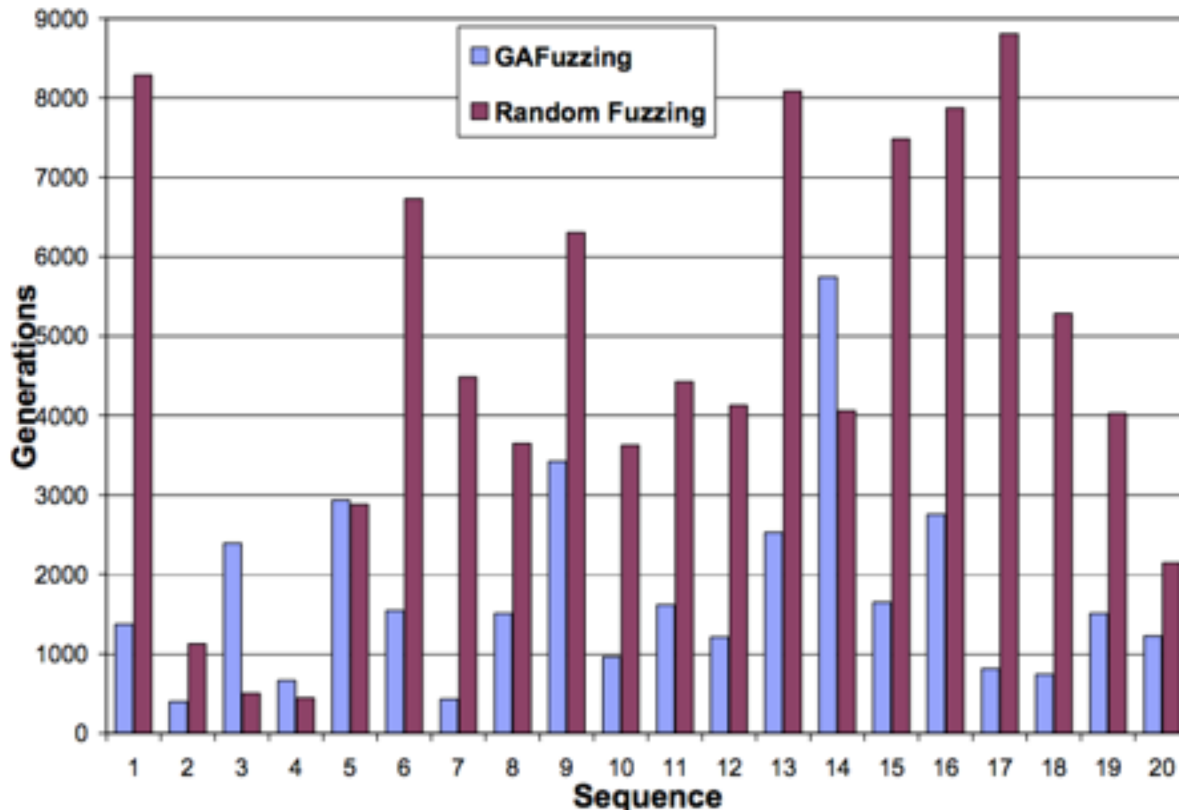
Use static code analysis to target most severe defects (as per FindBugs Bugrank metric)

Percentage (%)	Difference in Mean Coverage (%)	Difference in Std Deviation	Baseline # Mutations	Full Best-Fit # Mutations	Difference in # Mutations
10	0	0	67788	41964	25824
20	0	0.0025	67788	51648	16140
30	0	0	67788	41964	25824
40	0.0009	0.0001	67788	35508	32280

Speed-up between 24 and 48%

Shortt, Caleb James. Hermes: A Targeted Fuzz Testing Framework. MSc Thesis. University of Victoria, 2015.

A Similar Approach using Path Coverage: GA Fuzzer



Guang-Hong, Liu, et al. "Vulnerability analysis for x86 executables using genetic algorithm and fuzzing." Convergence and Hybrid Information Technology, 2008. ICCIT'08. 3rd Intl. Conf. on. Vol. 2. IEEE, 2008.

Discussion of the Evolutionary Approach

Benefits

- Can use existing Fuzzing frameworks / protocol definitions
- Solution not language-dependent

Limitation

- Complex / deep vulnerabilities may still not be triggered

```
x  :=  2*get_input(·)
y  :=  5 + x
goto  y
```

Schwartz, et al. "All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask)." Security and Privacy, 2010 IEEE Symposium on

Approach: Concolic Execution

- Combines Symbolic Execution with Concrete test case randomization
- Start with concrete test cases, trace execution conditions and systematically negate them. Use constraint solver to find data that satisfies alternate paths
  - -> Concolic Execution

Haller, Istvan, et al. "Dowsing for Overflows: A Guided Fuzzer to Find Buffer Boundary Violations." Usenix Security. 2013.

Limitations of Concolic Execution

- Scalability issues with symbolic execution
- Heuristics geared to specific vulnerability patterns
- Geared towards specific languages (heavy weight)

Summary and Question

- Fuzzing (Random negative testing) is an important tool for detecting security vulnerabilities
  - Has successfully been used for generating defeaters in AC
- Unsuccessful undirected fuzz testing provides weak (no) evidence in assurance cases
- Directed approaches utilize vulnerability pattern detection, taint analysis, evolutionary fuzzing, concolic execution, or a combination of these.
  - How to quantify the evidence created by these directed random tests?