CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

Fuzzing: A Tale of 🦸🦹 Cultures

Andreas Zeller

Fuzzing Workshop @ NDSS'22 • April 24, 2022

Fuzzing

Random Testing at the System Level

```
[;x1-GPZ+wcckc];,N9J+?#6^6\e?]9lu2_%'4GX"0VUB[E/r
~fApu6b8<{%siq8Zh.6{V,hr?;{Ti.r3PIxMMMv6{xS^+'Hq!AxB"YXRS@!
Kd6;wtAMefFWM(`|J_<1~o}z3K(CCzRH JIIvHz>_*.\>JrlU32~eGP?
lR=bF3+;y$3lodQ<B89!5"W2fK*vE7v{')KC-i,c{<[~m!]o;{.'}Gj\(X}
EtYetrpbY@aGZ1{P!AZU7x#4(Rtn!q4nCwqol^y6}0|
Ko=*JK~;zMKV=9Nai:wxu{J&UV#HaU)*BiC<),`+t*gka<W=Z.
%T5WGHZpI30D<Pq>&]BS6R&j?#tP7iaV}-}`\?[_[Z^LBMPG-
FKj'\xwuZ1=Q`^`5,$N$Q@[!CuRzJ2D|vBy!^zkhdf3C5PAkR?V hn|
3='i2Qx]D$qs4O`1@fevnG'2\11Vf3piU37@55ap\zIyl"'f,
$ee,J4Gw:cgNKLie3nx9(`efSlg6#[K"@WjhZ}r[Scun&sBCS,T[/
vY'pduwgzDlVNy7'rnzxNwI)(ynBa>%|b`;`9fG]P_0hdG~$@6
3]KAeEnQ7lU)3Pn,0)G/6N-wyzj/MTd#A;r
```

Bart Miller

Coined “Fuzzing”

An Empirical Study of the Reliability

of

UNIX Utilities

Barton P. Miller
bart@cs.wisc.edu

Lars Fredriksen
L.Fredriksen@att.com

Bryan So
so@cs.wisc.edu

(1989)

Test Generation

Test Routines Based on Symbolic Logical Statements*

RICHARD D. ELDRED

Datamatic, Newton Highlands, Massachusetts

(1958)

SELECT--A FORMAL SYSTEM FOR TESTING AND DEBUGGING PROGRAMS BY SYMBOLIC EXECUTION*

(1975)

Robert S. Boyer
Bernard Elspas
Karl N. Levitt
Computer Science Group
Stanford Research Institute
Menlo Park, California 94025

Generating Test Programs from Syntax

By

W. H. Burkhardt, Camden, N. J.

With 18 Figures

(Received October 27, 1966)

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-10, NO. 4, JULY 1984

An Evaluation of Random Testing

JOE W. DURAN, MEMBER, IEEE, AND SIMEON C. NTAFOS, MEMBER, IEEE

Test Generation

Test Routines Based on Symbolic Logical Statements*

Richard D. Eldred

Datamatic, Newton Highlands, Massachusetts

(1958)

Generating Test Programs from Syntax

By

W. H. Burkhardt, Camden, N. J.

With 18 Figures

(Received October 27, 1966)

SELECT--A FORMAL SYSTEM FOR
TESTING AND DEBUGGING PROGRAMS
BY SYMBOLIC EXECUTION*

(1975)

Robert S. Boyer
Bernard Elspas
Karl N. Levitt
Computer Science Group
Stanford Research Institute
Menlo Park, California 94025

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-10, NO. 4, JULY 1984

An Evaluation of Random Testing

JOE W. DURAN, MEMBER, IEEE, AND SIMEON C. NTAFOS, MEMBER, IEEE

What's new?

Fuzzing

An Empirical Study of the Reliability

of

UNIX Utilities

Barton P. Miller
bart@cs.wisc.edu

Lars Fredriksen
L.Fredriksen@att.com

Bryan So
so@cs.wisc.edu

(1989)

Test Generation

test *own* programs

Fuzzing

test *other* programs

Software Engineering

test *own* programs

(“Test generation”)

Security

test *other* programs

(“Fuzzing”)

Software Engineering

test *own* programs

Security

test *other* programs

“Test generation” = “Fuzzing”

Software Engineering

test *own* programs

- You want to find **bugs**
- You are willing to invest **lots of time**
- You have the **source code**
- You have a **spec** and/or **test cases**
- You have an idea **where the bugs are**
- You have an idea of **what a bug is**
- You need **guidance** for testing decisions

Security

test *other* programs

- You want to find **bugs**
- You are not willing to invest **lots of time**
- You do not have the **source code**
- You may have some **inputs** or **traces**
- You have no idea **where the bugs are**
- You want **crashes** (= possible exploits)
- You want **full automation**

Two Cultures – Assumptions

- You want to find **bugs**
- You are willing to invest **lots of time**
- You have the **source code**
- You have a **spec** and/or **test cases**
- You have an idea **where the bugs are**
- You have an idea **what a bug is**
- You need **guidance** for testing decisions

- You want to find **bugs**
- You are not willing to invest **lots of time**
- You do not have the **source code**
- You may have some **inputs** or **traces**
- You have no idea **where the bugs are**
- You want **crashes** (= possible exploits)
- You want **full automation**

Two Cultures – Reviewing

- Focus is on **future**
- Want **guidance** for testing decisions
- Focus on **conceptual** improvements
- Want explanations on how **numbers** come to be
- Want details on **decisions** and **rationales**
- Want to know when and why things will **not** work
- Expect **open science** principles

- Focus is on **here and now**
- Expect **vulnerabilities** or even **exploits**
- Want **hard-to-test systems** as benchmarks
- Want no assumptions; expect **full automation**
- Expect **scalability** and **versatility**
- Expect you are **better** than others
- Expect tools + data only **after acceptance**

Two Cultures: Software Engineering vs Security

- “I have a great fuzzing technique that needs nothing except a full formal specification of …”

- Want no assumptions
- Expect **full automation**

Two Cultures: Software Engineering vs Security

- “I have a great Python fuzzer that improves code coverage by 50%”

- Expect **vulnerabilities** or even **exploits**
- Want **hard-to-test systems** as benchmarks

Two Cultures: Software Engineering vs Security

- Focus on **conceptual** improvements
- Want explanations on how **numbers** come to be
- Want to know when and why things will **not** work

- “I have applied fuzzing on <system> and found 3 new CVEs”

Two Cultures: Software Engineering vs Security

- Want details on **decisions** and **rationales**
- Want to know when and why things will **not** work
- Expect **open science** principles

- "After hyperparameter tuning, our method performs up to 10% better on the benchmark than the competition"

test *own* programs

test *other* programs

test *any* program

test *any* program

test *any* program

test any program

You want **full automation –** but also **control**:

- If you have **inputs** or **traces,** use them
- If you have the **source code,** use it
- If you have a **spec** and/or **test cases,** use them
- If you have an idea **where the bugs are,** use it
- If you have an idea of **what a bug is,** use it

Taming Fuzzers

One Fuzzer {

You want **full automation – but also control**:

- If you have **inputs** or **traces,** use them
- If you have the **source code,** use it
- If you have a **spec** and/or **test cases,** use them
- If you have an idea **where the bugs are,** use it
- If you have an idea of **what a bug is,** use it

Sneak Peek

Specifying Inputs

Context-Free Grammars

⟨xml-tree⟩ ::= ⟨xml-openclose-tag⟩
  | ⟨xml-open-tag⟩ ⟨inner-xml-tree⟩ ⟨xml-close-tag⟩
⟨inner-xml-tree⟩ ::= ⟨text⟩ | ⟨xml-tree⟩
  | ⟨inner-xml-tree⟩ ⟨inner-xml-tree⟩
⟨xml-open-tag⟩ ::= ‘<’⟨id⟩‘>’ | ‘<’⟨id⟩‘␣’⟨xml-attribute⟩‘>’
⟨xml-close-tag⟩ ::= ‘</’⟨id⟩‘>’
⟨xml-openclose-tag⟩ ::= ‘<’⟨id⟩‘/>’ | ‘<’⟨id⟩‘␣’⟨xml-attribute⟩‘/>’
⟨xml-attribute⟩ ::= ⟨id⟩‘="’⟨text⟩‘"’
  | ⟨xml-attribute⟩ ‘␣’ ⟨xml-attribute⟩

<O cfg="ej45"><Qr hh="21"></P>…

Specifying Inputs

Context-Free Grammars

⟨xml-tree⟩ ::= ⟨xml-openclose-tag⟩
    | ⟨xml-open-tag⟩ ⟨inner-xml-tree⟩ ⟨xml-close-tag⟩
⟨inner-xml-tree⟩ ::= ⟨text⟩ | ⟨xml-tree⟩
    | ⟨inner-xml-tree⟩ ⟨inner-xml-tree⟩
⟨xml-open-tag⟩ ::= ‘<’⟨id⟩‘>’ | ‘<’⟨id⟩‘␣’⟨xml-attribute⟩‘>’
⟨xml-close-tag⟩ ::= ‘</’⟨id⟩‘>’
⟨xml-openclose-tag⟩ ::= ‘<’⟨id⟩‘/>’ | ‘<’⟨id⟩‘␣’⟨xml-attribute⟩‘/>’
⟨xml-attribute⟩ ::= ⟨id⟩ ‘="’ ⟨text⟩ ‘"’
    | ⟨xml-attribute⟩ ‘␣’ ⟨xml-attribute⟩

⟨text⟩ ::= ‘"; DROP TABLE Students; --’ | ⟨other-text⟩

Specifying Inputs

Context-Free Grammars

⟨xml-tree⟩ ::= ⟨xml-openclose-tag⟩
  | ⟨xml-open-tag⟩ ⟨inner-xml-tree⟩ ⟨xml-close-tag⟩
⟨inner-xml-tree⟩ ::= ⟨text⟩ | ⟨xml-tree⟩
  | ⟨inner-xml-tree⟩ ⟨inner-xml-tree⟩
⟨xml-open-tag⟩ ::= ‘<’⟨id⟩‘>’ | ‘<’⟨id⟩‘␣’⟨xml-attribute⟩‘>’
⟨xml-close-tag⟩ ::= ‘</’⟨id⟩‘>’
⟨xml-openclose-tag⟩ ::= ‘<’⟨id⟩‘/>’ | ‘<’⟨id⟩‘␣’⟨xml-attribute⟩‘/>’
⟨xml-attribute⟩ ::= ⟨id⟩‘="’⟨text⟩‘"’
  | ⟨xml-attribute⟩ ‘␣’ ⟨xml-attribute⟩

THIS IS <u>NOT</u> XML

<O cfg="ej45"><Qr hh="21"></P>…

Specifying Inputs

The ISLa Specification Language

⟨xml-tree⟩ ::= ⟨xml-openclose-tag⟩
  | ⟨xml-open-tag⟩ ⟨inner-xml-tree⟩ ⟨xml-close-tag⟩
⟨inner-xml-tree⟩ ::= ⟨text⟩ | ⟨xml-tree⟩
  | ⟨inner-xml-tree⟩ ⟨inner-xml-tree⟩
⟨xml-open-tag⟩ ::= ‘<’⟨id⟩‘>’ | ‘<’⟨id⟩‘␣’⟨xml-attribute⟩‘>’
⟨xml-close-tag⟩ ::= ‘</’⟨id⟩‘>’
⟨xml-openclose-tag⟩ ::= ‘<’⟨id⟩‘/>’ | ‘<’⟨id⟩‘␣’⟨xml-attribute⟩‘/>’
⟨xml-attribute⟩ ::= ⟨id⟩‘="’⟨text⟩‘"’
  | ⟨xml-attribute⟩ ‘␣’ ⟨xml-attribute⟩

Grammar

+

Constraints

forall <xml-tree> tree="<{⟨id⟩ opid}[⟨xml-attribute⟩]>
⟨inner-xml-tree⟩
</{⟨id⟩ clid}>" in start:
(= opid clid)

THIS IS XML

<O cfg="ej45"><Qr hh="21"></Qr>…

Input Constraints

The ISLa Specification Language

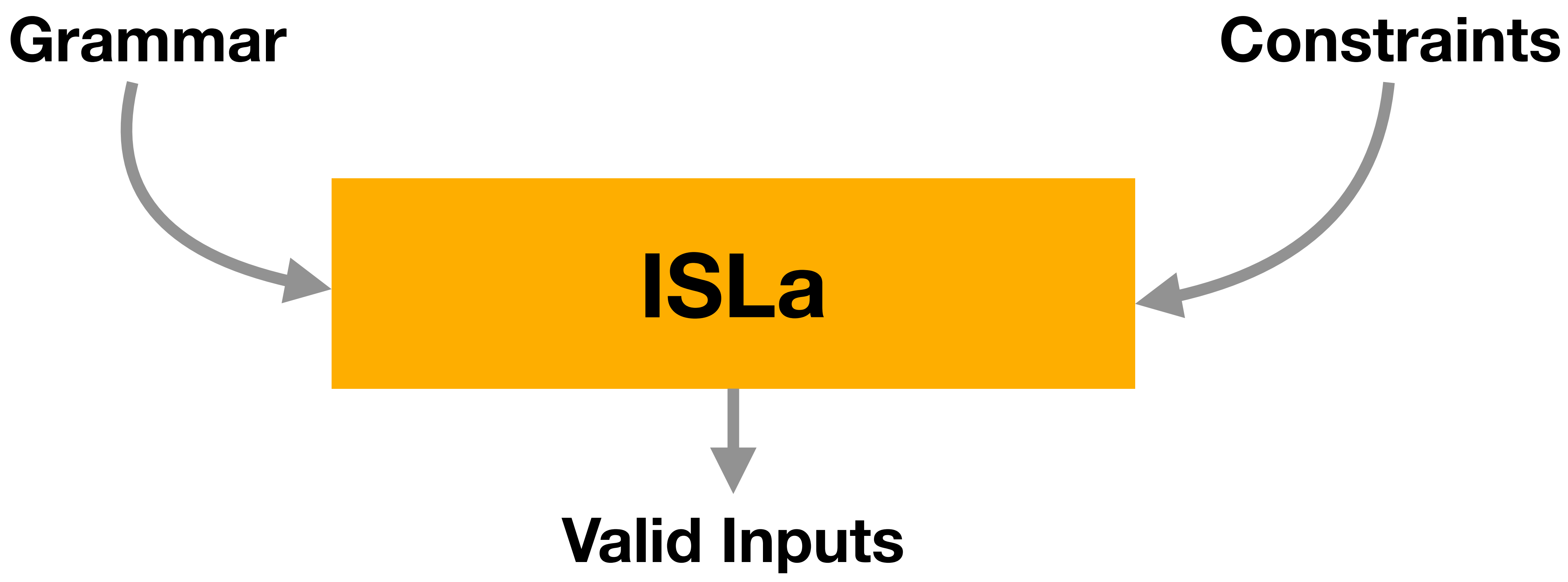
```
forall <xml-tree> tree="<{⟨id⟩ opid}[⟨xml-attribute⟩]>
                        ⟨inner-xml-tree⟩
                        </{⟨id⟩ clid}>" in start:
(= opid clid)
```

THIS IS XML

<O cfg="ej45"><Qr hh="21"></Qr>…

Satisfying Constraints

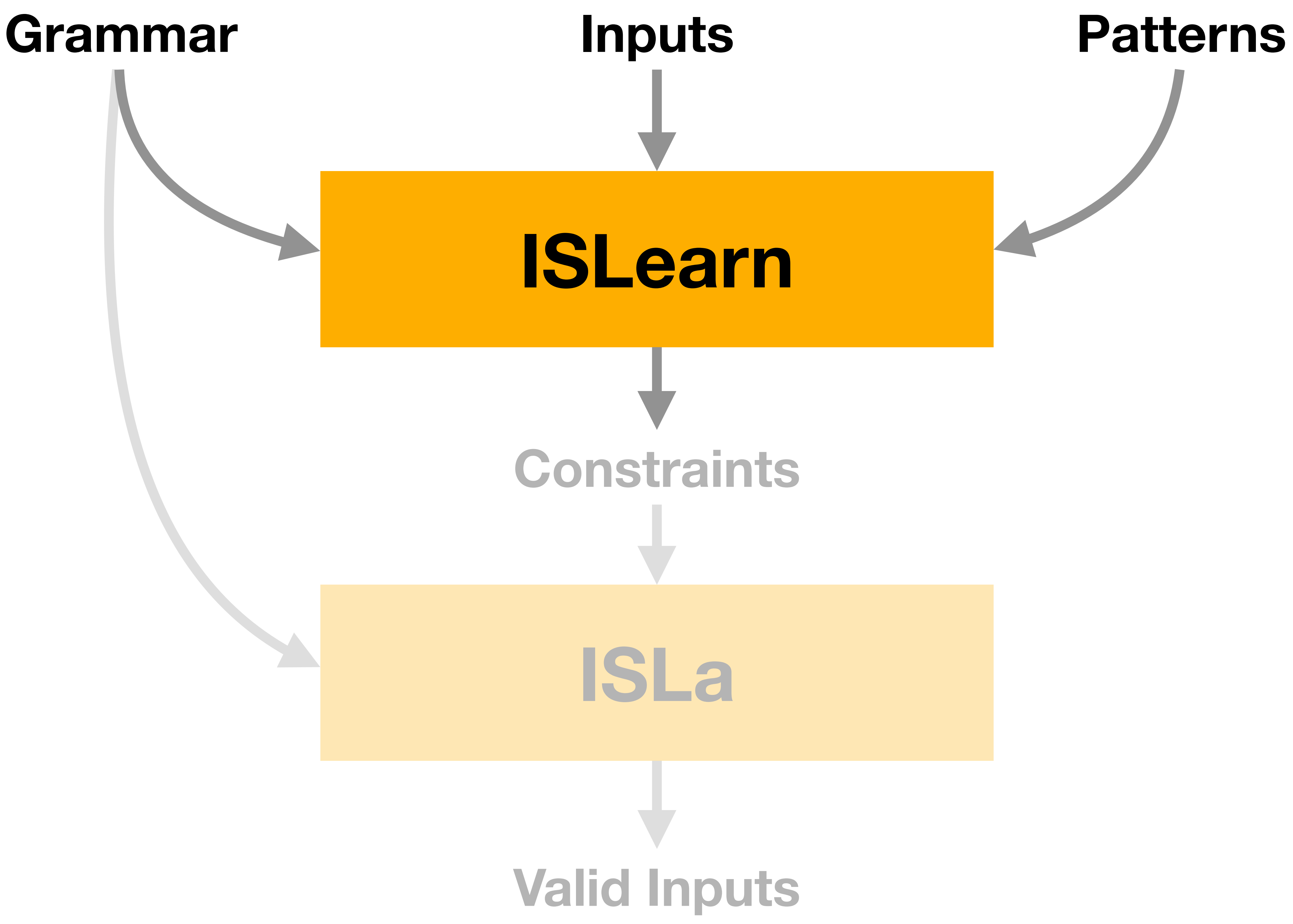
The ISLa Fuzzer + Checker



- ISLa takes a (context-free) **grammar** and (SMT) **constraints**
- **Produces** inputs that satisfy grammar + constraints
- Can **check** inputs whether they fulfill the constraints
- Full declarative specification of inputs
- Can be paired with any fuzzing strategy
- XML: 18 LOC, TAR: 61 LOC

Learning Input Languages

The ISLearn Invariant Miner



- ISLearn takes a (mined?) **grammar** and a set of **inputs**
- From a set of patterns, determines **constraints** that hold for all inputs
- **Validates** constraint candidates using extra (generated) tests
- Patterns include length constraints, checksums, def/use, and more
- Learned Racket, Dot, ICMP with 78–97% accuracy

DEMO

Taming Fuzzers

One Fuzzer

You want **full automation – but also control**:

- If you have **inputs** or **traces,** use them
  Can learn constraints from inputs
- If you have the **source code,** use it
  Can learn Grammar from code
- If you have a **spec** and/or **test cases,** use them
  Can write and/or edit language specs
- If you have an idea **where the bugs are,** use it
  Can direct generation towards bugs
- If you have an idea of **what a bug is,** use it
  Can apply checkers to outputs, Too

Learn More about ISLa

Dominic Steinhöfel

Input Invariants

Dominic Steinhöfel
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
dominic.steinhoefel@cispa.de

Andreas Zeller
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
zeller@cispa.de

ABSTRACT

Grammar-based fuzzers are highly efficient in producing syntactically valid system inputs. However, as context-free grammars cannot capture *semantic* input features, generated inputs will often be rejected as semantically invalid by a target program. We propose ISLa, a *declarative specification language for context-sensitive properties* of structured system inputs based on context-free grammars. With ISLa, it is possible to specify *input constraints* like "a variable has to be defined before it is used," "the length of the 'file name' block in a TAR file has to equal 100 bytes," or "the number of columns in all CSV rows must be identical."

ISLa constraints can be used for *parsing* or *validation* ("Does an input meet the expected constraint?") as well as for *fuzzing*, since we provide both an *evaluation* and *input generation component*. ISLa embeds SMT formulas as an island language, leveraging the power of modern solvers like Z3 to solve atomic semantic constraints. On top, it adds universal and existential *quantifiers* over the structure of derivation trees from a grammar, and structural ("X occurs before Y") and semantic ("X is the checksum of Y") *predicates*.

ISLa constraints can be specified manually, but also *mined* from existing input samples. For this, our ISLearn prototype uses a catalog of common patterns (such as the ones above), instantiates these over input elements, and retains those candidates that hold for the inputs observed and whose instantiations are fully accepted by input-processing programs. The resulting constraints can then again be used for fuzzing and parsing.

1 INTRODUCTION

Automated software testing with random inputs (*fuzzing*) [19] is an effective technique for finding bugs in programs. Pure random inputs can quickly discover errors in input processing. Yet, if a program expects complex structured inputs (e.g., C programs, JSON expressions, or binary formats), the chances of randomly producing *valid inputs* that are accepted by the parser and reach deeper functionality are low.

Language-based fuzzers [8, 12, 13] overcome this limitation by generating inputs from a *specification* of a program's expected input language, frequently expressed as a *Context-Free Grammar (CFG)*. This considerably increases the chance of producing an input passing the program's parsing stage and reaching its core logic. Yet, while being great for parsing, *CFGs are often too coarse for* producing *inputs*. Consider, e.g., the language of XML documents (without document type). This language is *not* context free.[1] Still, it can be approximated by a CFG
When we used a cover
from this grammar, exa
↪ B7</0>) contained
language-based fuzzers
for *parsing* which the
specification for *produ*
as hundreds of languag
To allow for precise
with more information

https://publications.cispa.saarland/3596/1/main.pdf

Software Engineering

test *own* programs

("Test generation")

Security

test *other* programs

("Fuzzing")

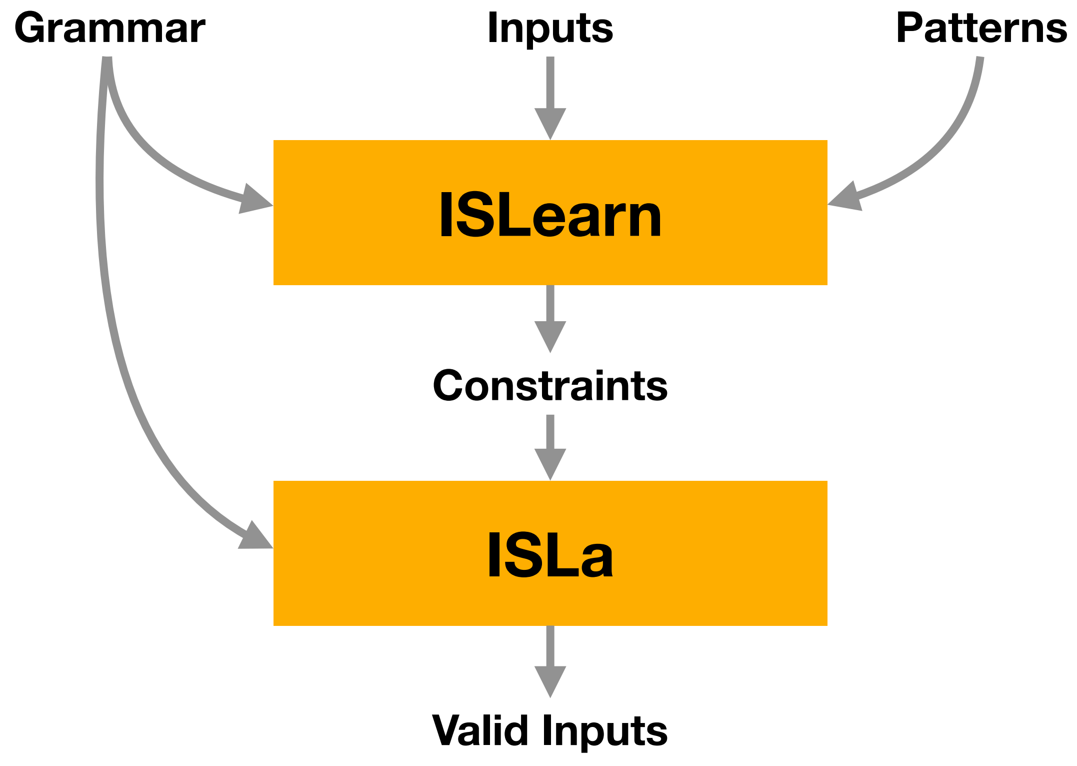
Two Cultures – Reviewing

- Focus is on **future**
- Want **guidance** for testing decisions
- Focus on **conceptual** improvements
- Want explanations on how **numbers** come to be
- Want details on **decisions** and **rationales**
- Want to know when and why things will **not** work
- Expect **open science** principles

- Focus is on **here and now**
- Expect **vulnerabilities** or even **exploits**
- Want **hard-to-test systems** as benchmarks
- Want no assumptions; expect **full automation**
- Expect **scalability** and **versatility**
- Expect you are **better** than others
- Expect tools + data only **after acceptance**

Input Invariants

ISLa input specification language + ISLearn invariant miner



- **ISLa** takes a (context-free) **grammar** and (SMT) **constraints**
- ISLa **produces** and **checks** inputs that satisfy grammar + constraints
- **ISLearn** learns constraints from given inputs
- Full **declarative** specification
- **Validated** through generated tests

Taming Fuzzers

One Fuzzer

You want **full automation – but also control**:

- If you have **inputs** or **traces,** use them
  CAN LEARN CONSTRAINTS FROM INPUTS
- If you have the **source code,** use it
  CAN LEARN GRAMMAR FROM CODE
- If you have a **spec** and/or **test cases,** use them
  CAN WRITE AND/OR EDIT LANGUAGE SPECS
- If you have an idea **where the bugs are,** use it
  CAN DIRECT GENERATION TOWARDS BUGS
- If you have an idea of **what a bug is,** use it
  CAN APPLY CHECKERS TO OUTPUTS, TOO

Software Engineering

test *own* programs

("Test generation")

Security

Two Cultures – Reviewing

- Focus is on **future**
- Want **guidance** for testing decisions
- … to be
- …t work

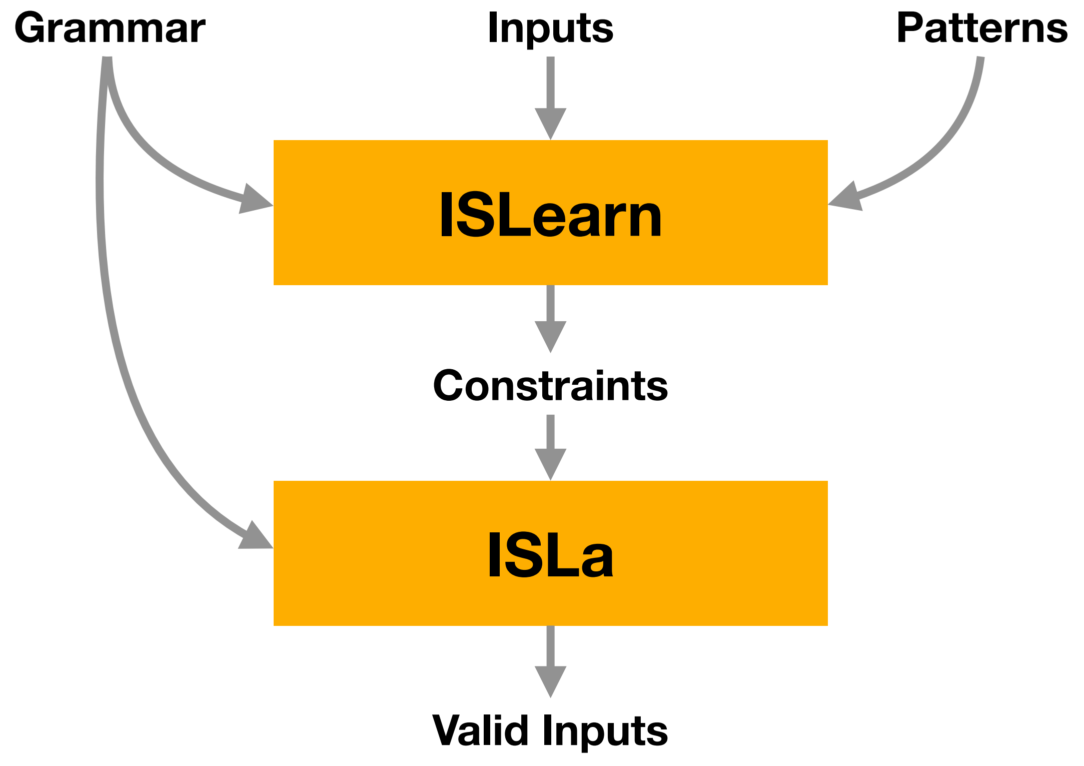
- Focus is on **here and now**
- Expect **vulnerabilities** or even **exploits**
- Want **hard-to-test systems** as benchmarks
- Want no assumptions; expect **full automation**
- Expect **scalability** and **versatility**
- Expect you are **better** than others
- Expect tools + data only **after acceptance**

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

We're hiring!

Input Invariants

ISLa input specification language + IS…



- **ISL…** an…
- ISL… sat…
- **ISLearn** learns constraints from given inputs
- Full **declarative** specification
- **Validated** through generated tests

One Fuzzer

…nt **full automation – but also control**:

- …u have **inputs** or **traces,** use them
  CAN LEARN CONSTRAINTS FROM INPUTS
- If you have the **source code,** use it
  CAN LEARN GRAMMAR FROM CODE
- If you have a **spec** and/or **test cases,** use them
  CAN WRITE AND/OR EDIT LANGUAGE SPECS
- If you have an idea **where the bugs are,** use it
  CAN DIRECT GENERATION TOWARDS BUGS
- If you have an idea of **what a bug is,** use it
  CAN APPLY CHECKERS TO OUTPUTS, TOO