Lecture 12 – Real-World Tools and Practice

SWS128: Software Analysis

Jihyeok Park



2026 Fall

Recall

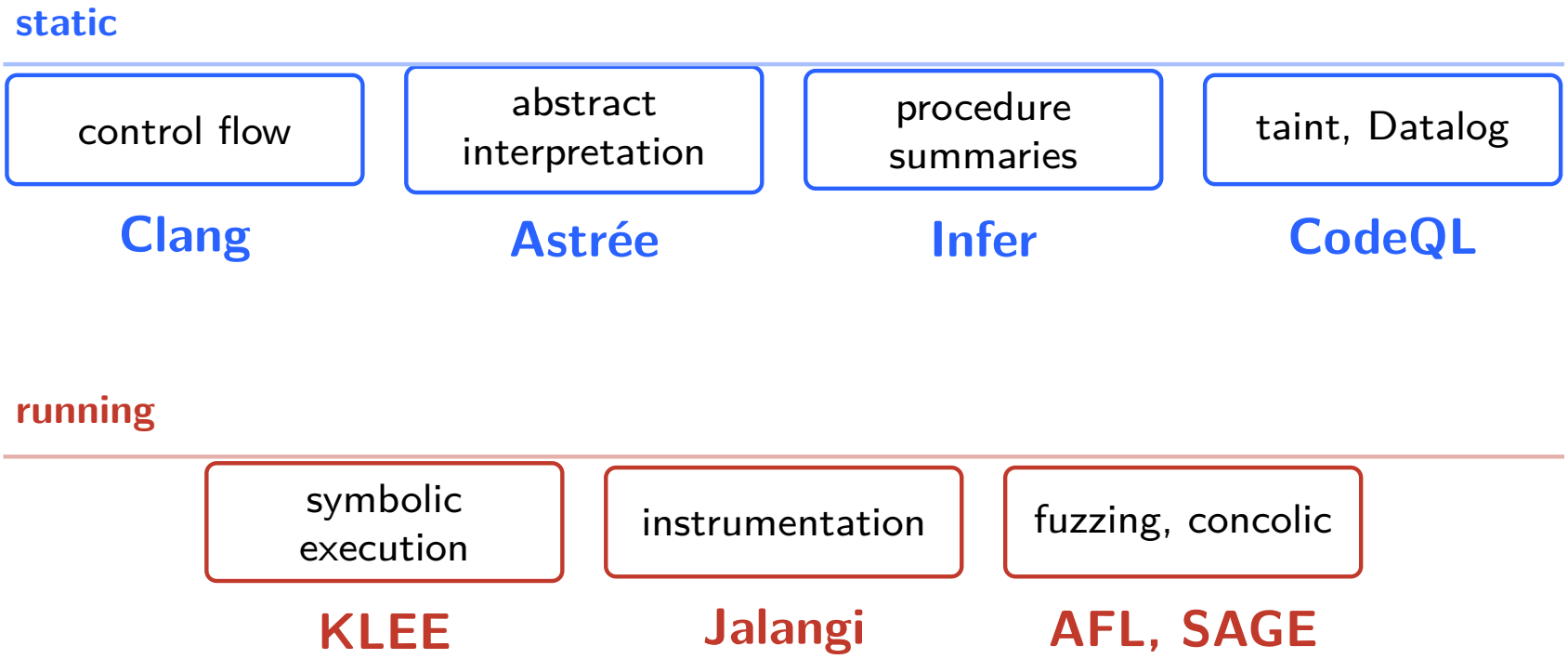
static

- control flow
- abstract interpretation
- procedure summaries
- taint, Datalog

running

- symbolic execution
- instrumentation
- fuzzing, concolic

Recall



- For each: how a real tool does what we did,

Recall

static

control flow	abstract interpretation	procedure summaries	taint, Datalog
Clang	**Astrée**	**Infer**	**CodeQL**
goto fail	Airbus A340	100,000 fixes	Apache Struts

running

symbolic execution	instrumentation	fuzzing, concolic
KLEE	**Jalangi**	**AFL, SAGE**
coreutils	IKEA, eBay, . . .	bash, Windows 7

- For each: how a real tool does what we did,
- and a **real bug** it found.

Contents

1. Static Analyzers
   - Clang: Unreachable Code
   - Astrée: Abstract Interpretation
   - Infer: Procedure Summaries
   - CodeQL: Taint as Datalog

2. Running Programs
   - KLEE: Symbolic Execution
   - Jalangi: Instrumentation
   - AFL and SAGE: Fuzzing

3. Beyond This Course

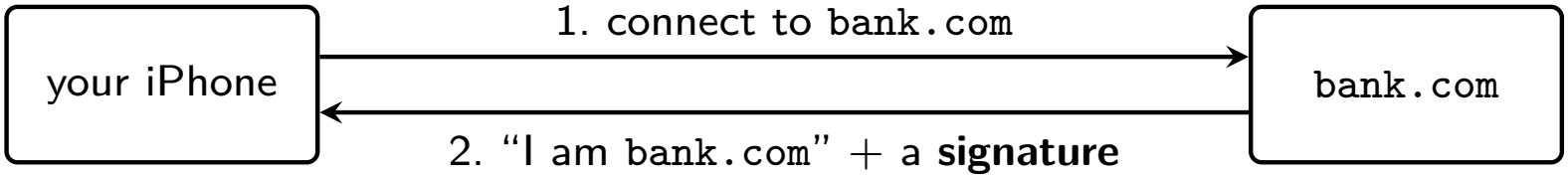
Contents

1. Static Analyzers
   - Clang: Unreachable Code
   - Astrée: Abstract Interpretation
   - Infer: Procedure Summaries
   - CodeQL: Taint as Datalog

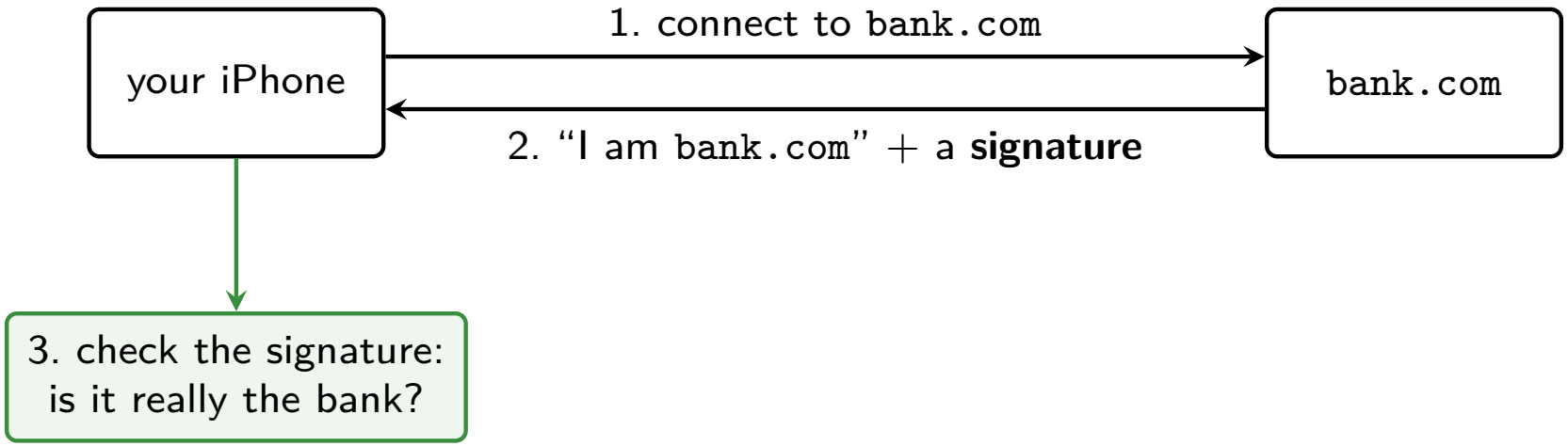
2. Running Programs
   - KLEE: Symbolic Execution
   - Jalangi: Instrumentation
   - AFL and SAGE: Fuzzing

3. Beyond This Course

Is This Really My Bank?

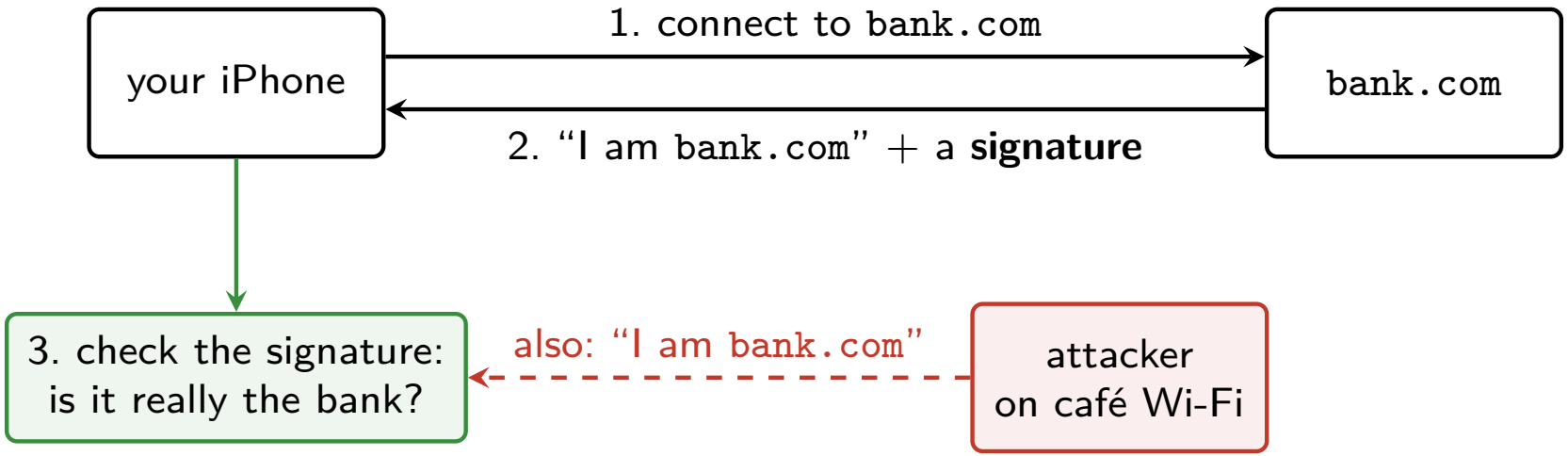


Is This Really My Bank?



- Step 3 is what makes the lock icon mean anything.

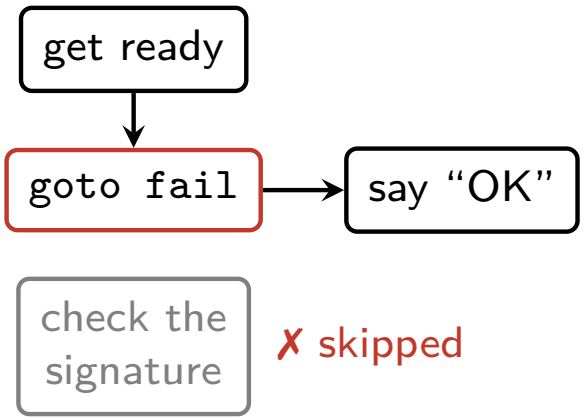
Is This Really My Bank?



- Step 3 is what makes the lock icon mean anything.
- Without it, anyone between you and the bank can say "I am the bank", and read everything you send.

goto fail

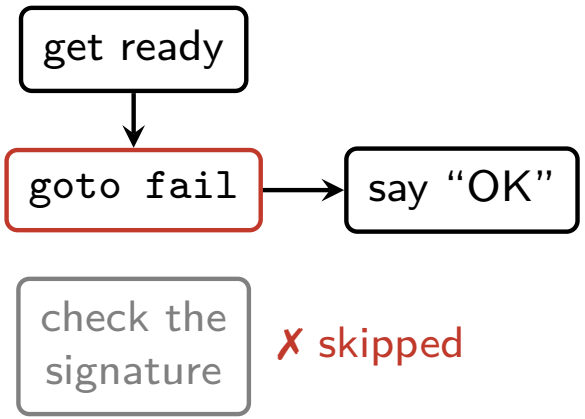
```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
...
fail:
```



Apple Security-55471, `sslKeyExchange.c` (shortened); NVD, CVE-2014-1266.

goto fail

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
...
fail:
```

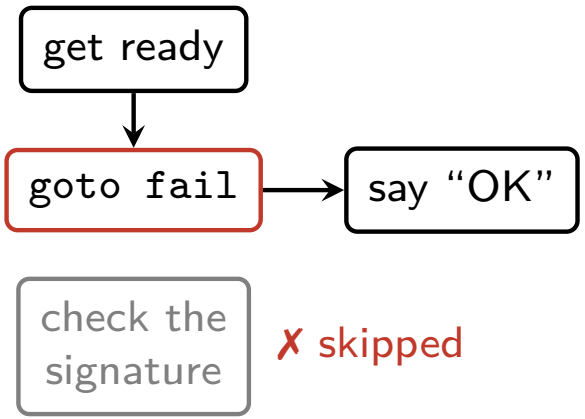


- The red line is an extra copy. It is **not** under the `if` above it, so it **always** jumps to the end.

Apple Security-55471, `sslKeyExchange.c` (shortened); NVD, CVE-2014-1266.

goto fail

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
...
fail:
```

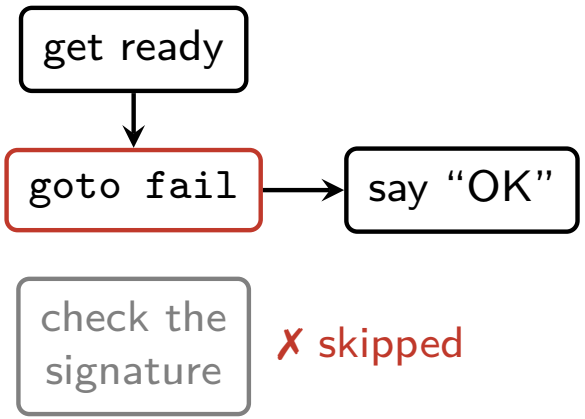


- The red line is an extra copy. It is **not** under the `if` above it, so it **always** jumps to the end.
- The real check never runs, and the function still says "OK".

Apple Security-55471, `sslKeyExchange.c` (shortened); NVD, CVE-2014-1266.

goto fail

```
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;
...
fail:
```



- The red line is an extra copy. It is **not** under the `if` above it, so it **always** jumps to the end.
- The real check never runs, and the function still says "OK".
- On the attacker's Wi-Fi, iPhones and Macs trusted a fake bank.

Apple Security-55471, `sslKeyExchange.c` (shortened); NVD, CVE-2014-1266.

The Warning Was There

`-Wall`: **no warning**

`-Wunreachable-code`: **warning**

- Clang finds unreachable code on the **CFG**, like our `cfg.py`.

Adam Langley, *Apple's SSL/TLS bug*, ImperialViolet, 2014.

The Warning Was There

`-Wall`: **no warning**

`-Wunreachable-code`: **warning**

- Clang finds unreachable code on the **CFG**, like our `cfg.py`.
- But the check was **off** by default, not even in `-Wall`.

Adam Langley, *Apple's SSL/TLS bug*, ImperialViolet, 2014.

The Warning Was There

`-Wall`: **no warning**

`-Wunreachable-code`: **warning**

- Clang finds unreachable code on the **CFG**, like our `cfg.py`.
- But the check was **off** by default, not even in `-Wall`.

- Why off? Probably too many **false positives** on real code bases.

Adam Langley, *Apple's SSL/TLS bug*, ImperialViolet, 2014.

The Warning Was There

`-Wall`: **no warning**

`-Wunreachable-code`: **warning**

- Clang finds unreachable code on the **CFG**, like our `cfg.py`.
- But the check was **off** by default, not even in `-Wall`.

- Why off? Probably too many **false positives** on real code bases.
- The analysis was easy. Getting people to turn it on was not.

Adam Langley, *Apple's SSL/TLS bug*, ImperialViolet, 2014.

Astrée Is Our Interval Analysis, Grown Up

we built	**Astrée**
intervals	intervals, for integers and floats
widening with thresholds	"goes through a number of thresholds"
narrowing	narrowing iterations
octagons	octagons, on small packs of variables
path-insensitive join	**trace partitioning**: some `ifs` kept apart

Cousot et al., ESOP 2005; Blanchet et al., PLDI 2003.

Astrée Is Our Interval Analysis, Grown Up

we built	**Astrée**
intervals	intervals, for integers and floats
widening with thresholds	"goes through a number of thresholds"
narrowing	narrowing iterations
octagons	octagons, on small packs of variables
path-insensitive join	**trace partitioning**: some `ifs` kept apart

- Goal: prove **no run-time error** — no out-of-bounds index, no division by zero, no overflow.

Cousot et al., ESOP 2005; Blanchet et al., PLDI 2003.

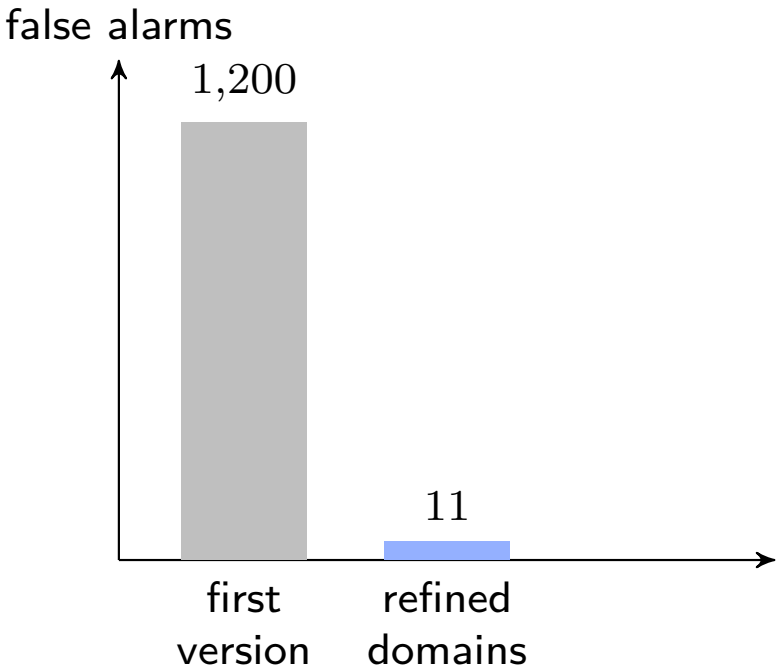
Proving Flight Control Software



- Airbus A340 primary flight control: **132,000** lines of C.

1,200 and 11: Blanchet et al., PLDI 2003. The A340 run: astree.ens.fr.

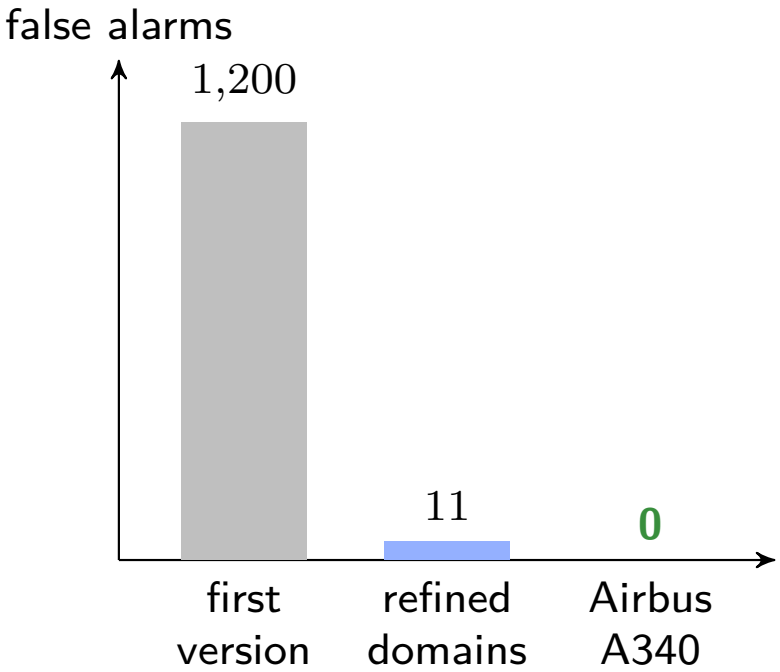
Proving Flight Control Software



- Airbus A340 primary flight control: **132,000** lines of C.
- Each false alarm removed by a sharper **domain**, not by hiding the warning.

1,200 and 11: Blanchet et al., PLDI 2003. The A340 run: astree.ens.fr.

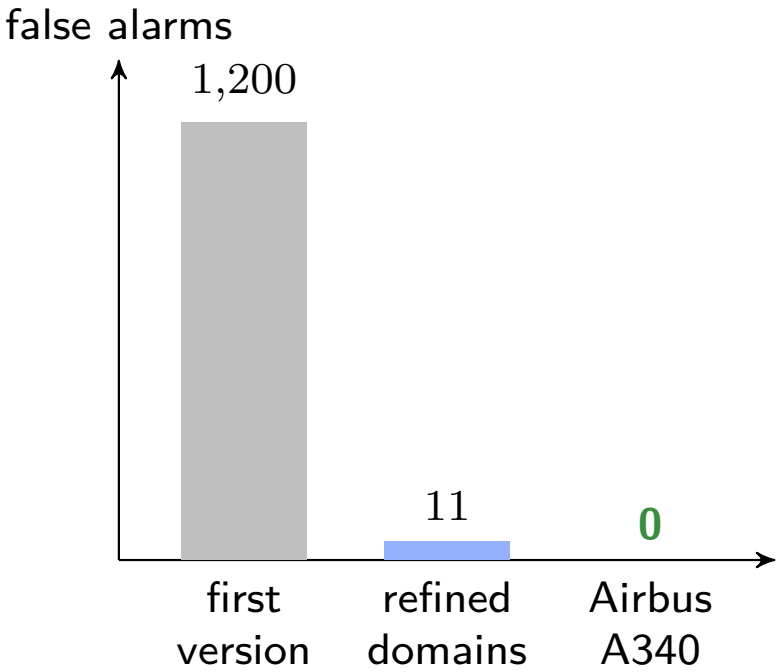
Proving Flight Control Software



- Airbus A340 primary flight control: **132,000** lines of C.
- Each false alarm removed by a sharper **domain**, not by hiding the warning.
- 2003: no run-time error, proved automatically, in 1 h 20.

1,200 and 11: Blanchet et al., PLDI 2003. The A340 run: astree.ens.fr.

Proving Flight Control Software

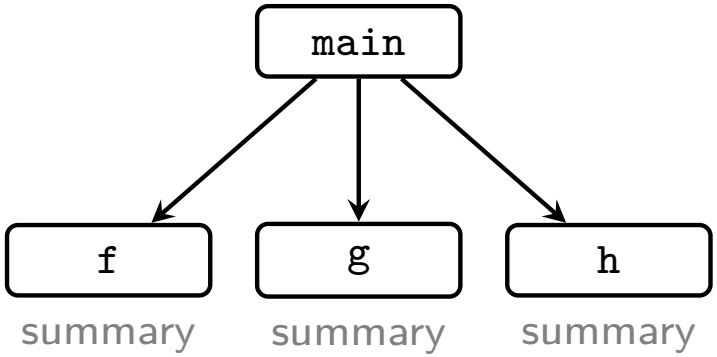


- Airbus A340 primary flight control: **132,000** lines of C.
- Each false alarm removed by a sharper **domain**, not by hiding the warning.
- 2003: no run-time error, proved automatically, in 1 h 20.

- Zero alarms: **sound and precise**. That is what you need where a miss is unacceptable.

1,200 and 11: Blanchet et al., PLDI 2003. The A340 run: astree.ens.fr.

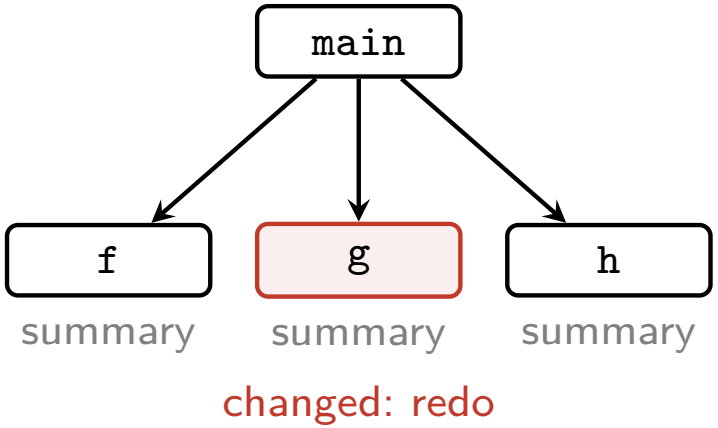
Infer: One Summary per Procedure



- Each procedure is analyzed **once**, into a summary. Callers use the summary.

Distefano et al., CACM 2019. Example: fbinfer.com, Hello, World.

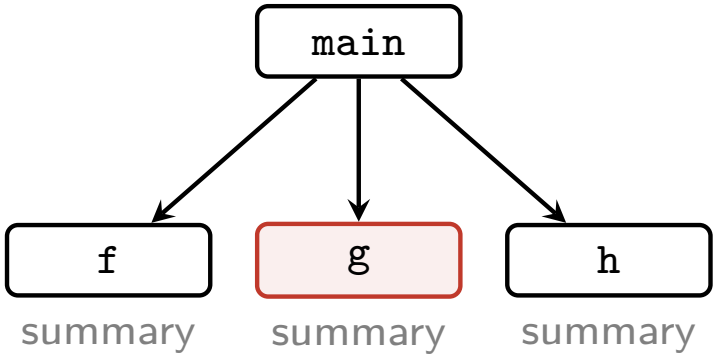
Infer: One Summary per Procedure



- Each procedure is analyzed **once**, into a summary. Callers use the summary.
- Change g: only g is redone.

Distefano et al., CACM 2019. Example: fbinfer.com, Hello, World.

Infer: One Summary per Procedure



- Each procedure is analyzed **once**, into a summary. Callers use the summary.
- Change g: only g is redone.

```
void test() {
  int *s = NULL;
  *s = 42;
}
```

```
$ infer run -- gcc -c hello.c
hello.c:5: error: NULL_DEREFERENCE
  pointer s last assigned on line 4 could be null
  and is dereferenced at line 5, column 10
```

Distefano et al., CACM 2019. Example: fbinfer.com, Hello, World.

Same Bugs, Different Moment

Infer at Facebook showed its reports to developers in two ways:

a list, every night

issue 1: null dereference in . . .
issue 2: resource leak in . . .
issue 3: null dereference in . . .
. . .

0% fixed

Distefano et al., *Scaling Static Analyses at Facebook*, CACM 2019.

Same Bugs, Different Moment

Infer at Facebook showed its reports to developers in two ways:

a list, every night

issue 1: null dereference in ...
issue 2: resource leak in ...
issue 3: null dereference in ...
...

0% fixed

a comment on your change

```
  int *s = NULL;
+ *s = 42;
```

Infer: pointer `s` could be null and is dereferenced at line 5

70% fixed

- Same analysis, same kind of bugs. Only **when and where** the developer sees them changed.

Distefano et al., *Scaling Static Analyses at Facebook*, CACM 2019.

Same Bugs, Different Moment

Infer at Facebook showed its reports to developers in two ways:

a list, every night

issue 1: null dereference in ...
issue 2: resource leak in ...
issue 3: null dereference in ...
...

0% fixed

a comment on your change

```
  int *s = NULL;
+ *s = 42;
```

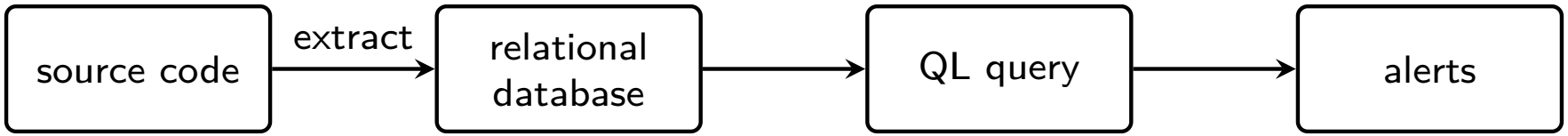
Infer: pointer `s` could be null and is dereferenced at line 5

70% fixed

- Same analysis, same kind of bugs. Only **when and where** the developer sees them changed.
- Since 2014: **over 100,000** Infer reports fixed.

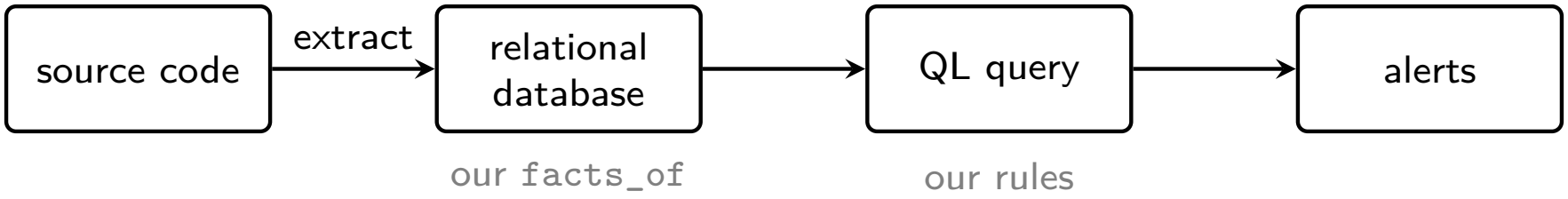
Distefano et al., *Scaling Static Analyses at Facebook*, CACM 2019.

CodeQL: The Program Is a Database



codeql.github.com, *About the QL language*; docs.github.com.

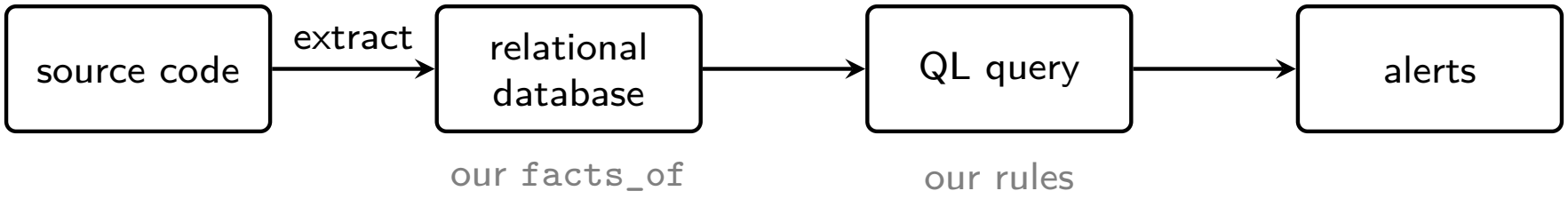
CodeQL: The Program Is a Database



- Exactly our Datalog lecture: facts from the program, rules on top.

codeql.github.com, *About the QL language*; docs.github.com.

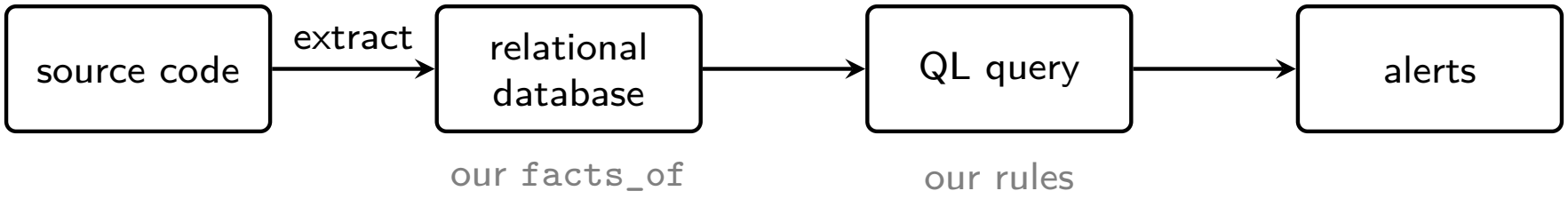
CodeQL: The Program Is a Database



- Exactly our Datalog lecture: facts from the program, rules on top.
- "The semantics of QL are based on **Datalog** . . . QL inherits recursive predicates from Datalog."

codeql.github.com, *About the QL language*; docs.github.com.

CodeQL: The Program Is a Database



- Exactly our Datalog lecture: facts from the program, rules on top.
- "The semantics of QL are based on **Datalog** . . . QL inherits recursive predicates from Datalog."
- The engine behind GitHub code scanning.

codeql.github.com, *About the QL language*; docs.github.com.

Remote Code Execution in Apache Struts

PLRG

```
class ContentTypeHandlerInput extends FlowSource {
  ContentTypeHandlerInput() {
    exists(ToObjectDeserializer des |
      des.getParameter(0).getAnAccess() = this
    )
  }
}
from ContentTypeHandlerInput source, UnsafeDeserializationSink sink
where source.flowsTo(sink)
select source, sink
```

source	the XML a client sends
sink	deserialization
result	**exactly one**

Man Yue Mo, GitHub Security Lab, 2017 (query shortened); Apache Struts S2-052.

Remote Code Execution in Apache Struts

PLRG

```
class ContentTypeHandlerInput extends FlowSource {
  ContentTypeHandlerInput() {
    exists(ToObjectDeserializer des |
      des.getParameter(0).getAnAccess() = this
    )
  }
}
from ContentTypeHandlerInput source, UnsafeDeserializationSink sink
where source.flowsTo(sink)
select source, sink
```

source	the XML a client sends
sink	deserialization
result	**exactly one**

- A new source, an existing sink, a few lines of QL.

Man Yue Mo, GitHub Security Lab, 2017 (query shortened); Apache Struts S2-052.

Remote Code Execution in Apache Struts

PLRG

```
class ContentTypeHandlerInput extends FlowSource {
  ContentTypeHandlerInput() {
    exists(ToObjectDeserializer des |
      des.getParameter(0).getAnAccess() = this
    )
  }
}
from ContentTypeHandlerInput source, UnsafeDeserializationSink sink
where source.flowsTo(sink)
select source, sink
```

source	the XML a client sends
sink	deserialization
result	**exactly one**

- A new source, an existing sink, a few lines of QL.
- The one result: anyone could run code on the server (CVE-2017-9805).

Man Yue Mo, GitHub Security Lab, 2017 (query shortened); Apache Struts S2-052.

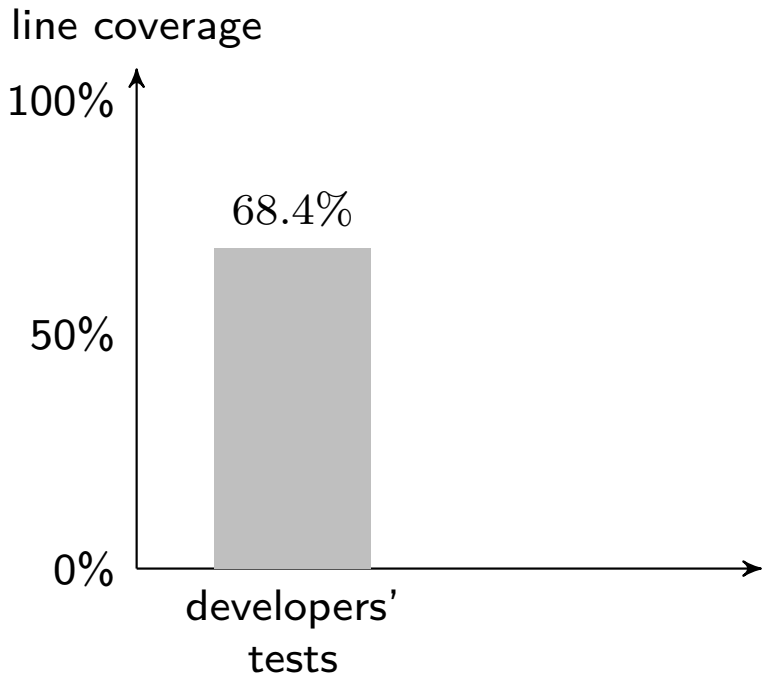
Contents

1. Static Analyzers
   - Clang: Unreachable Code
   - Astrée: Abstract Interpretation
   - Infer: Procedure Summaries
   - CodeQL: Taint as Datalog

2. Running Programs
   - KLEE: Symbolic Execution
   - Jalangi: Instrumentation
   - AFL and SAGE: Fuzzing

3. Beyond This Course

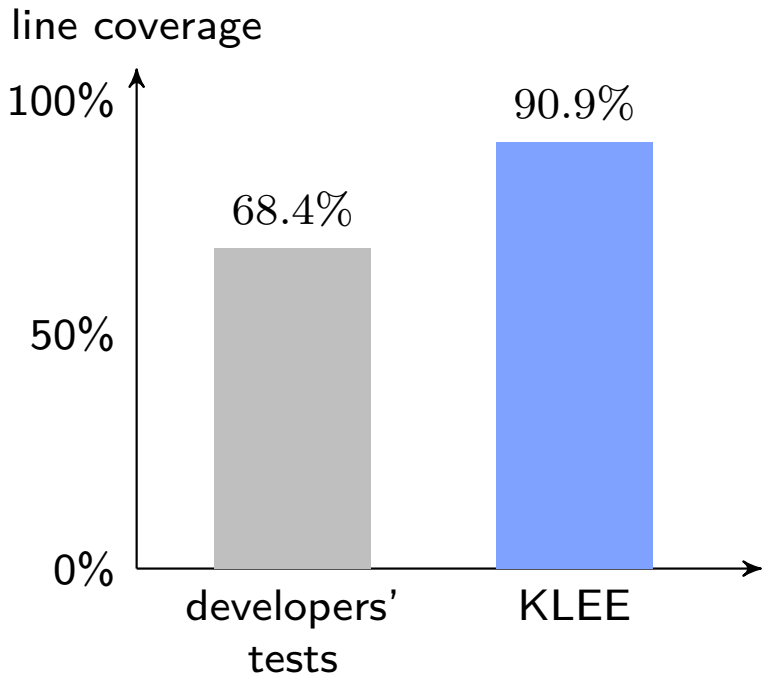
KLEE on GNU Coreutils



- 89 programs: `ls`, `cat`, `pr`, ...
- Arguments and files are **symbolic**. Library calls go to hand-written **models**.

Cadar, Dunbar, Engler, *KLEE*, OSDI 2008.

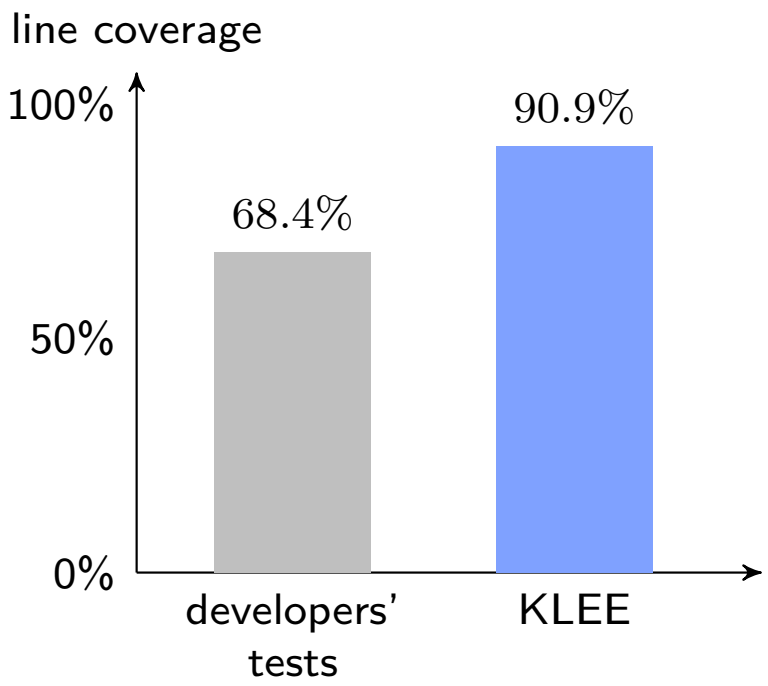
KLEE on GNU Coreutils



- 89 programs: `ls`, `cat`, `pr`, ...
- Arguments and files are **symbolic**. Library calls go to hand-written **models**.
- Per tool, on average: 90.9% against 68.4%.

Cadar, Dunbar, Engler, *KLEE*, OSDI 2008.

KLEE on GNU Coreutils



- 89 programs: `ls`, `cat`, `pr`, ...
- Arguments and files are **symbolic**. Library calls go to hand-written **models**.
- Per tool, on average: 90.9% against 68.4%.

- 89 hours of KLEE beat a test suite written over fifteen years.

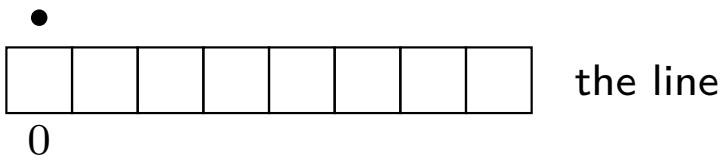
Cadar, Dunbar, Engler, *KLEE*, OSDI 2008.

A Bug from 1992

`pr` lays text out in columns for printing. KLEE made this input:

```
$ pr -e t2.txt              # t2.txt: "\b\b\b\b\b\b\b\t"
```

Seven backspaces, then one tab. `-e` turns each tab into spaces.



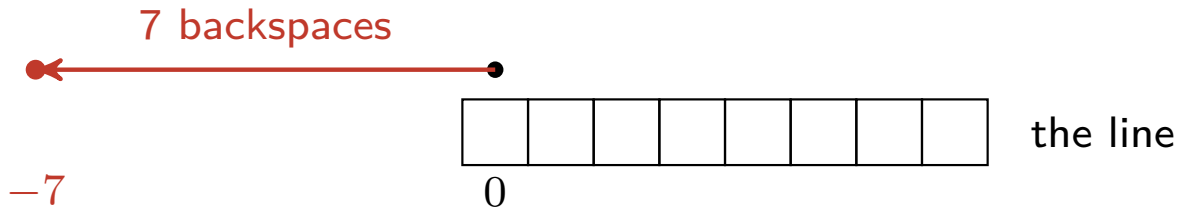
Cadar, Dunbar, Engler, *KLEE*, OSDI 2008, Figure 7.

A Bug from 1992

`pr` lays text out in columns for printing. KLEE made this input:

```
$ pr -e t2.txt              # t2.txt: "\b\b\b\b\b\b\b\t"
```

Seven backspaces, then one tab. `-e` turns each tab into spaces.



- Each backspace moves the cursor one column left. From the start of the line, seven of them reach -7.

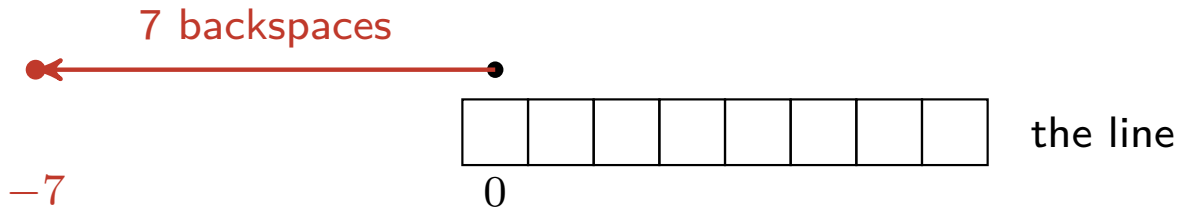
Cadar, Dunbar, Engler, *KLEE*, OSDI 2008, Figure 7.

A Bug from 1992

`pr` lays text out in columns for printing. KLEE made this input:

```
$ pr -e t2.txt              # t2.txt: "\b\b\b\b\b\b\b\t"
```

Seven backspaces, then one tab. `-e` turns each tab into spaces.



- Each backspace moves the cursor one column left. From the start of the line, seven of them reach -7.
- Nobody expected a column below zero, not since 1992. KLEE tried it, because the input was symbolic.

Cadar, Dunbar, Engler, *KLEE*, OSDI 2008, Figure 7.

Fifteen Spaces in Eight Bytes

Tab stops are every 8 columns:
a tab is $8 - (\text{column} \bmod 8)$ spaces.

column	spaces
3	$8 - 3 = 5$

- Never more than 8, so `pr` set aside **8 bytes**.

Cadar, Dunbar, Engler, *KLEE*, OSDI 2008, Figure 8 (code simplified).

Fifteen Spaces in Eight Bytes

Tab stops are every 8 columns:
a tab is $8 - (\text{column} \bmod 8)$ spaces.

column	spaces
3	$8 - 3 = 5$
-7	$8 - (-7) = 15$

- Never more than 8, so `pr` set aside **8 bytes**.
- In C, `-7 % 8` is −7, not 1. So the tab becomes **15** spaces.

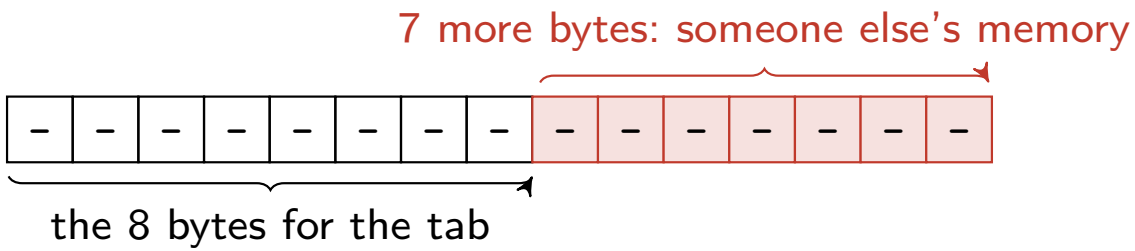
Cadar, Dunbar, Engler, *KLEE*, OSDI 2008, Figure 8 (code simplified).

Fifteen Spaces in Eight Bytes

Tab stops are every 8 columns:
a tab is $8 - (\text{column} \bmod 8)$ spaces.

column	spaces
3	$8 - 3 = 5$
-7	$8 - (-7) = 15$

- Never more than 8, so `pr` set aside **8 bytes**.
- In C, `-7 % 8` is `-7`, not `1`. So the tab becomes **15** spaces.



- 15 spaces into 8 bytes: `pr` overwrites other memory and **crashes**.

Cadar, Dunbar, Engler, *KLEE*, OSDI 2008, Figure 8 (code simplified).

Jalangi: Hooks for JavaScript

```
J$.analysis = {
  conditional : function (iid, result) {
    var id = J$.getGlobalIID(iid);
    var branchInfo = branches[id];
    if (!branchInfo) {
      branchInfo = branches[id] = {trueCount: 0, falseCount: 0};
    }
    if (result) { branchInfo.trueCount++; } else { branchInfo.falseCount++; }
  }
};
```

- Jalangi rewrites the JavaScript so every operation calls a **hook**: `binary`, `read`, `write`, `conditional`, ...

Sen et al., ESEC/FSE 2013. Code: Jalangi 2 tutorial, shortened.

Jalangi: Hooks for JavaScript

PLRG

```
J$.analysis = {
  conditional : function (iid, result) {
    var id = J$.getGlobalIID(iid);
    var branchInfo = branches[id];
    if (!branchInfo) {
      branchInfo = branches[id] = {trueCount: 0, falseCount: 0};
    }
    if (result) { branchInfo.trueCount++; } else { branchInfo.falseCount++; }
  }
};
```

- Jalangi rewrites the JavaScript so every operation calls a **hook**: `binary`, `read`, `write`, `conditional`, ...
- This one hook counts true and false per branch: **branch coverage**.

Sen et al., ESEC/FSE 2013. Code: Jalangi 2 tutorial, shortened.

Jalangi: Hooks for JavaScript

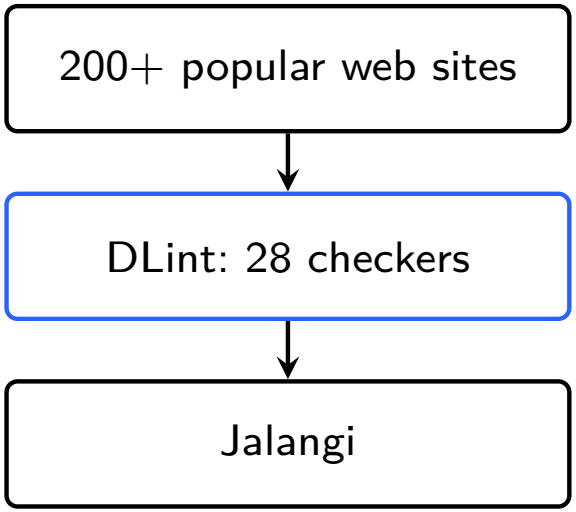
PLRG

```
J$.analysis = {
  conditional : function (iid, result) {
    var id = J$.getGlobalIID(iid);
    var branchInfo = branches[id];
    if (!branchInfo) {
      branchInfo = branches[id] = {trueCount: 0, falseCount: 0};
    }
    if (result) { branchInfo.trueCount++; } else { branchInfo.falseCount++; }
  }
};
```

- Jalangi rewrites the JavaScript so every operation calls a **hook**: `binary`, `read`, `write`, `conditional`, ...
- This one hook counts true and false per branch: **branch coverage**.
- The source column of our probe picture: one framework, many analyses.

Sen et al., ESEC/FSE 2013. Code: Jalangi 2 tutorial, shortened.

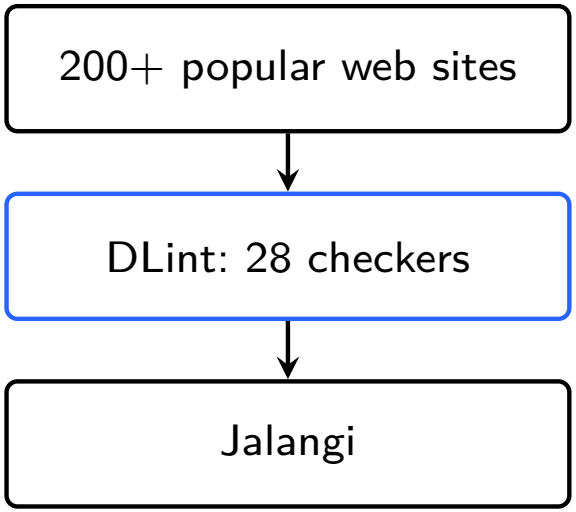
DLint: Bugs on Real Web Sites



- Each checker: a few hooks that watch the program **run**.

Gong, Pradel, Sridharan, Sen, *DLint*, ISSTA 2015.

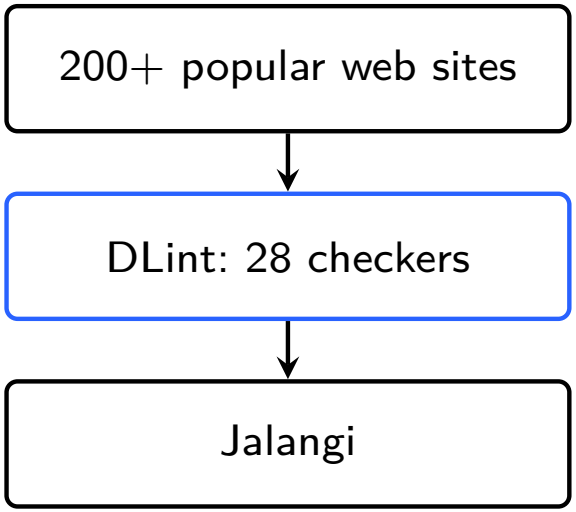
DLint: Bugs on Real Web Sites



- Each checker: a few hooks that watch the program **run**.
- On average, **49 problems per site** that static checkers missed.

Gong, Pradel, Sridharan, Sen, *DLint*, ISSTA 2015.

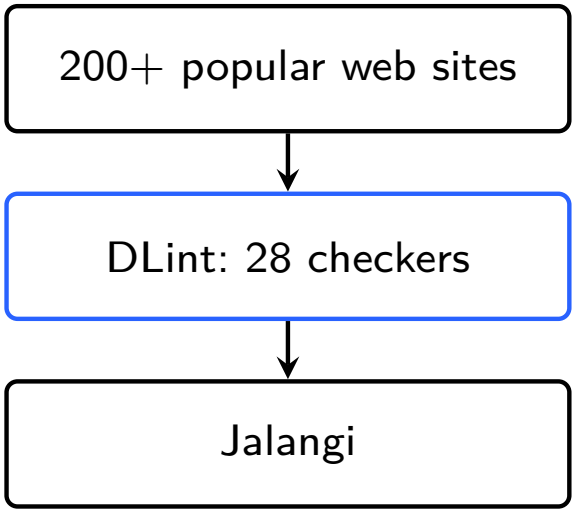
DLint: Bugs on Real Web Sites



- Each checker: a few hooks that watch the program **run**.
- On average, **49 problems per site** that static checkers missed.
- Visible bugs on the sites of IKEA, Hilton, eBay, and CNBC.

Gong, Pradel, Sridharan, Sen, *DLint*, ISSTA 2015.

DLint: Bugs on Real Web Sites



- Each checker: a few hooks that watch the program **run**.
- On average, **49 problems per site** that static checkers missed.
- Visible bugs on the sites of IKEA, Hilton, eBay, and CNBC.

- Static and dynamic checking found different problems.

Gong, Pradel, Sridharan, Sen, *DLint*, ISSTA 2015.

AFL: Coverage in Three Lines

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

- A compiler pass puts this at **every branch**: a counter per edge, in a 64 kB table.

AFL *technical_details.txt* and trophy case; Zalewski's blog, 2014.

AFL: Coverage in Three Lines

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

- A compiler pass puts this at **every branch**: a counter per edge, in a 64 kB table.
- A mutated input that hits a **new** edge is kept. Others are dropped. Our fuzzing loop, exactly.

AFL *technical_details.txt* and trophy case; Zalewski's blog, 2014.

AFL: Coverage in Three Lines

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]++;
prev_location = cur_location >> 1;
```

- A compiler pass puts this at **every branch**: a counter per edge, in a 64 kB table.
- A mutated input that hits a **new** edge is kept. Others are dropped. Our fuzzing loop, exactly.

some bugs AFL found
bash after Shellshock: CVE-2014-6277, CVE-2014-6278
OpenSSL, OpenSSH, sqlite, libpng, Firefox, Safari, . . .

AFL *technical_details.txt* and trophy case; Zalewski's blog, 2014.

SAGE: Flip Every Branch

```
void top(char input[4]) {
  int cnt = 0;
  if (input[0] == 'b') cnt++;
  if (input[1] == 'a') cnt++;
  if (input[2] == 'd') cnt++;
  if (input[3] == '!') cnt++;
  if (cnt >= 4) abort();
}
```

good

all `if`s false

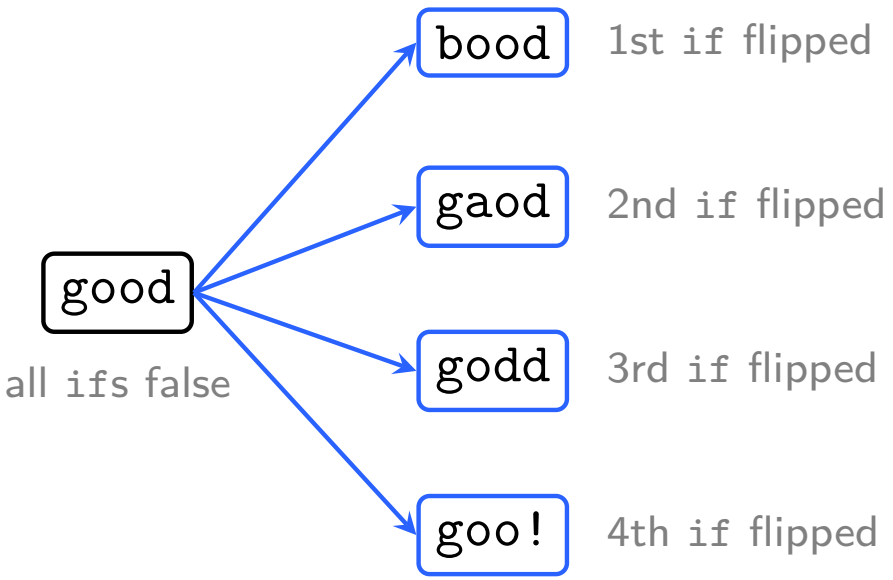
The bug needs exactly `"bad!"`.

Godefroid, Levin, Molnar, *SAGE: Whitebox Fuzzing*, CACM 2012, Figure 2.

SAGE: Flip Every Branch

```
void top(char input[4]) {
  int cnt = 0;
  if (input[0] == 'b') cnt++;
  if (input[1] == 'a') cnt++;
  if (input[2] == 'd') cnt++;
  if (input[3] == '!') cnt++;
  if (cnt >= 4) abort();
}
```

The bug needs exactly "bad!".



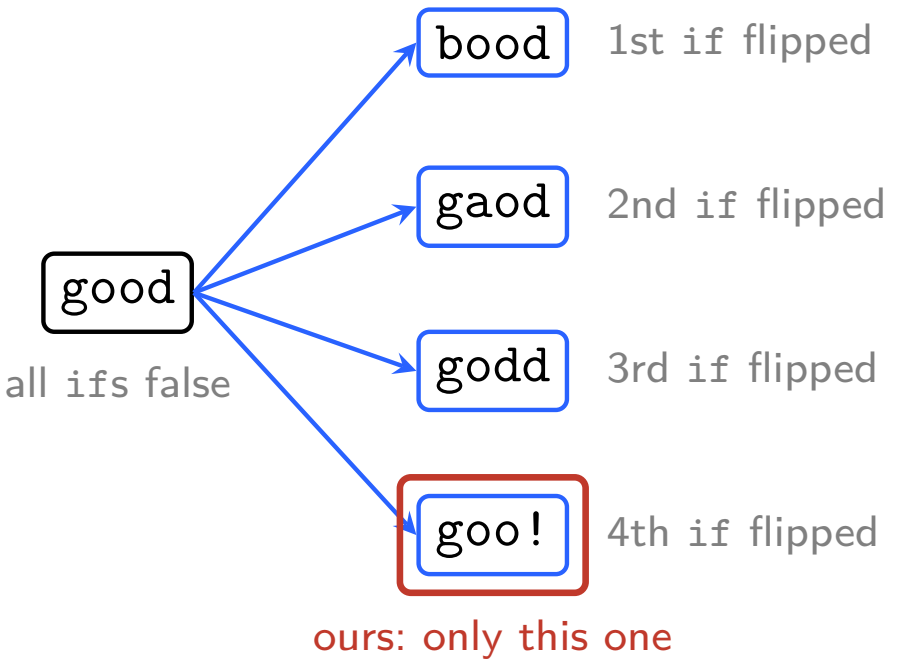
- One run of "good" gives four new inputs: flip each `if`, one at a time.

Godefroid, Levin, Molnar, *SAGE: Whitebox Fuzzing*, CACM 2012, Figure 2.

SAGE: Flip Every Branch

```
void top(char input[4]) {
  int cnt = 0;
  if (input[0] == 'b') cnt++;
  if (input[1] == 'a') cnt++;
  if (input[2] == 'd') cnt++;
  if (input[3] == '!') cnt++;
  if (cnt >= 4) abort();
}
```

The bug needs exactly "bad!".



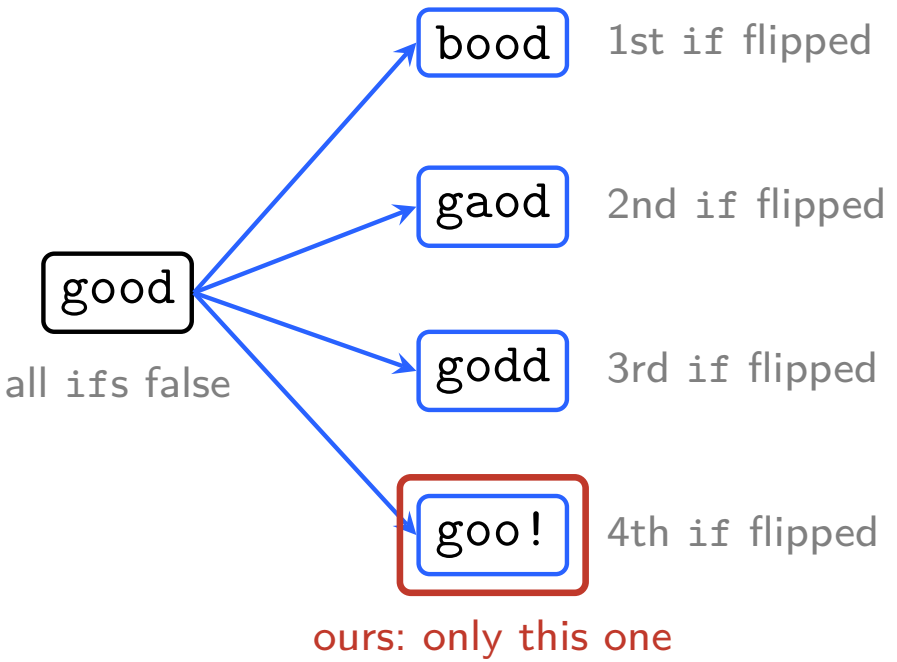
- One run of "good" gives four new inputs: flip each `if`, one at a time.
- Our concolic loop flipped only the last `if`: one input per run.

Godefroid, Levin, Molnar, *SAGE: Whitebox Fuzzing*, CACM 2012, Figure 2.

SAGE: Flip Every Branch

```
void top(char input[4]) {
  int cnt = 0;
  if (input[0] == 'b') cnt++;
  if (input[1] == 'a') cnt++;
  if (input[2] == 'd') cnt++;
  if (input[3] == '!') cnt++;
  if (cnt >= 4) abort();
}
```

The bug needs exactly "bad!".



- One run of "good" gives four new inputs: flip each `if`, one at a time.
- Our concolic loop flipped only the last `if`: one input per run.
- Keep going, and "bad!" comes out in four generations.

Godefroid, Levin, Molnar, *SAGE: Whitebox Fuzzing*, CACM 2012, Figure 2.

SAGE at Microsoft



- SAGE ran last, after static analysis and blackbox fuzzing.

Godefroid, Levin, Molnar, *SAGE: Whitebox Fuzzing*, CACM 2012.

SAGE at Microsoft



1/3 of the file-fuzzing bugs of Windows 7

- SAGE ran last, after static analysis and blackbox fuzzing.
- It still found roughly **one third** of all bugs found by file fuzzing during Windows 7.

Godefroid, Levin, Molnar, *SAGE: Whitebox Fuzzing*, CACM 2012.

SAGE at Microsoft



1/3 of the file-fuzzing bugs of Windows 7

- SAGE ran last, after static analysis and blackbox fuzzing.
- It still found roughly **one third** of all bugs found by file fuzzing during Windows 7.
- Since 2008 it has run nonstop on 100+ machines, with the Z3 solver.

Godefroid, Levin, Molnar, *SAGE: Whitebox Fuzzing*, CACM 2012.

Contents

1. Static Analyzers
    - Clang: Unreachable Code
    - Astrée: Abstract Interpretation
    - Infer: Procedure Summaries
    - CodeQL: Taint as Datalog

2. Running Programs
    - KLEE: Symbolic Execution
    - Jalangi: Instrumentation
    - AFL and SAGE: Fuzzing

3. Beyond This Course

What We Did Not Cover

- **Sparse analysis**: skip the points that cannot change the result.
- **Model checking**: explore every state of a finite model.
- **Verification**: Hoare logic and proof assistants. Sound and complete, if you write the proof.
- **LLMs**: drafting rules and fuzz harnesses, triaging alarms. An analyzer still checks what they produce.

The Course in One Table

question	**static**	**dynamic**
which points?	CFG (Clang)	coverage (AFL, Jalangi)
which values?	intervals (Astrée)	likely invariants
which taint?	may-taint (CodeQL)	runtime taint
which input?	symbolic (KLEE)	fuzzing, concolic (SAGE)

The Course in One Table

question	**static**	**dynamic**
which points?	CFG (Clang)	coverage (AFL, Jalangi)
which values?	intervals (Astrée)	likely invariants
which taint?	may-taint (CodeQL)	runtime taint
which input?	symbolic (KLEE)	fuzzing, concolic (SAGE)

- Every row: one side reports too much, the other too little.

The Course in One Table

question	**static**	**dynamic**
which points?	CFG (Clang)	coverage (AFL, Jalangi)
which values?	intervals (Astrée)	likely invariants
which taint?	may-taint (CodeQL)	runtime taint
which input?	symbolic (KLEE)	fuzzing, concolic (SAGE)

- Every row: one side reports too much, the other too little.
- For a new tool, ask: which row, which side, and what does it skip?

Summary

PLRG

1. Static Analyzers
   - Clang: Unreachable Code
   - Astrée: Abstract Interpretation
   - Infer: Procedure Summaries
   - CodeQL: Taint as Datalog

2. Running Programs
   - KLEE: Symbolic Execution
   - Jalangi: Instrumentation
   - AFL and SAGE: Fuzzing

3. Beyond This Course

Thank You

- Homework 3 is due today on the **LMS**.

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr