Secure Software Development with Continuous & Collaborative Fuzzing

MONASH University

Thuan Pham
Research Fellow

Bugs found by Google ClusterFuzz as of January 2019

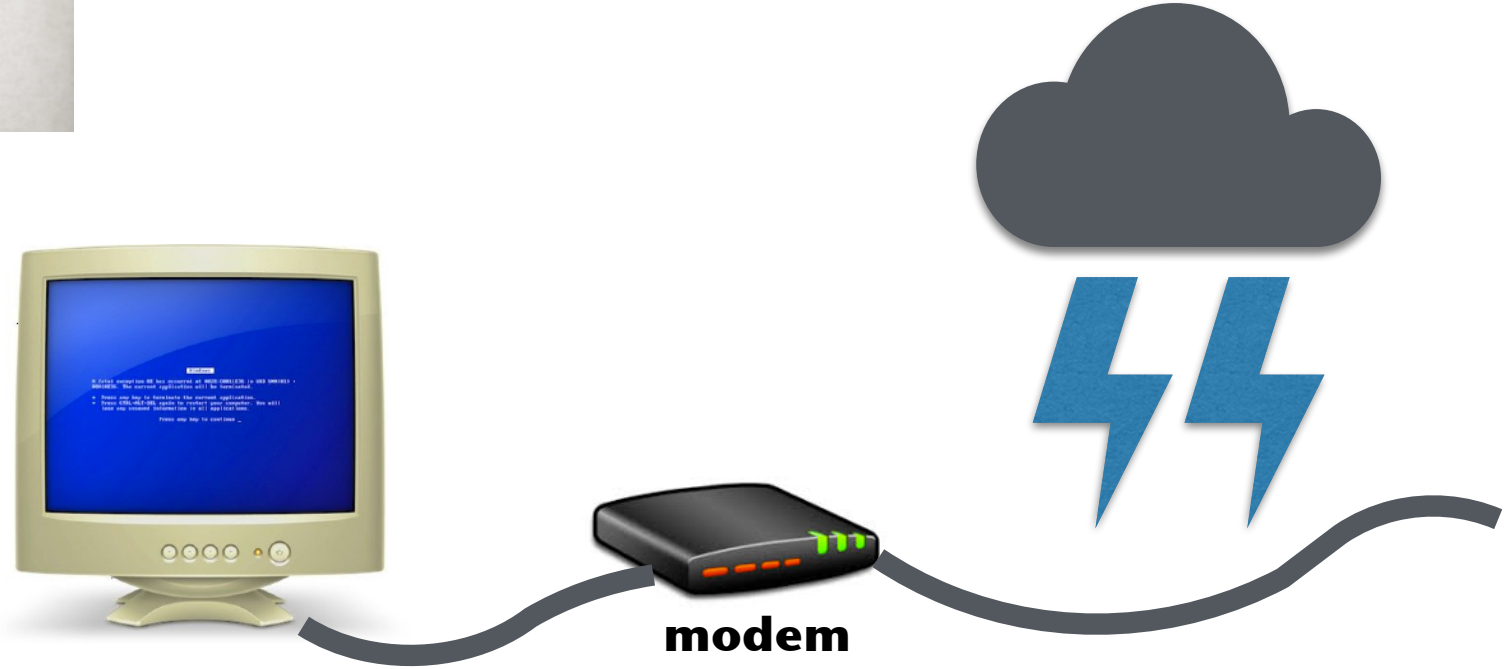
~16,000

~11,000

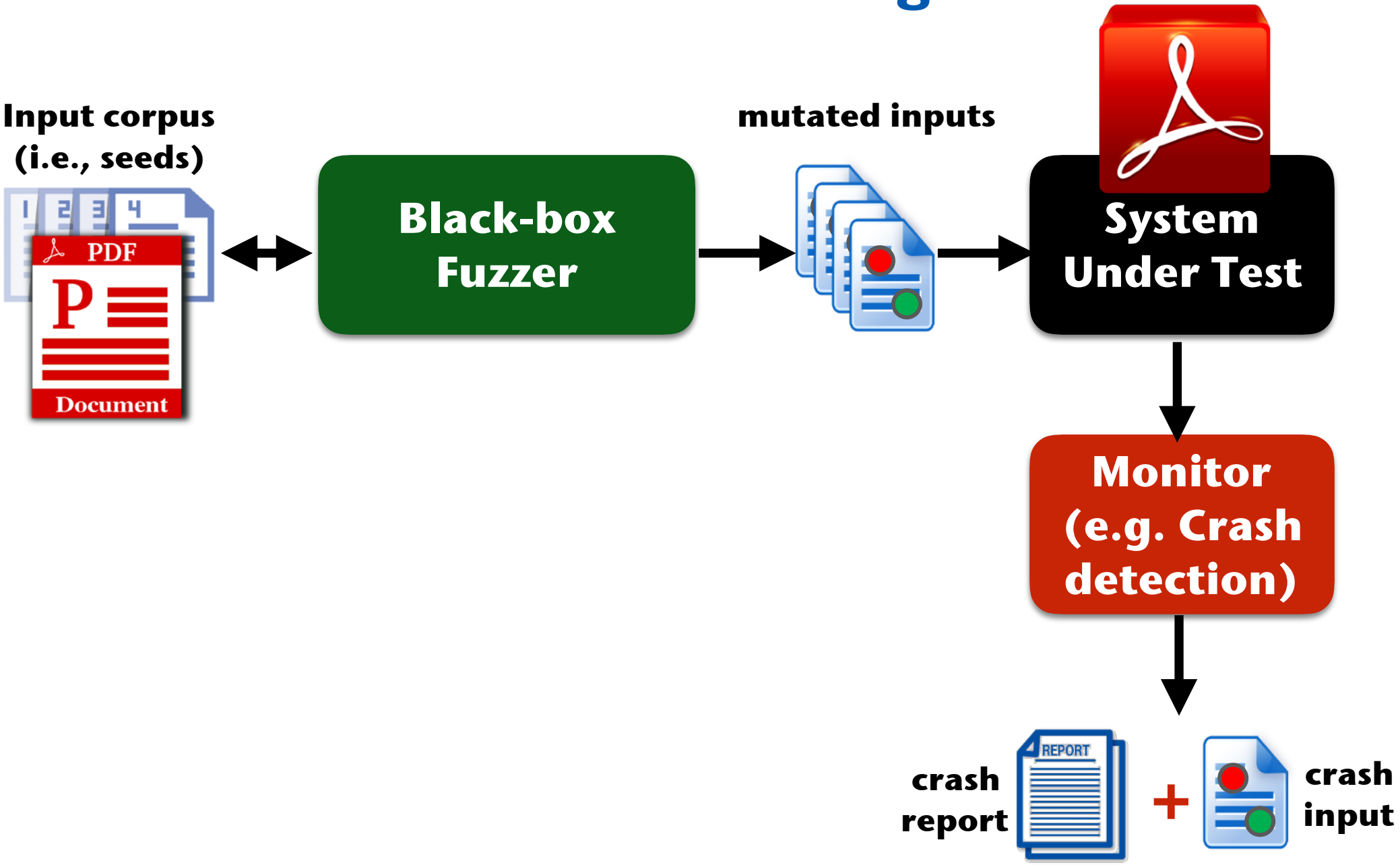
In 160+ open-source projects integrated with OSS-Fuzz

History of Fuzzing

Prof. Barton Miller (University of Wisconsin, Madison), "the father of fuzzing", coined the term in 1988

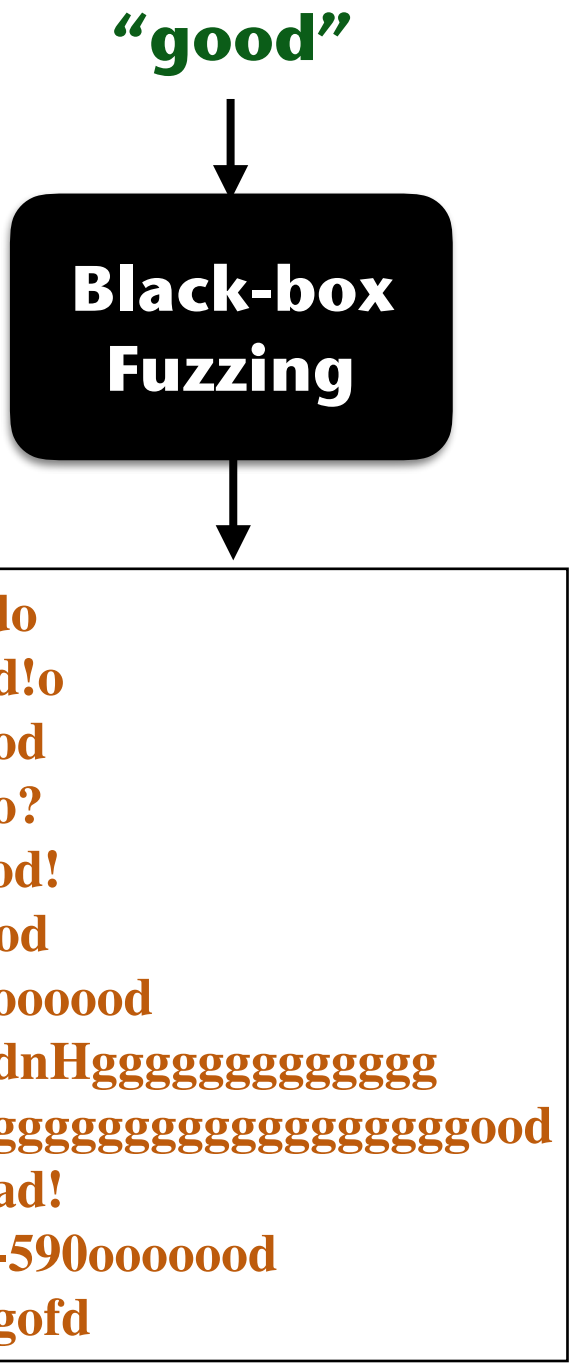


How does *Black-box* Fuzzing work?



Example

```
void Fn(char buf[4])
{
  if (buf[0] == 'b') {
    if (buf[1] == 'a') {
      if (buf[2] == 'd') {
        if (buf[3] == '!') {
          CRASH();
        }
      }
    }
  }
}
```

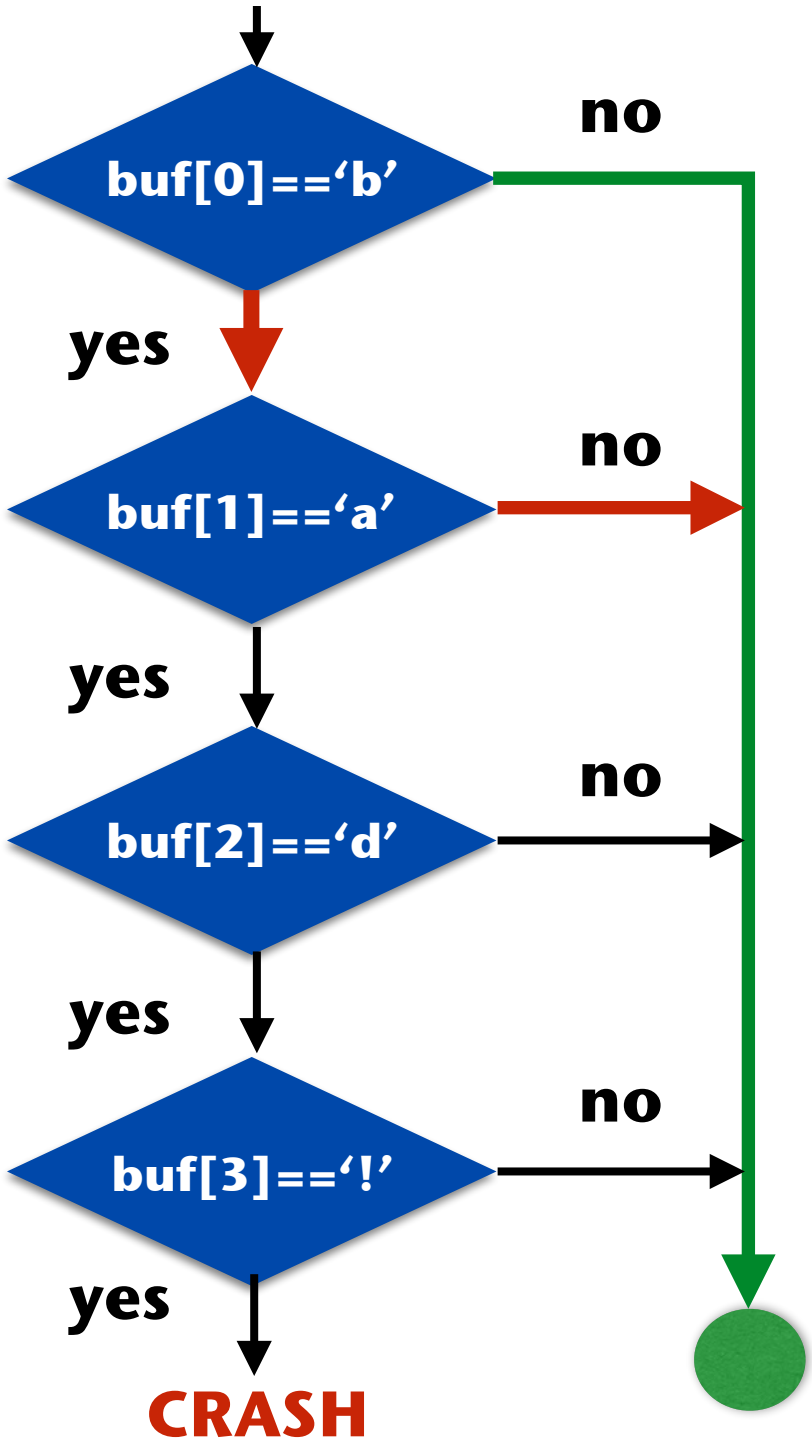


A deeper look

o!do
god!o
good
?oo?
cood!

b?od

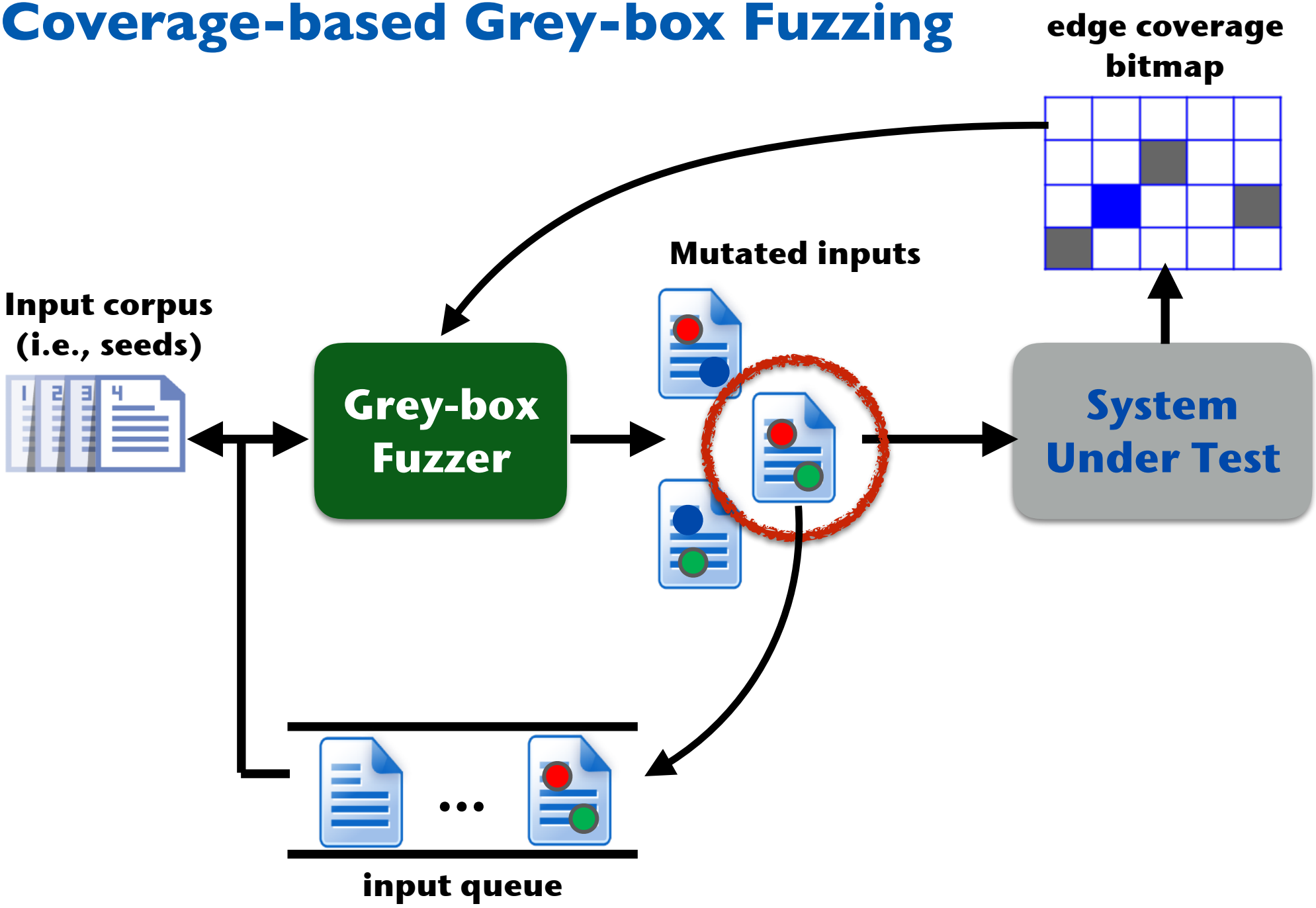
goooooood
godnHgggggggggggg
gggggggggggggggggggood
goad!
go-590oooooood
$ggofd



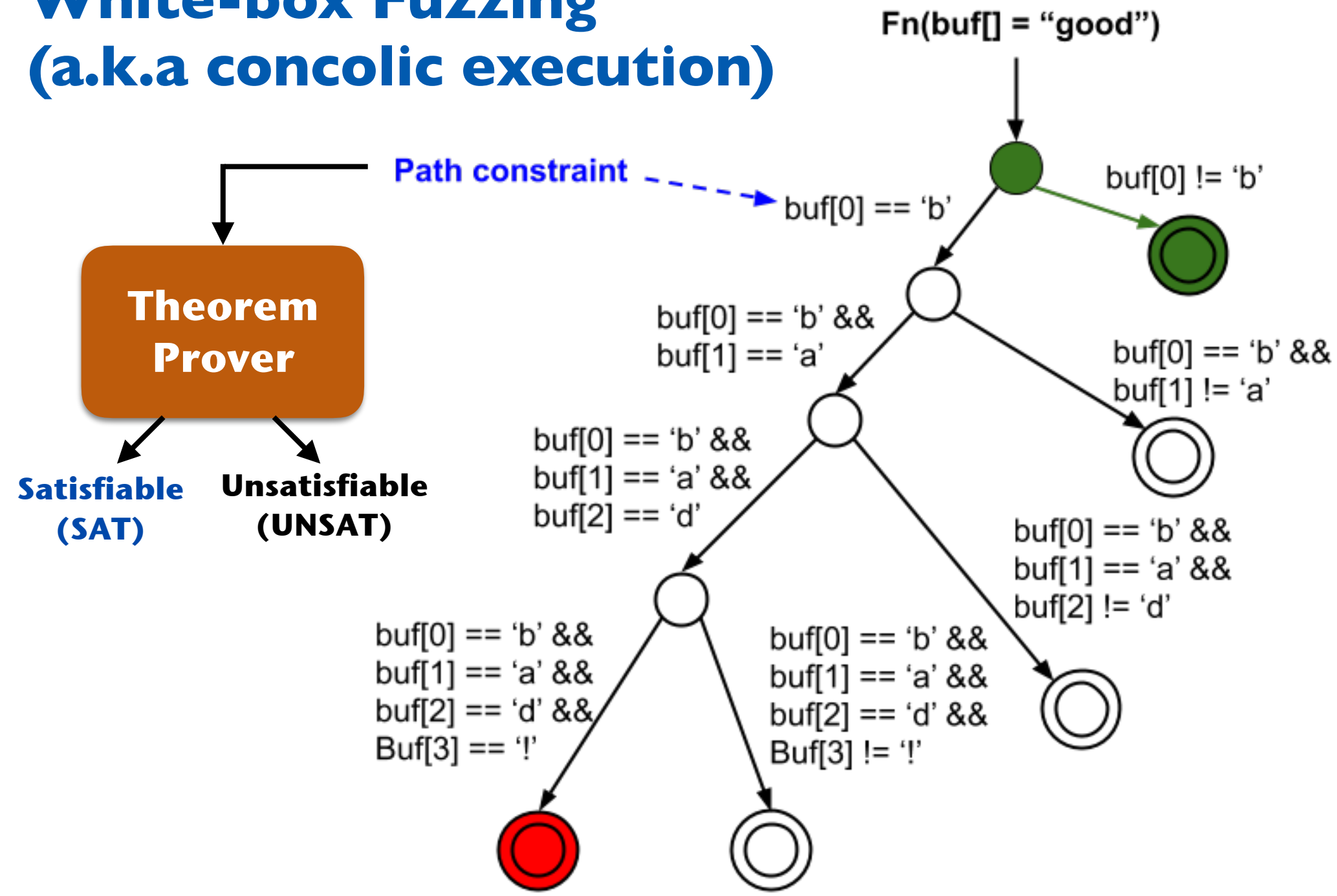
White-, Grey-box Fuzzing

- Generate an input & run the program
- Watch the program path traversed by the input
- Try never to repeat the path twice
- Monitor for abnormal behaviours (e.g., crashes)

Coverage-based Grey-box Fuzzing

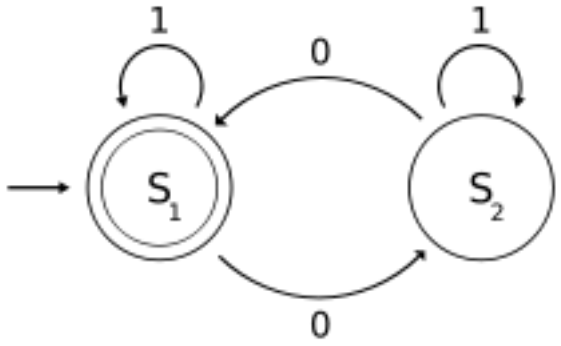


White-box Fuzzing (a.k.a concolic execution)



Directed Fuzzing

Structure-aware Fuzzing

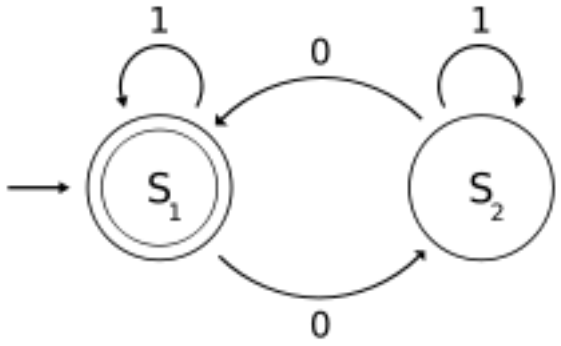


Stateful Fuzzing

Directed Fuzzing

(ICSE'15, CCS'17)

Structure-aware Fuzzing



Stateful Fuzzing

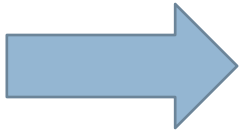
Directed Fuzzing

Choose "directions" to manage the search space & discover paths which are more likely to trigger program bugs in shorter time

Motivations for Directed Fuzzing

- Patch testing - continuous testing
- Crash reproduction
- Targeted vulnerability discovery
- Variant analysis

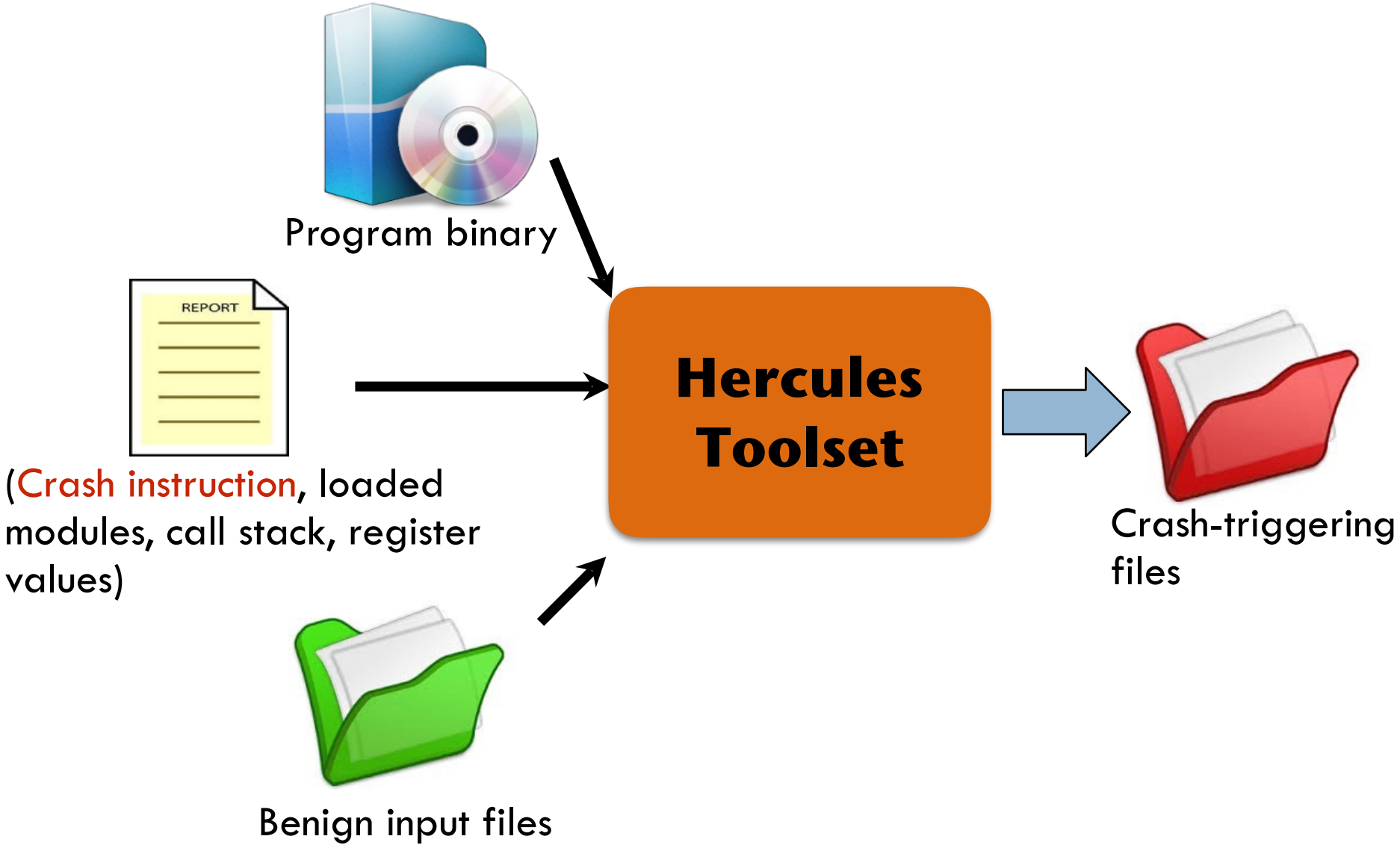
Directed Search in White-box Fuzzing
For Crash Reproduction Problem



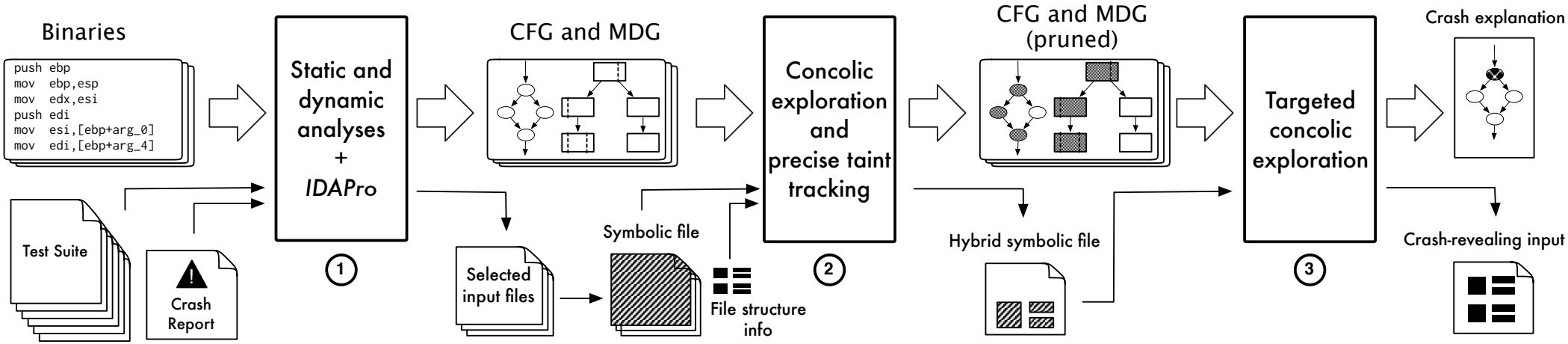
Crash reproducing supports

- In-house debugging and fixing
- Vulnerability assessment

Overview

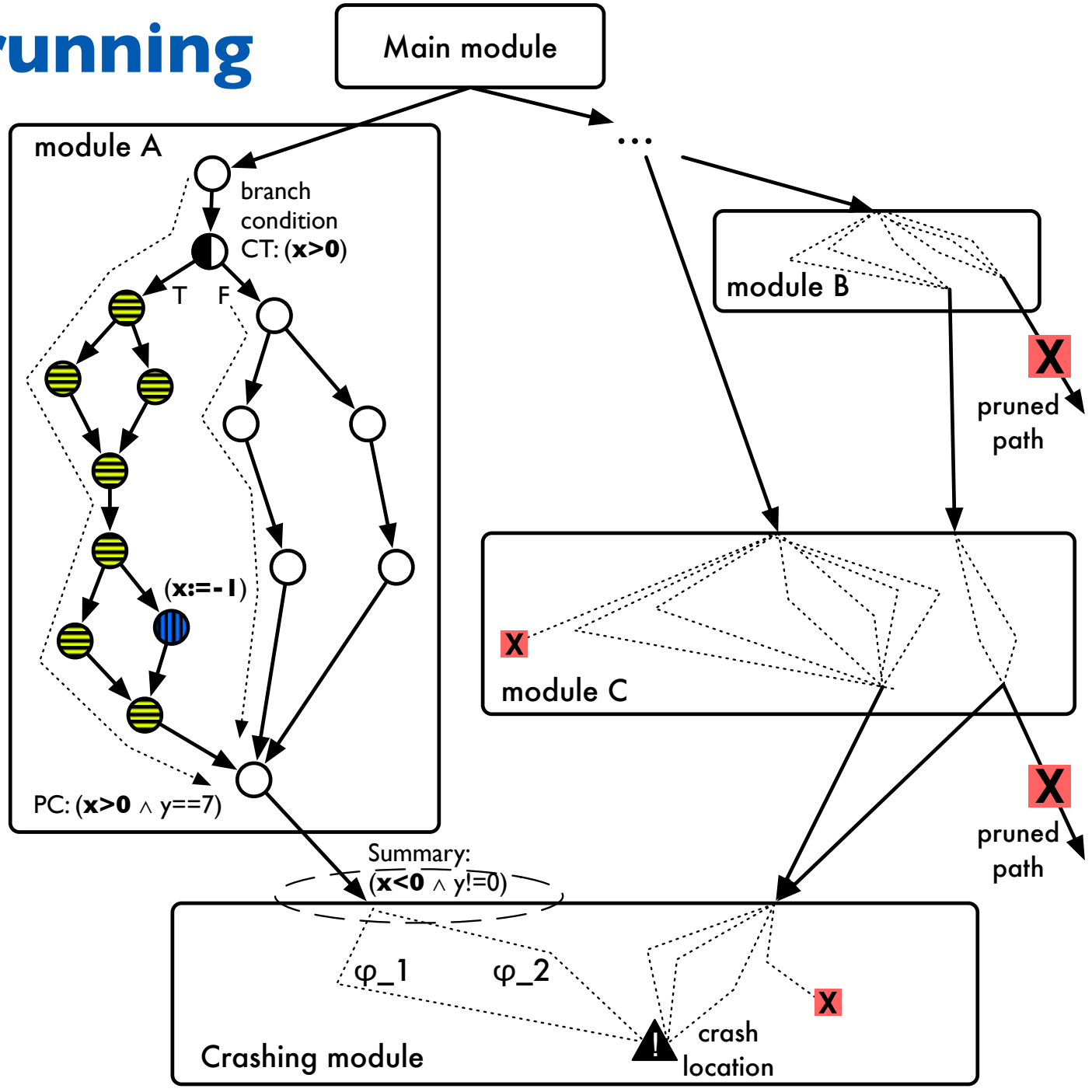

Program binary
REPORT
(Crash instruction, loaded modules, call stack, register values)
Hercules Toolset
Crash-triggering files
Benign input files

3-stage workflow



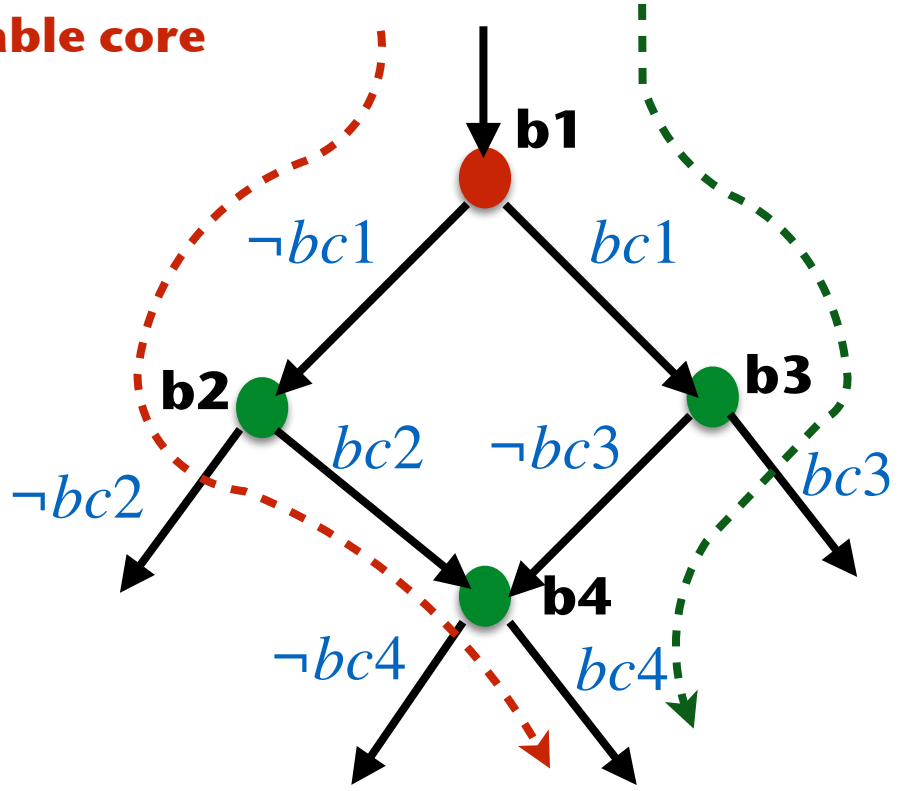
- ***involving a set of techniques***
  - control flow graph extraction
  - symbolic-execution based taint analysis
  - graph prunning
  - loop & string functions handling
  - systematic guided backtracking

Graph prunning



UNSAT-core Guided Backtracking

minimal unsatisfiable core



X **Crash Instruction**
(Crash condition: CC)

1st attempt:
PC = $bc1 \wedge \neg bc3 \wedge bc4$
CheckSAT(PC ^ CC) == UNSAT
bc1 contradicts CC

→ 1) Backtrack to b1
2) Take another branch →

2nd attempt
PC' = $\neg bc1 \wedge bc2 \wedge bc4$
CheckSAT(PC' ^ CC) == SAT
Generate crashing input

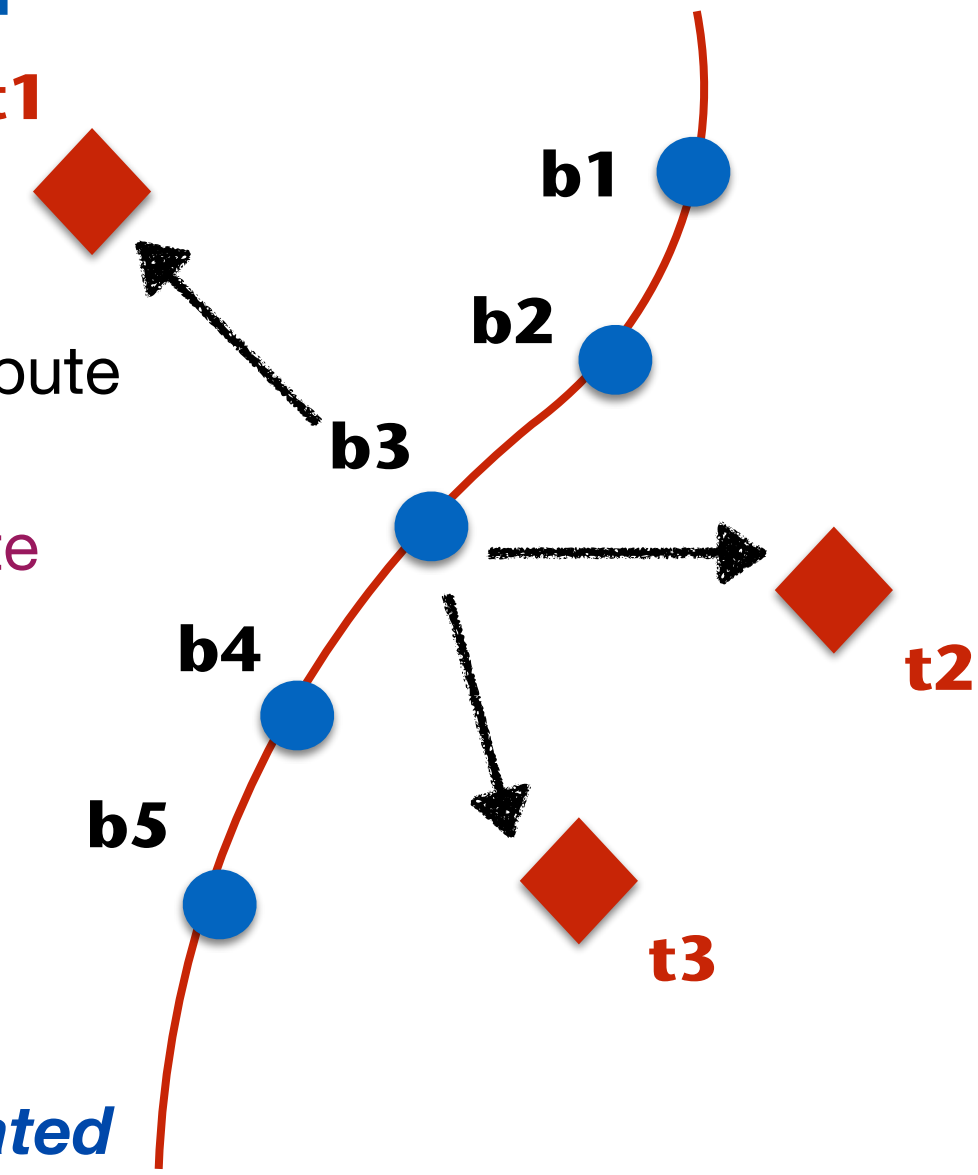
Results

Program	Advisory ID	#Seed files	Hercules	Peach	S2E
WMP 9.0	CVE-2014-2671	10	✓	✗	✗
WMP 9.0	CVE-2010-0718	10	✓	✗	✗
AR 9.2	CVE-2010-2204	10	✓	✗	✗
RP 1.0	CVE-2010-3000	10	✓	✗	✗
MP 0.35	CVE-2011-0502	10	✓	✓	✓
OV 1.04	CVE-2010-0688	10	✓	✓	✗

Hercules scales to large binary programs such as Adobe Reader and Windows Media Player and outperforms the baselines

Directed Greybox as Optimisation Problem

1. Instrumentation Time:
   - Extract call graph (CG) and control-flow graphs (CFGs).
   - For each basic block (BB), compute distance to target locations.
   - Instrument program to aggregate distance values.
2. Runtime, for each input
   - collect coverage and distance information, and
   - decide how long to be fuzzed based on distance using ***simulated annealing*** algorithm.

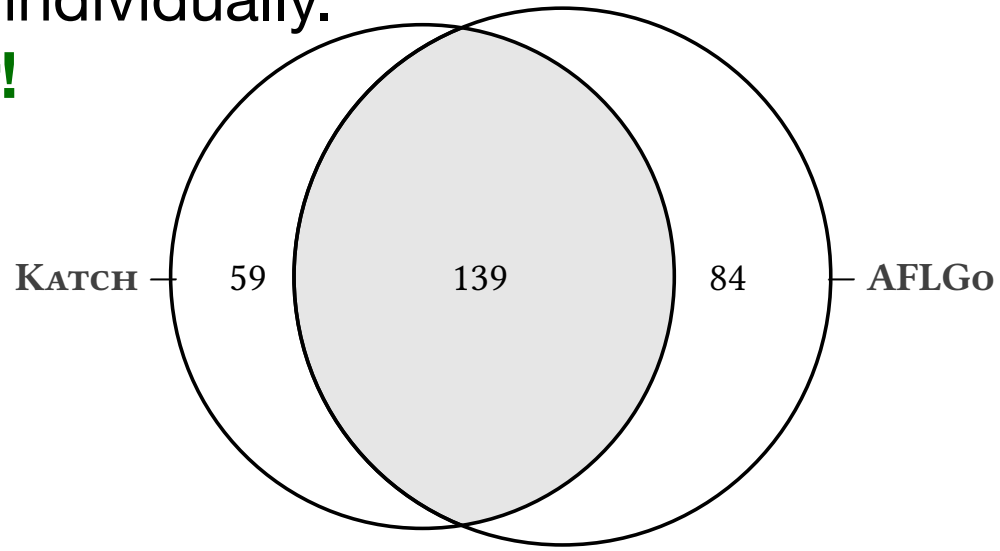


Results

Patch Testing: Reach changed statements

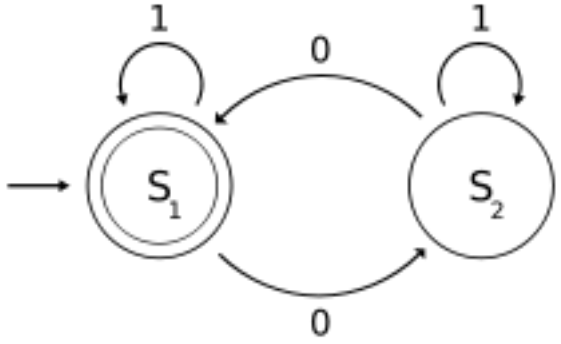
- State-of-the-art in patch testing
  - **KATCH** (based on KLEE symbolic execution tool)

- Patch Coverage (#changed BBs reached)
  - While we would expect KATCH to take a substantial lead, **AFLGo outperforms KATCH** in terms of patch coverage.
  - **BUT**: **Together** they cover **42%** and **26%** more than **KATCH** and **AFLGo** individually. **They complement each other!**



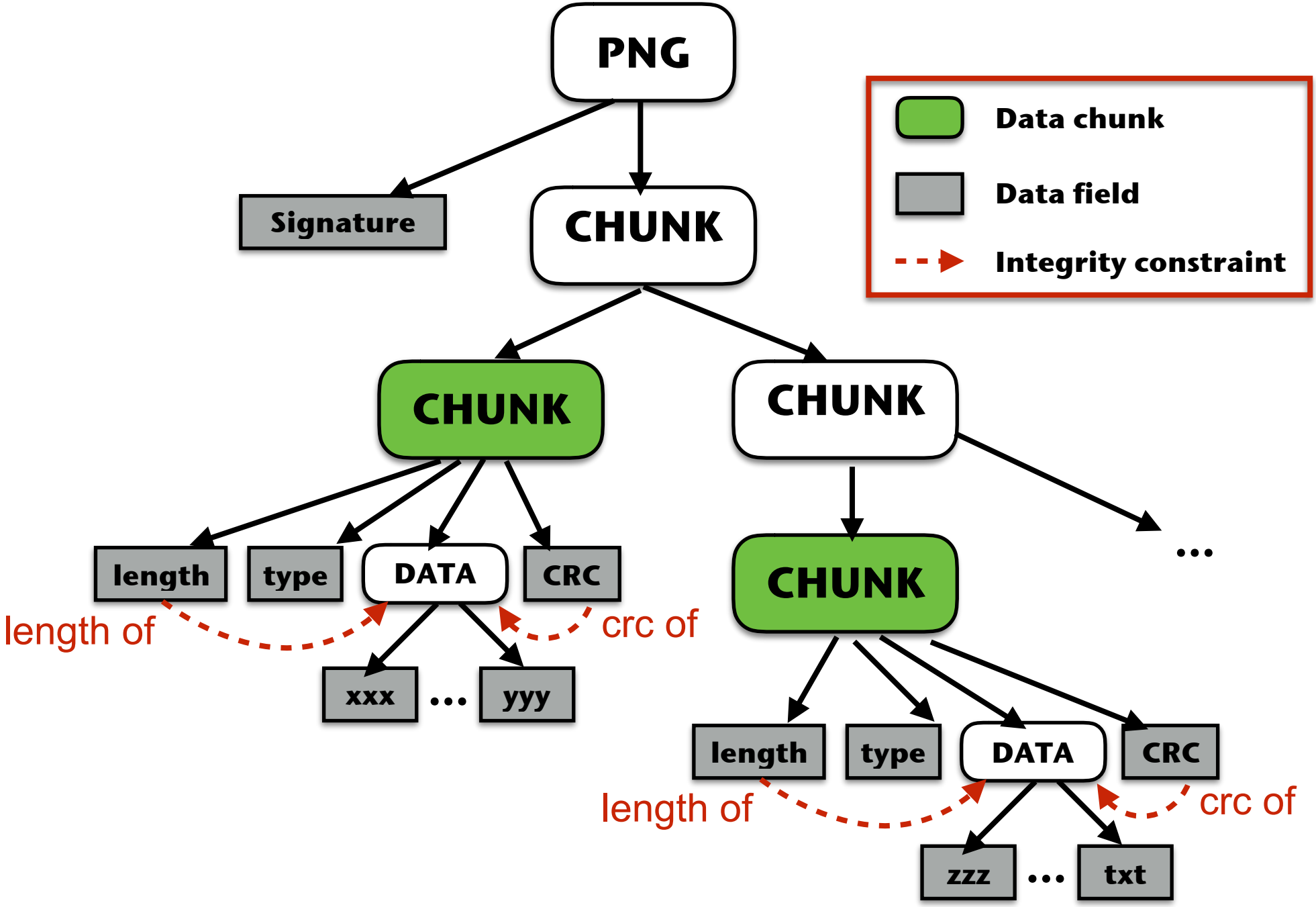
Directed Fuzzing

Structure-aware Fuzzing



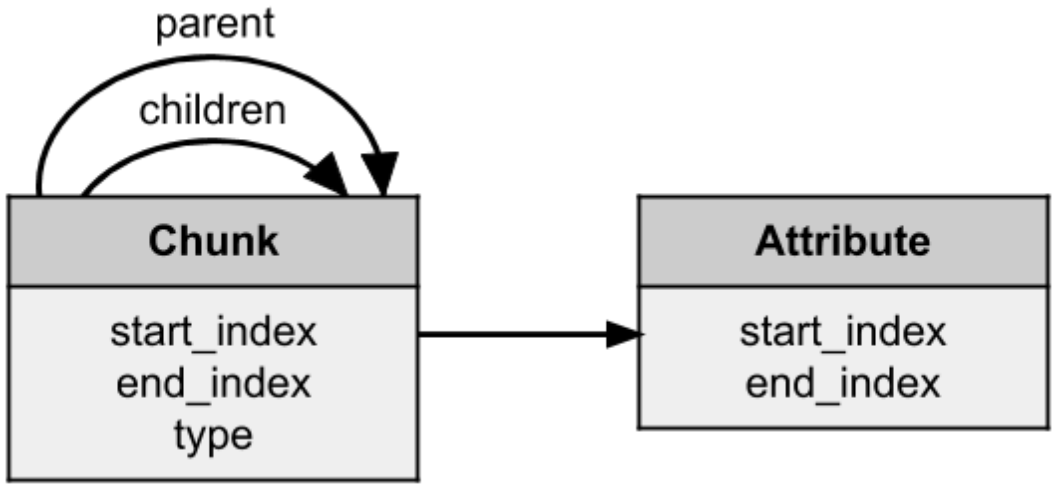
Stateful Fuzzing

(ASE'16, TSE'19)


PNG
Signature
CHUNK
Data chunk
Data field
Integrity constraint
CHUNK
CHUNK
length
type
DATA
CRC
length of
crc of
xxx
...
yyy
CHUNK
...
length
type
DATA
CRC
length of
crc of
zzz
...
txt

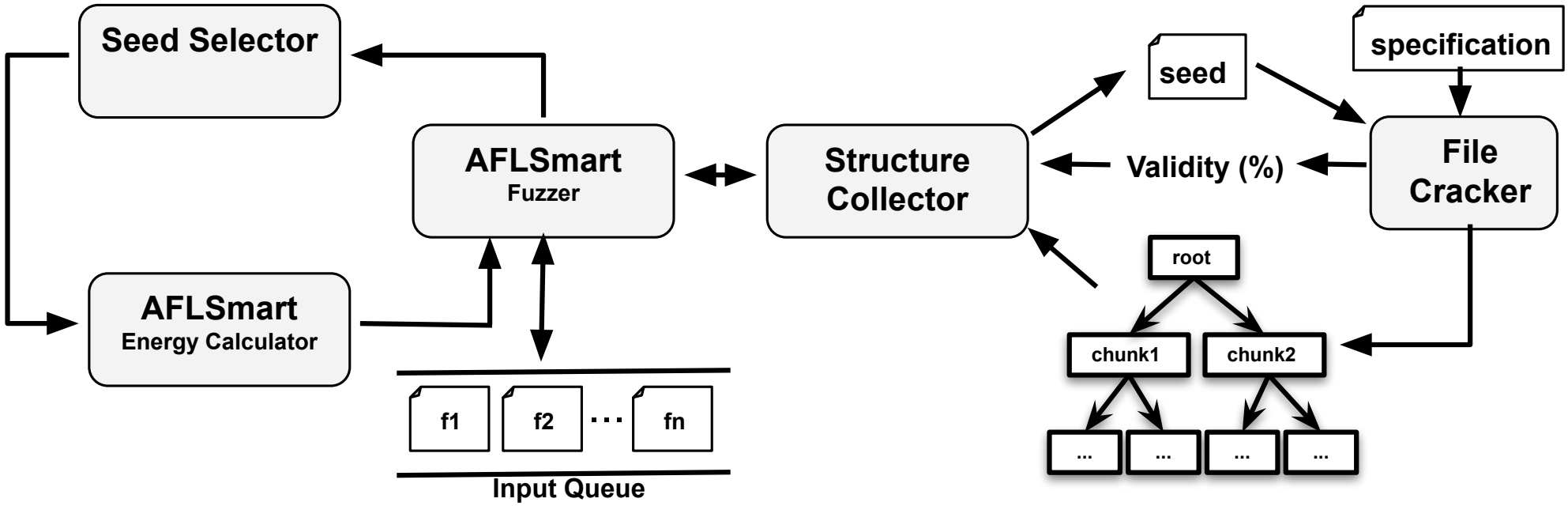
AFLSmart: Smart Greybox Fuzzing

- Design a *high-level structural representation* (virtual file structure) representing all chunk-based file formats



- Apply *higher-order mutation operators* that work at "chunks" level together with bit-/byte-level mutators
- Prioritise *semantically valid* seeds. (i.e., 100% valid means the whole seed can be successfully parsed by the parser)

Architecture of AFLSmart

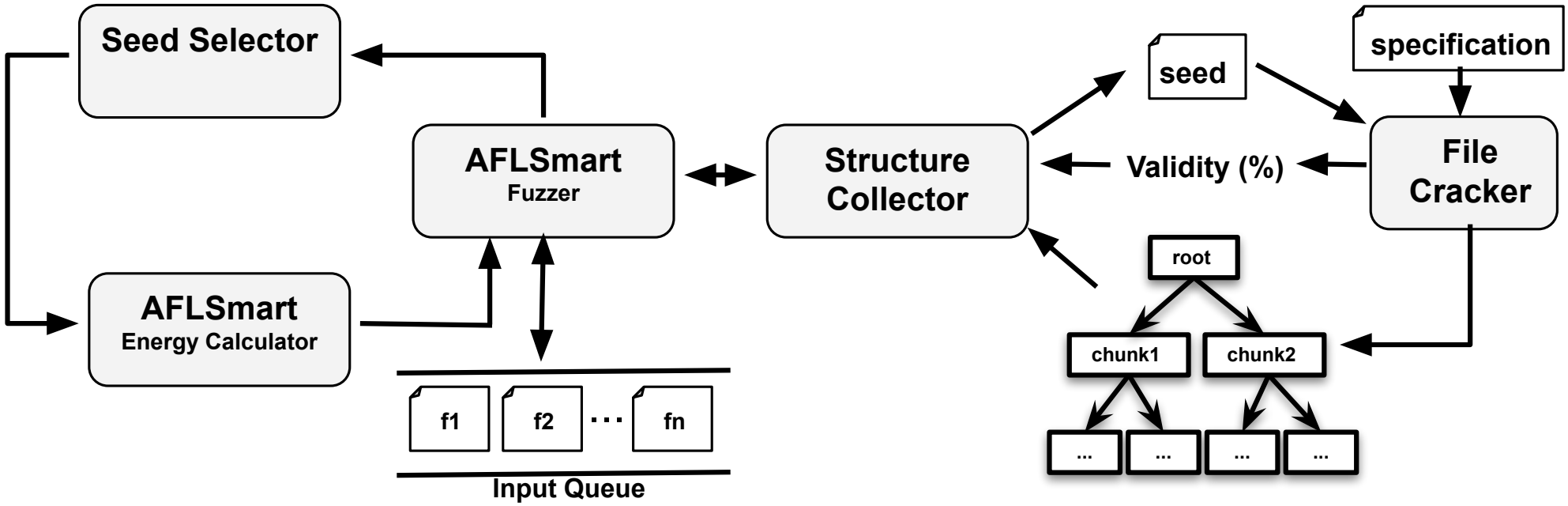


- **File Cracker**: transforms seeds (e.g., PNG files) to a tree-based structure, given an ***input model***
- **Energy Calculator**: spends more time to fuzz more valid inputs

Sample input model of PNG image file

```
<DataModel name="Chunk">
  <String name="ckID" length="4"/>
  <Number name="cksize" size="32" >
    <Relation type="size" of="Data"/>
  </Number>
  <Blob name="Data"/>
  <Padding alignment="16"/>
</DataModel>
<DataModel name="ChunkFmt" ref="Chunk">
   <String name="ckID" value="fmt "/>
   <Block name="Data">
      <Number name="wFormatTag" size="16"/>
      <Number name="nChannels" size="16"/>
      <Number name="nSampleRate" size="32"/>
      <Number name="nAvgBytesPerSec" size="32"/>
      <Number name="nBlockAlign" size="16" />
      <Number name="nBitsPerSample" size="16"/>
   </Block>
</DataModel>
...
<DataModel name="Wav" ref="Chunk">
  <String name="ckID" value="RIFF"/>
  <String name="WAVE" value="WAVE"/>
  <Choice name="Chunks" maxOccurs="30000">
    <Block name="FmtChunk" ref="ChunkFmt"/>
    ...
    <Block name="DataChunk" ref="ChunkData"/>
  </Choice>
</DataModel>
```

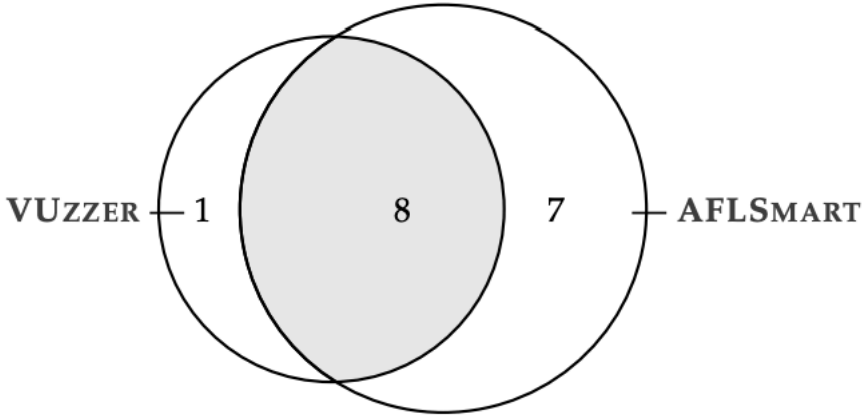
Architecture of AFLSmart



- **File Cracker**: transforms seeds (e.g., PNG files) to a tree-based structure, given an ***input model***
- **Energy Calculator**: spends more time to fuzz more valid inputs

Results

Subject	Bug-ID	AFL	AFLFast	Peach	AFLSmart
WavPack	CVE-2018-10536	✗	✗	✗	20/20
	CVE-2018-10537	✗	✗	✗	12/20
	CVE-2018-10538	✗	✗	✗	20/20
	CVE-2018-10539	✗	✗	✗	15/20
	CVE-2018-10540	10/20	15/20	11/20	12/20
Binutils	Bugzilla-23062	10/20	11/20	✗	11/20
	Bugzilla-23063	13/20	12/20	✗	10/20
	CVE-2018-10372	16/20	18/20	✗	16/20
	CVE-2018-10373	11/20	12/20	✗	14/20
	Bugzilla-23177	✗	✗	✗	13/20
LibPNG	CVE-2018-13785	✗	✗	✗	6/20
Libjasper	Issue-174	8/20	9/20	✗	9/20
	Issue-175	12/20	14/20	✗	12/20
	CVE-2018-19539	✗	✗	✗	15/20
	CVE-2018-19540	✗	✗	✗	7/20
	CVE-2018-19541	✗	✗	✗	6/20
	CVE-2018-19542	✗	7/20	✗	9/20
	CVE-2018-19543	8/20	12/20	✗	13/20
	Issue-182-6	19/20	20/20	✗	18/20
	Issue-182-7	16/20	18/20	✗	19/20
	Issue-182-8	12/20	13/20	✗	16/20
	Issue-182-9	12/20	14/20	✗	11/20
	Issue-182-10	14/20	11/20	✗	15/20
OpenJPEG	Email-Report-1	✗	✗	✗	8/20
	Email-Report-2	✗	✗	✗	13/20
	Issue-1125	✗	✗	✗	15/20
LibAV	Bugzilla-1121	✗	✗	✗	5/20
	Bugzilla-1122	✗	✗	✗	6/20
	Bugzilla-1123	18/20	18/20	✗	18/20
	Bugzilla-1124	15/20	18/20	✗	16/20
	Bugzilla-1125	✗	✗	✗	8/20
	Bugzilla-1127	13/20	15/20	✗	18/20
FFmpeg	Email-Report-3	✗	✗	✗	3/20



AFLSmart vs Vuzzer on Vuzzer's benchmark

42 zero-day bugs found
23 CVEs assigned

aflsmart / aflsmart

Unwatch 42 | Star 317 | Fork 61

Code | Issues 4 | Pull requests 0 | Actions | Projects 0 | Wiki | Security | Insights

Smart Greybox Fuzzing (https://thuanpv.github.io/publications/TSE19_aflsmart.pdf)

247 commits | 1 branch | 0 packages | 0 releases | 9 contributors

Hot fuzz: Bug detectives whip up smarter version of classic ...

https://www.theregister.co.uk › 2018/11/28 › better_fuzzer_aflsmart

Nov 28, 2018 - Known as **AFLSmart**, this fuzzing software is built on the powerful American ... We're told **AFLSmart** is pretty good at testing applications for common .... The Register - Independent **news** and views for the tech community.

AFLSmart | Latest AFLSmart News, Articles and Updates

https://cyware.com › tags › aflsmart

AFLSmart - Check out latest **news** and articles about **AFLSmart** on Cyware.com. We provide machine learning based curation engine brings you the top and ...

Researchers Introduce Smart Greybox Fuzzing | SecurityWeek ...

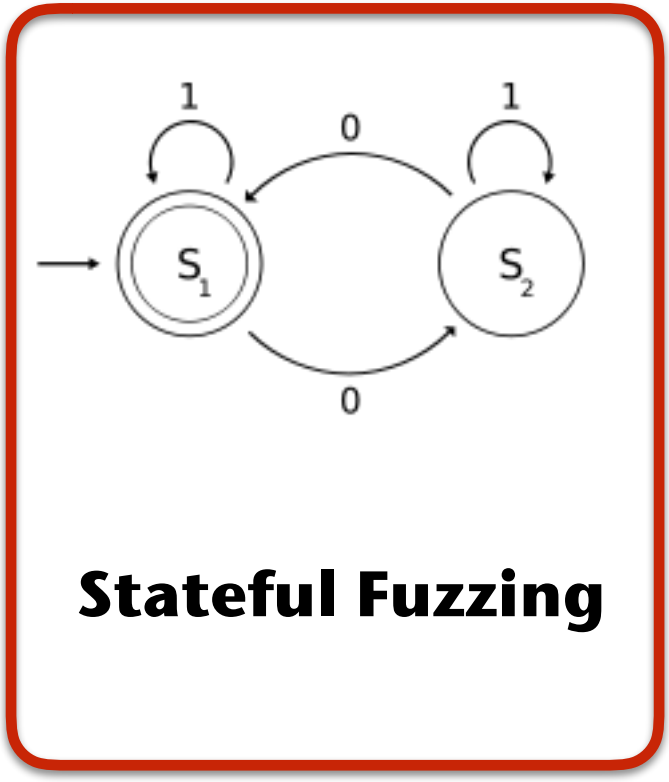
https://www.securityweek.com › researchers-introduce-smart-greybox-fuzz...

Nov 29, 2018 - Information Security **News**, IT Security **News** and Cybersecurity Insights: ... According to the experts, **AFLsmart** is highly efficient in analyzing ...

Google OSS-Fuzz

Directed Fuzzing

Structure-aware Fuzzing



Stateful Fuzzing

(ICST'20)

Stateful Greybox Fuzzing: Motivation

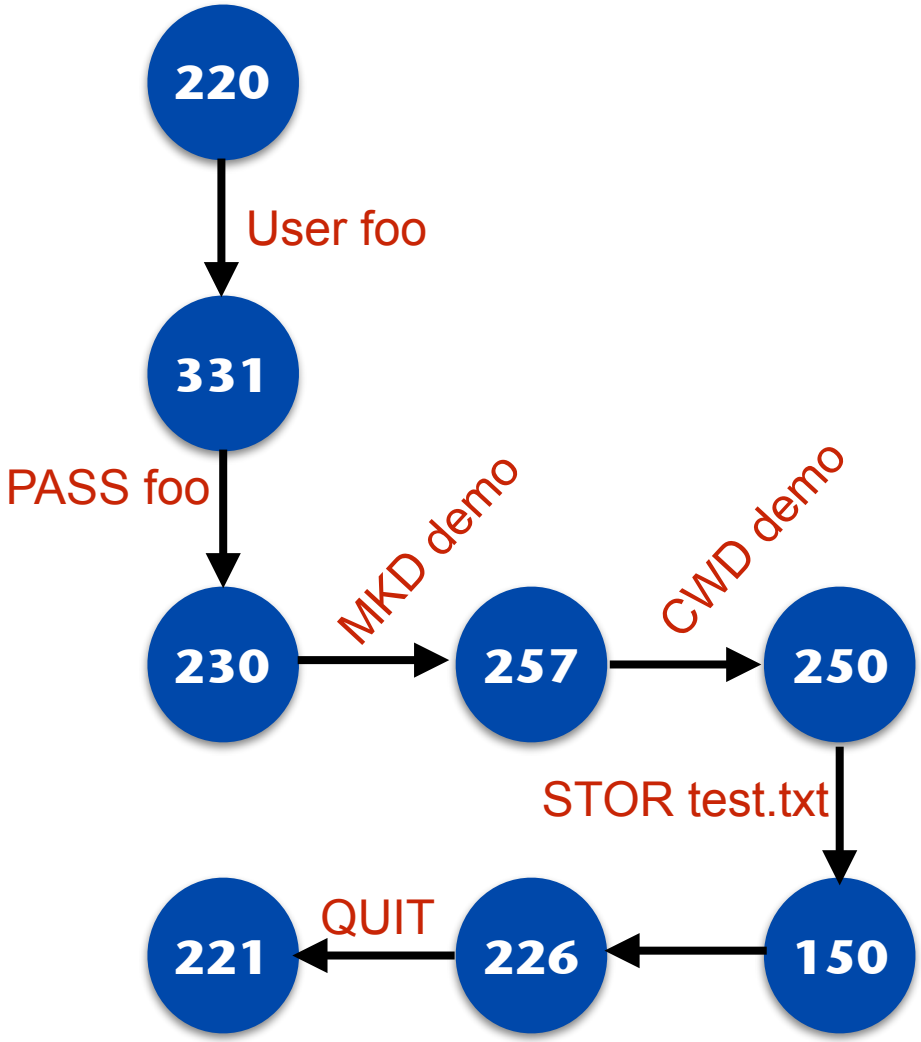
- **Stateless**
  - Program behaviour only depends on the *current input*
  - e.g., file processing programs
- **Stateful**
  - Program behaviour depends on the *current input & current program state*

> *"One of the things that I struggle with is the limitation AFL seems to have, in that it only performs fuzzing with one input (a file). For many systems such as network protocols, it would be useful if fuzzing could be done on a sequence of inputs. This sequence of inputs might be for example messages necessary to complete a handshake in TLS/TCP."*
>
> — Paul (a member of the AFL's user group) [8]

> *"I'm interested in doing something fairly non-traditional and definitely not currently supported by AFL. I would like to perform fuzzing of a large and complex external server that cannot easily be stripped down into small test cases."*
>
> — Tim Newsham (a member of the AFL's user group) [8]

Example - FTP protocol

220 FTP Server ready
USER foo
331 User foo OK. Password required
PASS foo
230 User logged in, proceed.
MKD demo
257 Directory created.
CWD demo
250 Requested file action okay, completed.
STOR test.txt
150 File status okay
226 Transfer complete
QUIT
221 Goodbye!



A sample FTP session to upload a file (test.txt) to a new folder (demo) on the server

Key challenge in testing stateful servers

Stateful servers only accept sequences of (valid) messages in (valid) orders

- We need a *state machine representing the implemented protocol* to guide the test generation process

Q1. How to know the current server state?

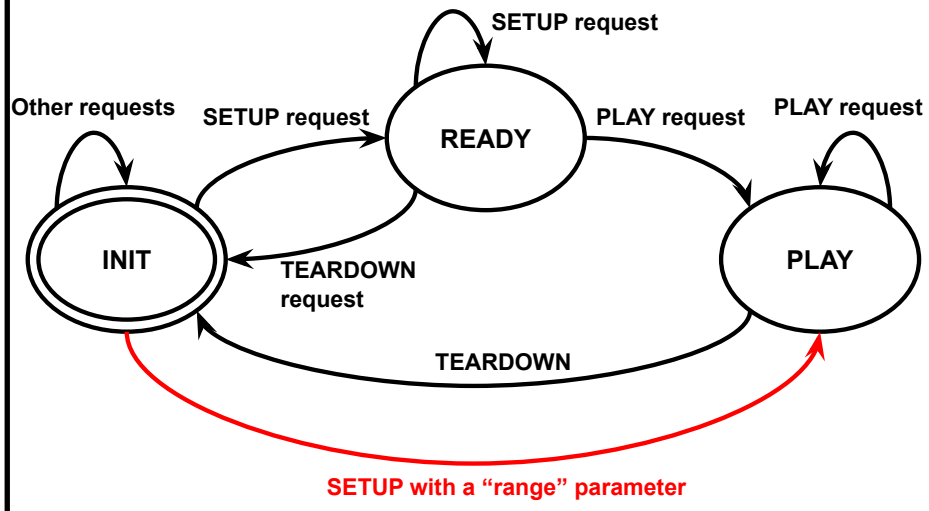
Use the server response code (e.g., 230 – user logged in)

Q2. How to construct the state machine?

Q3. How to effectively explore the state machine?

Q2. How to construct the state machine

- Time consuming
- Require domain knowledge
- Implemented protocol could be different from the standard specification



Manual & static approach

- Not time consuming
- Capture the exact implemented protocol

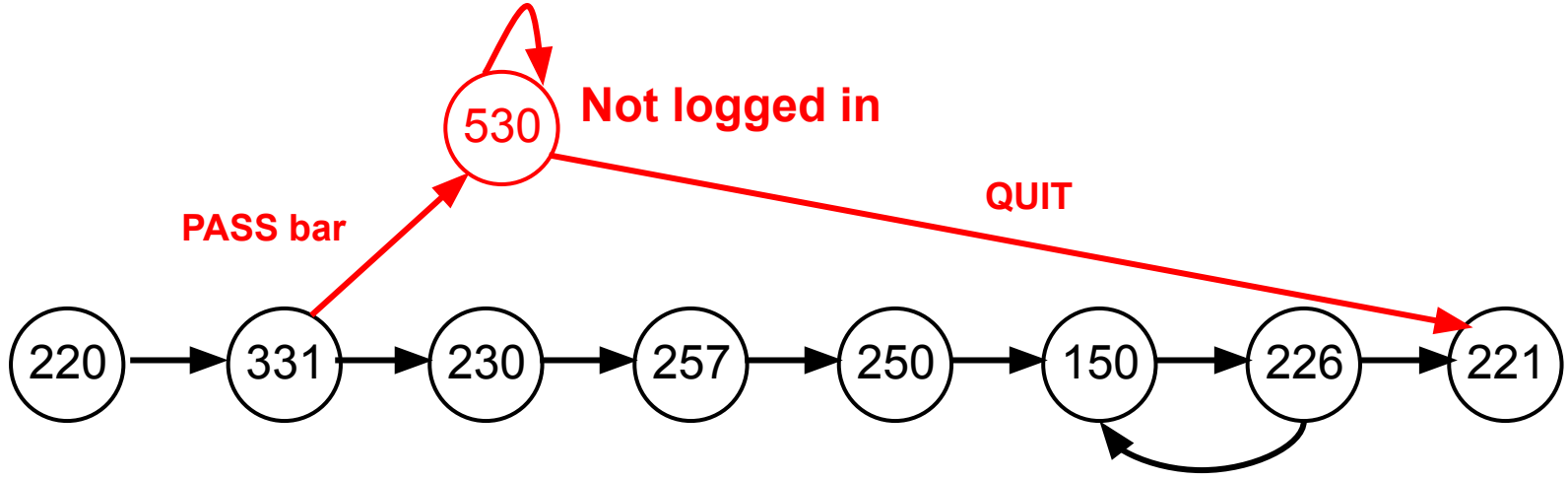
Automatic & dynamic approach

Q3. How to explore the state machine

- Prioritise *"progressive" states* that contribute more towards increased code coverage than others.
  - Step-1: Select a target state
  - Step-2: "Replay" to reach a selected progressive state
  - Step-3: Mutate the message(s) consumed by the server at that state

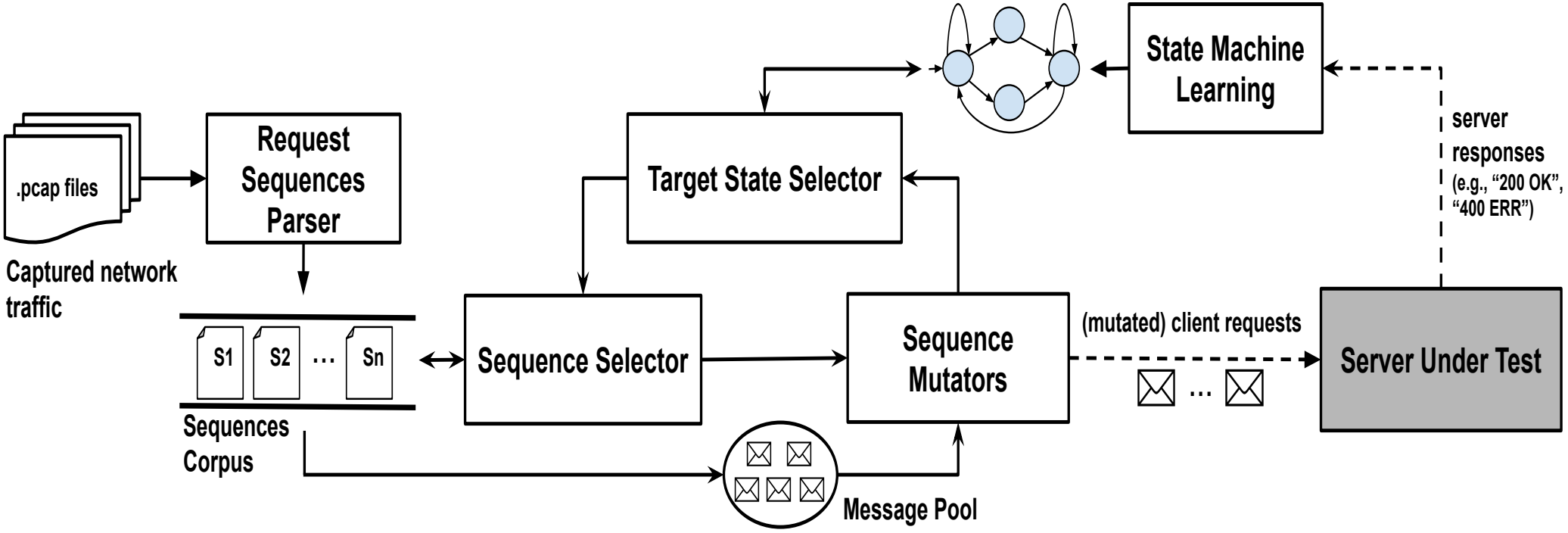
Original message sequence (i.e., seed input)

And then, state 331 (User OK) is targeted



inferred state machine

Architecture of AFLNet



- ***Input:*** captured network traffic in .pcap files
- ***Output:***
  - Implemented state machine
  - bug-triggering inputs

Results

		Branch Coverage			Statement Coverage			State Coverage		
		%Increase	$\hat{A}_{12}$	*p-value*	*%Increase*	$\hat{A}_{12}$	*p-value*	*%Increase*	$\hat{A}_{12}$	*p-value*
AFLNET vs AFLNWE	lightftp	121.06 %	1.000	< 0.001	79.45 %	1.000	< 0.001	85.00 %	1.000	< 0.001
	live555	3.49 %	0.335	0.076	2.44 %	0.228	0.003	8.58 %	0.392	0.230
AFLNET vs BOOFUZZ	lightftp	57.73 %	1.000	0.026	49.72 %	1.000	0.026	37.00 %	1.000	0.020
	live555	64.13 %	1.000	0.026	62.09 %	1.000	0.026	100.00 %	1.000	0.019

AFLNet outperforms BooFuzz and AFL

2 critical zero-day vulnerabilities found (CVE score 9.8)

D-ViewCam

Alban Lecocq @skeetmtp · 13 Jan

I'm using Afl to find "packet of death" for 3 years, but never manage to detect statefull bug with it. Indeed there little litterature on the subject. Can't wait to read more details on #AFLNet

Extensions

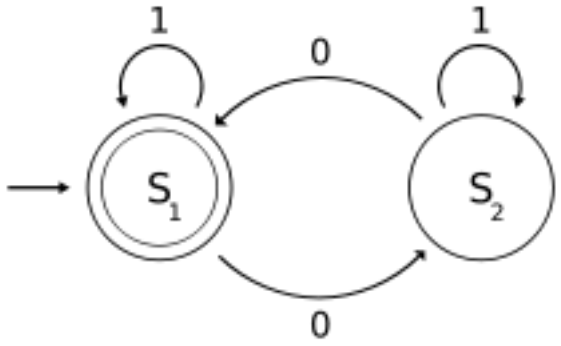
- Make state machine learning more fine grained

A joint project with

- Make AFLNet work for IoT/Industrial network protocols (e.g., CAN bus, TLS/DTLS protocols)

Directed Fuzzing

Structure-aware Fuzzing



Stateful Fuzzing

CYBER GRAND_CHALLENGE

Hot fuzz: Bug detectives whip up smarter version of classic ...

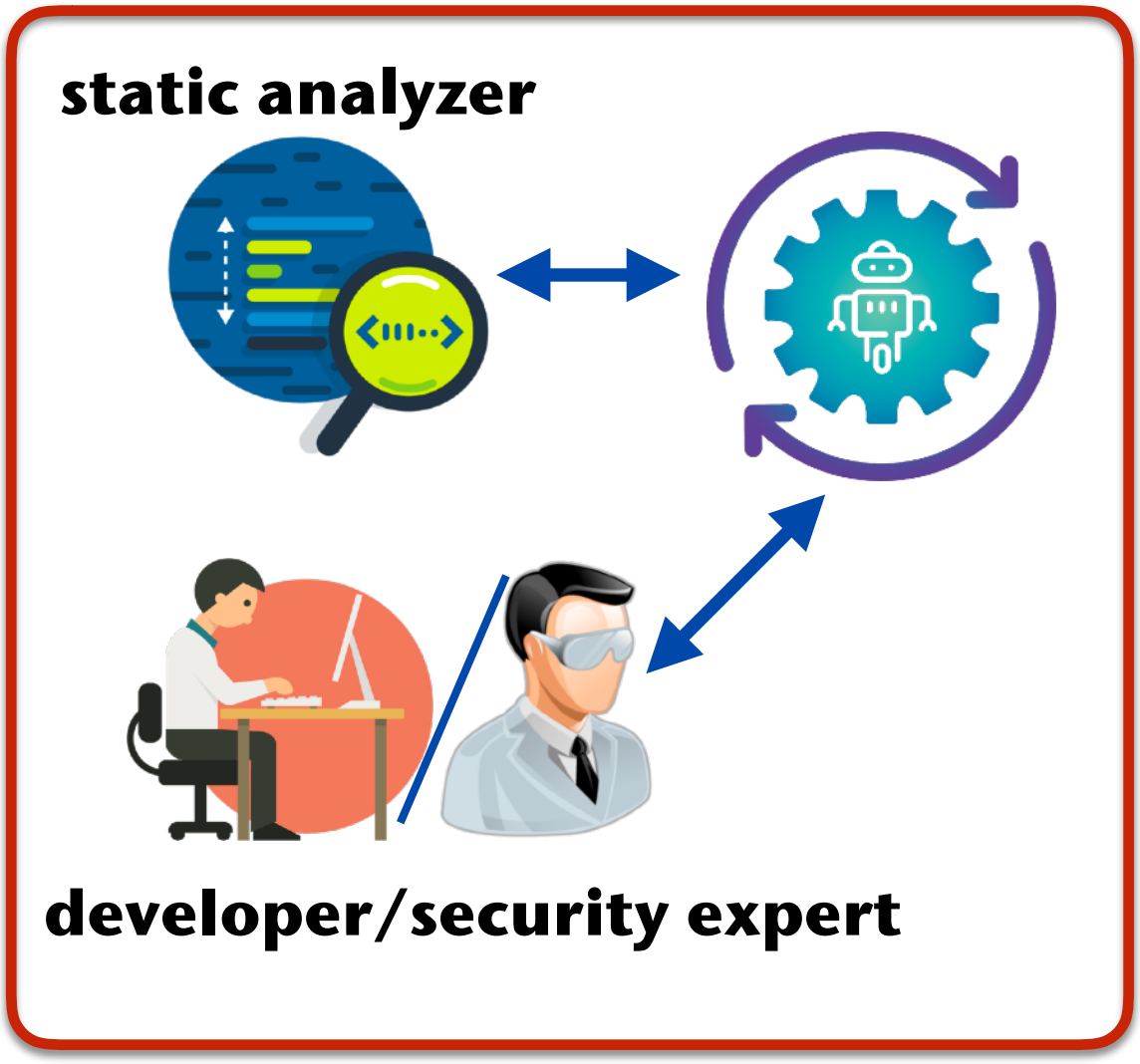
https://www.theregister.co.uk › 2018/11/28 › better_fuzzer_aflsmart

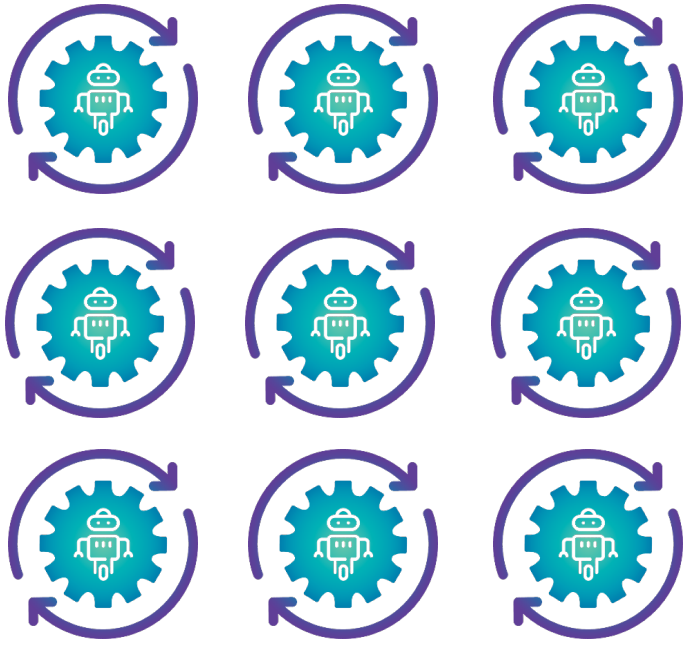
Nov 28, 2018 - Known as **AFLSmart**, this fuzzing software is built on the powerful American ... We're told **AFLSmart** is pretty good at testing applications for common .... The Register - Independent **news** and views for the tech community.

Shonan Meeting 160 (Japan)
Fuzzing & Symbolic Execution

BSIDES MELBOURNE

Collaborative Fuzzing

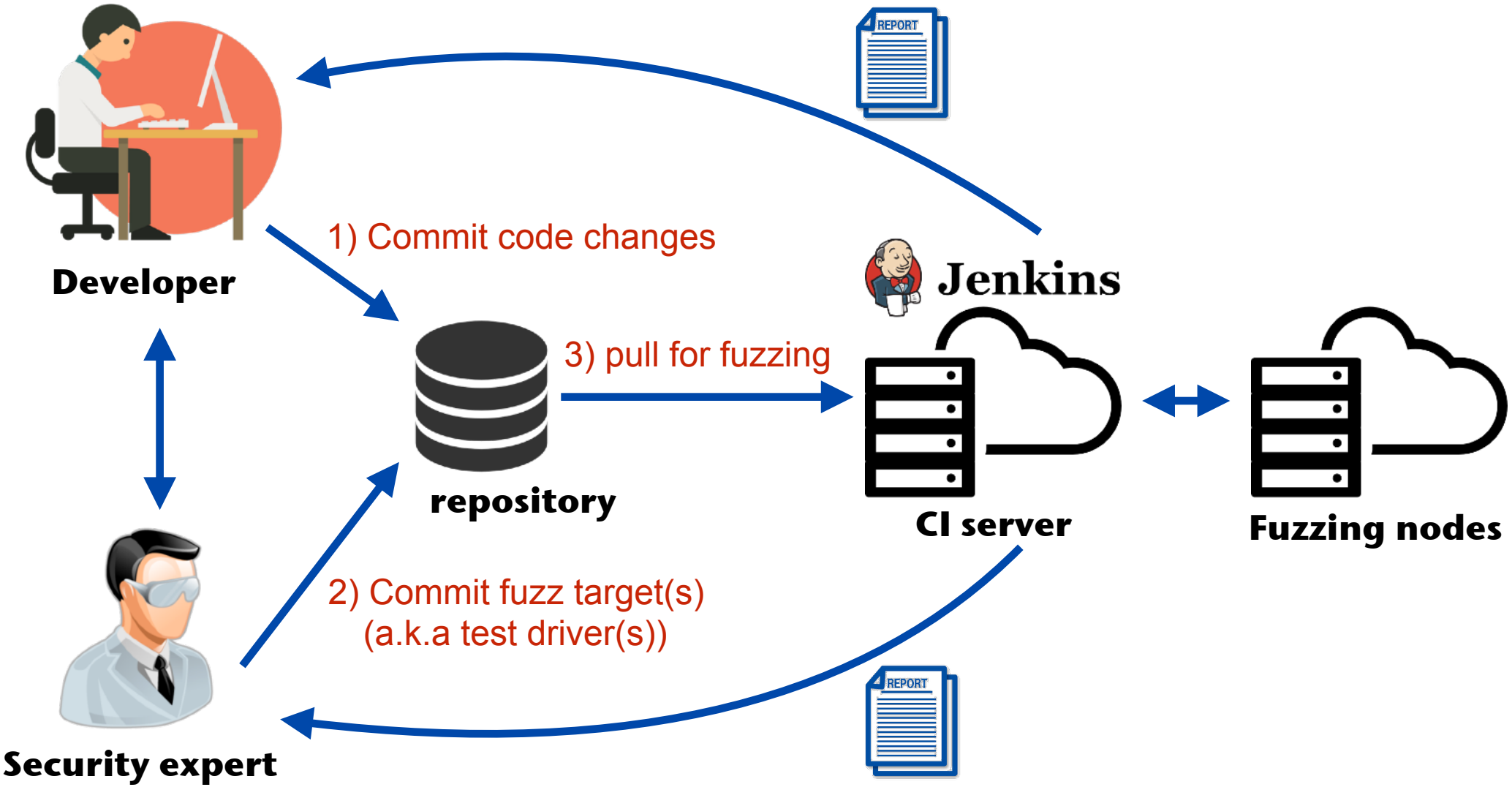




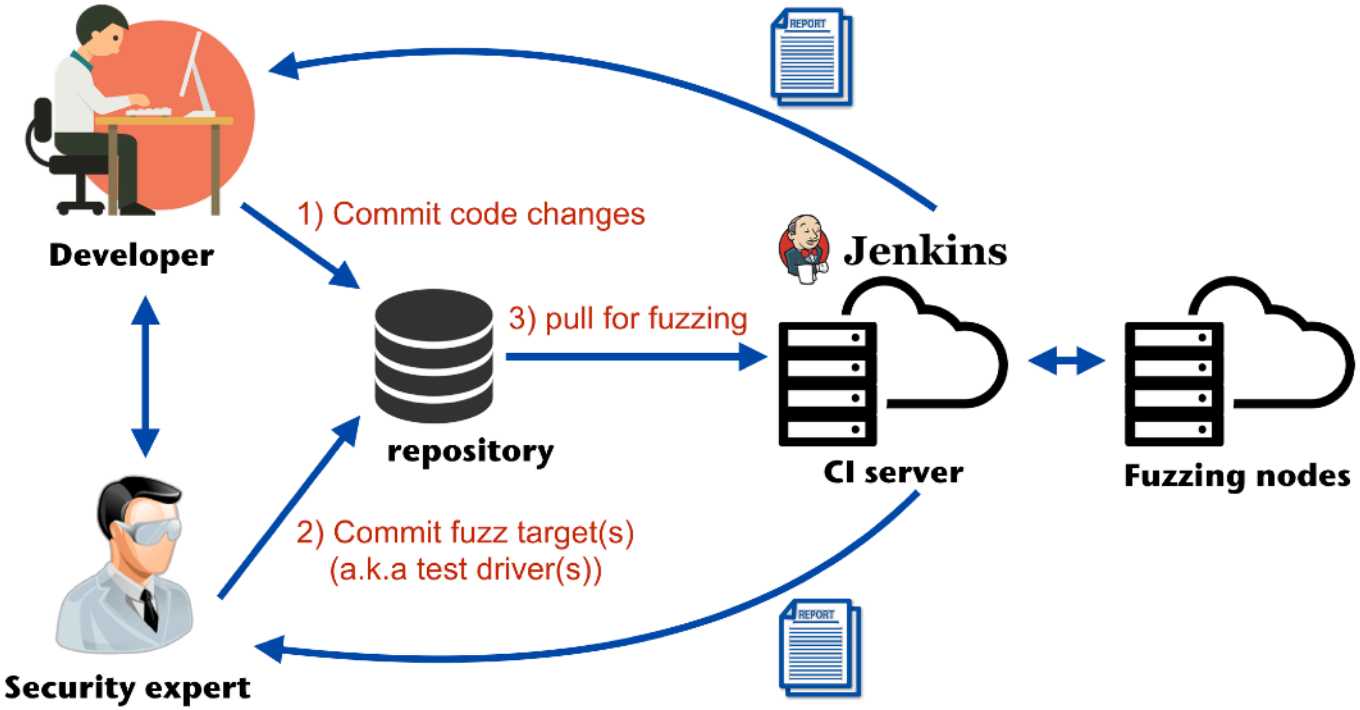
Parallel Fuzzing

Secure Software Development with Fuzzing

In a continuous integration (CI) setup



Problems in this setup



- Shortage of security experts
- Delays caused by the communication between developers & the security experts
- Security expert are not familiar with the code to be fuzzed

Secure Software Development with Fuzzing
At Google

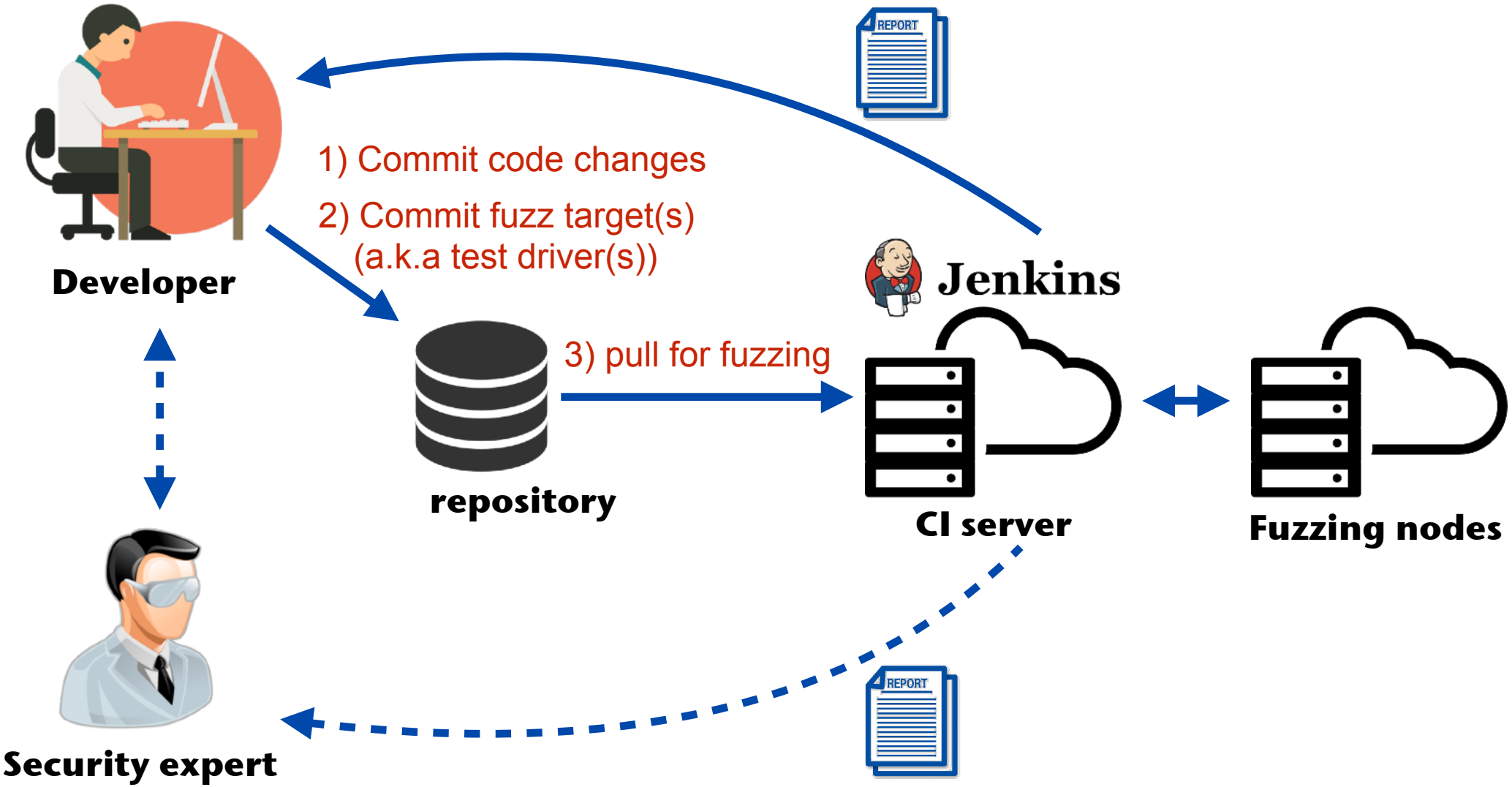
Kostya Serebryany
@kayseesee

Security expert. Creator of AddressSanitizer, MemorySanitizer, ThreadSanitizer, and libFuzzer

"Fuzzing is widely used at Google because **code owners** are writing their own fuzz targets, as opposed to security experts trying to find bugs in code they aren't familiar with." @Shonan meeting on Fuzzing and Symbolic execution (Sept 24-27, 2019, Tokyo, Japan)

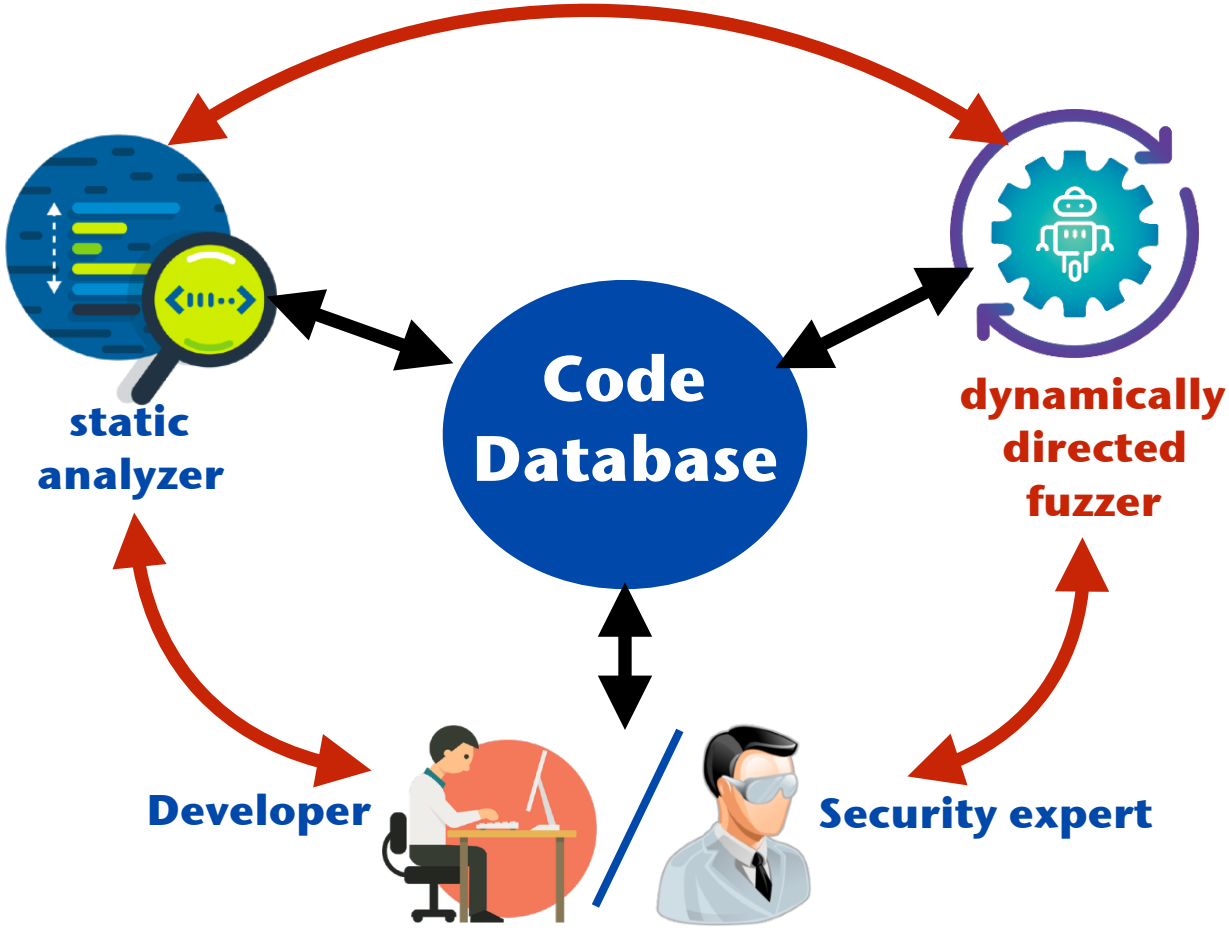
Secure Software Development with Fuzzing

In a continuous integration (CI) setup



Collaborative Directed Fuzzing

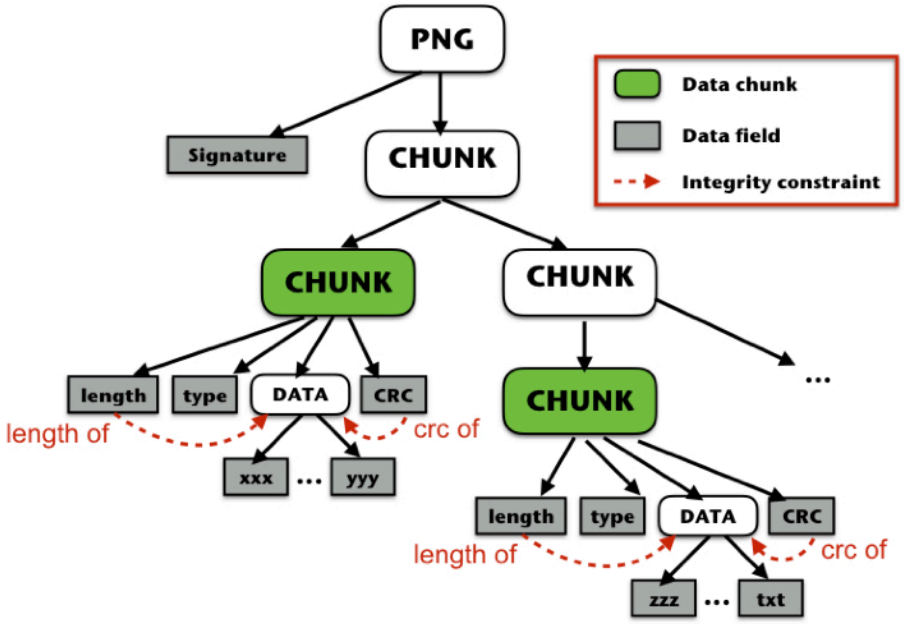
- Code Database as shared knowledge with a unified *queryable interface* (e.g., Github CodeQL)
- Directed Fuzzer accepts *dynamic guidance* & learn to gradually become *self-guided*



Directed Fuzzing

Choose "directions" to manage the search space & discover paths which are more likely to trigger program bugs in shorter time

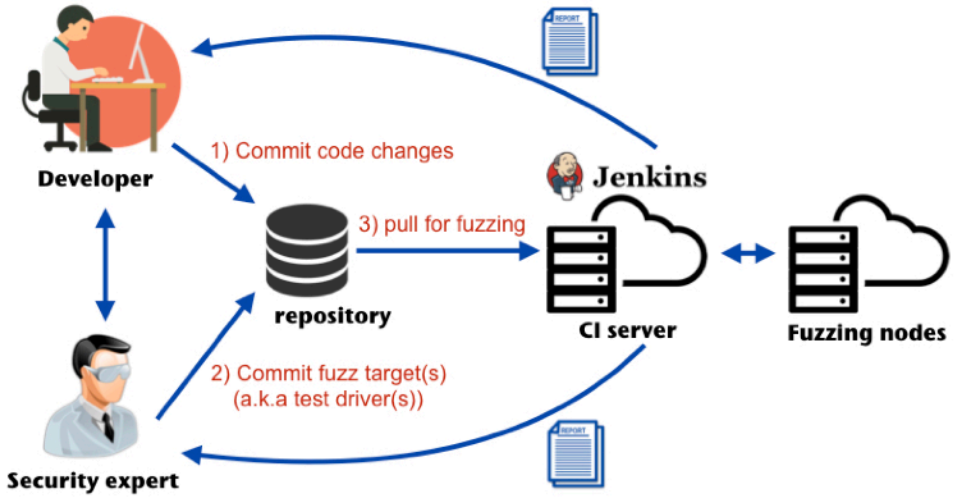
12



23

Secure Software Development with Fuzzing

In a continuous integration (CI) setup

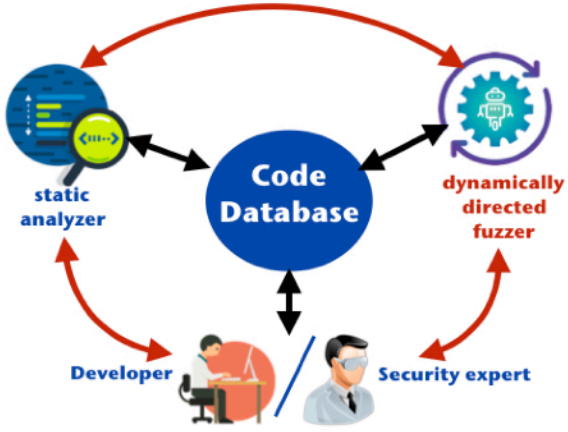


44

Collaborative Directed Fuzzing

-Code Database as shared knowledge with a unified *queryable interface* (e.g., Github CodeQL)

-Directed Fuzzer accepts *dynamic guidance* & learn to gradually become *self-guided*



48