AntiFuzz: Impeding Fuzzing Audits of Binary Executables

Usenix Security 2019, Santa Clara

August 16, 2019

Emre Güler, Cornelius Aschermann, Ali Abbasi, and Thorsten Holz

Chair for Systems Security
Ruhr-Universität Bochum

Motivation

Motivation

Trusted Party

Untrusted Party

Motivation

Trusted Party

- Can find bugs

Untrusted Party

- Can't find bugs

Motivation

Trusted Party

- Can examine code $\implies$ Can find bugs

Untrusted Party

- Can't examine code $\implies$ Can't find bugs ?

Motivation

Trusted Party

- Can examine code $\implies$ Can find bugs

Untrusted Party

- Can't examine code $\implies$ Can't find bugs ?
- But what about automated bug finding tools?

Impeding Fuzzing Audits

Impeding Fuzzing Audits

- Analyze diverse set of fuzzers

Impeding Fuzzing Audits

- Analyze diverse set of fuzzers
- Find assumptions fuzzers need to make

Impeding Fuzzing Audits

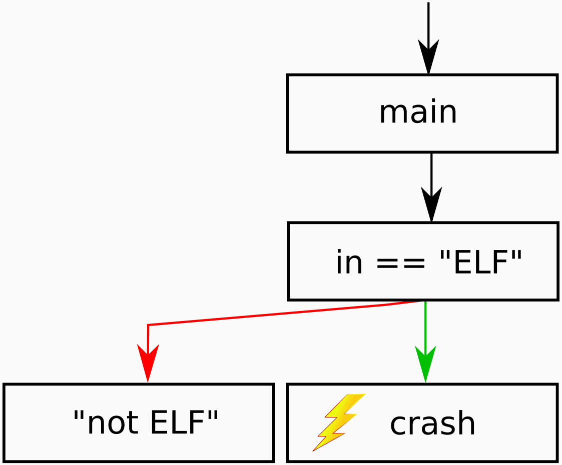
- Analyze diverse set of fuzzers
- Find assumptions fuzzers need to make
- Invalidate those assumptions

Finding Assumptions

Assumptions

	A	B	C	D
● AFL	✓	✓	✓	✗
kAFL	✓	✓	✓	✗
AFLFast	✓	✓	✓	✗
CollAFL	✓	✓	✓	✗
AFLGo	✓	✓	✓	✗
WinAFL	✓	✓	✓	✗
steelix	✓	✓	✓	✗
Redqueen	✓	✓	✓	✗
● Honggfuzz	✓	✓	✓	✗
● VUzzer	✓	✓	✓	✗
● Driller	✓	✓	✓	✓
● Klee	✗	✗	✗	✓
● zzuf	✗	✓	✓	✗
● Peach	✗	✓	✓	✗
● QSym	✓	✓	✓	✓
T-Fuzz	✓	✓	✓	✓
Angora	✓	✓	✓	✗
radamsa	✗	✓	✓	✗
libFuzzer	✓	✓	✓	✗

Demo Application



Assumptions

(A) Coverage Yields Relevant Feedback

Coverage Yields Relevant Feedback

Blind Fuzzer

Coverage-guided Fuzzer

Coverage Yields Relevant Feedback

Blind Fuzzer

- Mutate input

Coverage-guided Fuzzer

- Mutate input

Coverage Yields Relevant Feedback

Blind Fuzzer

- Mutate input
- See if it crashes with given input

Coverage-guided Fuzzer

- Mutate input
- See if it crashes with given input

Coverage Yields Relevant Feedback

Blind Fuzzer

- Mutate input
- See if it crashes with given input

Coverage-guided Fuzzer

- Mutate input
- See if it crashes with given input
- … or if new coverage is found

Coverage Yields Relevant Feedback

Blind Fuzzer

- Mutate input
- See if it crashes with given input

Coverage-guided Fuzzer

- Mutate input
- See if it crashes with given input
- … or if new coverage is found
- If so, add input to queue

Coverage Yields Relevant Feedback

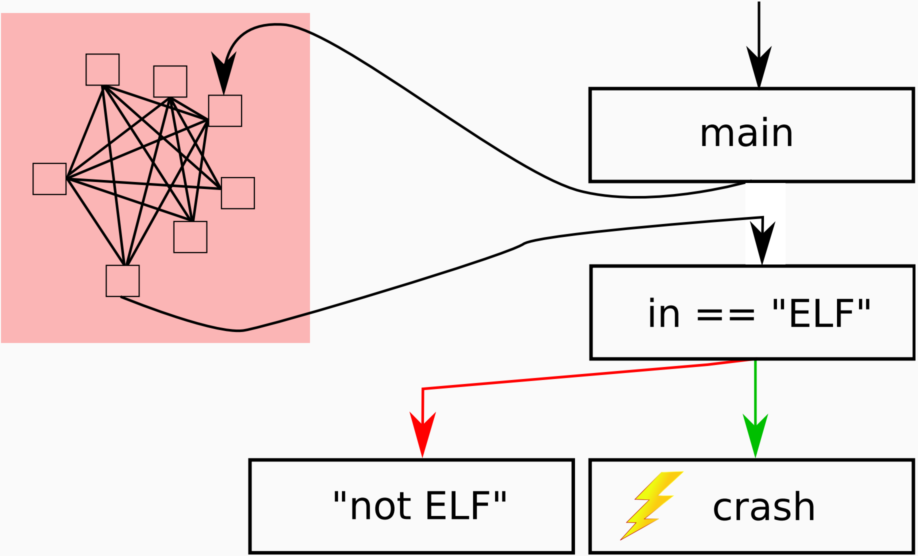
Blind Fuzzer

- Mutate input
- See if it crashes with given input

Coverage-guided Fuzzer

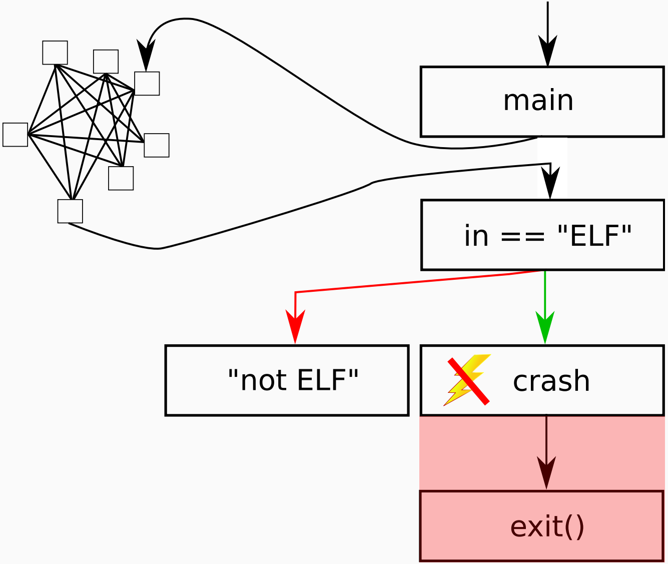
- Mutate input
- See if it crashes with given input
- … or if new coverage is found
- If so, add input to queue
- New Coverage $\implies$ New Behavior

Coverage Yields Relevant Feedback



(B) Crashes Can Be Detected

Crashes Can Be Detected



(C) Many Executions Per Second

Many Executions Per Second

Why is AFL so good?

Many Executions Per Second

Why is AFL so good?

- No ”human knowledge” about target necessary

Many Executions Per Second

Why is AFL so good?

- No ”human knowledge” about target necessary
- Super fast implementation (thousands of executions per second)

Many Executions Per Second

Why is AFL so good?

- No ”human knowledge” about target necessary
- Super fast implementation (thousands of executions per second)
- $\rightarrow$ As long as we are fast, we don’t need to be smart.

Many Executions Per Second

Bad approach

Many Executions Per Second

Bad approach

- Slow down application

Many Executions Per Second

Bad approach

- Slow down application
- But: real-world usage also slows down

Many Executions Per Second

Bad approach

- Slow down application
- But: real-world usage also slows down

Better approach

Many Executions Per Second

Bad approach

- Slow down application
- But: real-world usage also slows down

Better approach

- Slow down application ...

Many Executions Per Second

Bad approach

- Slow down application
- But: real-world usage also slows down

Better approach

- Slow down application …
- … only when it's being fuzzed?

Many Executions Per Second

What is the implication of being fuzzed?

Many Executions Per Second

What is the implication of being fuzzed?

- Most inputs will be malformed

Many Executions Per Second

What is the implication of being fuzzed?

- Most inputs will be malformed
- But in real-world scenarios, most inputs are well-formed

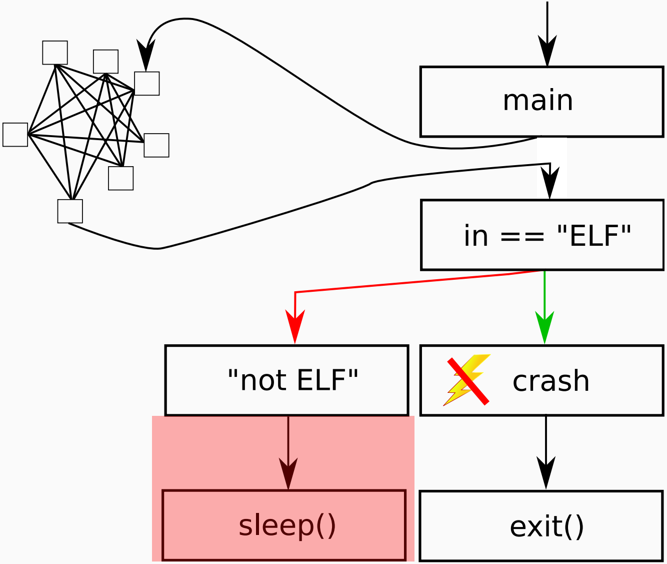
Many Executions Per Second

What is the implication of being fuzzed?

- Most inputs will be malformed
- But in real-world scenarios, most inputs are well-formed

Solution: slow down application if input is malformed

Many Executions Per Second



(D) Constraints Are Solvable with Symbolic Execution

Constraints Are Solvable with Symbolic Execution

Why use symbolic execution?

Constraints Are Solvable with Symbolic Execution

Why use symbolic execution?

- Some constraints are too hard to solve via random mutations

Constraints Are Solvable with Symbolic Execution

Why use symbolic execution?

- Some constraints are too hard to solve via random mutations
- Let’s get help from symbolic execution

Constraints Are Solvable with Symbolic Execution

Why use symbolic execution?

- Some constraints are too hard to solve via random mutations
- Let's get help from symbolic execution

Assumption: some constraints are too hard to be solved by random mutations alone, but could be solved by symbolic execution

Constraints Are Solvable with Symbolic Execution

How to break this assumption? Two techniques:

Constraints Are Solvable with Symbolic Execution

How to break this assumption? Two techniques:

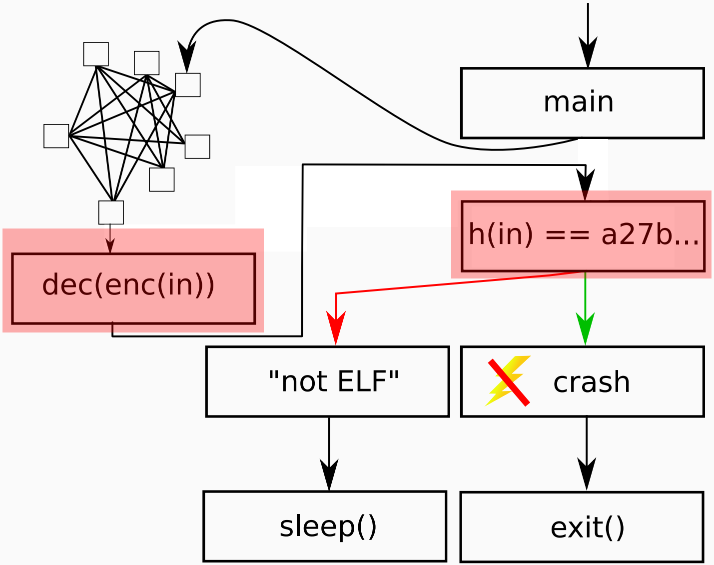
- Replace constants comparisons by hash comparisons

Constraints Are Solvable with Symbolic Execution

How to break this assumption? Two techniques:

- Replace constants comparisons by hash comparisons
- Put input through encryption and decryption before using

Constraints Are Solvable with Symbolic Execution



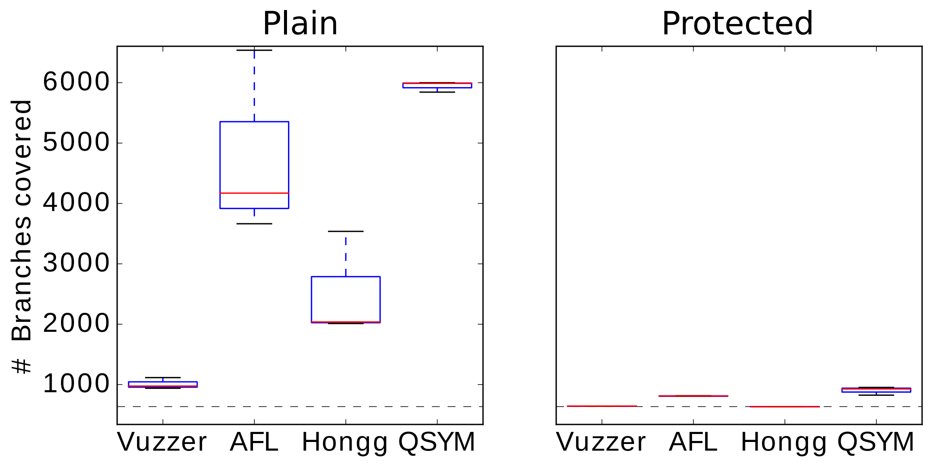
Evaluation

Dummy Application

	None	Coverage	Crash	Speed	Symbolic Exec.	All
AFL	✗	✓	✓	✓	✗	✓
Honggfuzz	✗	✓	✓	✓	✗	✓
Vuzzer	✗	✓	✓	✓	✓	✓
Driller	✗	✓	-	-	✗	✓
Klee	✗	✓	✓	✓	✓	✓[a]
zzuf	✗	✗	✓	✗	✗	✓
Peach	✗	✗	✓	✓	✗	✓
QSYM	✗	✓	✓	✓	✗	✓

✓ = No crashes were found

objdump



Plain
Protected
Branches covered
6000
5000
4000
3000
2000
1000
Vuzzer
AFL
Hongg
QSYM
Vuzzer
AFL
Hongg
QSYM

Conclusion

- Systematic analysis reveals: contemporary fuzzers rely on four core assumptions

Conclusion

- Systematic analysis reveals: contemporary fuzzers rely on four core assumptions
  - Coverage Yields Relevant Feedback
  - Crashes Can Be Detected
  - Many Executions Per Second
  - Constraints Are Solvable With Symbolic Execution

Conclusion

- Systematic analysis reveals: contemporary fuzzers rely on four core assumptions
  - Coverage Yields Relevant Feedback
  - Crashes Can Be Detected
  - Many Executions Per Second
  - Constraints Are Solvable With Symbolic Execution
- `AntiFuzz` breaks these assumptions to impede fuzzing attempts

Conclusion

- Systematic analysis reveals: contemporary fuzzers rely on four core assumptions
  - Coverage Yields Relevant Feedback
  - Crashes Can Be Detected
  - Many Executions Per Second
  - Constraints Are Solvable With Symbolic Execution
- `AntiFuzz` breaks these assumptions to impede fuzzing attempts
- `https://github.com/RUB-SysSec/antifuzz`

Thank You

Q & A