DifFuzz: Differential Fuzzing for Side-Channel Analysis

Shirin Nilizadeh — UNIVERSITY OF TEXAS ARLINGTON

Yannic Noller — HUMBOLDT-UNIVERSITÄT ZU BERLIN

Corina S. Pasareanu — Carnegie Mellon University, NASA Ames Research Center

Side-Channel Analysis

- leakage of **secret** information
- **software** side-channels
- **observables**:
  - execution time,
  - memory consumption,
  - response size,
  - …

Example: Side-Channel Vulnerability

```
0  boolean pwcheck_unsafe (byte[] pub, byte[] sec) {
1      if (pub.length != sec.length) {
2          return false;
3      }
4      for (int i = 0; i < pub.length; i++) {
5          if (pub[i] != sec[i]) {
6              return false;
7          }
8      }
9      return true;
10 }
```

Unsafe Password Checking

Example: Side-Channel Vulnerability

```
0  boolean pwcheck_unsafe (byte[] pub, byte[] sec) {
1      if (pub.length != sec.length) {
2          return false;
3      }
4      for (int i = 0; i < pub.length; i++) {
5          if (pub[i] != sec[i]) {
6              return false;
7          }
8      }
9      return true;
10 }
```

Unsafe Password Checking

Example: Side-Channel Vulnerability

```
0  boolean pwcheck_unsafe (byte[] pub, byte[] sec) {
1      if (pub.length != sec.length) {
2          return false;
3      }
4      for (int i = 0; i < pub.length; i++) {
5          if (pub[i] != sec[i]) {
6              return false;
7          }
8      }
9      return true;
10 }
```

Unsafe Password Checking

Side-Channel Analysis

- *secure* if the secret data can not be inferred by an attacker through their observations of the system (aka *non-interference*)
- can be solved by self-composition [Barthe2004]

Non-Interference by Self-Composition

[Barthe2004]

public value **secret value**

program execution $P[pub, sec_1]$

cost observation $c(P[pub, sec_1])$

two secret values

$c(P[pub, sec_1])$ $c(P[pub, sec_2])$

equivalence $c(P[pub, sec_1]) = c(P[pub, sec_2])$

$$\forall pub, sec_1, sec_2 : c(P[pub, sec_1]) = c(P[pub, sec_2])$$

Side-Channel Analysis

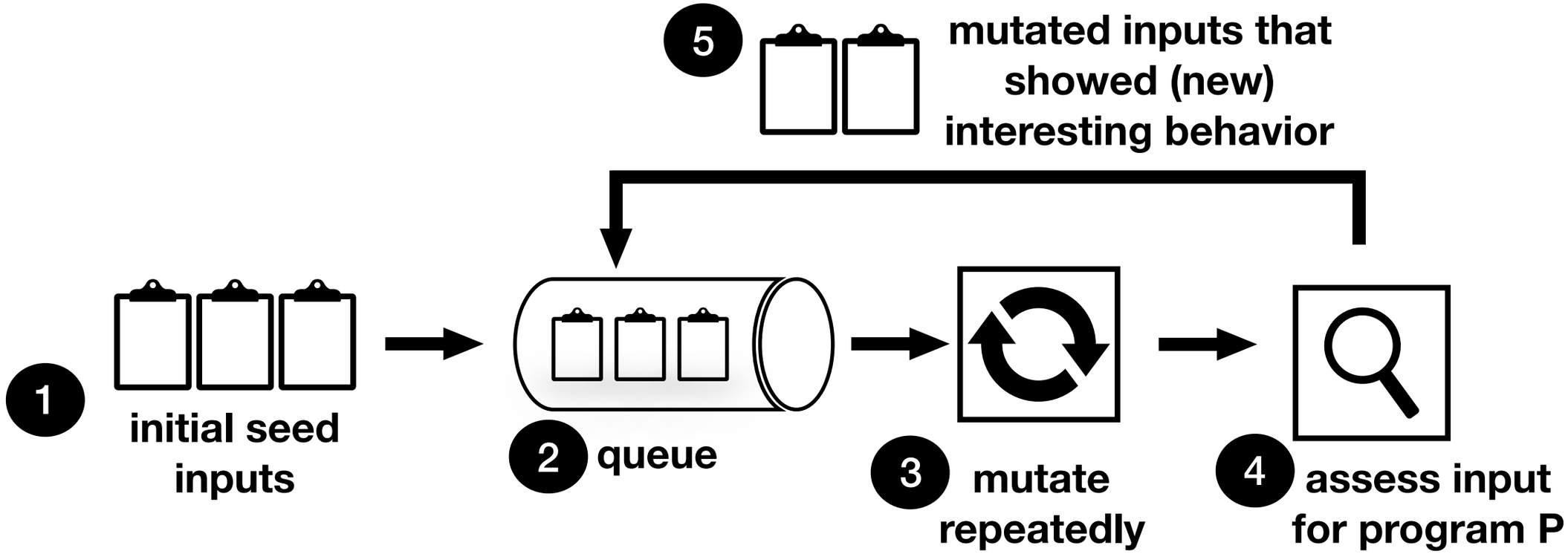
- *secure* if the secret data can not be inferred by an attacker through their observations of the system (aka *non-interference*)
- can be solved by self-composition [Barthe2004]

$$\forall pub, sec_1, sec_2 : c(P[pub, sec_1]) = c(P[pub, sec_2])$$

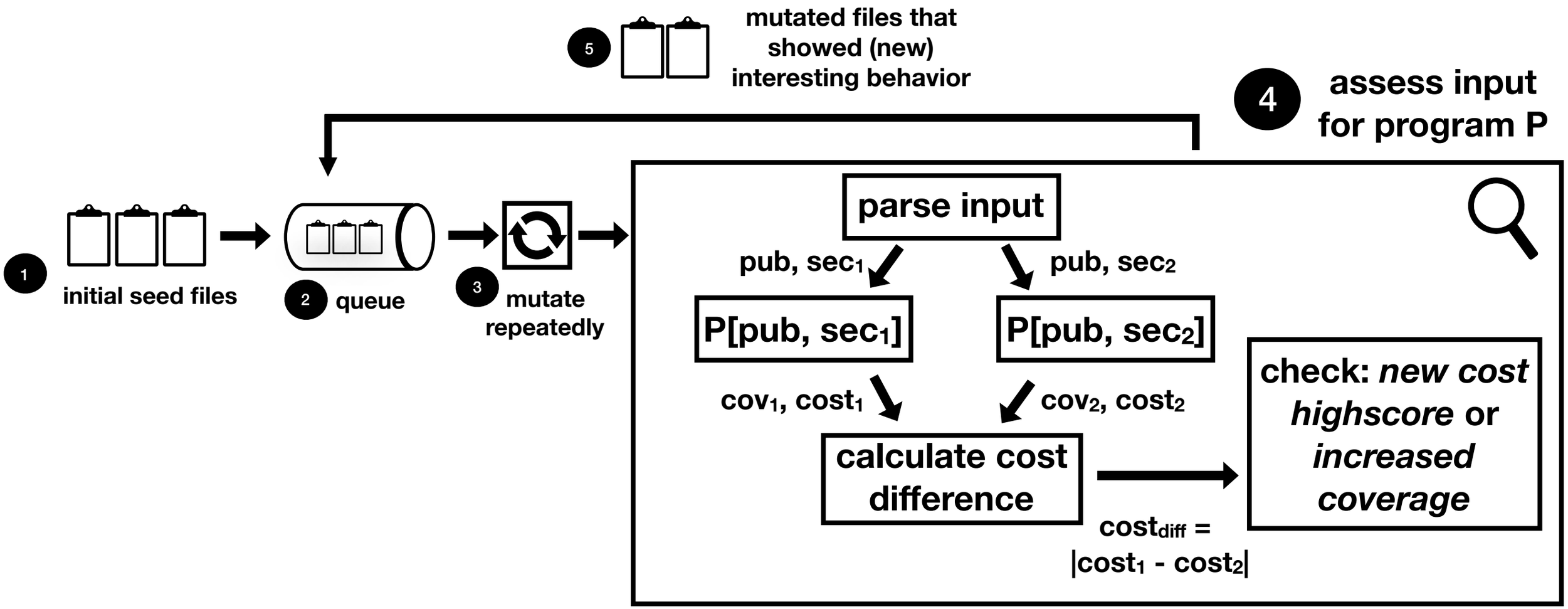
- ε-bounded non-interference [Chen2017]

$$\forall pub, sec_1, sec_2 : |c(P[pub, sec_1]) - c(P[pub, sec_2])| < \epsilon$$

Differential Fuzzing for Side-Channel Analysis



Input Assessment to find Side-Channel vulnerabilities



Side-Channel Analysis

- can be solved by self-composition [Barthe2004]

$$\forall pub, sec_1, sec_2 : c(P[pub, sec_1]) = c(P[pub, sec_2])$$

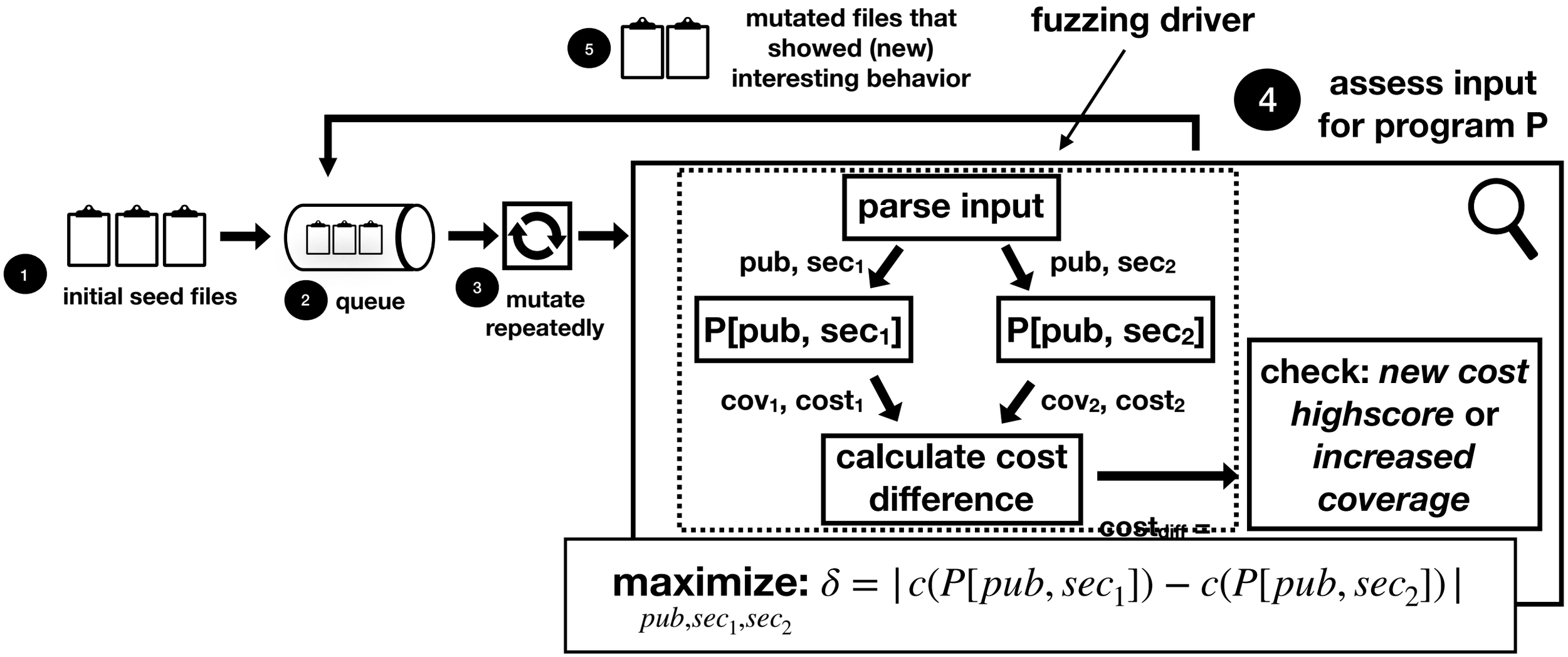
- ε-bounded non-interference [Chen2017]

$$\forall pub, sec_1, sec_2 : |c(P[pub, sec_1]) - c(P[pub, sec_2])| < \epsilon$$

- differential fuzzing for side-channel analysis:

$$\textbf{maximize:} \max_{pub, sec_1, sec_2} \delta = |c(P[pub, sec_1]) - c(P[pub, sec_2])|$$

Differential Fuzzing for Side-Channel Analysis



Differential Fuzzing Driver

```
1:  pub, sec1, sec2 ← parse(input, constraints)
2:  cost1 ← measure(P(pub,sec1))
3:  cost2 ← measure(P(pub,sec2))
4:  costDiff ← |cost1 - cost2|
5:  setUserDefinedCost(costDiff)
```

Example

```
0  boolean pwcheck_unsafe (byte[] pub, byte[] sec) {
1      if (pub.length != sec.length) {
2          return false;
3      }
4      for (int i = 0; i < pub.length; i++) {
5          if (pub[i] != sec[i]) {
6              return false;
7          }
8      }
9      return true;
10 }
```

Unsafe Password Checking

timing side-channel: measured by number of instructions executed

Example Results

Initial Input: $cost_{Diff} = 0$

```
secret₁ = [72, 101, 108, 108, 111,  32,  67]
secret₂ = [97, 114, 110, 101, 103, 105, 101]
public  = [32,  77, 101, 108, 108, 111, 110]
```

$cost_{Diff} > 0$ after ~ 5 sec

Input with highscore $cost_{Diff} = 47$ after ~ 69 sec (maximum length = 16 bytes):

```
secret₁ = [72, 77, -16, -66, -48, -48, -48, -48, -28, 0, 100, 0, 0, 0, 0, -48]
secret₂ = [-48, -4, -48, 7, 17, 0, -24, -48, -48, 16, -48, -3, 108, 72, 32, 0]
public  = [-48, -4, -48, 7, 17, 0, -24, -48, -48, 16, -48, -3, 108, 72, 32, 0]
```

Experiments

- build on top of AFL [AFL, Kersten2017, Noller2018]
- **Blazer** [Antonopoulos2017]
- **Themis** [Chen2017]
- and more projects from **GitHub** and **STAC** [DARPA2018]
- runtime: 30min

RQ1: Effectiveness

Blazer

Benchmark	Subject	Version	Average δ	Std. Error	Maximum
MicroBench	Array	Safe	1.00	0.00	1
		Unsafe	192.00	2.68	195
	LoopAndbranch	*Safe*	*1,468,212,312.40*	*719,375,479.77*	*4,278,268,7*
		Unsafe	4,283,404,852.40	4,450,278.15	4,294,838,7
	Sanity	Safe	0.00	0.00	0
		Unsafe	4,213,237,198.00	60,857,888.00	4,290,510,8
	Straightline	Safe	0.00	0.00	0
		Unsafe	8.00	0.00	8
	unixlogin	Safe	3.00	0.00	3
		Unsafe	2,880,000,008.00	286,216,701.00	3,200,000,0
STAC	modPow1	Safe	0.00	0.00	0
		Unsafe	2,576.00	168.21	3,068
	modPow2	Safe	0.00	0.00	0
		Unsafe	1,471.00	891.00	5,206
	passwordEq	Safe	0.00	0.00	0
		Unsafe	86.40	20.31	127
Literature	k96	Safe	0.00	0.00	0
		Unsafe	338.00	185.13	3,087,339
	gpt14	*Safe*	*163.20*	*79.84*	*517*
		Unsafe	6,673,760.00	2,211,811.00	12,965,890
	login	Safe	0.00	0.00	0
		Unsafe	62.00	0.00	62

RQ1: Effectiveness

Benchmark	Version	DiffFuzz			Themis	
		Average δ	Std. Error	Maximum	ϵ = 64	ϵ = 0
Spring-Security	Safe	1.00	0.00	1	✓	✓
	Unsafe	149.00	0.00	149	✓	✓
JDK-MsgDigest	Safe	1.00	0.00	1	✓	✓
	Unsafe	10.215.00	6.120.00	34.479	✓	✓
Picketbox	Safe	1.00	0.00	1	✓	X
	Unsafe	4.954.00	1.295	8.794	✓	✓
Tomcat	Safe	12.20	1.61	14	✓	X
	Unsafe	*33.20*	*3.40*	*37*	✓	✓
Jetty	*Safe*	*5454.00*	*1330.88*	*8898*	✓	✓
	Unsafe	10786.60	2807.51	16020	✓	✓
oriented	Safe	6.00	0.00	6	✓	X
	Unsafe	6.604.00	3.681	19.300	✓	✓
pac4j	Safe	10.00	0.00	10	✓	X
	Unsafe	*11.00*	*0.00*	*11*	✓	✓
	Unsafe*	39.00	0.00	39	-	-
boot-auth	Safe	5.00	0.00	5	✓	X
	Unsafe	101.00	0.00	101	✓	✓
tourPlanner	Safe	0.00	0.00	0	✓	✓
	Unsafe	522.40	18.60	576	✓	✓
DynaTable	Unsafe	95.80	0.44	97	✓	✓
Advanced_table	Unsafe	92.40	1.54	97	✓	✓
OpenMRS	Unsafe	206.00	0.00	206	✓	✓
OACC	*Unsafe*	*47.00*	*0.00*	*47*	✓	✓

RQ1: Effectiveness

Benchmark	Subject	Version	Average δ	Std. Error	Maximum
STAC	CRIME	Unsafe	295.40	117.05	782
	ibasys	Unsafe	191.00	20.88	262
Zero-day Vulnerabilities	Apache ftpserver Clear	Unsafe	47.00	0.00	1
	Apache ftpserver MD5	Unsafe	151.00	0.00	151
	Apache ftpserver SaltedPW	Unsafe	178.80	5.13	193
	Apache ftpserver StringUtils	Unsafe	53.00	0.00	53
	AuthmeReloaded	Unsafe	383.00	0.00	383

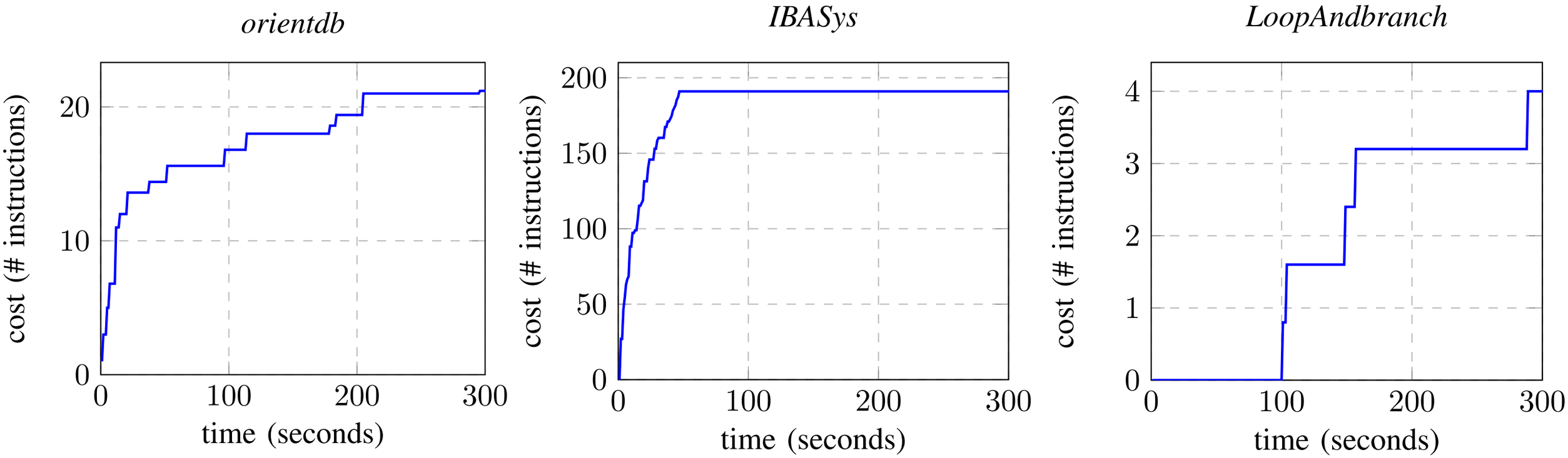
RQ2: Analysis Time

Benchmark	Subject	Version	Time (sec) DifFuzz δ > 0	Blazer	Themis
MicroBench	Array	Safe	7.40 (+/- 1.21)	1.60	0.28
		Unsafe	7.40 (+/- 0.93)	0.16	0.23
	LoopAndbranch	Safe	18.60 (+/- 6.40)	0.23	0.33
		Unsafe	10.60 (+/- 2.62)	0.65	0.16
	Sanity	Safe	-	0.63	0.41
		Unsafe	163 (+/- 40.63)	0.30	0.17
	Straightline	Safe	-	0.21	0.49
		Unsafe	14.60 (+/- 6.53)	22.20	5.30
	unixlogin	Safe	510.00 (+/- 91.18)	0.86	-
		Unsafe	464.20 (+/- 64.61)	0.77	-
STAC	modPow1	Safe	-	1.47	0.61
		Unsafe	*4.80 (+/- 1.11)*	218.54	14.16
	modPow2	Safe	-	1.62	0.75
		Unsafe	*23.00 (+/- 3.48)*	7813.68	141.36
	passwordEq	Safe	-	2.70	1.10
		Unsafe	8.60 (+/-2.11)	1.30	0.39
Literature	k96	Safe	-	0.70	0.61
		Unsafe	3.40 (+/- 0.98)	1.29	0.54
	gpt14	Safe	4.20 (+/- 0.80)	1.43	0.46
		Unsafe	*4.40 (+/- 1.03)*	219.30	1.25
	login	Safe	-	1.77	0.54
		Unsafe	10.00 (+/- 2.92)	1.79	0.70

RQ2: Analysis Time

Benchmark	Version	Time (sec) DifFuzz δ > 0	Time (sec) Themis
Spring-Security	Safe	9.00 (+/- 1.26)	1.70
	Unsafe	8.80 (+/- 1.16)	1.09
JDK-MsgDigest	Safe	15.80 (+/- 3.93)	1.27
	Unsafe	7.40 (+/- 1.29)	1.33
Picketbox	Safe	29.20 (+/- 5.00)	1.79
	Unsafe	16.80 (+/- 2.58)	1.79
Tomcat	Safe	13.80 (+/- 1.29)	9.93
	Unsafe	128.60 (+/- 87.20)	8.64
Jetty	Safe	9.40 (+/- 1.86)	2.50
	Unsafe	7.00 (+/- 1.05)	2.07
oriented	Safe	*3.20 (+/- 0.97)*	37.99
	Unsafe	*3.00 (+/- 0.84)*	38.09
pac4j	Safe	5.00 (+/- 1.22)	3.97
	Unsafe	8.00 (+/- 2.76)	1.85
	Unsafe*	10.80 (+/- 5.80)	-
boot-auth	Safe	*5.20 (+/- 0.20)*	9.12
	Unsafe	*5.20 (+/- 0.20)*	8.31
tourPlanner	Safe	-	22.22
	Unsafe	*19.20 (+/- 0.80)*	22.01
DynaTable	Unsafe	3.60 (+/- 1.21)	1.165
Advanced_table	Unsafe	11.20 (+/- 1.62)	2.01
OpenMRS	Unsafe	11.60 (+/- 3.22)	9.71
OACC	Unsafe	7.00 (+/- 1.30)	1.83

RQ2: Analysis Time



DifFuzz: Differential Fuzzing for Side-Channel Analysis

Example: Side-Channel Vulnerability

```
0  boolean pwcheck_unsafe (byte[] pub, byte[] sec) {
1      if (pub.length != sec.length) {
2          return false;
3      }
4      for (int i = 0; i < pub.length; i++) {
5          if (pub[i] != sec[i]) {
6              return false;
7          }
8      }
9      return true;
10 }
```

Unsafe Password Checking

Non-Interference by Self-Composition

[Barthe2004]

public value, secret value

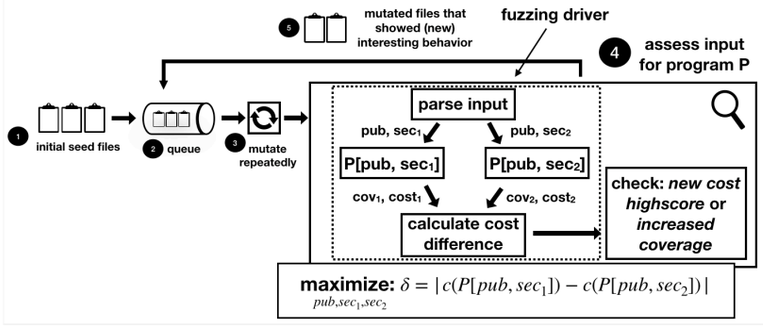
program execution: $P[pub, sec_1]$

cost observation: $c(P[pub, sec_1])$ — two secret values: $c(P[pub, sec_1])$, $c(P[pub, sec_2])$

equivalence: $c(P[pub, sec_1]) = c(P[pub, sec_2])$

$$\forall pub, sec_1, sec_2 : c(P[pub, sec_1]) = c(P[pub, sec_2])$$

Differential Fuzzing for Side-Channel Analysis



RQ1: Effectiveness

Benchmark	Version	DifFuzz Average δ	DifFuzz Std. Error	DifFuzz Maximum	Themis ε = 64	Themis ε = 0
Spring-Security	Safe	1.00	0.00	1	✓	✓
	Unsafe	149.00	0.00	149	✓	✓
JDK-MsgDigest	Safe	1.00	0.00	1	✓	✓
	Unsafe	10,215.00	6,120.00	34,479	✓	✓
Picketbox	Safe	1.00	0.00	1	✓	X
	Unsafe	4,954.00	1,295	8,794	✓	✓
Tomcat	Safe	12.20	1.61	14	✓	X
	Unsafe	33.20	3.40	37	✓	✓
Jetty	Safe	5454.00	1330.88	8898	✓	✓
	Unsafe	10786.60	2807.51	16020	✓	✓
oriented	Safe	6.00	0.00	6	✓	X
	Unsafe	6,604.00	3,681	19,300	✓	✓
pac4j	Safe	10.00	0.00	10	✓	X
	Unsafe	11.00	0.00	11	✓	✓
	Unsafe*	39.00	0.00	39	-	-
boot-auth	Safe	5.00	0.00	5	✓	X
	Unsafe	101.00	0.00	101	✓	✓
tourPlanner	Safe	0.00	0.00	0	✓	✓
	Unsafe	522.40	18.60	576	✓	✓
DynaTable	Unsafe	95.80	0.44	97	✓	✓
Advanced_table	Unsafe	92.40	1.54	97	✓	✓
OpenMRS	Unsafe	206.00	0.00	206	✓	✓
OACC	Unsafe	47.00	0.00	47	✓	✓

`git clone https://github.com/isstac/diffuzz.git`

References

[AFL] Website. american fuzzy lop (AFL). http://lcamtuf.coredump.cx/afl/.

[Antonopoulos2017] Timos Antonopoulos, Paul Gazzillo, Michael Hicks, Eric Koskinen, Tachio Terauchi, and Shiyi Wei. 2017. Decomposition instead of self-composition for proving the absence of timing channels. In Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2017). ACM, New York, NY, USA, 362-375.

[Barthe2004] G. Barthe, P. R. D'Argenio and T. Rezk, "Secure Information Flow by Self-Composition," *Computer Security Foundations Workshop, IEEE(CSFW)*, Pacific Grove, California, 2004, pp. 100.

[Chen2017] Jia Chen, Yu Feng, and Isil Dillig. 2017. Precise Detection of Side-Channel Vulnerabilities using Quantitative Cartesian Hoare Logic. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). ACM, New York, NY, USA, 875-890.

[DARPA2018] Mr. Dustin Fraze. Space/Time Analysis for Cybersecurity (STAC). https: //www.darpa.mil/program/space-time-analysis-for-cybersecurity. Accessed: 2018-08-21.

[Kersten2017] Rody Kersten, Kasper Luckow, and Corina S. Păsăreanu. 2017. POSTER: AFL-based Fuzzing for Java with Kelinci. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17).

[Noller2018] Yannic Noller, Rody Kersten, and Corina S. Păsăreanu. 2018. Badger: Complexity Analysis with Fuzzing and Symbolic Execution. In Proceedings of 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'18). ACM, New York, NY, USA.

END OF DOCUMENT