Systematic Evaluation of Automated Tools for Side-Channel Vulnerabilities Detection

Antoine Geimer, Mathéo Vergnolle, Frédéric Recoules, Lesly-Ann Daniel, Sébastien Bardin, Clémentine Maurice

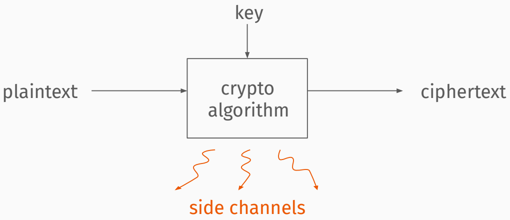
November 23, 2023

Context

Side-channels everywhere

Definition

Side-channels are side-effects in a program's execution that can leak information

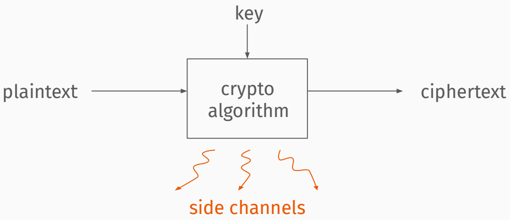


Side-channels everywhere

Definition

Side-channels are side-effects in a program's execution that can leak information

- Focus on *microarchitectural side-channels*: execution time, cache access patterns, port congestions, etc
- Requires running on the same hardware as the victim's program
- Important implications in the era of cloud computing



Constant-time programming

- Root hardware cause unlikely to be fixed

Constant-time programming

- Root hardware cause unlikely to be fixed
- Countermeasure: constant-time programming
- Ensures the microarchitectural state independent of secret values

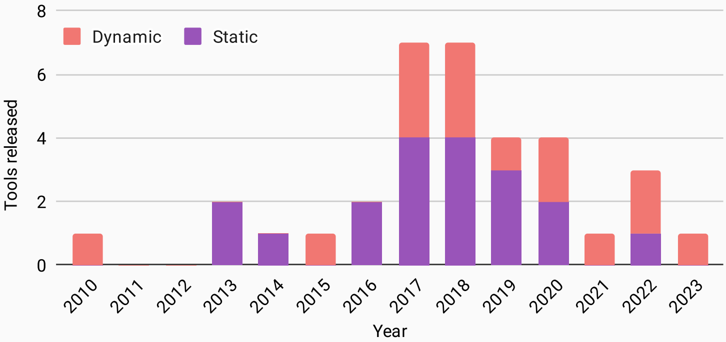
Constant-time programming

- Root hardware cause unlikely to be fixed
- Countermeasure: constant-time programming
- Ensures the microarchitectural state independent of secret values
- Hard to implement in practice → tools to check this property automatically

Definition

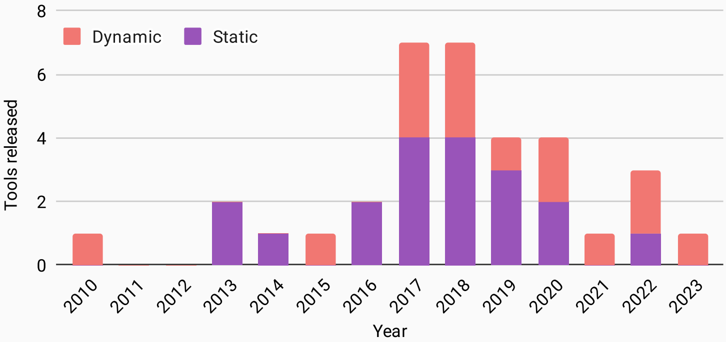
Vulnerability: any branch or memory access that depends on a secret!

Research questions



Paradox: many CT bug-finding tools proposed...

Research questions



Paradox: many CT bug-finding tools proposed...

...yet most vulnerabilities remain manually found!

Contributions

Research questions:

- RQ1 How can we compare these frameworks?
- RQ2 Could an existing framework have found these vulnerabilities?
- RQ3 What features might be missing from existing frameworks?

Contributions

Research questions:

RQ1 How can we compare these frameworks?

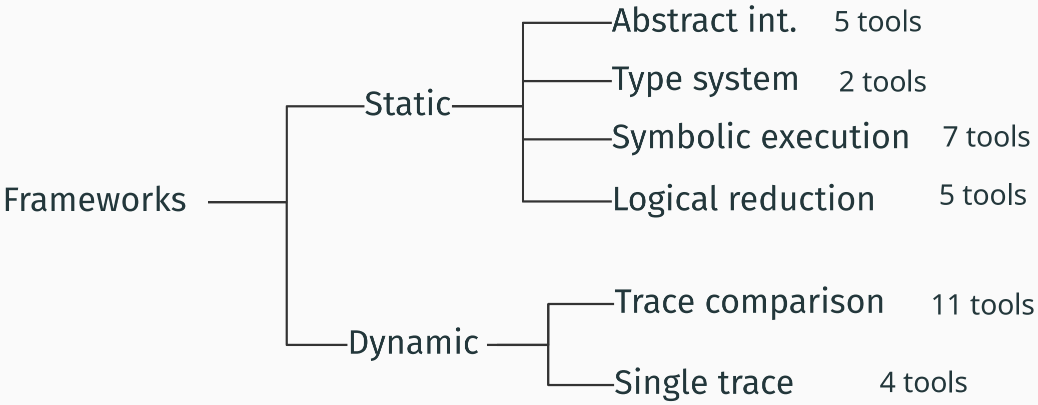
RQ2 Could an existing framework have found these vulnerabilities?

RQ3 What features might be missing from existing frameworks?

Contributions:

- (RQ1) Multi-criteria classification of existing tools
- (RQ2) State-of-the-art of recent vulnerabilities
- (RQ1 and RQ3) Unified benchmark of 5 different tools
- (RQ2 and RQ3) Case-study of vulnerabilities from 3 publications

Classification: detection tools



- 34 different approaches: 15 dynamic, 19 static
- Broad classification by methods used

Classfication criterion

Input

Type of input program supported: binary, source code, LLVM, etc

Output

Type of information outputed by the analysis: leakage site, estimation, witness

Policy

Property checked by the analysis: constant-time, cache-oblivious, constant-resource

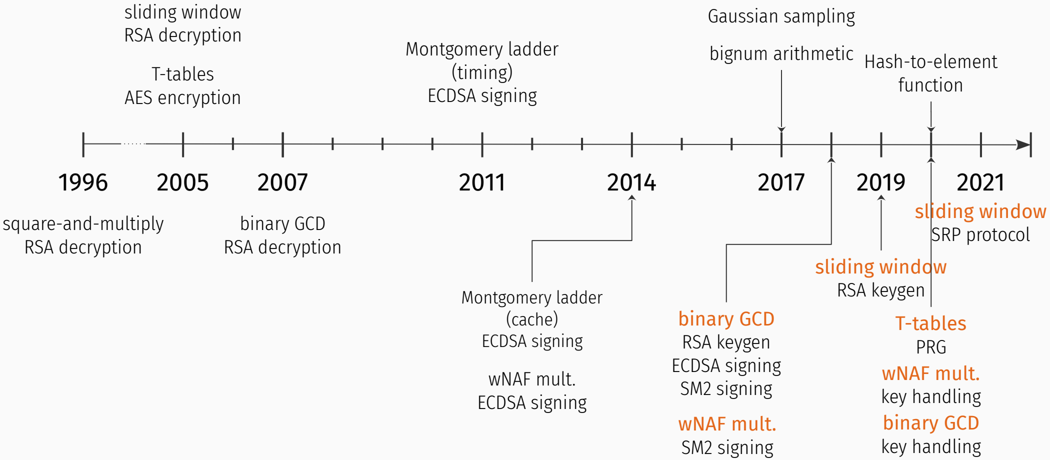
Scalability

How well the analysis scales: from simple toy programs to large cryptographic algorithms

Other criterion: blinding support, soundness, availability

Classification: recent vulnerabilities

We compare recent vulnerabilities (post-2017) with past vulnerabilities.



Recent vulnerabilities

New vulnerabilities:

- Arithmetic functions
- Hash-to-element functions
- Functions from new cryptography

New contexts:

- Key generation
- Key parsing and handling
- Random number generation

[1] García "Side-Channel Analysis and Cryptography Engineering: Getting OpenSSL Closer to Constant-Time (Manuscript)" (University of Tampere 2022)

Recent vulnerabilities

New vulnerabilities:

- Arithmetic functions
- Hash-to-element functions
- Functions from new cryptography

New contexts:

- Key generation
- Key parsing and handling
- Random number generation

Cryptographic primitives themselves are now generally safe... but not always correctly used. Example: OpenSSL's `BN_FLG_CONSTTIME` flag[1]

Takeaway: most vulnerabilities stem from code already known vulnerable

[1] García "Side-Channel Analysis and Cryptography Engineering: Getting OpenSSL Closer to Constant-Time (Manuscript)" (University of Tampere 2022)

Benchmark setup

Unified benchmark representative of cryptographic operations:

- Tools considered: Binsec/Rel[2], Abacus[3], ctgrind[4], dudect[5], Microwalk-CI[6]
- Total: 25 benchmarks from 3 libraries (OpenSSL, MbedTLS, BearSSL)
- Primitives: symmetric (AES, Chacha20), AEAD, asymmetric (RSA, ECDSA, EdDSA)
- Benchmark design: limits the amount of operations besides the target one (e.g. encryption)
- Common timeout limit (⌛): 1 hour

[2] Daniel et al. "Binsec/Rel" (S&P 2020)

[3] Bao et al. "Abacus: Precise Side-Channel Analysis" (ICSE 2021)

[4] Langley *Ctgrind* (https://github.com/agl/ctgrind 2010)

[5] Reparaz et al. "Dude, Is My Code Constant Time?" (DATE 2017)

[6] Wichelmann et al. "Microwalk-CI" (CCS 2022)

Benchmark results

Benchmark	Binsec/Rel2		Abacus		ctgrind		Microwalk		dudect	
	#V	T	#V	T	#V	T	#V	T	S	T
AES-CBC-bearssl (T)	36	0.10	36	3.65	36	0.16	36	1.39	○	100.51
AES-CBC-bearssl (BS)	0	0.31	0	10.69	0	0.17	0	1.55	◐	⌛
AES-GCM-openssl (EVP)	0	21.19	0	104.27	70	0.71	8	5.66	◐	⌛
RSA-bearssl (OAEP)	2	⌛	💣	356.41	87	0.57	0	146.52	◐	⌛
RSA-openssl (PKCS)	1	⌛	0	551.72	321	1.32	46	52.06	○	618.73
RSA-openssl (OAEP)	1	⌛	💣	535.91	546	1.73	61	59.90	○	771.3

Benchmark results

Benchmark	Binsec/Rel2		Abacus		ctgrind		Microwalk		dudect	
	#V	T	#V	T	#V	T	#V	T	S	T
AES-CBC-bearssl (T)	36	0.10	36	3.65	36	0.16	36	1.39	○	100.51
AES-CBC-bearssl (BS)	0	0.31	0	10.69	0	0.17	0	1.55	◐	⌛
AES-GCM-openssl (EVP)	0	21.19	0	104.27	70	0.71	8	5.66	◐	⌛
RSA-bearssl (OAEP)	2	⌛	💣	356.41	87	0.57	0	146.52	◐	⌛
RSA-openssl (PKCS)	1	⌛	0	551.72	321	1.32	46	52.06	○	618.73
RSA-openssl (OAEP)	1	⌛	💣	535.91	546	1.73	61	59.90	○	771.3

- Tools generally agree on symmetric crypto, not for asymmetric crypto
- Support for vector instructions is essential

Vulnerability case-study

Replication of published vulnerabilities:

- 7 vulnerable functions from 3 publications
- Both the function itself and its context are targeted
- Total: 11 additional benchmarks

Example from two vulnerabilities:

- Two vulnerable functions: modular inversion and GCD computation
- Two different contexts: RSA key generation[7] and RSA key validation[8]

[7] Aldaya et al. “Cache-Timing Attacks on RSA Key Generation” (TCHES 2019)

[8] García et al. “Certified Side Channels” (USENIX 2020)

Case-study results

Benchmark	Binsec/Rel2 V	Binsec/Rel2 T	Abacus V	Abacus T	ctgrind V	ctgrind T	Microwalk V	Microwalk T
RSA valid. (MbedTLS)		⌛		490.01	✓	0.40	✓	278.94
GCD		⌛		37.74		0.21	✓	22.96
modular inversion		⌛		242.1	✓	0.24	✓	141.82
RSA keygen (OpenSSL)		0.17	💣	8.66		6.36	✓	842.02
GCD	✓	⌛		⌛	✓	0.19	✓	3.61
modular inversion		⌛		⌛	✓	0.21	✓	5.96

- Tools struggle to scale on these functions, except Microwalk
- Other limitations: support for indirect flows and internal secrets

Recommendations

1. Provide support for SIMD instructions

2. Provide support for indirect flows

3. Provide support for internally generated secrets (e.g. key generation)

4. Promote usage of a standardized benchmark

5. Improve usability for static tools (e.g. core-dump initialization)

6. Make libraries more static analysis friendly

Conclusion

- We surveyed the state-of-the-art of vulnerability detection tools
- We introduced a common benchmark allowing fair comparison of these tools
- We identified limitations in the current literature and issued recommendations for the community
- Our benchmark will soon be available on: `https://github.com/ageimer/sok-detection/`

Q&A