Poster: Symbolic Path Cost Analysis for Side-Channel Detection*

Tegan Brennan
University of California Santa Barbara
tegan@cs.ucsb.edu

Seemanta Saha
University of California Santa Barbara
seemantasaha@cs.ucsb.edu

Tevfik Bultan
University of California Santa Barbara
bultan@cs.ucsb.edu

ABSTRACT

We present a static, scalable analysis technique for detecting side channels in software systems. Our method is motivated by the observation that a sizable class of side-channel vulnerabilities occur when the value of private data results in multiple distinct control flow paths with differentiable observables. Given a set of secret variables, a type of side channel, and a program, our goal is to detect the set of branch conditions responsible for potential side channels of the given type in the program, and generate a pair of witness paths in the control flow graph for the detected side channel. Our technique achieves this by analyzing the control flow graph of the program with respect to a cost model (such as time or memory usage), and identifies if a change in the secret value can cause a detectable change in the observed cost of the program behavior. It also generates a pair of witness paths in the control flow graph, differing only on the branch conditions influenced by the secret, and whose observable output under the given side channel differs by more than some user defined threshold. We implemented our approach in a prototype tool, CoCo-Channel (**Co**mpositional **Co**nstraint-based side **Channel** analyzer), for analyzing Java programs.

ACM Reference Format:
Tegan Brennan, Seemanta Saha, and Tevfik Bultan. 2018. Poster: Symbolic Path Cost Analysis for Side-Channel Detection. In *Proceedings of 40th International Conference on Software Engineering Companion, Gothenburg, Sweden, May 27-June 3, 2018 (ICSE '18 Companion),* 2 pages.
https://doi.org/10.1145/3183440.3195039

1 OVERVIEW

Modern software systems frequently store and manipulate personal data, such as location, credit card information or medical history. Protection of such private data is critical for users of these systems. A class of information leaks, referred to as side channels, use non-functional properties of program execution to obtain information about private data. Potential side channels include execution time, memory usage, size and timings of network packets, and power consumption. The growing number of demonstrations of realistic side-channel attacks that result in critical security vulnerabilities [5, 8, 9] motivates the need for scalable analysis techniques and tools devoted to the detection of side-channel vulnerabilities [6].

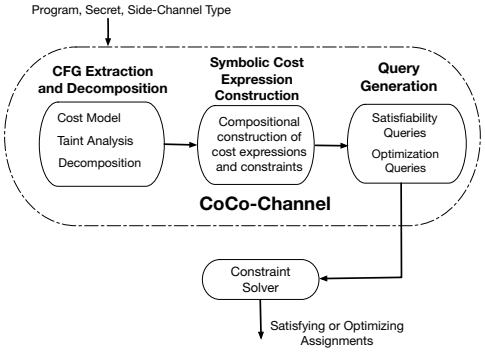

Figure 1: Architecture of CoCo-Channel**.**

We present such an analysis. The key part of our technique is compositional construction of symbolic cost expressions for control flow components of a program. These cost expressions provide a symbolically-expressed over-approximation of all possible cost values that each program component can incur. Once these cost expressions are developed, queries about the variability of a component's cost under a given side channel can be formulated and answered using existing constraint solvers.

Figure 1 gives an overview of the three phases of our technique:

(1) *Annotated Control Flow Graph Extraction and Decomposition:* We first use taint analysis to identify the branch conditions in the program that are influenced by the secret information. Then, we extract a reducible control flow graph where each basic block is annotated with a cost based on the cost model used. We decompose the control flow graph to a set of nested loop and branch components to facilitate compositional symbolic cost expression construction.
(2) *Symbolic Cost Expression Construction:* We construct cost expressions in a compositional manner starting with simple branch and loop components, and iteratively traversing the control flow graph based on the nesting relationships of the components. We use one integer variable per branch and loop component, that encode which branch is taken and how many times a loop is executed, respectively. Figure 2(a) shows the annotation of a simple branch component. The symbolic variable b_v is introduced and denotes which branch is taken. Figures 2(b) and (c) show the annotation of a loop component containing this branch component. The variable k_v denotes the number of iterations of the loop, while h_v denotes how many times out of the k_v iterations the if branch is taken. Reuse of the expression computed in (a) can be seen.

*This material is based on research sponsored by NSF under grant CCF-1548848 and by DARPA under the agreement number FA8750-15-2-0087. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195039

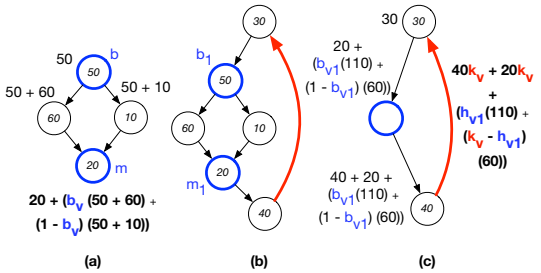

Figure 2: Annotation of (a) a simple branch component and (b), (c) a loop component containing the branch in (a).

Constraints on the symbolic variables (such as the restriction of b_v to either 0 or 1 or $h_v \leq k_v$) are also generated.

(3) *Constraint Solver Query Generation:* We generate queries for constraint solvers that are satisfiable if only if there exist two paths in the control flow graph, that differ only on the secret-dependent branch conditions, whose observable output under the given side channel differs by more than some user defined threshold. The set of variables in a cost expression can be partitioned into two categories, the set of secret-dependent variables $\vec{v_s}$ and the set of secret-independent variables $\vec{v}$. We create two instances of the cost expression E of a component and its corresponding constraints F. The set of secret-dependent variables in E_1 is $\vec{v}_{s1}$ and that in E_2 is $\vec{v_{s2}}$. The set of public variables $\vec{v}$ is shared across both. We then formulate and send the following query to an SMT-Solver:

$$\exists \vec{v_{s1}}, \vec{v_{s2}}, \vec{v} \text{ such that } |E_1 - E_2| > \delta \wedge F_1 \wedge F_2 \quad (1)$$

As an extension, we also generate optimization queries that identify two such paths while minimizing the number of loop executions or maximizing the cost difference.

2 EXPERIMENTS

We implemented our analysis in a Python prototype CoCo-Channel, which performs the decomposition and annotation described above and interfaces with tools for control flow graph extraction and satisfiability and optimization querying.

Control Flow Graph Extraction and Taint Analysis. The extraction of the control flow graph and necessary taint analysis was done using Janalyzer [4], an in-development research tool for the static analysis of Java Byte-code developed by Balasubramanian et. al.

Satisfiability and Optimization Queries. We integrated CoCo-Channel with the SMT-Solver Z3 [7] through Z3Py, the Z3 API in Python [3]. All queries of the type formulated above are sent to Z3 in order to determine either their unsatisfiability or a satisfying or optimizing assignment of the symbolic variables.

2.1 Results on the STAC Benchmark

We evaluate CoCo-Channel on a benchmark suite of fourteen examples from the DARPA Space/Time Analysis for Cybersecurity (STAC) program [1, 2]. All fourteen questions require determining the vulnerability of large Java client-server or peer-to-peer applications to side-channel attacks in time. Each specifies a secret value, and we evaluate CoCo-Channel's ability to determine if there is a side channel in time that leaks information about that secret.

Table 1: Experimental Results

Application	Secret	Vulnerable	Size	Result
GabFeed_1	secretKey	Yes	90	SC
GabFeed_1	password	No	12	Safe
GabFeed_2	secretKey	No	86	SC
GabFeed_3	searchTerm	Yes	286	SC
SnapBuddy_2	secretKey	Yes	71	SC
SnapBuddy_3	password	No	13	Safe
PowerBroker_1	myOffer	Yes	83	SC
PowerBroker_4	myBid	No	94	Safe
PowerBroker_4	userKey	Yes	18	SC
WithMi_4	userKey	No	21	Safe
WithMi_4	connected	Yes	901	SC
WithMi_5	connected	No	916	SC
Collab	userData	No	12	Safe
LawDB	isRestricted	Yes	736	SC

Our experiments were run on a single computer with an Intel Core i5-6600K CPU @ 3.50GHz and 32GB of RAM. The results are given in Table 1. The **Application** column identifies the application and version under test. The **Secret** column defines the value is specified as secret. **Vulnerable** denotes whether or not the STAC program considers the given application version to leak information about the given secret. **Size** gives the number of basic blocks in our extracted control flow graph. **Result** gives CoCo-Channel's verdict on the application's vulnerability.

CoCo-Channel's analysis agrees with DARPA's official verdict in all but two cases. Importantly, no vulnerable case is incorrectly classified. Moreover, CoCo-Channel is able to correctly specify the component of the control flow graph that introduces the side channel in each correctly identified vulnerable case. In the two cases in which our answers differ from DARPA's, we manually inspected the results and determined that there is indeed leakage in both cases, but that this leakage is not strong enough for an attacker to fully recover the secret. The quantification of side-channel strength is beyond the scope of CoCo-Channel at present.

REFERENCES

[1] [n. d.]. https://github.com/Apogee-Research/STAC/tree/master/Engagement_Challenges. ([n. d.]).
[2] [n. d.]. Space/Time Analysis for Cybersecurity (STAC). https://www.darpa.mil/program/space-time-analysis-for-cybersecurity. ([n. d.]).
[3] [n. d.]. Z3 API in Python. ([n. d.]). http://www.cs.tau.ac.il/~msagiv/courses/asv/z3py/guide-examples.htm
[4] Daniel Balasubramanian, Zhenkai Zhang, Dan McDermet, and Gabor Karsai. 2017. Janalyzer: A Static Analysis Tool for Java Bytecode. (08/2017 2017).
[5] David Brumley and Dan Boneh. 2005. Remote timing attacks are practical. *Computer Networks* 48, 5 (2005), 701–716.
[6] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. 2010. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 191–206.
[7] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.
[8] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical timing side channel attacks against kernel space ASLR. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 191–205.
[9] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 605–622.