Scaling Application-Level Dynamic Taint Analysis to Enterprise-Scale Distributed Systems

Xiaoqin Fu
Washington State University
Pullman, WA, USA
xiaoqin.fu@wsu.edu

Haipeng Cai
Washington State University
Pullman, WA, USA
haipeng.cai@wsu.edu

ABSTRACT

With the increasing deployment of enterprise-scale distributed systems, effective and practical defenses for such systems against various security vulnerabilities such as sensitive data leaks are urgently needed. However, most existing solutions are limited to centralized programs. For real-world distributed systems which are of large scales, current solutions commonly face one or more of scalability, applicability, and portability challenges. To overcome these challenges, we develop a novel dynamic taint analysis for enterprise-scale distributed systems. To achieve scalability, we use a multi-phase analysis strategy to reduces the overall cost. We infer implicit dependencies via partial-ordering method events in distributed programs to address the applicability challenge. To achieve greater portability, the analysis is designed to work at application level without customizing platforms. Empirical results have shown promising scalability and capabilities of our approach.

CCS CONCEPTS

• **Security and privacy → Distributed systems security**; **Software security engineering**.

KEYWORDS

Distributed systems, dynamic taint analysis, scalability, new bugs

ACM Reference Format:

Xiaoqin Fu and Haipeng Cai. 2019. Scaling Application-Level Dynamic Taint Analysis to Enterprise-Scale Distributed Systems. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE 2020), May 23–29, 2020, Seoul, South Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3338906.3342506

1 PROBLEM AND MOTIVATION

With increasing demands for various computational tasks, more and more enterprise-scale software systems are becoming distributed. These systems suffer from peculiar security vulnerabilities (e.g., data leaks across processes) due to their great complexity, large scale, and distributed design. For instance, if sensitive data (e.g., username and password) leak, there may be serious resulting losses and damages. In this context, we need an appropriate technique, such as a taint analysis, to detect sensitive information flows across multiple decoupled processes of a distributed program to defend distributed software against security vulnerabilities. However, there are multiple challenges to most existing taint analyses when applied to real-world, enterprise-scale distributed software systems, including (1) scalability challenge because of the high complexity and large code size of those systems, (2) applicability challenge due to the implicit dependencies among their decoupled (distributed) processes as in common distributed programs, and (3) portability challenge resulting from the platform customizations often required by the analyses.

ICSE 2020, May 23–29, 2020, Seoul, South Korea

ACM ISBN 978-1-4503-5572-8/19/08.
https://doi.org/10.1145/3338906.3342506

2 BACKGROUND AND RELATED WORK

Most early taint analyses [12, 14–16] are static and suffer from imprecision because of the nature of static analysis. They are also unsound for modern languages with dynamic features [11]. In addition, traditional static analyses can hardly apply to distributed programs because of exacerbated inaccuracies due to implicit dependencies among decoupled components of distributed programs, and thus these static analyses face applicability challenges. On the other hand, since most existing dynamic analyses [8, 9, 17] need customized platforms or architecture-specific emulators/frameworks, they face portability challenges. In particular, while the approach in [3] could compute inter-process dependencies, it has not been implemented nor evaluated on enterprise-scale distributed systems, and its (heavyweight) design implies scalability challenges. Several other dynamic approaches [1, 2] target JavaScript programs and do not work with common distributed systems either.

3 APPROACH

Based on Soot [10], we have developed an application-level dynamic taint analysis scalable to enterprise-scale distributed programs. Our approach computes statement-level taint paths as the final results after a rapid but rough computation of method-level results in a pre-analysis phase to balance the analysis precision and overheads while attaining high scalability. The overall workflow of our solution is depicted in Figure 1. It takes three **inputs** from the user: a distributed program D under analysis, the program input I for D, and a user configuration C including two message-passing API lists of sources and sinks.

Our technique works in three phases. In the first phase (**pre-analysis**), it computes approximated method-level taint paths according to the source/sink pairs in C. Then, in the second phase (**coverage-analysis**), it creates a statement coverage only for executed methods on the method-level taint paths from the first phase. Finally, in the third phase (**refinement**), the technique derives all valid statement-level taint paths, where the statements are covered and on associated method-level paths, as the final results.

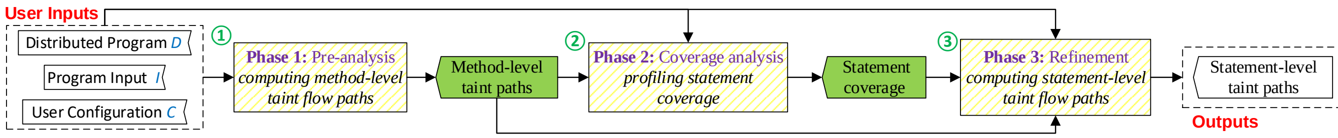

Figure 1: The overall workflow of the proposed technique for dynamic taint analysis of enterprise-scale distributed software.

Our tool reuses relevant code from previous work [4–7, 13] for method execution profiling, hybrid dependence abstraction, and threading-induced dependence analysis. It greatly reduces the overall cost by narrowing down the scope of the fine-grained (statement-level) analysis according to the intermediate results, hence overcoming the scalability challenge. To solve the applicability challenge, it computes implicit inter-process dependencies derived from happens-before relations among executed method events. To achieve portability, it is designed as an application-level solution without any platform customizations.

4 EVALUATION

We implemented our technique as an open-source tool and applied it to eight Java distributed system subjects of various application domains, architectures, and scales. The executions analyzed were driven by integration, load, system tests coming along with these systems. All possible pairs of (24) sources and (39) sinks manually curated were considered as taint-flow queries. With this setup, we assessed the scalability and effectiveness of our approach.

Scalability. Our technique was shown as promisingly scalable and efficient for enterprise-scale distributed systems. It took, on average per subject execution, 7 seconds to answer each individual query beyond a 15-minute one-time cost for all possible queries, with an almost-negligible storage cost (only 81MB) and an acceptable (less than 1x) run-time overhead.

Table 1: New vulnerabilities discovered by our technique

Subject	Vulnerability	Status	#Cases	#Confirmed
Netty	Issue 9456	Fixed	1	1
Thrift	Issue 4924	Confirmed	5	4
	Issue 4926	Confirmed		
	Issue 4928	Confirmed		
	Issue 4929	Pending		
	Issue 4930	Confirmed		
Voldemort	Issue 505	Pending	4	0
	Issue 506	Pending		
	Issue 507	Pending		
	Issue 508	Pending		
xSocket	Bug 25	Pending	1	0

Effectiveness. Our empirical results also revealed promising capabilities of the proposed solution in terms of effectiveness. Beyond finding 16 out of 22 existing real-world information flow vulnerabilities (that are documented as publicly disclosed CVEs), our approach successfully discovered 11 new security vulnerabilities, as outlined in Table 1. These new bugs are related to several enterprise-scale distributed systems (e.g., Netty, Thrift, Voldemort, and xSocket). All of these 11 cases have been confirmed by our own manual inspection. Furthermore, 5 of these have been confirmed by the developers/maintenainers of respective systems, including one case (on Netty) already fixed after a relevant pull request was opened, and a new branch, including the fixed code, was merged to the master branch of the project's repository.

5 CONCLUSION

We developed a scalable application-level dynamic taint analysis for enterprise-scale distributed systems, addressing several challenges faced by existing peer techniques via a multi-phase, refinement-based analysis strategy working purely at application level (hence avoiding any platform customizations). We implemented our technique for Java and applied it to eight distributed systems against diverse executions. Our empirical results demonstrated its promising scalability for enterprise-scale distributed systems and the capability of finding both existing and new security vulnerabilities.

REFERENCES

[1] Thomas H Austin and Cormac Flanagan. 2009. Efficient purely-dynamic information flow analysis. In *Proceedings of the ACM SIGPLAN Fourth Workshop on Programming Languages and Analysis for Security*. ACM, 113–124.

[2] Thomas H Austin and Cormac Flanagan. 2010. Permissive dynamic information flow analysis. In *Proceedings of the 5th ACM SIGPLAN Workshop on Programming Languages and Analysis for Security*. ACM, 3.

[3] Soubhagya Sankar Barpanda and Durga Prasad Mohapatra. 2011. Dynamic slicing of distributed object-oriented programs. *IET software* 5, 5 (2011), 425–433.

[4] Haipeng Cai. 2018. Hybrid Program Dependence Approximation for Effective Dynamic Impact Prediction. *IEEE Transactions on Software Engineering* 44, 4 (2018), 334–364.

[5] Haipeng Cai and Raul Santelices. 2014. DIVER: Precise Dynamic Impact Analysis Using Dependence-based Trace Pruning. In *Proceedings of International Conference on Automated Software Engineering*. 343–348.

[6] Haipeng Cai, Raul Santelices, and Douglas Thain. 2016. DiaPro: Unifying Dynamic Impact Analyses for Improved and Variable Cost-Effectiveness. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 2 (2016), 18.

[7] Haipeng Cai and Douglas Thain. 2016. DistIA: A cost-effective dynamic impact analysis for distributed programs. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 344–355.

[8] James Clause, Wanchun Li, and Alessandro Orso. 2007. Dytan: a generic dynamic taint analysis framework. In *Proceedings of the 2007 international symposium on Software testing and analysis*. ACM, 196–206.

[9] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 5.

[10] Patrick Lam, Eric Bodden, Ondrej Lhoták, and Laurie Hendren. 2011. Soot - a Java Bytecode Optimization Framework. In *Cetus Users and Compiler Infrastructure Workshop*.

[11] Benjamin Livshits, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z Guyer, Uday P Khedker, Anders Møller, and Dimitrios Vardoulakis. 2015. In defense of soundiness: a manifesto. *Commun. ACM* 58, 2 (2015), 44–46.

[12] Andrew C Myers. 1999. JFlow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 228–241.

[13] Venkatesh Prasad Ranganath and John Hatcliff. 2007. Slicing concurrent Java programs using Indus and Kaveri. *International Journal on Software Tools for Technology Transfer* 9, 5-6 (2007), 489–504.

[14] Sanjay Rawat, Laurent Mounier, and Marie-Laure Potet. 2011. Static taint-analysis on binary executables.

[15] Andrei Sabelfeld and Andrew C Myers. 2003. Language-based information-flow security. *IEEE Journal on selected areas in communications* 21, 1 (2003), 5–19.

[16] Xinran Wang, Yoon-Chan Jhi, Sencun Zhu, and Peng Liu. 2008. Still: Exploit code detection via static taint and initialization analyses. In *2008 Annual Computer Security Applications Conference (ACSAC)*. IEEE, 289–298.

[17] David Yu Zhu, Jaeyeon Jung, Dawn Song, Tadayoshi Kohno, and David Wetherall. 2011. TaintEraser: Protecting sensitive data leaks using application-level taint tracking. *ACM SIGOPS Operating Systems Review* 45, 1 (2011), 142–154.