Towards Scalable Defense of Information Flow Security for Distributed Systems

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

ABSTRACT

It is particularly challenging to defend common distributed systems against security vulnerabilities because of the complexity and their large sizes. However, traditional solutions, that attack the information flow security problem, often fail for large, complex real-world distributed systems due to scalability problems. The problem would be even exacerbated for the online defense of continuously-running systems. My proposed research consists of three connected themes. First, I have developed metrics to help users understand and analyze the security characteristics of distributed systems at runtime in relation to their coupling measures. Then, I have also developed a highly scalable, cost-effective dynamic information flow analysis approach for distributed systems. It can detect implicit dependencies and find real security vulnerabilities in industrial distributed systems with practical portability and scalability. In order to thoroughly solve the scalability problem in general scenarios, I am developing a self-adaptive dynamic dependency analysis framework to monitor security issues during continuous running. In this proposal, I outline the three projects in a related manner as to how they consistently target the central objective of my thesis research.

CCS CONCEPTS

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

KEYWORDS

Information flow, Security, Scalability, Distributed system

ACM Reference Format:
Xiaoqin Fu. 2019. Towards Scalable Defense of Information Flow Security for Distributed Systems. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3293882.3338988

1 THE RESEARCH PROBLEMS

With more and more performance and scalability demands by various computation tasks, distributed systems are developed increasingly. Compared with centralized software, distributed systems have multiple unique characteristics. Their code size is generally very large. And their decoupled components run at different machines asynchronously without a global timing mechanism. These characteristics not only complicate security issues in distributed systems [1–4, 12], but also bring about severe challenges to information flow analysis techniques.

Among a variety of security threats (e.g., code injection) distributed software suffers, a major type lies in assorted vulnerabilities in information flow paths in distributed programs. In these programs, sensitive information (e.g., username or password) might leak and cause serious losses/damages. To defend against such information flow threats, it is crucial to check sensitive data that passes throughout the entire system (across its distributed components and corresponding processes).

Effective information flow analysis often requires fine-grained (e.g., statement-level) computation of control and data flows. However, precise, fine-grained information flow analysis is usually very expensive. The great complexity of distributed systems is a major reason that most existing relevant approaches are not applicable (e.g., due to scalability barriers) or very limited utility (e.g., only for single component/process). For many distributed systems (e.g., online/cloud services) that are normally running continuously, it is desirable to keep monitoring them against security threats. In these scenarios, it is even more difficult to achieve and maintain the scalability.

Moreover, in addition to the scalability problem, there are multiple additional challenges to cost-effective information flow analysis solutions for distributed systems. One such difficulty is that inter-process flows among components are implicit since the components of the distributed system are decoupled [9]. These implicit flows are often not modeled by traditional information flow analysis approaches, which rely on explicit reference among code entities to compute the dependencies among them. Yet another problem concerns portability—the analysis approach based on customized environments would require frequent updating due to the constant evolution of the underlying platforms.

In the rest of this proposal, I outline my current contributions. The focus is put on how to design realistic solutions to deal with the scalability challenges in information flow security defense for large, complex distributed systems. Then, I lay out ongoing/future research that aims to generalize those solutions to more challenging use scenarios (e.g., continuously system running).

2 EXISTING AND EXPECTED CONTRIBUTIONS

The objective of the research is to establish a cost-effective dynamic information flow security analysis framework, which can address

ISSTA '19, July 15–19, 2019, Beijing, China

ACM ISBN 978-1-4503-6224-5/19/07…$15.00
https://doi.org/10.1145/3293882.3338988

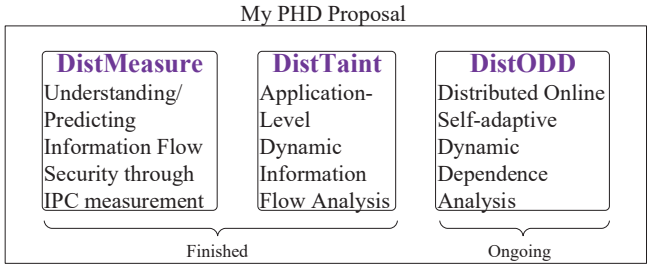

Figure 1: An overview of thesis research on scalable information flow security analysis for distributed software.

the scalability and other relevant challenges (e.g., portability and implicit dependency) for distributed systems. Accordingly, my thesis research has been focused on three associated themes which are depicted in Figure 1.

2.1 DistMeasure: Understanding and Predicting Information Flow Security through IPC Measurement

In distributed systems, inter-process communications (IPC) represent a major aspect of their (run-time) behaviors. Using IPC metrics, I attempted to predict security factors such as the number and length of information flow paths after understanding the communication with respect to typical user input. However, there were not many appropriate metrics for IPC in distributed software. Thus, I defined four novel IPC metrics: *run-time message coupling (RMC)*, *run-time class coupling (RCC)*, *class central coupling (CCC)*, and *inter-process reuse (IPR)* with various (method/class/process/system) levels [16].

Firstly, the process-level RMC is the number of messages sent from one process to another. The system-level RMC is then defined as the average of the process-level RMC on all communicating process pairs. Moreover, the class-level RCC is the ratio of the total number of methods in the first class in the first process that is dependent on any method in the second class in the second process, to the total number of methods in any process, other than the first process, that is dependent on any method in the second process. The process-level RCC metric is then set to the size of the union set of entire dependence sets of all methods in all classes of the process, where the numerator is the size of the union set of remote dependence sets of those methods. The system-level RCC metric is the average of such process-level RCC measures on the Cartesian product of two sets which include all processes.

Next, the process-level CCC is the sum of all RCC of all possible class pairs which one class is in any local process and another in any remote process. The system-level CCC is then defined as the mean of process-level CCC measures. For inter-process reuse, the method-level IPR is the ratio of the intersection of the local dependence set (set of methods in the process that depends on the given method) and remote dependence set (set of methods that depend on the given method but are in any process other than the first process), to the size of the entire set of methods covered in the execution. The system-level IPR metric is then set to the ratio of the sum of method-level IPR measures on all methods in the execution, to the size of the entire set of these methods. CCC and IPR are derived from coupling metrics essentially.

Table 1: Correlations between IPC metrics and (the direct measures of) information flow security

Quality factors	IPC metrics			
	RMC	**RCC**	**CCC**	**IPR**
Security (path count)	0.159	0.348	**0.618**	**-0.450**
Security (path length)	0.196	**0.453**	**0.583**	-0.371

As shown in Table 1, I got the information flow path number and length have positive correlations with three metrics (RMC, RCC, and CCC). Correlations are significant for CCC especially. The reason is perhaps that those higher values indicate more complicated systems which should have more and longer information flow paths (lower security). On the contrary, IPR has significant, negative correlations with both direct security measures. It means that more sharing of functionalities (measured by IPR) related to fewer and shorted vulnerable paths (higher security) while the more inter-process dependencies (measured by RMC/RCC/CCC) are related to more and longer taint flow paths (lower security) [16]. Moreover, we could predict the security trend of the system, upward or downward, according to trends of IPC metrics and relevant correlations.

In sum, I defined (four) IPC metrics which provide a promising approach for quantifying information flow security in executions. They could help us understand, analyze, and even predict security factors of distributed systems.

2.2 DistTaint: Application-level Dynamic Information Flow Analysis for Distributed Systems

To resolve the aforementioned challenges (applicability, portability, and scalability) of information flow security solutions, I also developed *DistTaint*, a creative application-level dynamic information flow analysis approach for distributed programs. First, it addresses the applicability challenge by inferring inter-process implicit dependencies from happens-before relations among method events. Conversely, most traditional information flow analyses, such as DTA++ [18], are only applied to centralized programs because of the lack of the capability of detecting implicit dependencies. Second, as an application-level approach, it overcomes the portability challenge without requiring any platform customizations. By contrary, as system-level analysis, TaintDroid [15] needs to customize Android with the portability problem. Lastly, even though DistTaint is an application-level dynamic information flow analysis, it can solve the scalability challenge through a multi-phase analysis strategy. It precedes a pre-analysis to produce method-level results employed to narrow down the scope of the fine-grained flow analysis in order to reduce the overall analysis overhead.

DistTaint computes an approximated set of method-level information flows with respect to the sources and sinks (*Phase 1*) firstly, followed by a statement coverage profiling (*Phase 2*). Guided by method-level flows and the statement coverage, DistTaint then refines the approximated results to statement-level using the information of multiple modalities and levels—static dependencies at

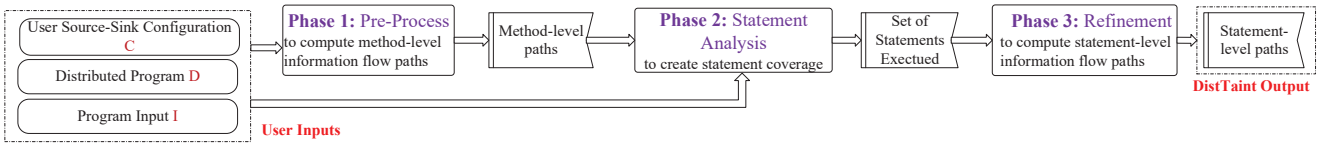

Figure 2: The architecture and overall work flow of DistTaint

statement-level and execution events at method-level (*Phase 3*). DistTaint checks all possible pairs of sources and sinks exercised during the analyzed system execution and reports valid statement-level information flow paths from the sources to sinks with source, remote and sink statement paths. Besides the statement-level information flow analysis time, efficiency metrics such as execution/analysis time and storage costs are recorded too. The architecture with the overall work flow of DistTaint is shown in Figure 2.

DistTaint Phase 1 (Pre-Process): In the first step, DistTaint inserts probes for communication events using a list of IPC APIs (e.g., Socket APIs) to identify probe points based on string matching [10]. In the second step, monitors reuse relevant parts of DistIA [10] and piggyback the additional items (the data length, a logical clock and the sender id) to it. In the third step, DistTaint creates method-level source-sink pairs from the source-sink configuration file given by the user. Lastly, method-level path analysis gathers execution traces to compute the method-level information flow paths using an offline algorithm, which retrieves the partial ordering of method-execution events by comparing their time stamps.

DistTaint Phase 2 (Statement-Coverage Analysis): Phase 2 contains three steps. First, DistTaint constructs intra-procedural control dependency graphs for all components and inserts one coverage monitoring probe for each branch of each component. Second, when the instrumented system executes, covered branches are recorded. Eventually, from covered branches and control dependencies on graphs, covered statements are inferred.

DistTaint Phase 3 (Refinement): In this phase, DistTaint computes statements-level information flow paths as the final output, using the pre-computed method-level paths. DistTaint also utilizes statement coverage gathered from phase 2 to optimize its performance and reduce the analysis costs. Specifically, it utilizes DIVER [8], a hybrid dynamic dependence analysis for single-process programs and executions, to profile method-execution events at instance-level. Next, DistTaint merges distributed execution traces (collected from distributed processes) and then builds a dynamic dependency graph (DDG) followed by pruning it with the statement coverage. Finally, statement-level information flow paths are computed from the DDG.

Discovering Information Flow Paths and Real-world Vulnerabilities: DisTaint detected information flow paths in four Java distributed software with high precision. We could find real vulnerabilities related to information flow security from sources (e.g., CVE database and bug repositories). DistTaint successfully discovered 9 of 10 real vulnerabilities, such as *CVE-2014-0085* [1]. It only missed one for Voldemort subject because it was not exercised in the executions considered.

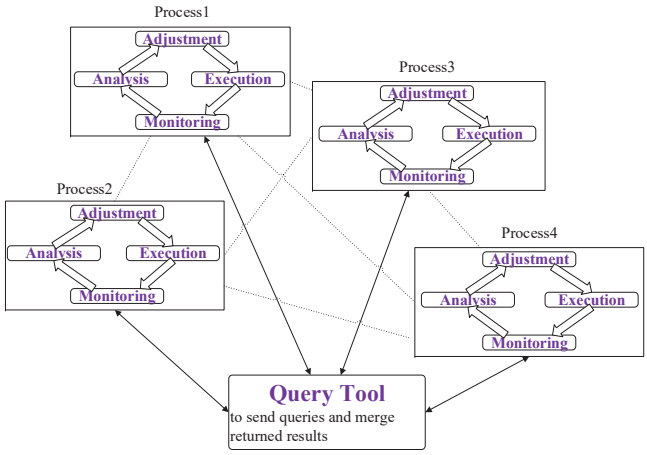

Figure 3: The high-level architecture of DistODD

2.3 DistODD: Distributed Online Self-adaptive Dynamic Dependency Analysis

As an ongoing effort, I am developing DistODD, a self-adaptive dynamic dependency analysis approach, against the scalability problem in monitoring information flow paths in continuously-running distributed systems. DistODD will monitor the distributed system continuously. It will compute and return relevant dependency sets from all distributed processes of the system in an *online* fashion according to the queries provided by users.

DistODD will have feedback control loops that will realize the self-adaption. These feedback control loops are infinite and decentralized with four activities: monitoring, analysis, adjustment, and execution. Every decoupled process should have a control loop. I will develop a query tool whose client sends queries and merges results computed and returned from tool services instrumented in distributed processes. The high-level architecture of DistODD is illustrated in Figure 3.

As a self-adaptive dynamic dependency analysis approach, DistODD should automatically choose a suitable configuration and computation strategy to meet the requirements (e.g., cost budget) defined by users. For example, it monitors the system to gather and analyze performance data such as the calculation time of a dependency set. If the analysis cost exceeds a predefined threshold (e.g., 2 seconds, as part of the user's cost budget), DistODD can adjust itself via choosing and exerting optimal analysis configurations. And a self-adaptation action would be initiated. DistODD would then change to the most effective configuration and computation strategy that should finish within 2 seconds. Next, DistODD

would compute new analysis results according to configuration and strategy adjusted. After the execution (computation), DistODD will monitor the system again and a new control loop will start.

3 RELATED WORK

Dynamic coupling metrics were defined by Arisholm et al. [6] for object-oriented software. The dynamic class export/import coupling (IC_CD and EC_CD) metrics initially motivated us to define RMC metric. Dynamic coupling metrics are also used to measure complexity [17] and architectural risks [27]. Most of these metrics were defined according to explicit dependencies among the entities (e.g., method, class, instance, etc.). So, they are not suitable to measure inter-process communication in distributed systems. By contrary, our IPC metrics can be used to measure inter-process communication and to predict security and performance factors for distributed systems.

For information flow security, there are various static and dynamic analysis solutions developed. Some static information flow analysis tools [20] [20] [25] [5] have been developed for mainstream languages. And several other static analysis algorithms for distributed systems were developed, but they focus on special types of systems rather than the generically most type of distributed systems [11], such as EAndroid [21]/DroidForce [22]/IccTA [19] for Android applications, and points-to analysis [24] for Java RMI programs. The techniques based on dynamic analysis can monitor information flows during executions [23]. They can analyze information flow paths more precisely and efficiently than static analyses do [7]. Dynamic information flow analyses are able to keep runtime overhead moderate via gathering some static information before the executions.

Automating software activities allow software systems to automatically monitor system behaviors and to trigger self-adaptation to compensate for deviations if thresholds predefined are exceeded [14]. There are two dimensions for a self-adaptive application: the managed system (the system to be controlled), and the controller [26]. Adaption feedback control loops should be main entities throughout the life-cycle of self-adaptive applications [13]. They should be called **MAPE** loops because of their four activities: **M**onitor, **A**nalyze, **P**lan and **E**xecute [13].

For distributed systems, I have to consider the interactions among the different activities of control loops realized in different components [13]. According to the different interaction methods, there are five patterns: hierarchical control, master/slave, regional planner, fully decentralized, and information sharing [13]. For DistODD, the fully decentralized pattern is employed such that each component of the analysis itself has a complete control loop for adaptability and scalability purposes.

4 CONCLUSION

My thesis research aims at a scalable approach to information flow security defense for distributed systems. Towards this goal, I have been exploring three connected directions. First, I proposed a novel set of IPC metrics for measuring distributed systems' run-time behaviors in relation to information flow security with several levels. I also demonstrated the practical usefulness of the IPC metrics for understanding and predicting information flow security factors computed from relevant datasets collected from diverse sources. As future work, I will explore how IPC *coupling* measures *predict* different security factors that are difficult to measure directly. I also plan to expand the scope to include some *cohesion* metrics of distributed systems.

Second, I have developed an application-level dynamic information flow analysis approach (DistTaint) for distributed programs, with several advantages to solve key analysis challenges (implicit dependence, portability, and scalability). In the first place, DistTaint could detect explicit and implicit information flows (i.e., data and control dependencies, respectively). Also, it is an application-level approach so that it could work transparently on distributed systems without customizing the underlying platforms. Along this line, I also plan for a few following lines of work. Firstly, I will research ways to further optimize the efficiency of DistTaint. Then, I will gather feedback information from users to estimate whether should the approach contain the variable-level analysis with apt costs.

As an ongoing research project, I am currently working on a self-adaptive dynamic dependency analysis approach, called DistODD. It is able to enable online information flow security defense for continuously-running systems. This would address the scalability problem in a more general context. The approach is expected to meet the scalability needs of users via its decentralized control that realizes the self-adaptation.

REFERENCES

[1] 2014. Vulnerability Details : CVE-2014-0085. https://www.cvedetails.com/cve/CVE-2014-0085/.
[2] 2017. CVE-2017-5637. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5637.
[3] 2017. Top 10 Security Vulnerabilities of 2017. https://resources.whitesourcesoftware.com/blog-whitesource/top-10-security-vulnerabilities-of-2017.
[4] 2018. CVE - Common Vulnerabilities and Exposures (CVE). https://cve.mitre.org/.
[5] AdaCore. 201o. SPARKPro. https://www.adacore.com/sparkpro.
[6] Erik Arisholm, Lionel C Briand, and Audun Foyen. 2004. Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering* 30, 8 (2004), 491–506.
[7] 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.
[8] 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.
[9] Haipeng Cai and Douglas Thain. 2016. DISTEA: Efficient Dynamic Impact Analysis for Distributed Systems. *arXiv preprint arXiv:1604.04638* (2016).
[10] 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.
[11] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. 2011. *Distributed Systems: Concepts and Design* (5th ed.). Addison-Wesley Publishing Company.
[12] CVE. 2015. CVE-2015-3254. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3254.
[13] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. 2013. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*. Springer, 1–32.
[14] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. 2008. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering* 15, 3-4 (2008), 313–341.
[15] 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.
[16] Xiaoqin Fu and Haipeng Cai. 2019. Measuring Interprocess Communications in Distributed Systems. In *IEEE/ACM International Conference on Program Comprehension (ICPC)*. http://chapering.github.io/pubs/icpc19.pdf.

[17] Anjana Gosain and Ganga Sharma. 2017. Object-oriented dynamic complexity measures for software understandability. *Innovations in Systems and Software Engineering* 13, 2-3 (2017), 177–190.

[18] Min Gyung Kang, Stephen McCamant, Pongsin Poosankam, and Dawn Song. 2011. Dta++: dynamic taint analysis with targeted control-flow propagation.. In *NDSS*.

[19] Li Li, Alexandre Bartel, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, Steven Arzt, Siegfried Rasthofer, Eric Bodden, Damien Octeau, and Patrick McDaniel. 2015. Iccta: Detecting inter-component privacy leaks in android apps. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 280–291.

[20] Andrew C Myers, Lantian Zheng, Steve Zdancewic, Stephen Chong, and Nathaniel Nystrom. 2001. Jif: Java information flow. *Software release. Located at http://www.cs.cornell.edu/jif* 2005 (2001).

[21] Damien Octeau, Patrick McDaniel, Somesh Jha, Alexandre Bartel, Eric Bodden, Jacques Klein, and Yves Le Traon. 2013. Effective Inter-Component Communication Mapping in Android with Epicc: An Essential Step Towards Holistic Security Analysis. In *Proceedings of USENIX Security Symposium*.

[22] Siegfried Rasthofer, Steven Arzt, Enrico Lovat, and Eric Bodden. 2014. Droidforce: Enforcing complex, data-centric, system-wide policies in android. In *2014 Ninth International Conference on Availability, Reliability and Security*. IEEE, 40–49.

[23] Alejandro Russo and Andrei Sabelfeld. 2010. Dynamic vs. static flow-sensitive security analysis. In *Proceedings of the 23rd IEEE International Conference on Computer Security Foundations Symposium (CSF)*. IEEE, 186–199.

[24] Mariana Sharp and Atanas Rountev. 2006. Static analysis of object references in RMI-based Java software. *IEEE Transactions on Software Engineering* 32, 9 (2006), 664–681.

[25] Vincent Simonet. [n.d.]. Flow Caml. https://www.normalesup.org/~simonet/soft/flowcaml/.

[26] Norha M Villegas, Hausi A Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. 2011. A framework for evaluating quality-driven self-adaptive software systems. In *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems*. ACM, 80–89.

[27] Sherif M Yacoub and Hany H Ammar. 2002. A methodology for architecture-level reliability risk analysis. *IEEE Transactions on Software engineering* 28, 6 (2002), 529–547.