On the Scalable Dynamic Taint Analysis for Distributed Systems

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

ABSTRACT
To protect the privacy and search sensitive data leaks, we must solve multiple challenges (e.g., applicability, portability, and scalability) for developing an appropriate taint analysis for distributed systems. We hence present DistTaint, a dynamic taint analysis for distributed systems against these challenges. It could infer implicit dependencies from partial-ordering method events in executions to resolve the applicability challenge. DistTaint fully works at application-level without any customization of platforms to overcome the portability challenge. It exploits a multi-phase analysis to achieve scalability. By proposing a pre-analysis, DistTaint narrows down the following fine-grained analysis' scope to reduce the overall cost significantly. Empirical results showed DistTaint’s practical applicability, portability, and scalability to industry-scale distributed programs, and capabilities of discovering security vulnerabilities in real-world systems.

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

KEYWORDS
Distributed systems, Dynamic taint analysis, Scalability

ACM Reference Format:
https://doi.org/10.1145/nnnnnn.nnnnnnn

1 PROBLEM AND MOTIVATION
Distributed systems have been increasingly developed for various computation tasks. However, Distributed systems are usually complex and have large code sizes. Their decoupled components often run at different machines without a global clock. They hence suffer from security vulnerabilities, such as data leakage, owing to these characteristics. For example, sensitive information (e.g., account/password) might leak and cause serious losses and damages. Thus, we need to check sensitive data flowing throughout a distributed program (across its decoupled processes) to defend against such information flow threats.

Taint analysis is employed for defending against code vulnerabilities by identifying where the sensitive data may be leaked from taint flows. However, existing (static or dynamic) taint analyses have suffered several challenges. The applicability challenge appears because most of them were developed for centralized software and relied on explicit dependencies, but dependencies among decoupled (distributed) components in common distributed software are implicit. The portability challenge exists since some existing taint analyzers may depend on customized or modified platforms. The scalability challenge appears owing to the complexity and large-sizes of the most real-world industry-scale distributed systems.

2 BACKGROUND AND RELATED WORK
Developers have developed both static and dynamic taint analyses. Most early analyses are static [15] (e.g., FlowCaml [17] and JFlow [12]). These solutions suffer the imprecision of static analysis naturally and are unsound owing to the dynamic features of modern languages [11]. Some other static analyses focus on special programs rather than the most common distributed systems [7], such as points-to analysis [16] for Java RMI systems, and EAndroid [13] for Android apps. Thus, they face applicability challenges. On the other hand, most dynamic analyses [6, 8, 9, 18, 19] require architecture-specific frameworks/emulators (e.g., PIN and QEMU) or platform customizations, hence portability challenges exist. Several other dynamic analyses [1, 2] focus on special JavaScript programs and they are unsuitable for general distributed systems so that there are also applicability challenges. In particular, the algorithm CGCA [3] could induce inter-process dependencies. However, it has not been implemented on large distributed programs so that it has to face the scalability challenge.

3 APPROACH
To support information flow security defense for distributed software, we have developed DistTaint, an
application-level dynamic taint analyzer (a.k.a information flow analyzer) based on Soot framework [10]. It reused some codes of previous works [4, 5] and Indus [14]. DISTTAINT could solve challenges to existing analyzers. To overcome the applicability challenge, DISTTAINT infers statically implicit inter-process dependencies from a global partial ordering of executed methods via monitoring happens-before relations among method events in the executions. As an application-level solution, it eliminates the requirement of platform customizations so as to resolve the portability challenge. DISTTAINT generates the final results (statement-level taint paths) after a rapid but rough computation of method-level results in a pre-analysis. By using approximate intermediate results to narrow down the scope of the fine-grained analysis, DISTTAINT reduces the overall cost greatly to solve the scalability challenge.

The overall workflow of DISTTAINT is illustrated in Figure 1. To balance effectiveness and cost, DISTTAINT has three phases for the distributed system: pre-analysis, coverage analysis, and refinement. There are three inputs from the user: the distributed program D under analysis, the input I for D needed by DISTTAINT, and a user configuration C including two lists of message-passing APIs as the sources and sinks between which DISTTAINT will compute all valid flow paths.

The approach starts with the pre-analysis phase which computes an approximated set of method-level taint paths with respect to the sources and sinks (Phase 1). Next, coverage-analysis phase produces a statement coverage via filtering the methods in these method-level taint paths (Phase 2). Finally, refinement phase refines the approximated results to statement-level using method-level flows and the statement coverage (Phase 3). DISTTAINT checks all possible pairs of sources and sinks exercised during the analyzed system execution and reports final results: valid statement-level taint paths from the sources to sinks consisting of three parts: source, remote and sink statement paths. Besides statement-level and method-level coverage/paths, efficiency metrics such as the execution/analysis time and storage cost are recorded too.

4 EVALUATION

We applied DISTTAINT to eight distributed Java programs, most of which are real-world systems, with various architectures, code sizes, domains, and message-passing ways. Three types (Integration, load, and system) of testing were implemented. We considered pairs of all (24) sources and (39) sinks from the default user configuration as queries to each execution. We aimed to estimate the effectiveness, scalability, and practicality of DISTTAINT. And we answer three research questions through the evaluation:

RQ1 How effective is DISTTAINT in terms of its precision?
Answers: DISTTAINT significantly reduced the taint checking effort by users so that it was effective. It showed high precision and potentially promising recall after we validated with random samples by hand.

RQ2 How efficient/scalable is DISTTAINT?
Answers: DISTTAINT was promisungly scalable and reasonably efficient for distributed systems. It took 15 minutes for analyses on all possible queries from a given user configuration and 7 seconds on a query (a source/sink pair). The executions had less than 1x run-time slowdown and a small (81MB) storage cost.

Table 1: Real-world vulnerabilities discovered by DISTTAINT

<table>
<thead>
<tr>
<th>Subject</th>
<th>Vulnerability</th>
<th>Found</th>
<th># Cases</th>
<th># FNs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thrift</td>
<td>CVE-2015-3274</td>
<td>Yes</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Voldemort</td>
<td>Issue 101</td>
<td>Yes</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Issue 381</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Issue 387</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Issue 352</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Issue 378</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Issue 377</td>
<td>No</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Issue 155</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZooKeeper</td>
<td>CVE-2014-0089</td>
<td>Yes</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Bug 2569</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CVE-2018-8012</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RQ3 Can DISTTAINT discover real-world vulnerabilities?
Answers: DISTTAINT showed promising capabilities in successfully discovering 9 of 10 real-world security vulnerabilities, shown in Table 1. Meanwhile, the only one false negative demonstrated that DISTTAINT relies on the vulnerabilities exercised during the analyzed executions of distributed systems. Nevertheless, DISTTAINT is able to find real vulnerabilities related to information flow paths.

5 CONCLUSION AND FUTURE WORK

We developed DISTTAINT, an application-level dynamic taint analysis tool, for distributed systems, that overcomes several practicality challenges to existing taint analysis approaches. It approximates inter-process dependencies based on happens-before relations among methods to address the applicability challenge (implicit dependencies). It transparently works on distributed systems without changing underlying platforms to overcome the portability challenge. Finally, DISTTAINT resolves the scalability challenge by using a multi-phase analysis strategy. We implemented DISTTAINT for Java and applied it to several large-scale distributed systems against diverse executions, and demonstrated its promising scalability and effectiveness, along with its capability of discovering various real security vulnerabilities. Based on DISTTAINT, we plan to develop a distributed online self-adaptive dynamic analysis framework for continuously running distributed services.
REFERENCES


