Field-based Static Taint Analysis for Industrial Microservices

Zexin Zhong*§, Jiangchao Liu*, Diyu Wu*, Peng Di*, Yulei Sui§ and Alex X. Liu*
Ant Group, Hangzhou, China* University of Technology Sydney, Australia§
{zhongzexin.zzx,jiangchao.ljc,wudiyu.wdy,dipeng.dp,alexliu}@antgroup.com,yulei.sui@uts.edu.au

ABSTRACT

Taint analysis is widely used for tracing sensitive data. However, the state-of-the-art taint analyzers face challenges on recall, scalability, and precision when applied on industrial microservices. To overcome these challenges, we present a field-based static taint analysis approach, which does not distinguish different instances of the same type but distinguishes fields of the same kind for tracing sensitive data on industrial microservices. The experimental results demonstrate that our approach is practical in industrial scenarios.

CCS CONCEPTS

• **Theory of computation** → **Program analysis**; • **Security and privacy** → *Information flow control.*

KEYWORDS

program analysis, taint analysis, microservices, security

ACM Reference Format:
Zexin Zhong*§, Jiangchao Liu*, Diyu Wu*, Peng Di*, Yulei Sui§ and Alex X. Liu*. 2022. Field-based Static Taint Analysis for Industrial Microservices . In *44nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3510457.3513075

1 INTRODUCTION

Because of the increasing damages of data breaches (a single data breach cost 4.42 million US dollars on average in 2020 [4]), many enterprises pay much attention to software security. Taint analysis, a information tracking technique that aims to reason about the control and data dependences from sources (of sensitive data) to sinks (e.g., possible leakage), has been widely used for finding potential data breachs [1, 2, 5, 6]. With the continuous expansion of industrial microservices used in companies (e.g., Ant Group), there is an urgent need for scalable taint analysis tools to trace sensitive data on large-scale microservices. However, the current state-of-the-art taint analyzers face challenges on recall, scalability, and precision when applied on industrial microservices.

Challenge1: **Recall** is one of the core challenges for static taint analysis to run on industrial applications. Most previous static taint analyzers run their analyses based on pre-built call graphs [1, 3, 6]. However, it is hard to obtain a thorough call graph statically in large-scale industrial microservice applications because of complex framework behaviors such as AOP (Aspect Oriented Programming), IoC (Inversion of Control), message services and events. While we can supplement the call graph by manually modeling these framework behaviors, it is costly and time-consuming, requiring substantial human resources. What is worse, it is hard to guarantee that all framework behaviors are properly modeled. Any missed framework behaviors can result in missing caller-callee relations in the generated call graph, which reduces the recall rate of taint analyzers.

Challenge2: **Scalability** is another obstacle for static taint analysis. Industrial applications are typically large-scale and complex, consisting of multiple modules. Even though it is possible to get a sound and precise call graph, the obtained call graph will be very large. It is costly to run precise context sensitive inter-procedural analysis on such large-scale call graphs. What is worse, the memory usage could be overly huge if the heap is abstracted precisely for instances with field-sensitive analysis. Thus, taint analyzers face scalability challenge with respect to both time and memory consumption. Field-insensitive abstraction of heap does not distinguish fields of an object, which is scalable but can worsen the next challenge: precision.

Challenge3: **Precision** is another significant concern for static taint analysis. Industrial microservices can heavily use complex containers (e.g., map, list, or JSON object). Sensitive data can be propagated through these containers. Most field-sensitive analyses cannot handle such containers precisely, which leads to many false positives. These large amounts of false-positives make the analysis results useless for tracing sensitive data on industrial applications.

To resolve the above challenges and make taint analysis effective for large-scale industrial microservices, we present our field-based static taint analysis for tracing sensitive data.

2 OUR APPROACH

In this section, we present our field-based algorithm and explain how it solves the recall, scalability and precision challenges of static taint analysis on industrial microservices applications.

We choose "field-based" rather than "field-sensitive" which is a common choice in other taint analyzers for two reasons: (1) As we have mentioned before, the thorough call graphs of industrial applications cannot be easily built and are usually too large for current taint analyzers. Compared to field-sensitive analysis, field-based analysis relies less on pre-built call graphs (which will be shown later on the example in Figure 1); (2) Our investigation into the industrial applications found that each field usually represents a concept that keeps unchanged in all its usages in an application. Sensitive data is usually propagated among the fields assigned with specific concepts. Thus, even though field-based analysis is less precise than field-sensitive analysis, the precision loss is limited, especially on tracing sensitive data.

ICSE-SEIP '22, May 21–29, 2022, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-9226-6/22/05…$15.00
https://doi.org/10.1145/3510457.3513075

We present key features of our approach with the example in Figure 1. The function `foo(Request request)` (line 2 at Figure 1) is the request handler that will read the sensitive user phone number from the request and store it into an object of `Model`. Then it calls `bar(Model model)` which writes the user phone number to DB through `SampleDO`. The interception function `interceptorInvoke` of `TaskInterceptor`, which has been configured in the Spring XML configuration, will be executed before the invocation of `bar`. This interceptor will print the sensitive user phone number to a log file.

```
1  public Handler{
2    public void foo(Request request){
3      Model model = new Model();
4      model.setPhoneNumber(request.getPhoneNumber());
5      bar(model);}
6    public void bar(Model model){
7      SampleDO sampleDO = new SampleDO();
8      sampleDO.setPhoneNumber(model.getPhoneNumber());
9      DAO.insert(sampleDO);} }
10 public TaskInterceptor extends Interceptor {
11   public void interceptorInvoke (MethodInvocation
         invocation){
12     Model interModel = invocation.getMethod().
           getParameter().get(0);
13     Log.print(interModel.getPhoneNumber());} }
```

Figure 1: Motivating Example

There are two potential sensitive data leakage: one is to the DB in `bar`; the other is to the log file in the function `interceptorInvoke`. Most static taint analyzers can find the first leakage easily. However, they would fail to find the second one if the interception mechanism defined in Spring XML files are not properly modeled. In field-based analysis, the field `phoneNumber` of the class `Model` is seen as the same everywhere. Our field-based analysis will find the sensitive phone number is propagated to both DB and the log file, without any modeling of the interception mechanism in the Spring framework. This feature of the field-based algorithm indicates better scalability and recall for tracing sensitive data on large-scale microservices applications. It is also worth mentioning that, the field `phoneNumer` is assigned with a specific concept: the phone number of users. Programmers usually only use it to carry phone numbers. What is more, other fields (e.g.,`name`) are usually never used to carry user phone numbers. This phenomenon undermines the precision loss of field-based analysis compared to field sensitive analysis.

Furthermore, our approach models the operation of the frequently used containers (map, JSON, list, etc.) to support the field-based algorithm to trace sensitive data precisely within those containers. This container model improves the precision of our tool because of the precise data propagation in complex containers.

3 EXPERIMENT AND EVALUATION

We have implemented the proposed field-based approach on Java with GraalVM at Ant Group. The implementation has been deployed on a set of elastic cloud clusters, each of which is with eight 2.6GHz cores and a RAM of 64 GB. In industrial code, libraries are heavily used. We set a blacklist to ignore most of them. To evaluate the efficiency of the proposed approach, we use a set of production applications, which consists of 6 microservices applications in Java used in Ant Group for the experiment. The experimental results are shown in Table 1. The Column "Name" indicates the names of each microservices, which are anonymized. The Column "App" indicates the size of application bytecode of the analyzed microservices, measured in Megabytes. The Column 'Lib' indicates the sizes of the library bytecode in Megabytes. The column 'Time' represents the time consumption of our implementation for the analysis on the benchmarks in seconds.

The results demonstrate that our approach runs efficiently on large industrial microservices. The average time consumption of the proposed approach on the production-benchmark is 161 seconds. Even though for the worst case, which has 26.4MB application bytecode, the field-based approach generates the taint analysis results in 369 seconds.

Name	App(MB)	Lib(MB)	Time(s)
M1	1.3	193.9	14
M2	4.7	78	67.97
M3	15.4	68.2	166.89
M4	17.6	85.5	167.41
M5	18.2	80.6	180.8
M6	26.4	173.3	369
Avg	13.94	113.25	161.0

Table 1: Production-benchmark and results

4 CONCLUSION

This paper presents a field-based static taint analysis approach for tracking sensitive data in large-scale industrial microservices. Benefitting from the the field-based algorithm, which does not distinguish instances of the type but distinguishes fields of the same type, our approach is practical in industrial scenarios for sensitive data tracking.

5 ACKNOWLEDGMENTS

This work was supported by Ant Group through Ant Research Program.

REFERENCES

[1] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. FlowDroid: Precise Context, Flow, Field, Object-Sensitive and Lifecycle-Aware Taint Analysis for Android Apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '14)*. 259–269.

[2] Michael I. Gordon, Deokhwan Kim, Jeff H. Perkins, Limei Gilham, Nguyen Nguyen, and Martin C. Rinard. 2015. Information Flow Analysis of Android Applications in DroidSafe. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015*. https://www.ndss-symposium.org/ndss2015/information-flow-analysis-android-applications-droidsafe

[3] Wei Huang, Yao Dong, Ana Milanova, and Julian Dolby. 2015. Scalable and Precise Taint Analysis for Android. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015)*. 106–117.

[4] IBM Security and the Ponemon Institute. 2021. *Cost of a Data Breach Report 2021*. Technical Report. IBM Corporation.

[5] Manu Sridharan, Shay Artzi, Marco Pistoia, Salvatore Guarnieri, Omer Tripp, and Ryan Berg. 2011. F4F: Taint Analysis of Framework-Based Web Applications. In *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA '11)*. 1053–1068.

[6] Jie Wang, Yunguang Wu, Gang Zhou, Yiming Yu, Zhenyu Guo, and Yingfei Xiong. 2020. Scaling Static Taint Analysis to Industrial SOA Applications: A Case Study at Alibaba. In *ESEC/FSE'20*. 1477–1486.