Deep Understanding on Taint Analysis: a Survey on Existing Taint Analysis Techniques

Course Project Report

Aw, Snap!: Weijia Sun, Xinyu Lyu, Mengmei Ye
Rutgers University

ABSTRACT

This is a course project report for the course 16:332:579 - Malware Analysis and Reverse Engineering in Spring 2019 at Department of Electrical and Computer Engineering, Rutgers University.

Taint analysis has been actively discussed and studied in both the security and software engineering communities. The techniques behind the taint analysis mainly originate from the software engineering community. In the meantime, taint analysis has been widely used in the security domain to defend against many threat models in multiple applications. Considering the background and the applications of taint analysis, it is significant to have a comprehensive study on the existing taint analysis work, so that the researchers could have views from both of the domains.

This course report presents a survey on the existing taint analysis techniques to perform a comprehensive study by explaining the software engineering techniques and security usage, as well as the design/implementation challenges in the existing work. In addition, in the experiment section, we further evaluate the existing malware analysis tools, where these tools leverage taint analysis.

KEYWORDS

Malware Analysis, Software Engineering, Software Security, Mobile Security

ACM Reference Format:
Aw, Snap!: Weijia Sun, Xinyu Lyu, Mengmei Ye. 2019. Deep Understanding on Taint Analysis: a Survey on Existing Taint Analysis Techniques: Course Project Report. In *Proceedings of Malware Analysis and Reverse Engineering (16:332:579).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Taint analysis has been widely applied to enhance application/system security. Researchers/engineers on the security community and the software engineering community have been actively working on taint analysis for many years. Considering the significance of taint analysis in both of the community, we plan to collect the recent research work and do a survey report on the existing taint analysis techniques.

Particularly, we are going to target on the following three research questions (RQs) in this course project.

- RQ1: what are the software-engineering concepts behind taint analysis?
- RQ2: what are the advantages and limitations of taint analysis?
- RQ3: how efficient are the taint analysis techniques for Android malware analysis?

To answer the questions, we will be studying on recent research work published in the security or software-engineering conferences. Also, we will be empirically evaluating the efficiency of the open-source taint analysis tools about how they successfully analyze and detect Android malware. The reason we would like to target on Android malware analysis is that Android security is significantly important and the Android market share is more than 80% in the whole mobile market[12].

2 MOTIVATION

There has been much excellent research work published in both of the security community and the software engineering community. By reviewing the research papers published in recent years and evaluating the open-source tools for taint analysis, this comprehensive study could help researchers obtain the summary of recent work related to taint analysis as well as help us gain technical experience on the security and software engineering domains. In detail, the motivation for the project is listed in the following.

Contribution to the software-engineering domain. The taint analysis technology originates from the software engineering domain. The survey could be helpful for software-engineering researchers to have a comprehensive view about how the taint analysis technology was developed and improved in recent years.

Contribution to the security domain. Based on the RQs in our report, security researchers could have a collection of information on the advantages and disadvantages of taint analysis, which could help them further protect against the security vulnerabilities in the systems/applications.

Our personal interests and future career development. All of our group members have both backgrounds on software engineering and security. Working on the taint analysis is a great fit for us, and we have a strong passion for gaining a deep understanding of taint analysis. The survey on taint analysis would be a great opportunity for us to enhance technical skills related to software engineering and security.

3 RQ1: SOFTWARE ENGINEERING TECHNIQUES BEHIND TAINT ANALYSIS

In general speaking, the workflow of taint analysis could be the following steps.

(1) Taint an object (e.g., variable) as a source. Typically, the tainted object contains sensitive information.
(2) Analyze the program with an analysis tool, and generate the analysis log.

16:332:579, Spring 2019, Department of ECE Rutgers University
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

(3) If the tainted object eventually goes to the insecure or unauthorized destination, the analysis tool identifies that the behavior of the application is abnormal.

Specifically, the analysis techniques could be static analysis, dynamic analysis, and symbolic execution analysis. Static analysis mainly targets on the offline analysis of program structure and the source code without actually running the program [11]. Different from static analysis, dynamic analysis requires to execute the target program at runtime to detect the abnormal behaviors. In addition, compared to dynamic analysis that requires to provide actual inputs, symbolic execution analysis could obtain the tracing path of the tainted object with symbolic inputs [10].

4 RQ2: ADVANTAGES AND LIMITATIONS OF TAINT ANALYSIS

4.1 Applications and Usage

Based on our study, taint analysis has been leveraged in many different applications to defend against attacks, such as mobile and web malware. The web applications are vulnerable to injection attacks or cross-site scripting attacks. There has been much work conducting taint analysis to defend against the web attacks, such as [7][13]. In mobile security, the taint analysis mainly targets on whether the sensitive information (e.g., passwords, text messages, and location data) leaks to non-trusted party.

Because this course report mainly works on the Android malware analysis, we are discussing more details on this scope in the following.

Android security has been actively discussed in the security community. Many Android malware analysis tools apply static, or dynamic analysis to detect malware. Considering that sometimes the Android APK source code is not feasible, TaintDroid[5] leverages dynamic taint analysis with different levels of taint granularity by executing the apps and automatically tainting the sensitive sources.

However, dynamic taint analysis requires additional time consumption due to the app execution. Therefore, FlowDroid [1], as a static taint analysis tool, analyzes the bytecode of the target program with the context, flow, and object sensitivity. However, FlowDroid does not perform very well in the time consumption and memory usage, Huang et al. proposed DFlow and DroidInfer to achieve low performance and high precision with scalability [6].

Furthermore, due to imperfection of the basic analysis technology, researchers have been targeting on the sensitivity types in the analysis tools, such as context-based, flow-based, and type-based [1][6] in order to strengthen the effectiveness and efficiency of the taint analysis tools.

4.2 Limitations and Challenges

Trade-off between precision and performance of analysis. A great trade-off between precision and performance of analysis could effectively avoid over-tainting or under-tainting problems [10][11]. Specifically, although static analysis could analyze the program without causing additional app execution time, it could encounter over-taint/approximation problems. For example, an inappropriate static analysis could enroll in the infeasible dataflow, which is a waste of time and memory. On the other hand, although dynamic analysis could achieve high precision with appropriate test suites, it requires much time consumption to execute the apps in order to conduct the analysis. For the trade-off between time consumption and precision of analysis, a feasible solution could be adding additional work based on granularity and sensitivity as we mentioned in the previous section.

Test case generation and code coverage. Compared to static analysis, the dynamic analysis could effectively analyze the program based on the test suites. However, how to generate a suitable test suite is a significant but difficult topic. A great test suite could cover the most important portion of the program, namely high code coverage, so that the dynamic taint analysis tool could detect bugs with high efficiency. To mitigate this problem, there are many methods to generate suitable test suites, such as random fuzzing approach and symbolic testing. Recently, Wong et al. proposed a framework, namely IntelliDroid[14], to generate appropriate test suites with high code coverage, so that dynamic analysis could run the generated test suites in low-performance overhead.

Scalability. As we discussed in the previous subsections, researchers have spent great efforts on improving the precision of taint analysis. However, while achieving the precise taint analysis, researchers could encounter another problem, namely scalability. In details, after improving the precision, the time consumption and memory space usage heavily increase at the same time. When a program has a relatively large size, the taint analysis could spend a long time to finish the entire process and require a large memory space to trace the libraries and store the logs [6]. Therefore, to achieve scalability, it requires additional optimization work (e.g., using genetic algorithms [9]) or applies different analysis techniques (e.g., using data flow analysis instead of points-to analysis [6] or tracing sensitive data based on the lifespan of variables [1]).

5 CASE STUDY ON GENERAL TAINT ANALYSIS TOOLS

In order to have a better understanding before we actually work on the Android malware analysis tools, we first work on a dynamic taint analysis tool, namely Taintgrind[8], which uses the Valgrind[4] dynamic programming tool.

As is shown in the following source code example, we execute it with Taintgrind instructions provided by the authors. In the example, we assume the variable *a* in the *main* function contains sensitive data and further taint it. After the taint analysis, the variable *s* is treated as sensitive data as well, because the *get_sign* function assigns the value of *a* to the variable *s*.

```
#include "taintgrind.h"
int get_sign(int x, int y) {
    int s;
    if (x > 0)  s = x + y;
    else  s = x + 1;
    return s;
}
void main(int argc, char **argv){
    // Turns on printing
    TNT_START_PRINT();
    int a = 1000;
    int b = 2000;
    // Defines int a as tainted
    TNT_MAKE_MEM_TAINTED_NAMED(&a,4, "myint");
    int s = get_sign(a, b);
    // Turns off printing
    TNT_STOP_PRINT();
}
```

Furthermore, we extract the sensitive dataflow of this example from the log that is generated by Taintgrind, which is shown in the following. Based on the dataflow, it clearly presents the location of the sensitive computation and the flow direction of the tainted data.

```
0x4008EF: main (sign-ds.c:15) | t24_4548 = LOAD I32
    t21_5414 | 0x3e8 | 0xffffffff | --22310-- warning:
    evaluate_Dwarf3_Expr: unhandled DW_OP_ 0xf2
t24_4548 <- a
0x4007BC: get_sign (sign-ds.c:2) | STORE t41_1824 =
    t43_1624 | 0x3e8 | 0xffffffff | --22310-- warning:
    evaluate_Dwarf3_Expr: unhandled DW_OP_ 0xf2
x <- t43_1624
0x4007C2: get_sign (sign-ds.c:4) | t12_11253 = LOAD I32
    t49_1753 | 0x3e8 | 0xffffffff | --22310-- warning:
    evaluate_Dwarf3_Expr: unhandled DW_OP_ 0xf2
t12_11253 <- x
0x4007C8: get_sign (sign-ds.c:4) | t15_8648 = LOAD I32
    t12_11254 | 0x3e8 | 0xffffffff | --22310-- warning:
    evaluate_Dwarf3_Expr: unhandled DW_OP_ 0xf2
t15_8648 <- x
0x4007D0: get_sign (sign-ds.c:4) | STORE t27_2679 =
    t29_2209 | 0xbb8 | 0xffffffff | --22310-- warning:
    evaluate_Dwarf3_Expr: unhandled DW_OP_ 0xf2
s <- t29_2209
0x4007DE: get_sign (sign-ds.c:6) | t34_2455 = LOAD I32
    t31_2476 | 0xbb8 | 0xffffffff | --22310-- warning:
    evaluate_Dwarf3_Expr: unhandled DW_OP_ 0xf2
t34_2455 <- s
0x400904: main (sign-ds.c:15) | STORE t8_7902 = t10_9174
    | 0xbb8 | 0xffffffff | --22310-- warning:
    evaluate_Dwarf3_Expr: unhandled DW_OP_ 0xf2
s <- t10_9174
```

6 TENTATIVE PLAN FOR THE NEXT PHASE

We plan to finish all the RQs based on the deeper study of the existing work. In particular, for the experimental evaluation part, we plan to work with several open-source malware analysis tools with test benchmarks (e.g., Facebook Infer[3], TaintDroid[5], FlowDroid[1], and DroidBench[2]) on the following aspects:

- Effectiveness: whether the existing tools could detect abnormal behaviors (e.g., sensitive information leakage) in the malware.
- Recall and precision: whether the existing tools could correctly detect the malware in the system, and whether there are false positive problems in the results.
- Scalability: whether the existing tools could scale to the large Android Apps.

7 ADDITIONAL NOTES

In this submission, we mainly worked on the RQs 1 and 2 and added texts for the abstract and keywords sections, as well as sections 3, 4, 5, and 6.

Please notice that the text of sections 1 and 2 is from our project proposal. In the next phase, we are going to mainly focus on answering RQ3 and add additional text for RQs 1 and 2 with more information.

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)*. ACM, 259–269.

[2] Secure Software Engineering Group at Paderborn University and Fraunhofer IEM. [n. d.]. A micro-benchmark suite to assess the stability of taint-analysis tools for Android. https://github.com/secure-software-engineering/DroidBench

[3] Cristiano Calcagno and Dino Distefano. 2011. Infer: An Automatic Program Verifier for Memory Safety of C Programs. In *NASA Formal Methods*, Mihaela Bobaru, Klaus Havelund, Gerard J. Holzmann, and Rajeev Joshi (Eds.). Springer Berlin Heidelberg, 459–465.

[4] The Valgrind Developers. [n. d.]. Valgrind. http://valgrind.org/

[5] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, 393–407.

[6] 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*. ACM, 106–117.

[7] N. Jovanovic, C. Kruegel, and E. Kirda. 2006. Pixy: a static analysis tool for detecting Web application vulnerabilities. In *2006 IEEE Symposium on Security and Privacy (S P'06)*. 6 pp.–263.

[8] Wei Ming Khoo. [n. d.]. A taint-tracking plugin for the Valgrind memory checking tool. https://github.com/wmkhoo/taintgrind

[9] John R. Koza. 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. In *MIT Press*.

[10] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *2010 IEEE Symposium on Security and Privacy*. IEEE, 317–331.

[11] Elena Sherman and Matthew B. Dwyer. 2018. Structurally Defined Conditional Data-Flow Static Analysis. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dirk Beyer and Marieke Huisman (Eds.). Springer International Publishing, 249–265.

[12] StatCounter. 2019. Mobile Operating System Market Share Worldwide. http://gs.statcounter.com/os-market-share/mobile/worldwide

[13] Philipp Vogt, Florian Nentwich, Nenad Jovanovic, Engin Kirda, Christopher KrÃŒgel, and Giovanni Vigna. 2007. Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis.

[14] Michelle Y Wong and David Lie. 2016. IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware.. In *NDSS*, Vol. 16. 21–24.