SURVEY OF DYNAMIC TAINT ANALYSIS

Junhyoung Kim[1], TaeGuen Kim[1], Eul Gyu Im[2]

[1]Department of Computer and Software, Hanyang University, Seoul, Korea
[2]Division of Computer Science and Engineering, Hanyang University, Seoul, Korea
ebsud89@hanyang.ac.kr, cloudio17@hanyang.ac.kr, imeg@hanyang.ac.kr

Abstract: Dynamic taint analysis (DTA) is to analyze execution paths that an attacker may use to exploit a system. Dynamic taint analysis is a method to analyze executable files by tracing information flow without source code. DTA marks certain inputs to program as tainted, and then propagates values operated with tainted inputs. Due to the increased popularity of dynamic taint analysis, there have been a few recent research approaches to provide a generalized tainting infrastructure.

In this paper, we introduce some approaches of dynamic taint analysis, and analyze their approaches. Lam and Chiueh's approach proposed a method that instruments code to perform taint marking and propagation. DYTAN considers three dimensions: taint source, propagation policies, taint sink. These dimensions make DYTAN to be more general framework for dynamic taint analysis. DTA++ proposes an idea to vanilla dynamic taint analysis that propagates additional taints along with targeted control dependencies. Control dependency causes results of taint analysis to have decreased accuracies. To improve accuracies, DTA++ showed that data transformation containing implicit flows should propagate properly to avoid under-tainting.

Keywords: dynamic taint analysis

1 Introduction

Many software defects that cause memory and threading errors can be detected statically and dynamically. Static Analysis is to test and to evaluate an application by examining the code without execution of the application of an application by examining the source code without executing the application. It doesn't examine all possible execution paths and variable values also those invoked during execution. Dynamic analysis is to test and to evaluate an application during runtime. It reveals subtle defects or vulnerabilities whose cause is too complex to be discovered by static analysis.

The two approaches are complementary. To find application's vulnerabilities, we need to know execution paths and behaviors that user may use application actually. Although static analysis examines all code paths, it has weakness of a lot of execution time. Also, static analysis is possible to examine dead code that can't execute actually. But, dynamic analysis is not: it is reason that dynamic analysis is more effective tool to find vulnerabilities in application.

978-1-4799-4734-8 /14/$31.00 ©2014 IEEE

Dynamic analysis has become a fundamental tool in computer security research. Dynamic analysis is attractive because it allows us to motive about actual executions, and thus can perform precise security analysis based upon run-time information.

Two of the most commonly employed dynamic analysis techniques in security research are dynamic taint analysis and symbolic execution [8-9]. Dynamic taint analysis runs a program and observes which values are affected by predefined taint sources such as user input. Symbolic execution automatically builds a logical formula describing a program execution path. The two analysis approaches can be used to build formulas that represent only the parts of an execution that depends on tainted values.

Dynamic taint analysis is significant tools for analyzing application by tracing information flow without observing to source code. DTA observe certain inputs to a program as tainted, and then propagating other values operated with tainted inputs. Because of this, some techniques have been proposed based on dynamic taint tracking for detecting unknown vulnerabilities in software [1] or analyzing malicious software components leaking sensitive private data [2].

These several taint analysis techniques have accuracy problem caused by ranges on taint propagation. For instance, Schwartz et al. Ref. [3] point out several fundamental challenges including under-tainting and over-tainting. In this paper, we define problems in dynamic taint analysis and introduce DTA++ that solved these problems.

2 Dynamic taint analysis

Dynamic taint analysis is based on the observation that in order for an attacker to change the execution of a program illegitimately, attacker must cause a value that is normally derived from a trusted source to instead be derived from his own input.

We refer to data that originates or is derived arithmetically from an untrusted input as being tainted. In dynamic taint analysis, we first mark input data from un-trusted sources tainted, then monitor program execution to track how the tainted attribute propagates and to check when tainted data is used in dangerous ways.

2.1 General approach

Due to the increased popularity of dynamic tainting, there have been a few recent attempts at providing a generalized tainting infrastructure. Lam and Chiueh [7] propose an approach that instruments the code to perform taint marking and propagation. Their approach has two main drawbacks compared to ours. First, it requires the code to be recompiled, which is especially problematic when third party and system libraries are involved (as it is typically the case with real software). Second, their approach lacks support for control-flow based tainting. While it is true that most security applications of dynamic tainting only require data-flow based tainting, we believe that a truly general framework should support both types of propagation.

Brian Chess and Jacob West's approach [10] is that dynamic taint analysis is used for finding software's vulnerabilities. They monitor a target program as it executes in order to track untrusted user input. Their system works in conjunction with normal functional testing, so effort devoted to functional testing can be directly leveraged to uncover vulnerabilities.

2.2 Taint propagation

Taint Analysis is one kind of program flow analysis and we use it to define the influence of external data over the analyzed application. There is a need to follow this influence in order to determine the control over specific areas. Data from untrusted source become tainted object. If other object is computed with tainted object, this value also become tainted objected.

Taint Propagation is process that tainted object affect another object through computing in programs. The expression of this is below.

$$X \to t(Y) \text{ and } Y \to t(Z), \text{then } X \to t(Z)$$

X is a tainted object and t is a taint operator. If Y is affected by taint operator of tainted object X and Z is affected by object Y, then object Z is tainted by object X.

If different tainted objects propagate one object, the one has merged taint maker and propagates another objects as before. In dynamic taint analysis, this taint propagation is repeated about object from untrusted data.

2.3 DYTAN

Dynamic taint analyzer (DYTAN) [4] has three dimensions that characterize dynamic taint analysis: taint sources, propagation policy, and taint sinks. Because different dynamic taint analyses can be expressed by defining the analysis along these three dimensions, we defined our general framework in terms of these dimensions.

2.3.1 Taint source

Taint sources are a description of program data (memory locations) that should be initialized with taint markings. Memory locations can be of different types, including

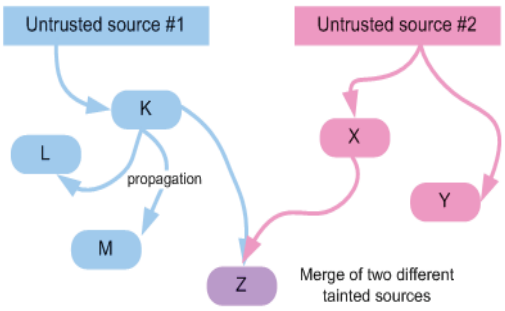

Figure 1 Diagram of taint propagation

variable names, function-return values, and data read from and I/O stream such as a file or a network connection.

2.3.2 Propagation policies

A propagation policy describes how taint markings should be propagated during execution. There are different ways in which to identify affecting data and define the mapping function.

2.3.3 Taint sink

At a high level, a taint sink is a location in the code where users want to perform some check on the taint markings of one or more memory locations. Taint sinks are characterized by four aspects: an ID, a memory location, a code location, and one or more checking operations to be performed at that code location and using the taint marking associated with that memory location.

2.3.4 DYTAN tools

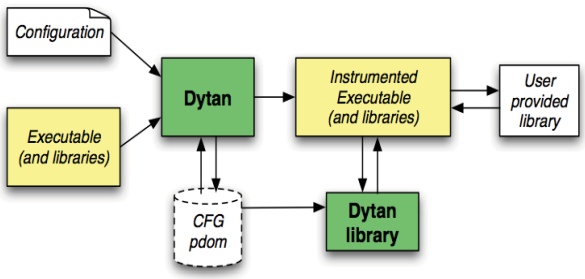

Figure 2 Overview of DYTAN tool

DYTAN is prototype framework for d supporting three dimensions: taint source, propagation policies, and propagation policies provides an overview of DYATAN's mode of operation. The user-provided configuration file specifies the kind of dynamic taint analysis to be performed in terms of taint sources, propagation policies, and taint sinks. Based on this configuration, DYTAN instruments the x86 executable on the fly to produce an instrumented executable. To per- form instrumentation on the fly, DYTAN leverages the PIN dynamic instrumentation framework, which is well supported and offers a rich APIs for manipulating x86 binaries.

2.4 DTA++

DTA++ [5] propose advanced idea to vanilla dynamic taint analysis that propagates additional taint along targeted control dependencies in order to ameliorate under-tainting caused by implicit flows. Specially, they concentrate on common case of information preserving transformations.

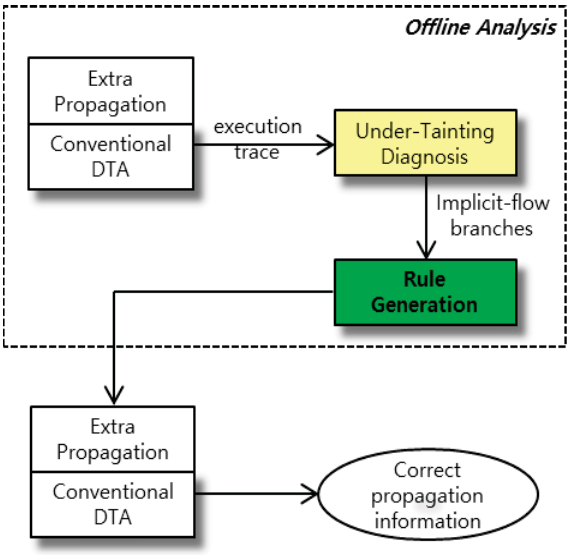

Figure 3 Overview of DTA++

2.4.1 System overview

DTA++ use BitBlaze platform to extend and enhance the taint analysis performed by BitBlaze's dynamic analysis component TEMU and its Tracecap plugin [6]. DTA++ is composed of two phases for performing efficiently. First, offline analysis phase detects and diagnoses under-tainting, generates DTA++ rules. Second, online analysis phase uses propagation rules generated by offline phase, performs future runs of dynamic taint analysis.

2.4.2 Offline analysis phase

Offline analysis phase generate DTA++ rules by using vanilla dynamic taint propagation. In this phase, if the condition at the branch is tainted, the values written by each control-dependent instruction should also be tainted.

2.4.2.1 Diagnosing under-tainting

In DTA++, diagnosis approach to search for parts of the execution that make control-flow decisions based on the input that is under-tainted, where the results of those decisions imply tight restrictions on the possible values of that input.

2.4.2.2 Rule generation

After diagnosing the cause of under-tainting, DTA++ determine how to decrease it by generating rules specifying how to propagate taint to the values affected by the flow. DTA++ adopt Clause et al. in DYTAN that apply taint propagation to all the conditional branches with a tainted condition, but DTA++ apply it to only the some branches in implicit flows found by the diagnosis algorithm.

2.4.3 Online analysis phase

Online analysis phase is taint propagation using DTA++ rules generated by the offline analysis phase. Using this propagation rules make DTA++ perform future runs of dynamic taint analysis with simple modification.

2.5 Analysis of approaches

There are two problem caused by control dependency in dynamic taint analysis. Control dependency is occurred in implicit flow. In DTA++, implicit flow is defined program structure in which tainted data affects control flow, so that the control flow difference might in turn affect other data.

Under-tainting is a situation where a value is not tainted even though it is affected by the tainted input. Considering too few control dependency make this situation, special in implicit flow. For instance, Vanilla dynamic taint analysis have under-tainting problem because no considering control dependency.

In contrast, too many control dependency in application lead to over-tainting problem. Some techniques propagate taint for all implicit flows, such as DYTAN.

3 Conclusions

Many software defects that cause memory and threading errors can be detected statically and dynamically. Static Analysis is the testing and evaluation of an application by examining the code without executing the application. Dynamic analysis is the testing and evaluation of an application during runtime. The two approaches are complementary because no single approach can find every error. Dynamic analysis has become a fundamental tool in computer security research.

Dynamic taint analysis is based on the observation that in order for an attacker to change the execution of a program illegitimately. Dynamic taint analysis is significant tools for analyzing application by tracing information flow without observing to source code. DTA observe certain inputs to a program as tainted, and then propagating other values operated with tainted inputs. Due to the increased popularity of dynamic tainting, there have been a few recent attempts at providing a generalized tainting infrastructure.

Taint propagation is main concept of dynamic taint analysis. There is a need to follow propagation in order to determine which tainted object affected another object in application. Tainted object is data from untrusted source. If other object is computed with tainted object, this value also become tainted objected.

In this paper, we introduce some approaches of dynamic taint analysis, and analyze main idea of these approaches. Lam and Chiueh's approach propose that instruments the code to perform taint marking and propagation.

DYTAN considers three dimensions: taint source, propagation policies, taint sink. These dimensions make DYTAN to be more general framework for dynamic taint analysis.

DTA++ proposes advanced idea to vanilla dynamic taint analysis that propagates additional taint along targeted control dependencies. Control dependency causes result of taint analysis to decrease accuracy. To improve this, DTA++ identify that data transformation containing implicit flows should propagate properly over avoiding under-tainting. DTA++, an enhancement to dynamic taint analysis that additionally propagates taint by considering control-flow dependencies, is technique that solves the under-tainting problem in implicit flows.

There are some problems that control dependency lead to under-tainting or over-tainting in dynamic taint analysis. There are ranges from no taint propagation to universal taint propagation for control dependency. For accuracy of analysis, many techniques are proposed in these days.

Acknowledgements

This research project was supported by Ministry of Culture, Sports and Tourism (MCST) and from Korea Copyright Commission in 2014.

References

[1] J. R. Crandall and Z. Su, On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In Computer and Communications Security (CCS), Alexandria, VA, USA, 2005, 235–248.

[2] A. Moser, C. Kruegel, and E. Kirda, Exploring multiple execution paths for malware analysis. In IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2007, 231–245.

[3] E. J. Schwartz, T. Avgerinos, and D. Brumley, All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2010, 317 - 331.

[4] Clause, J., Li, W., & Orso, A., Dytan: a generic dynamic taint analysis framework. In Proceedings of the 2007 international symposium on Software testing and analysis. ACM, 2007, 196-206.

[5] X Kang, M. G., McCamant, S., Poosankam, P., & Song, D, DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation. In NDSS, 2011.

[6] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. BitBlaze: A new approach to computer security via binary analysis (keynote invited paper). In International Conference on Information Systems Security (ICISS), Hyderabad, India, Dec. 2008, 1-25.

[7] L. C. Lam and T. Chiueh. A General Dynamic Information Flow Tracking Framework for Security Applications. In 22nd Annual Computer Security Applications Conference, 2006, 463–472.

[8] James C. King. Symbolic Execution and Program Testing. In Communications of the ACM, July, 1976, 385-394

[9] Edward J. Schwartz, Thanassis Avgerinos, and David Brumley. All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). In Proceedings of the 2010 IEEE Symposium on Security and Privacy. IEEE Computer Society, Washington, DC, USA, 317-331.

[10] Brian Chess, Jacob West. Dynamic taint propagation: Finding vulnerabilities without attacking, Information Security Technical Report, 2008, 33-39.