Dynamic Taint Analysis Tools: A Review

Abdullah Mujawib Alashjaee *alas0145@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Northern Borders University
Arar, 73222, Saudi Arabia

Salahaldeen Duraibi *dura6540@vandals.uidaho.edu*
[a] *Computer Science Department*
University of Idaho
Moscow, ID, 83844, USA
[b] *Computer Science Department*
Jazan University
Jazan, 45142, Saudi Arabia

Jia Song *jsong@uidaho.edu*
Computer Science Department
University of Idaho
Moscow, ID, 83844, USA

Abstract

Taint analysis is the trending approach of analysing software for security purposes. By using the taint analysis technique, tainted tags are added to the data entering from the sensitive sources into the applications, then the propagations of the tainted data are monitored carefully. Taint analysis can be done in two ways including static taint analysis where analysis is conducted without executing the program, and dynamic taint analysis where the tainted data is monitored during the program execution. This paper reviews the taint analysis technique, with a focus on dynamic taint analysis. In addition, some of the existing taint analysis tools and their application areas are reviewed. In the end, the paper summarises the defects associated with each of the tools and presents some of them.

Keywords: Taint Analysis, Static Analysis, Dynamic Analysis.

1. INTRODUCTION

Software security analysis is important for testing Commercial off the Shelf (COTS) systems. It can be accomplished by employing source code or binary code. However, source code is not available in most of the cases for software security analysis. Hence, binary code analysis is used for a number of reasons, including software forensics [1, 2], malware analysis [4], and performance analysis and debugging [3]. A number of binary code analysis approaches are in the literature, and the most popular ones include symbolic execution, concolic execution, static taint analysis and dynamic taint analysis [5].

Capitalizing on the issue of efficiency identified in the fuzzing techniques, symbolic execution, which is another conventional binary code analysis approach, has come into being [6]. Different from other techniques, such as concrete execution that take concrete input values, symbolic execution uses symbols that abstractly represent specified input values for vulnerability analysis [7]. However, the technique is suffering from the famous path-explosion problem when symbolically executing large programs [8].

In view of improving the problems identified in symbolic execution, researchers have started working on taint analysis. According to Xiajing Wang and colleagues [32], taint analysis has first been proposed by Funnywei [75]. Taint analysis works in a triple form manner of source, sink, and sanitizer. The source is where some untrusted or confidential input data is introduced to the application, probably from the application API or network interface. The sink is the sensitive point in the application that performs secure operations, such as sensitive banking transactions, and needs to be protected from violation of integrity, confidentiality, and availability of the application. Sanitizer refers to the process where the tainted input data is no longer considered harmful to the information security of the application by means of removal of harmful operations such as malicious programs that may cause the application to function out of its intended operation [7]. In short, taint analysis helps software analyzers to take an informed decision on whether the data introduced at the input point or source of the application can be allowed to propagate to the sink point without harm, or else the application will suffer from some security issues such as data leakage or other more dangerous operations such as buffer overflow.

There are two types of taint analysis approaches, Static Taint Analysis (STA) and Dynamic Taint Analyses (DTA). In STA, analyzers test an application by examining the intermediate code without the execution of the application. Static taint analysis is mostly carried out in a two-step manner, including disassembly of the intermediate code and conducting analysis on the resulting assembly code [9, 10]. It may sometimes use binary codes for application security analysis. However, since source code rarely comes with COTS software, it makes the STA approach harder to combat malicious programs, thus reducing its application. Similarly, analyzing binary codes with STA approaches have endured complications and challenges [11]. For instance, malware with strong evasion techniques can easily escape the STA approach [4]. These limitations have motivated the identification of alternative approaches that can overcome such defects to analyze applications accurately and reliably.

On the other hand, in DTA, applications are tested during runtime for possible vulnerabilities [12]. Both STA and DTA approaches have weaknesses and strengths. For example, when conducting information flow analysis in an application, DTA can suffer from runtime overhead which may make it fall short of analyzing all the code, causing it not to discover some potential threats. On the other hand, since STA analyzes the application code without executing it, it may suffer from an accuracy issue. As a result, some researchers proposed tools that mix the two techniques to analyze flaws or vulnerabilities in applications [13-15]. Some researchers have used the STA approach before or after DTA [15, 16]. In doing so, for example, STA is employed after DTA in order to see whether analysis has missed anything suspicious after using DTA. STA can be employed before DTA to analyze the behavior of the application prior to the code execution in a live environment.

Conducting vulnerability analysis on software in cases where the source code is not available, for example, COTS, software security analysts use the DTA approach as the ideal option [17].

Usually, DTA methods are implemented at the hardware level or code level. For instance, some of the DTA methods are implemented within the hardware [18-21]. Although this implementation relatively provides the lowest overhead, it is less flexible and the least practical because it requires significant architectural and microarchitectural changes to the processor. By using source code instrumentation to track the propagation of the tainted data, DTA can also be performed at the code level of the software [22-26]. This approach is also less practical since source code is hardly available for security analysis in most applications. However, to perform data flow tracking without hardware modification or source code, the DTA methods such as Dytan [27], Libdft [22], Argos [28], BitBlaze [29], and DTA++ [30] use binary code to perform security analysis. This approach is used more prevalently because it enables a wide variety of analysis. For this paper, the DTA approaches that use binary code is the focus of interest. This paper is aimed at presenting an analytical view of static and dynamic taint tools.

The rest of the paper is organized as follows: Section 2 is the background of the study providing basic knowledge of DTA. Section 3 presents a review on several commonly used STA tools, and Section 4 presents the review of DTA tools. Section 5 presents lessons learned, and Section 6 concludes the paper.

2. BACKGROUND

The primary focus of this paper is on the use of DTA approaches for software security testing. In the following subsections, we will provide the readers with the preliminary details for understanding the purpose, techniques, and key concerns of the research work of taint analysis. Different aspects of DTA are in a general manner summarized in Section 2.1.

2.1 Concepts of Dynamic Taint Analysis Approach

This section, explains basic concepts about DTA. DTA is also referred to as dynamic information flow tracking. The approach is about observing the behaviour of certain untrusted programs as they execute in a monitored environment. The central idea of the DTA approach is to label some incoming data values as tainted and to propagate them through operands as instructions execute. This happens by marking certain values in the CPU registers or memory locations as tainted, and observing the tainted data as they propagate during the code execution. A taint propagation policy is associated with each instruction to specify whether each output operand should be tainted or untainted based on the taint status of the input operands.

2.1.1 Analysis Techniques

DTA can be accomplished either by control or data flow tracking [1]. Control flow tracking is an approach to show how the hierarchical flow of control in a given application is sequenced. It makes an easier analysis of all possible execution paths of an application. The output of control flow analysis is usually expressed in Control Flow Graphs (CFG), where each instruction or a block of instructions is represented by a node and the control flow between two nodes is indicated by direct edges. On the other hand, based on the problem that needs to be investigated, DTA computes a set of possible values at every point in an application. That is, data flow tracking is for monitoring programs from the perspective of how the program processes the data [2].

2.1.2 Offline and Online Dynamic Taint Analysis

Dynamic information flow tracking can be performed offline or online. In offline analysis, the trace of program execution is recorded into trace files and later analyzed by replaying those trace files. In online analysis, the security analysis is conducted by monitoring the program execution. Online analysis is considered to be more accurate and easier to implement, but it suffers from slow execution [32]. Using traces for later analysis will let analysts get thorough information about what has happened, but the raw trace file may become complicated to understand [8]. On the other hand, in doing online analysis, incident response can be performed in a timely manner, but it may sometimes end up as a false alert [8, 32].

2.1.3 Modes of Implementation

Dynamic information flow tracking tools can be implemented at the user or kernel level of an operating system. This depends on the type of security matter under investigation and the level of information extraction needed for the analysis [3]. For instance, programs such as word processing and imaging applications are executed at the user level of operating systems. On the other hand, operating systems perform their operations at the kernel level. Hence, DTA tools can be developed as targeting either the analysis of the user applications that work at the user level or the analysis of the privileged applications that have direct access to the kernel level processes.

2.1.4 Taint Granularity

The granularity of tracking the application has important implications for the usage of DTA tools. In DTA, analysis can be conducted in a fine-grained or coarse-grained manner [34, 35]. In coarse-grained information flow, the tainted data is tracked at the granularity of a whole system level, while in fine-grained analysis, tainted data is tracked at the granularity at the process level.

At the data level, data units can be tagged as small as a bit or as large as chunks of memory [35]. That is, in coarse-grained analysis, fewer tags are required compared to fine-grained analysis. Coarse-grained analysis tools are often easier to design and implement but may inherit trackless information causing false alarms [35]. Conversely, by tracking the information flow at the fine granularity, the analysis is more flexible and more precise but may require more memory space [35]. In most cases, researchers consider one over another believing that, for example, one is more effective than the other. However, Vassena et al. argued that both coarse and fine granularities are equally important in DTA [34].

2.1.5 Dynamic Binary Instrumentation (DBI)

Dynamic Binary Instrumentation refers to the analysis of an executable code through injecting additional code into the compiled code at runtime. This is usually implemented using a Just-in-Time (JIT) compiler. In DBI, code is executed in basic blocks, and the code at the end of each block is modified so that control is passed to the analysis engine to perform a number of checks, such as whether a system call is being executed [6]. Two of the most popular frameworks for achieving dynamic instrumentation in Windows are DynamoRIO [7] and Intel Pin [8].

2.2 Challenges in Dynamic Taint Analysis

Challenges that DTA has to face when analyzing applications for security can include soundness, precision, and overhead. In some papers, these are referred to as over-tainted, under-tainted, and overhead [17]. Under-tainted refers to a situation where values expected to be marked as tainted are not, while over-tainted occurs when too many values are marked as tainted. For instance, tools are still suffering from the issue of accuracy where in some cases taint may spread too much or happen to be missing, causing over-tainting or under-tainting respectively. The issue of balancing speed and accuracy is another challenge [8]. DTA tools sometimes cause overhead, minimizing the performance of the system [12].

3. STATIC TAINT ANALYSIS TOOLS

The STA technique is used for application vulnerability testing. There are a number of software vulnerability testing tools that utilize STA for deep and exhaustive tracking and prevention of suspicious data. In most cases, STA is conducted outside the testing environment, but it provides better code coverage analysis compared to DTA [40]. Existing researches employing STA can be categorized into three main areas including software privacy analysis [41], software forensics [42], web application vulnerability analysis [43, 44, 45, 46].

Conventional privacy-enhancing technologies have fallen short of assessing and auditing the privacy of cutting edge technologies. Detailed and often manual examination that is needed for these technologies makes privacy assessment a more complex, time-consuming, and tiresome task. Taint analysis has recently been used for realtime privacy monitoring of system privacy [41]. For instance, Celik et al. present SAINT, a system that can be used by the IoT consumers to assess the privacy risks that can come with the adoption of IoT devices [47]. Likewise, in digital forensics identifying potential evidence is at the center of any investigation. Evidence identification is challenging where only executable code is available; for example, identifying the existence of malware at the memory of a system where there is no source code [48]. Fordroid is a fully automated forensics tool developed based on the STA approach [42].

Another security area where the STA approach is widely used is web application vulnerability analysis. Tripp et al., use an STA in the design and implementation of TAJ to analyze web application security vulnerability [49]. TAJ has later been improved into a more scalable and precise version called ACTARUS [50]. A method proposed by Kurniawan et al. detects web file injection vulnerability in web applications using a PHP parser to traverse abstract syntax trees of the source code [51], while the method uses source codes for web application vulnerability assessment [52]. F4F makes use of an augmented taint analysis engine that generates a web application's source code in a simple Web Application Framework Language (WAFL) [53]. Tripp et al. proposed the most popular Web application security analysis tool called ANDROMEDA [54].

Sources	Year	Tools	Security Focus area	Need Source Code	Used Platform	Automated/Manual	Specific Area
[47]	2018	SAINT	Data leak (Privacy)	YES	SmartThings/ OpenHAB/ Apple's HomeKit	automated	Commodity IoT
[41]	2018	Fordroid	digital forensics	YES	Android	automated	Android applications
[49]	2009	TAJ	Vulnerability analysis	YES	Java	automated	Web Applications
[50]	2011	ACTARUS	Vulnerability analysis	YES	Java	automated	Web Applications
[53]	2011	F4F	Vulnerability analysis	YES	Java	Manual	Web Applications
[54]	2013	ANDROMEDA	Vulnerability analysis	YES	Java	automated	Web Applications

TABLE 1: STA Tools.

Table 1 summarizes and compares the STA tools reviewed in this section. Most of the tools are vulnerability mining tools that require source code for the analysis.

4. DYNAMIC TAINT ANALYSIS TOOLS

There are a number of research areas where DTA has been used for solving security problems, including private data leak detection, application vulnerability detection, malware analysis, and forensics [17]. For example, several researchers presented a Privacy Scope approach that uses DTA to find application leaks [55]. The approach is believed to be accurate and efficient and is implemented at the user environment to help pinpoint information leaks even if the sensitive data is encrypted. This approach uses function call summaries to handle taint propagation to reduce the overhead of the information flow tracking. In addition, this approach uses on-demand instrumentation to enable fast loading and to be able to run on large applications to precisely track information. Different from TightLip [56] and Privacy Oracle [57], information leakage detecting tools that are limited to applications whose outputs only depend on inputs, Privacy Scope can trace multiple input data.

TaintEraser is another DTA tool proposed for the prevention of sensitive data leaks [58]. TaintEraser conducts its analysis at the application level to let off-the-shelf application users run their applications while preventing unwanted information exposure. Similarly, researchers implement the taint propagation within the kernel for a reduced overhead in tracking in which they try to achieve near-real-time analysis. TaintEraser uses on-demand instrumentation to enable fast loading of large applications, and a semantic-aware instruction-level tainting for increased accuracy. The tool is tested with Internet Explorer, Yahoo! Messenger, and Windows Notepad where it generated no false positives, precisely preventing user sensitive data that would have otherwise been leaked to unwanted channels [58]. TaintEraser uses PIN [58] as a dynamic binary translator to accomplish its application-level analysis. The tool supports a simple privacy policy whereby a user first specifies sensitive input data to monitor, and subsequently TaintEraser blocks any data derived from the sensitive input data from moving to output channels that are specified as restricted. In doing so, TaintEraser monitors applications with input data marked 'sensitive.' Once such applications are moving out of the network, TaintEraser would replace sensitive bytes in those applications with randomly chosen bytes [58].

Information flow tracking is one of the widely used information leak detection methods for smartphones. For instance, TaintDroid is a tool that provides Android smartphone users a means of testing whether third-party applications collect and share their private data [59]. TaintDroid uses a system-wide information flow tracking to analyze Android apps for data leakage. The system is a near-real-time tool and is capable of tracking multiple sources of sensitive data at one time. Researchers benchmarked their work with Android's Activity Manager. It is detected that Taintdroid adds 3% overhead. In addition, by employing Taintdroid to monitor the behavior of 30 Android apps, 68 instances of potential misuse of the users' private data were detected. At the time of its development, according to the authors, TaintDroid was the most effective and efficient privacy testing tool for Android apps [59]. In this light, Taindroid is the prime candidate tool that can help Android smartphone users make an informed use of third-party applications.

DTA is used for unknown vulnerability detection by looking for misuses of user input during a program execution [17]. Vigilante [60] is an end-to-end approach that collaboratively detects vulnerability at the end host. The tool runs instrumented software to detect worms at the host and broadcasts alerts upon the detection of one. Subsequently, once an alert is broadcasted the host automatically generates filters that would block infection of the suspected worm without blocking innocuous traffic. With the use of Vigilante, there is no need for trust between hosts because it uses a cooperative worm detection mechanism distributed all over the network, thereby making it hard for worms to evade from detectors. However, Vigilante requires hosts to run expensive detection engines that can spread highly accurate detection loads once a worm is detected over the network.

Lift [61] is another vulnerability detecting approach with a low-overhead information flow tracking mechanism. The tool is generic in the sense that it does not only target specific vulnerability exploits such as worm, buffer overflow, format string, etc. Rather, Lift is a software-only approach that exploits dynamic binary instrumentation and optimizations for detecting various types of security attacks. Likewise, Lift is more specific in selecting tag propagation paths because it eliminates unnecessary tracking, coalesces information checks, and efficiently switches between target programs and instrumented information flow tracking code. The tool is implemented on StarDBT [61], a dynamic binary translator, on Windows experimenting web applications from server and client sides. Compared to previous works, the tool shows relatively better results [61].

Newsome and Song propose another host-based DTA tool that automatically detects Format String and Overwrite attacks exploits on commodity software [62]. These researchers referred to their tool as TaintCheck. TaintCheck has been employed in testing a number of programs and turned out to not have false positives for any of the programs. Likewise, TaintCheck enables an automatic semantic analysis to generate a signature for attack filtering after an exploited attack has been detected.

Previous studies focused on the use of DTA for securing centralized software. However, implementing such tools to distributed systems have raised issues of applicability, tool portability and analysis scalability [63]. Hence, the development of dedicated DTA tools that can be used for distributed systems is sought to be necessary. DistTaint, an application-level dynamic taint analyzer, is proposed for this aspect [64, 65]. However, the tool uses Java source code for its analysis.

Some researchers have taken one step beyond and have tried to secure cloud computing with DTA tools. For example, Papagiannis and Pietzuch proposed CloudFilter [66], a DTA tool that allows a cloud consuming organization to have control of its sensitive data and not be leaked to the cloud without its consent. CloudFilter intercepts file transfers between the consumer organization and cloud services, and subsequently performs logging and enforces propagation policies. Similarly, the tool controls where files propagate after they have been uploaded to the cloud and ensure that only authorized users may gain access to them. The researchers successfully applied CloudFilter to Dropbox and GSS whereby they were able to control the data propagation [66].

CloudFence is another data flow tracking service model [67] that monitors data leaks in cloud services. Researchers propose the tool to be hosted by the cloud providers for consumers to independently audit their data residing in that same cloud. The tool can also give cloud brokering companies to confine the propagation of sensitive data of their customers within well-defined domains. CloudFence is based on runtime binary instrumentation that supports byte-level data tagging and uses PIN as a dynamic binary translator. Similarly, CloudFence enables fine-grained data tracking for up to four billion users. To evaluate the effectiveness and practicality of the tool, the researchers implemented a CloudFence prototype using two publicly disclosed data leakage vulnerabilities in two real-world applications. Compared to the DTA tools Libdft and SiteBar; CloudFence shows a runtime overhead which is comparable to that of Libdft and larger performance impact in comparison to SiteBar.

Some researchers have amazingly employed dynamic information flow tracking for forensics readiness purpose, where system call level logging is conducted in order to ease “after-the-myth” investigation of attacks. For instance in one of the latest developments of this aspect, researchers proposed Rain, an attack investigation system, which uses a record-replay technology to record system-call events during runtime [68]. The system has the ability to perform instruction-level DTA that can filter out processes unrelated to the case to minimize the number of processes to be investigated for attack causality accuracy. In previous works, for example, Xiao et al. proposes PoL-DFA, a forensic system that can log the execution traces of the processes being monitored for investigating applications data leakage and contamination. Likewise, Sun and Oliveira propose an IoT forensics framework DDIFT [70] that uses a DTA module running in the IoT system controlling a mobile device, a forensics analysis module running in the cloud, and distributed optimization to conduct a decentralized forensic analysis of IoT applets.

One of the research areas where the DTA approach is exhaustively used is in dynamic malware analysis. The use of DTA is preferred for malware analysis because it is not easily defeated by techniques such as obfuscation and polymorphism. In this paper, we will review some of the most popular Malware analysis tools developed based on the DTA approach. Some malware analysis tools such as *Panorama* [71] and *Ether* [72] use hardware instrumentation. These types of malware analysis tools are not in our scope and therefore were not studied in this paper. However, TQana is an internet explorer browser plug-in tool that uses binary instrumentation for the analysis of malicious codes [73]. TQana performs at the kernel level to monitor all calls made by the malware. It observes both the functional behavior and information traces of the malware execution. Whenever a URL is entered into the address bar of the internet explorer, TQana implements information flow tracking using the Navigate event of the web browser which in turn introduces taints to the system. Another binary instrumentation based malware analysis tool is Cloudtaint [74]. Cloudtaint uses elastic taint tracking based on data flow tracking as well as control flow for malware detection of cloud-based applications. One of the best-known analysis tools developed based on the DTA approach is Dyton [27]. Dyton uses PIN for binary instrumentation providing an API where its user can configure the source and the sink to track the control of the information flow.

Tables 2 (a) and (b) show a summary of the DTA based tools reviewed in this paper. In tables 2 (a) and (b), the sign (✓) shows the existence or use of the parameter, listed in the tables, by the tools. In cases where cells are left blank, the corresponding parameter is neither used nor discussed in the papers reporting about the tools.

5. COMPARATIVE DISCUSSION

In tables 2 (a) and (b), the tools were comparatively analyzed for their employment of certain parameters. For instance, starting from the left, the tools were evaluated based on their area of focus. Of the 15 DTA based tools reviewed in this paper, 5 were for analyzing data leaks. In the literature of DTA, some researchers were categorically referring DTA based tools as data privacy suitable tools. So no wonder that most of the reviewed tools are dedicated to data leak analysis. The application of DTA based tools towards digital forensics is now getting the momentum. Three of the 15 reviewed tools are developed for digital forensics. Starting from its early days the DTA approach was used for malware analysis. Some DTA tools are generic in a way that they are not specific for their implementation area. For example, tools such as Vigilante, Lift, TaintCheck, and DistTaint are in general for application vulnerability analysis.

The tools were also evaluated for the type of analysis techniques they followed. The four columns under the analysis techniques section of Table 2 (a) show that most tools explicitly follow the data flow tracking analysis method. A good Handful of the tools including, Vigilante, Lift, DistTaint, Rain, DDIFT, TQama, CloudTaint, and Dyton, use both dataflow and control-flow for their analysis. Usually, tools that track data at a fine-grained level have shown law performance compared to those performing tracking at a coarse-grained level. However, 9 out of the 15 tools reviewed have implemented tracking at data or process (fine-grained) level analysis. Likewise, in

Table 2 (a), the binary instrumentation tool used in each of the tools is depicted in the DBI tools column. Some the tools do not specify which binary instrumentation tool they employ. However, undeniable number of tools have used the most popular binary instrumentation tool PIN. That is, PIN is a good candidate for any prospected DTA tools.

In Table 2 (b), we evaluated the mode of the tools' implementations. Usually, DTA tools perform their analysis at the user or/and kernel levels of the operating system. Only 4 tools can conduct analysis at both user and kernel levels, while the remaining 11 tools do analysis at the user or kernel levels. In Table 2 (b), the soundness, precision, and performance of each of the tools are evaluated. We could hardly grab soundness of the tools because most of the researchers did not discuss in the relative papers. However, only have shown interest in indicating the soundness of the tools. We mostly based our evaluation on the literature, particularly what other researchers have said about the tools. As a result, most of fine-grained tools have shown high overhead. Furthermore, we have studied what kind of environment the model has been implemented. As indicated in the last two columns of Table 2 (b) most of the tools are implemented in virtualized environments.

Sources	*Year*	*Tools*	*Security Focus area*	*Analysis Technique*				*Taint Granularity*		DBI tool
				DataFlow	ControlFlow	Implicit	Explicit	Fine-grained	Coarse-grained	
[55]	2009	Privacy Scope	Data leak	✓			✓	✓		PIN
[58]	2011	TaintEraser	Data leak	✓			✓	✓		PIN
[46]	2014	TaintDroid	Data leak	✓		✓			✓	
[59]	2005	Vigilante	Vulnerability analysis	✓	✓				✓	Nirvana
[60]	2006	Lift	Vulnerability analysis	✓	✓		✓	✓		StarDBT
[61]	2005	TaintCheck	Vulnerability analysis		✓			✓		Valgrind
[63, 64]	2019	DistTaint	Vulnerability analysis	✓	✓	✓		✓		
[65]	2012	CloudFilter	Data leak	✓			✓		✓	
[66]	2013	CloudFence	Data leak	✓			✓	✓		PIN
[67]	2017	Rain	Digital Forensics	✓	✓			✓		PIN
[68]	2016	PoL-DFA	Digital Forensics	✓			✓	✓		ET-Analyzer
[69]	2018	DDIFT	Digital Forensics	✓	✓				✓	
[72]	2007	TQana	Malware analysis	✓	✓				✓	PIN
[73]	2014	Cloudtaint	Malware analysis	✓	✓			✓		PIN
[26]	2007	Dyton	Malware analysis	✓	✓				✓	PIN

TABLE 2 (a): Dynamic Taint Analysis Tools.

Sources	*Year*	*Tools*	*Security Focus area*	*Modes of Implementation*		*Challenges faced*			*Environment*	
				User	Kernel	Soundness	Precision	Overhead	Emulated	Virtual
[55]	2009	Privacy Scope	Data leak	✓	✓		High	Low		✓
[58]	2011	TaintEraser	Data leak		✓		High	Low		✓
[46]	2014	TaintDroid	Data leak	✓			High	High		✓
[59]	2005	Vigilante	Vulnerability analysis		✓	Low	High	High		✓
[60]	2006	Lift	Vulnerability analysis	✓	✓	High	High	Low		✓
[61]	2005	TaintCheck	Vulnerability analysis	✓		Low	High	High	✓	
[63, 64]	2019	DistTaint	Vulnerability analysis	✓	✓	Low	High	High		
[65]	2012	CloudFilter	Data leak	✓				High		
[66]	2013	CloudFence	Data leak	✓	✓	High	Low	High		✓
[67]	2017	Rain	Digital Forensics		✓	High	High	High	✓	
[68]	2016	PoL-DFA	Digital Forensics	✓			High	High		✓
[69]	2018	DDIFT	Digital Forensics	✓			High	Low		
[72]	2007	TQana	Malware analysis		✓			High	✓	
[73]	2014	Cloudtaint	Malware analysis		✓		High	High	✓	✓
[26]	2007	Dyton	Malware analysis	✓			High	High	✓	

TABLE 2 (b): Dynamic Taint Analysis Tools.

6. LESSONS LEARNED

Based on our review and current status of the DTA tools, it is believed that there is an urgent need of designing DTA based vulnerability analysis tools with a reduced false reporting rate. On the other hand, such tools may optimize the efficiency of the DTA by selectively controlling the number of taints to be spread for each analysis. This can be accomplished by removing unnecessary taints from the system.

In addition, the DTA tools in the literature can only detect some specific vulnerabilities. Hence, the development of a generic tool that combines existing techniques in order to detect myriad security vulnerabilities will be a value add to the domain. The literature is also lacking tools that can analyze inter-applications or inter-systems data leaks.

Adopting DTA to the analysis of cutting edge technology is also lagging behind. There are only a number of tools that have been applied to cloud computing and IoT environments. Worth mentioning is that none of these tools focused on the vulnerability analysis of cloud or IoT applications. Some focused on data leak detection while others were for either digital forensics or malware analysis. The primary reason why the vulnerability analysis DTA based tools are not extended to cloud and IoT technologies is because of the infancy of the two areas. Other reasons may include the heterogeneous nature of devices and applications involved in cloud and IoT technologies. Furthermore, how data is distributed, aggregated and processed in cloud and IoT technologies may pose challenges in the data flow tracking. Particularly, different types of IoT technologies, Operating systems, and network protocols from different vendors make it hard implementation of DTA tools to the IoT ecosystem.

6. CONCLUSION

In this paper, taint analysis tools have been studied. At first, different areas where the taint analysis approach is implemented are discussed. Subsequently, a brief overview of the STA and a number of tools that have been developed based on STA are presented. Likewise, the section about DTA is starting with the basics and definitions to consequently build on the description of the tools and frameworks in the literature. A number of DTA based tools are thoroughly reviewed. Their areas of implementation were studied together with the shortcomings reported in each of the tools. A deeper understanding of the DTA approach and the effective adaption of its tools will have an improving effect on software security analysis.

7. REFERENCES

[1] D Zou, J Zhao, W Li, Y Wu, W Qiang., "*A Multigranularity Forensics and Analysis Method on Privacy Leakage in Cloud Environment."* IEEE Internet of Things Journal, 2018. 6(2): p. 1484-1494.

[2] A.N. Moussa, N. Ithnin, and A. Zainal, "*CFaaS: bilaterally agreed evidence collection."* Journal of Cloud Computing, 2018. 7(1): p. 1.

[3] X. Meng, and B.P. Miller. "*Binary code is not easy*." in *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 2016. ACM.

[4] M. Shudrak, and V. Zolotarev. "*The technique of dynamic binary analysis and its application in the information security sphere*." in *Eurocon 2013*. 2013. IEEE.

[5] C Chen, B Cui, J Ma, R Wu, J Guo, W Liu. "*A systematic review of fuzzing techniques."* Computers & Security, 2018. 75: p. 118-137.

[6] R Baldoni, E Coppa, DC D'elia, C Demetrescu. "*A survey of symbolic execution techniques."* ACM Computing Surveys (CSUR), 2018. 51(3): p. 50.

[7] Z Feng, Z Wang, W Dong. "*Bintaint: A STA Method for Binary Vulnerability Mining*." in *2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB)*. 2018. IEEE.

[8] J Cai, P Zou, J Ma, J He. "*Sworddta: A dynamic taint analysis tool for software vulnerability detection."* Wuhan University Journal of Natural Sciences, 2016. 21(1): p. 10-20.

[9] K. Liu, H.B.K. Tan, and X. Chen, "*Binary code analysis.* Computer," 2013. 46(8): p. 60-68.

[10] C. Cadar, D. Dunbar, and D.R. Engler. "*KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs*." in *OSDI*. 2008.

[11] W. Aman, "*A framework for analysis and comparison of dynamic malware analysis tools."* arXiv preprint arXiv:1410.2131, 2014.

[12] J. Kim, T. Kim, and E.G. Im. "*Survey of dynamic taint analysis*." in *2014 4th IEEE International Conference on Network Infrastructure and Digital Content*. 2014. IEEE.

[13] E Zhu, X Li, F Liu, X Li, Z Ma. "*Constructing a hybrid taint analysis framework for diagnosing attacks on binary programs."* Journal of Computers, 2014. 9(3): p. 566-575.

[14] M Ahmad, V Costamagna, B Crispo "*TeICC: targeted execution of inter-component communications in Android*." in *Proceedings of the Symposium on Applied Computing*. 2017. ACM.

[15] M. Monga, R. Paleari, and E. Passerini. "*A hybrid analysis framework for detecting web application vulnerabilities*." in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Secure Systems*. 2009. IEEE Computer Society.

[16] A. Getman, V. Padaryan, and M. Solovyev. "*Combined approach to solving problems in binary code analysis"*. in *Proceedings of 9th International Conference on Computer Science and Information Technologies (CSIT'2013)*. 2013.

[17] P. Dai, Z. Pan, and Y. Li. "*A Review of Researching on Dynamic Taint Analysis Technique*." in *2018 3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018)*. 2018. Atlantis Press.

[18] S Chen, J Xu, N Nakka, Z Kalbarczyk. "*Defeating memory corruption attacks via pointer taintedness detection*." in *2005 International Conference on Dependable Systems and Networks (DSN'05)*. 2005. IEEE.

[19] GE Suh, JW Lee, D Zhang, S Devadas. "*Secure program execution via dynamic information flow tracking*." in *ACM Sigplan Notices*. 2004. ACM.

[20] G Venkataramani, I Doudalis, Y Solihin. "*Flexitaint: A programmable accelerator for dynamic taint propagation*." in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. 2008. IEEE.

[21] J Shin, H Zhang, J Lee, I Heo, YY "Chen *A hardware-based technique for efficient implicit information flow tracking*." in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2016. IEEE.

[22] VP Kemerlis, G Portokalidis, K Jee, AD Keromytis. "*libdft: Practical dynamic data flow tracking for commodity systems*." in *Acm Sigplan Notices*. 2012. ACM.

[23] W. Xu, S. Bhatkar, and R. Sekar. "*Taint-Enhanced Policy Enforcement: A Practical Approach to Defeat a Wide Range of Attacks*." in *USENIX Security Symposium*. 2006.

[24] V. Ganesh, T. Leek, and M. Rinard. "*Taint-based directed whitebox fuzzing*." in *Proceedings of the 31st International Conference on Software Engineering*. 2009. IEEE Computer Society.

[25] TR Leek, GZ Baker, RE Brown, MA Zhivich, "*Coverage maximization using dynamic taint tracing*." 2007, MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB.

[26] R Wang, G Xu, X Zeng, X Li, Z Feng *TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting.* Journal of Parallel and Distributed Computing, 2018. 118: p. 100-106.

[27] J. Clause, W. Li, and A. Orso. "*Dytan: a generic dynamic taint analysis framework*." in *Proceedings of the 2007 international symposium on Software testing and analysis*. 2007. ACM.

[28] G. Portokalidis, A. Slowinska, and H. Bos."*Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation*." in *ACM SIGOPS Operating Systems Review*. 2006. ACM.

[29] D Song, D Brumley, H Yin, J Caballero, I Jager "*BitBlaze: A new approach to computer security via binary analysis*." in *International Conference on Information Systems Security*. 2008. Springer.

[30] MG Kang, S McCamant, P Poosankam, D Song *Dta++: dynamic taint analysis with targeted control-flow propagation*. in *NDSS*. 2011.

[31] L Li, TF Bissyandé, M Papadakis, S Rasthofer. "*Static analysis of android apps: A systematic literature review."* Information and Software Technology, 2017. 88: p. 67-95.

[32] X Wang, R Ma, B Dou, Z Jian, H Chen, "*OFFDTAN: A New Approach of Offline Dynamic Taint Analysis for Binaries."* Security and Communication Networks, 2018. 2018.

[33] M Nunes, P Burnap, O Rana, P Reinecke, "*Getting to the root of the problem: A detailed comparison of kernel and user level data for dynamic malware analysis"* Journal of Information Security and Applications, 2019. 48: p. 102365.

[34] M Vassena, A Russo, D Garg, V Rajani, "*From fine-to coarse-grained dynamic information flow control and back."* Proceedings of the ACM on Programming Languages, 2019. 3(POPL): p. 76.

[35] H. Yin, and D. Song, "*Whole-system Fine-grained Taint Analysis for Automatic Malware Detection and Analysis."* Technical paper. College of William and Mary & Carnegie Mellon University, 2006.

[36] M Polino, A Continella, S Mariani, S D'Alessio *Measuring and defeating anti-instrumentation-equipped malware*. in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. 2017. Springer.

[37] D. Bruening, E. Duesterwald, and S. Amarasinghe. *Design and implementation of a dynamic optimization framework for Windows*. in *4th ACM Workshop on Feedback-Directed and Dynamic Optimization (FDDO-4)*. 2001.

[38] CK Luk, R Cohn, R Muth, H Patil, A Klauser. "*Pin: building customized program analysis tools with dynamic instrumentation*." in *Acm sigplan notices*. 2005. ACM.

[39] L.K. Yan, and H. Yin, "*SoK: On the Soundness and Precision of Dynamic Taint Analysis."*

[40] D. Boxler, and K.R. Walcott. *STA Tools to Detect Information Flows*. in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. 2018. The Steering Committee of The World Congress in Computer Science, Computer ….

[41] M. von Maltitz, C. Diekmann, and G. Carle. *Privacy Assessment Using STA (Tool Paper)*. in *International Conference on Formal Techniques for Distributed Objects, Components, and Systems*. 2017. Springer.

[42] X Lin, T Chen, T Zhu, K Yang, F Wei "*Automated forensic analysis of mobile applications on Android devices."* Digital Investigation, 2018. 26: p. S59-S66.

[43] Z Xing, Z Bin, F Chao, Z Quan "*Staticly Detect Stack Overflow Vulnerabilities with Taint Analysis*." in *ITM Web of Conferences*. 2016. EDP Sciences.

[44] C. Feng, and X. Zhang. *A Static Taint Detection Method for Stack Overflow Vulnerabilities in Binaries*. in *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*. 2017. IEEE.

[45] F. Pauck, and H. Wehrheim. *Together strong: cooperative Android app analysis*. in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019. ACM.

[46] S Arzt, S Rasthofer, C Fritz, E Bodden, A Bartel "*Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps*." in *Acm Sigplan Notices*. 2014. ACM.

[47] ZB Celik, L Babun, AK Sikder, H Aksu, G Tan "*Sensitive information tracking in commodity IoT*." in *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018.

[48] N. Rosenblum, X. Zhu, and B.P. Miller. *Who wrote this code? identifying the authors of program binaries*. in *European Symposium on Research in Computer Security*. 2011. Springer.

[49] O Tripp, M Pistoia, SJ Fink, M Sridharan, *TAJ: effective taint analysis of web applications.* ACM Sigplan Notices, 2009. 44(6): p. 87-97.

[50] S Guarnieri, M Pistoia, O Tripp, J Dolby *Saving the world wide web from vulnerable JavaScript*. in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. 2011. ACM.

[51] A Kurniawan, BS Abbas, A Trisetyarso *STA Traversal with Object Oriented Component for Web File Injection Vulnerability Pattern Detection.* Procedia Computer Science, 2018. 135: p. 596-605.

[52] M.L. Minsky, *Computation*. 1967: Prentice-Hall Englewood Cliffs.

[53] M Sridharan, S Artzi, M Pistoia, S Guarnieri *F4F: taint analysis of framework-based web applications*. in *ACM SIGPLAN Notices*. 2011. ACM.

[54] O Tripp, M Pistoia, P Cousot, R Cousot *Andromeda: Accurate and scalable security analysis of web applications*. in *International Conference on Fundamental Approaches to Software Engineering*. 2013. Springer.

[55] Y Zhu, J Jung, D Song, T Kohno, D Wetherall, *Privacy scope: A precise information flow tracking system for finding application leaks*. 2009, Citeseer.

[56] A.R. Yumerefendi,, B. Mickle, and L.P. Cox. *TightLip: Keeping Applications from Spilling the Beans*. in *NSDI*. 2007.

[57] J Jung, A Sheth, B Greenstein, D Wetherall "*Privacy oracle: a system for finding application leaks with black box differential testing*." in *Proceedings of the 15th ACM conference on Computer and communications security*. 2008. ACM.

[58] DY Zhu, J Jung, D Song, T Kohno, *TaintEraser: Protecting sensitive data leaks using application-level taint tracking.* ACM SIGOPS Operating Systems Review, 2011. 45(1): p. 142-154.

[59] W Enck, P Gilbert, S Han, V Tendulkar *TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones.* ACM Transactions on Computer Systems (TOCS), 2014. 32(2): p. 5.

[60] M Costa, J Crowcroft, M Castro, A Rowstron *Vigilante: End-to-end containment of internet worms*. in *ACM SIGOPS Operating Systems Review*. 2005. ACM.

[61] F Qin, C Wang, Z Li, H Kim, Y Zhou "*Lift: A low-overhead practical information flow tracking system for detecting security attacks*." in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. 2006. IEEE.

[62] J. Newsome, and D.X. Song. *Dynamic Taint Analysis for Automatic Detection, Analysis, and SignatureGeneration of Exploits on Commodity Software*. in *NDSS*. 2005. Citeseer.

[63] X Wang, H Ma, K Yang, H Liang "*An Uneven Distributed System for Dynamic Taint Analysis Framework*." in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*. 2015. IEEE.

[64] X. Fu, and H. Cai." *A dynamic taint analyzer for distributed systems*." in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019. ACM.

[65] X. Fu, "*On the scalable dynamic taint analysis for distributed systems*." in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2019. ACM.

[66] I. Papagiannis, and P. Pietzuch. "*Cloudfilter: practical control of sensitive data propagation to the cloud*." in *Proceedings of the 2012 ACM Workshop on Cloud computing security workshop*. 2012. ACM.

[67] V Pappas, VP Kemerlis, A Zavou *CloudFence: Data flow tracking as a cloud service*. in *International Workshop on Recent Advances in Intrusion Detection*. 2013. Springer.

[68] Y Ji, S Lee, E Downing, W Wang, M Fazzini "*Rain: Refinable attack investigation with on-demand inter-process information flow tracking*." in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017. ACM.

[69] G Xiao, J Wang, P Liu, J Ming, D Wu "*Program-object level data flow analysis with applications to data leakage and contamination forensics*." in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*. 2016. ACM.

[70] N. Sapountzis, R. Sun, and D. Oliveira. "*DDIFT: Decentralized Dynamic Information Flow Tracking for IoT Privacy and Security*." in *Workshop on Decentralized IoT Systems and Security (DISS)*. 2018.

[71] H Yin, D Song, M Egele, C Kruegel "*Panorama: capturing system-wide information flow for malware detection and analysis*." in *Proceedings of the 14th ACM conference on Computer and communications security*. 2007. ACM.

[72] A Dinaburg, P Royal, M Sharif, W Lee "*Ether: malware analysis via hardware virtualization extensions*." in *Proceedings of the 15th ACM conference on Computer and communications security*. 2008. ACM.

[73] M Egele, C Kruegel, E Kirda, H Yin, D Song. "*Dynamic spyware analysis."* 2007.

[74] J Yuan, W Qiang, H Jin, D Zou. "*CloudTaint: an elastic taint tracking framework for malware detection in the cloud."* The Journal of Supercomputing, 2014. 70(3): p. 1433-1450.

[75] Funnywei, “Bufer Overfow Vulnerability Mining Model [Z/OL],” 2003, http://xcon.xfocus.net/XCon2003/archives/ Xcon2003 funnywei.pdf.