Effective Dynamic Taint Analysis of Java Web Applications

Yan Huang[1](✉), Chaohui He[2], Chenglong He[1], and Chaoyong Wang[1]

[1] School of Software, Xi'an Jiao-Tong University, Xi'an, Shaanxi, China
hy1273383167@stu.xjtu.edu.cn, a3109405078@xjtu.edu.com
[2] School of Energy and Power Engineering, Xi'an Jiao-Tong University, Xi'an, Shaanxi, China
hechaohui@mail.xjtu.edu.cn

Abstract. With the rapid development of the Internet, network security is the most important issue for businesses and people. Vulnerabilities caused by user input and not treated harmlessly are the easiest to be exploited by hackers. In this paper, a tool named FastTaint is implemented, by using the principle of dynamic taint analysis, the vulnerability detection rate is high and the false positive rate is extremely low. First, the FastTaint tool is based on the proxy mode of behavior injection mode; then there are different instrumentation strategies for Source, Propagator, Sanitizer and Sink to make the detection range more accurate; finally, the taint is marked at the object level and the vulnerability is determined at the leaking point. The FastTaint tool abandons the traditional firewall that relies on the characteristics of requests to detect attacks and creatively uses Interactive Application Security Testing (IAST) technology. It is injected directly into the protected application's service to provide real-time, function-level protection, and can update the strategy without updating and detect or prevent unknown vulnerabilities without updating the protected application's code. Experiments show that this tool can quickly and efficiently detect multiple vulnerabilities without requiring the source code, FastTaint can detect multiple vulnerabilities, such as SQL Injection, Cross-Site Request Scripting, Path Traversal, Insecure Forwarding, XPath Injection, OS Injection, SSRF and other vulnerabilities.

Keywords: Dynamic Taint Analysis · Vulnerability Discovery · Computer Technology · Dataflow Analysis · Web Security

1 Introduction

At present, more and more common for people to contact and use the Internet. Most of the information and property of companies and individuals are stored on the Internet. Therefore, network security is the most important issue for enterprises and individuals. Taint analysis belongs to the category of data flow analysis. To detect information flow vulnerabilities, you need to determine the point of entry of the external input into the application. In the context of a web application, this is typically user input originating from a web form, but also includes, for example, HTTP headers, URL parameters, data, and cookies. In taint analysis, the identified entry point is called the taint source. The

D. Qiu et al. (Eds.): ICBBEM 2022, AHIS 5, pp. 987–997, 2023.
https://doi.org/10.2991/978-94-6463-030-5_97

source data is marked as tainted and how these tainted objects are analyzed and tracked throughout the application. Tainted objects are rarely kept only in the original marked objects, so they can propagate [9]. This means that the affected object is not the originally labeled one, which can happen directly or indirectly. For example, a contaminated string object is assigned in whole or in part to a new object.

The dynamic taint analysis (DTA) method analyzes the various execution paths in the application-specific runtime environment, traces the flow of data from the identified source to the sink, and controls how this analysis is performed. Static taint analysis is a method of analyzing the source code of an application. This means that this type of analysis can cover all possible execution paths, whereas dynamic taint analysis only covers the paths that are explicitly included in the analysis. Therefore, dynamic taint analysis today needs to focus on how to effectively and accurately analyze the quality of Java code.

In order to detect more types of vulnerabilities, reduce the overhead of program runtime and improve the efficiency of vulnerability detection, a tool called FastTaint is developed in this paper. FastTaint is used in the testing phase, has a very low false positive rate, high test coverage, real-time detection and in terms of vulnerability detection rate factors. In addition to the features related to the detection strategy provided by FastTaint, users can also customize strategies based on data flow. FastTaint also supports vulnerability detection in third-party components. FastTaint supports multiple web frameworks such as Spring, SpringBoot, Struts, MyBatis etc. and has detailed vulnerability details including queries, vulnerability types, vulnerability paths, leaked functions, and specific classes. FastTaint implements a code vulnerability detection system based on dynamic taint analysis. It is a non-intrusive target, is class-isolated, can be uninstalled at any time, is used by multiple users, and can detect multiple types of vulnerabilities. It can be used to create automated testing tools.

The contributions of this paper are as follows: (1) Using dynamic instrumentation technology to insert the call statement of taint propagation analysis library into the application code, realize the separation of business logic and taint propagation analysis, and realize the non-intrusive mechanism to support the hot deployment of the project. (2) The tracking of taint is precisely matched to the specific object storage area, and at the same time, the scope of taint data type processing is expanded to include not only character types, collection types, but also stream types and all object types that can be taint. (3) This paper can detect several types of vulnerabilities, including SQL injection, command injection, LDAP injection, third-party component collection, and vulnerability detection, weak encryption algorithms, weak hash algorithms, cross-site scripting attacks, cross-site request forgery, and other vulnerabilities. (4) It is possible to derive the propagation path of the taint in the application, i.e., the path that the taint travels from the source to the sink. Based on this path, the taint can be quickly localized. The tools developed in this paper can support a variety of frameworks. The application can be deployed and is very stable. It is suitable for large, medium and small enterprises to investigate vulnerabilities.

2 Related Works

In the context of static analysis, frameworks such as WALA [6], Doop [7], Soot [10], etc. all provide support for Java, and static analysis tool, such as Findbugs [4], requires source code and cannot analyze the data flow while the program is running. However dynamic taint analysis can analyze the taint information flow while the program is running. Currently, the analysis of Java applications using dynamic taint analysis is based on a modification of the operating system, such as the tool Phosphor [1] by Bell and Kaiser. Phosphor was developed using the Java bytecode manipulation library ASM framework. Phosphor uses shadow variables to solve the problem of pollution tracking of primitives and arrays. A shadow memory is a variable that maintains a pollution flag for unrecognizable objects. Shadow variables are injected into the application and placed next to each variable and array. Each method in the application is also checked to pass the shadow variable along with the unchecked object. Phosphor [1] does not support taint variable retention, making applications unsuitable for finding code injection vulnerabilities. Therefore Phosphor cannot detect security vulnerabilities in the data stream and create exploits. Phosphor requires a huge time and runtime overhead.

The third implementation of the dynamic taint tracker for Java applications is the Dynamic Security Taint Propagation tool [9], which is built on the AspectJ framework Java library and supports aspect-oriented programming in Java. However, this tool propagates only the pollution of the String, StringBuffer, and StringBuilder classes. The tool relies on aspect-oriented events that trigger taint propagation, taint data from taint sources, and a provision that ensures taint variables do not pass leaks. However, the tool does not support adding or deleting taint sources, leaks, and harmless treatment. The lack of this feature means that the availability of the application is not very high and it is prone to false negatives.

In addition, TaintDroid [12] is a popular dynamic taint analysis tool for android. TaintCheck [5], published by Newsome et al. of Valgrind, provides taint analysis of data streams to detect buffer overflow vulnerabilities. However, this tool has a large overhead and ignores control flow analysis. BitBlaze [8], DTA++ [3], WASP [11] and DECAF [2] try to achieve a combination of dynamic pollution analysis and symbolic execution, so they improve the path coverage of dynamic pollution analysis. Although the above techniques improve the performance of dynamic pollution analysis, there are still some problems that have not been solved permanently, including the high memory and runtime overhead.

3 Methodology

The definition of taint analysis can be abstracted in terms of triples <sources, sanitizers, sinks>, The process is shown in Fig. 1 where Sources is the source of taint, which is the direct introduction of untrusted data into the information system; Propagation is the propagation of taint, which is untrusted data in the program; Sanitizers is to verify the tainted object, so that the tainted object becomes credible data; Sinks is the leakage point, which means that the execution of the program here may cause data vulnerabilities.

The FastTaint model is shown in Fig. 2. First, various input functions for different Java web servers are collected, the data transmitted by the input functions are marked as

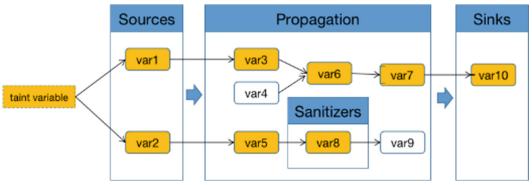

Fig. 1. An intuitive example about the process of taint analysis.

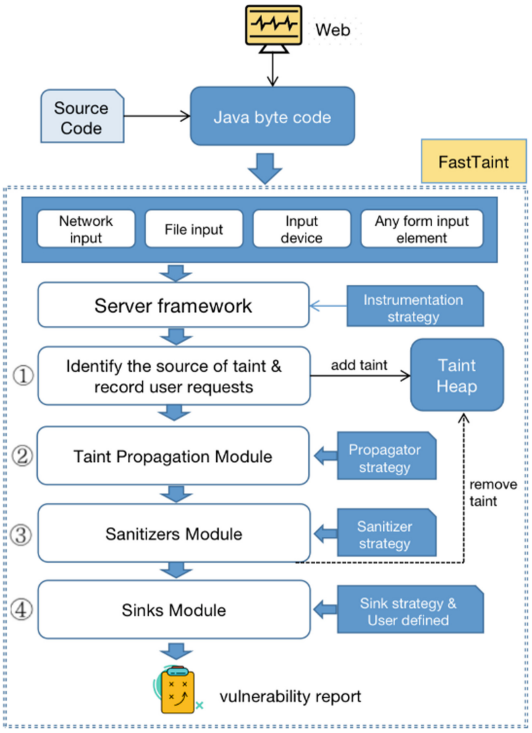

Fig. 2. Overall system architecture.

the source of taint, the hash code is calculated, and the objects are added to the taint heap; second, according to the taint propagation strategy, through the taint marking algorithm, the objects that propagate the taint in the program are added to the taint heap, and the taint propagation trajectory is recorded; according to the taint checking strategy provided by the framework or the user-defined taint checking strategy, remove the corresponding taint marker from the taint heap to reduce the memory consumption and improve the taint analysis efficiency; and finally according to the different leak point strategy of each framework, execute the taint checking strategy at the taint leak point. Thus, determine whether there is a vulnerability and issue a detailed report about the vulnerability.

3.1 Source Module

This article deals with the taint information that can be caused by mainstream servers and frameworks, and does extensive research on them. It collects all mainstream servers in the market, including Tomcat server, JBoss server, Wildfly server, Resin server, Jetty server, WebLogic server, WebShere server, Powerland BES server, and SpringBoot framework, and statistically analyzes the APIs of the above servers and frameworks, heuristically analyzes the input functions, and creates taint source files. The source points are classified into four categories based on the type of external input: external network input, file stream family input, database input, XML/JSON, and other file input. This article collects a total of 94 input functions.

Because Java types are numerous and complex, FastTaint decomposes complex objects, turns the complex objects into a single object to compute the object's address value, and then adds the address value to the taint heap. In the Java language, there are data structures such as integer, long, short, double, float, boolean, character, byte, map, set, list, array, etc., so in order to represent the taint accurately, for example, in an array only certain elements are tainted, so the entire array cannot be marked as tainted, which would result in excessive tainting. The FastTaint tool decomposes the array object into a single object and computes the hash code of the single object. Given a data structure like Map, we decompose it separately into Key and Value and assess whether Key and Value are suspicious data. If it is, we add the suspicious heap. Some of the Source methods in Table 1.

3.2 Taint Propagation Module

When developing the taint propagation strategy, be sure to consider the methods that can cause propagation in Java and some Java frameworks, such as some methods in the String, StringBuilder, StringBuffer, List, Map, and other classes. In total, 107 classes and 534 propagation functions are collected in this article. Some of the Propagation methods in Table 2.

Table 1. Some of the JavaWeb Source Methods

Source	Description
Tomcat.util.net.NioBufferHandler.getReadBuffer	Read the buffer of Apache server
org.apache.catalina,connector,Request.getStream	Return the input stream of a request
javax.servlet.http.Part.getHeaderNames	Gets the header names
Org.eclipse.jetty.server.Request.getReader	Shows a post form item
javax.servlet.hltp.HttpServletRequest.getHeader	Returns the value of the specified request header as a String
javax.servlet.http.HttpServletRequest.getCookies	Returns the Cookie object the client sent with this request
java.net.URLConnection.getInputStream	Gets the value of the parameter

Table 2. Some of the JavaWeb Propagation Methods

Propagation	Description
java.lang.String.valueOf	Returns the argument
java.lang.StringBuffer.append	Appends the argument to this sequence
java.net.HttpURLConnection.setRequestMethod	Set the method for the URL request
java.nio.file.Paths.get	Converts the given URI to a Path object
java.util.Collection.add	Ensures that this collection contains the specified element
javax.servlet.http.Http.HttpServletRequest.getAttribute	Returns the value of the named attribute
javax.xml.xpath.XPath.compile	Compile an XPath expression for later evaluation
org.springframework.jdbc.core.JdbcTemplate.queryForRowSet	Execute a query for an SqIRowSet, given static SQL

This paper is based on the taint propagation strategy, and there are whitelists and blacklists. Whitelist: Users can skip some functions to speed up the efficiency of instrumentation; blacklist can be used to introduce and eliminate taints in certain functions of the application. Once a request arrives in the application, it is first entered into the processing logic of the http node for tagging and preprocessing. The request enters the source point, the data information collected in the function by instrumentation is stored with event, placed in the taint path, and the result of the source is placed in the taint pool. When requesting to enter the propagator node again, determine whether the propagator node parameters are present in the taint pool according to the configuration, and if so, place the propagator node event in the taint path. When the application reaches the last sink point, it judges whether the parameter of the sink point is present in the taint pool according to the configuration of the sink point, and if so, the sink point is written to the taint path. If the program is called multiple times, it will re-enter multiple propagation nodes, and these nodes will also be included in the taint path. After the application is executed, it will return to the http node. When it finally executes to leave the http node, it calls graph generate function to construct the taint call graph and send it to the vulnerability assessment module.

3.3 Sanitizers Module

The Sanitizer module means that after this module processes the tainted data, the data itself will no longer contain sensitive information or the operations on the data will no longer cause harm to the system. In other words: After the corrupted data has passed through the harmless processing module, the “Taint” mark can be removed. Proper

Table 3. Some of the JavaWeb Sanitizer Methods

Sanitizer	Description
org.apache.commons.lang.StringEscapeUtils.escapeHtml	Escapes the characters in a String using HTML entities
org.sprirgframework.web.util.HtmlUtils.htmlEscape	Turn special characters into HTML character references
org.owasp.esapi.Encoder.encodcForHTML	Encode data for use in HTML using HTML entity encoding
com.ibatis.sqlmap.engine.mapping.result.XmlList.clear	Removes all of the elements from this collection
com.ibatis.sqlmap.engine.scope.BaseScope.reset	Clears all data out of the scope
java.sql.Statement.clearBatch	Empties this Statement object's current list of SQL commands
java.sql.Connection.prepareStatement	A new object containing the pre-compiled SQL statement
java.sql.PrepareStatement.cancel	Cancels this Statement object

application of harmless processing can reduce the number of stains in the system, improve the efficiency of taint analysis, and avoid the problem of inaccurate analysis results due to taint propagation.

In the Saniziter module, corresponding to the harmless strategy, the collection of this strategy also takes a lot of work in this paper. The custom and harmless processing functions provided by the framework accelerate the operational efficiency of the system, reduce the memory consumption through the single instance mode and minimize the redundancy of the system. Through the observer mode, the function in the harmless location will execute the data information through the stake and depending on the stain object in the stain heap when the stain label is removed. This paper collects a total of 16 classes and 101 functions provided by the framework, and users can add harmless functions themselves in the configuration file. Some of the Sanitizer methods are shown in Table 3.

3.4 Sinks Module

This paper specifies a large number of leak-point methods involving security issues. Care should be taken when defining leak-point methods, otherwise fasttaint will contaminate a large number of data streams that are not necessarily related to actual attacks. There is no need to keep track of redundant data streams, which would add more overhead for users and runtime performance. In addition to standard Java libraries and mainstream servers, this paper also extracts sink methods from popular frameworks (such as Apache Struts and Spring Boots). In this paper, 92 classes and 782 sink functions are collected.

On the leak point event, get the data in the data information event obtained by inserting the stack, build the sink model instance, and then tentatively identify whether

Table 4. Some of the JavaWeb Sink Methods

Sink	Description
java.lang.Runtime.exec	Executes the string command in a process
java.sql.Statement.executeQuery	Executes a given SQL statement with JDBC
java.net.URL.openConnection	Returns an HttpURLConnection object
javax.Servlet.jsp.JspWriter.println	Prints objects/strings/booleans on web documents
org.apache.struts.action.ActionForward.setPath	Sets URI to which control should be forwarded
org.hibernate.Session.createSQLQuery	Executes a given SQL statement in Hibernate
tomcat.util.net.NioBufferHeadler.getReaderBuffer	Reads the Buffer of Apache server

the sink point data has a redirection vulnerability or a SQL overpower vulnerability. First, get the parameter array contained in the methodvent, get the parameter object contained in it, and then set the cookie. If the current request is a login logic, create an initial cookie and send the cookie to the file with the vulnerability. If it is not login logic, store the cookie parameters in the global variable. Determine if the vulnerability in the sink object is a SQL injection vulnerability. If it is a SQL injection vulnerability, find out which database it is and then set the JDBC object to determine if it is login logic. Check if there is a login string by matching the string URI. If so, send a SQL ultra vires report. Finally, release the JDBC object you just set to avoid memory leaks.

To extract these methods, we used our experience and vulnerabilities reported in Common Vulnerabilities and Exposures (CVE). Table 4 shows some of a summary of the specified sink methods for each project in our benchmark.

3.5 The Advantages of FastTaint

In order to represent the advantages of FastTaint in comparison with related work, we compared the critical features in our approach with the essential contributions in related work; the result of this comparison is shown in Table 5. According to our comparison results, the number of reported true and false positives indicates that our dynamic taint analysis is more effective in detecting actual vulnerable tainted flows and pruning false-positive flows that do not involve runtime attacks and vulnerabilities present in the code. These precise results gained by accessing the actual runtime information, controlling the content of introduced untrusted inputs from our analysis.

Table 5. Comparison Between the Advancements in FastTaint and Related Work.

	FastTaint	FindBugs	Phosphor	TaintDroid	WASP
Bytecode Instrumentation	✓	✖	✓	✓	✓
Real-world case studies	✓	✓	✖	✓	–
Compatibility on standard JVMs	✓	–	–	–	✖
Object propagation	✓	–	✓	✓	–
Supporting reflection	✓	✓	✖	✓	–
Attack detection	✓	✓	✖	✖	–
False-positive filtration	✓	✖	✖	✖	–
Adaptable source specification	✓	✖	✖	✖	–

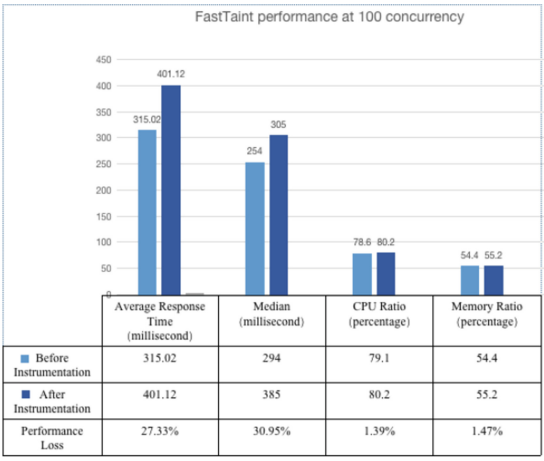

Fig. 3. FirstTaint performance at 100 concurrency.

4 Experiment

To test the effectiveness of the vulnerability detection model proposed in this paper, an experimental environment was created on the Mac platform. The experimental environment is the Mac operating system with 8 cores and 16 GB of physical memory. The development environment is based on tools such as JDK 8, ASM, and idea, and includes 12689 lines of code, including more than 2000 lines of the source point function assessment module, 5300 lines of code of the stain propagation module, the sanitizer module, and 6600 lines of the taint propagation module and the dynamic instrumentation module.

This paper uses a variety of benchmark suite tests, such as openrasp1.3.1, bodgeit1.4.0 and webgoat5.7 selected for the test environment. The test environment contains web applications with different vulnerability types. It can be seen from Table 6 and Fig. 3 that FastTaint can detect a variety of vulnerabilities and has a high detection

Table 6. FastTaint's Test Result

Benchmark			
Vulnerability	*openrasp*	*webgoat*	*bodgeit*
PathTraversal	✓	✓	–
OSCiommandInject	✓	✓	–
CrossSiteCripting	✓	✓	✓
BlindSQLInjection	✓	✓	✓
NumericSQLInjection	✓	✓	✓
StringSQLInjection	✓	✓	✓
WeakSessionCookies	–	✓	–
LDAPInjection	✓	✓	–
X Pathlnjaction	✓	✓	–
UnvalidatedForward	✓	–	–
SSRF	✓	–	✓
XXE	✓	✓	–
AccessControlFlaws	–	✓	–
Load Library	✓	–	–
InSecureCommunication	–	✓	–
FailOpenAuthentication	–	✓	–
DebugCode	–	–	✖
InsecureObjectReference	–	–	✖

accuracy rate. Figure 3 evaluates the functional and performance indicators of the prototype system, and shows the performance of FastTaint when the concurrency is 100, the test case of this paper has been open sourced: https://github.com/HQiXuan/FastTaintBenchmark. Figure 3 shows the superiority and high performance of this tool. When FastTaint increases in concurrency, the performance impact on CPU and Memory is about 3%.

5 Conclusion

This paper implements a dynamic analysis tool named FastTaint for JavaWeb application vulnerability detection. The goal was that the tool would combat integrity and confidentiality vulnerabilities. The results of the conducted evaluations show improved security when using FastTaint.

However, the tool in this article still has some drawbacks: FastTaint cannot mark primitives data type propagation and can only be used for JavaWeb. In further research, the detection scheme for the application's internal context data will cover more types

of vulnerabilities and try to achieve irregular vulnerability detection based on machine learning algorithms.

References

1. Bell, J., & Kaiser, G.E. (2015). Dynamic taint tracking for Java with phosphor (demo). Proceedings of the 2015 International Symposium on Software Testing and Analysis.
2. Henderson, A.: DECAF: a platform-neutral whole-system dynamic binary analysis platform. IEEE Trans. Softw. Eng. 43(2), 164–184 (2017)
3. Kang M G, McCamant S, Poosankam P, et al. Dta++: dynamic taint analysis with targeted control-flow propagation[C]//NDSS. 2011.
4. Nathaniel Ayewah, William Pugh, David Hovemeyer, J David Morgenthaler, and John Penix. Using static analysis to find bugs. IEEE software, 25(5):22–29, 2008.
5. Newsome J, Song D X. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software[C]//NDSS. 2005, 5: 3–4.
6. Stephen J. Fink et al. [n. d.]. T.J. Watson Libraries for Analysis (WALA). http://wala.sourceforge.net.
7. Smaragdakis Y, Bravenboer M. Using Datalog for fast and easy program analysis[M]//Datalog Reloaded.[S.l.]: Springer, 2011:245–251.
8. Song D, Brumley D, Yin H, et al. BitBlaze: A new approach to computer security via binary analysis[C]//International Conference on Information Systems Security. Springer, Berlin, Heidelberg, 2008: 1–25.
9. V. Haldar, D. Chandra and M. Franz, "Dynamic taint propagation for Java, " 21st Annual Computer Security Applications Conference (ACSAC'05), 2005, p. 9 pp.–311, doi: https://doi.org/10.1109/CSAC.2005.21.
10. Vallée-Rai R, Co P, Gagnon E, et al. Soot: A Java bytecode optimization framework[C]//CASCON First Decade High Impact Papers. .[S.l.]: [s.n.] , 2010:214–224.
11. W. Halfond, A. Orso and P. Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation, " in IEEE Transactions on Software Engineering, vol. 34, no. 1, pp. 65–81, Jan.-Feb. 2008, doi: https://doi.org/10.1109/TSE.2007.70748.
12. William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, ByungGon Chun, Landon PCox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS), 32(2):5, 2014.