Data-flow Based Analysis of Java Bytecode Vulnerability

Gang Zhao
Department of Computer science and Technology
Tsinghua University
zhao-g03@mails.tsinghua.edu.cn

Hua Chen
Dongxia Wang
Beijing Institute of System Engineering
chh@public.bise.ac.cn
wdx@public.bise.ac.cn

Abstract

Java is widely used because its security and platform independence. Although Java's security model is designed for protecting users from untrusted sources, Java's security is not under fully control at the application level. A large number of Java classes or Java class libraries have been used in network 应用 application development, whose source is unknown and trust unassured. Analyzing the vulnerability of Java bytecode is helpful for assessing the security of untrusted Java components. The data-flow based methods suit to vulnerability analysis because their data propagation character. The paper is about using data-flow based methods to analyze the vulnerability of Java program in bytecode.

1. Introduction

Java is widely used because its security and platform independence. By the popular application of J2EE, network servers developed with Java are spread everywhere. The security of Java comes to public attention then.

Java is a secure language for which the security has been taken into account in its design. Java provides special security mechanism to keep program from network exploiting. Even though there are still security risks in it. The disclosed security flaws of JAVA program showed that Java's security mechanism might lost control in some cases. The typical problem is command injection [1]. The input of program can be passed directly to some risk inner operation as parameters. The operation could be fully controlled by some skillful input. The behavior of program loses its way as a result. The system security is damaged then.

Vulnerability is about a program's defects or errors, which may be used to damage the confidentiality, integrality, and availability of the program as well as the whole system the program stays in. It means the vulnerability is about the usability of the program's flaw from space out of program. If there is a path that could pass outer data to some risk point of a program, there is vulnerability in. To analyze the vulnerability of a program means to evaluate the possibility and possible effect that the external data could affect the program.

Data-flow analysis is a method to provide the global information about how a program deals with its data. It could expose the influence and propagation of data in a program. With the help of data-flow analysis, we can know the way outer data operates on the program. It can be used to estimate the possibility the program may be exploited. The information is helpful to system security assurance.

Java program converts to Java bytecode by Java compiler. Java bytecode is the base of Java platform independence. The bytecode makes Java to compile once and execute anywhere. Java helped to promote the components technology and open source development. Large numbers of Java classes and Java class libraries with bytecode form have been widely used in network software development. Their sources were unknown and trusty unassured. Only analyzing the vulnerability for source code could not solve the whole problem. There is a definitely need to analyzing the vulnerability of Java bytecode. It will help to evaluate the security of untrusted Java component and its composite system. The analysis can support the Java security assurance.

In this paper we present a framework to use data-flow technique analyzing the vulnerability of Java bytecode program. We give a model of data-flow based vulnerability analysis and discuss how to use it in Java bytecode program analysis. The implementation is discussed then.

2. Vulnerability and program input

978-0-7695-3185-4/08 $25.00 © 2008 IEEE
DOI 10.1109/WAIM.2008.99

Vulnerability is a defect or error in the program which could result system security fault. A security fault is deferent from general program fault because it could be exploited. That means the vulnerability analysis concerns about flaws that may be used to control the behaviors of a program. There are two premises for a program to be exploited. First, there is some operation with security risk in the program. Second, external data could control the operation to transform the risk to a real security fault. An exploited program could be evil by intentional controlling, or just annoy by unintentional.

The program input includes all the data the program received from external. It includes user input from GUI, data from net or database, environment variables and so on. Only the input creates channels between outer and a program. It will be the sources of vulnerability if not processed properly. If there is a way to control a program with unintended manner, it must by input. According to the 2007 annual report of OWASP, the ten most critical web application security vulnerabilities [2] are more or less input dependent. It is obvious that there is a direct relationship between the external input and vulnerabilities in patterns such as XSS, Injection Flaws, Malicious File Execution and so on.

Many static vulnerability analysis tools search signatures in a program, which are some lexical patterns from known security faults. Their results cannot explain the relationship between the signature and the security fault so are difficult to understand. The analysis provides a rough view of program's security risk with high false alarm. On the other hand, there is not an efficient way to extract new signatures. So it is difficult to find new vulnerability in a program.

The input is a good place to begin the vulnerability analysis. By tracing the propagating of input data in a program, we could find the vulnerability points and understand how they exploited. The data-flow analysis will help.

3. Data-flow based vulnerability analysis

3.1. The data-flow based analysis model

Set a program as Prg. S is a point in Prg and E is another point. There is an operation named O_e in S which accepts data D_e from external. E includes an operation named O_{cia} which deals with confidentiality, integrity and availability of Prg. E is a security risk point of Prg, named as vulnerable point E_v. If O_{cia} at E_v has some parameter which is use of D_e, then the control flow path P from S to E is a vulnerability propagating path named as P_v.

The task of vulnerability analysis is to find all of the E_v in the program and deduce the related P_v. The result of a vulnerability analysis is a set of triple $<S_e, E_v, P_v>$. The D_e is gotten into program at S_e. D_e affects E_v through P_v. The S_e is defined as vulnerability propagation start point. The E_v is as vulnerability propagation end point.

D_e is defined by input operation of the program language O_e, which relies on the language input semantics. E_v is defined by secure operation O_{cia} of the system. O_{cia} depends on secure operation semantics and security policies of the system.

Data-flow is a static program analysis method. It is widely used to analyze the mode a program operates on its data. It is helpful to analyze the influence the data brings to the program as well. Data-flow analysis produces the information about data propagating in a program. The data-flow equation of a sentence is as below:

$$Out[S]=Gen[S] \cup (In[S]-Kill[S])$$

The definition of Gen() and Kill() is closely related to the application background. Gen() is about operations accept external data and Kill() stop the propagation of external data. In() is all the D_e reaches S before executing of S.

Following steps bellow to analyze vulnerabilities of a program by data-flow analysis:

(1) Create the syntax signatures of O_e and D_e. They are used to locate the start points by syntax or lexical analysis.
(2) Create the syntax signatures of Ocia and Ev. They are used to locate the end points by syntax or lexical analysis.
(3) Get the information of Pv with data-flow analysis.
(4) Analyze operations on De in Pv for each Pv. The information of relationship about Ev and De.
(5) Analyze the influence that Ev may give the program and system according to the definition of Ocia. Evaluate the possibility that Ev transforms to security fault on the relation between Ev and De.

3.2 The start point of vulnerability propagation

Any syntax element with input semantics should be included into the start point of vulnerability propagation. The input objects may be data, messages, events and even signals. So the parameters, input sentences of the language, operation on receiving events and messages will be taken as candidates.

From the viewpoint of syntax investigation, to check the signatures of vulnerabilities propagation means checking input relative operations which include parameters of main program, input sentences such as get and gets, file or net read operations,

environment variables read, network read and operations catching messages and signals.

As a secure rule, the information with higher trust should not be passed to lower trusty object to keep confidentiality of a system. So some input operations about inner sensitive information must be taken into account.

3.2. The end point of vulnerability propagation

All of the operations affecting system's confidentiality, integrity and availability are candidates of the vulnerability propagation end point. The semantics of security are complicated. The behaviors consisted a security operation are related to some concrete system security policies. The security operation's definition relies on application context.

In general, following operations are more danger:

Output operations: various writing operations. It includes writing to file, net, message and queue. There are two sides of writing: controlling the path or content. The former creates opportunity to write into a secure area of system, or write some sensitive data to unprivileged area. The latter can write some noisy data into specific file to destroy its integrity. It is often the first step to control writing to attack a system.

Process management: It includes operations create process and thread, commands start up remote server or local utility. The key to damage a system is to control the executed image. Whether the executable object could be controlled by maliciously is the most important.

Operations on sensitive data: For sensitive data accessed by a program such as password, key and some privilege object, attention should be paid to the operations in its vulnerability propagation path. There is a possibility that the sensitive data is propagated degraded.

Resources allocating operations: All the operations deal with resources allocating are included. The resources can be memory, disk, signal time and CPU time, etc. The consuming denial of service depends on controlling the space and time of a program.

Operations halt executing: There are usually some design errors which stop the system's executing. It includes calculating error, running error, incorrect exception subprogram. The halt error at the end point of vulnerability propagation path means there is a possibility to affect the executing path of a program. Contrived entering these points can result halt denial service.

Operations related to security policy and security mechanism: It depends on the specific environment of the system. These operations vary with the security level and application context. The Java native method is a typical sample.

3.3. The data-flow based vulnerability analysis framework

To analyze vulnerability with data-flow analysis, the most important factors are data process and data propagation characteristics of the program. They are decided by different components of a program. We divide the analysis into five layers as Figure1.

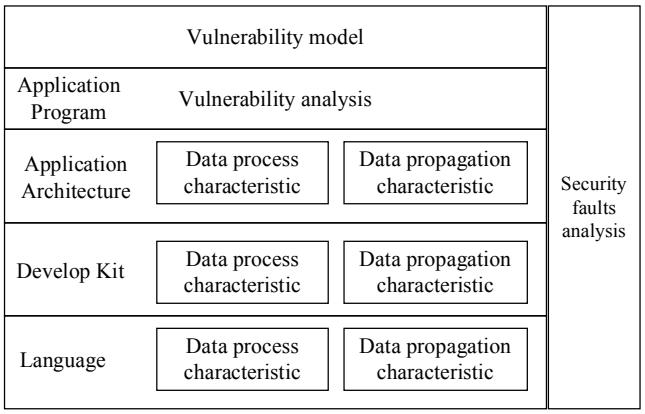

Figure1. Framework of vulnerability analysis

(1) The program language. Every program language has its special abilities to process and propagating data., which depend on the data types, control mechanisms and sentence.

(2) The develop platform. The data process and propagating characteristics of operation system, development kits, third part libraries and so on. They are about the security sensitive operations such as file I/O, password accessing and command executing.

(3) Application architecture. The architecture affects the relationship between the components of the application. It raises special data propagating effect of application and induces lots of data operations.

(4) Application program. The programmers have developing habits to make the programs more art than technique. Many program faults come from bad programming. The pattern's data characteristics are helpful to vulnerability analysis.

(5) Above all the program developing supporting artifacts, we can analysis the model of vulnerability in a synthesized view. The model can help find more vulnerabilities with more efficiency.

The framework can support to analyze programs with any language or mixed languages.

4. Data-flow analysis on Java bytecode

Java programs consist of Java classes. Class includes variables and methods. There are two ways to propagate data in Java program: passing parameters to a method or operating on a variable of class.

The Java class file is a binary file format for Java programs. Each Java class file represents a complete description of one Java class or interface. One class or interface is converted into a single class file. To analysis a Java program means to deal with a set of class files [3].

4.1. The hierarchy of Java analysis

The data-flow analysis of Java program is divided into two hierarchies: local and global. The data-flow within a method is due to the local analysis. On the other hand, the global analysis deduces the data-flow between methods. The local analysis is the base of global. The whole program's vulnerability propagation path will be clear only when the vulnerability propagation path of each method is clear.

The vulnerability analysis of a method deduces a data-flow propagating snapshot about the very method $M<S_e, E_v, P_v>$. The S_e can be one of following kinds: (1) message parameters of the method; (2) global fields in a class or package; (3) locals access an object out of the method such as file, net and database. The snapshot shows a method's data propagation ability, which is used to deduce the data propagation between methods.

The vulnerability propagation between methods is based on data-flow propagating snapshot. Vulnerability propagates from method A to B means there is a data propagation path between A and B. There must be a point which both in a inner vulnerability propagation path of A and B. To analyze a vulnerability propagation of inter methods is to find the intersections of P_v in analyzed methods. If the intersections of caller method and callee method are not empty, there is a vulnerability propagation path from caller to callee.

4.2. The bytecode of a method

Binary class file consists of the instructions flow of the Java virtual machine (JVM). The function of program is realized by JVM to interpret the JVM instructions. So the analysis of bytecode means the analysis the stream of JVM instructions.

Binary class file includes below information of a method:

```
ByteCodeMethod {
        String AccessModifier;
        String ClassName;
String[] ParemetersTable；
        int MaxStack；
        int MaxLocals;
        int CodeLength;
        JVMInstructions[] InstructionsStream;
ExceptionTable[] ExceptionTableItems；
LineNumberTable[] LineNumberTableItems;
LocalVariableTable[] LocalVariableTableItems;
}
```

It can be get from a class file that the modifier and name of method, the possible size of method stack, the numbers of local variable belong to a method and the instructions stream of the method and so on.

MaxStack is used to number the temporary variables in a method. MaxLocals is used to number the local variables of the method. They are useful to build the flow graph during data-flow analysis [4].

4.3. Control flow of bytecode

Constructing the basic block of a method is the start step of a control flow analysis in a method.

Four cases should be taken into account for Java bytecode instruction: the branch instruction of JVM, exception, finally clauses and implied exception

When Java source code transforms to bytecode the branch sentences become JVM branch instructions such as conditional branching instructions, unconditional branching instructions and conditional branching with tables. The JVM instruction throwing exception is athrow. The table index is at top of stack before athrow. The catch can be found with table index. Finally clause likes a miniature subroutine within a method which means a control branch.

It should be noticed that there are various JVM instructions which could raise exception impliedly. These instructions must be included in analysis.

Below is rule to deduce the leader of a basic in bytecode control flow analysis: The first instruction of a method, the label or offset of a branching instruction, the first instruction after the branching instruction, the catch clause of exception and the Finally clause and ret instruction.

The call graph of methods can be produced to search following instructions: invokevirtual, invokestatic and return instructions.

4.4. Data-flow of bytecode

JVM is a stack based virtual machine. The parameters, temporary results, local variables, return value and so on are saved in a data structure named Java Stack or Stack Frame.

The instructions relate to data operations are divided into five classes: stack and local variable operations, type conversion, integer arithmetic, logic and floating-point arithmetic. The dada definition of bytecode is defined as below: (1) Temporary variable definition. Each operation pushes data to stack. It includes constant push, local variable push, constant pool index push, object reference push and binary operation about stack top with result push. The local variable operations produce temporary variable definition, too. Another temporary variable definition comes from logic and floating-point arithmetic. (2) Local variables definition. The pop to local variables produces it. (3) Define temporary and local variables at the same time: arithmetic instruction with constants and local variables which push result to stack.

4.5. The start and end point of vulnerability propagation

During vulnerability analysis of bytecode, the I/O operations relate to the start point include: (1) The parameters in a parameter table of a method. (2) The I/O operations of the Java platform, which are various API of Java class library such as Java.io、Java.awt、Java.net、Java.io、 Javax.Servlet、Javax.Servlet.http and so on. (3) The native methods about input.

For the end of vulnerability propagation path, follow below rules: (1) The output parameters of a method. (2) Public variables of the class. (3) I/O API of output in Java class library as above. (4) API creates process and thread. (5) Synchronizing API.(6) The native methods about output.

The native methods should be emphasized because it can bypass the JVM's security mechanism. It became the most important causation of Java security fault.

5. Implementation

Build an auto vulnerability analysis tool could improve the analyzing efficiency and make it easy. There are some core functions in a data-flow based Java bytecode vulnerability analysis system.

5.1. Vulnerability Signature Database

The vulnerability signature is the description of some characteristics of vulnerability. The signatures should represent the syntax or lexical feature to support automatic analyzing. There a lot of tools to do so.

Although there are general rules to define a start and end point. The concrete vulnerability signature depends on the characteristics of program language. Different language possesses different signatures.

Below are the main items in the vulnerability signature database:

(1) Definitions of the vulnerability propagation start points. It includes the name, identity, lexical sign, syntax sign, and description and so on.

(2) Definitions of the vulnerability propagation end points. It is about name, identity, lexical sign, syntax sign, description, etc.

(3) Definition of vulnerability propagation effect. It relies on the data processing functions and their operation semantics in the specific language. The propagating capability and efficiency will be used to evaluate the possible security damage of the system. It includes naming of data process function, identity, the characteristic of propagation, description.

(4) The degree of data definition could influence its use is determined by propagating capability. For copy propagation, the definition may be passed to its use directly. It has a definite propagating effect. On the other hand, the reference propagation only has a possible influence. There is an obvious distinctness between them.

5.2. Parser

Parsing the language elements from object is the base of analysis. Every item useful for vulnerability analysis should be recognized from original file. It is transformed to suitable form for further analysis then. Besides, parsing engine produces the statistical information of symbols, which includes variables name, its attribute, API, system call, methods, native methods and so on, to help understand the program.

There are many ready-made tools to parse a program. They provide lexical analysis. The key is to define a security symbol table in addition to language syntax symbol table.

5.3. Signatures Checker

The signatures checker checks the vulnerability signatures in the parsing result according to the signatures database above. Its main function is to extract the start point of data-flow analysis.

The checker investigates the start or end point of a vulnerability propagation. When set off from a vulnerability propagation start point, the forward data-

flow analysis is used to infer the reaching end point. When set off from a end point of vulnerability propagation, it is the backward data-flow analysis is used to find out the live start point.

The checker records the position information of interested points. Information about method name, the number and type of parameters should be included.

Simple signature checker can be implemented with the help of lexical analysis tools. It could be combined with the implementing of parsing engine. To analyze behavior signature needs the model check technique. It could provide more complicated signature extracting.

5.3. Data-flow Analysis Engine

The engine provides various data-flow analysis functions, such as control flow analysis, data-flow analysis and inters method data-flow analysis.

Control flow analysis aims at local analysis of a single method. It extracts the basic blocks of a method, produces its control flow graph. The data-flow provides analysis for method, such as live variables analysis, reaching-definitions analysis, upwards-exposed uses analysis and other analysis methods.

Inter methods analysis aims at the whole program. It creates call graphs to show the global view of all of the methods. Side effect of the method could be considered to promote the quality of analysis.

In addition to above components, there are other components which a vulnerability analysis system should include.

The analysis report lists all of the results from analysis. The interim result which is helpful to understand the analysis could be provided. Some statistics of analysis, which are about parsing, vulnerable signatures, could be output by graphic interface to make it easy. The descriptions of the vulnerability signature in the database will promote the result.

The analysis workflow management makes the analysis procedure customized. The analysis schedule mechanism could be built to realize it.

The architecture of a data-flow based vulnerability analysis system is as Figure 2.

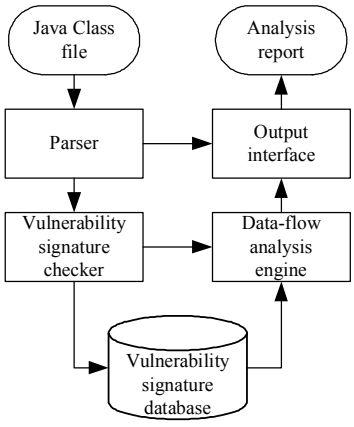

Figure 2. Architecture of the system

6. Conclusion

The vulnerability analysis is effective to evaluate the security risk of software. It is the base of security assurance technique to support intrusion detection, security assessment, security test and other activities to promote system security.

Data-flow analysis is a technique to expose the global influence of data in a program. Because of the input dependency of vulnerability, the vulnerability analysis needs to understand how the generalized input could propagate in the program. The data-flow analysis is a great help to it. There are studies to use data-flow analysis for the vulnerability analysis. There are vulnerability analysis tools based on data-flow analysis. But the architecture of the system has not been addressed.

The paper discusses the characteristics of the software vulnerability. The relationship of the vulnerability analysis and data-flow analysis is investigated. There is a suggestion about the framework of a data-flow based analysis system. Its implementation aims to Java bytecode is brought out.

It will be helpful to design and develop a data-flow based vulnerability analysis system.

7. References

[1] Ciera Nicole Christopher, “Analysis of Software Artifacts Evaluating Static Analysis Frameworks”, Carnegie Mellon University, May 10, 2006

[2] Open Web Application Security Project, The ten most critical web application security vulnerabilities, 2007 Update, Available: http://www.owasp.org

[3] Bill Venners, “Inside the Java virtual Machine”, The McGraw-Hill Companies, August 25, 1997

[4] Steven S. Muchnick, “Advanced Compiler Design and Implementation”, 1997

[5] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. “Compilers: Principles,Techniques, and Tools”, Addison-Wesley, 1986.

[6] Zhao Jianjun， ”Analyzing Comtrol Flow in Java”, Department of Computer Science Engineering, Fukuoka Institute of Technology, 1990

[7] Michael Martin, Benjamin Livshits, Monica S. Lam, “Finding Application Errors and Security Flaws Using PQL: a Program Query Language”, Computer Science Department, Stanford University, 2005

[8] Vivek Haldar, Deepak Chandra, Michael Franz, “Dynamic Taint Propagation for Java”, University of California, 2005

[9] V. Benjamin Livshits, Monica S. Lam , “Finding Security Vulnerabilities in Java Applications with Static Analysis” ,Computer Science Department, Stanford University, 2005
[10] Chris Anley., “Advanced SQL injection in SQL Server applications.”, 2002, Available:http://www.nextgenss.com.
[11] David Grove, Craig Chambers, “An Assessment of Call Graph Construction Algorithms”, IBM T.J. Watson Research Center, 2000
[12] Ana Milanova, Babara G. Ryder, “Parameterized Object Sensitivity for Points-to Analysis for Java”, Rensselaer Polytechnic Institute, Ohio State University
[13] Ondˇrej Lhot´ak , “Comparing Call Graphs”, David R. Cheriton School of Computer Science, University of Waterloo, 2007