Data-flow Based Vulnerability Analysis and Java Bytecode

HUA CHEN, TAO ZOU, and DONGXIA WANG
Beijing Institute of System Engineering, P.O.Box 9702-19, Beijing, PRC China 100101

Abstract: - The security of information systems has been the focus because of network applications. Vulnerability analysis is widely used to evaluate the security of a system to assure system security. With the help of vulnerability analysis, the security risk of a system can be predicted so that the countermeasures are arranged in advance. These will promote system security effectively. The object of vulnerability analysis is to find out the unknown security holes in a system. It could be helpful to understand the characteristics of security holes and to assess the security risk of a system. Data-flow based analysis shows its predominance in vulnerability analysis because the vulnerability is data-flow dependent. The paper discusses how to use data-flow analysis in vulnerability analysis. The way to apply data-flow analysis in Java bytecode vulnerability analyzing is presented.

Key-Words: - program analysis, vulnerability, Java bytecode, data-flow analysis

1 Introduction

The network applications make the information system security focus research area of all the world. The software security, which is the key of information system security, is widely studied because of its importance. The software development faces a number of challenges such as expanding size, compressed time and growing complexity. As a result, there are errors and weakness left in software system inevitably. These errors and weakness are the main sources raising security faults.

There are a lot of security assurance techniques that could be used to improve the security capability of a system to assure its security. Vulnerability analysis is one of the most important of them. By vulnerability analysis, the security state of the system could be analyzed. This will help to arrange the necessary countermeasures in advance to avoid possible risk.

Vulnerability analysis aims to find out the unknown security weakness in a system. It analyzes the security threats of a system, finds out the defects of the system and investigates the ways to exploit the defects to damage the system. It is the basis of many other assurance techniques.

The current vulnerability analysis techniques make use of the known security faults. By reverse engineering and faults injection, the new attack methods are created and tested. The methods are case to case in mass. Most of the resulted methods are suitable with some specific vulnerability pattern. The buffer overflow is a classic example with tools, such as Metasploit [1], to support exploiting. The main problems of vulnerability analysis techniques are unstructured, manual skill and inefficient. The result of analysis depends on analyzer's experience greatly. It is demanded to study new techniques for vulnerability analysis which could promote analysis capability and quality to help manage a more secure system.

Java is a secure language for the security had been taken into account in its original design. Java sets up special security mechanism to keep program from network exploiting but there are still some problems in Java security. Because of its security management hierarchy and granularity, some application security problems are not fully controlled by Java security mechanism. The typical problem is command injection [1]. The input out of a program can be passed directly to some risk inner operation as parameters. So the operation could be manipulated by some skillful input. As a result, the behavior of program is managed and the system security is destroyed.

Vulnerability is the program's defects or errors which may be used to spoil the confidentiality, integrality and availability of the program as well as the whole system. This means the vulnerability needs to be used by outer program. If there is a path to pass external data to some risk point, there is vulnerability in the program. To analyze the vulnerability of a program is to evaluate the possibility and possible effect that the external data could influence the inner program.

Data-flow analysis is a method to provide the global information about how a program processes its data. It could expose the influence and propagation of some data in a program. With the help of data-flow analysis, the way how external data propagates in a program will be clear. This can

be used to estimate the possibility the program may be exploited. The information is helpful to system security assurance.

Java program transforms to bytecode by Java compiler. Java bytecode is the base of Java platform independence. It is the bytecode that makes Java program to be executed anywhere. Java promotes the component technology and open source development. A lot of java classes and java class libraries with forms as bytecode have been widely used in network software development. Their sources were unknown and trust were unassured. Analyzing the vulnerability only with source code could not solve the whole problem. There is a definitely need to analyzing the vulnerability in java bytecode. It will help to evaluate the security of an untrusty java component or system. The bytecode vulnerability analysis can contribute to java security assurance. There are some characteristics of bytecode vulnerability analysis. First, the analysis without Java source code will expand the scope of java components under analysis. Second, java is difficult in program analysis because its object oriented characteristic. Bytecode analysis makes the program analysis at some a low language level. This simplifies the problem in a sense. Third, the size of the program will expand greatly when converted from source to bytecode. The space and time of analysis will be larger.

The paper discusses how to use data-flow analysis to analyze vulnerability. Problems to apply data-flow analysis in Java bytecode vulnerability analysis are presented.

2 The Input Dependency of Vulnerability

2.1 Vulnerability Depends on Program Input

Vulnerability is some defect or error in the program which could result system security fault. A security fault is different from general program fault because it should be exploited. There are two conditions for a program to be exploitable. First, there is some operation with security risk in the program. Second, an outer input of the program could influence the operation to violate the security rule of system to raise a security fault. An exploited program could be evil by unintentional or intentional design. Both should satisfy the preconditions above.

Only when the security risk operation in a program could be controlled by some external methods, the outer force of the program can make use of it to violate the system's security policies. For a program, accepting outer input data is the unique way to accept external controlling. So a vulnerable program inevitably depends on its input to be converted to fault.

2.2 Typical Security Vulnerabilities with Input Dependency

According to the 2007 report of OWASP, the ten most critical web application security vulnerabilities are [2]:

(A1) Cross Site Scripting (XSS)
(A2) Injection Flaws
(A3) Malicious File Execution
(A4) Insecure Direct Object Reference
(A5) Cross Site Request Forgery (CSRF)
(A6) Information Leakage and Improper Error Handling
(A7) Broken Authentication and Session Management
(A8) Insecure Cryptographic Storage
(A9) Insecure Communications
(A10) Failure to Restrict URL Access

The above web application security vulnerabilities are typical input dependent. There is a direct relationship between external input and vulnerability with A1, A2, A3, A4, A5, A8 and A9. By tracing the propagating path of input data in the program, it will be assessed how the fault point could be exploited. Though error handling is independent of input, there is a close connection from the input data and the event throwing the exception. Analyzing the mode that input affects the exception events will help to expose the vulnerability. "Denial of service" is another famous vulnerability. It can be divided into two classes: resources consuming and stop running. There are some operations abnormally allocate the system resources in the first class. It leads to system resources exhaustion, both space and time. On the other hand, there is some conditions that make program or system halted in second class, such as system halt errors and dead locks. Both classes could depend on external input of program. And only when the denial of service can be raised by external input of program, it belongs to vulnerability. The information leakage means sensitive information propagated to insecure area. Although is not input dependent, it is necessary to track the sensitive data in the program to catch the problem.

Insecure configuration is another common vulnerability. It includes ill-suited environment or debug mechanism left open after deployment and so on. They are related to generalized input of a program. These problems depend on input as well.

3 Data-flow Based Vulnerability Analysis

3.1 The Principle and Framework

Suppose there is a program 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 could be impacted by D_e in some way, then the control flow path P from S to E is a vulnerability propagating path named as P_v.

The goal 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 obtained into program at S_e. D_e affects E_v through P_v. The S_e is a vulnerability propagation start point. The E_v is a vulnerability propagation end point.

D_e is determined by input operations O_e in the language and platform, which is defined by the language input syntax. E_v is determined by secure operations O_{cia} of the system. O_{cia} depends not only on secure operations defined by general security rules, but also security policies of the application and system.

Data-flow is a static program analysis method. It is widely used to analyze the mode a program impacting on its data. It can also contribute to analyzing the influence which the data gives to the program as well. Data-flow analysis creates and solves the data-flow equation at each point of program. The information of the program's data-flow is gathered then. The data-flow equation of a sentence is :

Out[S]=Gen[S]□(In[S]-Kill[S])

The definition of Gen() and Kill() is closely related to the application background. Gen() is operations accepting external data and Kill() terminating the propagation of it. In() are all the D_e reach S before executing of S.

Follow steps below to analyze vulnerabilities of a program with 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 O_{cia} and E_v. They are used to locate the end points by syntax or lexical analysis.

(3) With located D_e and E_v, find out all possible P between them by data-flow analysis.

(4) For each P, analyze operations refer to D_e to deduce relationships between E_v and D_e and evaluate if the E_v could be affected by D_e.

(5) Investigate the influence that E_v may give to the program and system according to the definition of O_{cia}. Evaluate the possibility that E_v transforms to security fault on the relation between E_v and D_e.

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 inputted 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 view of syntax analysis, to check the signatures of vulnerabilities propagation means checking input relative operations which include parameters of main program, input sentences as get and gets, file 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 trusted object to keep confidentiality of a system. So some input operations about inner sensitive information must be taken into account.

3.3 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:

(1) Output operations: various writing operations. It includes writing to some objects like file, net, message and queue. There are two sides of writing poison: 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.

(2) 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 maliciously is the most important.

(3) 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.

(4) 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.

(5) Operations halting execution: There are usually some design errors which will stop the system's execution. It includes calculating error, running error, incorrect exception subprogram and so on. 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.

(6) 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.

3.4 The Data-flow Methods for Vulnerability Analysis

There are various methods could be used to analyze the data-flow in a program. Some are more specific for vulnerability analysis. The methods which are more dedicated to relation between variables are more suitable. They will effectively expose how external data affect the vulnerable points.

(1) Reaching-definitions analysis

Reaching-definitions analyze the definition of a variable that could reach some use of this variable.

Regard a variable gained external data as a definition of the variable. Vulnerability analysis traces the propagation of this definition. All of its uses will be analyzed to investigate if there is a vulnerable point in the found uses.

While analyze vulnerability with reaching-definition, to search out all of D_e in the program is the first step. Then all the uses of each D_e are analyzed to find out if there is some E_v according to O_{cia} in the use sentence. The control-flow paths from D_e to E_v consist the P_v. The set of $<S_e, E_v, P_v>$ is produced.

(2) Live variables analysis

Live variables analysis investigates if a specific point is a variable's use, which is in the path start from the variable.

Vulnerability analysis finds out the variables gain input from the external. For each control-flow path sets out from the found variables, the vulnerable points are evaluated. If the variable is livable at the vulnerable point, there is a security risk. All the D_e of the program is identified first. Then the set of live variables is calculated. If there is an O_{cia} that one of its parameters is alive, there is an E_v in the program. The P_v is the control-flow path from D_e to E_v.

(3) Upwards-exposed uses analysis

It is deducted by upwards-exposed uses analysis which uses could be reached by special definition for a given variable and its uses.

For a risk sentence, it is inferred whether some variables could be reached by some objects external. To find out all the O_{cia} in the program is the first step. Deduce all the definitions of the parameters of O_{cia}. If there is some D_e in the definition, there is a vulnerable point.

4 Data-flow Analysis of 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 in the 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 basis of the global. The whole program's vulnerability propagation paths will be clear only when the vulnerability propagation paths of each method is clear.

The vulnerability analysis of a method proceduces a set of $<S_e, E_v, P_v>$ of this method $M<S_e, E_v, P_v>$. The S_e can be further divided into two parts. First, the points gain data out of method by parameters of the method and variables. Second, other vulnerability propagation start points. The former can be used to deduce the data propagation between methods. It is a process different with the latter. The result of a method analysis is a set of

quaternion <$S1_e$, $S2_e$ E_v, P_v>. The S1 and S2 are distinctly related to the first and second type of S_e.

The vulnerability propagation between methods is based on inner method analysis. Vulnerability propagates from method A to B means there is a data propagation path between A and B. So there must be a point which belongs to an inner vulnerability propagation path of A and B simultaneously. To analyze a vulnerability propagation of inter methods becomes 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 from the 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 to analyze 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;
}
```

A class file includes the modifier and name of method, the possible size of method stack; the numbers of local variable belong to a method, the instructions stream of the method and so on.

MaxStack is used to record the numbers 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 the instruction athrow. The table index is at top of stack before athrow. The relative catch can be found with table index. Finally clause is 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 are rules to deduce the leader of a basic block in bytecode control flow analysis:

(1) The first instruction of a method.
(2) The label or offset of a branching instruction.
(3) The first instruction after the branching instruction.
(4) The catch clause of exception.
(5) The finally clause and ret instruction.

The call graph branch between methods can be produced to search following instructions:

(1) invokevirtual.
(2) invokestatic.
(3) return.

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.
- Local variable operations.
- Logic and floating-point arithmetic.

(2) Local variables definition: the pop to local variables.

(3) Defining the temporary variables 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 the Vulnerability Propagation

During vulnerability analysis of bytecode, the I/O operations relate to the start points 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, Javax.Servlet, Javax.Servlet.http and so on.

(3) The native methods about input, which are included in JNI interface and AWT native interface

To find the end points of vulnerability propagation in bytecode, 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) Process and thread API.

(5) Synchronizing API.

(6) The native methods about output.

The native methods should be emphasized because it can bypass the JVM's security mechanism. They have become the most important causations of Java security faults.

5 Implementation

Build an auto vulnerability analysis tool could improve the analyzing efficiency and make things 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 software 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 should be included in the vulnerability signature database:

(1) Definitions of the vulnerability propagation start points. It includes the name, identity, lexical sign, syntax sign, and relative 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 effects. 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 name of data process function, identity, the characteristic of propagation, some description.

The degree that a data definition could influence its uses is determined by propagating capability. For copy propagation, the definition may be passed to its use directly. It will be 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 the binary file is the basis of analysis. Every item useful for vulnerability analysis should be recognized from original file before analysis. It is transformed to suitable form for further process 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 binary program. They provide lexical analysis. The key is to define a security symbol table in addition to language syntax symbol table.

5.3 Signature Checker

The signature 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 points of 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 an 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 all 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. The model check techniques are necessary to analyze behavior signatures. They could provide more complicated signature extracting.

5.4 Data-flow Analysis Engine

The engine provides various data-flow analysis functions, such as control flow analysys, 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 of inner 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 be included.

The analysis report outputs all the results analysis. The interim result to help understand the state could be provided. Some statistics of analysis, which are about parsing, vulnerable signatures, could be output by graphic interface to make it easy accepted. The descriptions of the vulnerability signature in the database will make the result more understandable.

The analysis workflow management could make the analysis procedure configurable. The analysis schedule mechanism could be built to custom the procedure.

The architecture of a data-flow based vulnerability analysis system could be as Fig 1.

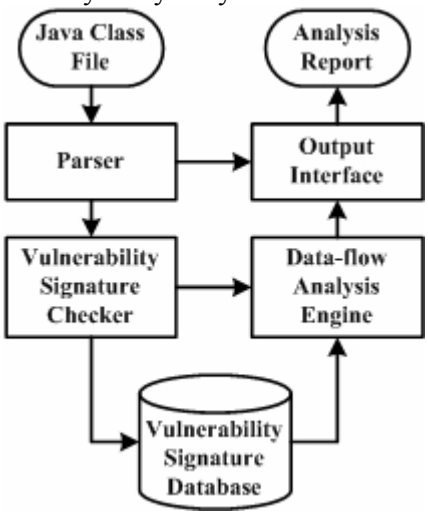

Fig. 1: System architecture

6 Conclusion

The vulnerability analysis is useful to evaluate the security risk of software. It is an important 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 will 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. Some vulnerability analysis tools based on data-flow analysis have been used. But the architecture of the system has not been addressed clearly.

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.

References:

[1] Metasploit Project, Available: http://www.metasploit.com/

[2] Ciera Nicole Christopher, "Analysis of Software Artifacts Evaluating Static Analysis Frameworks", Carnegie Mellon University, May 10, 2006

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

[4] Bill Venners, "Inside the Java virtual Machine", The McGraw-Hill Companies, August 25, 1997

[5] Steven S. Muchnick, "Advanced Compiler Design and Implementation", 1997

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

[7] Zhao Jianjun, "Analyzing Comtrol Flow in Java", Department of Computer Science Engineering, Fukuoka Institute of Technology, 1990.

[8] 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

[9] Vivek Haldar, Deepak Chandra, Michael Franz, "Dynamic Taint Propagation for Java", University of California, 2005

[10] V. Benjamin Livshits, Monica S. Lam , "Finding Security Vulnerabilities in Java Applications with Static Analysis" ,Computer Science Department, Stanford University, 2005

[11] Chris Anley., "Advanced SQL injection in SQL Server applications.", 2002, Available:http://www.nextgenss.com.

[12] Klaus Havelund, "Java PathExplorer - A Runtime Verification Tool", Kestrel Technology, NASA Ames Research Center, 2001

[13] Azadeh Farzan, Feng Chen, Jos´e Meseguer, Grigore Ro¸su, "Formal Analysis of Java Programs in JavaFAN", Department of Computer Science, University of Illinois at Urbana-Champaign, 2004.

[14] Kendra June Kratkiewicz, "Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code", Harvard University, March 2005.

[15] Paul E. Black , "SAMATE's Contribution to Information Assurance", National Institute of