Qualitative and Quantitative Analysis of Information Leakage in Java Source Code

Bo Chen Da-wei Xu Ling Yu
Department of Computer Science
Nanjing Normal University
Nanjing 210097, P.R.China
bchen@njnu.edu.cn

***Abstract*—Java is a kind of type-safe language, it introduces access control mechanism into bytecode and application layer, so as to guarantee the system resource and running environment avoid the invasion of the malicious code. However, in some information systems, information leakage is not due to the faultiness of the security model, but the absence of the information flow control policy and implementation of that in the source code. So, it is necessary to analyze how information leaks through the source code. This paper surveys information leakage in Java source code by qualitative analysis, and after defining conditional information entropy of the variables, quantitative analysis of information-leak in code is given. Language-based software security researches, new direction in the development of high trusted software, are introduced finally.**

Keywords- information leakage; covert channel; entropy; Java; source code; software security

I. INTRODUCTION

In one form or another, protecting programs is the core of computer security [1], because programs constitute so much of a computing system, such as the operating system, device drivers, database management systems, the network infrastructure and other applications, etc. In this paper, we call all these pieces of source code "programs".

In our opinion, for now, there are 3 key issues in program security:

- Nonmalicious program errors: buffer overflows, incomplete access control.
- Viruses and other malicious code: viruses, worms, Trojan horses.
- Covert channels: programs that leak information.

The first two issues have been studied and new researches unfolded [2-5]. Using type-safe languages, such as Java and .Net [6-8], is an important method as far as possible to eliminate the program errors.

Covert channels represent a real threat to secrecy in information systems. Security researchers have worked diligently to develop techniques for eliminating covert channels. The closure results have been bothersome. In ordinarily open environments, there is essentially no control over the subversion of a service program, nor is there an effective way of screening such programs for covert channels. And other than in a few very high security systems, operating systems cannot control the flow of information from a covert channel [1].

It is very important to prevent inappropriate information leakage from programs that handle sensitive data. Sensitive data should not be printed out and must be invisible to ordinary users. The values seen by the users with non-privileged access right must be non-sensitive data.

The conventional security mechanisms such as access control models employed today are inadequate in protecting confidentiality of data in information systems. Conventional security mechanisms focus on the controlling of the entities of information systems, such as file, procedure, etc. However, no restrictions are placed on the variables in source code.

In order to prevent information leakage in information systems, we need to answer two important questions:

- How the information leaks from programs?
- How much information leaks from programs?

This paper surveys information leakage in Java source code by qualitative analysis and applies information entropy theory to quantitatively analyze information leakage.

The rest of this paper is organized as follows. In section 2, we introduce Java conventional security mechanisms. In section 3, we do some qualitative analysis of information-leak, and point out that although Java has provided many security mechanisms, it does not avoid from information-leak through the covert channels in java programs. In section 4, we give quantitative analysis of information-leak in Java source code. Section 5 introduces language-based software security researches in response to software security challenges.

II. CONVENTIONAL SECURITY MECHANISMS OF JAVA

Java is a kind of simple, strong, secure, and dynamic programming language, and it is irrelevant to the platform. Java provides many security mechanisms [9,10]. It examines the errors in the program during the compiling. At the same time, Java is a kind of strongly-typed language, and the type checking in it is even stricter than C++. The type checking helps us to examine many errors occurred in the early development. Meanwhile, Java is in charge of the memory management also, and it provides a recycle mechanism of the garbage memory, which can effectively avoid the problem of memory leakage in C++. In addition, the security of Java can be ensured in the following two aspects: firstly, in the Java programming language, pointers and memory release are removed, which are used in C++. In this way, we can avoid the illegal operations to memory. Also, both the use of access control for the variables and the methods within the classes, and the use of bound checking for string

978-0-7695-3610-1/09 $25.00 © 2009 IEEE
DOI 10.1109/NSWCTC.2009.317

and array are introduced. On the other hand, Java adopts security architecture to ensure the security of the Java code. When we download the Java code from the Internet, and execute it locally, the security architecture of Java will make sure that the malicious code won't access our local computer resources optionally. For example, the operations of accessing local network resources and file deletion are both prohibited.

III. QUALITATIVE ANALYSIS OF INFORMATION LEAKAGE IN SOURCE CODE

Covert channels have been defined for the first time by Lampson in 1973[11]. He pointed out that in a trusted information system, even when all unauthorized accesses have been prevented, the information-leak might occur. A covert channel allows a process to transfer information in a manner that violates the system's security policy.

There are two kinds of covert channel: covert storage channel and covert timing channel. A covert storage channel uses an attribute of the shared resource. A covert timing channel uses a temporal or ordering relationship among accesses to a shared resource. This paper primarily focuses on storage channels.

Despite that Java has provided some security mechanisms, which can help the programmers write secure code, it does not control the information flow and can't prevent information leaking through the covert storage channels.

Java introduces access control mechanisms into bytecode and application layer, so as to guarantee the system resource and running environment out of the invasion of the exoteric malicious code. However, in some information systems, the information leakage is not because of the faultiness of the security model, but the absence of information flow control policy and implementation of that in the source code.

For the programmers have absolute rights in the process of programming, they may achieve some malicious purposes without changing the framework of the program. The simple program segment as following:

```
if (H==1)
  L=0;
else
  L=1;
```

If the information stored in variable *H* is confidential, and the information stored in variable *L* is public, after the execution of this conditional statement, we can obtain much information about variable *H* by looking into the value of variable *L*. Especially, if *H* is a boolean variable, we can deduce its value absolutely.

In addition, pay attention to the following program segment:

```
class Patient
{ private int hiv;
  private int age;
  private int num;
  private string name;
  public Patient(int h,int i,int j, string s)
        {hiv=h;age=i;num=j;name=s;}
  public int get_age(){return age;}
  public int get_num(){return num;}
  public int get_hiv(){return hiv;}
  public string get_name(){return name;}
}
class Doctor
{public string name;
  public int code;
  public void GetPatientInf(Patient p)  //(1)
  {
    FileOutputStream  L_Table;
    L_Table=new
FileOutputStream("patienttable.txt");
    int age,num;
    string name;
    age=p.get_age();
    num=p.get_num();
    name=p.get_name();
    L_Table.write(age);
    L_Table.write(num);
    L_Table.write(name);
  }
}
class HIV_Doctor extends Doctor
{ public void GetHivInformation (Patient p)
  {     FileOutputStream  H_Table;
        H_Table=new FileOutStream("hivtable.txt");
        ……//the same as Doctor:: GetPatientInf()
        int h=p.get_hiv();
   H_Table.write(h);
  }
}
class MAIN
{ public static void main(String[] args)
  { int code;
    Patient  pa=new Patient ();
    System.out.println ("Enter no:");
    code=System.in.read();
    if (code==1)    //ordinary section office
      { Doctor doc=new Doctor ();
        doc.GetPatientInf (pa);}
    if (code==2)    //AIDS section office
      { HIV_Doctor h_doc=new HIVDoctor ();
        h_doc.GetHivInformation(pa);}
  }
}
```

In the program segment above, class *Patient* records some patients' information, and member variable *age* denotes age, member variable *num* denotes the number of treatments, member variable *hiv* denotes whether is the AIDS patient or not, the value of '1' and '0' stands for "yes" and "no" separately, member variable *name* denotes patient's name.

Class *Doctor* denotes ordinary doctor, and member variable *name* denotes the doctor's name. Similarly, class *HIV_Doctor* denotes the doctors working in AIDS department. According to the access control policy, only the doctors working in AIDS department can obtain the confidential information that whether a certain patient is an AIDS patient or not, and can record this sensitive

information into the medical history "hivtable.txt"; Nevertheless, the ordinary doctors can only read the common patients' information, e.g. age, name, the number of treatments, and record these information into the medical history "patienttable.txt" (described in function *main* ()).

Whereas, if the malicious programmers inserted the member function as following into class *Patient* deliberately:

```
public boolean copy (int m, int n)
{ int z;
  z=hiv;
  n=-1;
  boolean flag;
  while(z==1)
    {  n=n+1;
       if(n==0)   z=m;
       else z=0;  }
 if(m==n)     flag=true;
 else     flag=false;
return flag;
}
```

At the same time, add some code as shown at label (2) into the function *GetPatientInf()* in class *Doctor*:

```
public void GetPatientInf (Patient p)
{
  ……//the same as Doctor:: GetPatientInf()
  if (p.copy (1, 0))   //(2)
    { age=1;L_Table.write(age);}
}
```

From the code above, we can know that the object of class *Doctor* can't call the function *get_hiv*() to read the patients' hiv information directly according to the access control policy. However, it can use the function *copy*() and deduce the patients' hiv information according to the execution result of this function. That is, if the function *copy*() returns a value of "true", we can deduce that the value of *hiv* is '1'. Meanwhile, we can assign '1' to a variable (such as *age*), and record it into medical history.

In view of the example above, we can know that although the proper access control policies are introduced into the function *main*() during the design of programming, the conventional Java is still lack of security information flow control policy. So, the problem of information leakage resulted from covert channels appears. That is to say, there is a information flow from *hiv* to *n*, which denotes from security to publicity in function *copy(m,n)*. Whereas, the existence of the information flow, such as function *copy(m,n)*, causes much harm.

From the example of program above, we can find out that information leakage in source code really exist. However, how to calculate the leakage of sensitive information in source code? In the next section, we will use information entropy theory to analyze the amount of sensitive information that may be leaked from programs written in Java.

IV. Quantitative Analysis Of Information Leakage In Java Program

A. The Concept of Statements Information-Leak Quantities

We start with recalling Shannon's basic definition, then give the definition of statements' information-leak quantities [12-14].

Definition 1. (The Variables' Information Entropy)

A random variable in a finite set X is a variable $x_i(i=1,2,\cdots,n)$ taking values in X and equipped with a probability distribution. $P(X=x_i)$ is the probability that X takes the value x_i, then the entropy of variable X is:

$$H(X) = -\sum_{i=1}^{n} P(X = x_i) \log_2 P(X = x_i)$$

Information entropy is a measure of the information content of variable X. According to the definition of information entropy, we can learn that the larger entropy is, the more uncertainty the variable X will be.

Definition 2. (The Variables' Conditional Information Entropy)

The conditional information entropy of variable X given variable Y is:

$$H(X \mid Y) = -\sum_{i=1}^{n} P(Y = y_i) \sum_{j=1}^{m} P(X = x_j \mid Y = y_i) \log_2 (P(X = x_j \mid Y = y_i))$$

Definition 3. (The Quantities of the Statements Information-Leak)

Suppose that the variables' sensitivity levels can be divided into two types in program, high level "H" and low level "L", variable $X_H \in H$, $X_L \in L$, and after the execution of statement S, X_L becomes to X_L'. The amount of information leakage of statement S is given:

$$I_S(X_H, X_L) = H(X_H \mid X_L) - H(X_H \mid X_L')$$

As the base of the log is '2', we take *bit* as the unit of the quantity of information-leak in this paper.

B. Analysis of Statements Information-Leak Quantities

Next, we will stress on analyzing the quantities of information-leak of assignment statements and conditional statements.

1) Assignment Statement $S \equiv a=b$;

Suppose that after the execution of statement S, a becomes to a', b becomes to b'.

If $a \in H$, $b \in L$, since $b=b'$, then $I_{a=b}(a,b)=H(a|b)-H(a|b')=0$, it means that there is no information-leak.

If $a \in L$, $b \in H$, as the value of a is unknown before the execution of the assignment statement S, so $H(b|a)=H(b)$. However, after the execution of S, the value of a, namely a', can be determined entirely by b. That is: $H(b|a')=0$, so:

$$I_{a=b}(b,a) = H(b|a) - H(b|a') = H(b|a) - 0 = H(b) \quad (1)$$

Suppose that b is a integer, and $b \in [1,n]$, then, the formulation (1) will change into:

$$I_{a=b}(b,a) = H(b) = -\sum_{i=1}^{n} P(b = i) \log_2 P(b = i) = -\sum_{i=1}^{n} \frac{1}{n} \log_2 (\frac{1}{n}) \quad (2)$$

Based on the discussion above, we can learn that there is no information-leak when we assigned the values of low confidential variables to high ones. Otherwise, the quantities of information-leak are the entropies of high confidential

variables. For this reason, we can find that the quantities of information-leak of assignment statements relate to the range of the variables' values that in the right side of the assignment, and irrelevant to the range of the assigned variables' values.

For example, suppose that a is a low confidential integer, b is a high confidential integer, and $b \in [1,2]$, then, when we assign b to a, the quantity of information-leak is:

$$I_{a=b}(b,a) = H(b) = -\sum_{i=1}^{2} P(b=i)\log_2 P(b=i) = \frac{1}{2}\log_2 2 + \frac{1}{2}\log_2 2 = 1(bit)$$

If $b \in [1,16]$, then, it changes into:

$$I_{a=b}(b,a) = H(b) = -\sum_{i=1}^{16} P(b=i)\log_2 P(b=i) = 16*(1/16)\log_2 16 = 4(bit)$$

2) Conditional Statement $S \equiv$ if e M;

Suppose that the expression e is a non-assignment function with regard to variable a, record it as $e=f(a)$, and the statement M is a assignment statement of variable b, write it as $M \equiv b=b_j$, then, conditional statement S may be translated into: $S \equiv$ if $f(a)$ $b=b_j$;

The probabilities of $f(a)$ to be "true" and "false" are both 1/2, namely $P(f(a)=true)=P(f(a)=false)=1/2$; the expression $P(a=a_i)$ represents the probability of a gets the value of a_i; likewise, the expression $P(b=b_j)$ represents the probability of b gets the value of b_j.

Suppose that after the execution of statement S, a changes into a', and b changes into b'.

If $a \in L$, $b \in H$, as $a=a'$, thus, $I_S(b,a) = H(b|a) - H(b|a') = 0$, denotes that there is no information leak.

If $a \in H$, $b \in L$, as the value of b is unknown before the execution of the conditional statement S, thus $H(a|b)=H(a)$, then:

$$I_S(a,b) = H(a|b) - H(a|b') = H(a) - H(a|b') \quad (3)$$

Let a and b are two integers, and $a \in [1,m]$, $b \in [1,n]$, then $P(b=b_j)=1/n$, $P(b \neq b_j)=(n-1)/n$.

If $f(a)$ is true, $P_{ij}=P(a=a_i|b=b_j)=nP(a=a_i)$, else, $P_{ij}=P(a=a_i|b=b_j)=0$

If $f(a)$ is false, $Q_{ij}=P(a=a_i|b \neq b_j)=nP(a=a_i)/(n-1)$, else, $Q_{ij}=P(a=a_i|b \neq b_j)=0$, then:

$$\begin{aligned}
H(a|b') &= -((n-1)/n)\sum_{i=1}^{m} Q_{ij}\log_2 Q_{ij} - (1/n)\sum_{i=1}^{m} P_{ij}\log_2 P_{ij} \\
&= -\frac{n-1}{n}P(f(a)=false)\sum_{i=1}^{m}\left(\frac{n}{n-1}P(a=a_i)\right)\log_2\left(\frac{n}{n-1}P(a=a_i)\right) \\
&\quad -\frac{1}{n}P(f(a)=true)\sum_{i=1}^{m}(nP(a=a_i))\log_2(nP(a=a_i)) \\
&= -\frac{n-1}{2n}\sum_{i=1}^{m}\left(\frac{n}{n-1}P(a=a_i)\right)\log_2\left(\frac{n}{n-1}P(a=a_i)\right) - \frac{1}{2n}\sum_{i=1}^{m}(nP(a=a_i))\log_2(nP(a=a_i)) \\
&= -\frac{1}{2}\sum_{i=1}^{m}P(a=a_i)\log_2\left(\frac{n}{n-1}P(a=a_i)\right) - \frac{1}{2}\sum_{i=1}^{m}P(a=a_i)\log_2(nP(a=a_i)) \\
&= -\sum_{i=1}^{m}P(a=a_i)\log_2 P(a=a_i) - \frac{1}{2}\sum_{i=1}^{m}P(a=a_i)\log_2\frac{n}{n-1} - \frac{1}{2}\sum_{i=1}^{m}P(a=a_i)\log_2 n \\
&= -\sum_{i=1}^{m}P(a=a_i)\log_2 P(a=a_i) - \frac{1}{2}\sum_{i=1}^{m}P(a=a_i)\log_2\frac{n^2}{n-1} \\
&= H(a) - \frac{1}{2}\sum_{i=1}^{m}P(a=a_i)\log_2\frac{n^2}{n-1}
\end{aligned}$$

Substitute the expression above into (3), then:

$$\begin{aligned}
I_S &= H(a) - H(a|b') = \frac{1}{2}\sum_{i=1}^{m} P(a=a_i)\log_2\frac{n^2}{n-1} \quad (4) \\
&= \frac{1}{2}\log_2\frac{n^2}{n-1}
\end{aligned}$$

According to the expression above, we can know that, in the conditional statements, if the sensitive levels of the variables in the conditional expression are higher than the variables' in the assignment statements, information-leak may occur. Moreover, the quantities of information-leak relate to the range of assigned variables' values in the execution statements, and irrelevant to the range of the variables' values in the conditional expressions.

Such as the following conditional expression:

if (x>8) y=1;

in which x and y are two integers.

If $x \in [1,16]$, $y \in [1,2]$, then, the quantities of information-leak are 1bit; similarly, if $x \in [1,1600]$, $y \in [1,2]$, the quantities are still 1bit. Hence, the maximum quantities leaking from x to y are 1bit.

Using the similar method, we can do some analysis on the quantities of information-leak for other statements. Due to space limitation, they cannot be discussed in this paper.

From the quantitative analysis above, we can not only conclude when the information-leak will occur, but also the accurate quantities of them. Based on expression (2) and (4), figure 1 shows the comparison of the information-leak quantities between the assignment statements and the conditional statements. So, we can see that the quantities of information-leak caused by assignment statements are more than the conditional statements' even though the variables have the same values range.

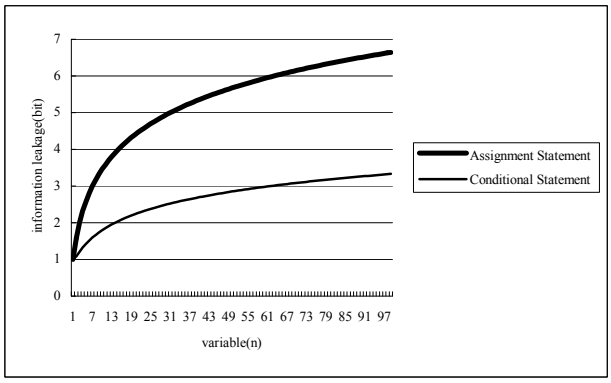

Figure 1. The comparison of the information-leak quantities between the assignment statements and the conditional statements.

In fig.1, the abscissa represents the right side n of the integer variable's range, that is to say, the integer variable gets its value among $[1,n]$, the ordinate represents the quantities of information-leak, takes *bit* as its unit.

V. The Introduction Of Language-Based Software Security

The analysis above illustrates that, although the security policy is employed during developing information system, we still can't avoid the information-leak absolutely. Because the granularity of controlling in security policy is the entity

in the system, it can't refine the access control from the entities to the variables. All these require us to choose a programming language that contains security checking of information flow itself when we are developing some information systems that ask for high-level security. The safety property of programming languages plays a fundamental role in the design and implementation of safety-critical software systems.

Therefore, language-based software security research is a new orientation in the development of high trusted software.

The typical work includes program analysis, program verification, type system and type-safe languages *etc*.

Based on the theories and methods of program analysis[15], many code-debugging tools were developed, such as LC-Lint、Purify. They can be used to find and locate the security holes in the program. However, it is hard to develop security software without covert channels only depend on these tools.

Program verification and axiomatic semantics[16,17] carry out program-property certification by drawing out the logical propositions in program, among these, the properties of programs include security attributes and partial correctness *etc*. However, program verification generally requires integrating the program-specification writing and program developing itself, which may increase the difficulty of programming. In addition, for there is no general algorithm to certify the logical propositions, many certifications of program can't be finished automatically.

The attempt of achieving security software using type-safe languages, e.g. ML, Java and C#, is keeping on[18]. The refined type systems are focusing on designing more advanced types for programming languages, in order to express finer security attributes. In recent years, the typical researches include dependent type, applied type, and singleton type *etc*[19].

In view of the challenges for security software researches and the shortages of existing research, we carried out some useful attempt on the design and implementation of security programming language, which is applicable to the high-level security information systems. We introduced our relative work in paper [20].

ACKNOWLEDGMENT

This work is supported by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China(Grant No. 08KJD520017) and Nation Natural Science Foundation of China (Grant No. 60873176).

REFERENCES

[1] Charles P. Pfleeger, Shari Lawrence Pfleeger, Security in Computing, 4th ed, Pearson Education,2006.

[2] Tom Gallagher, Bryan Jeffries, Lawrence Landauer, Hunting Security Bugs,Washington:Microsoft Press,2007.

[3] Hoglund G, Butler J. Rootkits: Subverting the Windows Kernel. Boston :Addison-Wesley, 2005.

[4] Bing Wu,Xiao-chun Yun, Qi Gao, "Network-based malcode detection technology",Journal on Communications,Vol. 28,no.11,2007,pp.87-91.

[5] Technical papers, http://www.sophos.com/security/,2008.

[6] Gong Li, Ellison G,Dageforde M. Inside Java™2 Platform Security:Architecture, API Design and Implementation. 2nd ed, Addison Wesley,2003.

[7] Freeman A., Jones A., Programming .NET Security, O'Reilly,2003.

[8] Brian A. Lamacchia, .NET Framework Security ,Pearson Education, 2002.

[9] Gosling J,Joy B, Steele G,*et al*. Java™ Language Specification 3rd ed Addison Wesley, 2005.

[10] Horstmann C S,Cornell G.Core Java TM2,Volume Ⅱ -Advanced Features. New Jersey: Prentice Hall,2004.

[11] B. Lampson. "A Note on the confinement problem",Communications of the ACM, Vol.16,no.10, pp. 613-615, Oct. 1973.

[12] D. E. R. Denning. Crgptographyand data security,Addison-Wesley, 1982.

[13] Matt Bishop. Computer Security:Art and Science. Boston: Addison-Wesley Professional, 2002.

[14] Claude Shannon, "A mathematical theory of communication", The Bell system technical Journal, http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html.

[15] Brian Chess, Jacob West, Secure Programming with Static Analysis, Boston :Addison-Wesley, 2007.

[16] Hoare C A R, "An axiomatic basis for computer programming", Communications of the ACM, Vol.12,no.10, 1969, pp. 576-580.

[17] Necula G C, "Proof-carrying code", Proceedings of the 24th ACM Sigplan-Sigact Symposium on Principles of Programming Languages, Paris, France, 1997, pp. 106-119.

[18] Galen Hunt, J ames Larus, "Singularity: Rethinking t he sof tware stack", Operating Systems Review, Vol.41,no.2, 2007, pp.37-49.

[19] Bao-Jian Hua, Yi-Yun Chen, Zhao-Peng Li, *et al*, "Design and proof of a safe programming language PointerC", Chinese Journal of Computers,vol.31(4), 2008 ,pp.556-564.

[20] Bo Chen, Ling Yu, "Covert Channel elimination technique based on security type system in java source code", The 3rd International Conference on Computer Science & Education (ICCSE'08), July 25-27, 2008, Henan, China, XiaMen University Press, pp. 278-282.