Journal Homepage: - www.journalijar.com
INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)
Article DOI: 10.21474/IJAR01/4225
DOI URL: http://dx.doi.org/10.21474/IJAR01/4225

RESEARCH ARTICLE

INFORMATION FLOW SECURITY IN MULTITHREADED JAVA APPLICATIONS.

Swati Pandey.
Department Of Computer Science and Engineering , Galgotias University , Greater Noida-India.

Manuscript Info

Manuscript History

Received: 18 March 2017
Final Accepted: 21 April 2017
Published: May 2017

Key words:-
Information flow control(IFC);explicit flow ; implicit flow; information leakage ; multithreaded application .

Abstract

The information leakage would be an application weakness because it reveals the sensitive information such as the technical information of the web application, or an user specific data . The hacker can damage and exploit its hosting network , the web application , or its user by making the illegal use of the sensitive information. Therefore it is must to take the prevention measures for avoiding such breach in information flow . We are also concerned about the integrity of labeled data during program runtime , because there exist system calls and we want to make sure that during these calls the sensitive information should not be leaked. Took this issue in to consideration and enforced security policies by changing the JVM and the OS modules , providing its own Application Interface (API) to make sure of preserving labeled data integrity. Bytecode Instrumentation plays a vital role to prevent the sensitive information flow to the Input –Output channel or to prevent the explicit information leakage . This paper is going to present all possible measures , models and techniques to provide the security to the data flow in multithreaded applications.

Introduction:-

IFC is a technique to provide security to a given program as per the security policies . The problem of information flow is majorly focused on preventing the flow of the data of high security level to the data of low security level . According to Geoffrey Smith[6] there are three type of channels through which a program could exchange data or the information from the surrounding environment they are legitimate channels , covert channels and storage channels . Channels are described as a path for a sensitive data (what you are trying to protect or keep secret to escape through)A covert channel is a channel that is hidden . This means that its existence is intentional , and additionally there is an intention to conceal or hide its existence from a person who is trying to protect the system by filtering or limiting data flow J Clause , W Li [10] example steganography .The side channels refer to information leakage from a system through characteristics of the system's operation . IFC focuses that nosensitive information should leak to the un trusted users or to the uncontrolled channels like internet I Roy , M Bond [8] such that it prevent the integrity and confidentiality of the information by preventing it to the flow to the covert channels.

Information flow is categorized in to two categories implicit and explicit information flow. The explicit flow is considered the simplest one because in this the secret info is leaked explicitly to an public observable

Corresponding Author:- Swati Pandey.
Address:- Department Of Computer Science and Engineering , Galgotias University , Greater Noida-India.

variable . With the help of the security classes SC the secret input can not flow to the public output apparently , A Russo [1] the public output can not be affected from the secret input.

because there exist security classes and it only permits the upward flow of data / information .H refers to high and L refers to low . when we use notation "=" it presents value assignment in that case three possibilities would be there that is H=H , L=L , H=L but the one L=H is not possible because it can leak the

secured information through the unsecured channel like printing and storing to an input / output device and making it available to the uncontrolled channels like internet.

The second typeis described as the implicit information flow , which causes information leak through program control flow or the branch in control flow , as in if – then - else statement . Null pointer exception , arithmetic exception , array index are the cause of implicit information flow. To avoid such problem programmer must caught these type of exception , whenever there is a call to method it throws the I / O exception that isnot necessarily due to programmer. According to the below program there are two variables k and l . The secret variable l its value gets revealedimplicitly to the variable k. All the bits of the variable l are disclosed , since the variable l is Boolean (if l is true k would be 5, else it is 75)

```
Var k , l
If  l = true then
k:=5
else
k:=75
```

Exception is considered as an abnormal behavior which arisesin the code sequence during the run time . It is called that the exception is a runtime error. The compiler never complains about them . Other source of implicit information flow are due to unchecked exceptions because they changes the control flow graph of the program , thatis by deviating it from predefined (CFG) control flow graph . Let"s say caller method gets an exception without any exception handler . If exception handler is installed for invocation byte code of this caller causative method , then this handler would handle the exception otherwise this causative caller method is goingto terminate abruptly cause the " re throwing" of exception . This schema would be applicable until an suitable handler found in Class stack else the execution thread is going to get terminate and theexception would be printed to user . In CFG there is an introduction of an predicate

node along with two successors and each of them excepting the calls instruction one models for normal termination while the other for abrupt termination for the called method

```
❖ void xyz( Object [] p) {
❖ try {
❖ bar (p);
❖ } catch (
❖ NullPointerException e) {
❖ print (" null ");
❖ }
❖ }
❖ void bar( Object [] p) {
❖ print (p. length );
❖ }
```

Code example of exception handling through SDG:-

In this paper the approach is to describe the instrumentation of byte code through the ASM framework , this framework can modify the dynamically generated classes or the existing classes along with the transformation and the analysis algorithm which allows to assemble the code analysis tools and the custom complex transformation which assured the security of the information flow in multithreaded applications of java . Till now the focus has been made on the single threaded applications [previous work].

The Remainder section of this paper is defined as Section (ii) - is for related work, Section (iii) – defines the ASM Framework (Methodology) , Section (iv) - represents the case study based on the intended model and lastly the conclusion and the future scope is presented in Section (v).

Related Work:-

The cause for the variety of attacks on our user applications is the improper validation of the user input . Taint mode can be described as an tracking mechanism of an information flow for the specific data of an program during the runtime . The languages that support the taint mode are Ruby , Perl , PHP .During runtime Taint mode Vivek Haldar [13] is working in an dynamic analysis mode and user input data is treated as being un trusted . Such problems can be detected through the static approaches which help during an time of development but it requires the source codeand the count of false positives are reported by it . Therefore , for securing an rapidly evolving and fully deployed applications it is of an little use . Among the explicit information flow and the implicit information flow this dynamic analysis method chooses an explicit flow of information . Like the pearl and ruby supports this tainted data but the java does not support it . So to handle the information flow control through java we need to find some other measures to preserve the integrity and confidentiality of IF . Jif (security type programming language)

that extends java with support for information flow control and access control enforced at both compile time and run time. Jif is written in java and is built using the polyglot extensible java compiler framework. It supports static information flow Kyle pullicino [14] analysis but it bypasses multithreading which still represent challenge for secure information flow.

Code can be added to every method call at the beginning and ending of it with the help of an external tool and this technique is known as (BCI) Byte Code Instrumentation , Every class with this technique allowing to get method related info and to measure the performance. We can perform a transaction trace while the transaction reaching to a JVM , The method calls that are serving HTTP calls can be intercepted by using BCI and that can create a unique key. Java can be written with the help of java agent interface and class loader can itself execute that code and within every class the bytecode manipulation is possible and thus making the whole process quite straightforward. Before entering the realm of Bytecode Instrumentation we must have knowledge of bytecode and the functioning of the JVM.

Bytecode is just like a shorthand language that store each keyword of java as a sign. And each sign take one byte of memory in RAM , Hence called the bytecode file . The SUNMICROSYSTEM takes the whole

initiative and had made a unique compiler that produces a platform independent bytecode , and specific JVM and Interpreter that converts bytecode into machine code that vary from OS to OS. JVM can be customized with the help of java options either by allocating maximum or minimum memory . It is an interface that‟s run the java programme S. Nair, P. Simpson[18] and get destroyed as soon as the programme ends . further

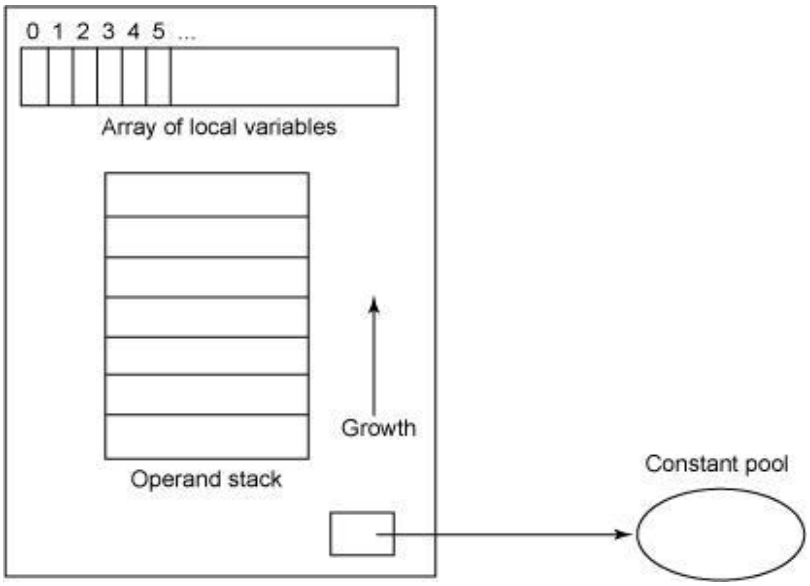


Figure1:- Stack Generation

JVM is known as stack based machine Timlindholm[12] , every thread consists JVM stack that stores the frames. When method is invoked , at that time the frame is created that consists of an array of local variable , an operand stack and a reference to the current method class at runtime constant pool .
Java is a tool used to invoke JVM , When JVM invoke.

1. A subprogram in JVM called class loader (or system class loader) starts and load the byte code in to OS memory or RAM.
2. Another subprogram Byte code verifier verify and ensure that the code do not violate the security rules. That‟s why java program is much secured and virus free.
3. The last subprogram execution engine finally converts byte code into machine code and name of that engine are in use today is JIT (Just In Time Compiler) .

An application can use reflection and introspection API in the JVM to discover what classes exist what interfaces they implement or classes they extend. What files and method they expose and execute method without having any compile time knowledge of the class in the question. Reflection and Introspection are powerful but they lack a key feature they can not alter the behaviour of the class. Before JIT , silicon and adaptive execution engine were used but there performance was very degraded and slow. If the program is suffering from errors it is necessary to remove it because these errors will propogate to the next level and anything we built upon it will also suffer from the errors. If we check for the race conditions and deadlock before instrumenting the byte code , that comes under a good practice and by doing so it ensures that the functionality of a program is correct. JPF is a software model checker that looks for the possible race condition or the deadlock in java byte code.

IF we have a need to modify a class at runtime, we are now entering a realm of Byte code instrumentation. 1-Java Assist - is a java library which allows to write the java code that is converted to Byte Code by its own internal compiler. 2 - BCEL-A powerful library that reads byte code from a class file ,allows you to modify that byte code and generate new classes and class files . 3- ASM –is used for manipulation of byte code and for framework analysis. It would also be used to modify the dynamically generated classes or the existing classes , directly into binary form , with the help of analysis algorithm and with commom transformation it allows to assemble the code analysis tools and the custom complex transformations. 4- CGL–This library can be used to implement the java classes and to implement the interfaces at runtime. It is also the underlying technology below BCEL and ASM as well as Hibernate and Spring.

Good efficiency and small size are the properties which makes ASM differ from the other . Jar library of an ASM weighs 21 Kb , while 350 kb for BCL and 150 kb of storage space for SERP . While instrumenting the classes the overhead that ASM adds is around 60% whereas , the same for BCEL is 700% and 1100 % for an SERP .

Methodology:-

This section presents the Byte code instrumentation process through the ASM framework. Instrument of Java .class files using the ASM framework will take place in to three parts.

Part 1 Presents javabytecode introduction and way to read disassembled .class files.
Part 2 Introduces the visitor pattern and this pattern is used throughout the ASM framework.
Part 3 By using ASM , we build an simple call trace instrumentation.

Instrumenting Java Bytecode with ASM:-
ASM is a framework to manipulate the java bytecode.Firstly we will compile the java program by using javac and then disassembled the .class file we use “javap-c” through which we can get the java bytecode. The file containing the java bytecode known as java class file reffered with .class filename extension and can be executed on the JVM. JVM compiler produces the class file from the source file with extension (.java files) of java programming language

Part 1:-
Example

```
public class Testing
{
```

```
public static void main(String[] args) {
printTestabc();
printTestabc();
printTestxyz();
}
public static void printTestabc() {
System.out.println("Hey , hello testing!!!");
}
public static void printTestxyz() {
printTestabc();
printTestabc();
}
}
```

Explaination:-

```
public class Testing {
public Testing();
Code:
0: aload_0
1: invokespecial #1                  // Method java/lang/Object."":()V
4: return
public static void main(java.lang.String[]);
Code:
0: invokestatic    #2    // Method printTestabc:()V
3: invokestatic    #2    // Method printTestabc:()V
6: invokestatic    #3    // Method printTestxyz:()V
9: return

public static void printTestabc();
Code:
0: getstatic       #4        // Field java/lang/System.out:Ljava/io/PrintStream;
3: ldc             #5        // String Hello World
5: invokevirtual #6          // Method java/io/PrintStream.println:(Ljava/lang/String;)V
8: return

public static void printTestxyz();
Code:
0: invokestatic    #2    // Method printTestabc:()V
3: invokestatic    #2    // Method printTestabc:()V
6: return
}
```

The byte code of public test() constructor consists of three opcode set of instructions. First one aload_0, pushes the value from index 0 onto the operand stack. Index 0 is the value of the table of local variable.The parameters can be passed to the methods through the local variable table.. The second opcode instruction is, invokespecial, the role of this instruction is to call the constructor of this class's as superclass. Because classes can implicitly inherit from java.lang.object(package) in case if they do not explicitly extend any other class.The necessary bytecode is provided by the compiler to invoke the contructor if this base class.The top value is popped from the operand stack during the second opcode instruction. Last instruction, return, what is needed to be returned.Some Opcode having parameters take space in bytecode array this is the reason why index number not found continuous.

Various string constants, interfaces names,field names,class names and others referred within the class file structure are represented in a table of structure of constant pool. The # in above bytecode represents the constant index seeking for the constant in constant pool.By using “javap –c –v” . We can have a look for a whole constant pool.
There are two kinds of methods applicable in java programming language that are instance method and class method.When class method is invoked by the jvm it invokes the method based on the type of object reference which

could be known only at the compile time, on the other hand when jvm invokes the class method ,it invokes only based on the actual class of the object, which could only be known at the run time.

Part 2**:-** Visitor Pattern

The programming language which supports single dispatch is required by the visitor pattern. we are going to consider two objects of some class type lets call one is “element” and other is “visitor”. Both of the element having a method ,an element is having a accept() method and this method can take visitor as an argument. The element would pass itself as an argument to visit() method of the visitor . Following steps would show how the above visitor pattern is used in ASM for bytecode manipulation.

Firstly - add the accept(Visitor) method to an “element”hierarchy.

Secondly – with respect to a visit method create a “visitor” base class for every element type.
Thirdly- generate a "visitor" derived class for each "operation" to do on "elements"

Lastly - The Client creates "visitor" objects and would passes each to accept() calls.

Part 3:-
Call Trace Instrumentation:-

Now we are going to perform call trace instrumentation by using ASM. Each method call will be log by the instrumented code and then return. The output log could easily be parsed into a Calling Context Tree for performing the above operation JDK,Apache ant and the ASM 5.0.3 must be downloaded and installed In our first program we will simply make a copy of .class file. Which will refer to us as the ASM boilerplate, and work as a template which can be used further for other different instrumentation purpose.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import org.objectweb.asm.ClassReader;
import org.objectweb.asm.ClassWriter;
public class Copy {
public static void main(final String args[]) throws Exception {
FileInputStream is = new FileInputStream(args[0]);
ClassReader cr = new ClassReader(is);
ClassWriter cw = new ClassWriter(ClassWriter.COMPUTE_FRAMES);
cr.accept(cw, 0);
FileOutputStream fos = new FileOutputStream(args[1]);
fos.write(cw.toByteArray());
fos.close();
}
}
```

Further describing in brief the call trace instrumentation

From the first copy program which consists two command-line arguments: arg[0] is used for to copy from and arg[1] is for to copy to. Class Reader and Class Writer are the two ASM classes class reader reads the java byte code and class writer writes the byte code to the file .Visitor pattern is used by the ASM Class Visitor interface implements through the ClassWriter,ClassReader traverse the byte code input while call to cr.accept(cw,0) ,which generates a series of calls to cw that would generate the same bytecode as the output .Instrument a call sites to print to standard error (stderr) before and after the call.

Like while instrumenting the test class the result would be equivalent as
following:-public class TestingInstrumentation

```
{
public static void main(String[] args) {
```

```
System.err.println("CALL printTestabc");
printTestabc();
System.err.println("RETURN printTestabc");
System.err.println("CALL printTestabc");
printTestabc();
System.err.println("RETURN printTestabc");
System.err.println("CALL printTestxyz");
printTestxyz();
System.err.println("RETURN printTestxyz");
}
public static void printTestabc() {
System.err.println("CALL println");
System.out.println("Hey , hello testing!");
System.err.println("RETURN println");
}
public static void printTestxyz() {
System.err.println("CALL printTestabc");
printTestabc();
System.err.println("RETURN printTestabc");
System.err.println("CALL printTestabc");
printTestabc();
System.err.println("RETURN printTestabc");
}
}
```

In between two classes class reader and class writer there is a need to insert code and we will perform this by using an adapter pattern .As adapter can override the methods and can wrap the object , through which it would be helpful to alter the behaviour of an wrapped object. While the byte code for call site is emit , the byte code to produce trace log before and after the call is also emitted and this can only be done through adapting the class writer.Firstly we will adapt the object of class writer by using the ClassAdapter class.By default the methods of ClassAdapter are inherited from Class Visitor due to which same method will be called on the adapted Class Writer.The method which is called once during each declaration of method is ClassWriter.VisitMethod and we are going to override the behaviour of this method. MethodVisitor object is an return type of visit Method which is further used to process the method body. The byte code for a method is produced by a return value of visit Method and then we will adapt this return type and the additional instructions are added ,that can print the call trace.

```
class ClassAdapter extends ClassVisitor implements Opcodes {
public ClassAdapter(final ClassVisitor cv) {
super(ASM5, cv);
}
@Override

public MethodVisitor visitMethod(final int access, final String name, final
String desc, final String signature, final String[] exceptions) { MethodVisitor
mv = cv.visitMethod(access, name, desc, signature, exceptions); return mv ==
null ? null : new MethodAdapter(mv);
}
}
class MethodAdapter extends MethodVisitor implements Opcodes {
public MethodAdapter(final MethodVisitor mv) {
super(ASM5, mv);
}
@Override
public void visitMethodInsn(int opcode, String owner, String name, String desc, boolean itf)
{ /* TODO: System.err.println("CALL" + name); */
/* do call */
```

```
mv.visitMethodInsn(opcode, owner, name, desc, itf);
/* TODO: System.err.println("RETURN" + name);  */
}
}
```

So far, our MethodAdapter class doesn't add any instrumentation -- it simply delegates to the wrapped MethodVisitor mv. We have a good idea how to write the instrumentation in Java syntax, but we don't know how to express it using ASM's API. For this, we will use the ASMifier tool, which is distributed with ASM.

Now, lets give a trial on our Testing example:-

```
# Build Instrumenter$ javac -cp asm-all-5.0.3.jar Instrumenter.java
# Build Example

$ javac Test.java

# Move Testing.class out of the way
$ cp Testing.class Testing.class.bak
# Instrument Testing

$ java -cp .:asm-all-5.0.3.jar Instrumenter Testing.class.bak Testing.class
# Run!

$ java Test
CALL printTestabc

CALL println
Hey , hello testing !

RETURN println
RETURN printTestabc

CALL printTestabc
CALL println

Hey , hello testing !
RETURN println

RETURN printTestabc
CALL printTestxyz

CALL printTestabc
CALL println

Hey , hello testing !
RETURN println

RETURN printTestabc
CALL printTestabc

CALL println
Hey , hello testing !

RETURN println
RETURN printTestabc

RETURN printTestxyz
```

The output is exactly what we expected, plus the four "Hey , hello testing !" lines printed to stdout (which is interleaved with stderr). As we can see that with the help of above process we had achieved a dynamic class generation and modification and for this we required a very small and very fast tool , needed a tool primarily adapted for simple transformations and do not needed a complete control over the produced classes. Our approach in this is as we had used an visitor pattern without using an explicit object model completely hide the (de)serialization and constant pool management details, represents jump offset by label objects , automatic computation of the max stacksize and StackMap.

Part Iv Case Study Based On The Real World Example:-

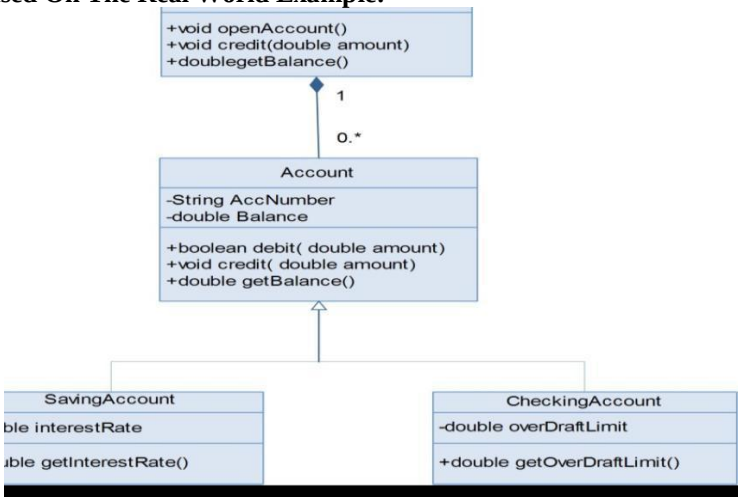


Figure 2:- Class Diagram(UML) Of Banking System

The problem in the above example would come while considering the byte code of the method[Boolean debit(double amount)] of account class as per above UML diagram. While purchasing, if the fund is not equal and greater to the purchasing or the debiting amount in that case warning statement is given by the system, This statement consist the sensitive information of the customer account that must not be disclosed in any case.If it discloses the sensitive information then it is a case of explicit information leakage. The issue is that the byte code reads the sensitive information from the bank account of customer and concatenates it (if available fund is not equals debiting amount) with the string ”Low Fund” at line #21 and then passes it to the line#36.As shown in the below byte code figure.-3The attacker can easily attack and abuse and can perform illegal actions like illegal purchase transaction which causes a much damage as shown in figure -3 we know that println ()is an static method of Print Stream class and in this code byte code of get field() is called just before static method pritnln () of the PrintStream class . Which would cause the possible leak of sensitive information. Mohamed Sharaf[16] **.** When SIF MJ model modifies the arc between Ready to Run and Running state by adding an additional set of states that provides security check and information sealant against any information leakage. which shows the newly added states between Ready to Run and Running state as shown in dotted box. Firstly when the thread is in RTR state , the thread has to present its capability and required resources to get approved by the scheduler to get executed . Set of operations are thread capability which the thread is contracted to perform .Each thread must submit before starting , that is a thread that is allowed to read the account balance may not be permitted to write to the same field or disclose it to the insecure channel.

If the thread satisfies the security policies then scheduler will allow it to execute further , otherwise thread will come in to a Timeout state in which the thread is removed from the scheduler Ready to Run queue now it has to wait for a longer period of time to re-enter in the ready to run state. If in case thread violates any security rules it would not be allowed to enter in the running state . In this way with the help of predefined security policies there would not be any information leakage , as sum up in the block diagram 6.

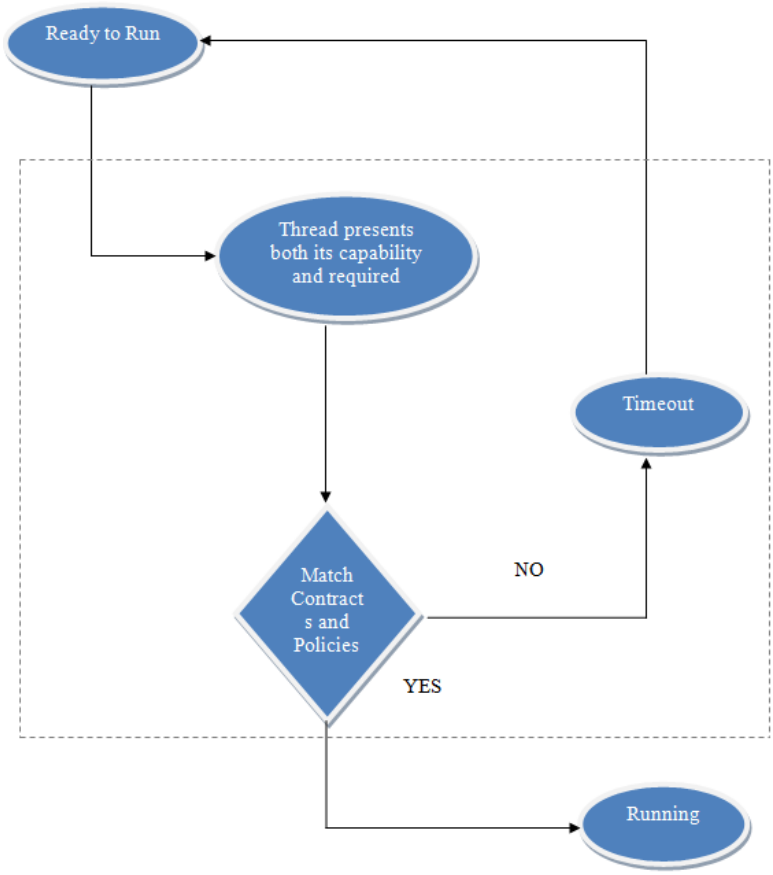


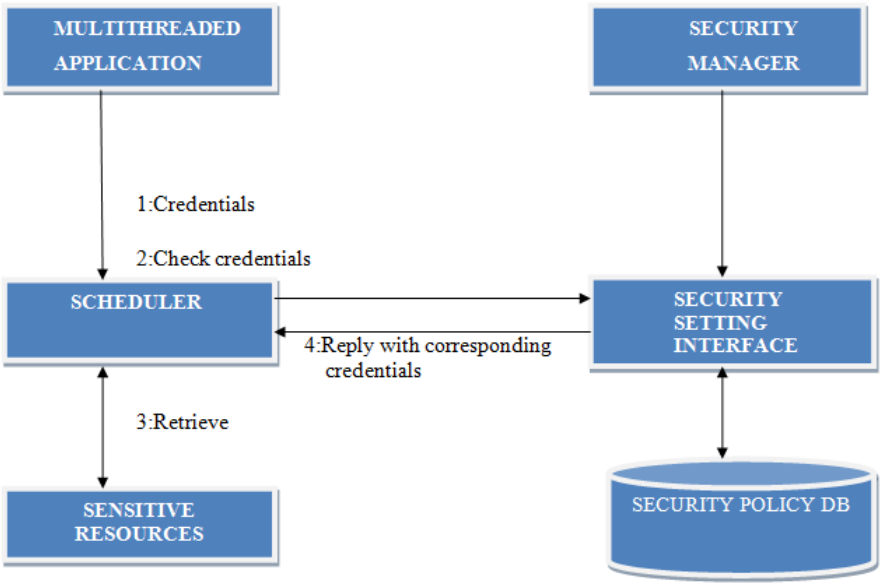
Figure 3:- Newly added states between RTR and Running state

Figure4:- Breach in information flow Mohamed Sharaf[16]

```
iconst_1
istore_3
dload_1
aload_0
getfield #3 <Accountl/balance D>
dcmpl
ifle 24 (+16)
getstatic #4 <java/lang/System/out Ljava/io/PrintStream;>
ldc #5 <Not Enough Fund : >
invokevirtual #6 <java/io/PrintStream/println(Ljava/lang/String;)V>
iconst_0
istore_3
goto 34 (+13)
aload_0
dup
getfield #3 <Accountl/balance D>
dload_1
dsub
putfield #3 <Accountl/balance D>
iload_3
```

Figure 5:- Instrumented Bytecode

The byte code above in figure – 4 shows the violation of rules which leads to the insecure or BJFB explicit information leak . From SIF-MJ model this breach can be fixed and with the byte code instrumentation process . Figure -5 shows the prevention of the balance filed of above code from propagating to insecure input-output channel.



Mohamed Sharaf [16]

Figure 6:- Secured Java Multithreading Scheduler Mohamed Sharaf [16]

Part V Concluding Remarks And Future Work:-

Information flow control in multithreaded application is a challenging task. We have achieved a lot of it from SIF-MJ model **,** the instrumentation which we are performing from third policy of SIF-MJ Mohammed Sharaf [16] is benefitial and proved much helpful for Java developers because it provides the potential visibility to method call and to the every single class because if there is deeper visibility, overhead would be higher .The organization can not bear to impact huge overhead , hence it is required that the trace process must limited by design. A limited trace has to be tailored to every Java application, which makes implementation in real-life scenarios a much more costly task.

For future work we need to improve and extend SIF-MJ (Security) model .For multithreaded java applications providing a secure framework that should be supported by RMI. Further BCI has its own limitations because of which we need to look and renew it . Another work is required to be done in an area known as "reverse engineering" in which it is required the process of deobfuscating the obfuscated code prior before making it available for instrumentation process.

Refrences:-

1. Russo, A.Sabelfeld, "Dynamic vs. static flow sensitive security analysis", Proceedings of 23rd IEEE Computer Security Foundations Symposium 2010 (CSF '10), July 2010.
2. Sabelfeld, A. Myers, "Language-based information-flow security", IEEE Journal on Selected Areas in Communications, Vol. 21, No. 1, Jan. 2003.
3. Lampson, "A note on the confinement problem," Communication Magazine of the ACM, Vol. 16, No. 10, pp. 613–615, 1973.
4. Bell, L. LaPadula, "Secure computer system: unified exposition and multics interpretation", MTR-2997 Rev.1. Technical report, MITRE, 1976.
5. King , B. Hicks , M. Hicks , T. Jaeger , "Implicit flows : can‟t live with „em, can‟t live without „em" ,
6. LNCS 5352 , pp. 65-70 , Springer-Verlag Berlin Heidelberg 2008 (ICISS '08) , Dec. 2008.
7. Geoffrey Smith "Principals for secure information analysis"January 2007 G. Smith, "Principles of secure information flow analysis", Malware Detection, pp. 291-307, Springer-Verlag, 2007.
8. I . Roy, D. Porter, M. Bond, K. McKinley, E. Witchel, "Laminar: practical fine-grained decentralized information flow control " , Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation 2009 (PLDI '09) , June 2009 .
9. J. Howarth, I. Altas , B. Dalgarno, K.-F. Lee, "Information flow controlusing the Java Virtual Machine Tool Interface (JVMTI)" , Proceedings of Fifth International Conference on Availability, Reliability , and Security 2010 (ARES '10) , Feb. 2010.
10. J. Clause, W. Li, A. Orso, "Dytan: a generic dynamic taint analysis framework", Proceedings of sInternational Symposium on Software Testing and Analysis 2007 (ISSTA '07), July 2007.
11. JPF, Java Path Finder, http://javapathfinder.sourceforge.net/ , Stable release 6.0/November 30 ,2010
12. The Java Virtual Machine Specification ,TimLindholm ,Frank Yellin ,May 2014 http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC. doc.html
13. VivekHaldar ,DeepakChandra ,Michael Franz"Dynamic taint propogation for java" IEEE Dec. 2005
14. J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software", ACM Transactions on Information and System Security, Volume 12, Issue 2,
15. Dec. 2008.
16. Language based information flow security in java , Kyle Pullicino, 30 dec 2014
17. Mohamed Sharaf ,Jie Huang , Chin-TserHuang ,"Using bytecode instrumentation to secure information flow in multithreaded java applications"2013IEEE 33rd international conference on distributed computing systems .
18. P. Li, S. Zdancewic, "Downgrading policies and relaxed noninterference", Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages 2005 (POP '05), Jan. 2005.
19. S. Nair, P. Simpson, B. Crispo, A. Tanenbaum, "A Virtual machine based information flow control system for policy enforcement", Journal of Electronic Notes in Theoretical Computer Science, Vol. 197, No. 1, pp. 3-16, Feb 2008.
20. Yin Liu , Ana Milanova,"static information flow analysis with handling of implicit flows and a study on effects of explicits flows vs implicit flows" March 2010.
21. S. Yoshihama T. Yoshizawa Y.Watanabe , M.Kudoh , K. Oyanagi , "Dynamic information flow control architecture for web applications " , Proceeding of 12th European Symposium on Research in Computer Security (ESORICS'07), Dec 2007.