**A PROJECT REPORT ON**

**“Decomp-Java” -** utility to decompile class code of java

Project report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology (Computer Science & Engg.)

By

Karan Kumar

(8718120)



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**STATE INSTITUTE OF ENGINEERING & TECHNOLOGY, NILOKHERI**

**(2018 – 2022)**

**DECLARATION**

I hereby declare that the Project report entitled **"Decomp-Java"** submitted to the Department of Computer Science & Engineering, SIET Nilokheri in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology (Computer Science & Engineering) is a record of original work done by me, under the guidance and supervision of **Sh.Ved Prakash** and it has not formed the basis for the award of any Degree/Diploma/Associateship/ Fellowship or other similar title to any candidate of any University.

**Dated** **Signature of the Candidate**

**ACKNOWLEDGEMENT**

I would like to express my special thanks of gratitude to my teachers who give me the golden opportunty to do this wonderful project on the website development. Working on this project

would not be completed without the aid and support of different websites,videos and study material.

I would also to thank my university and college “State Institute of Engineering and Technology”, for alowing me to work on this project.

My completion of this project could not have been accomplished without the support of my family members,teachers,friends and atlast Google and DuckDuckgo search engines who always make me help and motivated to complete this projects, Thanks to all of you.

**Karan Kumar**

**ABSTRACT**

Reverse Engineering java programs, having java byte code or .class code is always a headache for security researcher or reverse engineers.

Analyzing and reversing the java code in order to improve product or find security holes is always so tough cause java byte code is not easy to understand.

So i made this tool “Decomp-Java” to make work easy and efficient.

**Warning ,** the main motive of development of this tool is to provide security researchers, computer hackers, pentesters and application auditors, ease to reverse or understand whats happening on java program

Author has no liability , in any whatever manner this tool is going to use , user should aware that wherever he/she using this tool is have permission, so using this tool without permission is subject to criminal activity , can put user behind rods,so be careful.

**Karan Kumar**

**TABLE OF CONTENTS**

**Page**

Declaration i

Acknowledgment ii

Abstract iii

Table of contents iv-v

**Chapters**

1. Introduction 1

1.1 Problem statement 1

1.2 Problem objectives 1

1.3 Project goals 2

1.4 Project scopes 2

2. Literature survey 3

2.1 How JVM works – JVM Architecture ? 3

2.1.1 Linking 4

2.1.2 Initialisation 4

2.1.3 Execution 5

2.1.4 JVM memory 6

2.1.5 JNI 6

2.2 How compilers works 7

2.2.1 Phases of compilation 7

2.2.2 Lexical Analysis 7

2.2.3 Syntax Analysis 8

2.2.4 Semantic Analysis 8

2.2.5 Intermediate Code Generation 8

2.2.6 Optimization 9

2.2.7 Code Generation 9

2.2.8 Practical Example 9

2.3 Compiler Vs Interpreter 11

2.4 What is Java-Byte Code ? 11

2.4.1 How Java-Bytecode works ? 12

2.4.2 Advantages of Java-Bytecode 12

2.5 Java security architecture 13

2.5.1 Java security model 13

2.5.2 The sandbox defined 13

2.5.3 The sandbox pervasive 14

3. Methodology used 15

3.1 How we use this tool `decomp-java` 15

3.2 What are requirements of the vulnerable labs ? 15

3.3 What is decompilation ? 15

3.4 Brief details of reverse engineering 17

3.4.1 Great Overview 17

3.4.2 Common situation of reverse engineering 18

3.5 What is security audits ? 24

3.5.1 Why do a security audit ? 24

3.5.2 When is security needed? 24

3.5.3 Types of security audits 25

3.5.4 What systems does an audit cover 25

3.5.5 Steps involved in security audit 26

3.5.6 Test vs assessment vs audit 26

4. Over-all depth information about project 27

4.1 What is decomp-java ? 27

4.2 Technologies used in decomp-java and vulnerable web labs . 27

4.3 Introduction to the various technologies used in cssi 25

4.3.1 JAVA 28

4.3.2 ANTLR 32

4.3.3 Gradle 34

4.3.4 Java-Swing 35

4.3.5 Java-Awt 39

4.3.6 Docker ` 41

4.3.7 Spring-Web-Framework 44

5. Results / Snapshots 44-60

6. Conclusion 61

7. References 62

**1. INTRODUCTION**

**1.1. Problem statement**

Reverse Engineering java programs, having java byte code or .class code is always a headache for security researcher or reverse engineers.

Analyzing and reversing the java code in order to improve product or find security holes is always so tough cause java byte code is not easy to understand.

So in order to prove this vulnerability exist in your client program you need to first exploit the vulnerability and than need to generate poc (proof of concept.), so making poc and exploit for this type of vulnerability is such headache security researcher or analyst to analyze code or whats happening back side.

So there should anything we can get same code to analyze easily whats happening on program.

**1.2. Problem objectives**

There should be some suite or utility which having these features :

1. Tool should be cross-platform.
2. Should be easy to install.
3. Tool should generate best precise and efficient code for help reverse engineers to find whats happening in code.
4. Tool should have features like search for strings, backward or forward navigation through code.
5. Tool should work with many extensions having java bytecode or .class code.
6. Tool should decompile the code in efficient manner.

Reverse engineering or security auditing is the game of mouse and cat so tool should have precise result which can help to find security holes or improve product.

Effiecency should be great for ease of working.

**1.3 Project goals**

We need to achieve some goals to setup this type of website or platforms

This should achieves following -

1. The gui suite to get best generated legitimated code from program
2. Utility should work with so many extensions having java byte code or class code.
3. Tool should be very user friendly to use, user just need knowledge of exploitation.
4. Tool should have these features like search for string,backward or forward navigation for codes etc.
5. Tool should help to analyze things so much precise to help in the improvement of product program or find security vulnerabilities.

**1.4 Project scopes**

In regarding the scope of this project, there are many options we can provide with this project in different scopes.

This tool can be used in these types of scenarios–

1. Can be used in bug-bounties program to create poc of vulnerability.
2. Can be used in security auditing in company to find any vulnerability exist or not.
3. Can be used by pentester in penetration testing of webapps
4. Can be used in reverse engineering.
5. Can be used in CTFs challenge and many more.
6. So scope of this tool , not only limited to computer user can deploy this in vps for robust exploitations and exfilteration.

**2. LITERATURE SURVEY**

**2.1 How JVM works – JVM Architecture ?**

JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a java code. JVM is a part of JRE(Java Runtime Environment).

Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.

When we compile a *.java* file, *.class* files(contains byte-code) with the same class names present in *.java* file are generated by the Java compiler. This *class* file goes into various steps when we run it. These steps together describe the whole JVM.

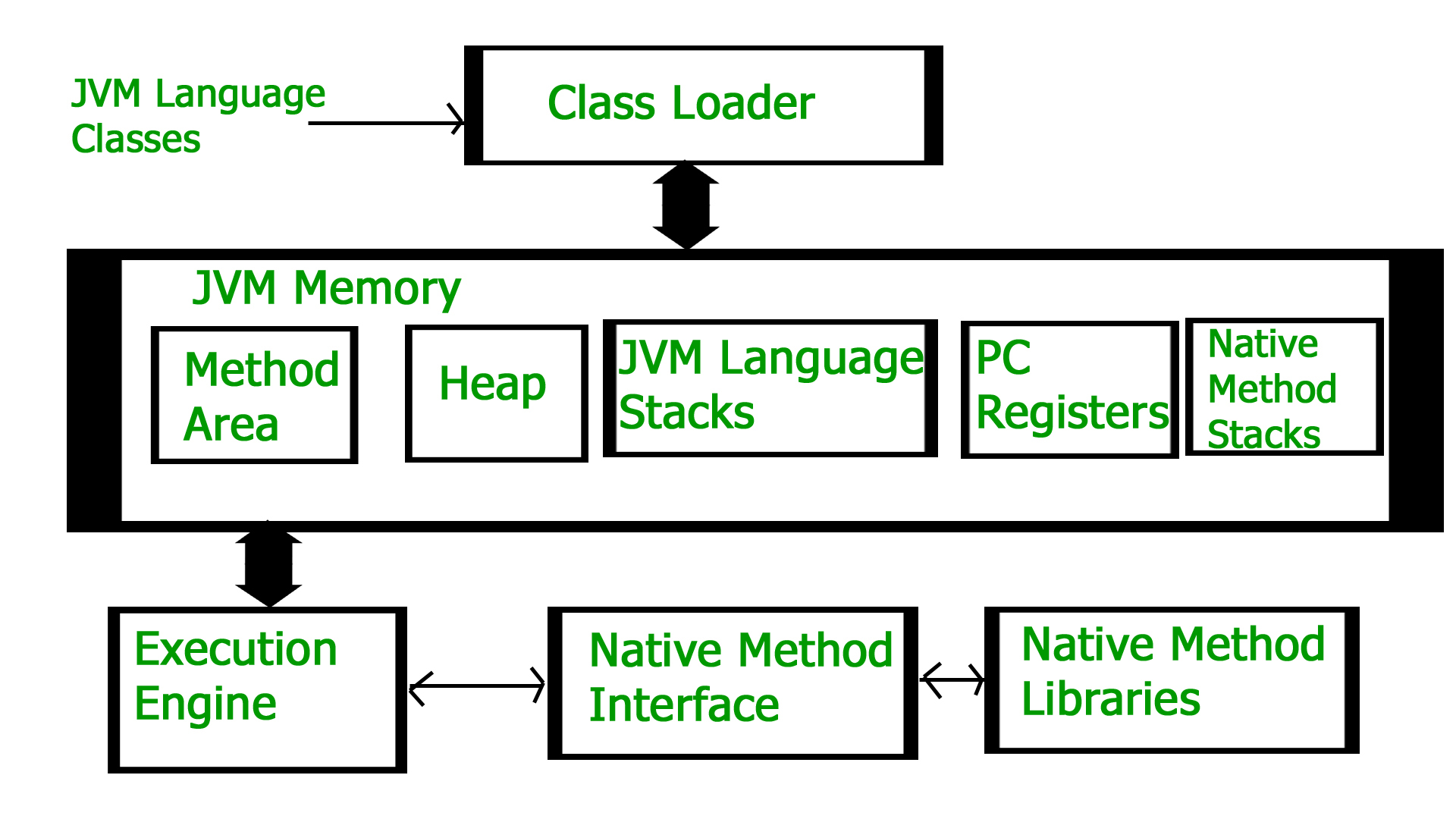


Fig.2.1

Class Loader Subsystem

It is mainly responsible for three activities.

* Loading
* Linking
* Initialization

Loading: The Class loader reads the “.*class”f*ile, generate the corresponding binary data and save it in the method area. For each “*.class”* file, JVM stores the following information in the method area.

* The fully qualified name of the loaded class and its immediate parent class.

**Fig.2.1**

* Whether the “*.class”* file is related to Class or Interface or Enum.
* Modifier, Variables and Method information etc.

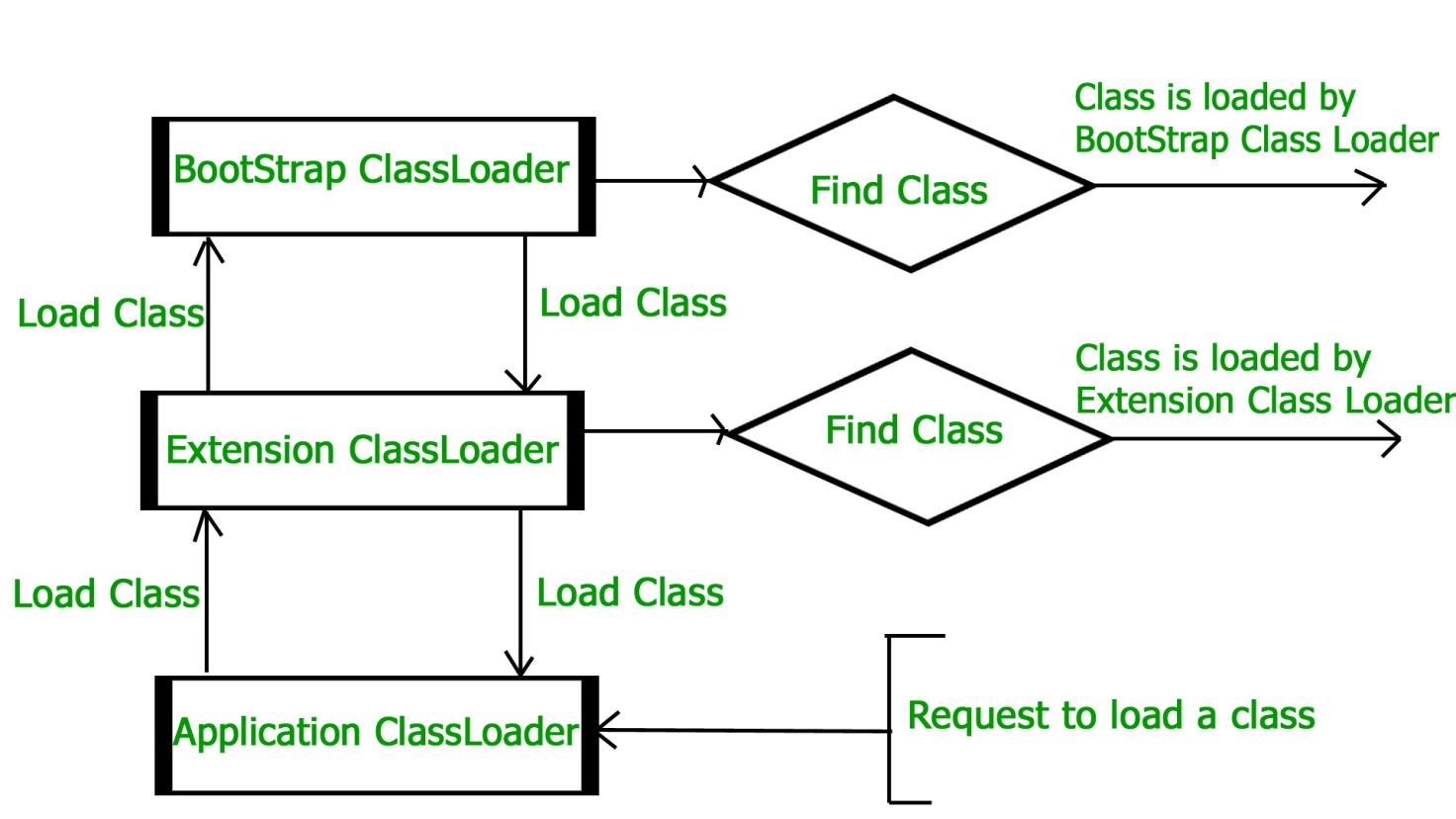
After loading the “*.class”* file, JVM creates an object of type Class to represent this file in the heap memory. Please note that this object is of type Class predefined in *java.lang.package*.

**2.1.1 Linking:** Performs verification, preparation, and (optionally) resolution. 

* *Verification*: It ensures the correctness of the *.class* file i.e. it checks whether this file is properly formatted and generated by a valid compiler or not. If verification fails, we get run-time exception *java.lang.VerifyError*. This activity is done by the component ByteCodeVerifier. Once this activity is completed then the class file is ready for compilation.
* *Preparation*: JVM allocates memory for class variables and initializing the memory to default values.
* *Resolution*: It is the process of replacing symbolic references from the type with direct references. It is done by searching into the method area to locate the referenced entity.

**2.1.2 Initialization:** In this phase, all static variables are assigned with their values defined in the code and static block(if any).

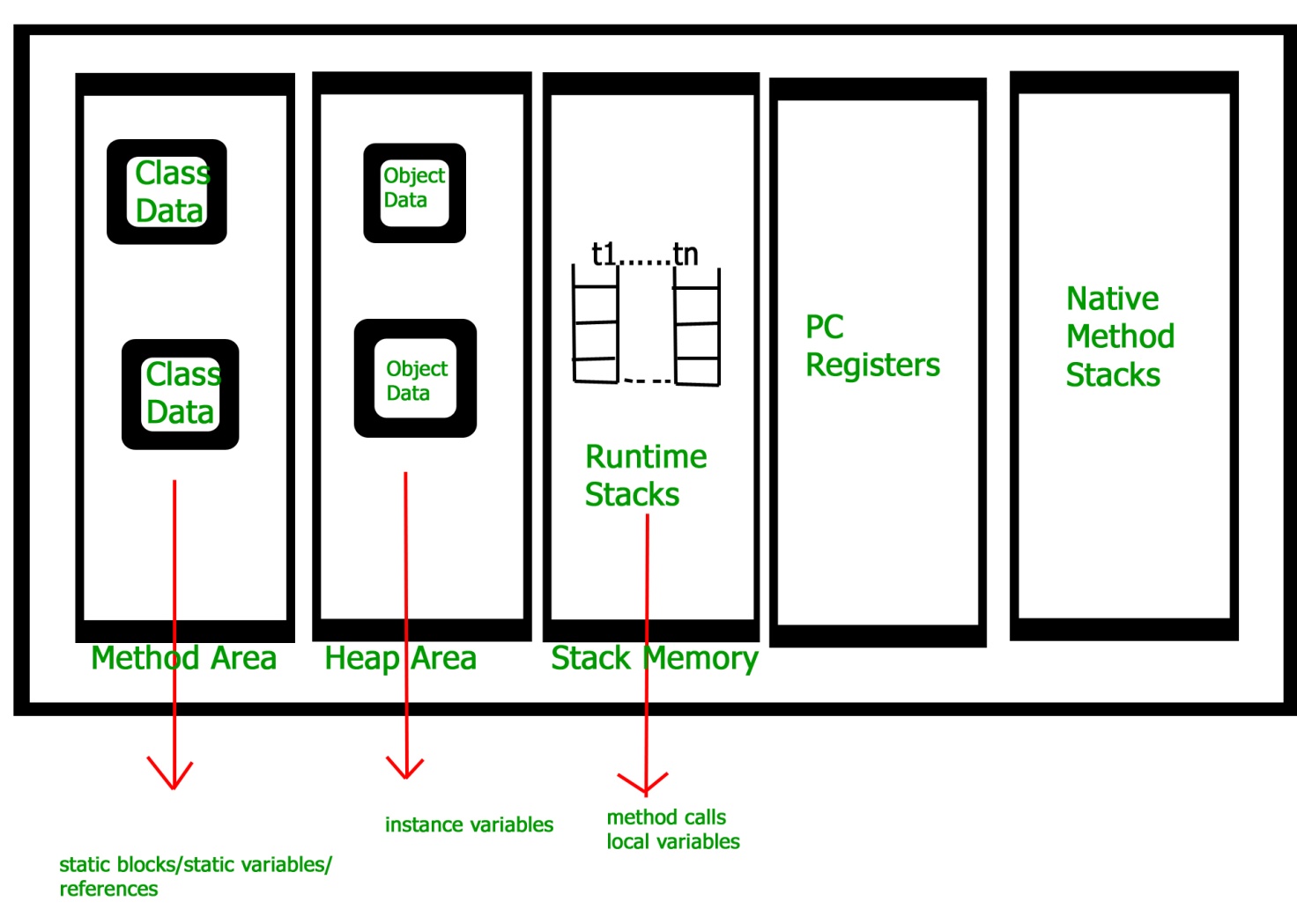
* ***Bootstrap class loader***: Every JVM implementation must have a bootstrap class loader, capable of loading trusted classes. It loads core java API classes present in the “*JAVA\_HOME/jre/lib”* directory. This path is popularly known as the bootstrap path. It is implemented in native languages like C, C++.
* ***Extension class loader***: It is a child of the bootstrap class loader. It loads the classes present in the extensions directories “*JAVA\_HOME/jre/lib/ext”*(Extension path) or any other directory specified by the java.ext.dirs system property. It is implemented in java by the *sun.misc.Launcher$ExtClassLoader* class.
* ***System/Application class loader***: It is a child of the extension class loader. It is responsible to load classes from the application classpath. It internally uses Environment Variable which mapped to java.class.path. It is also implemented in Java by the *sun.misc.Launcher$AppClassLoader* class.



## ­Fig.2.2

**2.1.3 JVM Memory**

1. Method area: In the method area, all class level information like class name, immediate parent class name, methods and variables information etc. are stored, including static variables. There is only one method area per JVM, and it is a shared resource.
2. Heap area: Information of all objects is stored in the heap area. There is also one Heap Area per JVM. It is also a shared resource.
3. Stack area: For every thread, JVM creates one run-time stack which is stored here. Every block of this stack is called activation record/stack frame which stores methods calls. All local variables of that method are stored in their corresponding frame. After a thread terminates, its run-time stack will be destroyed by JVM. It is not a shared resource.
4. PC Registers: Store address of current execution instruction of a thread. Obviously, each thread has separate PC Registers.
5. Native method stacks: For every thread, a separate native stack is created. It stores native method information.



**Fig. 2.3**

**2.1.4 Execution Engine**

Execution engine executes the “*.class”* (bytecode). It reads the byte-code line by line, uses data and information present in various memory area and executes instructions. It can be classified into three parts:

* *Interpreter*: It interprets the bytecode line by line and then executes. The disadvantage here is that when one method is called multiple times, every time interpretation is required.
* *Just-In-Time Compiler(JIT)* : It is used to increase the efficiency of an interpreter. It compiles the entire bytecode and changes it to native code so whenever the interpreter sees repeated method calls, JIT provides direct native code for that part so re-interpretation is not required, thus efficiency is improved.
* *Garbage Collector*: It destroys un-referenced objects. For more on Garbage Collector, refer Garbage Collector.

**2.1.5 Java Native Interface (JNI) :**

It is an interface that interacts with the Native Method Libraries and provides the native libraries(C, C++) required for the execution. It enables JVM to call C/C++ libraries and to be called by C/C++ libraries which may be specific to hardware.

## 2.2 How Compilers work ?

Programming languages were created to allow developers to write human-readable source code. However, computers work with machine code, which people can hardly write or read. Thus, **compilers translate the programming language’s source code to machine code dedicated to a specific machine**.

In this article, we’ll analyze the compilation process phases. Then, we’ll see the differences between compilers and interpreters. Finally, we’ll introduce examples of a few compilers of modern programming languages.

**2.2.1 Compilation Phases :**

As we already mentioned, the compilation process converts high-level source code to a low-level machine code that can be executed by the target machine. Moreover, an essential role of compilers is to inform the developer about errors committed, especially syntax-related ones.

The compilation process consists of several phases:

1. Lexical analysis
2. Syntax analysis
3. Semantic analysis
4. Intermediate code generation
5. Optimization
6. Code generation

**2.2.2 Lexical Analysis**

The first stage of the compilation process is lexical analysis. During this phase, the compiler splits source code into fragments called lexemes. **A lexeme is an abstract unit of a specific language’s lexical system.**

String greeting = "hello";

In the above statement, we have five lexemes:

1. *String*
2. *greeting*
3. *=*
4. *“hello”*
5. *;*

After splitting code into lexemes, a sequence of tokens is created. The sequence of tokens is a final product of lexical analysis. Thus, lexical analysis is also often called tokenization. **A token is an object that describes a lexeme.** It gives information about the lexeme’s purpose, such as whether it’s a keyword, variable name, or string literal. Moreover, it stores the lexeme’s source code location data.

**2.2.3 Syntax Analysis**

During syntax analysis, the compiler uses a sequence of tokens generated in the first stage. Tokens are used to create a structure called an abstract syntax tree (AST), which is a tree that represents the logical structure of a program.

**In this phase, the compiler checks a grammatic structure of the source code and its syntax correctness.** Meanwhile, any syntax errors result in a compilation error, and the compiler informs the developer about them

**2.2.4 Semantic Analysis**

**In this stage, the compiler uses an abstract syntax tree to detect any semantic errors,** for example:

* assigning the wrong type to a variable
* declaring variables with the same name in the same scope
* using an undeclared variable
* using language’s keyword as a variable name

Semantic analysis can be divided into three steps:

1. Type checking – inspects type match in assignment statements, arithmetic operations, functions, and method calls.
2. Flow control checking – investigates if flow control structures are used correctly and if classes and objects are correctly accessed.
3. Label checking – validates the use of labels and identifiers.

To achieve all the above goals, during semantic analysis, the compiler makes a complete traversal of the abstract syntax tree. Semantic analysis finally produces an annotated AST that describes the values of its attributes.

**2.2.5 Intermediate Code Generation**

During the compilation process, a compiler can generate one or more intermediate code forms.

**Intermediate code is machine-independent.**Thus, there is no need for unique compilers for every different machine**.** Besides, optimization techniques are easier to apply to intermediate code than machine code. Intermediate code has two forms of representation:

1. High-Level – similar to the source language. In this form, we can easily boost the performance of source code. However, it’s less preferred for enhancing the performance of the target machine.
2. Low-Level – close to the machine’s machine code. It’s preferred for making machine-related optimizations.

**2.2.6 Optimization**

**In the optimization phase, the compiler uses a variety of ways to enhance the efficiency of the code**. Certainly, the optimization process should follow three important rules:

1. The resulting code can’t change the original meaning of the program.
2. Optimization should focus on consuming fewer resources and speeding up the operation of the software.
3. The optimization process shouldn’t significantly impact the overall time of compilation.

Let’s see a few examples of optimization techniques:

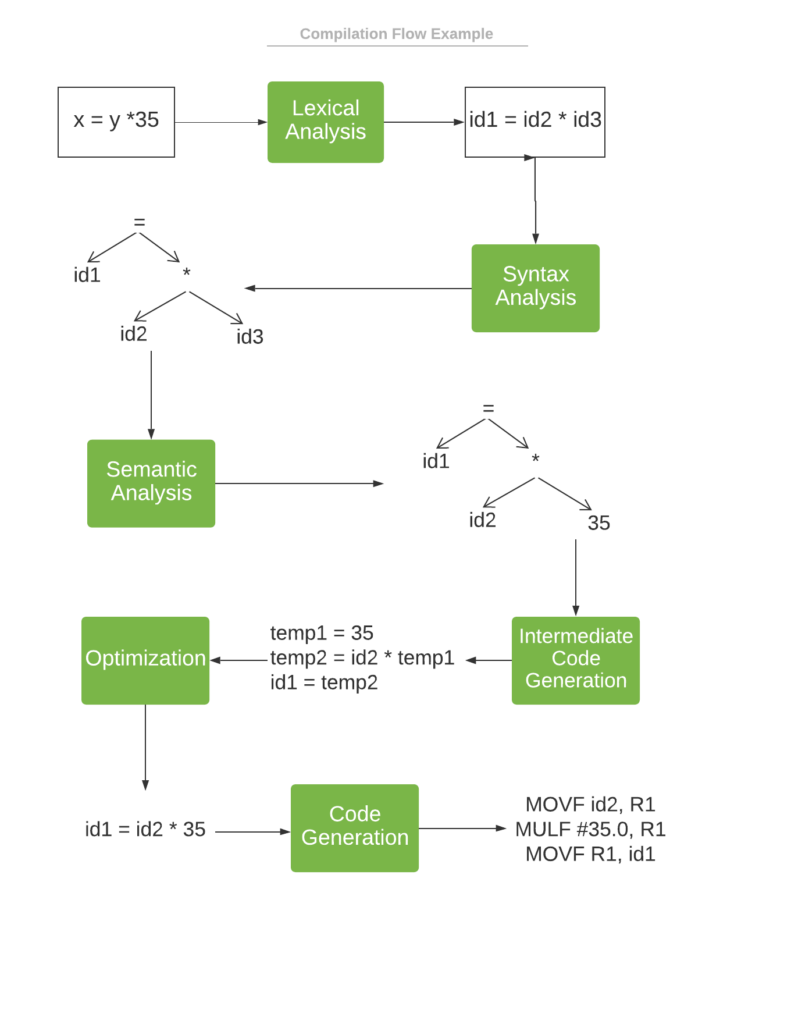
1. Function inlining – replacing the function call with its body.
2. Dead code elimination – compiler gets rid of code that is never executed, or if executed, its returned result isn’t used.
3. Loop fusion – executing, in one loop, operations from the adjacent loops that have the same iteration conditions.
4. Instruction combining – instructions realizing similar operations are combined into one; for example, *x = x + 10; x = x – 7;*could be replaced with *x = x + 3;*

**2.2.7 Code Generation**

Finally, the compiler converts the optimized intermediate code to the machine code dedicated to the target machine. The final code should have the same meaning as source code and be efficient in terms of memory and CPU resource usage. Furthermore, the code generation process must also be efficient.

**2.2.8 Practical Example**

In the below flowchart, we can see an example of the compilation process of a simple statement.



**Fig. 2.4**

### 2.3 Compiler vs Interpreter

As we already know, the compiler converts high-level source code to low-level code. Then, the target machine executes low-level code. On the other hand, **the interpreter analyzes and executes source code directly.** An interpreter usually uses one of several techniques:

1. Analyzes (parses) the source code and executes it directly.
2. Converts high-level source code into intermediate code and executes it immediately.
3. Explicitly executes stored precompiled code generated by a compiler. In this case, the compiler belongs to the interpreter system.

Let’s see a brief comparison between an interpreter and a compiler:

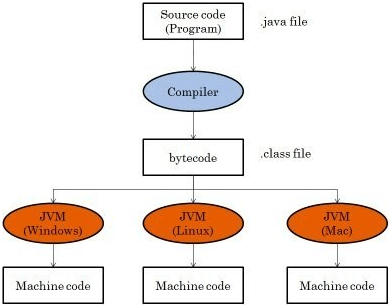
|  |  |
| --- | --- |
| COMPILER: | INTERPRETER: |
| 1. Converts the code but doesn’t execute it. | 1. Executes code directly. |
| 2. Implementing a compiler requires knowledge about the target machine. | 2. No need for knowledge about the target machine, since an interpreter executes the code. |
| 3. Each instruction is translated only once. | 3. The same instruction can be analyzed multiple times. |
| 4. The compiled program is faster to run. | 4. Interpreted programs are slower to run, but take less time to interpret than to compile and run. |
| 5. Consumes more memory due to intermediate code generation. | 5. Usually executes input code directly, thus it consumes less memory. |
| 6. Compiled language examples: Java, C++, Swift, C#. | 6. Interpreted language examples: Ruby, Lisp, PHP, PowerShell. |

## 2.4 What is Java-Bytecode ?

Java bytecode is the instruction set for the Java Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code. As soon as a java program is compiled, java bytecode is generated. In more apt terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.

**2.4.1 How does java-byte code work ?**

When we write a program in Java, firstly, the compiler compiles that program and a bytecode is generated for that piece of code. When we wish to run this .class file on any other platform, we can do so. After the first compilation, the bytecode generated is now run by the Java Virtual Machine and not the processor in consideration. This essentially means that we only need to have basic java installation on any platforms that we want to run our code on. Resources required to run the bytecode are made available by theJava Virtual Machine, which calls the processor to allocate the required resources. JVM's are stack-based so they stack implementation to read the codes.



**Fig 2.5**

**2.4.2 Advantages of Java-Bytecode ?**

Platform independence is one of the soul reasons for which James Gosling started the formation of java and it is this implementation of bytecode which helps us to achieve this. Hence bytecode is a very important component of any java program.The set of instructions for the JVM may differ from system to system but all can interpret the bytecode. A point to keep in mind is that bytecodes are non-runnable codes and rely on the availability of an interpreter to execute and thus the JVM comes into play.

Bytecode is essentially the machine level language which runs on the Java Virtual Machine. Whenever a class is loaded, it gets a stream of bytecode per method of the class. Whenever that method is called during the execution of a program, the bytecode for that method gets invoked.Javac not only compiles the program but also generates the bytecode for the program. Thus, we have realized that the bytecode implementation makes Java a **platform-independent** language. This helps to add portability to Java which is lacking in languages like C or C++. Portability ensures that Java can be implemented on a wide array of platforms like desktops, mobile devices, severs and many more. Supporting this, Sun Microsystems captioned JAVA as *"write once, read anywhere" or "WORA"* in resonance to the bytecode interpretation

**2.5 Java’s security architecture**

Java's security model is one of the language's key architectural features that makes it an appropriate technology for networked environments. Security is important because networks provide a potential avenue of attack to any computer hooked to them. This concern becomes especially strong in an environment in which software is downloaded across the network and executed locally, as is done with Java applets, for example. Because the class files for an applet are automatically downloaded when a user goes to the containing Web page in a browser, it is likely that a user will encounter applets from untrusted sources. Without any security, this would be a convenient way to spread viruses. Thus, Java's security mechanisms help make Java suitable for networks because they establish a needed trust in the safety of network-mobile code

**2.5.1 Java security model**

Java's security model is focused on protecting users from hostile programs downloaded from untrusted sources across a network. To accomplish this goal, Java provides a customizable "sandbox" in which Java programs run. A Java program must play only inside its sandbox. It can do anything within the boundaries of its sandbox, but it can't take any action outside those boundaries. The sandbox for untrusted Java applets, for example, prohibits many activities, including:

* Reading or writing to the local disk
* Making a network connection to any host, except the host from which the applet came
* Creating a new process
* Loading a new dynamic library and directly calling a native method

By making it impossible for downloaded code to perform certain actions, Java's security model protects the user from the threat of hostile code.

**2.5.2 The sandbox defined**

Traditionally, you had to trust software before you ran it. You achieved security by being careful only to use software from trusted sources, and by regularly scanning for viruses just to make sure things were safe. Once some software got access to your system, it had full rein. If it was malicious, it could do a great deal of damage to your system because there were no restrictions placed on the software by the runtime environment of your computer. So, in the traditional security scheme, you tried to prevent malicious code from ever gaining access to your computer in the first place.

The sandbox security model makes it easier to work with software that comes from sources you don't fully trust. Instead of security being established by requiring you to prevent any code you don't trust from ever making its way onto your computer, the sandbox model lets you welcome code from any source. But as it's running, the sandbox restricts code from untrusted sources from taking any actions that could possibly harm your system. The advantage is you don't need to figure out what code you can and can't trust, and you don't need to scan for viruses. The sandbox itself prevents any viruses or other malicious code you may invite into your computer from doing any damage.

**2.5.3 The sandbox is pervasive**

If you have a properly skeptical mind, you'll need to be convinced that a sandbox has no leaks before you trust it to protect you. To make sure the sandbox has no leaks, Java's security model involves every aspect of its architecture. If there were areas in Java's architecture in which security was weak, a malicious programmer (a "cracker") potentially could exploit those areas to "go around" the sandbox. To understand the sandbox, therefore, you must look at several different parts of Java's architecture and understand how they work together.

The fundamental components responsible for Java's sandbox are:

* Safety features built into the Java virtual machine (and the language)
* The class loader architecture
* The class file verifier
* The security manager and the Java API

**3. METHODOLOGY USED**

**3.1 How we use this tool `decomp-java` ?**

Before using of this tool ,we need to clear some knowledge of how we can analyse java byte code file to reverse them into source code,

I made vulnerable labs having these components :

* Based on the usage of this tool
* You need to find security vulnerability by reversing the jar file given.
* Web app vulnerabilities cause this is web app.
* Need to find any vulnerability related to java or web app server.
* The main motive is to get rce

**3.2 What are requirements of the vulnerable labs ?**

1. To show attack work you need to get rce from server**.**
2. I already implemented netcat utility,which can help you get reverse shell from server.
3. In order to show your attack work need to get flag.txt file in server.

To understand the working of this tool you there are some knowledge and prequirestics required firs t ?

## 3.3 What is decompilation ?

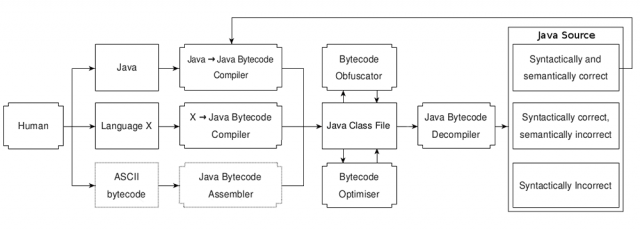
Compilation is the act of transforming a high-level language, into a low-level language such as machine code or bytecode. Decompilation is the reverse. It is the act of transforming a low-level language into a high-level language. Java source code is compiled into an intermediate language known as Java bytecode.

A Java virtual machine executes Java bytecode in class files conforming to the class file specification which is part of the Java Virtual Machine Specification and updated in JSR202. The open specification allows tools other than Sun’s Java compiler to generate and/or manipulate Java bytecode. Java bytecode can be generated in three ways:

1. from a Java source program using a Java compiler (such as Sun’s javac),
2. using a language other than Java to Java Bytecode compiler (such as JGNAT) or
3. by writing a class file by hand.

Java bytecode can also be manipulated by tools such as obfuscators and optimisers which perform semantics-preserving transformations on bytecode contained within Java class file.

Figure I shows the Java bytecode cycle from generation to decompilation to Java source. Java bytecode retains type information about fields, method returns and parameters but it does not, for example, contain type information for local variables. The type information in the Java class file renders the task of decompilation of bytecode easier than decompilation of machine code. Decompiling Java bytecode, thus, requires analysis of most local variable types, flattening of stackbased instructions and structuring of loops and conditionals.



**Fig.2.1**

The task of bytecode decompilation, however, is much harder than compilation. We show that often decompilers cannot fully perform their intended function. Decompilation has many applications including legitimate uses, such as the recovery of lost source code for a crucial application and non-legitimate uses such as reverseengineering a proprietary application. Consider the case in which a company has lost the source code for their application and hence to continue development on the software they require recovery of source code from Java class files.

The company must decompile the Java class files and attempt to recover Java source equivalent to the originally lost source. In this case, in comparison to an illegitimate use, it is likely that the company knows more about how the Java class files were generated. Knowledge of how class files are generated provides information useful in the recovery of the original source as a decompiler can be optimised for the compiler used.

If the purpose of decompilation is to simply understand a program, the syntactical correctness of a complete decompiled program may not be a high priority. Correct portions of an incorrect program could help in the understanding of a program, in contrast to the case of source recovery where correct source is needed.

**3.4 Brief details on reverse engineering ?**

**Reverse engineering** (also known as **backwards engineering** or **back engineering**) is a process or method through which one attempts to understand through deductive reasoning how a previously made device, process, system, or piece of software accomplishes a task with very little (if any) insight into exactly how it does so.

**3.4.1 Great Overview**

There are many reasons for performing reverse engineering in various fields. Reverse engineering has its origins in the analysis of hardware for commercial or military advantage. However, the reverse engineering process, as such, is not concerned with creating a copy or changing the artifact in some way. It is only an analysis to deduce design features from products with little or no additional knowledge about the procedures involved in their original production.

In some cases, the goal of the reverse engineering process can simply be a redocumentation of legacy systems. Even when the reverse-engineered product is that of a competitor, the goal may not be to copy it but to perform competitor analysis. Reverse engineering may also be used to create interoperable products and despite some narrowly-tailored United States and European Union legislation, the legality of using specific reverse engineering techniques for that purpose has been hotly contested in courts worldwide for more than two decades.

Software reverse engineering can help to improve the understanding of the underlying source code for the maintenance and improvement of the software, relevant information can be extracted to make a decision for software development and graphical representations of the code can provide alternate views regarding the source code, which can help to detect and fix a software bug or vulnerability. Frequently, as some software develops, its design information and improvements are often lost over time, but that lost information can usually be recovered with reverse engineering. The process can also help to cut down the time required to understand the source code, thus reducing the overall cost of the software development. Reverse engineering can also help to detect and to eliminate a malicious code written to the software with better code detectors. Reversing a source code can be used to find alternate uses of the source code, such as detecting the unauthorized replication of the source code where it was not intended to be used, or revealing how a competitor's product was built. That process is commonly used for "cracking" software and media to remove their copy protection, or to create a possibly-improved copy or even a knockoff, which is usually the goal of a competitor or a hacker.

Malware developers often use reverse engineering techniques to find vulnerabilities in an operating system to build a computer virus that can exploit the system vulnerabilities. Reverse engineering is also being used in cryptanalysis to find vulnerabilities in substitution cipher, symmetric-key algorithm or public-key cryptography.

There are other uses to reverse engineering:

* **Interfacing**. Reverse engineering can be used when a system is required to interface to another system and how both systems would negotiate is to be established. Such requirements typically exist for interoperability.
* **Military or commercial espionage**. Learning about an enemy's or competitor's latest research by stealing or capturing a prototype and dismantling it may result in the development of a similar product or a better countermeasure against it.
* **Obsolescence**. Integrated circuits are often designed on proprietary systems and built on production lines, which become obsolete in only a few years. When systems using those parts can no longer be maintained since the parts are no longer made, the only way to incorporate the functionality into new technology is to reverse-engineer the existing chip and then to redesign it using newer tools by using the understanding gained as a guide. Another obsolescence originated problem that can be solved by reverse engineering is the need to support (maintenance and supply for continuous operation) existing legacy devices that are no longer supported by their original equipment manufacturer. The problem is particularly critical in military operations.
* **Product security analysis**. That examines how a product works by determining the specifications of its components and estimate costs and identifies potential patent infringement. Also part of product security analysis is acquiring sensitive data by disassembling and analyzing the design of a system component. Another intent may be to remove copy protection or to circumvent access restrictions.
* **Competitive technical intelligence**. That is to understand what one's competitor is actually doing, rather than what it says that it is doing.
* **Saving money**. Finding out what a piece of electronics can do may spare a user from purchasing a separate product.
* **Repurposing**. Obsolete objects are then reused in a different-but-useful manner.
* **Design**. Production and design companies applied Reverse Engineering to practical craft-based manufacturing process. The companies can work on “historical” manufacturing collections through 3D scanning, 3D re-modeling and re-design. In 2013 Italian manufactures Baldi and Savio Firmino together with University of Florence optimized their innovation, design, and production processes.

**3.4.2 Common situations of reverse engineering ?**

1. **Machines**

As computer-aided design (CAD) has become more popular, reverse engineering has become a viable method to create a 3D virtual model of an existing physical part for use in 3D CAD, CAM, CAE, or other software. The reverse-engineering process involves measuring an object and then reconstructing it as a 3D model. The physical object can be measured using 3D scanning technologies like CMMs, laser scanners, structured light digitizers, or industrial CT scanning (computed tomography). The measured data alone, usually represented as a point cloud, lacks topological information and design intent. The former may be recovered by converting the point cloud to a triangular-faced mesh. Reverse engineering aims to go beyond producing such a mesh and to recover the design intent in terms of simple analytical surfaces where appropriate (planes, cylinders, etc.) as well as possibly NURBS surfaces to produce a boundary-representation CAD model. Recovery of such a model allows a design to be modified to meet new requirements, a manufacturing plan to be generated, etc.

Hybrid modeling is a commonly used term when NURBS and parametric modeling are implemented together. Using a combination of geometric and freeform surfaces can provide a powerful method of 3D modeling. Areas of freeform data can be combined with exact geometric surfaces to create a hybrid model. A typical example of this would be the reverse engineering of a cylinder head, which includes freeform cast features, such as water jackets and high-tolerance machined areas.

Reverse engineering is also used by businesses to bring existing physical geometry into digital product development environments, to make a digital 3D record of their own products, or to assess competitors' products. It is used to analyze how a product works, what it does, what components it has; estimate costs; identify potential patent infringement; etc.

Value engineering, a related activity that is also used by businesses, involves deconstructing and analyzing products. However, the objective is to find opportunities for cost-cutting.

1. **Software**

In 1990, the Institute of Electrical and Electronics Engineers (IEEE) defined (software) reverse engineering (SRE) as "the process of analyzing a subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction" in which the "subject system" is the end product of software development. Reverse engineering is a process of examination only, and the software system under consideration is not modified, which would otherwise be re-engineering or restructuring. Reverse engineering can be performed from any stage of the product cycle, not necessarily from the functional end product.

There are two components in reverse engineering: redocumentation and design recovery. Redocumentation is the creation of new representation of the computer code so that it is easier to understand. Meanwhile, design recovery is the use of deduction or reasoning from general knowledge or personal experience of the product to understand the product's functionality fully. It can also be seen as "going backwards through the development cycle." In this model, the output of the implementation phase (in source code form) is reverse-engineered back to the analysis phase, in an inversion of the traditional waterfall model. Another term for this technique is program comprehension. The Working Conference on Reverse Engineering (WCRE) has been held yearly to explore and expand the techniques of reverse engineering. Computer-aided software engineering (CASE) and automated code generation have contributed greatly in the field of reverse engineering.

Software anti-tamper technology like obfuscation is used to deter both reverse engineering and re-engineering of proprietary software and software-powered systems. In practice, two main types of reverse engineering emerge. In the first case, source code is already available for the software, but higher-level aspects of the program, which are perhaps poorly documented or documented but no longer valid, are discovered. In the second case, there is no source code available for the software, and any efforts towards discovering one possible source code for the software are regarded as reverse engineering. The second usage of the term is more familiar to most people. Reverse engineering of software can make use of the clean room design technique to avoid copyright infringement.

On a related note, black box testing in software engineering has a lot in common with reverse engineering. The tester usually has the API but has the goals to find bugs and undocumented features by bashing the product from outside.

Other purposes of reverse engineering include security auditing, removal of copy protection ("cracking"), circumvention of access restrictions often present in consumer electronics, customization of embedded systems (such as engine management systems), in-house repairs or retrofits, enabling of additional features on low-cost "crippled" hardware (such as some graphics card chip-sets), or even mere satisfaction of curiosity.

1. **Binary software**

Binary reverse engineering is performed if source code for a software is unavailable. This process is sometimes termed *reverse code engineering*, or RCE. For example, decompilation of binaries for the Java platform can be accomplished by using Jad. One famous case of reverse engineering was the first non-IBM implementation of the PC BIOS, which launched the historic IBM PC compatible industry that has been the overwhelmingly-dominant computer hardware platform for many years. Reverse engineering of software is protected in the US by the fair use exception in copyright law. The Samba software, which allows systems that do not run Microsoft Windows systems to share files with systems that run it, is a classic example of software reverse engineering since the Samba project had to reverse-engineer unpublished information about how Windows file sharing worked so that non-Windows computers could emulate it. The Wine project does the same thing for the Windows API, and OpenOffice.org is one party doing that for the Microsoft Office file formats. The ReactOS project is even more ambitious in its goals by striving to provide binary (ABI and API) compatibility with the current Windows operating systems of the NT branch, which allows software and drivers written for Windows to run on a clean-room reverse-engineered free software (GPL) counterpart. WindowsSCOPE allows for reverse-engineering the full contents of a Windows system's live memory including a binary-level, graphical reverse engineering of all running processes.

Another classic, if not well-known, example is that in 1987 Bell Laboratories reverse-engineered the Mac OS System 4.1, originally running on the Apple Macintosh SE, so that it could run it on RISC machines of their own.

**Binary software techniques**

Reverse engineering of software can be accomplished by various methods. The three main groups of software reverse engineering are

1. Analysis through observation of information exchange, most prevalent in protocol reverse engineering, which involves using bus analyzers and packet sniffers, such as for accessing a computer bus or computer network connection and revealing the traffic data thereon. Bus or network behavior can then be analyzed to produce a standalone implementation that mimics that behavior. That is especially useful for reverse engineering device drivers. Sometimes, reverse engineering on embedded systems is greatly assisted by tools deliberately introduced by the manufacturer, such as JTAG ports or other debugging means. In Microsoft Windows, low-level debuggers such as SoftICE are popular.
2. Disassembly using a disassembler, meaning the raw machine language of the program is read and understood in its own terms, only with the aid of machine-language mnemonics. It works on any computer program but can take quite some time, especially for those who are not used to machine code. The Interactive Disassembler is a particularly popular tool.
3. Decompilation using a decompiler, a process that tries, with varying results, to recreate the source code in some high-level language for a program only available in machine code or bytecode.
4. **Software classification**

Software classification is the process of identifying similarities between different software binaries (such as two different versions of the same binary) used to detect code relations between software samples. The task was traditionally done manually for several reasons (such as patch analysis for vulnerability detection and copyright infringement), but it can now be done somewhat automatically for large numbers of samples.

This method is being used mostly for long and thorough reverse engineering tasks (complete analysis of a complex algorithm or big piece of software). In general, statistical classification is considered to be a hard problem, which is also true for software classification, and so few solutions/tools that handle this task well.

1. **Source code**[edit]

A number of UML tools refer to the process of importing and analysing source code to generate UML diagrams as "reverse engineering." See List of UML tools.

Although UML is one approach in providing "reverse engineering" more recent advances in international standards activities have resulted in the development of the Knowledge Discovery Metamodel (KDM). The standard delivers an ontology for the intermediate (or abstracted) representation of programming language constructs and their interrelationships. An Object Management Group standard (on its way to becoming an ISO standard as well), KDM has started to take hold in industry with the development of tools and analysis environments that can deliver the extraction and analysis of source, binary, and byte code. For source code analysis, KDM's granular standards' architecture enables the extraction of software system flows (data, control, and call maps), architectures, and business layer knowledge (rules, terms, and process). The standard enables the use of a common data format (XMI) enabling the correlation of the various layers of system knowledge for either detailed analysis (such as root cause, impact) or derived analysis (such as business process extraction). Although efforts to represent language constructs can be never-ending because of the number of languages, the continuous evolution of software languages, and the development of new languages, the standard does allow for the use of extensions to support the broad language set as well as evolution. KDM is compatible with UML, BPMN, RDF, and other standards enabling migration into other environments and thus leverage system knowledge for efforts such as software system transformation and enterprise business layer analysis.

1. **Protocols**

Protocols are sets of rules that describe message formats and how messages are exchanged: the protocol state machine. Accordingly, the problem of protocol reverse-engineering can be partitioned into two subproblems: message format and state-machine reverse-engineering.

The message formats have traditionally been reverse-engineered by a tedious manual process, which involved analysis of how protocol implementations process messages, but recent research proposed a number of automatic solutions. Typically, the automatic approaches group observe messages into clusters by using various clustering analyses, or they emulate the protocol implementation tracing the message processing.

There has been less work on reverse-engineering of state-machines of protocols. In general, the protocol state-machines can be learned either through a process of offline learning, which passively observes communication and attempts to build the most general state-machine accepting all observed sequences of messages, and online learning, which allows interactive generation of probing sequences of messages and listening to responses to those probing sequences. In general, offline learning of small state-machines is known to be NP-complete, but online learning can be done in polynomial time. An automatic offline approach has been demonstrated by Comparetti et al. and an online approach by Cho et al.

Other components of typical protocols, like encryption and hash functions, can be reverse-engineered automatically as well. Typically, the automatic approaches trace the execution of protocol implementations and try to detect buffers in memory holding unencrypted packets.

1. **Integrated circuits/smart cards**

Reverse engineering is an invasive and destructive form of analyzing a smart card. The attacker uses chemicals to etch away layer after layer of the smart card and takes pictures with a scanning electron microscope (SEM). That technique can reveal the complete hardware and software part of the smart card. The major problem for the attacker is to bring everything into the right order to find out how everything works. The makers of the card try to hide keys and operations by mixing up memory positions, such as by bus scrambling.

In some cases, it is even possible to attach a probe to measure voltages while the smart card is still operational. The makers of the card employ sensors to detect and prevent that attack. That attack is not very common because it requires both a large investment in effort and special equipment that is generally available only to large chip manufacturers. Furthermore, the payoff from this attack is low since other security techniques are often used such as shadow accounts. It is still uncertain whether attacks against chip-and-PIN cards to replicate encryption data and then to crack PINs would provide a cost-effective attack on multifactor authentication.

Full reverse engineering proceeds in several major steps.

The first step after images have been taken with a SEM is stitching the images together, which is necessary because each layer cannot be captured by a single shot. A SEM needs to sweep across the area of the circuit and take several hundred images to cover the entire layer. Image stitching takes as input several hundred pictures and outputs a single properly-overlapped picture of the complete layer.

Next, the stitched layers need to be aligned because the sample, after etching, cannot be put into the exact same position relative to the SEM each time. Therefore, the stitched versions will not overlap in the correct fashion, as on the real circuit. Usually, three corresponding points are selected, and a transformation applied on the basis of that.

To extract the circuit structure, the aligned, stitched images need to be segmented, which highlights the important circuitry and separates it from the uninteresting background and insulating materials.

Finally, the wires can be traced from one layer to the next, and the netlist of the circuit, which contains all of the circuit's information, can be reconstructed.

**11 Military Applications**

Reverse engineering is often used by people to copy other nations' technologies, devices, or information that have been obtained by regular troops in the fields or by intelligence operations. It was often used during the Second World War and the Cold War. Here are well-known examples from the Second World War and later:

* Jerry can: British and American forces in WW2 noticed that the Germans had gasoline cans with an excellent design. They reverse-engineered copies of those cans, which cans were popularly known as "Jerry cans."
* Panzerschreck: The Germans captured an American bazooka during the Second World War and reverse engineered it to create the larger Panzerschreck.
* Tupolev Tu-4: In 1944, three American B-29 bombers on missions over Japan were forced to land in the Soviet Union. The Soviets, who did not have a similar strategic bomber, decided to copy the B-29. Within three years, they had developed the Tu-4, a nearly-perfect copy.
* SCR-584 radar: copied by the Soviet Union after the Second World War, it is known for a few modifications - СЦР-584, Бинокль-Д.
* V-2 rocket: Technical documents for the V-2 and related technologies were captured by the Western Allies at the end of the war. The Americans focused their reverse engineering efforts via Operation Paperclip, which led to the development of the PGM-11 Redstone rocket. The Soviets used captured German engineers to reproduce technical documents and plans and worked from captured hardware to make their clone of the rocket, the R-1. Thus began the postwar Soviet rocket program, which led to the R-7 and the beginning of the space race.
* K-13/R-3S missile (NATO reporting name AA-2 Atoll), a Soviet reverse-engineered copy of the AIM-9 Sidewinder, was made possible after a Taiwanese (ROCAF) AIM-9B hit a Chinese PLA MiG-17 without exploding in September 1958. The missile became lodged within the airframe, and the pilot returned to base with what Soviet scientists would describe as a university course in missile development.
* BGM-71 TOW missile: In May 1975, negotiations between Iran and Hughes Missile Systems on co-production of the TOW and Maverick missiles stalled over disagreements in the pricing structure, the subsequent 1979 revolution ending all plans for such co-production. Iran was later successful in reverse-engineering the missile and now produces its own copy, the Toophan.
* China has reversed engineered many examples of Western and Russian hardware, from fighter aircraft to missiles and HMMWV cars, such as the MiG-15 (which became the J-7) and the Su-33 (which became the J-15). More recent analyses of China's military growth have pointed to the inherent limitations of habitual reverse engineering for advanced weapon systems.
* During the Second World War, Polish and British cryptographers studied captured German "Enigma" message encryption machines for weaknesses. Their operation was then simulated on electromechanical devices, "bombes", which tried all the possible scrambler settings of the "Enigma" machines that helped the breaking of coded messages that had been sent by the Germans.
* Also during the Second World War, British scientists analyzed and defeated a series of increasingly-sophisticated radio navigation systems used by the Luftwaffe to perform guided bombing missions at night. The British countermeasures to the system were so effective that in some cases, German aircraft were led by signals to land at RAF bases since they believed that they had returned to German territory.

**3.5 What is security audit ?**

A security audit is a systematic evaluation of the security of a company's information system by measuring how well it conforms to an established set of criteria. A thorough audit typically assesses the security of the system's physical configuration and environment, software, information handling processes and user practices.

Security audits are often used to determine compliance with regulations such as the Health Insurance Portability and Accountability Act, the Sarbanes-Oxley Act and the California Security Breach Information Act that specify how organizations must deal with information.

These audits are one of three main types of security diagnostics, along with vulnerability assessments and penetration testing. Security audits measure an information system's performance against a list of criteria. A vulnerability assessment is a comprehensive study of an information system, seeking potential security weaknesses. Penetration testing is a covert approach in which a security expert tests to see if a system can withstand a specific attack. Each approach has inherent strengths and using two or more in conjunction may be the most effective approach.

Organizations should construct a security audit plan that is repeatable and updateable. Stakeholders must be included in the process for the best outcome.

**3.5.1 Why do a security audit ?**

There are several reasons to do a security audit. They include these six goals:

1. Identify security problems and gaps, as well as system weaknesses.
2. Establish a security baseline that future audits can be compared with.
3. Comply with internal organization security policies.
4. Comply with external regulatory requirements.
5. Determine if security training is adequate.
6. Identify unnecessary resources.

Security audits will help protect critical data, identify security loopholes, create new security policies and track the effectiveness of security strategies. Regular audits can help ensure employees stick to security practices and can catch new vulnerabilities.

**3.5.2 When is a security audit needed ?**

How often an organization does its security audits depends on the industry it is in, the demands of its business and corporate structure, and the number of systems and applications that must be audited. Organizations that handle a lot of sensitive data -- such as financial services and heathcare providers -- are likely to do audits more frequently. Ones that use only one or two applications will find it easier to conduct security audits and may do them more frequently. External factors, such as regulatory requirements, affect audit frequency, as well.

Many companies will do a security audit at least once or twice a year. But they can also be done monthly or quarterly. Different departments may have different audit schedules, depending on the systems, applications and data they use. Routine audits -- whether done annually or monthly -- can help identify anomalies or patterns in a system.

Quarterly or monthly audits may be more than most organizations have the time or resources for, however. The determining factors in how often an organization chooses to do security audits depends on the complexity of the systems used and the type and importance of the data in that system. If the data in a system is deemed essential, then that system may be audited more often, but complicated systems that take time to audit may be audited less frequently.

An organization should conduct a special security audit after a data breach, system upgrade or data migration, or when changes to compliance laws occur, when a new system has been implemented or when the business grows by more than a defined amount of users. These one-time audits may focus on a specific area where the event may have opened security vulnerabilities. For example, if a data breach just occurred, an audit of the affected systems can help determine what went wrong.

**3.5.3 Types of security audits?**

Security audits come in two forms, internal and external audits, that involve the following procedures:

* **Internal audits.**In these audits, a business uses its own resources and internal audit department. Internal audits are used when an organization wants to validate business systems for policy and procedure compliance.
* **External audits.** With these audits, an outside organization is brought in to conduct an audit. External audits are also conducted when an organization needs to confirm it is conforming to industry standards or government regulations.

There are two subcategories of external audits: second- and third-party audits. Second-party audits are conducted by a supplier of the organization being audited. Third-party audits are done by an independent, unbiased group, and the auditors involved have no association with the organization under audit.

**3.5.4 What systems does an audit cover?**

During a security audit, each system an organization uses may be examined for vulnerabilities in the following areas:

* **Network vulnerabilities.**Auditors look for weaknesses in any network component that an attacker could exploit to access systems or information or cause damage. Information as it travels between two points is particularly vulnerable. Security audits and regular network monitoring keep track of network traffic, including emails, instant messages, files and other communications. Network availability and access points are also included in this part of the audit.
* **Security controls.**With this part of the audit, the auditor looks at how effective a company's security controls are. That includes evaluating how well an organization has implemented the policies and procedures it has established to safeguard its information and systems. For example, an auditor may check to see if the company retains administrative control over its mobile devices. The auditor tests the company's controls to make sure they are effective and that the company is following its own policies and procedures.
* **Encryption.** This part of the audit verifies that an organization has controls in place to manage data encryption processes.
* **Software systems.** Here, software systems are examined to ensure they are working properly and providing accurate information. They are also checked to ensure controls are in place to prevent unauthorized users from gaining access to private data. The areas examined include data processing, software development and computer systems.
* **Architecture management capabilities.**Auditors verify that IT management has organizational structures and procedures in place to create an efficient and controlled environment to process information.
* **Telecommunications controls.** Auditors check that telecommunications controls are working on both client and server sides, as well as on the network that connects them.
* **Systems development audit.** Audits covering this area verify that any systems under development meet security objectives set by the organization. This part of the audit is also done to ensure that systems under development are following set standards.
* **Information processing.** These audits verify that data processing security measures are in place.

**3.5.5 Steps involved in a security audit**

These five steps are generally part of a security audit:

1. **Agree on goals.**Include all stakeholders in discussions of what should be achieved with the audit.
2. **Define the scope of the audit.** List all assets to be audited, including computer equipment, internal documentation and processed data.
3. **Conduct the audit and identify threats.** List potential threats related to each Threats can include the loss of data, equipment or records through natural disasters, malware or unauthorized users.
4. **Evaluate security and risks.** Assess the risk of each of the identified threats happening, and how well the organization can defend against them.
5. **Determine the needed controls.** Identify what security measures must be implemented or improved to minimize risks.

**3.5.6 Test vs assement vs audit**

Audits are a separate concept from other practices such as tests and assessments. An audit is a way to validate that an organization is adhering to procedures and security policies set internally, as well as those that standards groups and regulatory agencies set. Organizations can conduct audits themselves or bring in third parties to do them. Security audit best practices are available from various industry organizations.

A test, such as a penetration test, is a procedure to check that a specific system is working as it should. IT professionals doing the testing are looking for gaps that might open vulnerabilities. With a pen test, for instance, the security analyst is hacking into the system in the same way that a threat actor might, to determine what an attacker can see and access.

An assessment is a planned test such as a risk or vulnerability assessment. It looks at how a system should operate and then compares that to the system's current operational state. For example, a vulnerability assessment of a computer system checks the status of the security measures protecting that system and whether they are responding the way they should.

**4. OVER-ALL DEPTH INFORMATION ABOUT PROJECT**

**4.1 What is “decomp-java” ?**

**Decomp-java**  is tool developed in java, aim to decompile java compiled code ,powered by java great “procyon” library, To build this tool in your system should have java installed and full environment setup with java and also you should have maven and gradle setup.

**Features of “decomp-java”**

* Robust user-friendly Gui.
* Graphic user interface tool
* High performance apis to decompile .class code in one instant
* No need any setup you can run with .jar.
* Menu options
* Search functionality
* Backward and forward functionality for navigation from code to code.

**4.2 Technologies used in decomp-java and Vulnerable labs included for demonstration purpose.**

**In decomp-java :**

1. **JAVA -** Main programing language.
2. **ANTLR :-**ANTLR (ANother Tool for Language Recognition) is a tool for processing structured text
3. **Fife –**For ui parsing and showing.
4. **Gradle -** For building the app
5. **Java-Swing –** for building window-based gui.
6. **Java-Awt –** Abstract window toolkit
7. **Procyon –**Java Decompiler

**In Vulnerable labs :**

1. **Docker -** To deploy the container of vulnerable labs which is sandbox from main host.
2. **Docker-Compose -** To run containers with single command.
3. **Spring-Web-Framework –** Java Framework to build web apps.

**4.3 Introduction to various technologies used in decomp-java and Vulnerable labs**



**Fig.4.1**

**4.3.1 JAVA**

**JAVA** was developed by James Gosling at **Sun Microsystems** Inc in the year **1991**, later acquired by Oracle Corporation. It is a simple programming language. Java makes writing, compiling, and debugging programming easy. It helps to create reusable code and modular programs.

Java is a class-based, object-oriented programming language and is designed to have as few implementation dependencies as possible. A general-purpose programming language made for developers to *write once run anywhere* that is compiled Java code can run on all platforms that support Java. Java applications are compiled to byte code that can run on any Java Virtual Machine. The syntax of Java is similar to c/c++.

* **History**

Java’s history is very interesting. It is a programming language created in 1991.James Gosling, Mike Sheridan, and Patrick Naughton, a team of Sun engineers known as the **Green team**initiated the Java language in 1991. **Sun Microsystems** released its first public implementation in 1996 as **Java 1.0**. It provides no-cost -run-times on popular platforms. Java1.0 compiler was re-written in Java by Arthur Van Hoff to strictly comply with its specifications. With the arrival of Java 2, new versions had multiple configurations built for different types of platforms.

In 1997, Sun Microsystems approached the ISO standards body and later formalized Java, but it soon withdrew from the process. At one time, Sun made most of its Java implementations available without charge, despite their proprietary software status. Sun generated revenue from Java through the selling of licenses for specialized products such as the Java Enterprise System.

On November 13, 2006, Sun released much of its Java virtual machine as free, open-source software. On May 8, 2007, Sun finished the process, making all of its JVM’s core code available under open-source distribution terms.

The principles for creating java were simple, robust, secured, high performance, portable, multi-threaded, interpreted, dynamic, etc. **James**Gosling in 1995 developed Java, who is known as the Father of Java. Currently, Java is used in mobile devices, internet programming, games, e-business, etc.

* **Java programming language is named JAVA. Why?**

After the name OAK, the team decided to give a new name to it and the suggested words were Silk, Jolt, revolutionary, DNA, dynamic, etc. These all names were easy to spell and fun to say, but they all wanted the name to reflect the essence of technology. In accordance with James Gosling, **Java** the among the top names along with **Silk**, and since java was a unique name so most of them preferred it.

Java is the name of an **island** in Indonesia where the first coffee(named java coffee) was produced. And this name was chosen by James Gosling while having coffee near his office. Note that Java is just a name, not an acronym.

* **Java Terminology**

Before learning Java, one must be familiar with these common terms of Java.

**1.  Java Virtual Machine(JVM):** This is generally referred to as JVM. There are three execution phases of a program. They are written, compile and run the program.

* Writing a program is done by a java programmer like you and me.
* The compilation is done by the **JAVAC** compiler which is a primary Java compiler included in the Java development kit (JDK). It takes Java program as input and generates bytecode as output.
* In the Runningphase of a program,**JVM** executes the bytecode generated by the compiler.

Now, we understood that the function of Java Virtual Machine is to execute the bytecode produced by the compiler. Every Operating System has a different JVM but the output they produce after the execution of bytecode is the same across all the operating systems. This is why Java is known as a**platform-independent language.**

**2. Bytecode in**the **Development process:**As discussed, the Javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. It is saved as **.class** file by the compiler. To view the bytecode, a disassembler like javap can be used.

**3. Java Development Kit(JDK):**While we were using the term JDK, when we learn about bytecode and JVM . So, as the name suggests, it is a complete Java development kit that includes everything including compiler, Java Runtime Environment (JRE), java debuggers, java docs, etc. For the program to execute in java, we need to install JDK on our computer in order to create, compile and run the java program.

**4. Java Runtime Environment (JRE):**JDK includes JRE. JRE installation on our computers allows the java program to run, however, we cannot compile it. JRE includes a browser, JVM, applet supports, and plugins. For running the java program, a computer needs JRE.

**5. Garbage Collector:**In Java, programmers can’t delete the objects. To delete or recollect that memory JVM has a program called Garbage Collector. Garbage Collectors can recollect the of objects that are not referenced. So Java makes the life of a programmer easy by handling memory management. However, programmers should be careful about their code whether they are using objects that have been used for a long time. Because Garbage cannot recover the memory of objects being referenced.

**6. ClassPath:**The classpath is the file path where the java runtime and Java compiler look for **.class** files to load. By default, JDK provides many libraries. If you want to include external libraries they should be added to the classpath.

* **Primary/Main Features of Java**

**1. Platform Independent:**Compiler converts source code to bytecode and then the JVM executes the bytecode generated by the compiler. This bytecode can run on any platform be it Windows, Linux, macOS which means if we compile a program on Windows, then we can run it on Linux and vice versa. Each operating system has a different JVM, but the output produced by all the OS is the same after the execution of bytecode. That is why we call java a platform-independent language.

**2. Object-Oriented Programming Language:**Organizing the program in the terms of collection of objects is a way of object-oriented programming, each of which represents an instance of the class.

The four main concepts of Object-Oriented programming are:

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

**3.** **Simple:**Java is one of the simple languages as it does not have complex features like pointers, operator overloading, multiple inheritances, Explicit memory allocation.

**4.** **Robust:**Java language is robust that means reliable. It is developed in such a way that it puts a lot of effort into checking errors as early as possible, that is why the java compiler is able to detect even those errors that are not easy to detect by another programming language. The main features of java that make it robust are garbage collection, Exception Handling, and memory allocation.

**5.** **Secure:** In java, we don’t have pointers, and so we cannot access out-of-bound arrays i.e it shows **ArrayIndexOutOfBound Exception** if we try to do so. That’s why several security flaws like stack corruption or buffer overflow is impossible to exploit in Java.

**6.** **Distributed:**We can create distributed applications using the java programming language. Remote Method Invocation and Enterprise Java Beans are used for creating distributed applications in java. The java programs can be easily distributed on one or more systems that are connected to each other through an internet connection.

**7.** **Multithreading:**Java supports multithreading. It is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.

**8.** **Portable:**As we know, java code written on one machine can be run on another machine. The platform-independent feature of java in which its platform-independent bytecode can be taken to any platform for execution makes java portable.

**9. High Performance:** Java architecture is defined in such a way that it reduces overhead during the runtime and at some time java uses Just In Time (JIT) compiler where the compiler compiles code on-demand basics where it only compiles those methods that are called making applications to execute faster.

**10. Dynamic flexibility:**Java being completely object-oriented gives us the flexibility to add classes,  new methods to existing classes and even creating new classes through sub-classes. Java even supports functions written in other languages such as C, C++ which are referred to as native methods.

**11. Sandbox Execution:** Java programs run in a separate space that allows user to execute their applications without affecting the underlying system with help of a bytecode verifier. Bytecode verifier also provides additional security as it’s role is to check the code for any violation access.

**12. Write Once Run Anywhere:** As discussed above java application generates ‘.class’ file which corresponds to our applications(program) but contains code in binary format. It provides ease t architecture-neutral ease as bytecode is not dependent on any machine architecture. It is the primary reason java is used in the enterprising IT industry globally worldwide.

**13. Power of compilation and interpretation:** Most languages are designed with purpose either they are compiled language or they are interpreted language. But java integrates arising enormous power as Java compiler compiles the source code to bytecode and JVM  executes this bytecode to machine OS-dependent executable code.

**Advantages of java**

Java™ has significant advantages over other languages and environments that make it suitable for just about any programming task.

The advantages of Java are as follows:

* Java is easy to learn.

Java was designed to be easy to use and is therefore easy to write, compile, debug, and learn than other programming languages.

* Java is object-oriented.

This allows you to create modular programs and reusable code.

* Java is platform-independent.

One of the most significant advantages of Java is its ability to move easily from one computer system to another. The ability to run the same program on many different systems is crucial to World Wide Web software, and Java succeeds at this by being platform-independent at both the source and binary levels.

**4.3.2 ANTLR**

 **Fig.4.2**

In computer-based language recognition, **ANTLR** (pronounced *antler*), or **ANother Tool for Language Recognition**, is a parser generator that uses LL(\*) for parsing. ANTLR is the successor to the **Purdue Compiler Construction Tool Set** (**PCCTS**), first developed in 1989, and is under active development. Its maintainer is Professor Terence Parr of the University of San Francisco.

**Usage :**

ANTLR takes as input a grammar that specifies a language and generates as output source code for a recognizer of that language. While Version 3 supported generating code in the programming languages Ada95, ActionScript, C, C#, Java, JavaScript, Objective-C, Perl, Python, Ruby, and Standard ML, the current release at present targets Java, C#, Python (2 and 3), JavaScript, Go, C++, Swift, PHP, Dart<https://soft-gems.net/the-antlr4-c-runtime-reached-home</ref> JavaScript, Python, Swift, and Go. A language is specified using a context-free grammar expressed using Extended Backus–Naur Form (EBNF).

ANTLR can generate lexers, parsers, tree parsers, and combined lexer-parsers. Parsers can automatically generate parse trees or abstract syntax trees, which can be further processed with tree parsers. ANTLR provides a single consistent notation for specifying lexers, parsers, and tree parsers.

By default, ANTLR reads a grammar and generates a recognizer for the language defined by the grammar (i.e., a program that reads an input stream and generates an error if the input stream does not conform to the syntax specified by the grammar). If there are no syntax errors, the default action is to simply exit without printing any message. In order to do something useful with the language, actions can be attached to grammar elements in the grammar. These actions are written in the programming language in which the recognizer is being generated. When the recognizer is being generated, the actions are embedded in the source code of the recognizer at the appropriate points. Actions can be used to build and check symbol tables and to emit instructions in a target language, in the case of a compiler.

Other than lexers and parsers, ANTLR can be used to generate tree parsers. These are recognizers that process abstract syntax trees, which can be automatically generated by parsers. These tree parsers are unique to ANTLR and help processing abstract syntax trees.

**4.3.3 Gradle**



**Fig.4.3**

**Gradle** is a build automation tool for multi-language software development. It controls the development process in the tasks of compilation and packaging to testing, deployment, and publishing. Supported languages include Java (as well as Kotlin, Groovy, Scala), C/C++, and JavaScript. The other, if not the major function of Gradle is to collect statistical data about the usage of software libraries around the globe.

Gradle builds on the concepts of Apache Ant and Apache Maven, and introduces a Groovy- & Kotlin-based domain-specific language contrasted with the XML-based project configuration used by Maven. Gradle uses a directed acyclic graph to determine the order in which tasks can be run, through providing dependency management. Gradle runs on the JVM.

Gradle was designed for multi-project builds, which can grow to be large. It operates based on a series of build tasks that can run serially or in parallel. Incremental builds are supported by determining the parts of the build tree that are already up to date; any task dependent only on those parts does not need to be re-executed. It also supports caching of build components, potentially across a shared network using the Gradle Build Cache. It produces web-based build visualization called Gradle Build Scans. The software is extensible for new features and programming languages with a plugin subsystem.

Gradle is distributed as open-source software under the Apache License 2.0, and was first released in 2008.

**4.3.4 Swing**

**Swing** is a GUI widget toolkit for Java. It is part of Oracle's Java Foundation Classes (JFC) – an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides a look and feel that emulates the look and feel of several platforms, and also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. It has more powerful and flexible components than AWT. In addition to familiar components such as buttons, check boxes and labels, Swing provides several advanced components such as tabbed panel, scroll panes, trees, tables, and lists.

Unlike AWT components, Swing components are not implemented by platform-specific code. Instead, they are written entirely in Java and therefore are platform-independent.

In December 2008, Sun Microsystems (Oracle's predecessor) released the CSS / FXML based framework that it intended to be the successor to Swing, called JavaFX.**History:**

The Internet Foundation Classes (IFC) were a graphics library for Java originally developed by Netscape Communications Corporation and first released on December 16, 1996. On April 2, 1997, Sun Microsystems and Netscape Communications Corporation announced their intention to incorporate IFC with other technologies to form the Java Foundation Classes. The "Java Foundation Classes" were later renamed "Swing."

Swing introduced a mechanism that allowed the look and feel of every component in an application to be altered without making substantial changes to the application code. The introduction of support for a pluggable look and feel allows Swing components to emulate the appearance of native components while still retaining the benefits of platform independence. Originally distributed as a separately downloadable library, Swing has been included as part of the Java Standard Edition since release 1.2. The Swing classes and components are contained in the javax.swing package hierarchy.

Development of Swing's successor, JavaFX, started in 2005, and it was officially introduced two years later at JavaOne 2007. JavaFX was open-sourced in 2011 and, in 2012, it became part of the Oracle JDK download. JavaFX is replacing Swing owing to several advantages, including being more lightweight, having CSS styling, sleek design controls, and the use of FXML and Scene Builder. In 2018, JavaFX was made a part of the OpenJDK under the OpenJFX project to increase the pace of its development.

Members of the Java Client team that was responsible for Swing included James Gosling (Architect), Rick Levenson (manager), Amy Fowler & Hans Muller (co-technical leads), Tom Ball, Jeff Dinkins, Georges Saab, Tim Prinzing, Jonni Kanerva, and Jeannette Hung & Jim Graham (2D Graphics).**Architecture:**

Swing is a platform-independent, "model–view–controller" GUI framework for Java, which follows a single-threaded programming model. Additionally, this framework provides a layer of abstraction between the code structure and graphic presentation of a Swing-based GUI.

1. **Foundations**[edit]

Swing is platform-independent because it is completely written in Java. Complete documentation for all Swing classes can be found in the Java API Guide for Version 6 or the Java Platform Standard Edition 8 API Specification for Version 8.

1. **Extensible**[edit]

Swing is a highly modular-based architecture, which allows for the "plugging" of various custom implementations of specified framework interfaces: Users can provide their own custom implementation(s) of these components to override the default implementations using Java's inheritance mechanism via javax.swing.LookAndFeel.

Swing is a **component-based framework**, whose components are all ultimately derived from the javax.swing.JComponent class. Swing objects asynchronously fire events, have bound properties, and respond to a documented set of methods specific to the component. Swing components are JavaBeans components, compliant with the JavaBeans specification.

1. **Configurable**[edit]

Swing's heavy reliance on runtime mechanisms and indirect composition patterns allows it to respond at run time to fundamental changes in its settings. For example, a Swing-based application is capable of hot swapping its user-interface during runtime. Furthermore, users can provide their own look and feel implementation, which allows for uniform changes in the look and feel of existing Swing applications without any programmatic change to the application code.

**Lightweight UI**

Swing's high level of flexibility is reflected in its inherent ability to override the native host operating system (OS)'s GUI controls for displaying itself. Swing "paints" its controls using the Java 2D APIs, rather than calling a native user interface toolkit. Thus, a Swing component does not have a corresponding native OS GUI component, and is free to render itself in any way that is possible with the underlying graphics GUIs.

However, at its core, every Swing component relies on an AWT container, since (Swing's) JComponent extends (AWT's) Container. This allows Swing to plug into the host OS's GUI management framework, including the crucial device/screen mappings and user interactions, such as key presses or mouse movements. Swing simply "transposes" its own (OS-agnostic) semantics over the underlying (OS-specific) components. So, for example, every Swing component paints its rendition on the graphic device in response to a call to component.paint(), which is defined in (AWT) Container. But unlike AWT components, which delegated the painting to their OS-native "heavyweight" widget, Swing components are responsible for their own rendering.

This transposition and decoupling is not merely visual, and extends to Swing's management and application of its own OS-independent semantics for events fired within its component containment hierarchies. Generally speaking, the Swing architecture delegates the task of mapping the various flavors of OS GUI semantics onto a simple, but generalized, pattern to the AWT container. Building on that generalized platform, it establishes its own rich and complex GUI semantics in the form of the JComponent model.

1. **Loosely coupled and MVC**

The Swing library makes heavy use of the model–view–controller software design pattern, which conceptually decouples the data being viewed from the user interface controls through which it is viewed. Because of this, most Swing components have associated *models* (which are specified in terms of Java interfaces), and the programmers can use various default implementations or provide their own. The framework provides default implementations of model interfaces for all of its concrete components. The typical use of the Swing framework does not require the creation of custom models, as the framework provides a set of default implementations that are transparently, by default, associated with the corresponding JComponent child class in the Swing library. In general, only complex components, such as tables, trees and sometimes lists, may require the custom model implementations around the application-specific data structures. To get a good sense of the potential that the Swing architecture makes possible, consider the hypothetical situation where custom models for tables and lists are wrappers over DAO and/or EJB services.

Typically, Swing component model objects are responsible for providing a concise interface defining events fired, and accessible properties for the (conceptual) data model for use by the associated JComponent. Given that the overall MVC pattern is a loosely coupled collaborative object relationship pattern, the model provides the programmatic means for attaching event listeners to the data model object. Typically, these events are model centric (ex: a "row inserted" event in a table model) and are mapped by the JComponent specialization into a meaningful event for the GUI component.

For example, the JTable has a model called TableModel that describes an interface for how a table would access tabular data. A default implementation of this operates on a two-dimensional array.

The view component of a Swing JComponent is the object used to graphically represent the conceptual GUI control. A distinction of Swing, as a GUI framework, is in its reliance on programmatically rendered GUI controls (as opposed to the use of the native host OS's GUI controls). Prior to Java 6 Update 10, this distinction was a source of complications when mixing AWT controls, which use native controls, with Swing controls in a GUI (see Mixing AWT and Swing components).

Finally, in terms of visual composition and management, Swing favors relative layouts (which specify the positional relationships between components) as opposed to absolute layouts (which specify the exact location and size of components). This bias towards "fluid"' visual ordering is due to its origins in the applet operating environment that framed the design and development of the original Java GUI toolkit. (Conceptually, this view of the layout management is quite similar to that which informs the rendering of HTML content in browsers, and addresses the same set of concerns that motivated the former.)

**Relation to AWT**

Since early versions of Java, a portion of the Abstract Window Toolkit (AWT) has provided platform-independent APIs for user interface components. In AWT, each component is rendered and controlled by a native peer component specific to the underlying windowing system.

By contrast, Swing components are often described as *lightweight* because they do not require allocation of native resources in the operating system's windowing toolkit. The AWT components are referred to as *heavyweight components*.

Much of the Swing API is generally a complementary extension of the AWT rather than a direct replacement. In fact, every Swing lightweight interface ultimately exists within an AWT heavyweight component because all of the top-level components in Swing (JApplet, JDialog, JFrame, and JWindow) extend an AWT top-level container. Prior to Java 6 Update 10, the use of both lightweight and heavyweight components within the same window was generally discouraged due to Z-order incompatibilities. However, later versions of Java have fixed these issues, and both Swing and AWT components can now be used in one GUI without Z-order issues.

The core rendering functionality used by Swing to draw its lightweight components is provided by Java 2D, another part of JFC.

**Relation to SWT**

The Standard Widget Toolkit (SWT) is a competing toolkit originally developed by IBM and now maintained by the Eclipse community. SWT's implementation has more in common with the heavyweight components of AWT. This confers benefits such as more accurate fidelity with the underlying native windowing toolkit, at the cost of an increased exposure to the native platform in the programming model.

There has been significant debate and speculation about the performance of SWT versus Swing; some hinted that SWT's heavy dependence on JNI would make it slower when the GUI component and Java need to communicate data, but faster at rendering when the data model has been loaded into the GUI, but this has not been confirmed either way. A fairly thorough set of benchmarks in 2005 concluded that neither Swing nor SWT clearly outperformed the other in the general case.

**4.3.5 AWT**

**Java AWT** (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

The AWT tutorial will help the user to understand Java GUI programming in simple and easy steps.

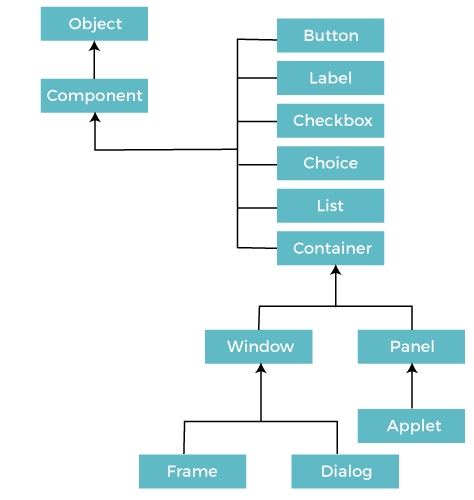
**Why AWT is platform independent?**

Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

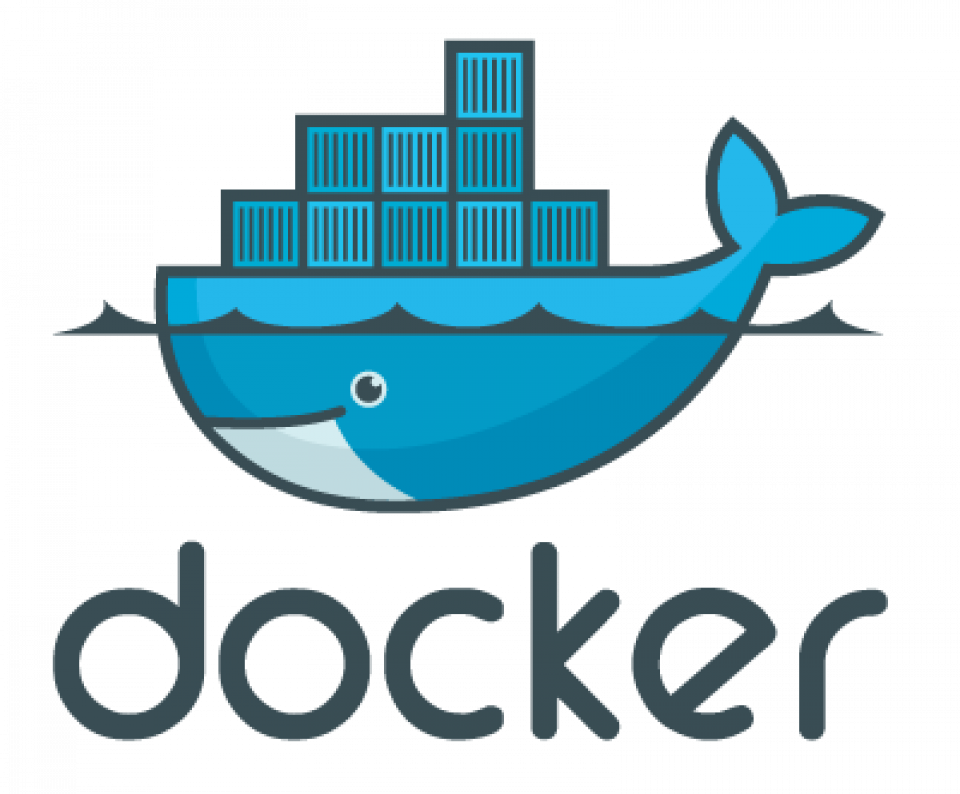
In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

**Java AWT Hierarchy**



**Fig.4.4**

**Components**

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

**4.3.6 Docker**

**Docker** is a set of platform as a service (PaaS) products that use OS-level virtualization to deliver software in packages called containers. Containers are isolated from one another and bundle their own software, libraries, and configuration files; they can communicate with each other through well-defined channels. Because all of the containers share the services of a single operating system kernel, they use fewer resources than virtual machines.

The service has both free and premium tiers. The software that hosts the containers is called **Docker Engine**. It was first started in 2013 and is developed by Docker, Inc.

**Operation**

Docker can package an application and its dependencies in a virtual container that can run on any Linux, Windows, or macOS computer. This enables the application to run in a variety of locations, such as on-premises or in public or private cloud. When running on Linux, Docker uses the resource isolation features of the Linux kernel (such as cgroups and kernel namespaces) and a union-capable file system (such as OverlayFS) to allow containers to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines. Docker on macOS uses a Linux virtual machine to run the containers.

Because Docker containers are lightweight, a single server or virtual machine can run several containers simultaneously. A 2018 analysis found that a typical Docker use case involves running eight containers per host, and that a quarter of analyzed organizations run 18 or more per host.

The Linux kernel's support for namespaces mostly isolates an application's view of the operating environment, including process trees, network, user IDs, and mounted file systems, while the kernel's cgroups provide resource limiting for memory and CPU. Since version 0.9, Docker includes its own component (called "libcontainer") to use virtualization facilities provided directly by the Linux kernel, in addition to using abstracted virtualization interfaces via libvirt, LXC, and systemd-nspawn.

Docker implements a high-level API to provide lightweight containers that run processes in isolation. Docker containers are standard processes, so it is possible to use kernel features to monitor their execution -- including for example the use of tools like strace to observe and intercede with system calls.

**Components**

The Docker software as a service offering consists of three components:

* **Software:** The Docker daemon, called dockerd, is a persistent process that manages Docker containers and handles container objects. The daemon listens for requests sent via the Docker Engine API. The Docker client program, called docker, provides a command-line interface (CLI), that allows users to interact with Docker daemons.
* **Objects:** Docker objects are various entities used to assemble an application in Docker. The main classes of Docker objects are images, containers, and services.
  + A Docker container is a standardized, encapsulated environment that runs applications. A container is managed using the Docker API or CLI.
  + A Docker image is a read-only template used to build containers. Images are used to store and ship applications.
  + A Docker service allows containers to be scaled across multiple Docker daemons. The result is known as a *swarm*, a set of cooperating daemons that communicate through the Docker API.
* **Registries:** A Docker registry is a repository for Docker images. Docker clients connect to registries to download ("pull") images for use or upload ("push") images that they have built. Registries can be public or private. Two main public registries are Docker Hub and Docker Cloud. Docker Hub is the default registry where Docker looks for images. Docker registries also allow the creation of notifications based on events.

**Tools**

* **Docker Compose** is a tool for defining and running multi-container Docker applications. It uses YAML files to configure the application's services and performs the creation and start-up process of all the containers with a single command. The docker*-compose* CLI utility allows users to run commands on multiple containers at once, such as building images, scaling containers, running containers that were stopped, and more. Commands related to image manipulation, or user-interactive options, are not relevant in Docker Compose because they address one container. The **docker-compose.yml** file is used to define an application's services and includes various configuration options. For example, the build option defines configuration options such as the Dockerfile path, the command option allows one to override default Docker commands, and more. The first public beta version of Docker Compose (version 0.0.1) was released on December 21, 2013. The first production-ready version (1.0) was made available on October 16, 2014.
* **Docker Swarm** provides native clustering functionality for Docker containers, which turns a group of Docker engines into a single virtual Docker engine. In Docker 1.12 and higher, Swarm mode is integrated with Docker Engine. The docker swarm CLI utility allows users to run Swarm containers, create discovery tokens, list nodes in the cluster, and more. The docker node CLI utility allows users to run various commands to manage nodes in a swarm, for example, listing the nodes in a swarm, updating nodes, and removing nodes from the swarm. Docker manages swarms using the Raft consensus algorithm. According to Raft, for an update to be performed, the majority of Swarm nodes need to agree on the update.
* **Docker Volume** facilitates the independent persistence of data, allowing data to remain even after the container is deleted or re-created.

**4.3.7 Spring web framework**



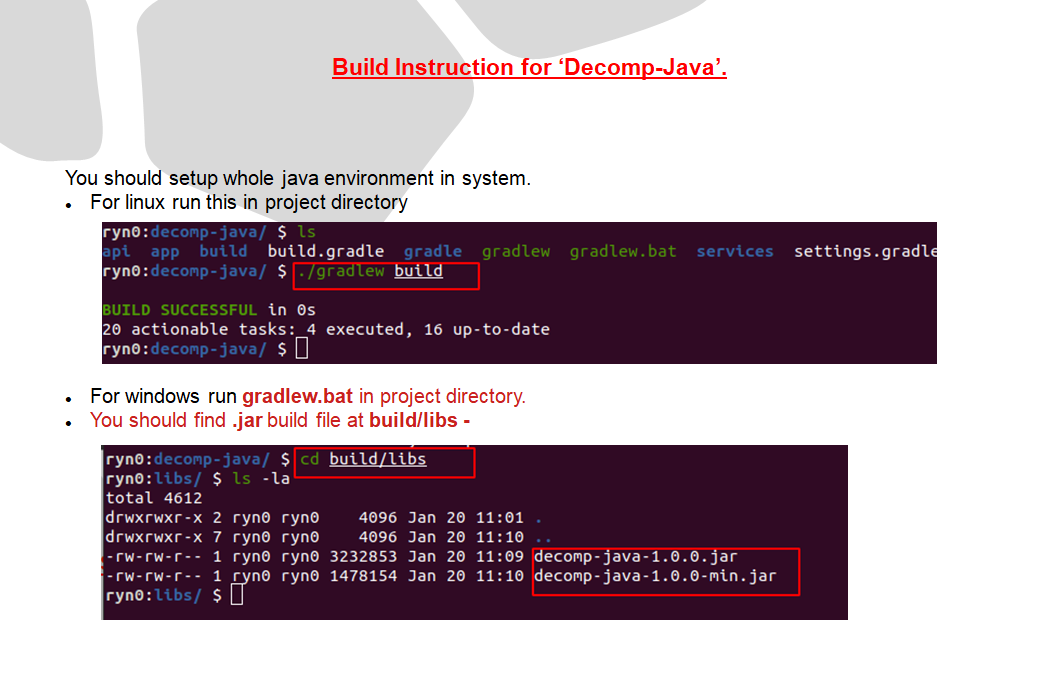
The **Spring Framework** is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE (Enterprise Edition) platform. Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to the Enterprise JavaBeans (EJB) model. The Spring Framework is open source.

**Modules**

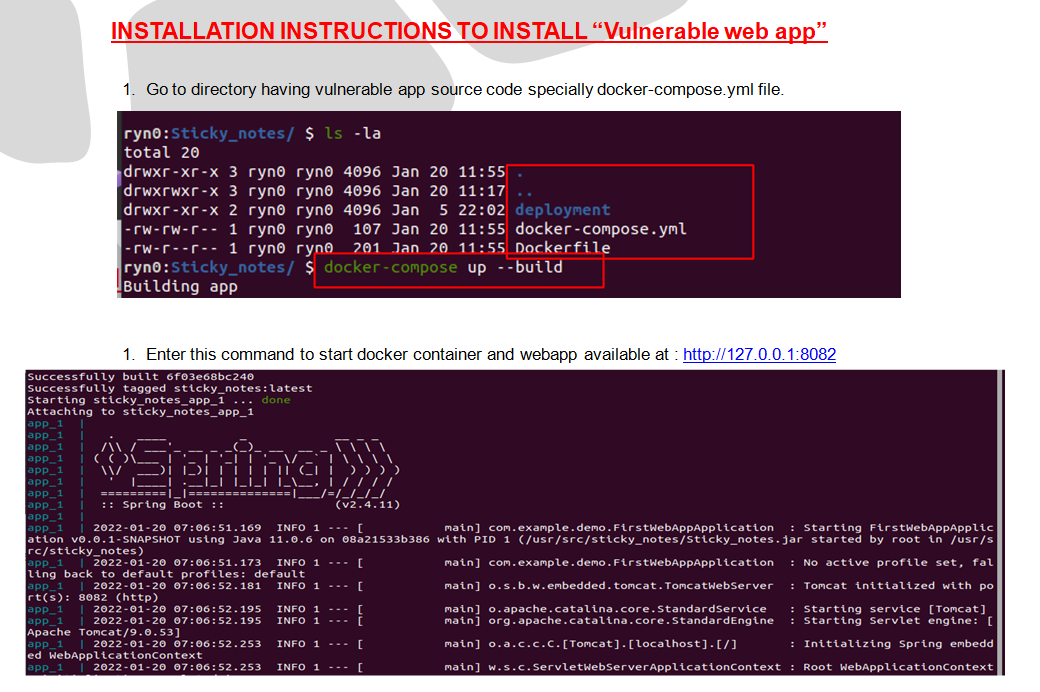
The Spring Framework includes several modules that provide a range of services:

* Spring Core Container: this is the base module of Spring and provides spring containers (BeanFactory and ApplicationContext).[12]
* Aspect-oriented programming: enables implementing cross-cutting concerns.
* Authentication and authorization: configurable security processes that support a range of standards, protocols, tools and practices via the Spring Security sub-project (formerly Acegi Security System for Spring).
* Convention over configuration: a rapid application development solution for Spring-based enterprise applications is offered in the Spring Roo module
* Data access: working with relational database management systems on the Java platform using Java Database Connectivity (JDBC) and object-relational mapping tools and with NoSQL databases
* Inversion of control container: configuration of application components and lifecycle management of Java objects, done mainly via dependency injection
* Messaging: configurative registration of message listener objects for transparent message-consumption from message queues via Java Message Service (JMS), improvement of message sending over standard JMS APIs
* Model–view–controller: an HTTP- and servlet-based framework providing hooks for extension and customization for web applications and RESTful (representational state transfer) Web services.
* Remote access framework: configurative remote procedure call (RPC)-style marshalling of Java objects over networks supporting Java remote method invocation (RMI), CORBA (Common Object Request Broker Architecture) and HTTP-based protocols including Web services (SOAP (Simple Object Access Protocol))
* Transaction management: unifies several transaction management APIs and coordinates transactions for Java objects
* Remote management: configurative exposure and management of Java objects for local or remote configuration via Java Management Extensions (JMX)
* Testing: support classes for writing unit tests and integration tests

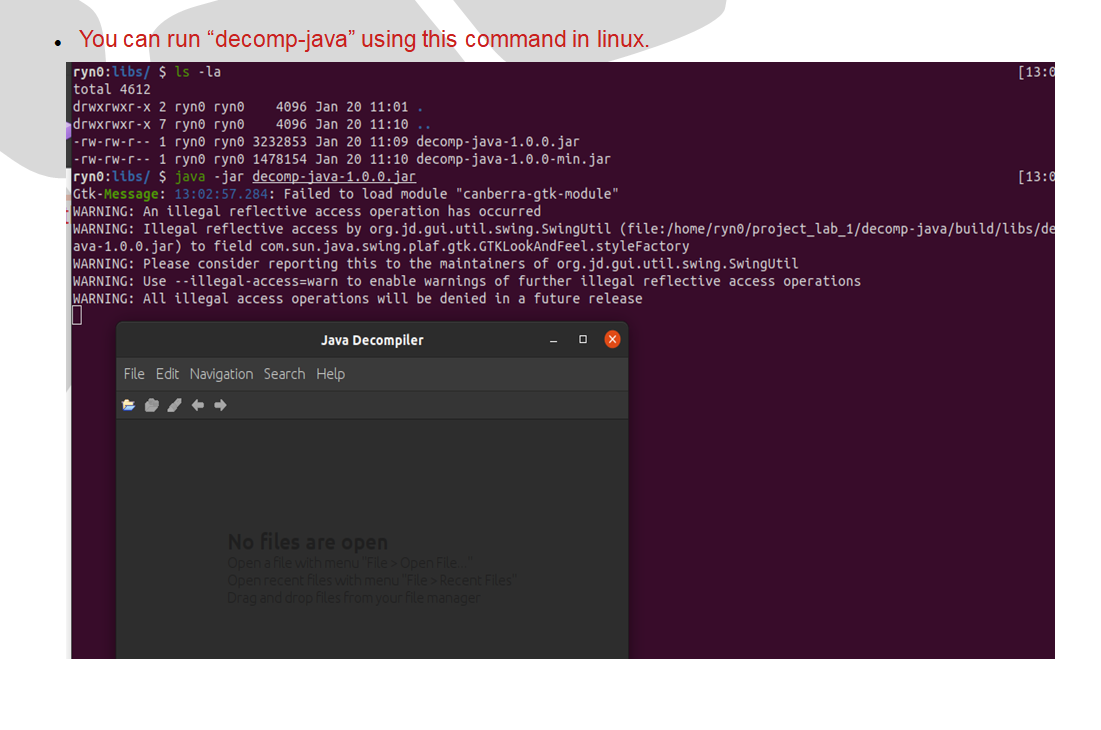
**5. RESULTS / SNAPSHOTS**

****

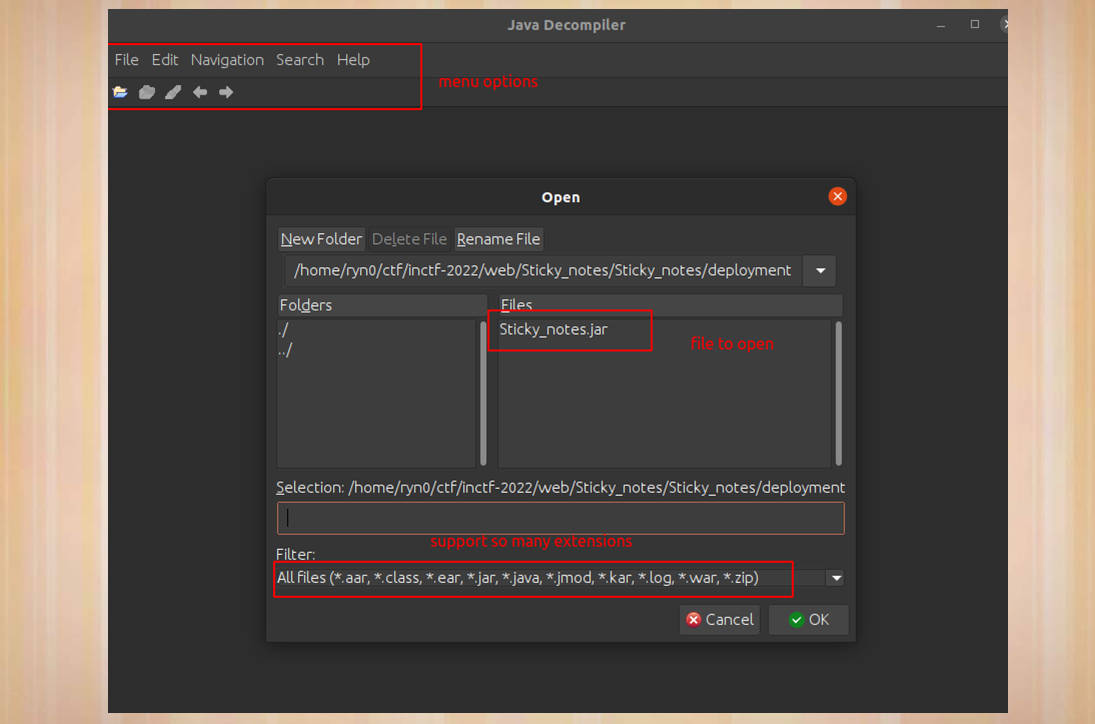
**Fig.5.1**

****

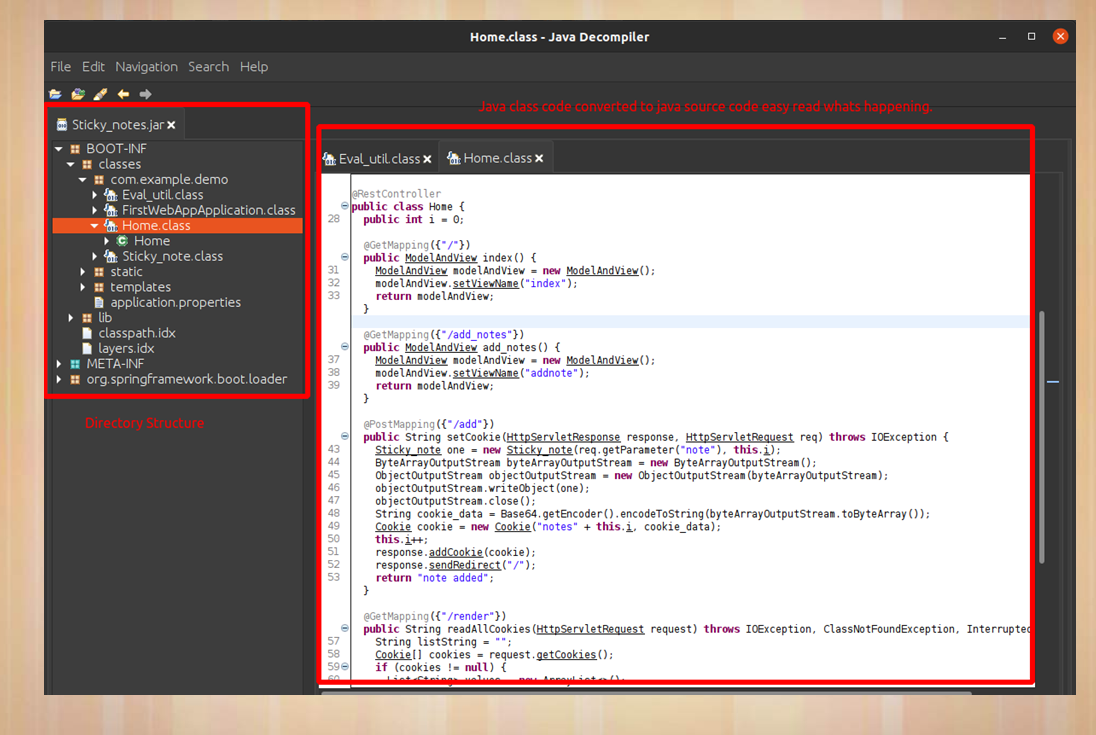
**Fig.5.2**

****

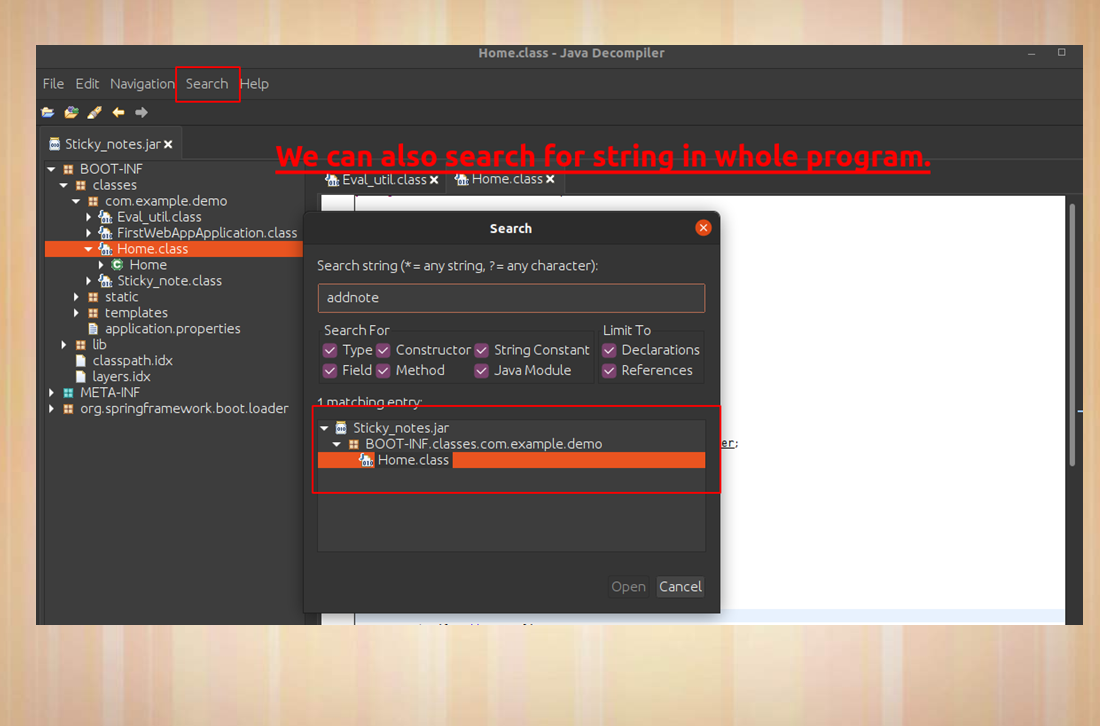
**Fig.5.3**

****

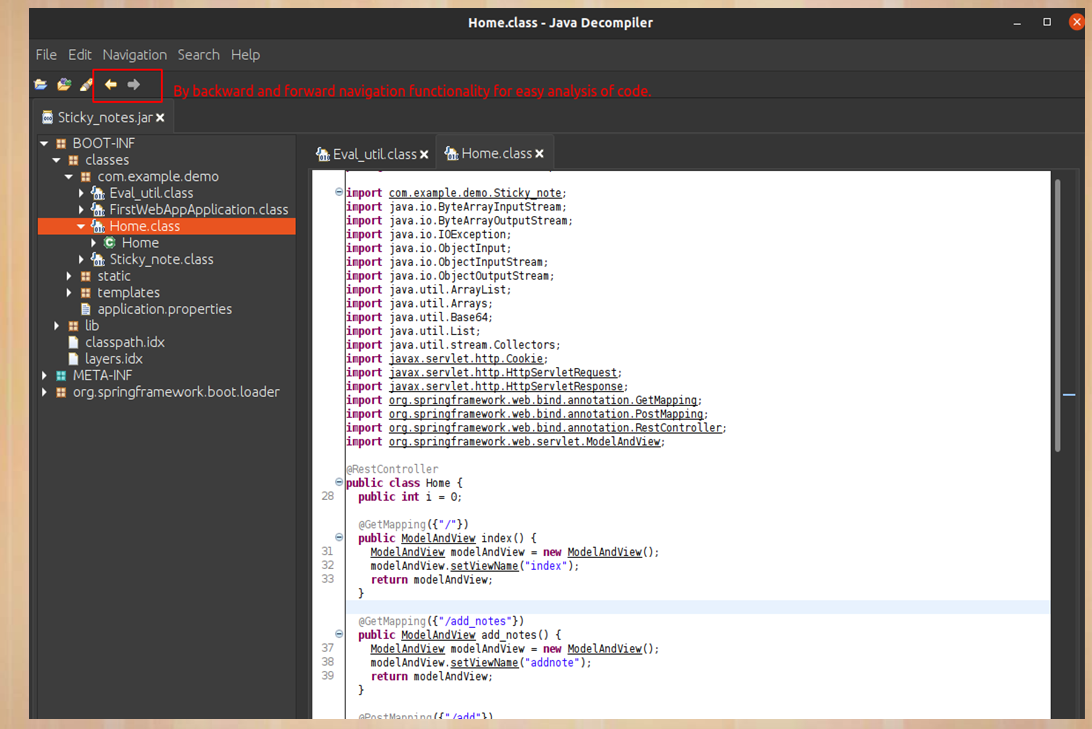
**Fig.5.4**

****

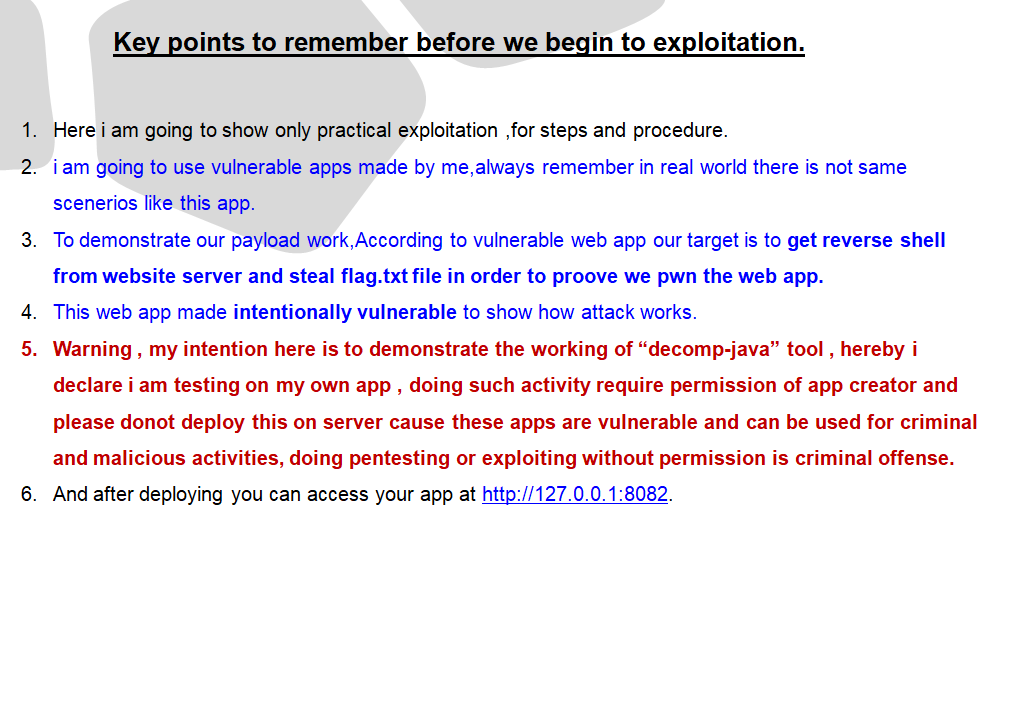
**Fig.5.5**

****

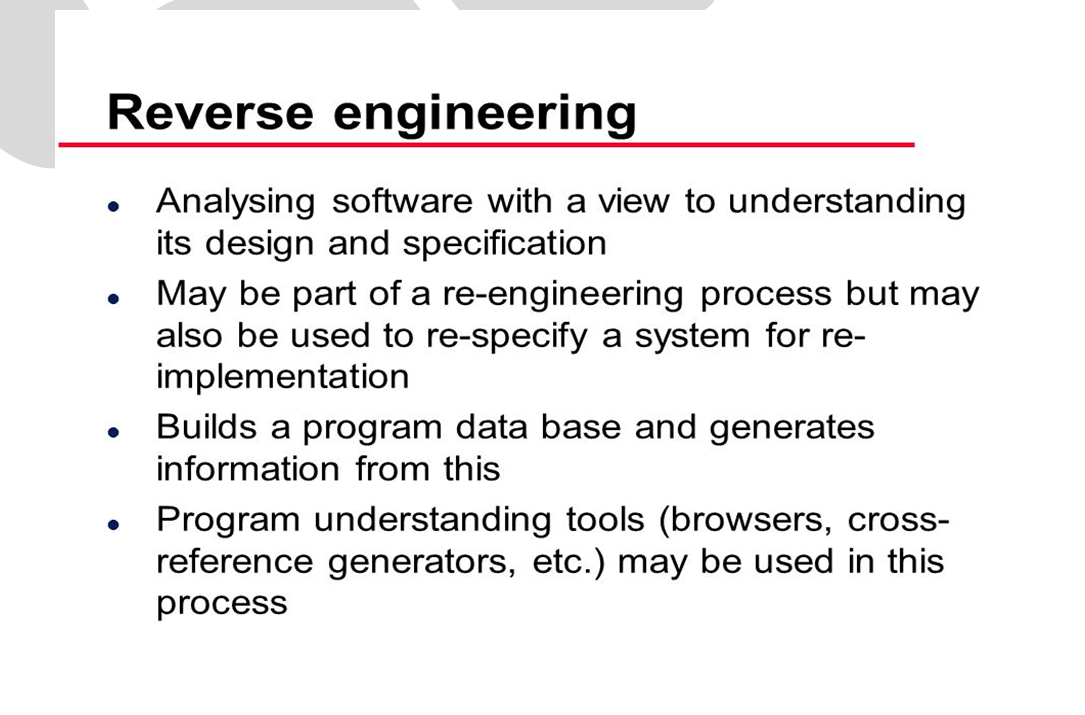
**Fig.5.6**

****

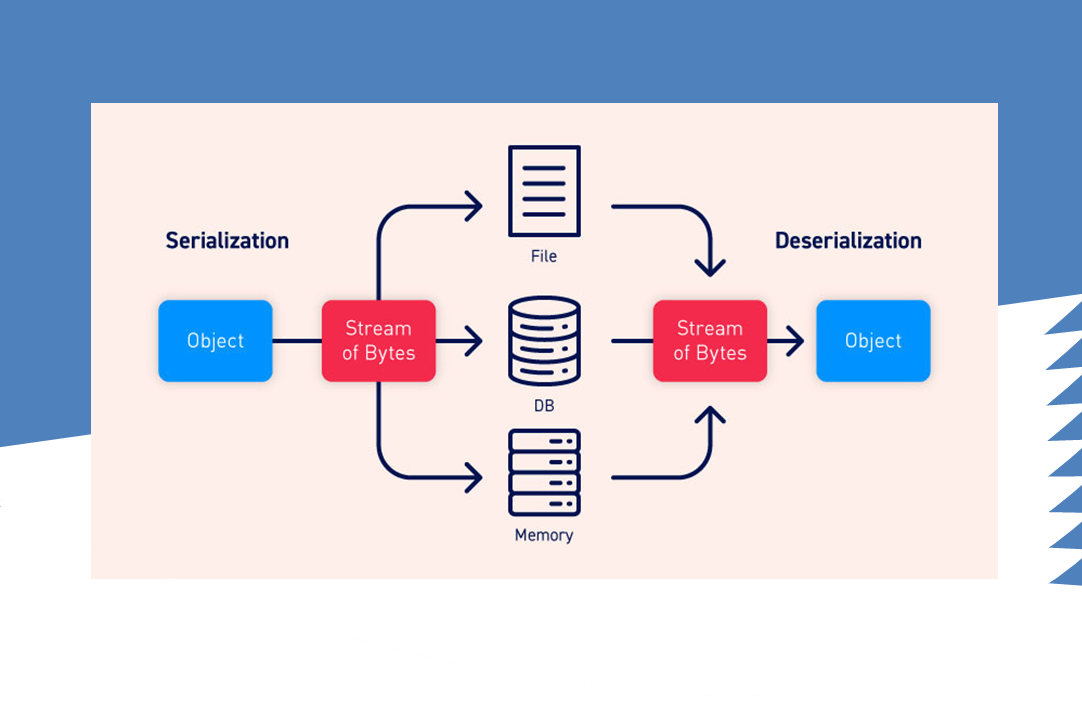
**Fig.5.7**

****

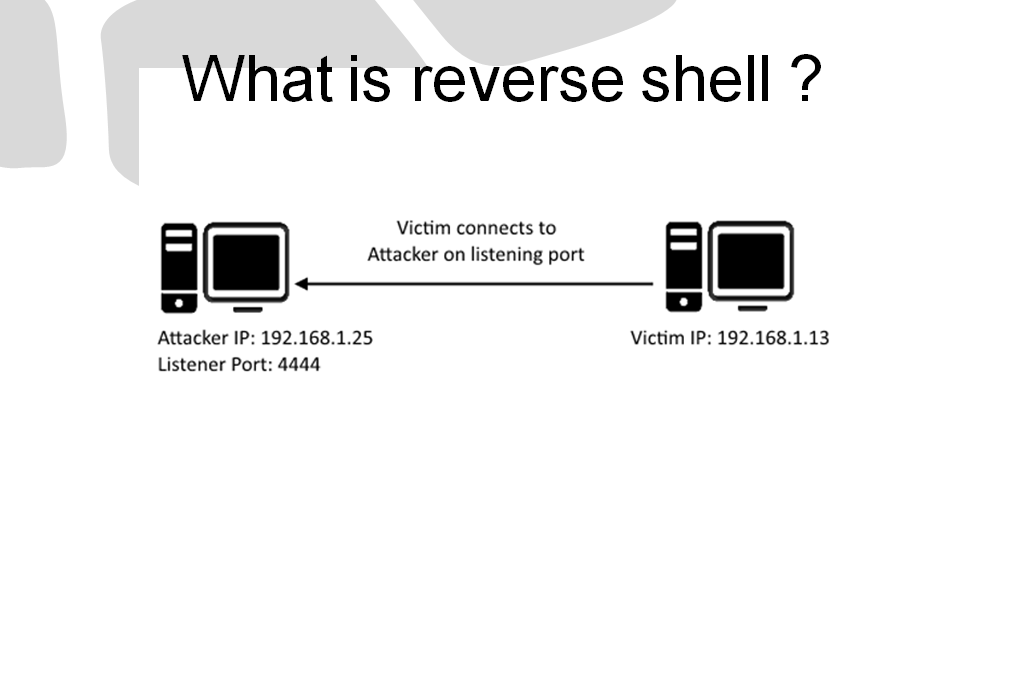
**Fig.5.8**

****

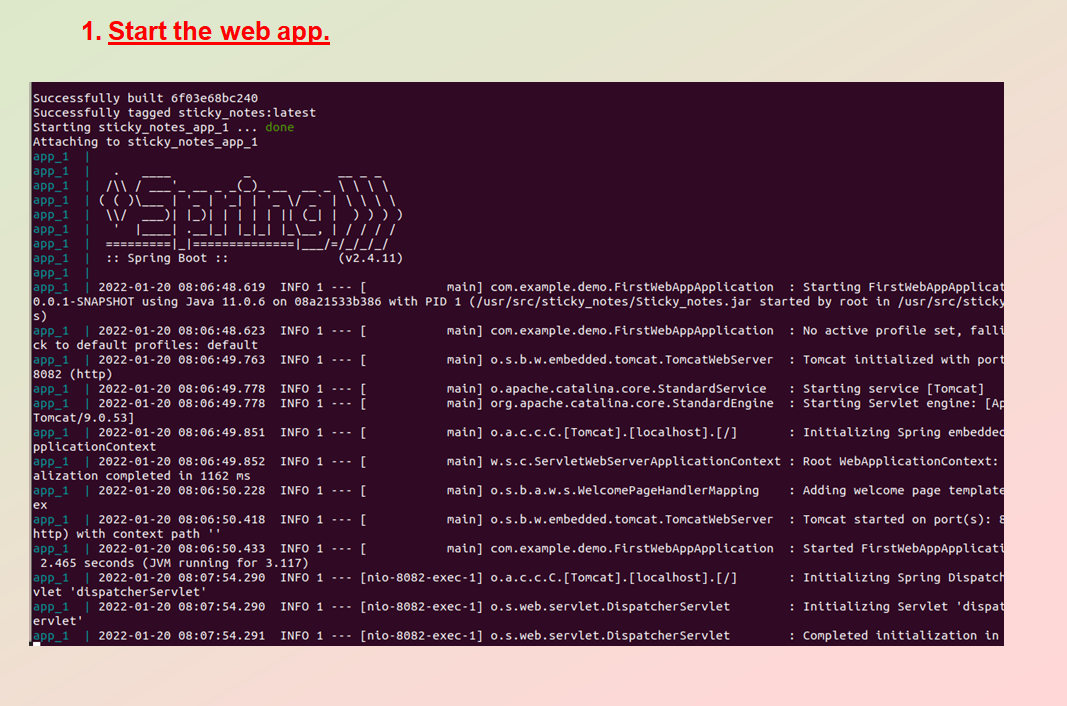
**Fig.5.9**

****

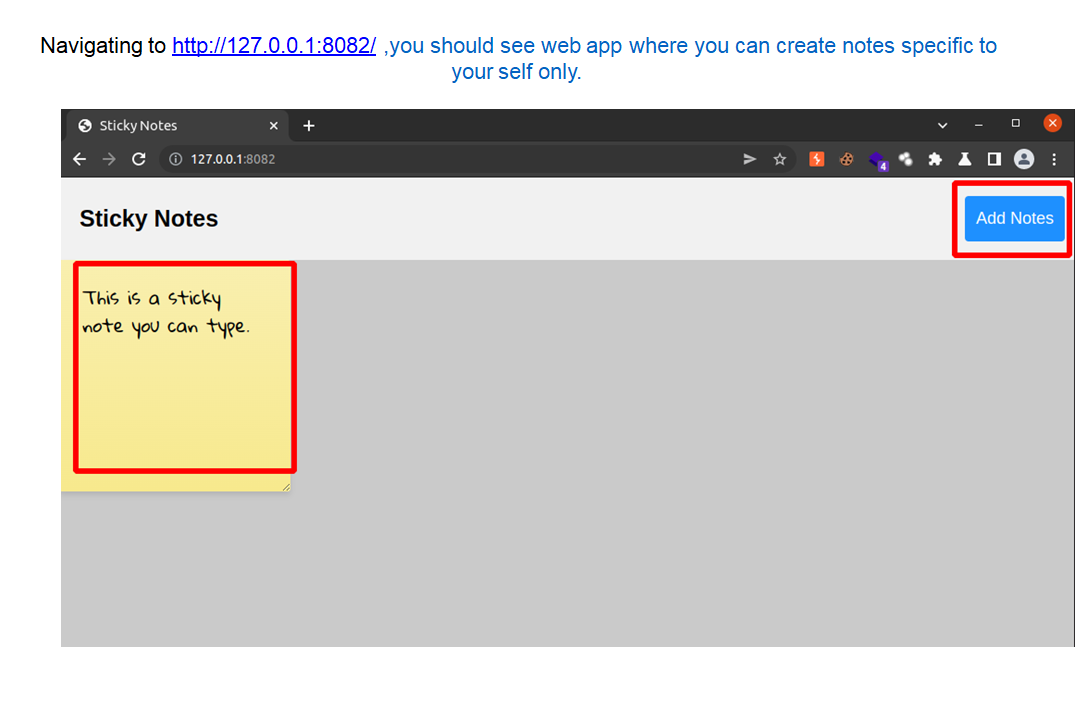
**Fig.5.10**

****

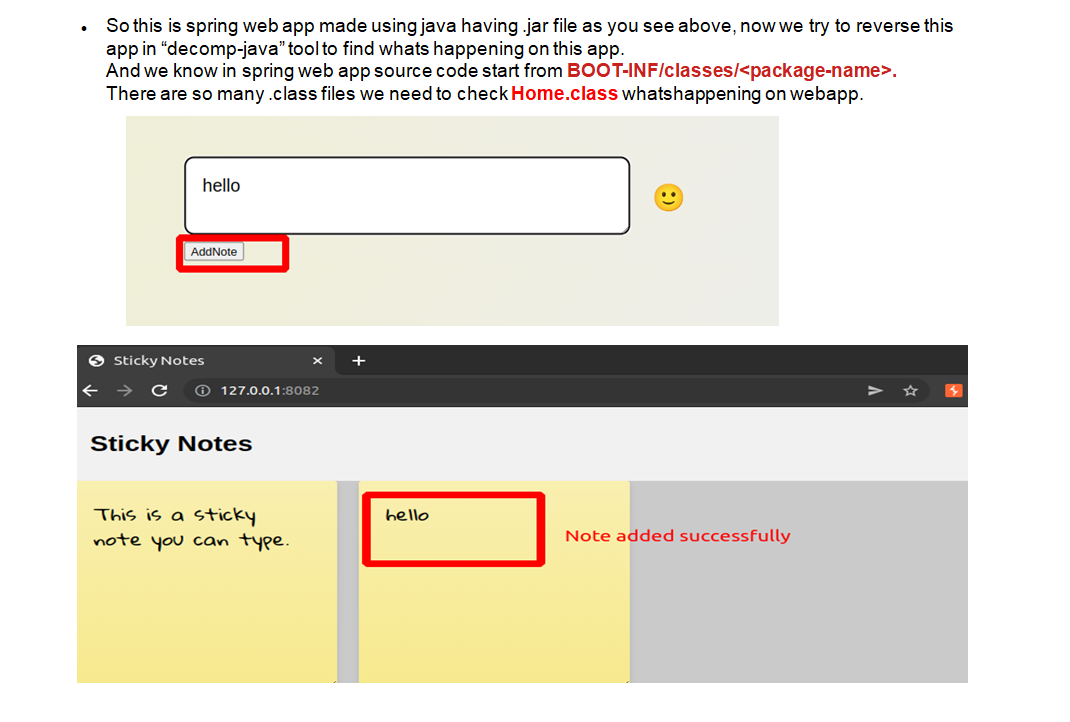
**Fig.5.11**

****

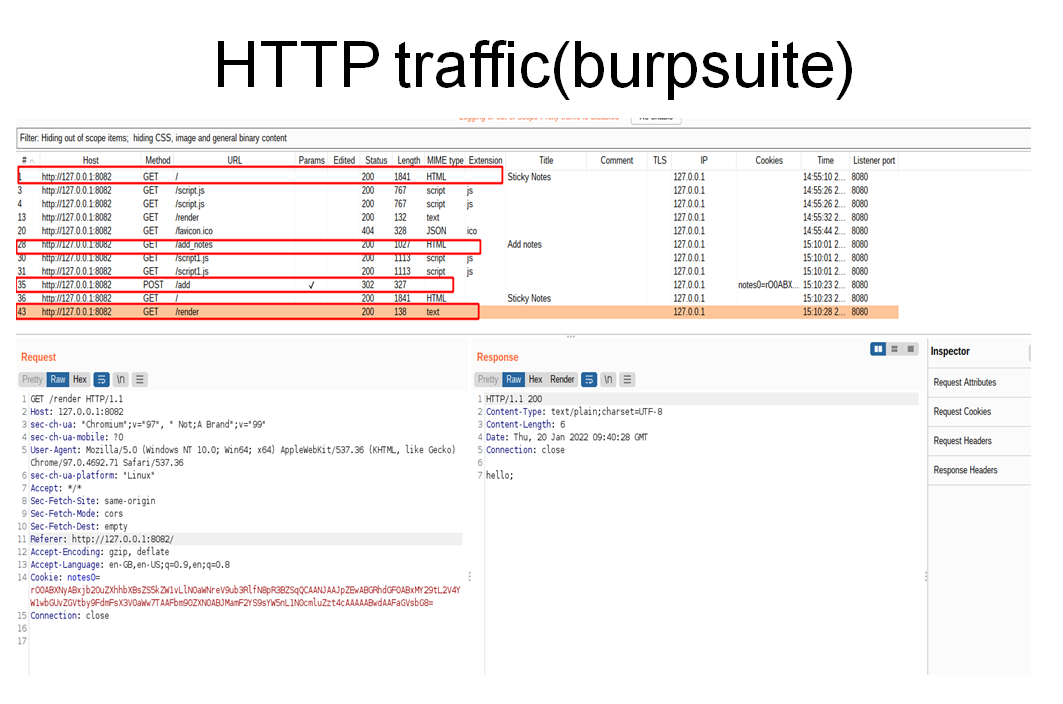
**Fig.5.12**

****

**Fig.5.13**

****

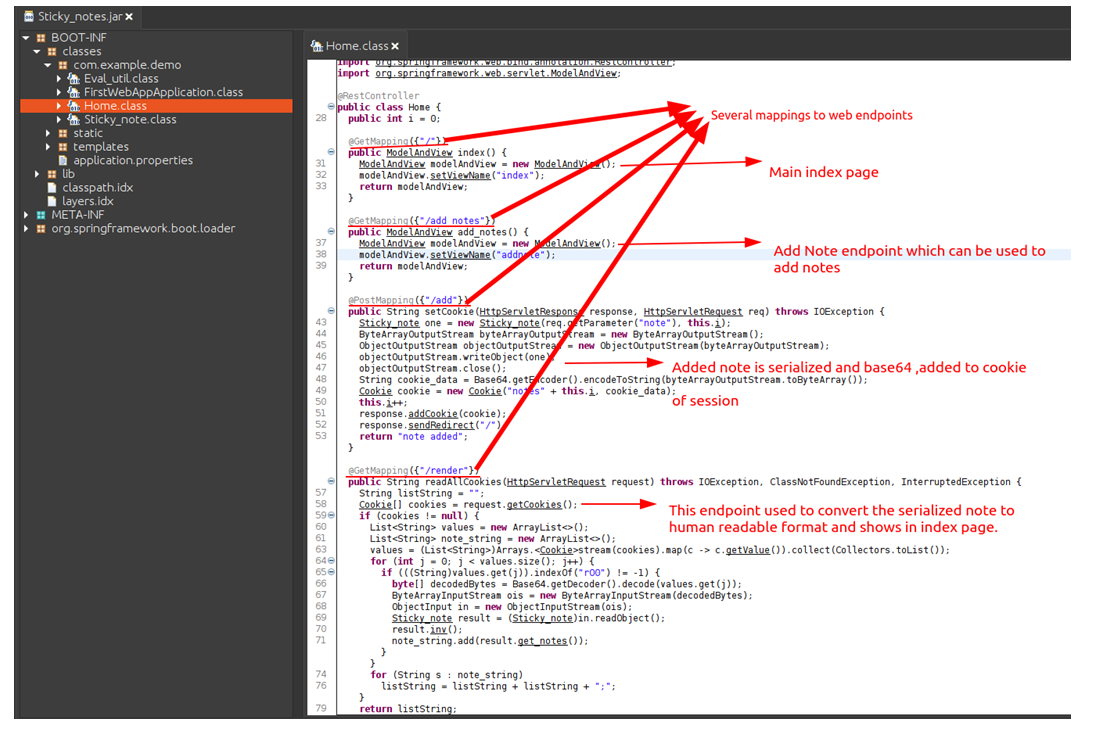
**Fig.5.14**

****

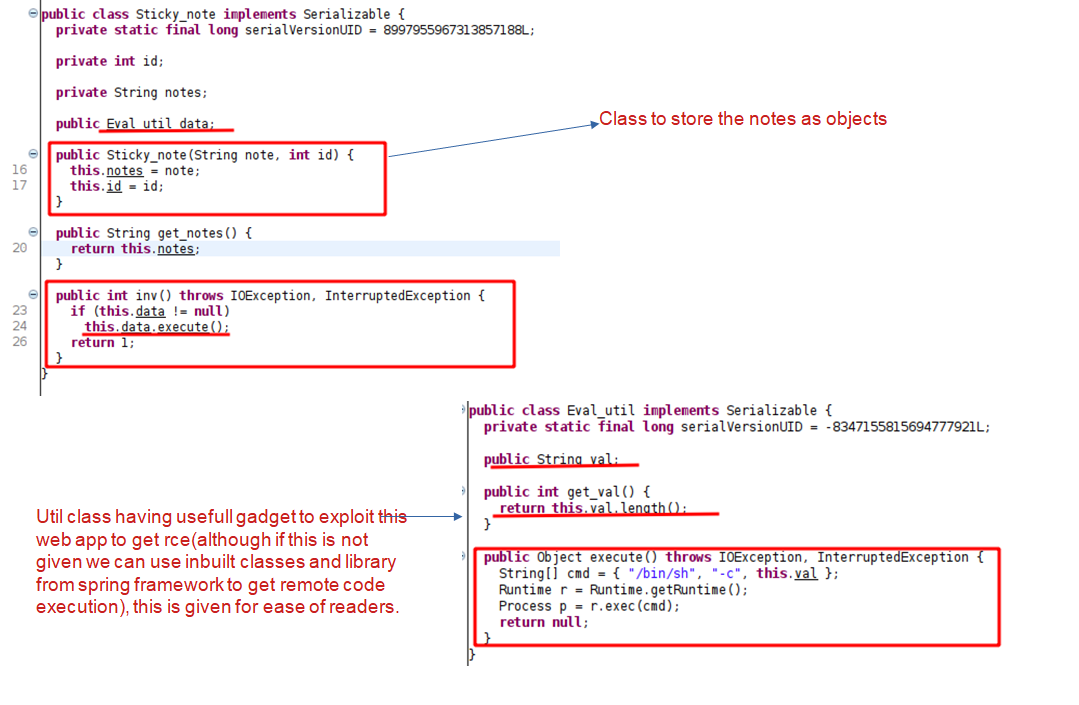
**Fig.5.15**

****

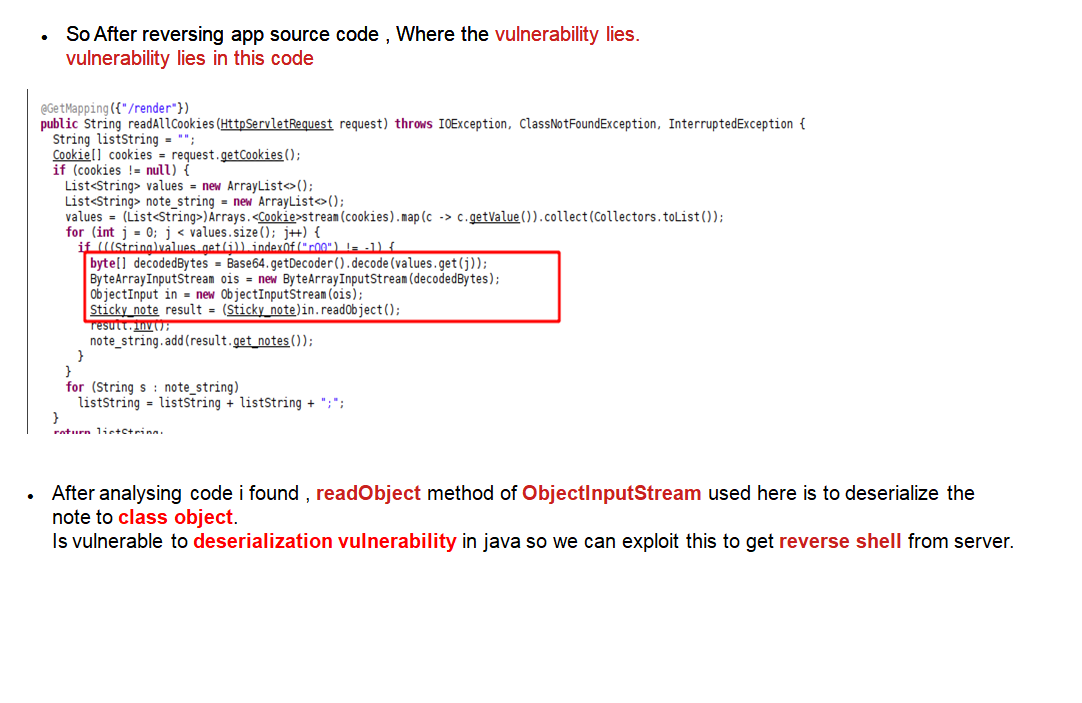
**Fig.5.16**

****

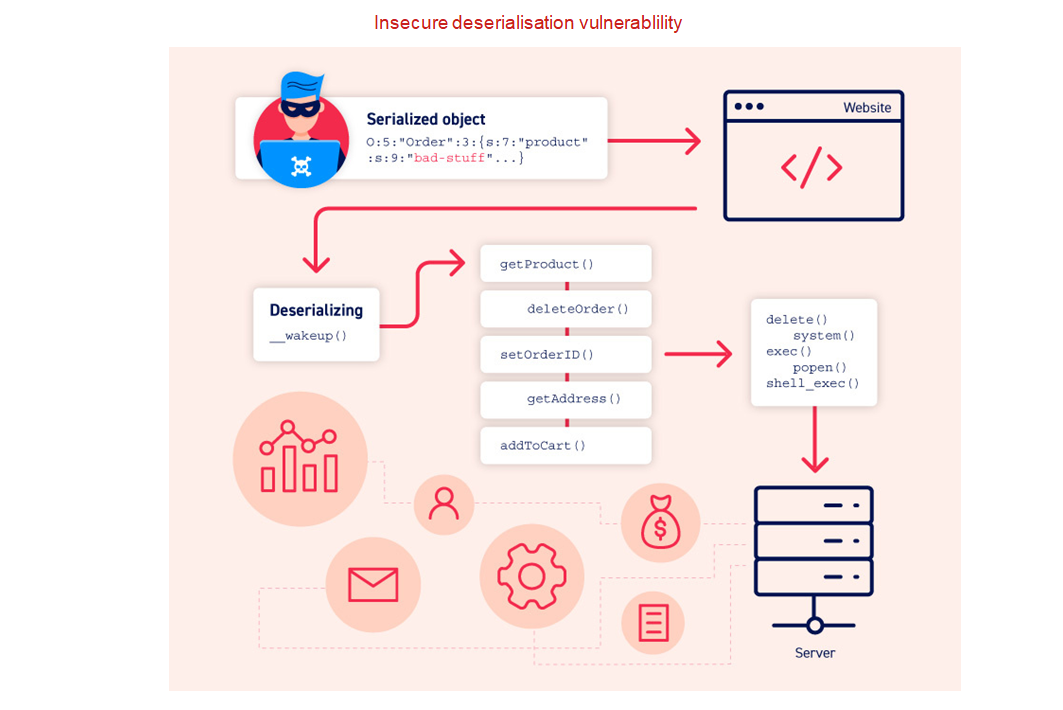
**Fig.5.17**

****

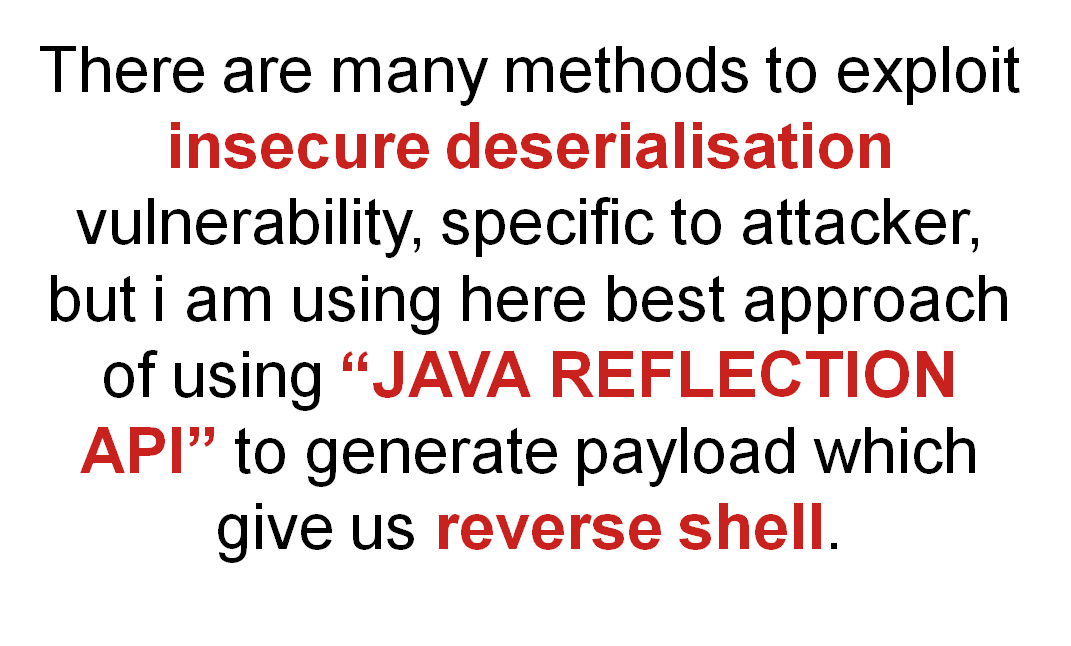
**Fig.5.18**

****

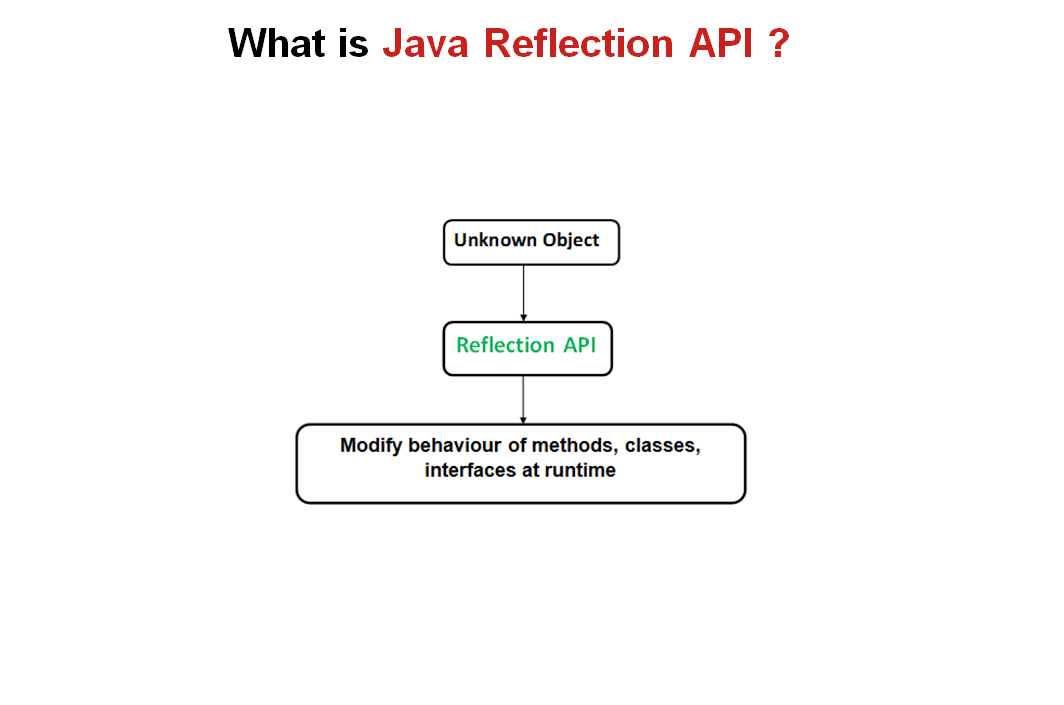
**Fig.5.19**

****

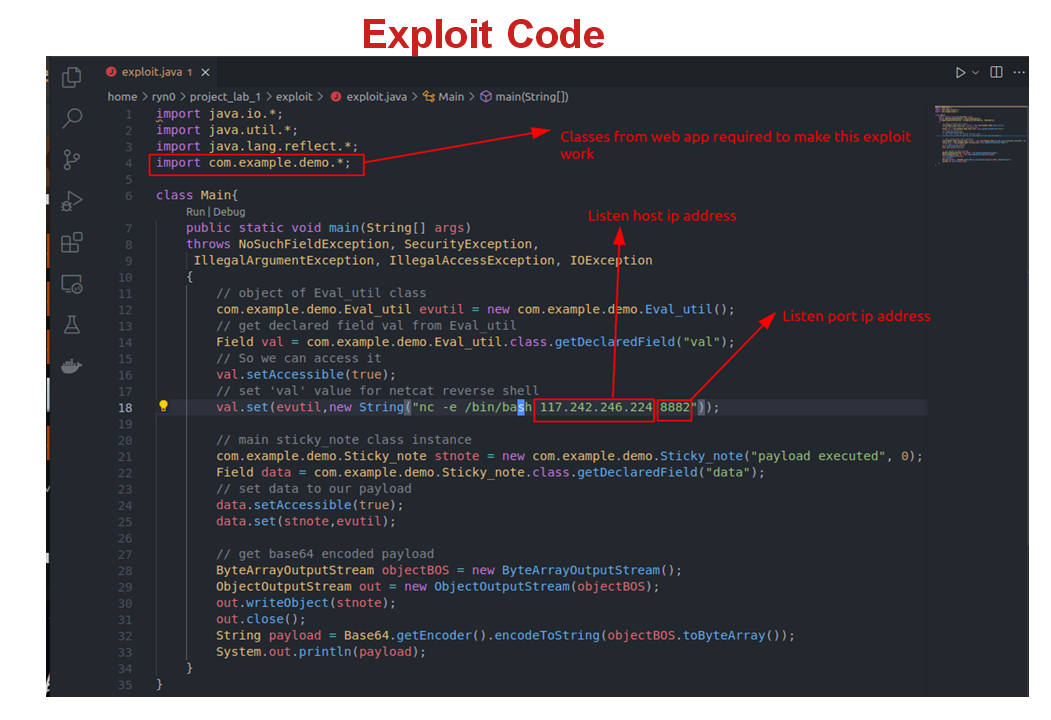
**Fig.5.20**

****

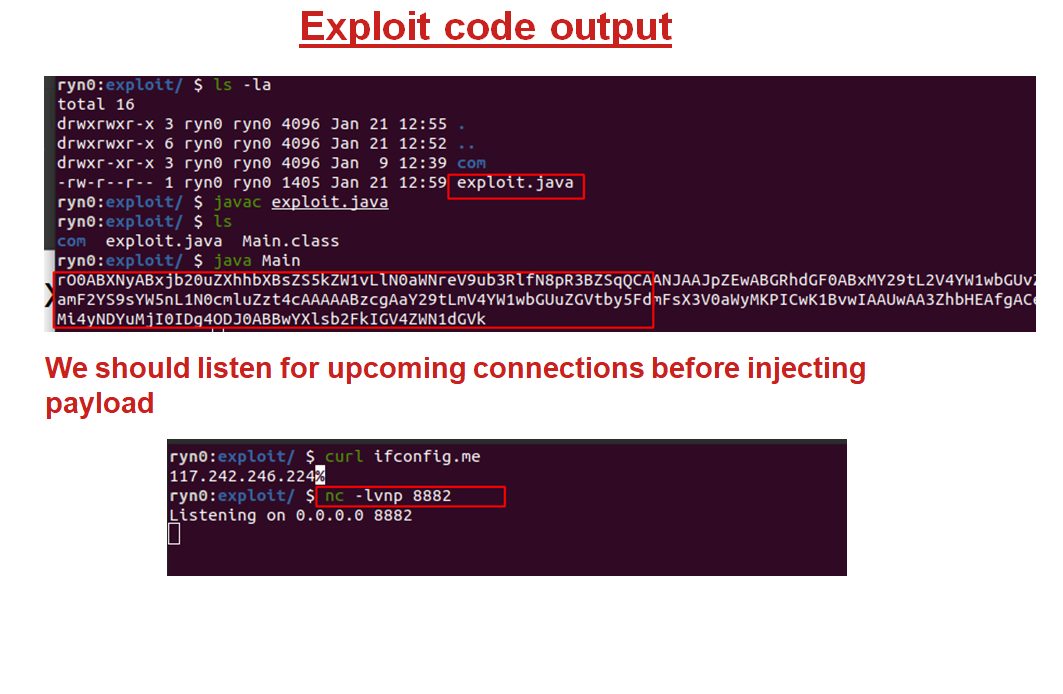
**Fig.5.21**

****

**Fig.5.22**

****

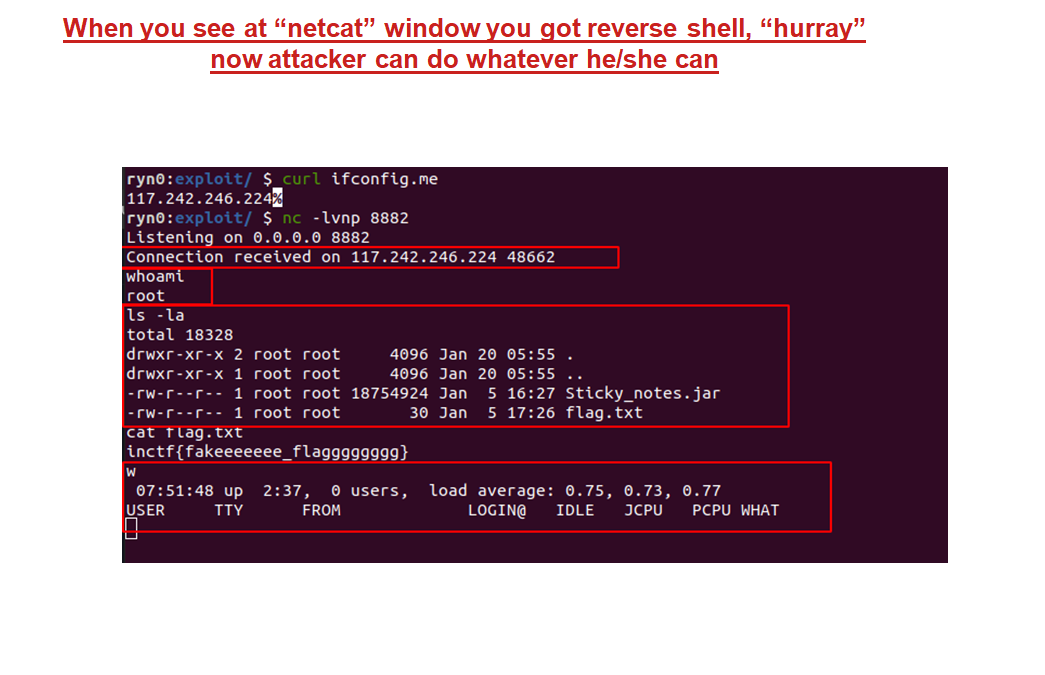
**Fig.5.23**

****

**Fig.5.24**

****

**Fig.5.25**

****

**Fig.5.26**

**6. CONCLUSION**

**JAVA** is great language especially in gui programming , it make tools or software very robust ,user-friendly and efficient.

* Gui utility which cross-platform and very easy to setup.
* You can reverse any java byte code in any file extension.
* Menu options,search functionality,forward and backword navigations to easily analyse code of program.
* This program is going to be so precise and efficient ,so reverse-engineer ,product analyst, security penetration tester will not going to make mistake.

Thank you all

**Karan Kumar**

**7.References**

* <https://www.geeksforgeeks.org/jvm-works-jvm-architecture/>
* <https://www.baeldung.com/cs/how-compilers-work>
* <https://www.javatpoint.com/java-bytecode>
* <https://jameshamilton.eu/research/what-decompilation>
* <https://www.infoworld.com/article/2076989/java-s-security-architecture.html>
* <https://en.wikipedia.org/wiki/Reverse_engineering>
* <https://searchcio.techtarget.com/definition/security-audit>
* <https://www.geeksforgeeks.org/introduction-to-java/>
* <https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java>