Security vulnerabilities in Java Based Applications

-By Siwareddy

Below are the security vulnerabilities often found in Java based Applications and will have high impact and severity on the application reliability.

1) **SQL Injection**
2) **Unclosed Resources/Memory leaks**
3) **Directory Traversal Attack/Malicious File uploads :**
4) **Cross site Scripting**
5) **Password Management(logging, hard coding, weak password)**
6) **Poor Logging**
7) **Improper Exception Handling**
8) **Improper validations on user inputs**

1) **SQL Injection : Inject malicious SQL statements in user input**

What is the Impact?

- An attacker can inject malicious SQL code in the user input
- An Attacker can view sensitive data stored in the database
- An Attacker can bypass the authentication/authorization check
- An Attacker can update/delete/drop the tables data based on the type of the SQL injection

How to avoid it?

- Proper input validation on the user input data to make sure that user input must not have the SQL injections (E.g verify for characters ', SQL keywords like OR, like, where, =, <> etc..)
- Always use the prepared statements to execute the SQL statements instead of preparing the SQL query by appending the user inputs directly to the SQL query
- Use source code analyzer tools like SonarLint\SonarQube\FindBugs in the development environment. These tools analyze the source code and provide feedback to the developer if it found any SQL injections found in the source code.

Example : query = “select userid, username from Users where userid=” + userId ;

UserId is input value from user then attacker can send userId as 101 OR 1=1 instead of 101

Then above query give result set with all users available in the table.

2) Unclosed resources\Memory Leaks

- In Java, a memory leak happens when your code holds a reference permanently, so that some object is never garbage collected
- Classes implementing the interface java.io.Closeable (since JDK 1.5) and java.lang.AutoCloseable (since JDK 1.7) are considered to represent external resources, which should be closed using method close(), when they are no longer needed.

Example: Connection, Statement,ResultSet I\O Streams etc..

What is the Impact?

- Most operating systems limit the number of open files. If you fail to close your streams then GC may take a very long time to close them for you; the net result may be that you run out of system file descriptors and your code fails to open one more file
- Similar issue for opening many database connections/statements
- JVM can go OutOfMemory if too many resources are in open

How to avoid it ?

- For Classes implementing java.io.Closeable, java.lang.AutoCloseable must close the resources using finally block, from JDK 1.7 onwards can use try-with-resources
- Use static code analyzer tools like SonarLint\SonarQube\FindBugs in the development environment. These tools analyze the source code and provide feedback to the developer if it found any resources are not closed in the source code.

3) Directory Traversal Attack/Malicious File uploads:

- Directory traversal is also known as the ../ (dot dot slash) attack, directory climbing, and backtracking
- The goal of this attack is to use an affected application to gain unauthorized access to the file system

GET/POST www.vulnerabilities.com/filename=input.txt

Now attacker can give filename = ../../../../../../../passwords.txt

How to avoid it?

- Must place the validations to check filename contains ../ or ..\ or
- Must check for file extensions, allowed special characters in the file names, filename length etc...
- Must verify file path coming in the request is absolute path or relative path and do validation against the relative path configured in app
- Must have validations against the file upload size

4) Cross site Scripting: Inject client-side scripts into web pages viewed by other users

For the XSS attack, three things are mandatory. 1) Attacker 2) Vulnerability web site 3) victim (An user of the vulnerability web site)

XSS attack can happen in two ways 1) Non – persistent 2) persistent

5) Password Management (logging, hard coding, weak password):

- Application must have a strong password policy(password contains combination of special characters, numbers, lower/upper case letters with minimum length of 9)
- Sensitive information like PII(passwords, credit cards, SSN) must not print on the loggers
- Never hardcode DB passwords, IP address, app server admin credentials in the source code
- Proper ACL in place to authenticate / authorized to access the resources on the server

6) Poor Logging:

- Adding proper loggers in the application makes maintenance of the application easy(troubleshooting the issues)
- Adding more logger to the application results into performance overhead in the application
- Add proper logs when an exception thrown\catch in the application
- Never log add PII to the loggers
- Take wise decision while adding logger to level (error, info, debug, trace, warn, all)

7) Improper Exception Handling:

- Never swallow the exception (e.g. catch IOException but rethrow FileNotFoundException\SQLException)
- Exception handling order should be from specific exception type to generalized one (E.g. First catch FileNotFoundException then IOException then Exception)
- Never catch Throwable Exception
- Cleanup resources always after the catch block using the finally Block
- Never catch exception in finally block
- Better to use try-with-resources to clean up the resources (JDK 1.7 onwards), lower JDK 1.7 versions have close resources using finally block
- Add the proper log before throw\catch the exceptions
- Make sure null pointer exception handled properly (i.e.before invoke the operation on the object check for null condition)

8) Improper validations on User inputs:

- Your application takes integer but user can pass as String then need to have the proper validation to check for String to Integer conversion
- Validation against the filename\file extensions\file size
- Validation against the length of the user input etc.
- Validation against the null value for the user inputs