**SAD Lab**

**EXPERIMENT NO. 4**

**Aim**: Study of SAST tools. Use SonarQube to test a simple Java project for vulnerabilities.

**Theory**:

1. What is Secure Application Security Testing?

**Secure Application Security Testing**

Static application security testing (SAST), or static analysis, is a testing methodology that analyses source code to find security vulnerabilities that make your organization’s applications susceptible to attack. SAST scans an application before the code is compiled. It’s also known as white box testing. SAST is used to secure software by reviewing the software's source code to identify sources of vulnerabilities.

Although the process of statically analyzing the source code has existed as long as computers have existed, the technique spread to security in the late 90s. The first public discussion of SQL injection was in 1998 when Web applications integrated new technologies like JavaScript and Flash.

SAST tools focus on the core content of the application, white-box testing. A SAST tool scans the source code of applications and their components to identify potential security vulnerabilities in their software and architecture. Static analysis tools can detect an estimated 50% of existing security vulnerabilities.

Static analysis tools examine the text of a program syntactically. They look for a fixed set of patterns or rules in the source code. Theoretically, they can also examine a compiled form of the software. Static analysis can be done manually as a code review or auditing of the code for different purposes, including security, but it is time-consuming.

SAST tools run automatically, either at the code level or application level and do not require interaction. When integrated into a CI/CD context, SAST tools can be used to automatically stop the integration process if critical vulnerabilities are identified.

1. What are the advantages and disadvantages of SAST?

**Advantages of Secure Application Security Testing**

SAST takes place very early in the software development life cycle (SDLC) as it does not require a working application and can take place without code being executed. It helps developers identify vulnerabilities in the initial stages of development and quickly resolve issues without breaking builds or passing on vulnerabilities to the final release of the application.

SAST tools give developers real-time feedback as they code, helping them fix issues before they pass the code to the next phase of the SDLC. This prevents security-related issues from being considered an afterthought. SAST tools also provide graphical representations of the issues found, from source to sink. These help you navigate the code easier. Some tools point out the exact location of vulnerabilities and highlight the risky code. Tools can also provide in-depth guidance on how to fix issues and the best place in the code to fix them, without requiring deep security domain expertise.

Developers can also create the customized reports they need with SAST tools; these reports can be exported offline and tracked using dashboards. Tracking all the security issues reported by the tool in an organized way can help developers remediate these issues promptly and release applications with minimal problems. This process contributes to the creation of a secure SDLC.

Developers dramatically outnumber security staff. It can be challenging for an organization to find the resources to perform code reviews on even a fraction of its applications. A key strength of SAST tools is the ability to analyze 100% of the codebase. Additionally, they are much faster than manual secure code reviews performed by humans. These tools can scan millions of lines of code in a matter of minutes. SAST tools automatically identify critical vulnerabilities—such as buffer overflows, SQL injection, cross-site scripting, and others—with high confidence. Thus, integrating static analysis into the SDLC can yield dramatic results in the overall quality of the code developed.

**Disadvantages of Secure Application Security Testing**

Even though developers are positive about the usage of SAST tools, there are different challenges to the adoption of SAST tools by developers.

With Agile Processes in software development, early integration of SAST generates many bugs, as developers using this framework focus first on features and delivery.

Scanning many lines of code with SAST tools may result in hundreds or thousands of vulnerability warnings for a single application.

It generates many false positives, increasing investigation time and reducing trust in such tools.

This is particularly the case when the context of the vulnerability cannot be caught by the tool.

1. What are the tools used in SAST?

**SonarQube**

SonarQube community edition provides bug and vulnerability detection, code smell tracking, technical debt reviews and remediations, and code quality history and metrics. You can integrate SonarQube with CI/CD and extend its functionality further using more than 60 community plugins.

SonarQube can detect injection flaws and provides real-time IDE notifications. It can also add a quality gate and pull request information to the Application Lifecycle Management (ALM) interface.

The Developer Edition has all the features of the community editions and more, catering for more languages, 22 languages to be exact (ABAP, C, C++, CSS, Flex, HTML, Go, JavaScript, Java, Objective-C, Kotlin, PL/SQL, PHP, C#, Python, Ruby, Scala, Swift, T-SQL, VB.Net, TypeScript and XML) and also includes injection flaw detection, real-time notifications in the IDE as part of SonarLint smart notifications, pull request decoration where information from the Pull Request analysis and the Quality Gate is added to the interface of the tools used to manage the Application Lifecycle Management (ALM). There is also an open source project by OWASP where there is a version called OWASP SonarQube.

Delivery model: On-premises

**Checkmarx**

Checkmarx helps in checking for errors in the source code and detecting issues with security and regulation compliance. The system works by giving a flow of the code and then checking whether there are any issues.

For each language, the system has a list of security vulnerability issues. You can configure or inquire about other issues yourself through the CxSAST auditing tool; you get either static reports or displayed on the interface.

Not only do you get accurate feedback on your code, but you can also set the system to display false positives. The application allows the user to obtain security reports at any time in the cycle of the project. The CxSAST has an open-source analysis software that supports most languages; hence, an organization can effectively secure its code analysis components.

The software can be integrated into the building of automation tools, software development, and vulnerability management.

Checkmarx SAST (CxSAST) is a static analysis tool providing the ability to find security vulnerabilities in source code in a number of different programming and scripting languages.

Language support: Supports over 18 languages including Java, C#, VB.NET, ASP, C/C++, PHP, Ruby, JavaScript, HTML5, PL/SQL, Groovy, and Scala.

Delivery model: Cloud, on-premises, and hybrid

**Veracode SAST**

Veracode analyzes application source code and provides automated security feedback via the CI/CD pipeline and IDE. It provides software composition analysis (SCA), security management, audit trail, and reporting.

Veracode offers a manual penetration testing system that allows professionals to analyze the results of security tests to minimize application risk, ensure regulatory compliance, and provide security posture reports. Veracode also enables employees to set security goals for Dev teams, configure risk mitigation workflows, and streamline policy management operations.

Veracode integrates with CI/CD tools including Apache Ant, Docker, Artifactory, Bugzilla, Bamboo, Gradle, Jira, Github, and more, and offers an API for further customization.

Language support: Supports 30 languages including Java (Java SE, Java EE), JDK and OpenJDK, C# and .NET, ASP.NET, C++, JavaScript and TypeScript, PHP, and Scala

Delivery model: Cloud

**Procedure:**

*Create a Java Maven project named sonar\_example in Eclipse and edit the code in App.java and pom.xml as shown below*:

**App.java**:

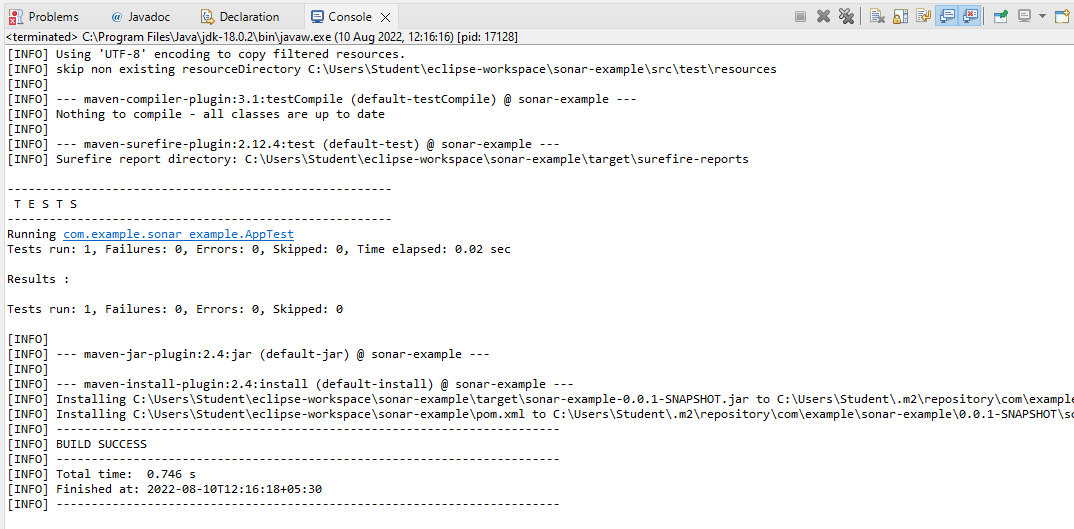
| package com.example.sonar\_example;  import com.sun.tools.javac.util.List; import java.util.\*;  public class App  {      public App() {  System.out.println("Constructor");  }    public void m1() {  String s = "";  ArrayList<Integer> list = new ArrayList<Integer>();  list.add(1);  list = null;  list.add(2);  Object obj = getData();  System.out.println(obj.toString());  }    public Object getData() {  return null;  }   public static void main(String[] args) {  System.out.println("Hello World!");  App app = new App();  app.m1();  } } |
| --- |

**pom.xml:**

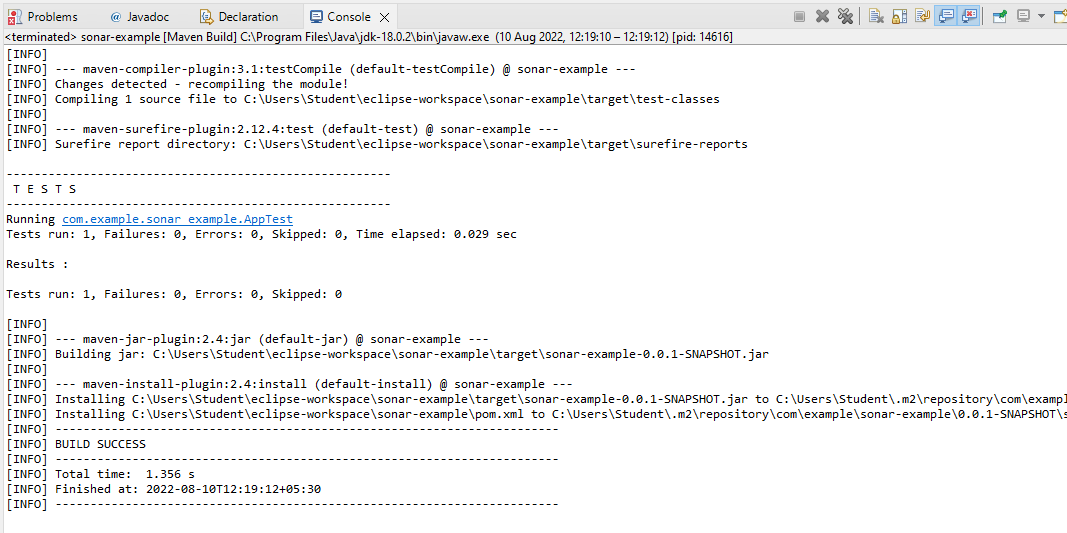
| <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>   <groupId>com.example</groupId>  <artifactId>sonar-example</artifactId>  <version>0.0.1-SNAPSHOT</version>   <name>sonar-example</name>  *<!-- FIXME change it to the project's website -->*  <url>http://www.example.com</url>   <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <maven.compiler.source>1.7</maven.compiler.source>  <maven.compiler.target>1.7</maven.compiler.target>  </properties>   <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.11</version>  <scope>test</scope>  </dependency>  *<!-- https://mvnrepository.com/artifact/org.sonarsource.scanner.maven/sonar-maven-plugin -->*  <dependency>  <groupId>org.sonarsource.scanner.maven</groupId>  <artifactId>sonar-maven-plugin</artifactId>  <version>3.4.0.905</version>  </dependency>   </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.sonarsource.scanner.maven</groupId>  <artifactId>sonar-maven-plugin</artifactId>  <version>3.4.0.905</version>  </plugin>  <plugin>  <groupId>org.jacoco</groupId>  <artifactId>jacoco-maven-plugin</artifactId>  <version>0.8.4</version>  </plugin>  </plugins>  </build> </project> |
| --- |
|  |

Compile, build and install the maven project.

*mvn clean install*

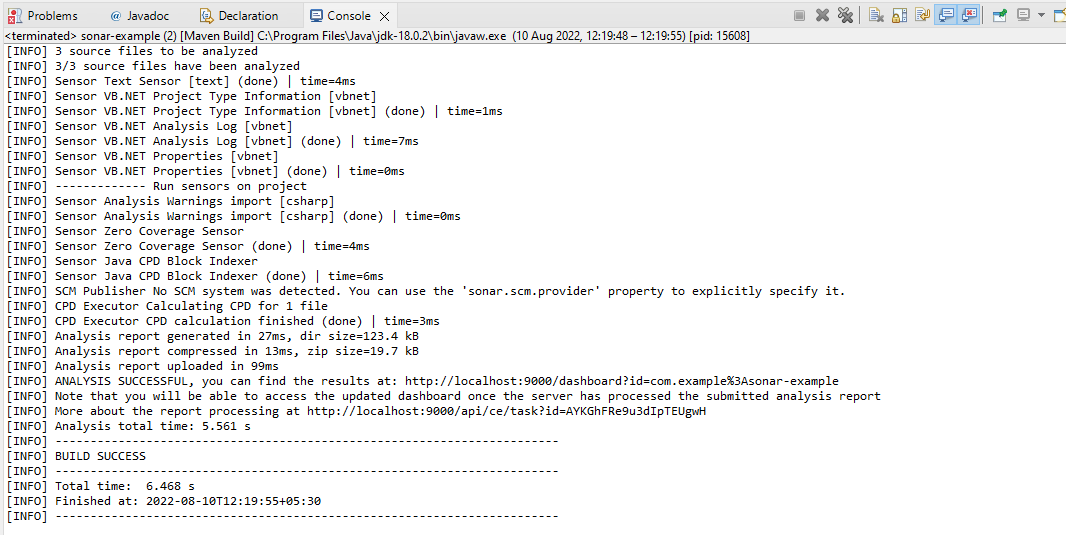


*mvn build sonar-example: clean.org.jacoco:jacoco-maven-plugin:prepare-agent install*

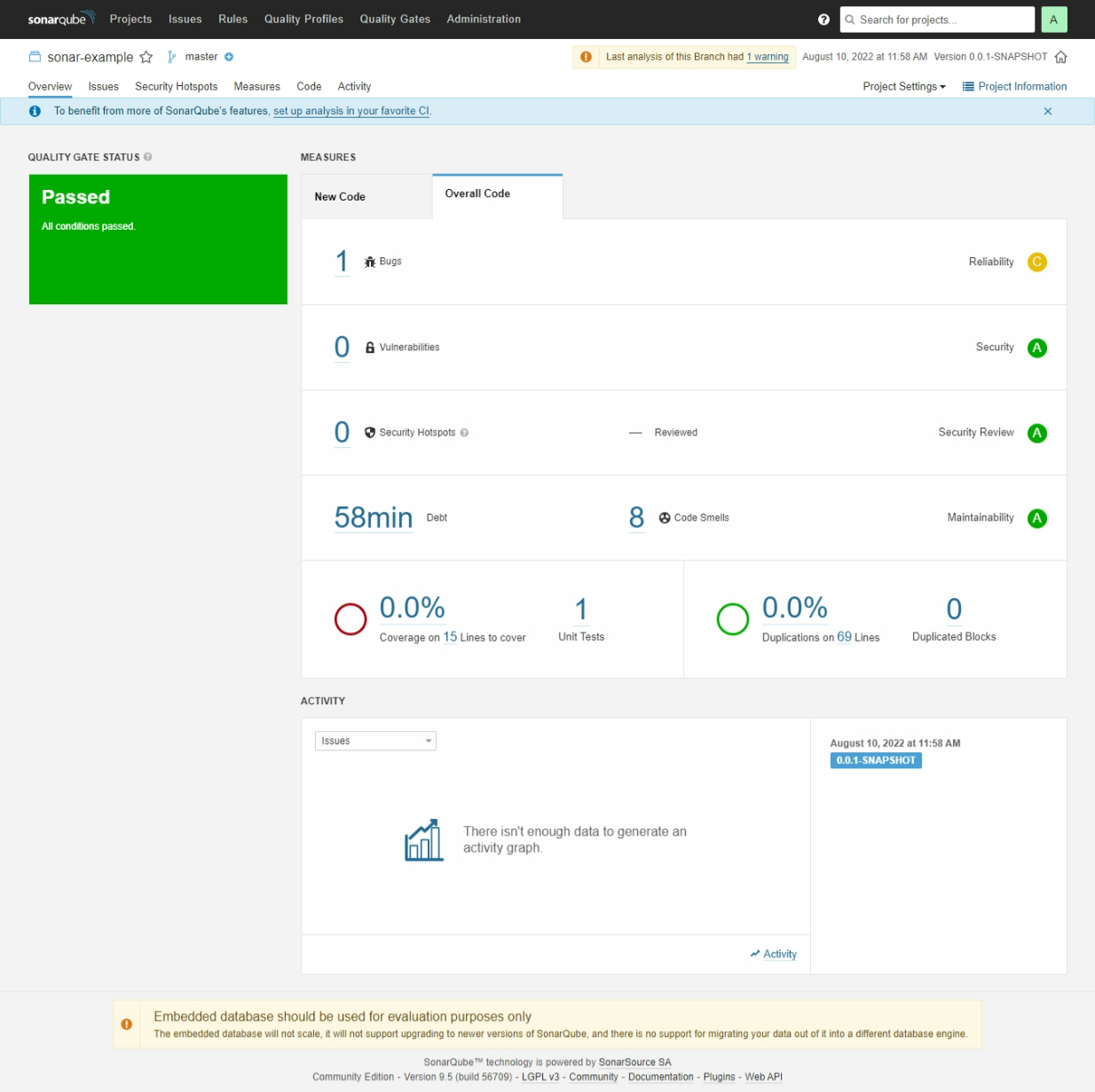


Run the project with goal sonar:sonar.

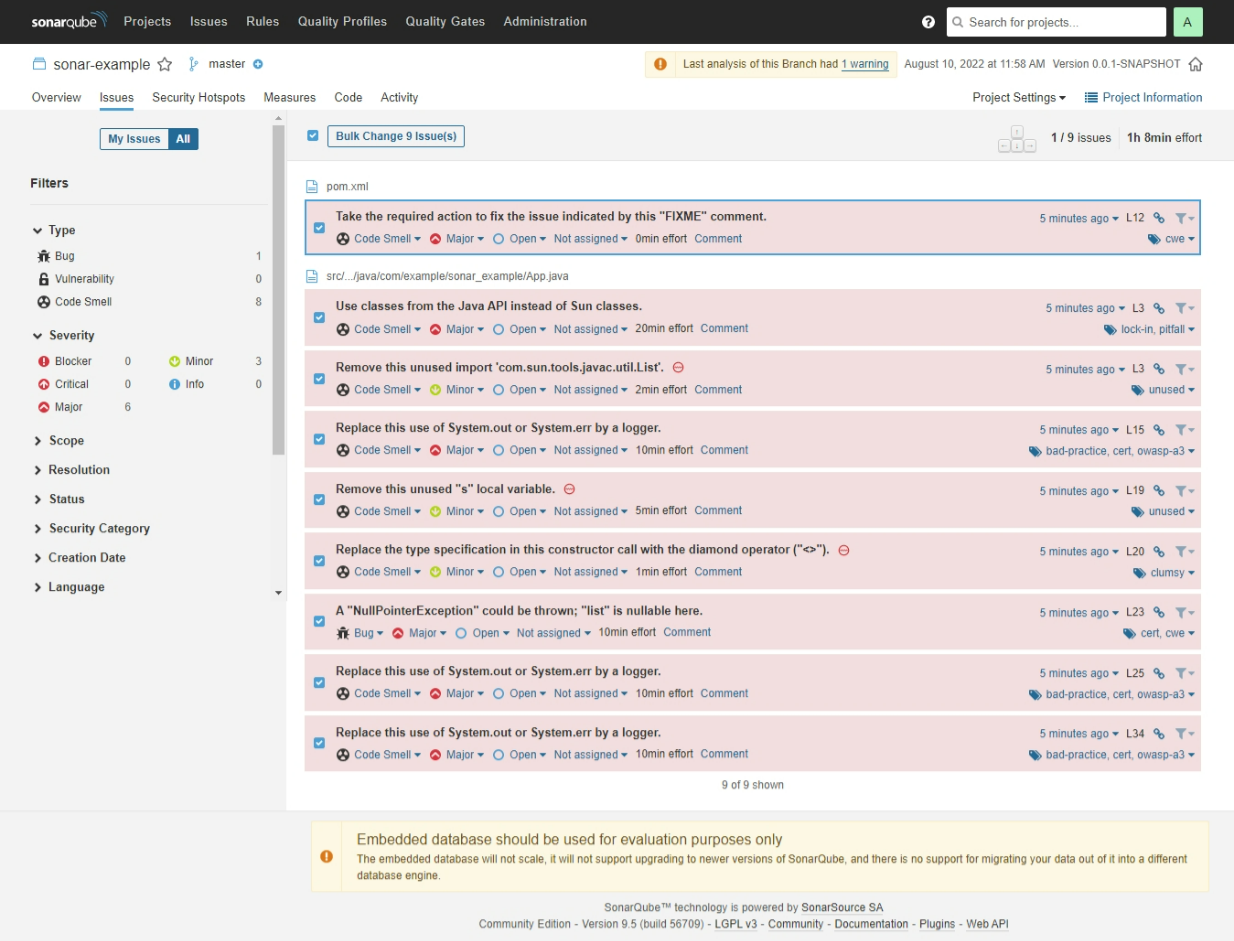
*mvn sonar:sonar -Dsonar.login=admin -Dsonar.password=root@513*



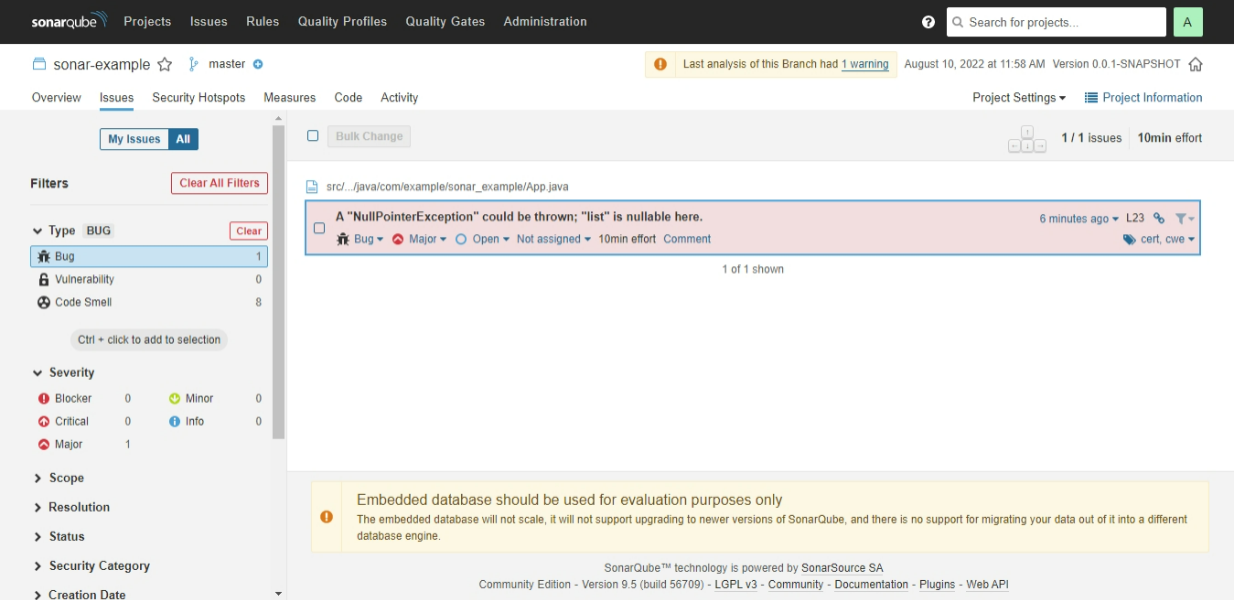
Open the default URL of SonarQube and navigate to the Projects page. Our project must be reflected in this page. Click on the project name.

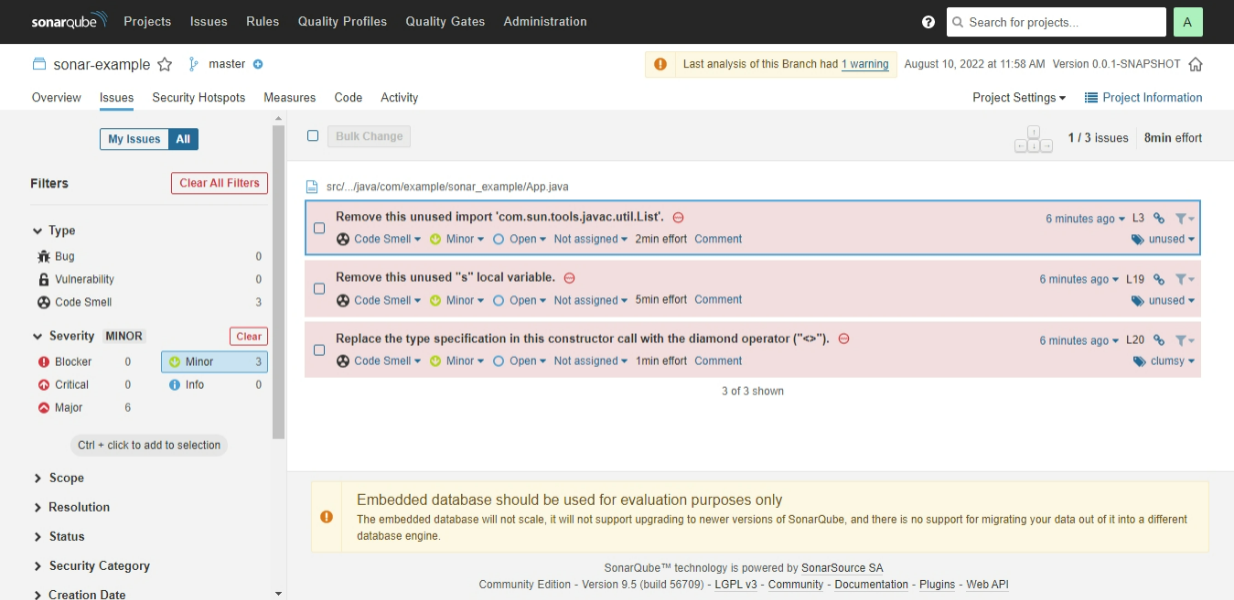


The issues tab lists down all the errors and warnings found in the code of the project.



We can apply filters like choosing the Type, Severity, etc of the errors.





**Conclusion**:

Thus, we have learnt about Secure Application Security Testing, its role in Software development and also used SonarQube Tool for performing SAST on a Java Maven project.