
Internship Project Title	Application of Static Application Security Testing (SAST) Tools
Name of the Company	TCS iON
Name of the Industry Mentor	Vineet Kannoly/Shreya Masurkar
Name of the Institute	Talent Battle

Start Date	End Date	Total Effort (hrs.)	Project Environment	Tools used
22/11/2021	6/12/2021	27	Windows	SonarQube

Expertise Professional SAST report of all Defects Uncovered in Source Code

TABLE OF CONTENT

Acknowledgements

I am fortunate to have been a part of this internship program provided by TCS-iON, and I would like to thank Talent Battle for letting me know about this exceptional internship opportunity.

I would like to express my deepest gratitude to my industry mentor, Vineet Kannoly and Shreya Masurkar for providing me with guidance at every step of this internship and enlightening me with their industry expertise. Also, I greatly appreciate Mr. Himanshu Saraswat for his great feedback, excellent encouragement, and guidance. Thanks a lot, Sir.

Finally, I am grateful for my parents whose constant love and support keep me motivated and confident. My accomplishments and success are because they believed in me.

Objective

Hacker-proof web-based applications that are accessible to the public on the Internet are necessary. Hackers seek for and exploit hidden faults (also known as vulnerabilities) in all websites for a multitude of reasons, including pleasure and personal gain. Hackers can abuse a vulnerability to induce a crucial software to collapse, or they can deface a website, or they can undertake illegal activities to get an unfair advantage, or they can steal sensitive data and hold the website hostage.

The project's goal is to determine the most prevalent coding errors committed by web-based application programmers The aim is to gain a basic understanding of the most prevalent web-based application vulnerabilities, as well as how to find these major security flaws from the inside out by studying the source code.

Introduction

Web applications have become an incredibly important part of our daily lives due to their extensive and ever-increasing consumption. As a result of the large number of users linked with online applications, the current environment has seen them become increasingly susceptible. Hackers aim to seize web applications in order to procure personal information from consumers. Unfortunately, most web applications are vulnerable to attackers because their source code is improperly structured and written. Developers must create secure web apps by avoiding security flaws.

Static code analysis, as contrast to dynamic code analysis (testing software by executing programs), is the study of computer software that is conducted without the actual execution of the programs developed from that software.

As automated source code analysis tools play a crucial role in finding and correcting security-related issues, source code analysis is becoming increasingly vital for the expansive advancement of web applications.

The primary objective of this document is to provide a concise description of static code analysis, its attributes, and prospects, as well as an overview of the constructs and technologies that underpin this type of software development approach, as well as the tools that enable software developers to use code reviewing tools to assist in the development of applications, allowing them to optimize the code and rectify blunders before it is executed. For the case study, I have evaluated the source code of http://demo.testfire.net/.

Internship Activities

In the first phase of the project activities, I learnt about OWASP (Open Web Application Security Project) which is an online platform that publishes papers, approaches, documentation, tools, and technologies in the domain of web application security that are all publicly accessible. The Open Web Application Security Project offers resources that are both free and open.

I gained a detailed understanding of the top 10 OWASP vulnerabilities. The OWASP Top 10 is a standard awareness document for web application security and developers. It reflects widespread agreement on the most significant security threats to online applications.

Following are the top 10 OWASP vulnerabilities 2021:

- **Broken Access Control**: Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.

  Common access control vulnerabilities include:

  - Access should only be allowed for capabilities, roles, or users, but is open to everybody, in violation of the concept of least privilege or restrict by default.
  - By giving a unique identification for someone else's account, you may see or update it (insecure direct object references).
  - POST, PUT, and DELETE access restrictions are absent from the API.
  - Elevation of privilege. Acting as a user while not logged in, or as an admin when logged in as a user.
  - As an unauthenticated user, force browsing to authorized pages, or as a normal user, force browsing to privileged pages.

  Prevention methods include:

  - Restrict by default, with the exception of public resources.
  - Develop access control methods once and reuse them across the program, minimizing the utilization of Cross-Origin Resource Sharing (CORS).

- Instead of allowing the user to create, read, edit, or delete any record, model access controls should guarantee record ownership.
- Failures in access control are logged, and administrators are notified as necessary (e.g., repeated failures).

- **Cryptographic Failures**: Prevention methods for cryptographic failures:

  - Don't keep critical information around needlessly. It should be discarded as quickly as feasible, or it should be tokenized or truncated in accordance with PCI DSS. It is impossible to steal data that is not saved.
  - Make sure all critical information is encrypted at rest.
  - Verify that standard algorithms, protocols, and keys are up to date and robust; utilize correct key management.
  - Encrypt all data in transit using secure protocols like TLS, which uses forward secrecy (FS) cyphers, server cypher priority, and secure parameters.
  - Cached responses containing sensitive data should be disabled.
  - Implement the necessary security procedures in accordance with the data categorization.
  - When sending sensitive data, avoid using older protocols like FTP and SMTP.
  - Store passwords using Argon2, scrypt, bcrypt, or PBKDF2, which are effective adaptive and salted hashing methods with a work factor (delay factor).
  - Instead of just encryption, always employ authenticated encryption.
  - Use MD5, SHA1, and PKCS number 1 v1.5 instead of outdated cryptographic functions and padding schemes.

- **Injection**: An application is vulnerable to injection attack when:

  - The application does not check, filter, or sanitise user-supplied data.
  - In the interpreter, dynamic queries or non-parameterized calls without context-aware escaping are utilised directly.
  - To extract additional, sensitive records, hostile data is employed inside object-relational mapping (ORM) search criteria.

- Violent data is utilised directly or concatenated. In dynamic queries, commands, or stored procedures, the SQL or command contains the structure and harmful data.

Prevention methods:

- To avoid bulk exposure of records in the event of SQL injection, use LIMIT and other SQL restrictions within queries.
- Use server-side input validation that is affirmative or "whitelisted."
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.

- **Insecure Design**: Insecure design is a wide term that encompasses a variety of flaws and is defined as "missing or poor control design." The absence of business risk profile inherent in the software or system being produced, and hence the failure to decide what level of security design is necessary, is one of the reasons that lead to unsafe design.

  Prevention Methods:

  - To assist analyze and build security and privacy-related controls, create and employ a safe development lifecycle with AppSec specialists.
  - For crucial authentication, access control, business logic, and key flows, use threat modelling.
  - User stories should include security language and controls.
  - Integrate plausibility checks into your application at each level (from frontend to backend).

- **Security Misconfiguration**: The application might be vulnerable if the application is:
  - Inadequately set permissions on cloud services or a lack of suitable security hardening across any element of the application stack.
  - Unneeded features have been activated or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
  - Default accounts and passwords are still active and unaltered.
  - The software is either outdated or insecure.

- **Vulnerable and Outdated Components**: Following can be the reasons for this vulnerability:

    - If you don't know the versions of all the components you use, don't use them (both client-side and server-side).
    - If the software is outdated, unsupported, or insecure.
    - If you don't check for vulnerabilities on a regular basis and subscribe to security bulletins for the components you use, you're at risk.
    - If software developers do not test updated, upgraded, or patched libraries for compatibility.

- **Identification and Authentication Failures**: Following can be the reasons for Identification and Authentication Failures:

    - Allows for automated assaults like credential stuffing, in which an attacker has a list of legitimate users and passwords.
    - Allows for automated assaults such as brute force.
    - Allows passwords like "Password1" or "admin/admin" that are default, weak, or well-known.
    - Uses insecure or inadequate credential recovery and forgotten-password procedures, such as "knowledge-based responses" that can't be made secure.
    - Passwords are stored in plain text, encrypted, or weakly hashed form.
    - Multi-factor authentication is either missing or ineffective.

    Prevention Methods:

    - Multi-factor authentication should be used wherever possible.
    - Default credentials should not be sent or deployed, especially for admin users.
    - Check for weak passwords, such as by comparing new or updated passwords to the top 10,000 worst passwords list.
    - Password length, complexity, and rotation rules should all be in line with N.

- **Software and Data Integrity Failures**: Code and infrastructure that do not guard against integrity violations are referred to as software and data integrity failures. Many applications now have auto-update capabilities, which allows updates to be

downloaded without adequate integrity checking and deployed to previously trusted applications. Attackers might theoretically distribute and run their own updates across all systems.

Prevention Methods:

- Use digital signatures or other similar measures to ensure that the software or data is genuine and has not been tampered with.
- To reduce the risk of harmful code or configuration being introduced into your development pipeline, make sure there is a review procedure in place for code and configuration modifications.

- **Security Logging and Monitoring Failures**: Insufficient logging, detection, monitoring, and active response occurs any time:

  - Logins, unsuccessful logins, and high-value transactions, for example, are not tracked.
  - Warnings and errors provide no, insufficient, or ambiguous log messages.
  - Suspicious behavior is not tracked in application and API logs.
  - Only local logs are kept.
  - There are no or ineffective alerting thresholds or response escalation systems in place.
  - Active cyberattacks cannot be detected, escalated, or alerted in real-time or near real-time by the application.

  Prevention Methods:

  - Ensure that all login, access control, and server-side input validation errors are documented with enough user context to identify suspicious or malicious accounts, and that the logs are kept for long enough to allow for delayed forensic investigation.
  - Ensure that logs are created in a format that can be readily consumed by log management software.
  - To prevent injections or assaults on logging or monitoring systems, make sure log data is encoded appropriately.
  - DevSecOps teams should set up good monitoring and alerting so that suspicious activity can be recognized and dealt with swiftly.

- **Server-Side Request Forgery (SSRF)**: When a web application fetches a remote resource without verifying the user-supplied URL, an SSRF fault occurs. Even if the application is secured by a firewall, VPN, or another sort of network access control list, an attacker can force it to send a forged request to an unexpected location (ACL).

  Prevention Methods:

  - The easiest way to remediate SSRF is to whitelist any domain or address that your application accesses.
  - Never send a raw response body from the server to the client. Responses that the client receives need to be expected.
  - Allow only URL schemas that your application uses. There is no need to have ftp://, file:/// or even http:// enabled if you only use https://.
  - Never trust user input. Always sanitize any input that the user sends to your application. Remove bad characters, standardize input (double quotes instead of single quotes for example).

I studied about some of the most common vulnerabilities found in application source code such as:

- **Hardcoded Passwords and Empty Passwords**: Hardcoded passwords may jeopardize system security in ways that are difficult to fix. Hardcoding a password is never a smart idea. Not only does hardcoding a password allow all of the project's developers to see it, but it also makes it incredibly difficult to resolve the problem. The password cannot be changed once the code is in production without updating the programme. If the password-protected account is compromised, the system's owners will have to choose between security and availability.

- **Unreleased resources:** It's possible that the software will fail to release a system resource. The majority of unreleased resource issues result in general software dependability concerns, however if an attacker can deliberately cause a resource leak, the attacker may be able to execute a denial-of-service assault by emptying the resource pool. Resource leaks have at least two common causes:

  - Error situations and other extraordinary events are the most typical sources of resource leakage.
  - There is some confusion over which element of the software is in charge of releasing the resource.

- **Missing check against null**: A NullPointerException might be thrown if the software dereferences a null pointer. Null pointer errors are frequently caused by a breach of one or more programming assumptions. The majority of null pointer issues result in general software reliability issues, but if an attacker can deliberately prompt a null pointer dereference, the attacker may be able to use the resulting exception to bypass security logic or cause the application to expose debugging information that will be useful in strategizing consequent cyberattacks.

- **Path manipulation**: When user-controllable data is placed in a file or URL path that is utilized on the server to access local resources, which may be within or outside the web root, file path manipulation vulnerabilities occur. If the file path is insecure, an attacker can change it to access alternative resources, some of which may include sensitive data. Even if an assault is limited to the web root, objects that are ordinarily restricted from direct access, such as application configuration files, source code for server-executable scripts, or files with extensions that the web server is not set to serve directly, can often be retrieved.

- **System information leak**: Information leakage permits sensitive data, such as application technical details, developer comments, the environment, or user-specific data, to be revealed by an application. This sensitive information might be used by an attacker to gain access to the target programme, its hosting network, or its users. Information leakage is most caused by one or more of the following conditions: a failure to scrub away HTML/script comments containing sensitive data; inappropriate application or server setups; or variations in page replies for valid vs. invalid input.

- **Poor error handling**: Poor error handling can cause a range of security issues for a website. The most typical issue is when the user sees comprehensive internal error messages such as stack traces, database dumps, and error codes (hacker). These notifications expose facts about the implementation that should never be

revealed. Such facts might provide hackers valuable information about the site's possible weaknesses, and such notifications can be upsetting to regular users.

I studied about Static Application Security Testing and Dynamic Application Security Testing:

- **Static Application Security Testing**: Source code analysis tools, commonly known as Static Application Security Testing (SAST) tools, can assist uncover security problems by analyzing source code or developed versions of code. SAST tools can be integrated into your IDE. Such technologies can aid in the detection of flaws in software development. When opposed to uncovering vulnerabilities later in the development cycle, SAST tool feedback can save time and effort.
  SAST occurs early in the software development life cycle (SDLC) since it does not need a functional application and may be performed without executing any code. It assists developers in identifying vulnerabilities early in the development process and promptly resolving bugs without disrupting builds or exposing vulnerabilities in the application's final release.
  SAST tools provide real-time feedback to developers while they code, allowing them to address problems before moving on to the next step of the SDLC. As a result, security-related concerns aren't treated as an afterthought. SAST tools also display graphical depictions of issues discovered, from source to sink. These make it easier to navigate the code. Some technologies show the dangerous code and pinpoint the specific position of vulnerabilities. Without having significant security domain expertise, tools may also give in-depth recommendations on how to repair concerns and the appropriate location in the code to fix them.

- **Dynamic Application Security Testing**: Dynamic analysis security testing, often known as a DAST test, is an application security solution that can assist in the detection of specific vulnerabilities in web applications while they are in use. Because it is done without access to the internal source code or application architecture, a DAST test is often known as a black box test. It effectively utilizes the same approaches that an attacker would use to uncover possible flaws.
  A DAST test may detect a wide range of flaws, including as input/output validation errors that might expose an application to cross-site scripting or SQL injection.

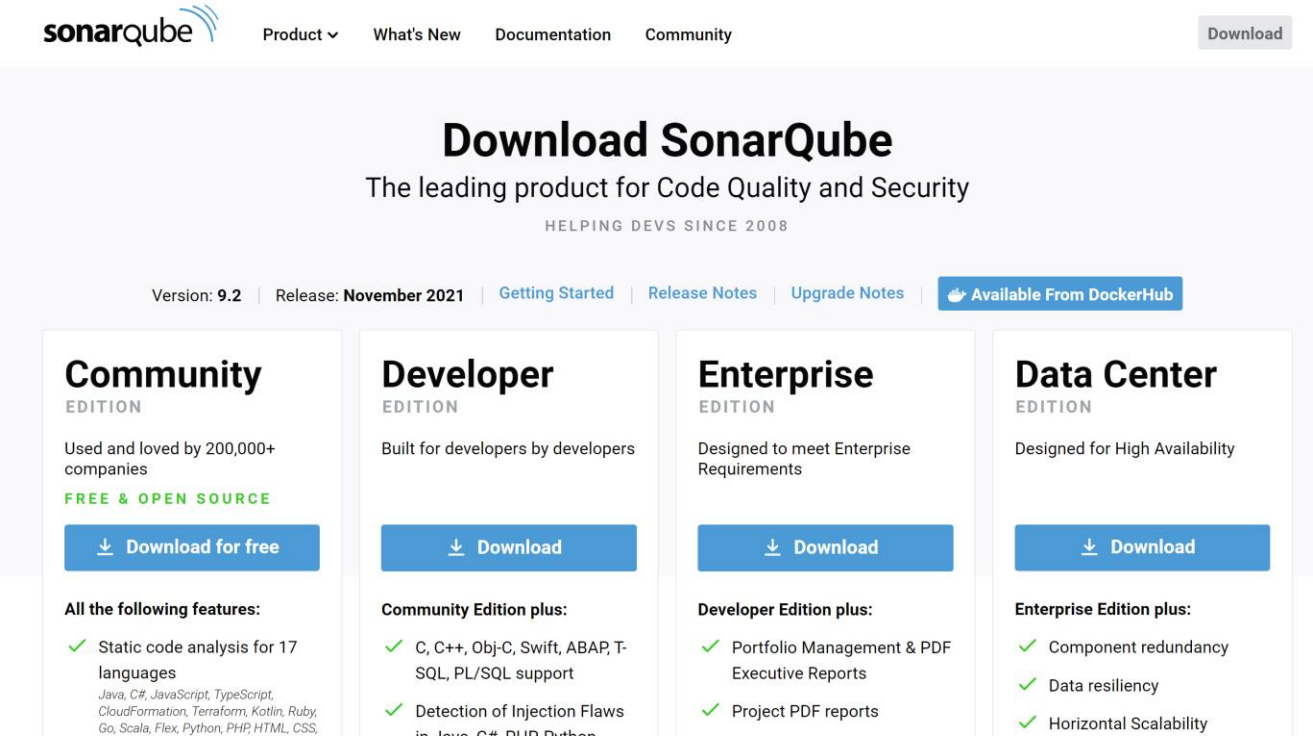
Difference between SAST and DAST:

S.No.	Static Application Security Testing	Dynamic Application Security Testing
1.	SAST is a type of White Box security testing.	DAST is type of Black Box security testing.
2.	In SAST, application is tested from inside out.	In DAST, application is tested from outside in.
3.	This type testing is a developers approach of testing.	This type testing is a hacker's approach of testing.
4.	No deployed application is required for Static Application Security Testing.	A running application is required for Dynamic Application Security Testing.
5.	Finding vulnerabilities, identifying, and fixing bugs is easier in SAST.	Finding vulnerabilities towards end of SDLC.
6.	Fixing vulnerabilities is possible with little cost assistance.	It finds vulnerabilities towards end of SDLC; hence it is expensive to do so.
7.	SAST cannot discover issues related run time and environment.	DAST can discover issues related to run time and environment.
8.	Typically, it supports all types of software like web applications, web services, thick client.	Typically, it only scans apps like web applications, web services but no other types of software.
9.	In this testing, developer has knowledge about design, application framework and implementation.	In this testing, tester has no knowledge about application, design, frameworks, and implementation that application is built on.
10.	SAST testing requires source code to perform testing operation.	DAST testing does not require source code to perform testing operation.
11.	As it scans static code and performs its testing operation that is why it is called Static Application Security Testing (SAST).	As it scans dynamic code and performs its testing operation that is why it is called Dynamic Application Security Testing (DAST).

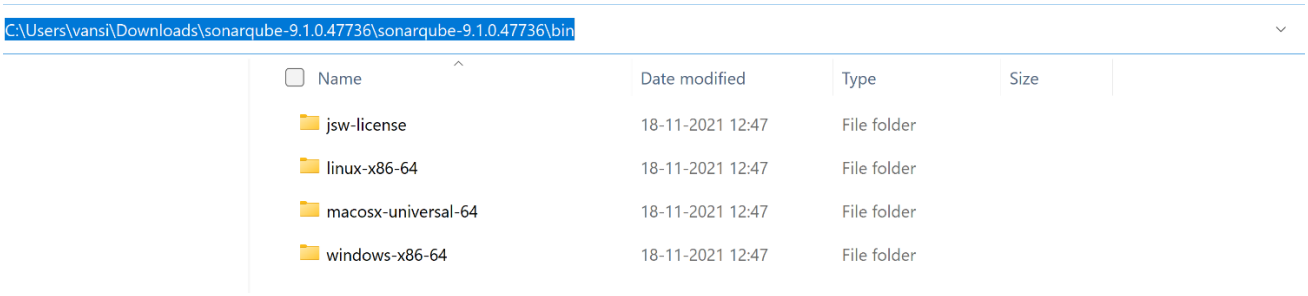
12.	This testing is performed in early stages of Software Development Life Cycle (SDLC).	This testing is performed at end of Software Development Life Cycle (SDLC).
13.	In SAST, there is costly long duration dependent on experience of tester.	In DAST, tester is unable to perform comprehensive application analysis since this is carried our externally.
14.	In SAST, tester can perform comprehensive application analysis.	DAST can be done faster as compared to other types of testing due to restricted scope.

Installation and Configuration of SonarQube

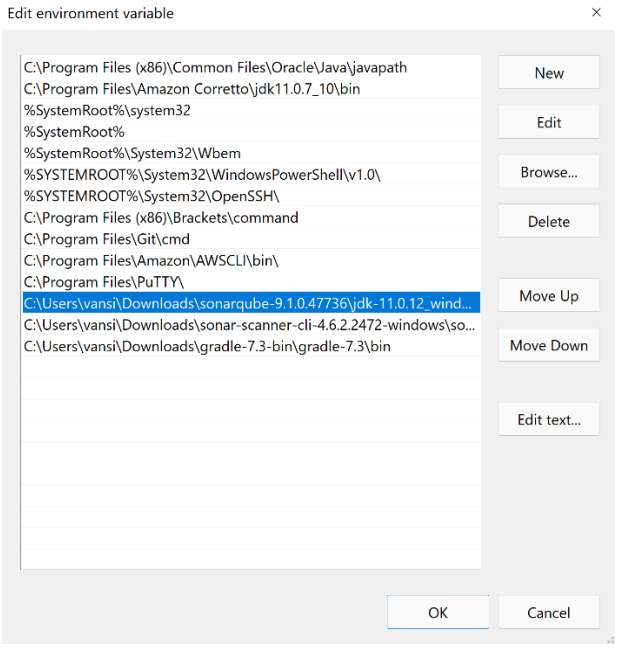
Step-1: Go on https://www.sonarqube.org/downloads/ and select the Community version to download the software.



Step-2: After Downloading go to its folder and unzip the files. Then click on the folder and go to bin and then copy its path.



Step-3: Add that path to the Environment Variables.



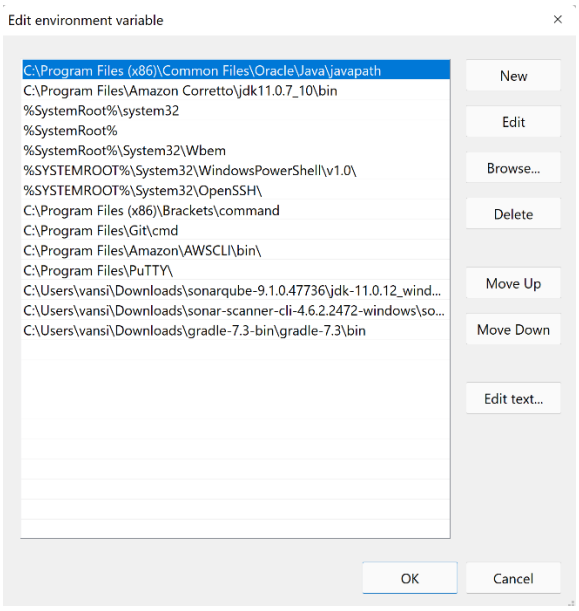
Step-4: Download Java version 11 from https://www.oracle.com/in/java/technologies/javase/jdk11-archive-downloads.html for the SonarQube software to run.

Java SE Development Kit 11.0.12

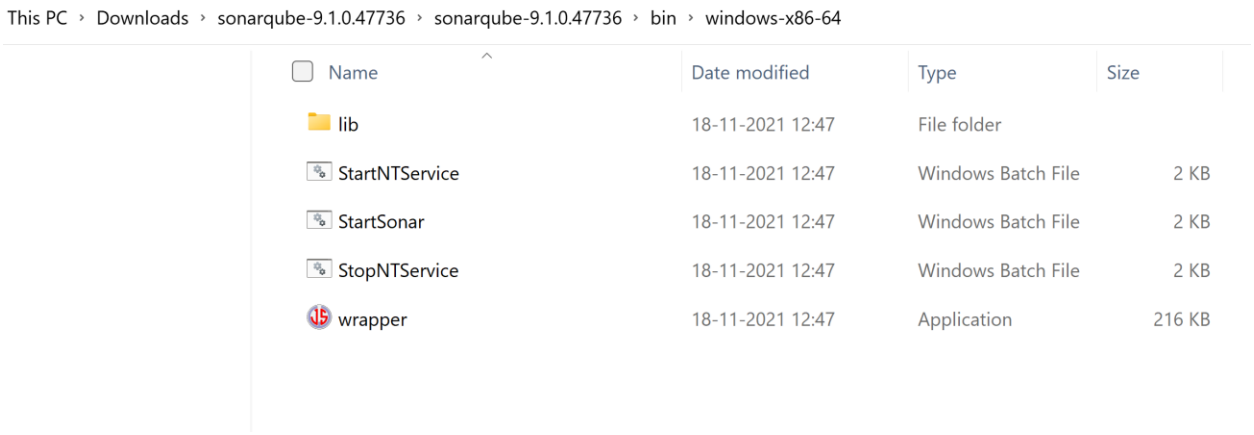
This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE

Product / File Description	File Size	Download
Linux ARM 64 Debian Package	145.98 MB	jdk-11.0.12_linux-aarch64_bin.deb
Linux ARM 64 RPM Package	152.55 MB	jdk-11.0.12_linux-aarch64_bin.rpm
Linux ARM 64 Compressed Archive	169.92 MB	jdk-11.0.12_linux-aarch64_bin.tar.gz
Linux x64 Debian Package	149.74 MB	jdk-11.0.12_linux-x64_bin.deb
Linux x64 RPM Package	156.45 MB	jdk-11.0.12_linux-x64_bin.rpm

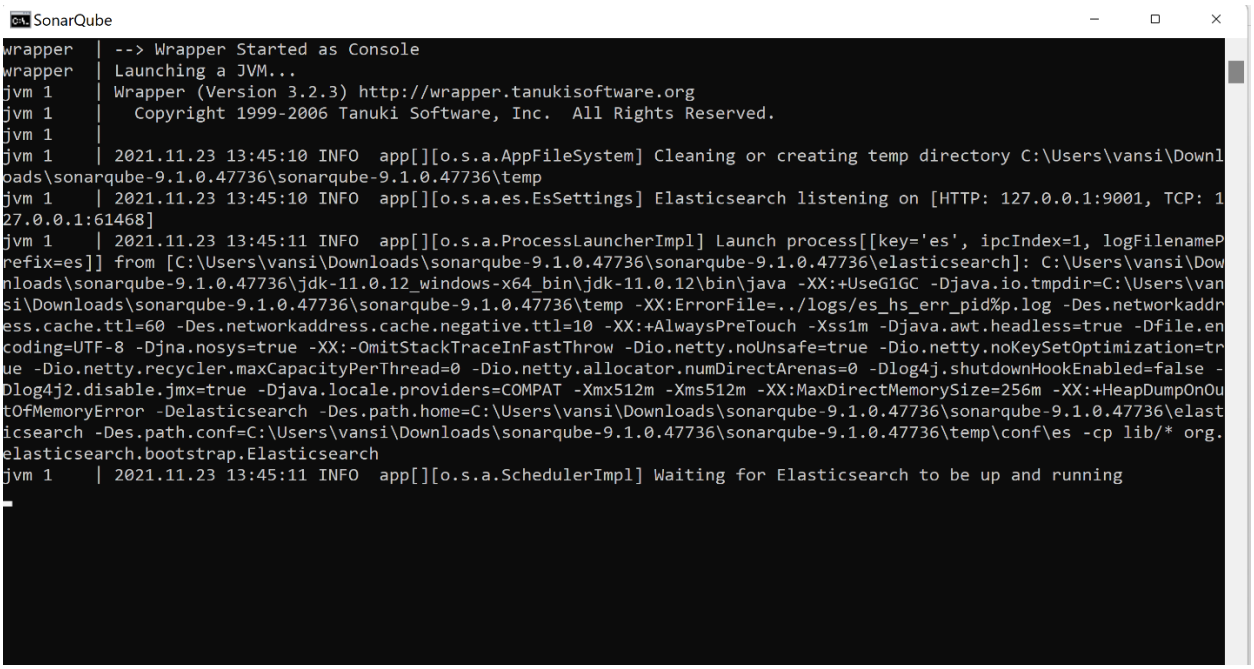
Step-5: After Downloading Java 11, go to its bin folder and add its path in the Environment Variables for SonarQube to run.

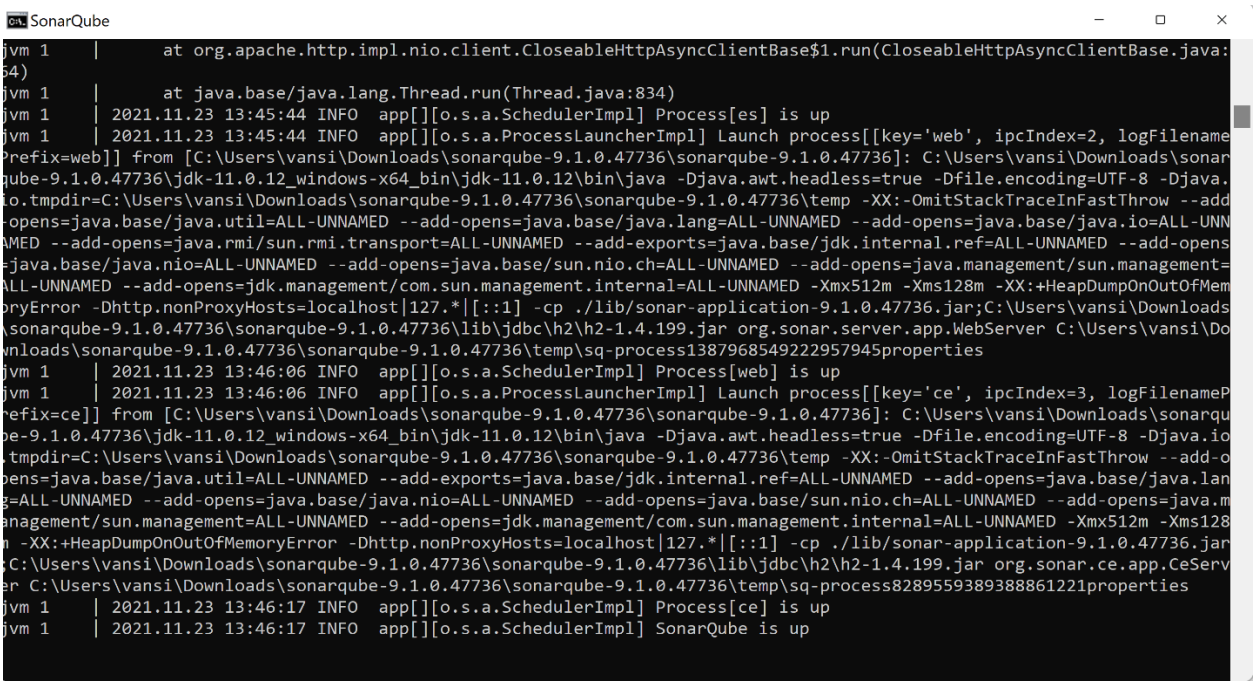


Step-6: Go to SonarQube folder --> bin folder --> Select your OS --> Click on StartSonar

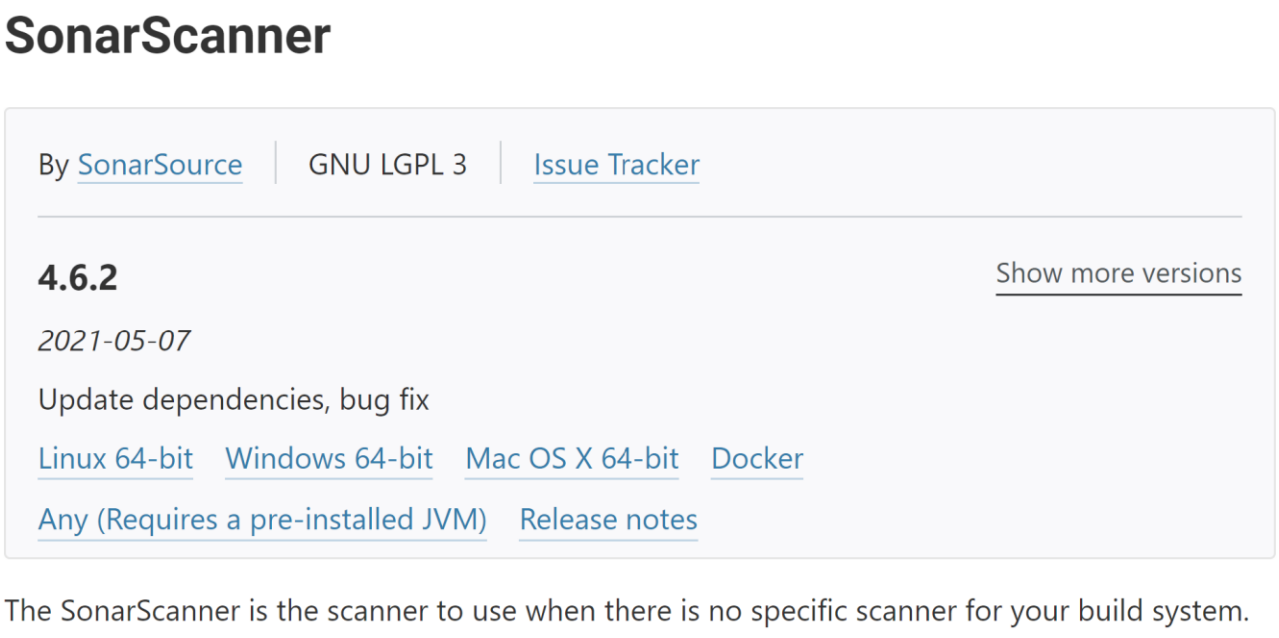


Step-7: SonarQube is up and running will appear in the Command Prompt.

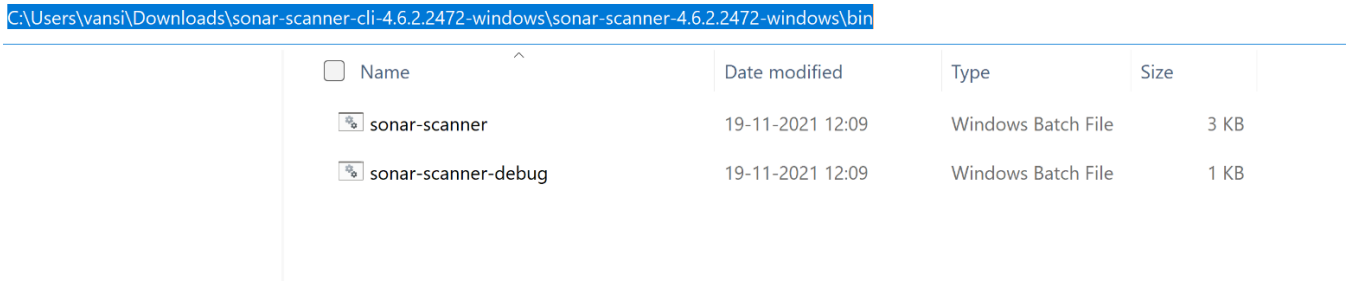




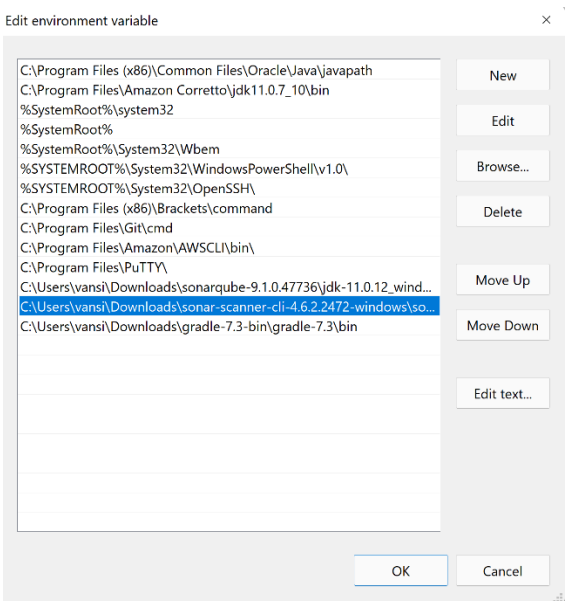
Step-7: Go to https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/ and download Sonar Scanner.



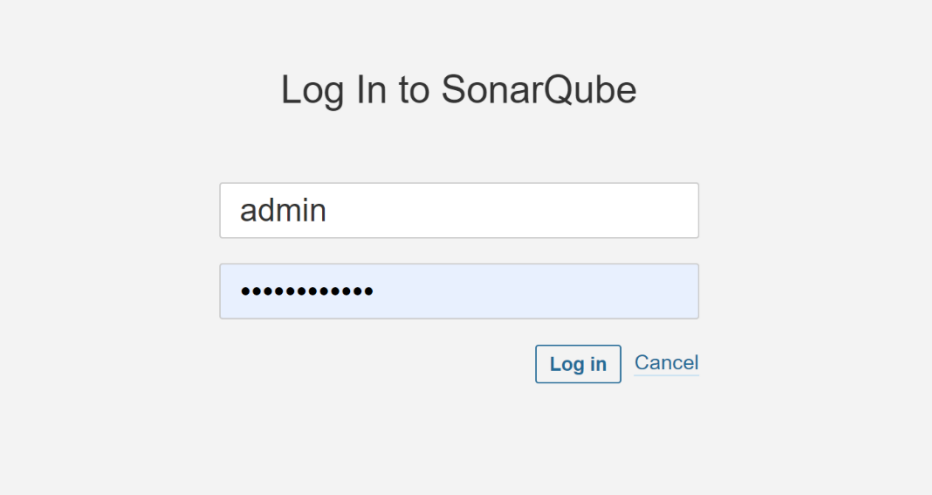
Step-8: Go to sonar scanner's bin folder.



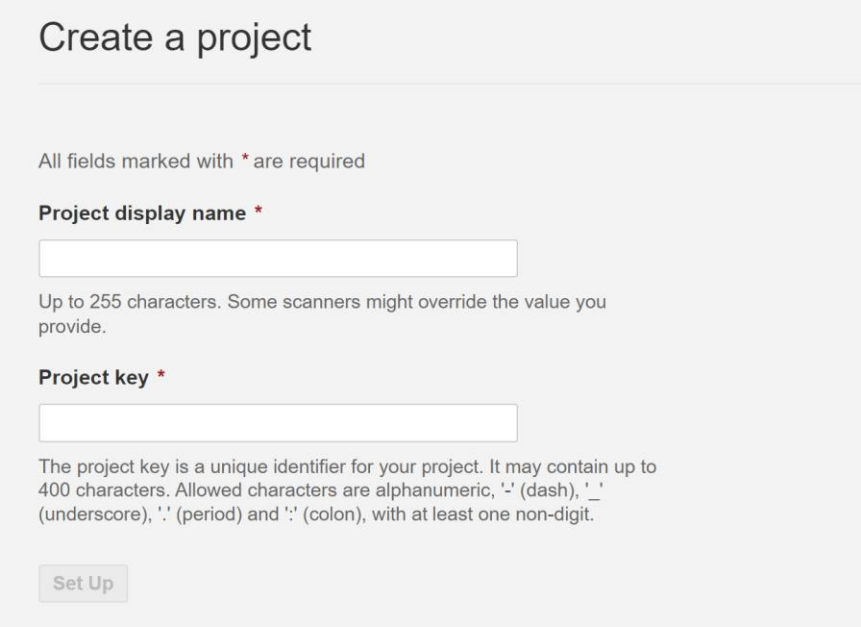
Step-9: Add this path to the Environment Variables.



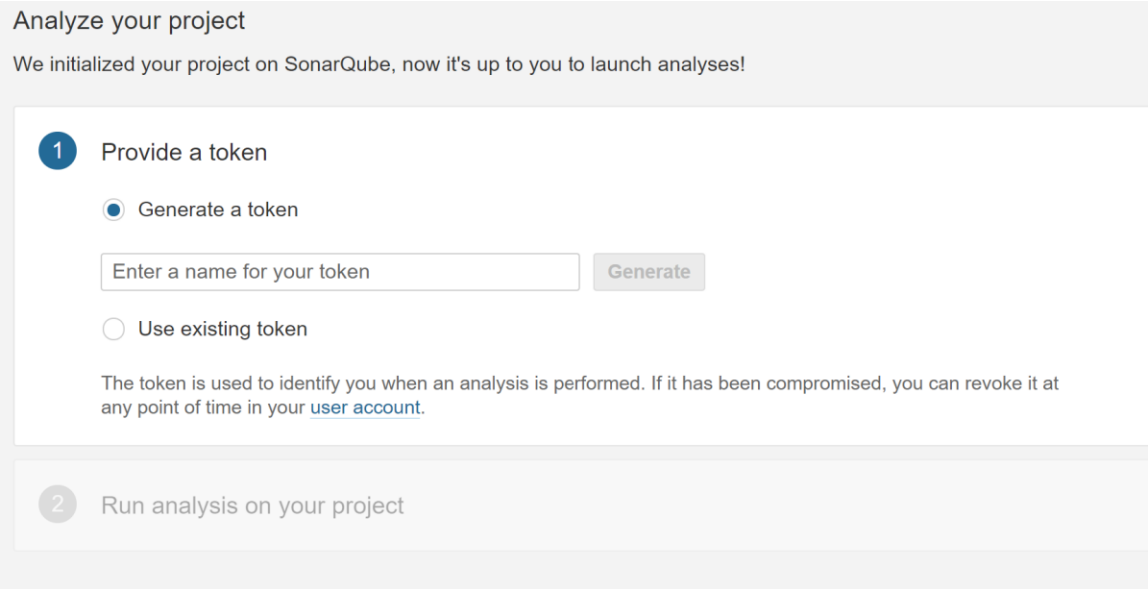
Step-9: Go to Google Chrome and type localhost:9000. The default password and username are admin and admin respectively.



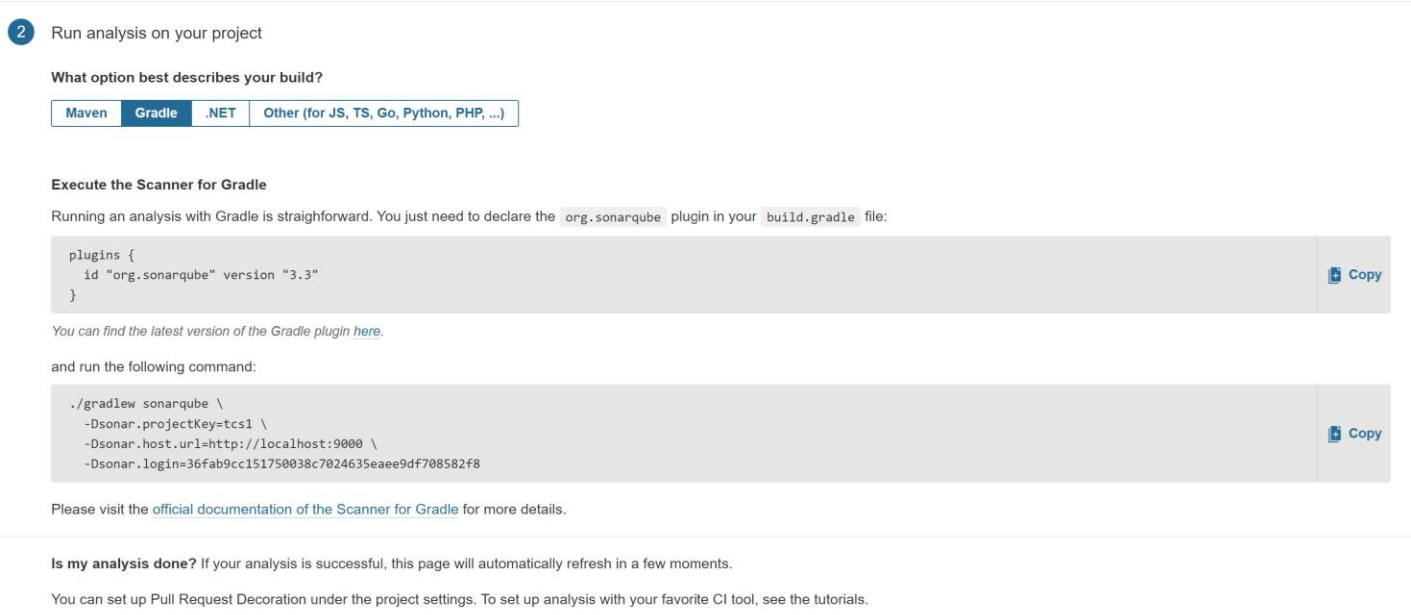
Step-10: Create a Project Name and Project Key. They both can be same.



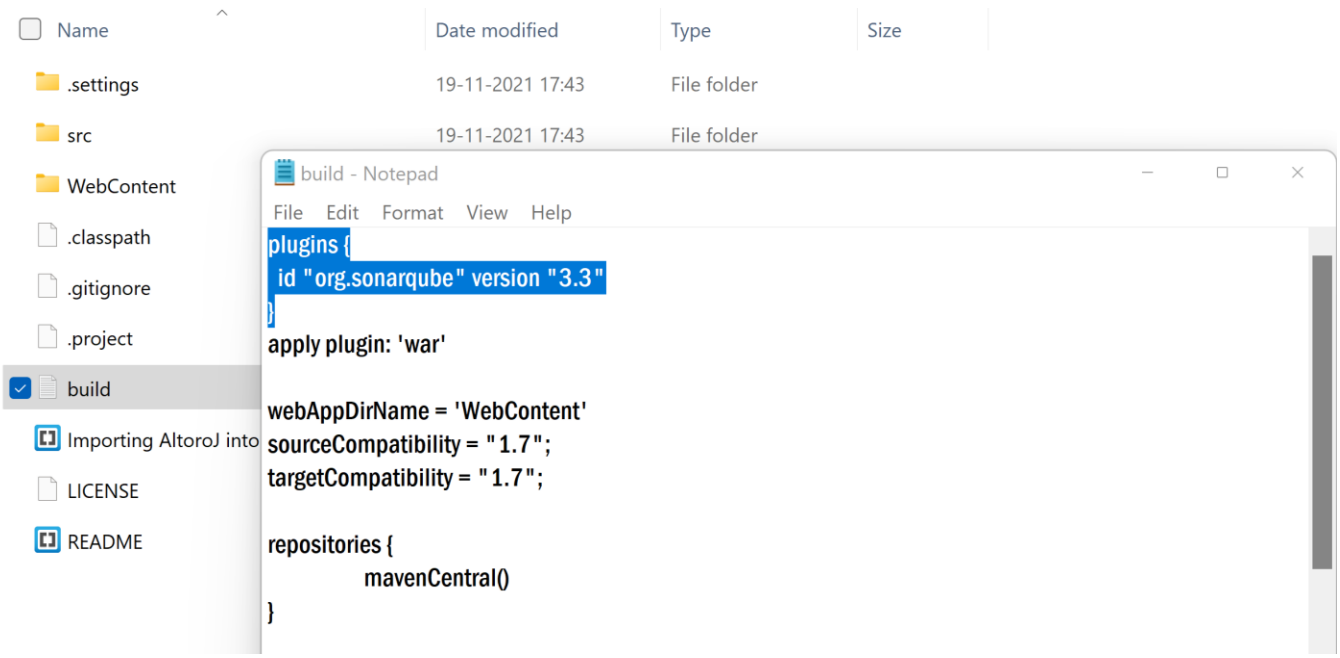
Step-11: Generate a token for your project.



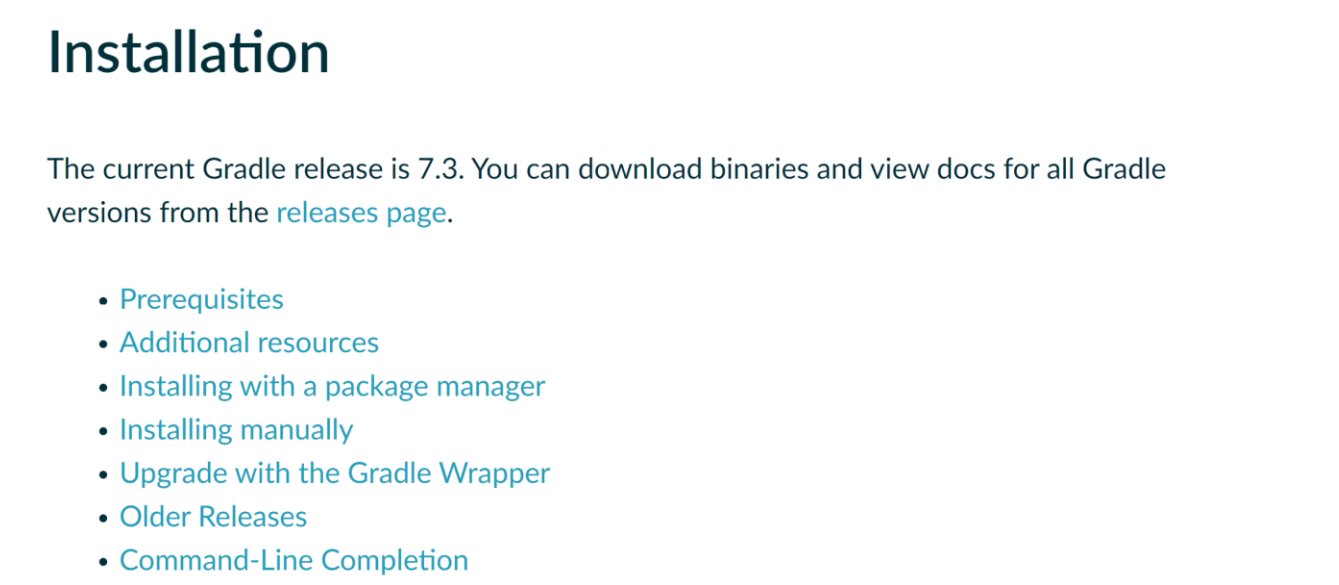
Step-12: Click on the Gradle option.



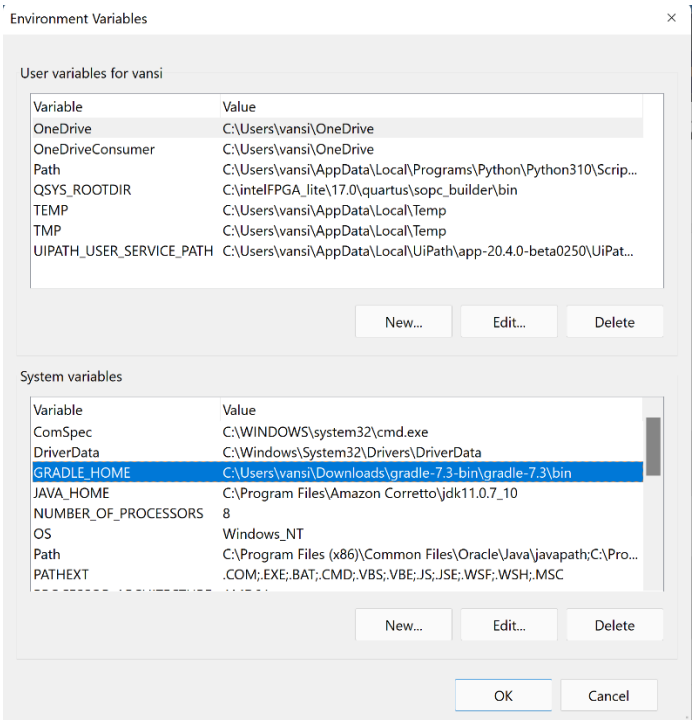
Step-13: Add the given plugin in the build.gradle file of the project.



Step-14: Go to https://gradle.org/install/ install Gradle if not already installed.



Step-15: Create a new system variable called GRADLE_HOME and add its path.



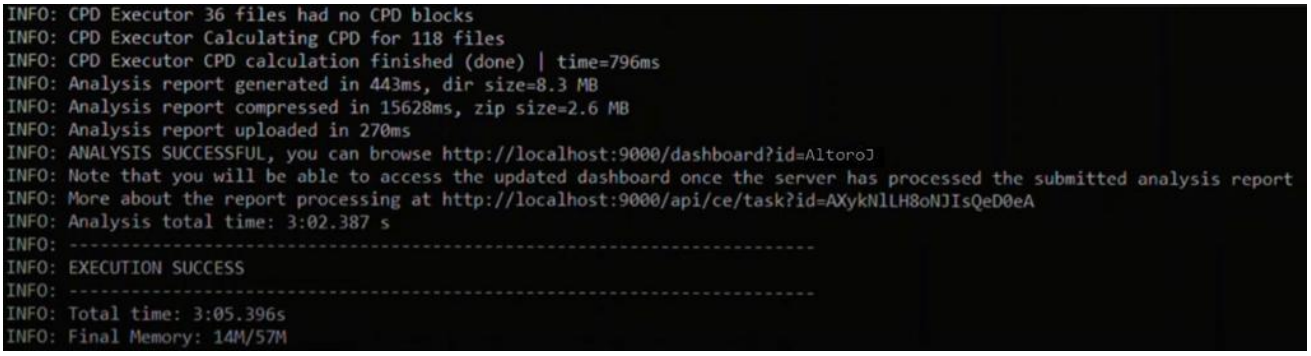
Step-16: Go to Command Prompt and check if Gradle was installed.

Step-17: Run the following command on Command Prompt.

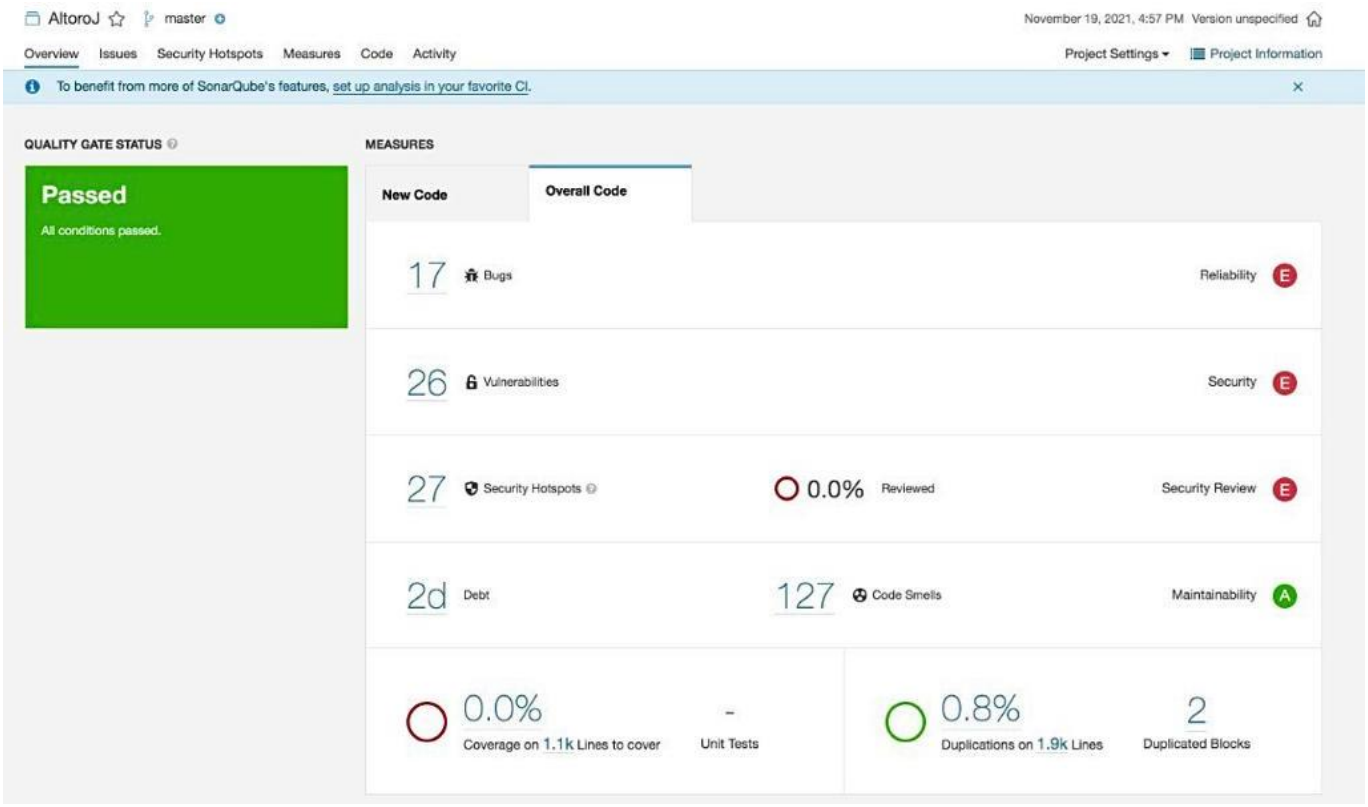
and run the following command:

```
./gradlew sonarqube \
  -Dsonar.projectKey=tcs1 \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.login=36fab9cc151750038c7024635eaee9df708582f8
```

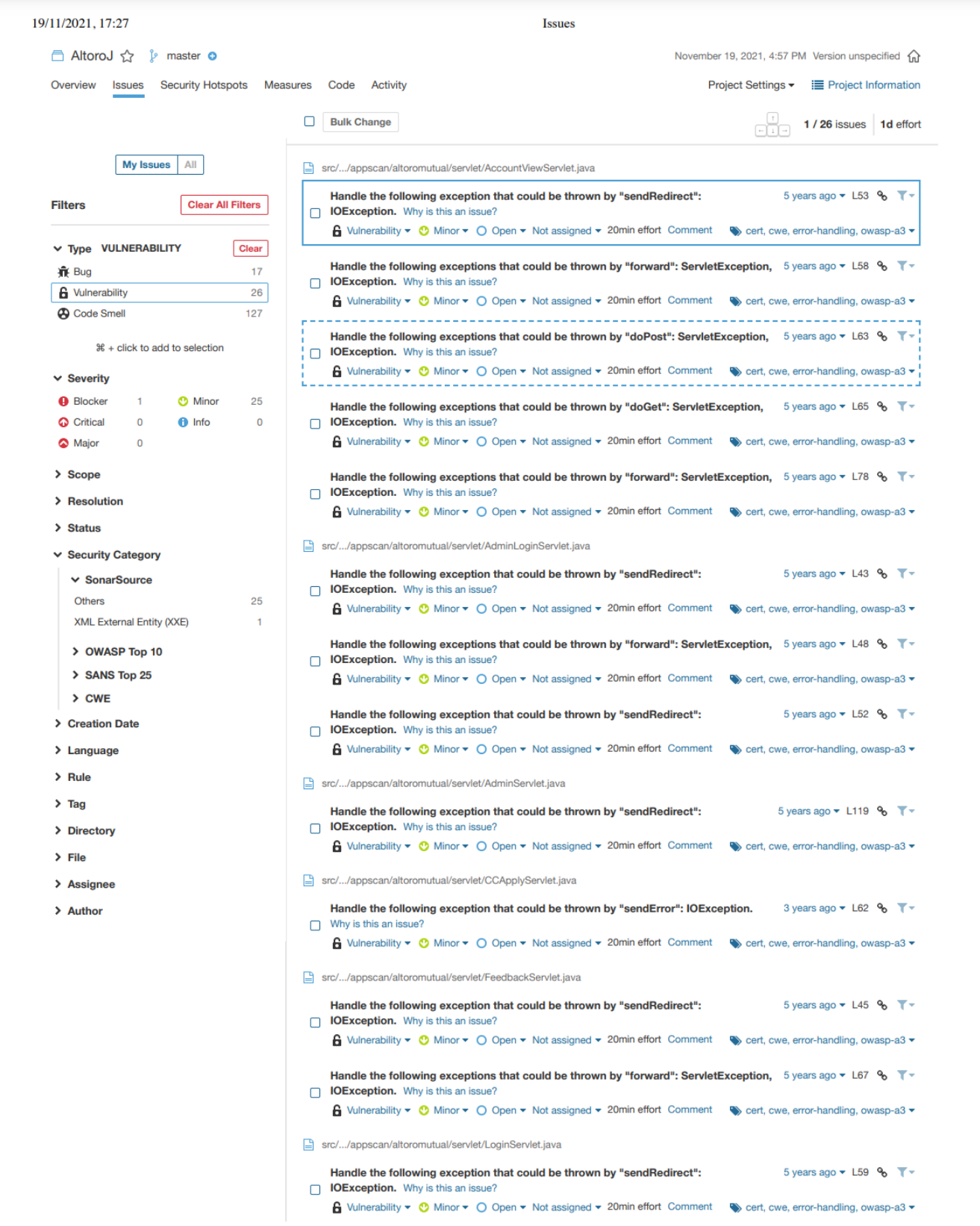
Please visit the official documentation of the Scanner for Gradle for more details.

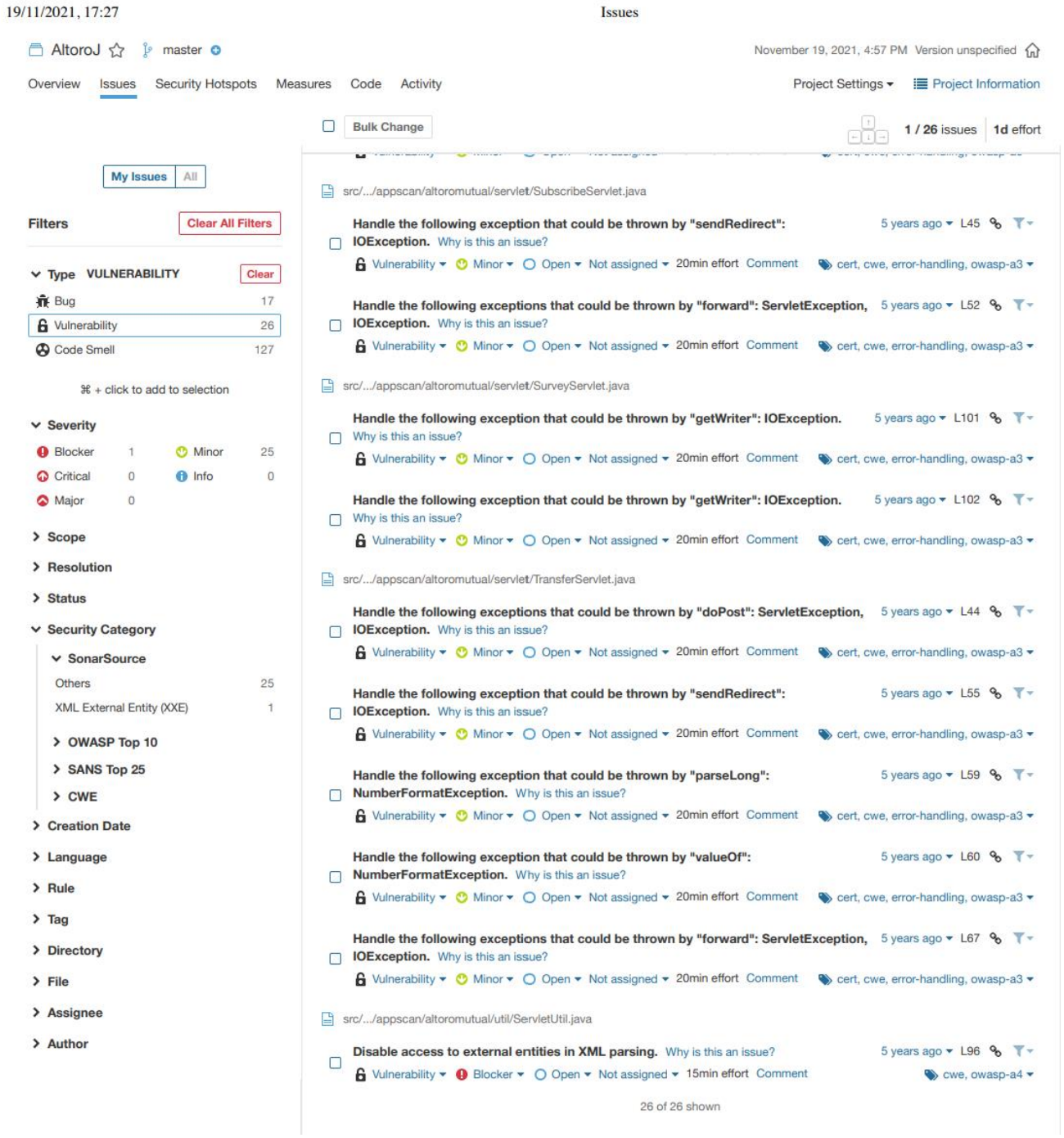


Step-18 Refresh the Page



Step-19: Click on Vulnerabilities.




19/11/2021, 17:27
Issues
AltoroJ
master
November 19, 2021, 4:57 PM Version unspecified
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
Bulk Change
1 / 26 issues
1d effort
My Issues
All
Filters
Clear All Filters
Type VULNERABILITY
Clear
Bug 17
Vulnerability 26
Code Smell 127
⌘ + click to add to selection
Severity
Blocker 1
Minor 25
Critical 0
Info 0
Major 0
Scope
Resolution
Status
Security Category
SonarSource
Others 25
XML External Entity (XXE) 1
OWASP Top 10
SANS Top 25
CWE
Creation Date
Language
Rule
Tag
Directory
File
Assignee
Author
src/.../appscan/altoromutual/servlet/SubscribeServlet.java
Handle the following exception that could be thrown by "sendRedirect": IOException. Why is this an issue?
5 years ago L45
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
Handle the following exceptions that could be thrown by "forward": ServletException, IOException. Why is this an issue?
5 years ago L52
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
src/.../appscan/altoromutual/servlet/SurveyServlet.java
Handle the following exception that could be thrown by "getWriter": IOException.
5 years ago L101
Why is this an issue?
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
Handle the following exception that could be thrown by "getWriter": IOException.
5 years ago L102
Why is this an issue?
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
src/.../appscan/altoromutual/servlet/TransferServlet.java
Handle the following exceptions that could be thrown by "doPost": ServletException, IOException. Why is this an issue?
5 years ago L44
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
Handle the following exception that could be thrown by "sendRedirect": IOException. Why is this an issue?
5 years ago L55
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
Handle the following exception that could be thrown by "parseLong": NumberFormatException. Why is this an issue?
5 years ago L59
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
Handle the following exception that could be thrown by "valueOf": NumberFormatException. Why is this an issue?
5 years ago L60
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
Handle the following exceptions that could be thrown by "forward": ServletException, IOException. Why is this an issue?
5 years ago L67
Vulnerability Minor Open Not assigned 20min effort Comment
cert, cwe, error-handling, owasp-a3
src/.../appscan/altoromutual/util/ServletUtil.java
Disable access to external entities in XML parsing. Why is this an issue?
5 years ago L96
Vulnerability Blocker Open Not assigned 15min effort Comment
cwe, owasp-a4
26 of 26 shown

Approach/ Methodology

Static Application Security Testing (SAST), also known as static analysis, is a testing procedure that examines source code for security flaws that render your company's apps vulnerable to attack. Before the code is compiled, SAST examines the application. White box testing is another name for it.

Programmers are prone to making little errors, such as omitting a semicolon here, adding an additional parenthesis there, and so on. Most of the time, these errors are minor; the compiler notifies the programmer, who patches the code, and the development process persists. Most security vulnerabilities, on the other hand, might lay dormant for an extended period of time before being discovered, thus this rapid cycle of input and response does not pertain to them.

I conducted manual reviewing of the source code to find vulnerabilities. Manual reviewing or auditing is a time-consuming method of static analysis, and in order to execute it properly, human code auditors must first understand what types of problems they should look for before thoroughly examining the code.

Initially, I focused on the website's authentication system and attempted to determine how secure it is. To guard against authentication-related threats, it's important to confirm the user's identity, authenticate them, and manage their sessions.

Authentication flaws may exist if the application:

- Enables for automated cyberattacks like credential stuffing, in which an attacker has a list of legitimate users and passwords.
- Enables for automated cyberattacks such as brute force.
- Allows passwords like "Password1" or "admin/admin" that are default, weak, or well-known.
- If the password field's autocomplete HTML feature is not deactivated.
- Uses insecure or inadequate credential recovery and forgotten-password procedures, such as "knowledge-based responses" that can't be made secure.

I checked if the site was susceptible to Brute Force Attacks. A brute force cyberattack is a way of determining an unknown value by running a large number of potential values through an automated procedure. The exploit takes advantage of the fact that the values' entropy is lower than it appears. While an 8-character alphanumeric password has a potential value of 2.8 trillion, many individuals will choose their passwords from a

considerably narrower selection of common phrases and concepts. There should be an account locked enforced after a sufficient amount failed login attempt tries.

I evaluated if the website was susceptible to any information leakage threats. Information Leakage is an application vulnerability in which sensitive data, such as web application technical information, environment, or user-specific data, is revealed. A perpetrator might utilize sensitive data to gain access to the intended web application, its hosting network, or its consumers. As a result, if feasible, sensitive data loss should be avoided or mitigated. In its most basic form, information leakage is caused by one or more of the following factors: Failure to scrub away HTML/Script comments containing sensitive data, incorrect application or server setups, or variations in page replies for valid and invalid data.

Next, I examined the code for phishing attacks. Phishing is a sort of social engineering assault that is frequently used to obtain sensitive information from users, such as login passwords and credit card details. It happens when a hacker poses as a trustworthy entity and convinces a victim to open an email, instant text, or message. The receiver is subsequently duped into visiting a malicious link, which can result in malware insertion in the system, system freeze as part of a ransomware cyberattack, or the disclosure of sensitive information.

Detailed Explanation of the Vulnerabilities Found (Part-1)

1. **Autocomplete HTML Attribute Not Disabled for Password Field**

   Location of the vulnerability: https://demo.testfire.net/login.jsp

   File name: login.jsp

   Line number: 62

   URL to the file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/WebContent/login.jsp

Explanation of the vulnerability: This vulnerability possesses a threat of information leakage. The website's authentication mechanism may be very uncomplicated to bypass and exploit the user's personal information. This vulnerability is caused due to issues in programming or configuration of the website.

As it can be seen the password field does not impose any disabling of the autocomplete feature. The HTML5 standard has established the "autocomplete" feature. According to the W3C's website, the attribute has two states: "on" and "off," and that leaving it blank is equal to setting it to "on."
This page is insecure because the "autocomplete" property for the "password" field in the "input" element is not set to "off."
This might allow an unauthorized user (with local access to an authorized client) to log in to the site by autofilling the username and password boxes.

```
    <td>
      <input type="text" id="uid" name="uid" value="" style="width: 150px;">
    </td>
    <td>
    </td>
  </tr>
  <tr>
    <td>
      Password:
    </td>
    <td>
      <input type="password" id="passw" name="passw" style="width: 150px;">
      </td>
  </tr>
  <tr>
      <td></td>
      <td>
        <input type="submit" name="btnSubmit" value="Login">
      </td>
    </tr>
```

Fix to this error: Although the HTML autocomplete function enhances the user experience, it also poses security vulnerabilities. The autocomplete function, while intended to make the user experience easier, can become a risk if hackers get significant exposure to the user's computers or browsers. Software developers can use the HTML autocomplete attribute to regulate the autocomplete capability and avoid situations like this. It operates by letting websites to manage whether

sensitive data supplied in forms and input elements is kept in the browser cache of the user.

Example of a vulnerable site:

```
<form action="AppScan.html" method="get"> Username: <input type="text" name="firstname" /><br
    /> Password: <input type="password" name="lastname" /> <input type="submit" value="Submit"
    /> <form>
```

Example of a non-vulnerable site:

```
<form action="AppScan.html" method="get"> Username: <input type="text" name="firstname" /><br
    /> Password: <input type="password" name="lastname" autocomplete="off"/> <input type
    ="submit" value="Submit" /> <form>
```

2. Inadequate Account Lockout

Location of the vulnerability: https://demo.testfire.net/index.jsp

Explanation of the vulnerability: Brute force password guessing cyberattacks are mitigated by account lockout methods. Accounts are often locked after 3 to 5 failed login attempts and may only be restored after a specified amount of time, by a self-service release mechanism, or with administrator involvement. But in this web application, even after 10-15 failed login attempts, it lets the user login with the correct password.

Fix to this error: The developers should decide on the number of login attempts (3-5 maximum), and they should implement a complete lockout from the website to prevent hackers from conducting any brute force attacks.
To minimize unwanted support calls from legitimate customers who have been locked out of their account and need it reactivated, account activity can be temporarily suspended and then reactivated after a certain amount of time.

Generally, locking the account for five to ten minutes or so is enough to prevent brute force cyberattacks.

3. Phishing through URL Redirection

Location of the vulnerability: https://demo.testfire.net/login.jsp

File name: customize.jsp

URL to the file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/WebContent/bank/customize.jsp

Explanation of the vulnerability: URL redirectors are a typical feature of websites that allows them to redirect incoming requests to a different resource. This may be performed for a multitude of purposes, but it's most commonly done to allow resources to be relocated within the directory structure while retaining compatibility for consumers who requested the resource at its former position. URL redirectors can also be utilized for load balancing, logging outbound links, and exploiting shortened URLs.

It is conceivable to mislead a gullible consumer to provide sensitive data such as login, password, credit card number, social security number, and so on.
Phishing is a social engineering approach in which an adversary poses as a genuine company with which the victim may conduct transaction in order to get the consumer to divulge personal information (usually authentication credentials) that the adversary may later utilize. Phishing is simply a method of acquiring information or "fishing" for it.

A URL value was discovered in a http parameter, which caused the web application to redirect the request to the provided URL. An adversary can successfully conduct a phishing scheme and acquire user credentials by changing the URL value to a fraudulent site.
The perpetrator benefits from the fact that the changed link's server name is comparable to the original site's, giving his phishing attempts a more trustworthy image.

```
<div id="wrapper" style="width: 99%;">
        <jsp:include page="membertoc.jspf"/>
    <td valign="top" colspan="3" class="bb">
                <div class="fl" style="width: 99%;">

                <%
                        String content = request.getParameter("content");
                        if (content != null && !content.equalsIgnoreCase("customize.jsp")){
                                if (content.startsWith("http://") || content.startsWith("https://")){
                                        response.sendRedirect(content);
                                }
                        }
                %>
```

Fix to this error: Prevent external site redirection depending on parameter values.

4. Email Address Pattern Found

Location of the Vulnerability: https://demo.testfire.net/swagger/swagger-ui-bundle.js

File name: swagger-ui-bundle.js

Line number: 10 & 27

URL to the file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/WebContent/swagger/swagger-ui-bundle.js

Explanation of the vulnerability: It is conceivable to collect sensitive online application information such as users, passwords, machine names, and/or sensitive file locations. Spambots scour the internet for e-mail addresses in order to compile mailing lists from which they may send unwanted e-mail (spam). Furthermore, the e-mail addresses discovered may be personal and hence should not be made public.

```
 * @author   Feross Aboukhadijeh <feross@feross.org> <http://feross.org>
 * @license  MIT
 */
var r=n(529),o=n(530),i=n(261);function a(){return s.TYPED_ARRAY_SUPPORT?2
/*!
  Copyright (c) 2016 Jed Watson.
  Licensed under the MIT License (MIT), see
  http://jedwatson.github.io/classnames
*/
/*!
```

```
/*!
 * @description Recursive object extending
 * @author Viacheslav Lotsmanov <lotsmanov89@gmail.com>
 * @license MIT
 *
 * The MIT License (MIT)
 *
 * Copyright (c) 2013-2018 Viacheslav Lotsmanov
 *
```

Fix to this error: Do not leave personal email addresses and personal information in the source code/ comments of the source code. It is very uncomplicated for the hacker to access these for malicious activities.

5. Client-Side (JavaScript) Cookie References

Location of the Vulnerability: AltoroJ/WebContent/swagger/swagger-ui-bundle.js

File name: swagger-ui-bundle.js

Line number: 70

URL to the file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/WebContent/swagger/swagger-ui-bundle.js

```
P.headers.Cookie=U}return P.cookies&&delete P.cookies,(0,m.mergeInQueryOrForm)(P),P}function
```

```
;t.headers.Cookie=u+(0,a.default)({key:n.name,value:r,escape:!1,style:n.style||"form",explode:void
```

Explanation of the vulnerability: The worst-case scenario for this attack depends on the context and role of the cookies that are created at the client side. When the server relies on protection mechanisms placed on the client side, an attacker can modify the client-side behavior to bypass the protection mechanisms resulting in potentially unexpected interactions between the client and server. The consequences will vary, depending on what the mechanisms are trying to protect. A cookie is a piece of information usually created by the Web server and stored in the Web browser. The cookie contains information used by web applications mainly (but not only) to identify users and maintain their state. It can be seen that the JavaScript code at the client side is used to manipulate (either create or modify) the site's cookies. It is possible for an attacker to view this code, understand its logic, and use it to compose his own cookies, or modify existing ones,
based on this knowledge. The damage an attacker may cause depends on how the application uses its cookies, or what information it stores in them. Among other things, cookie manipulation may lead to session hijacking or privilege escalation. Other vulnerabilities caused by cookie poisoning contain SQL injection and Cross-Site scripting.

Fix to this error: Avoid placing business/security logic at the client side. Find and remove insecure client-side JavaScript code which may pose a security threat to the site.

6. Detection of Weak Password on Target Web Application Form

Location of the Vulnerability: http://demo.testfire.net/bank/login.aspx

File name: login.jsp

URL to the file: http://demo.testfire.net/bank/login.aspx

Explanation of the vulnerability: This web application allows its users to keep a weak password. A weak password is short, common, a system default, or something that could be rapidly guessed by executing a brute force attack using a subset of all possible passwords, such as words in the dictionary, proper names, words based on the username or common variations on these themes. Depending on the nature of the password-protected resource, an attacker can mount one or more of the following types of attacks: Access the contents of the password-protected resources; Access password-protected administrative mechanisms such as "dashboard", "management console" and "admin panel," potentially progressing to gain full control of the application.

Fix to this Error: Restrict login failures to lock the account when certain amount of login failures has reached. Increase the complexity of the password. The user-registered password must meet the intended complexity requirements. For example, the password must be a combination of no less than 8 characters of the following: lowercase letters, uppercase letters, digits, and special characters ~!@#$%^&*()-+_. For users with unqualified passwords, the registration should not be allowed. Users are advised to change the default passwords occurring during installation to make them satisfy the preceding requirements.

7. Clickjacking: X-Frame-Options Not Configured

Location of the Vulnerability: http://demo.testfire.net/

URL to the file: http://demo.testfire.net/

Explanation of the vulnerability: There are invalid links on the page direct to resources that do not exist. Clickjacking is a kind of visual deceptive means. Attackers can put a transparent and invisible iframe into a web page and trick an innocent user into clicking the transparent iframe page. By adjusting the position of the iframe page, attackers can tempt users to click some function buttons on the iframe page. The X-Frame-Options field in the HTTP response header specifies whether or not a browser should render a page in an <iframe> tag. If X-Frame-Options is absent from the header of the response from the server, the website is vulnerable to Clickjacking attacks. By setting X-FrameOptions, you can prevent your website pages from being embedded with other pages, thereby eliminating the possibility of Clickjacking attacks.

Fix to this Error: Modify the web server configuration by adding the X-Frame-Options response header. It has three values:

1. DENY: The page can never be embedded into a <frame> or <iframe> tag.
2. SAMEORIGIN: The page can be embedded into an <iframe> or <frame> tag on the same origin as the page itself.
3. ALLOW-FROM uri: The page can be embedded into a frame on the specified origin.

For example: The http.conf file can be configured on the Apache:

```
<IfModule headers_module>
Header always append X-Frame-Options "DENY"
</IfModule>
```

Detailed Explanation of the Vulnerabilities Found (SonarQube Analysis Report) of http://demo.testfire.net/

1. Improper exception handling (Minor)

Handle the following exception that could be thrown by "sendRedirect": IOException.

1) Error-1:

File name: AccountViewServlet.java

Line Number: 53

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/AccountViewServlet.java

Code Snippet:

```
18  package com.ibm.security.appscan.altoromutual.servlet;
19
20  import java.io.IOException;
21
22  import javax.servlet.RequestDispatcher;
23  import javax.servlet.ServletException;
24  import javax.servlet.http.HttpServlet;
25  import javax.servlet.http.HttpServletRequest;
26  import javax.servlet.http.HttpServletResponse;
27
28
29  /**
30   * This servlet allows the users to view account and transaction information.
31   * Servlet implementation class AccountServlet
32   * @author Alexei
33   *
34   */
35  public class AccountViewServlet extends HttpServlet {
36      private static final long serialVersionUID = 1L;
37
38      /**
39       * @see HttpServlet#HttpServlet()
40       */
41      public AccountViewServlet() {
42          super();
43      }
44
45      /**
46       * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
    response)
47       */
48      protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
49          //show account balance for a particular account
50          if (request.getRequestURL().toString().endsWith("showAccount")){
51              String accountName = request.getParameter("listAccounts");
52              if (accountName == null){
53                  response.sendRedirect(request.getContextPath()+"/bank/main.jsp");
```

Handle the following exception that could be thrown by "sendRedirect": IOException. Why is this an issue?

5 years ago ▾ L53

Vulnerability ▾ Minor ▾ Open ▾ Not assigned ▾ 20min effort Comment

cert, cwe, error-handling, owasp-a3 ▾

2) Error-2:

File name: AdminLoginServlet.java

Line Number: 43

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/AdminLoginServlet.java

Code Snippet:

```
18          package com.ibm.security.appscan.altoromutual.servlet;
19
20          import java.io.IOException;
21
22          import javax.servlet.RequestDispatcher;
23          import javax.servlet.ServletException;
24          import javax.servlet.http.HttpServlet;
25          import javax.servlet.http.HttpServletRequest;
26          import javax.servlet.http.HttpServletResponse;
27
28          import com.ibm.security.appscan.altoromutual.util.ServletUtil;
29
30          /**
31           * Administrator login servlet
32           * @author Alexei
33           */
34          public class AdminLoginServlet extends HttpServlet {
35              private static final long serialVersionUID = 1L;
36
37              /**
38               * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
            response)
39               */
40              protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
41                  String password = request.getParameter("password");
42                  if (password == null){
43                      response.sendRedirect(request.getContextPath()+"/admin/login.jsp");
```

Handle the following exception that could be thrown by "sendRedirect": IOException. Why is this an issue?

5 years ago ▾ L43

Vulnerability ▾ Minor ▾ Open ▾ Not assigned ▾ 20min effort Comment

cert, cwe, error-handling, owasp-a3 ▾

3) Error-3:

File name: AdminLoginServlet.java

Line Number: 52

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/AdminLoginServlet.java

Code Snippet:

```
49                      return;
50                  } else {
51                      request.getSession(true).setAttribute(ServletUtil.SESSION_ATTR_ADMIN_KEY,
            ServletUtil.SESSION_ATTR_ADMIN_VALUE);
52                      response.sendRedirect(request.getContextPath()+"/admin/admin.jsp");
```

Handle the following exception that could be thrown by "sendRedirect": IOException. Why is this an issue? 5 years ago L52

Vulnerability Minor Open Not assigned 20min effort Comment

cert, cwe, error-handling, owasp-a3

Explanation of the Vulnerability: An HTTP redirect is when a webserver returns an HTTP status code like 302 to tell the browser to make a request to a different URL. Redirects can also be performed in client-side JavaScript code by updating the browser URL directly. Redirects are commonly used to push a user to an authentication page before viewing some protected content. In this scenario, the user will often be redirected again back to the original resource once they have successfully logged in.
If a redirect URL is pulled from a preceding HTTP request, you need to check the URL is safe before redirecting the user. Typically, this means checking the URL is a relative URL to a resource hosted under your web domain. Open redirects - which allow a maliciously crafted link to redirect the user to arbitrary third-party domains - are often used by spammers to disguise harmful links in emails.

Fix to this Error: Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.
Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.
Many open redirect problems occur because the programmer assumed that certain inputs could not be modified, such as cookies and hidden form fields.

When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."

2. Improper exception handling (Minor)

Handle the following exceptions that could be thrown by "forward": ServletException, IOException.

1) Error-1:

File name: AccountViewServlet.java

Line Number: 58

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/AccountViewServlet.java

Code Snippet:

```
54                      return;
55                  }
56          //          response.sendRedirect("/bank/balance.jsp&acctId=" + accountName);
57                  RequestDispatcher dispatcher =
            request.getRequestDispatcher("/bank/balance.jsp?acctId=" + accountName);
58                  dispatcher.forward(request, response);
```

Handle the following exceptions that could be thrown by "forward": ServletException, IOException. Why is this an issue? 5 years ago L58

Vulnerability Minor Open Not assigned 20min effort Comment

cert, cwe, error-handling, owasp-a3

```
59                  return;
```

2) **Error-2:**

File name: AccountViewServlet.java

Line Number: 78

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/AccountViewServlet.java

Code Snippet:

```
71          protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
72              //show transactions within the specified date range (if any)
73              if (request.getRequestURL().toString().endsWith("showTransactions")){
74                  String startTime = request.getParameter("startDate");
75                  String endTime = request.getParameter("endDate");
76
77                  RequestDispatcher dispatcher =
        request.getRequestDispatcher("/bank/transaction.jsp?" +
        ((startTime!=null)?"&startTime="+startTime:"") +
        ((endTime!=null)?"&endTime="+endTime:""));
78                  dispatcher.forward(request, response);
```

Handle the following exceptions that could be thrown by "forward": ServletException, IOException. Why is this an issue?

5 years ago ▾ L78

Vulnerability ▾ Minor ▾ Open ▾ Not assigned ▾ 20min effort Comment

cert, cwe, error-handling, owasp-a3 ▾

3) **Error-3:**

File name: AdminLoginServlet.java

Line Number: 48

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-

[3.2/src/com/ibm/security/appscan/altoromutual/servlet/AdminLoginServlet.java]

Code Snippet:

Explanation of the Vulnerability: Unvalidated redirects, and forwards are possible when a web application accepts untrusted input that could cause the web application to redirect the request to a URL contained within untrusted input. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials.

Because the server's name in the modified link is identical to the original site, phishing attempts may have a more trustworthy appearance. Unvalidated redirect and forward attacks can also be used to maliciously craft a URL that would pass the application's access control check and then forward the attacker to privileged functions that they would normally not be able to access.

Fix to this Error: Safe use of redirects and forwards can be done in a number of ways:

- Simply avoid using redirects and forwards.
- If used, do not allow the URL as user input for the destination.
- Where possible, have the user provide short name, ID or token which is mapped server-side to a full target URL.
- This provides the highest degree of protection against the attack tampering with the URL.
- Be careful that this doesn't introduce an enumeration vulnerability where a user could cycle through IDs to find all possible redirect targets

- If user input can't be avoided, ensure that the supplied value is valid, appropriate for the application, and is authorized for the user.
- Sanitize input by creating a list of trusted URLs (lists of hosts or a regex).
- This should be based on an allow-list approach, rather than a block list.
- Force all redirects to first go through a page notifying users that they are going off your site, with the destination clearly displayed, and have them click a link to confirm.

3. Improper exception handling (Minor)

Handle the following exceptions that could be thrown by "doGet": ServletException, IOException.
File name: AccountViewServlet.java

Line Number: 65

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/AccountViewServlet.java

Code Snippet:

```
64                  else
65                      super.doGet(request, response);
```

Handle the following exceptions that could be thrown by "doGet": ServletException, IOException. Why is this an issue?

5 years ago ▾ L65

Vulnerability ▾ Minor ▾ Open ▾ Not assigned ▾ 20min effort Comment

cert, cwe, error-handling, owasp-a3 ▾

```
66              }
```

Explanation of the Vulnerability: Even though the signatures for methods in a servlet include throws IOException, ServletException, it's a bad idea to let such exceptions be thrown. Failure to catch exceptions in a servlet could leave a system in a vulnerable state, possibly resulting in denial-of-service attacks, or the exposure of sensitive information because when a servlet throws an exception, the servlet container typically sends debugging information back to the user. And that information could be very valuable to an attacker. This rule applies to server-side

applications as well as to clients. Attackers can glean sensitive information not only from vulnerable web servers but also from victims who use vulnerable web browsers.

Fix to this Vulnerability: Ensure that the site is built to gracefully handle all possible errors. When errors occur, the site should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details. Certain classes of errors should be logged to help detect implementation flaws in the site and/or hacking attempts. Very few sites have any intrusion detection capabilities in their web application, but it is certainly conceivable that a web application could track repeated failed attempts and generate alerts. Note that the vast majority of web application attacks are never detected because, so few sites have the capability to detect them. Therefore, the prevalence of web application security attacks is likely to be seriously underestimated.

4. Improper exception handling (Minor)

Handle the following exception that could be thrown by "sendError": IOException.

1) Error-1:

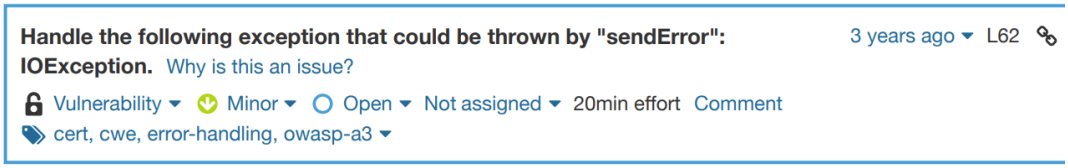
File name: CCApplyServlet.java

Line Number: 62

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/CCApplyServlet.java

Code Snippet:

```
61                  } catch (Exception e) {
62                      response.sendError(500);
```



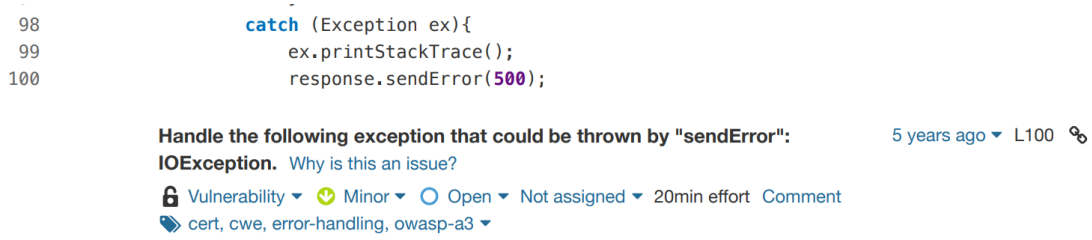
2) **Error-2:**

File name: LoginServlet.java

Line Number: 100

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/LoginServlet.java

Code Snippet:



Explanation of the Vulnerability: Improper handling of errors can introduce a variety of security problems for a web site. The most common problem is when detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user (hacker). These messages reveal implementation details that should never be revealed. Such details can provide hackers important clues on potential flaws in the site and such messages are also disturbing to normal users.

Web applications frequently generate error conditions during normal operation. Out of memory, null pointer exceptions, system call failure, database unavailable, network timeout, and hundreds of other common conditions can cause errors to be generated. These errors must be handled according to a well thought out scheme that will provide a meaningful error message to the user, diagnostic information to the site maintainers, and no useful information to an attacker.

Even when error messages don't provide a lot of detail, inconsistencies in such messages can still reveal important clues on how a site works, and what information is present under the covers. For example, when a user tries to access a file that does not exist, the error message typically indicates, "file not found". When accessing a file that the user is not authorized for, it indicates, "access denied". The user is not supposed to know the file even exists, but such inconsistencies will readily reveal the presence or absence of inaccessible files or the site's directory structure.

One common security problem caused by improper error handling is the fail-open security check. All security mechanisms should deny access until specifically granted, not grant access until denied, which is a common reason why fail open errors occur. Other errors can cause the system to crash or consume significant resources, effectively denying or reducing service to legitimate users.

Fix to this Error: Good error handling mechanisms should be able to handle any feasible set of inputs, while enforcing proper security. Simple error messages should be produced and logged so that their cause, whether an error in the site or a hacking attempt, can be reviewed. Error handling should not focus solely on input provided by the user but should also include any errors that can be generated by internal components such as system calls, database queries, or any other internal functions.
In the implementation, ensure that the site is built to gracefully handle all possible errors. When errors occur, the site should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details. Certain classes of errors should be logged to help detect implementation flaws in the site and/or hacking attempts. Very few sites have any intrusion detection capabilities in their web application, but it is certainly conceivable that a web application could track repeated failed attempts and generate alerts. Note that the vast majority of web application attacks are never detected because, so few sites have the capability to detect them. Therefore, the prevalence of web application security attacks is likely to be seriously underestimated.

5. Improper exception handling (Minor)

Handle the following exception that could be thrown by "getWriter": IOException.

1) **Error-1:**

File name: SurveyServlet.java

Line Number: 101

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/SurveyServlet.java

Code Snippet:

```
98                      request.getSession().setAttribute("surveyStep", step);
99                  }
100                 response.setContentType("text/html");
101                 response.getWriter().write(content);
```

Handle the following exception that could be thrown by "getWriter": IOException. Why is this an issue? 5 years ago L101

Vulnerability Minor Open Not assigned 20min effort Comment

cert, cwe, error-handling, owasp-a3

2) **Error-2:**

File name: SurveyServlet.java

Line Number: 102

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/SurveyServlet.java

Code Snippet:

```
102                 response.getWriter().flush();
```

Handle the following exception that could be thrown by "getWriter": IOException. Why is this an issue? 5 years ago L102

Vulnerability Minor Open Not assigned 20min effort Comment

cert, cwe, error-handling, owasp-a3

Explanation of the Vulnerability: Failure to catch exceptions in a servlet could leave a system in a vulnerable state, possibly resulting in denial-of-service attacks, or the exposure of sensitive information because when a servlet throws an exception, the servlet container typically sends debugging information back to the user. And that information could be very valuable to an attacker.

Fix to this Error: By throwing an exception in a servlet you expose the stacktrace and possible other sensitive information to the world. You should catch the exception and print/show a nice error message.

6. NumberFormatException (Minor)

Handle the following exception that could be thrown by "parseLong": NumberFormatException.

File name: TransferServlet.java

Line Number: 59

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/servlet/TransferServlet.java

Code Snippet:

```
56                      return ;
57                  }
58                  String accountIdString = request.getParameter("fromAccount");
59                  long creditActId = Long.parseLong(request.getParameter("toAccount"));
```

Handle the following exception that could be thrown by "parseLong": NumberFormatException. Why is this an issue? 5 years ago ▾ L59

Vulnerability ▾ Minor ▾ Open ▾ Not assigned ▾ 20min effort Comment

Explanation of this Vulnerability: The NumberFormatException occurs when an attempt is made to convert a string with improper format into a numeric value. That means, when it is not possible to convert a string in any numeric type (float, int, etc.), this exception is thrown. It is a Runtime Exception (Unchecked Exception) in Java. It is a subclass of IllegalArgumentException class. To handle this exception, try–catch block can be used.

While operating upon strings, there are times when we need to convert a number represented as a string into an integer type. The method generally used to convert String to Integer in Java is parseInt().

Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

7. Disable access to external entities in xml parsing. (Critical)

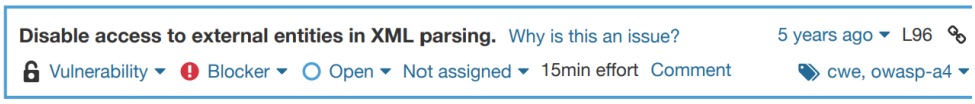
File name: ServletUtil.java

Line Number: 96

URL to the Erroneous file: https://github.com/HCL-TECH-SOFTWARE/AltoroJ/blob/AltoroJ-3.2/src/com/ibm/security/appscan/altoromutual/util/ServletUtil.java

Code Snippet:

```
89              public static String[] searchArticles(String query, String path) {
90                  ArrayList<String> results = new ArrayList<String>();
91
92                  File file = new File(path);
93                  Document document;
94                  try {
95                      document = DocumentBuilderFactory.newInstance()
96                              .newDocumentBuilder().parse(file);
```



Explanation of this Vulnerability: The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output. XML documents optionally contain a Document Type Definition (DTD), which, among other features, enables the definition of XML entities. It is possible to define an entity by providing a substitution string in the form of a URI. The XML parser can access the contents of this URI and embed these contents back into the XML document for further processing.
By submitting an XML file that defines an external entity with a file:// URI, an attacker can cause the processing application to read the contents of a local file. This attack may lead to the disclosure of confidential data, denial of service, Server-Side Request Forgery (SSRF), port scanning from the perspective of the machine where the parser is located, and other system impacts. The following guide provides concise information to prevent this vulnerability.

Fix to This Error: Since the entire XML document is communicated from an untrusted client, it is not usually possible to selectively validate or escape tainted data within the system identifier in the DTD. Therefore, the XML processor should be configured to use a local static DTD and disallow any declared DTD included in the XML document.
The safest way to prevent XXE is always to disable DTDs (External Entities) completely. Disabling DTDs also makes the parser secure against denial of services (DOS) attacks such as Billion Laughs. If it is not possible to disable DTDs completely, then external entities and external document type declarations must be disabled in the way that's specific to each parser.

Detailed Explanation of the Vulnerabilities Found (SonarQube Analysis Report) of WebGoat

https://www.vulnspy.com/webgoat-8/

1. Don't use the default "PasswordEncoder" relying on plain text (Critical)

File name: WebSecurityConfig.java

Line Number: 80

URL to the Erroneous file:
https://github.com/WebGoat/WebGoat/blob/develop/webgoat-container/src/main/java/org/owasp/webgoat/WebSecurityConfig.java

Code Snippet:

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    ExpressionUrlAuthorizationConfigurer<HttpSecurity>.ExpressionInterceptUrlRegistry security = http
            .authorizeRequests()
            .antMatchers("/css/**", "/images/**", "/js/**", "fonts/**", "/plugins/**", "/registration", "/register.mvc").permitAll()
            .anyRequest().authenticated();
    security.and()
            .formLogin()
            .loginPage("/login")
            .defaultSuccessUrl("/welcome.mvc", true)
            .usernameParameter("username")
            .passwordParameter("password")
            .permitAll();
    security.and()
            .logout().deleteCookies("JSESSIONID").invalidateHttpSession(true);
    security.and().csrf().disable();

    http.headers().cacheControl().disable();
    http.exceptionHandling().authenticationEntryPoint(new AjaxAuthenticationEntryPoint("/login"));
}

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
  auth.userDetailsService(userDetailsService); //.passwordEncoder(bCryptPasswordEncoder());
```

Don't use the default "PasswordEncoder" relying on plain-text. Why is this an issue? 5 years ago L80
Vulnerability Critical Open Not assigned 30min effort No tags

```
}
```

Explanation of the Vulnerability: Password management issues occur when a password is stored in plaintext in an application's properties, configuration file, or memory. Storing a plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource. In some contexts, even storage of a plaintext password in memory is considered a security risk if the password is not cleared immediately after it is used. A user password should never be stored in clear text, instead a hash should be produced from it using a secure algorithm:

- not vulnerable to brute force attacks.
- not vulnerable to collision attacks
- and a salt should be added to the password to lower the risk of rainbow table attacks.

This rule raises an issue when a password is stored in clear-text or with a hash algorithm vulnerable to brute force attacks. These algorithms, like md5 or SHA-family functions are fast to compute the hash and therefore brute force attacks are possible (it's easier to exhaust the entire space of all possible passwords) especially with hardware like GPU, FPGA or ASIC. Modern password hashing algorithms such as bcrypt, PBKDF2 or argon2 are recommended.

Fix to this Vulnerability: Avoid storing passwords in easily accessible locations. Consider storing cryptographic hashes of passwords as an alternative to storing in plaintext.
A programmer might attempt to remedy the password management problem by obscuring the password with an encoding function, such as base 64 encoding, but this effort does not adequately protect the password because the encoding can be detected and decoded easily.

2. Change this code to not construct SQL queries directly from user-controlled data.

File name: Assignment5.java

Line Number: 60

URL to the Erroneous file:
https://github.com/WebGoat/WebGoat/blob/develop/webgoat-

[lessons/challenge/src/main/java/org/owasp/webgoat/challenges/challenge5/Assignment5.java]

Code Snippet:

```
public AttackResult login(@RequestParam String username_login,     @RequestParam String     password_login) throws Exception {
    if (!StringUtils.hasText(username_login) || !StringUtils.hasText(password_login)) {
        return failed(this).feedback("required4").build();
    }
    if (!"Larry".equals(username_login)) {
        return failed(this).feedback("user.not.larry").feedbackArgs(username_login).build();
    }
    try (var connection = dataSource.getConnection()) {
        PreparedStatement statement =     connection.prepareStatement(    "select password from challenge_users where userid = '"
        ResultSet resultSet =     statement.executeQuery();
```


```
        if (resultSet.next()) {
            return success(this).feedback("challenge.solved").feedbackArgs(Flag.FLAGS.get(5)).build();
        } else {
            return failed(this).feedback("challenge.close").build();
        }
    }
```

Explanation of the Vulnerability: An SQL injection is a technique that attackers apply to insert SQL query into input fields to then be processed by the underlying SQL database. These weaknesses are then able to be abused when entry forms allow user-generated SQL statements to query the database directly. An SQL injection attack consists of insertion or “injection” of either a partial or complete SQL query via the data input or transmitted from the client (browser) to the web application. A successful SQL injection attack can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file existing on the DBMS file system or write files into the file system, and, in some cases, issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands. A successful SQL Injection attack requires the attacker to craft a syntactically correct SQL Query. If the application returns an error message generated by an incorrect query, then it may be easier for an attacker to reconstruct the logic of the original query and, therefore, understand how to perform the injection correctly. However, if the application hides the error details, then the tester must be able to reverse engineer the logic of the original query.

Fix to the Vulnerability:

- **Continuous Scanning and Penetration Testing:** The automated web application scanner has been the best choice to point out vulnerabilities within the web applications for quite some time now. Now, with SQL injections getting smarter in exploiting logical flaws, website security professionals should explore manual testing with the help of a security vendor. They can authenticate user inputs against a set of rules for syntax, type, and length. It helps to audit application vulnerabilities discreetly so that you can patch the code before hackers exploit it to their advantage.

- **Restrict Privileges:** It is more of a database management function but enforcing specific privileges to specific accounts helps prevent blind SQL injection attacks. Begin with no privileges account and move on to 'read-only', 'edit', 'delete' and similar privilege levels. Minimizing privileges to the application will ensure that the attacker, who gets into the database through the application, cannot make unauthorized use of specific data.

- **Use Query Parameters:** Dynamic queries create a lot of troubles for security professionals. They have to deal with variable vulnerabilities in each application, which only gets graver with updates and changes. It is recommended that you prepare parameterized queries. These queries are simple, easy to write, and only pass when each parameter in SQL code is clearly defined. This way, your info is supplied with weapons to differentiate between code and information inputs.

- **Instant Protection:** A majority of organizations fail the problems like outdated code, scarcity of resources to test and make changes, no knowledge of application security, and frequent updates in the application. For these, web application protection is the best solution. A managed web application firewall can be deployed for immediate mitigation of such attacks. It contains custom policies to block any suspicious input and deny information breach instantly. This way, you do not have to manually look for loopholes and mend problems afterward.

3. Change this code to not deserialize user-controlled data

File name: InsecureDeserializationTask.java

Line Number: 56

URL to the Erroneous file:
https://github.com/WebGoat/WebGoat/blob/develop/webgoat-lessons/insecure-deserialization/src/main/java/org/owasp/webgoat/deserialization/InsecureDeserializationTask.java

Code Snippet:

```
public AttackResult completed(    @RequestParam String     token) throws IOException {
    String b64token;
    long before;
    long after;
    int delay;

    b64token =        token.replace('-', '+').replace('_', '/');

    try (    ObjectInputStream ois =     new ObjectInputStream(    new ByteArrayInputStream(    Base64.getDecoder().decode(b64toke
        before = System.currentTimeMillis();
        Object o =      ois.readObject();
```

Change this code to not deserialize user-controlled data. Why is this an issue? 2 years ago L56
Vulnerability Blocker Open Not assigned 30min effort No tags

Explanation of the Vulnerability: Deserializing untrusted data using any deserialization framework that allows the construction of arbitrary functions is easily exploitable and, in many cases, allows an attacker to execute arbitrary code. It is often convenient to serialize objects for communication or to save them for later use. However, deserialized data or code can often be modified without using the provided accessor functions if it does not use cryptography to protect itself. Furthermore, any cryptography would still be client-side security which is a dangerous security assumption. Data that is untrusted cannot be trusted to be well-formed. When developers place no restrictions on "gadget chains," or series of instances and method invocations that can self-execute during the deserialization process (i.e., before the object is returned to the caller), it is sometimes possible for attackers to leverage them to perform unauthorized actions, like generating a shell.

Fix to the Vulnerability: If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified. When deserializing data, populate a new object rather than just deserializing. The result is that the data flows through safe input validation and that the functions are safe. Explicitly define a final object() to prevent deserialization. Make fields transient to protect them from deserialization. An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the transient data should be. This is an excellent way to prevent time, environment-based, or sensitive variables from being carried over and used improperly. Avoid having unnecessary types or gadgets available that can be leveraged for malicious ends. This limits the potential for unintended or unauthorized types and gadgets to be leveraged by the attacker. Add only acceptable classes to an allowlist. Note: new gadgets are constantly being discovered, so this alone is not a sufficient mitigation.

4. **Change this code to not construct the path from user-controlled data**

File name: FileServer.java

Line Number: 73

URL to the Erroneous file:
https://github.com/WebGoat/WebGoat/blob/develop/webwolf/src/main/java/org/owasp/webwolf/FileServer.java

Code Snippet:

```
@PostMapping(value = "/WebWolf/fileupload")
public ModelAndView importFile(@RequestParam("file") MultipartFile myFile) throws IOException {
    WebGoatUser user = (WebGoatUser) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    File destinationDir = new File(fileLocation, user.getUsername());
    destinationDir.mkdirs();
    myFile.transferTo(    new File(destinationDir,     myFile.getOriginalFilename()));
```

Change this code to not construct the path from user-controlled data. Why is this an issue? 4 years ago L73

Vulnerability Blocker Open Not assigned 30min effort No tags

```
    log.debug("File saved to {}", new File(destinationDir, myFile.getOriginalFilename()));
    Files.createFile(new File(destinationDir, user.getUsername() + "_changed").toPath());

    ModelMap model = new ModelMap();
    model.addAttribute("uploadSuccess", "File uploaded successful");
    return new ModelAndView(
            new RedirectView("files", true),
            model
```

Explanation of the Vulnerability: Using user-controlled data in a permissions check may allow a user to gain unauthorized access to protected functionality or data. The application uses a protection mechanism that relies on the existence or values of an input, but the input can be modified by an untrusted actor in a way that bypasses the protection mechanism. Developers may assume that inputs such as cookies, environment variables, and hidden form fields cannot be modified. However, an attacker could change these inputs using customized clients or other attacks. This change might not be detected. When security decisions such as authentication and authorization are made based on the values of these inputs, attackers can bypass the security of the software. Without sufficient encryption, integrity checking, or other mechanism, any input that originates from an outsider cannot be trusted.

Fix to this Vulnerability: When checking whether a user is authorized for a particular activity, do not use data that is controlled by that user in the permissions check. If necessary, always validate the input, ideally against a fixed list of expected values. Similarly, do not decide which permission to check for based on user data. In particular, avoid using computation to decide which permissions to check for. Use fixed permissions for particular actions, rather than generating the permission to check for.

5. **Disable access to external entities in XML parsing.**

File name: Comments.java

Line Number: 90

URL to the Erroneous file:
https://github.com/WebGoat/WebGoat/blob/develop/webgoat-lessons/xxe/src/main/java/org/owasp/webgoat/xxe/Comments.java

Code Snippet:

```
protected Comment parseXml(String xml, boolean secure) throws JAXBException, XMLStreamException {
    var jc = JAXBContext.newInstance(Comment.class);
    var xif = XMLInputFactory.newInstance();
```

Disable access to external entities in XML parsing. Why is this an issue? 2 years ago L90
Vulnerability Blocker Open Not assigned 15min effort No tags

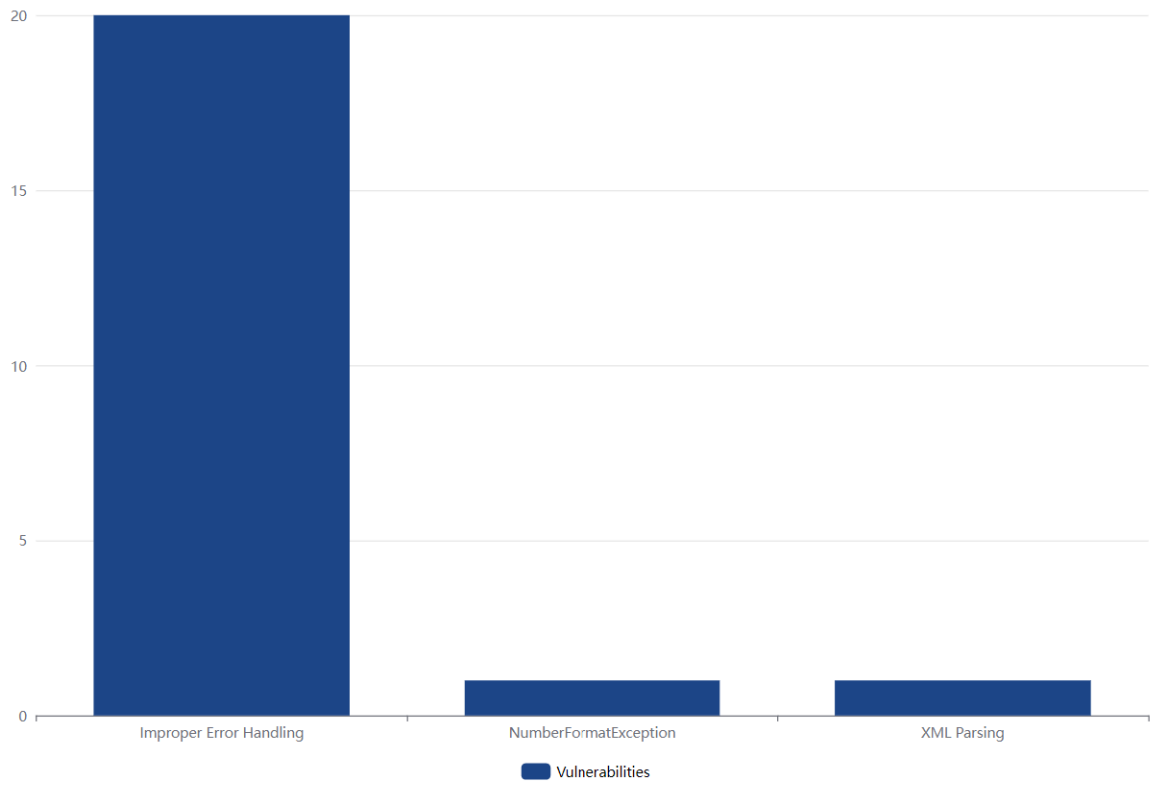
```
    if (secure) {
        xif.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, ""); // Compliant
        xif.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");  // compliant
    }

    var xsr = xif.createXMLStreamReader(new StringReader(xml));
```

Explanation of this Vulnerability: The software processes an XML document that can contain XML entities with URIs that resolve to documents outside of the intended sphere of control, causing the product to embed incorrect documents into its output. XML documents optionally contain a Document Type Definition (DTD), which, among other features, enables the definition of XML entities. It is possible to define an entity by providing a substitution string in the form of a URI. The XML parser can access the contents of this URI and embed these contents back into the XML document for further processing.
By submitting an XML file that defines an external entity with a file:// URI, an attacker can cause the processing application to read the contents of a local file. This attack may lead to the disclosure of confidential data, denial of service, Server-Side Request Forgery (SSRF), port scanning from the perspective of the machine where the parser is located, and other system impacts. The following guide provides concise information to prevent this vulnerability.

Fix to This Error: Since the entire XML document is communicated from an untrusted client, it is not usually possible to selectively validate or escape tainted data within the system identifier in the DTD. Therefore, the XML processor should be configured to use a local static DTD and disallow any declared DTD included in the XML document.

The safest way to prevent XXE is always to disable DTDs (External Entities) completely. Disabling DTDs also makes the parser secure against denial of services (DOS) attacks such as Billion Laughs. If it is not possible to disable DTDs completely, then external entities and external document type declarations must be disabled in the way that's specific to each parser.

Summary of Vulnerabilities Found in SonarQube Scan of http://demo.testfire.net/

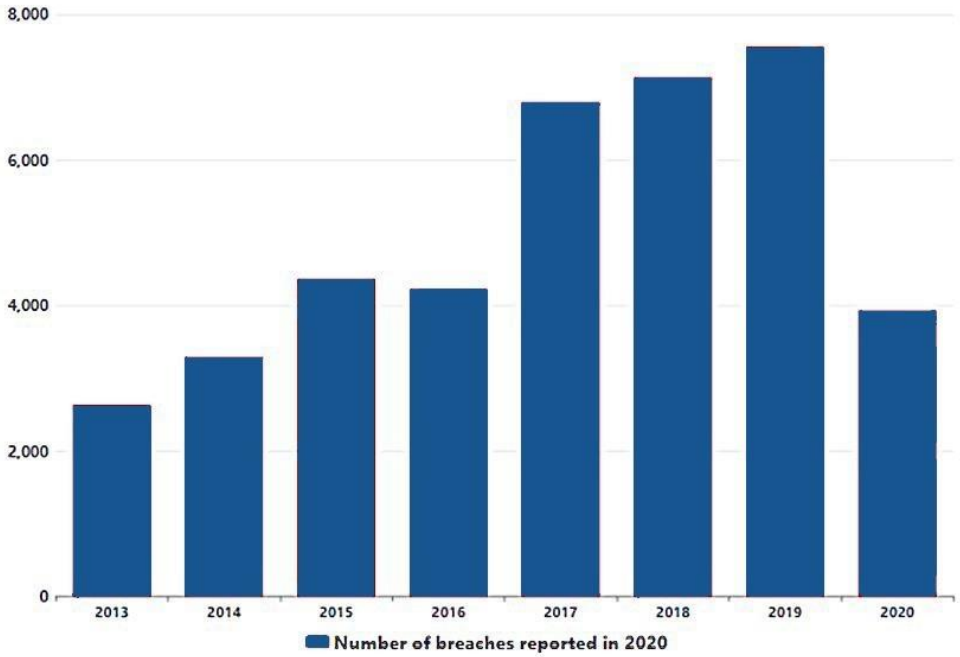


Challenges

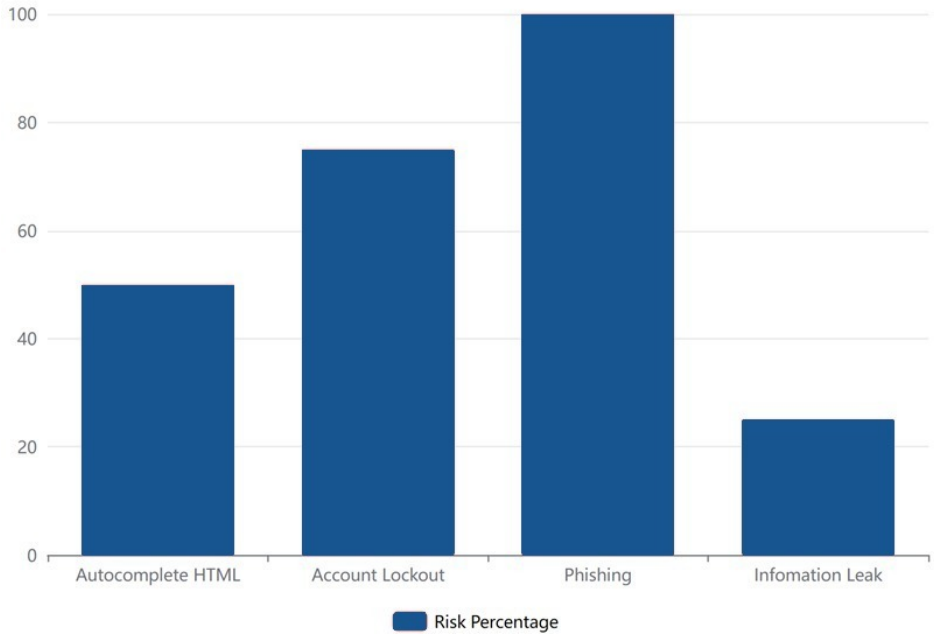
There are many cons associated with static source code analysis, some of them are as following:

- When done manually, it takes a long time.
- There aren't enough qualified people to do a full static code analysis.
- It gives the misleading impression that everything is being taken care of.
- It does not detect vulnerabilities that have been introduced into the runtime environment.
- There are only a certain number of vulnerabilities that can be discovered through source code analysis which leaves out many critical vulnerabilities.

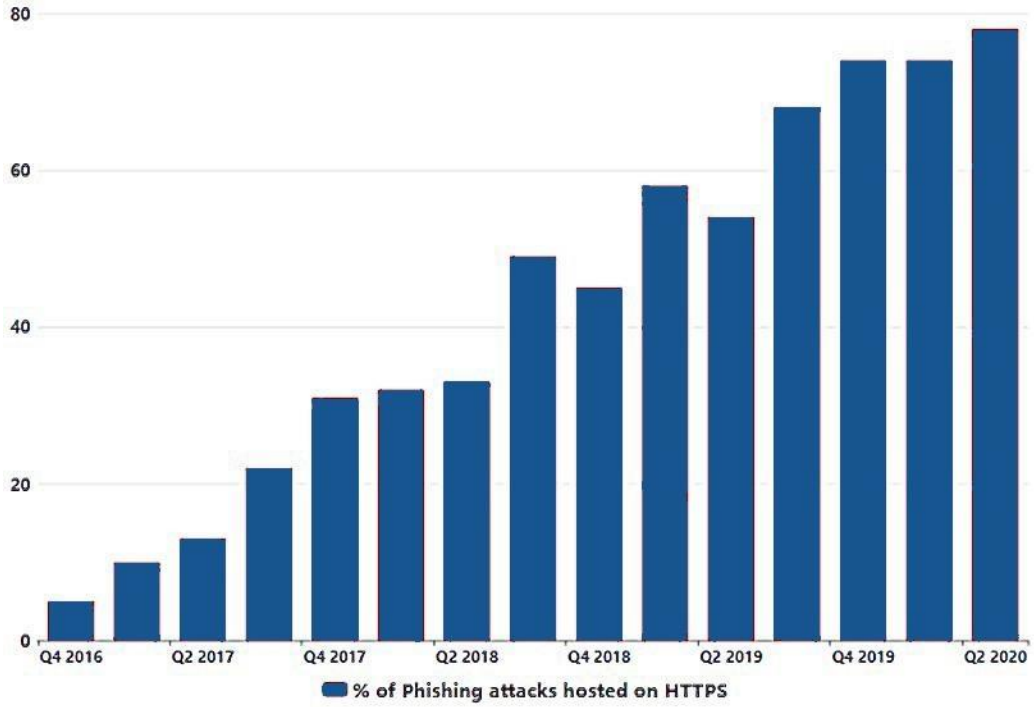
Information Leakage Throughout the Years



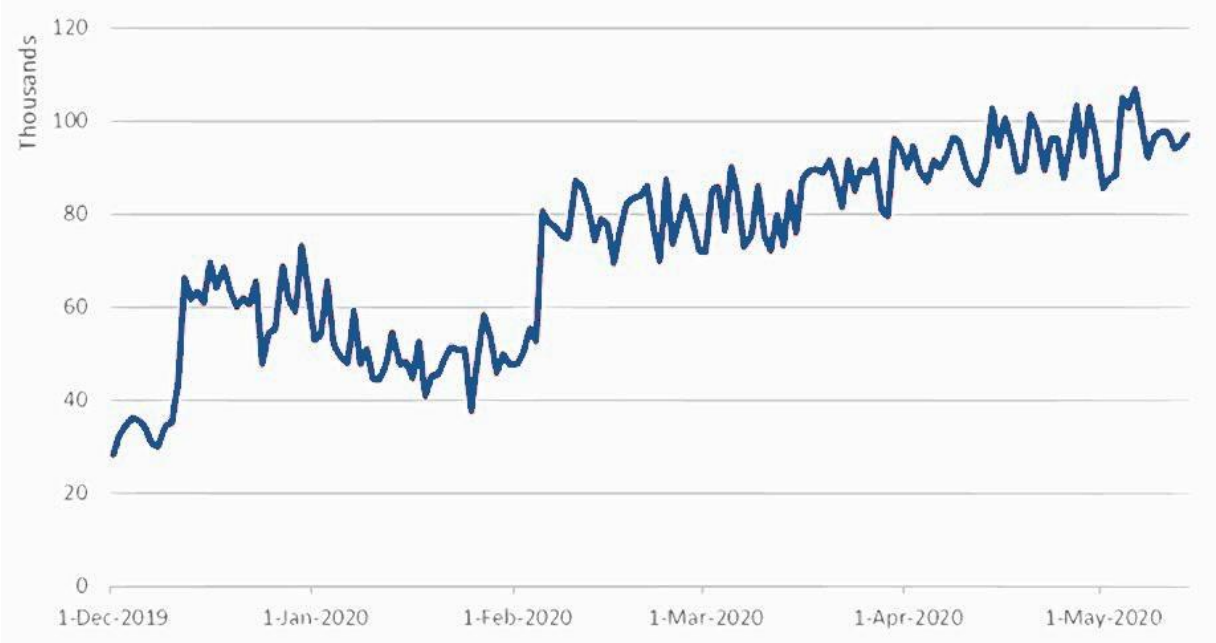
Bar Chart According to Risk of Severity Percentage



Phishing Attacks Throughout the Years



Brute Force Attacks



Reflections on the Internship

This was a very time-consuming project which helped me gain a lot of practical knowledge about Static Analysis Security Testing. Patience was the crucial and essential factor that was needed in this project in my view. Fortunately, I learned and gained the patience.
It required a lot of patience and determination to go through numerous code files and try to search for vulnerabilities manually. In the process, I gained a lot of knowledge on the subject.
I learned a lot about many kinds of vulnerabilities and the risks associated with them. Analyzing the source code of an insecure application is a great way of learning about static analysis security testing.

Conclusion

Building safe systems requires time and commitment, especially for companies that are not used to prioritizing security. The software security approach should include code review. Static code analysis verifies that source code adheres to predetermined design and style criteria in a quick and automated manner.

Implementing such criteria results in more consistent code, as well as the identification of possible security, performance, interoperability, and globalization flaws. Human-led code reviews cannot be replaced by static code analysis technologies. Rather, they may perform a first pass of the code base and identify places that require more attention from a senior developer.

Link to code and executable file

Link to the web application:
https://demo.testfire.net/

https://www.vulnspy.com/webgoat-8/

Link to GitHub page:
https://github.com/HCL-TECH-SOFTWARE/AltoroJ

https://github.com/WebGoat/WebGoat