STATIC AND DYNAMIC ANALYSIS OF VNV PROJECT

This document was divided into two parts. Static analysis and dynamic analysis for the given feedback gathering tool were performed and the results of these testing are performed

GROUP PARTICIPATION:

Name	P.number	Mail
Paruchuri Sai Chand	9505220096	saichandmoon@gmail.com
Maheshwar Kota	9212199070	mako16@student.bth.se
Chandan Srivatsa Ganuga	9408193234	chandan@gnmail.com
Qi Pengyang	9502136931	qpytzzj@gmail.com
Feng Shaofeng	9106060016	ilxy060821@gmail.com
Poornanada Varma Nadimpalli	9407153775	pona17@student.bth.se

STATIC ANALYSIS

***Abstract*—**In this document we have performed static analysis on the open source code of feedback gathering tool. Static analysis is testing the code without running the code. We used visual code grepper open source static tool to analyze the code. The result of the static analysis and the analysis of the outcomes is presented below.

INTRODUCTION

Software testing is an important part of software lifecycle. Testing is done to identify the errors in the system before its deployment. These errors when unidentified in early stages can increase the costs during the maintenance phase. This can affect the customer satisfaction with the product. Software testing can be done by performing static analysis and dynamic analysis of code. In this section, we use static automated analysis to identify the errors in the feedback gathering tool.

Static analysis involves analysis of the code without executing the code [1]. It is considered as an automated code review process which identifies the errors in the source code in the earlier stages and mitigates the risk of loss of time and reduce maintenance costs. We have discussed about the inspection goals and selected a tool that would satisfy our goals. The tool we have chosen to perform static analysis is visual code grepper.

I. Static analysis Goals:

***a. Identification & selection of relevant goals**:*

For testing a product, goals are very important. The testing goals of the product will have direct impact on the quality of the product. There are many goals which were used to determine the quality of a code.

For this project, we selected a feedback gathering tool which was plugin used for gathering users feedback for a web application. For the static analysis of the given code we identified the potential goals that affect the quality of the given product. Then as a team we decided and selected some of the most relevant goals among those mentioned below. We further provided the motivation for selecting or rejecting the mentioned goals.

The relevant inspection goals identified for this product are: [a]

1.Performance: Performance of a software depends upon the speed and reaction time of the product. It can

be a potential quality goal for the feedback monitor application to enhance the performance of the product.

2.Scalability: Scalability is the capability of a system to handle the growing amount of work load. Here scalability of the monitor feedback application depends upon the performance of the application while increasing the users.

But we rejected both the performance and scalability as our static analysis. The major reason for rejection was the metrics for performance are suitable for measuring with the dynamic tools. Similarly, testing the scalability (which was highly related to performance) was not effective with static analysis when compared to dynamic analysis. [b]

3.Size: Size of a software was defined based on the lines of the code. It was also identified as a potential goal for our analysis. This goal was selected later because there are many open source static tools available for measuring the line of code.

4.Complexity: The complexity of a software is defined as interactions between several entities. The complexity of the software increases with the increase of interactions among the internal entities. In our project, we selected this goal because of availability of the static analysis tools which can be effectively the measure the metrics of the code complexity.

5. Usability: Usability of the monitor feedback system was the look of the application and the ease to understand for the user to use this application. So, usability was also one of the possible goal identified for the static analysis. However, we not selected this as our static analysis goal because, according to our team measuring the usability of a product with the static tool was inappropriate as it depends upon the look and feel of user.

6.Maintainability: Maintainability of a software is the ease of the product to upgrade or modification. Here this was a relevant goal how the monitor feedback system was maintained. We selected this as one our goal for static analysis. The motivation behind selecting this was availability of open source tools for measuring the maintainability of software code that developed in Java.

7.Security: Security was the most important quality aspect. The software with security flaws was considered as very risky to use. Hence the security of the monitor feedback was considered as the important goal for our static analysis. This was prioritized as our primary goal for static analysis. There are many tools available to detect the security vulnerabilities of the Java code.

8.Reliability: Reliability for a software defined as how trustworthy a software product was. Reliability was a very important quality attribute. This goal was rejected because the reliability of a web software was evaluated effectively with failure data extracted from the server logs[c].

b. Conclusion:

We selected *size, maintainability, security* and *complexity* of the code as our quality goals and we rejected the goals like *performance, scalability, usability* and *reliability* for the static analysis of this project. The motivation for these decisions were explained above.

1	**Selected Goals**	Size, Maintainability, Security, Complexity
2	**Rejected Goals**	Performance, Scalability, Usability, Reliability

II. Static analysis tools

a. Identification of relevant tools:

1.FindBugs: FindBugs is a static analysis tool for java. It is a highly configurable tool which allows loading custom rules. To identify the suspicious vulnerabilities in the code, FindBugs syntactically match the source code. It also uses dataflow analysis in few cases to identify the bugs. It generates reports in HTML, XML. It can be used as standalone GUI application or Eclipse plugin. [d][e]

2. Codepro analytix: Codepro analytix was an eclipse plugin for static analysis. The configuration rules of this tool was very similar to FindBugs tool. But it has additional features. It generates the reports in HTML. The Codepro analytix was generally easy to use and configure. It also has few more additional rules than FindBugs tool.

3.PMD: PMD was a stand-alone script tool which was used in static analysis for java. This tool finds the

errors in the code like empty catch blocks, unnecessary object creation and unused variables. PMD includes many bug detectors based on the style of the code. It generates reports in XML, HTML and Command Line. PMD performs syntactic analysis checks on the program source but unlike FindBugs it does not have data flow component. [e][f]

4.VCG: The virtual code grepper tool was a statistic analysis tool for java. It was a fully customizable and stand-alone tool. Here it has an advantage that the user can add new patterns for the vulnerabilities to be detected. This tool generates the reports in text, program. This tool increases the efficiency of the analysis as it implements the quick access to the affected file by highlighting the errors.

5.JLint: JLint was also a tool used in static analysis for java. It have some components which were interfile. This will help in finding the deadlocks by building a lock graph. It was not expandable easily. [e][g]

6. Check style: It was static analysis tool for checking java. This tool helps in improving the quality of the code, readability of the code. This tool was also cost effective. But the content was not analysed by the checks. It only clarifies the inspecting goals and does not confirm the completeness of the code.

b. Motivation for selecting/ rejecting the tools:

We selected the following tools because of following reasons

s.n	Tool	Decision	Reason
1	FindBugs	Rejected	Although it provides the results effectively, it was hard to write and maintain. It was tasking and takes time when compared with visual code grepper.
2	Codepro analytix	Rejected	it was not concentrated more on typical java bugs. The installation was not simple. Menu items for code coverage are not existent.
3	PMD	Rejected	It was limited to one class and does not implement advanced checks
4	VCG	Accepted	It was very easy to use and produces the results effectively and quickly.
5	JLint	Rejected	It was not easily expandable. It does not generate the reports properly.
6	Check style	Rejected	It was very tedious to use with the style constraints. It lacks completeness.

III. Reflection of outcomes*:*

a. Classification of warnings:

Using virtual code grepper we detected warnings which were divided into seven categories. We totally found errors in the static analysis. All these warnings are with different severity levels like standard, medium and low.

The warnings and the description of them are mentioned below category wise:

1. **Public class not declared as final:** The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.

2. **Operation on primitive data type:**
This function acts as an entry point for external data and the code should be manually checked to ensure the data obtained is correctly validated and/or sanitised. Additionally, careful checks/sanitisation should be applied in any situation where the user may

be able to control or affect the filename.

3.**The comment indicates potentially unfinished code:** It indicates a suspicious comment.

4. **Java.io.file:** This functionality acts as an entry point for external data and the code should be manually checked to ensure the data obtained is correctly validated and/or sanitised. Additionally, careful checks/sanitisation should be applied in any situation where the user may be able to control or affect the filename.

5. **Java.io.FileOutputStream:** This functionality acts as an entry point for external data and the code should be manually checked to ensure the data obtained is correctly validated and/or sanitised. Additionally, carefull checks/sanitisation should be applied in any situation where the user may be able to control or affect the filename.

6. **Failure to release resources in all cases:** There appears to be no release of resources in the 'finally' block, potentially resulting in DoS conditions from excessive resource consumption.

7. **FileInputStream:** This function acts as an entry point for external data and the code should be manually checked to ensure the data obtained is correctly validated and/or sanitised. Additionally, careful checks/sanitisation should be applied in any situation where the user may be able to control or affect the filename.

b. Review of warnings:

After performing the static analysis with the visual code grepper we found a total of 156 defects which were divided into above mentioned seven categories. Then to identify whether these defects are truly positive or false positive all the team members are evaluated manually. Each team member initially reviewed the code individually.

All these warnings are treated as true positive or false positive or undecided. The warning which was a true positive can potentially create harm to the plugin. A false positive warning is a test result which wrongly indicates a vulnerability. As all the team members are new to testing due to lack of experience, some of the warnings were undecided. Here the evaluators can't decide whether the warning was true positive or false positive.

Later all our team members discussed together to get into consensus. It took approximately 52 hours collectively for the team to execute the entire code and finding the warnings. The end results are stated below in the appendices.

fig1:results displayed in VCG

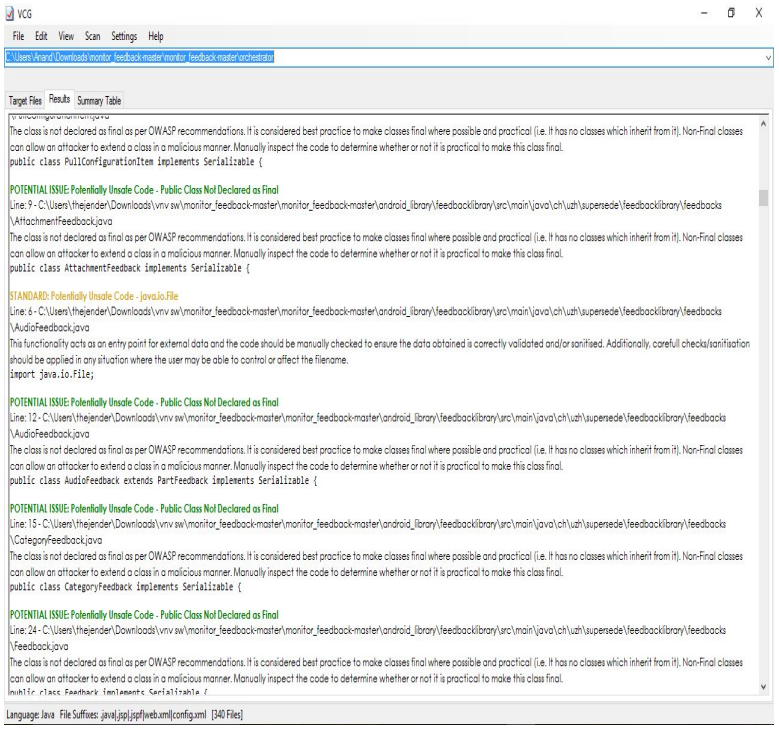
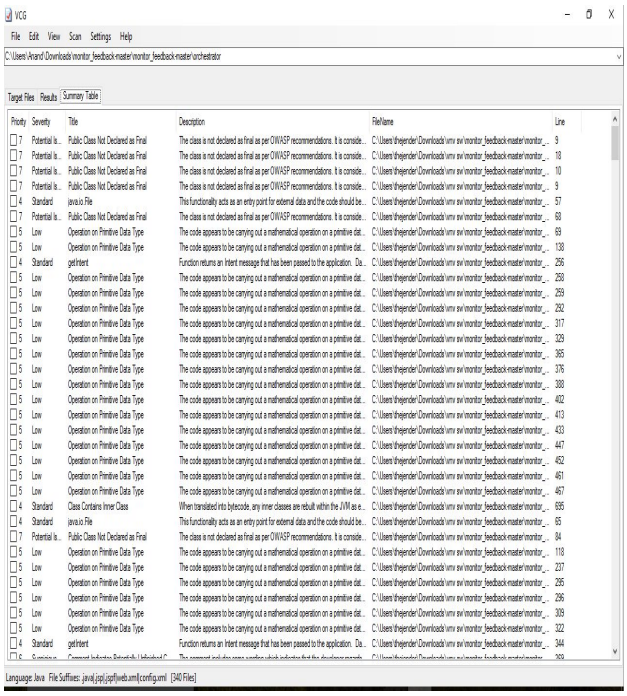


Fig 2:Summary in VCG



a. Tool performance & cost effectiveness:

The main objectives for the static analysis like security, size, maintainability and complexity were achieved by this tool. We evaluate the cost effectiveness of the tool based on data of time and true errors found. Here we have performed static analysis on feedback gathering tool with the automated tool visual code grepper. It took less than one minute to execute the results with the visual code grepper. In the total 156 errors, we detected 134 errors as true positive errors. For every individual, it took approximately four minutes for each warning to decide whether it was true or not.

This tool was highly cost effective and the performance of the tool results very good results as it has very good bug detecting capacity and good execution efficiency.

IV. Conclusion:

In this section we illustrate the static analysis part i.e. selection of goals, tools and evaluation of outcomes. Here for this project we selected the tool visual code grepper for static analysis. This tool helps to test the goals of our project. Visual code grepper was easy to use and the reports generated by them are very clear. By evaluating the results we found total **jfdslk** warnings, in that after further discussion we find **sa%** of them are true positives and **fg%** are false positives. It was detected all these warnings within less than one minute which was very high when compared to remaining tools. This tool was highly cost-effective and highly efficient detecting the bugs.

REFERENCES

[a] Chung, Lawrence, et al. *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media, 2012.

[b] Lee, Lyndon C., et al. "The stability, scalability and performance of multi-agent systems." *BT Technology Journal* 16.3 (1998): 94-103.

[c] Tian, Jeff, Sunita Rudraraju, and Zhao Li. "Evaluating web software reliability based on workload and failure data extracted from server logs." *IEEE Transactions on Software Engineering* 30.11 (2004): 754-769.

[d] Hovemeyer, David, and William Pugh. "Finding bugs is easy." *ACM Sigplan Notices* 39.12 (2004): 92-106.

DYNAMIC ANALYSIS

***Abstract*—In this document we have performed dynamic analysis on the open source code of feedback gathering tool. Dynamic analysis is testing the code while running the code. The result of the static analysis and the analysis of the outcomes is presented below.**

Introduction

Dynamic analysis involves analyzing the code based on the system execution. Structural testing and functional testing are the two categories in the dynamic analysis. Errors in the code are identified while the system is running. This can also enable us to understand the behavior of the system.

Tools for the analysis of the code are selected based on the inspection goals. These tools are used to generate automated test cases to analyze the code. The following sections present details about how the analysis is carried out.

Ⅰ. SCOPE

The purpose of this section is to perform a dynamic code analysis on the Feedback Gathering tool. In this section, we will use two dynamic code analysis tools, the selected tool generation Automatic test cases, these test cases are executed to identify errors in the code.

Stages of automated test processes are

·Automated unit testing
·Automated System Testing
·Automated Acceptance testing

We finally chose the automated unit testing, and the reason is as follows:

Unit testing: Unit testing is a type of software testing performed on each individual unit of the software, the generation of test cases is essential in dynamic code analysis [1]. Manually generating test cases is time-consuming and costly. In the context of our project, the use of unit testing is designed to detect the quality of each module of the software and to identify defects in each module.

System testing: System testing is used to detect the entire product system, verify that the function meets the requirements, and finds the difference from the specification. System testing should cover all components. If the software specification (like the design of the software instructions, the software requirements specification and other documents) is not complete, the system test is more dependent on the tasters work experience and judgment, this test is not sufficient [2]. So it is not suitable for our dynamic analysis.

Acceptance testing: Acceptance testing is a formal test to determine whether the system meets its acceptance criteria, satisfies the user's point of view, the developer's point of view, and the customer's point of view. from a user's point of view-and to enable the customer to determine whether or not to accept the system Acceptance testing is a phase of testing which is used to verify if the code conforms to the user requirements or business logic [3]. Although the full source code for the Feedback Gathering tool is available, the requirements for advanced users are unclear and the acceptance testing may not be able to find the expected defects due to subjective reasons. So we do not choose acceptance testing.

Ⅱ. INSPECTION GOALS

Use the dynamic test to understand the quality and functionality of the feedback Gathering tool. By running the test program, check the difference between the running results and the expected results, analyze the operational efficiency and correctness. So here are our 2 inspection goals:

Correctness: The correctness of the program can be categorized into several aspects like traceability, completeness and consistency [4]. The unit tests for each functional unit can be performed as specified, so the integrity of the detection code is more suitable for this test.

Efficiency: Program efficiency refers to the execution speed of the program and the storage space occupied, it is also of prime concern that required the performance is high in accordance with the amount of code files under the specified conditions [4]. The feedback Gathering tool is used to collect the user's feedback on the Web application plug-in, we focus on the testing time, which is the duration that the application run the

test case [4]. So, we focus on testing time, test program efficiency.

III. TOOLS SELECTION

With the development of software testing, automatic testing tools are more and more popular in the tester community. From the initial concept, to the present many types of automatic testing tools, they are classified by object, testing levels, testing types and so on, like Junit which object is Java coding language and object testing level is unit testing, PHPunit which object is PHP coding language and object testing level is unit testing too. For this project, our object software is a web application named Feedback Gathering tool which is basis on JSP coding language, so our group decide to use selenium and Junit, they are from 5 first chosen tools after we have a few understanding of the automatic testing tool. There are the introductions of those tools in the table:

Name of tool	**Introduction**	**Accepted/ Rejected**
Selenium	Selenium is an open source software, this web application is test framework software, it can support Python, Java, C#, JavaScript and so on, at the time this application can execute different platform like Windows, Linux and Macintosh.	Accepted
Watir	Web Application Testing in Ruby(Watir) is a test automatic framework, this is a smaller and more convenience web application, you can use Ruby code script.	Rejected
Sahi	Sahi is a web application too; this automatic testing tool can run any browser support Javascript.	Rejected
Junit	Junit is a unit test framework, be developed by Erich Gamma and KentBeck. This open source tool use white-box testing approach, and now this software becomes more and more popular in testing of Java development.	
Jtest	Jtest is a white-box testing tool too, it can achieve unit automatic testing. Jtest biggest advantage lies in static code analysis.	rejected

Reason of Rejection:

Watir: This open source software be compared with Selenium; the tool only can support Ruby script and it not includes script record function.

Sahi: This tool is similar as Selenium and Watir, but it script language support is less than selenium too and it costs more test time than Selenium, so Selenium is our first selection.

Jtest: A tool is same as Junit, it is open source too, but if you use this unit automatic testing framework you must check your Junit code and maintain script by yourself and here is another disadvantage is that the script created by this Jtest cover less than Junit.

Reason of acceptance:

Selenium:

This application is an open source software and automatic testing tool first, then it supports variety browser, variety platform and variety scripting language. Tester is easier to create testing case and less testing time, at the same the script created by itself have higher coverage than other software.

Junit: First of all, this tool suite project requirement, i

is free and can create automatic test case, this tool can

Dynamic testing test multiple test case at the same time. This tool can provide API to tester who write reusable test case and there are three ways to show the results of test.

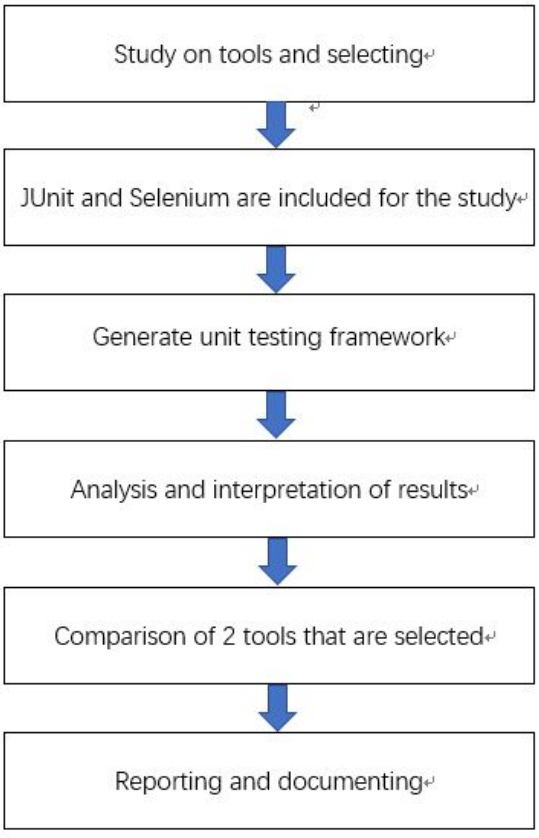
IV. REVIEW PROCESS

TEST MODEL

Automated test design is a test design method, which uses automated machines to generate test artifacts, such as from the formal test of test cases and test data, most commonly known as the test model [5]. In software testing, we need to verify whether the software meets the established technical and business needs, measuring the quality of the software. The tests include unit testing, component testing, integration testing, system testing, and acceptance testing, which require different tools for testing.

TEST PLAN

We first determined the scope of the test, and then decided to test tools and methods, and build a test environment to be delivered results. We use WINDOW 10 as a platform tool to eclipse as a code to run the tool. The flowchart is as follows:



TEST CASES/TEST DATA

We use Selenium to generate automated test cases using JUnit format and perform these test cases in the Eclipse IDE-based JUnit framework. Test input, expected results for specific target program settings, to verify that specific requirements are met.

V. Results and Analysis

		Selenium	**Junit**
1	Cases generating time	50mins	75mins
2	Number of test case	50	43
3	Average coverage of test cases	73.1%	80.7%
4	Total goals	97	69
5	Number of covered goals	43	37
6	Number of errors	1	3

Comparison

There are 6 aspects of comparison between the Selenium and Junit Results. From this table, we can see that Selenium is better than Junit at many aspects like case generating time, number of errors but Junit has good reusability of test cause, so average coverage of test cases is higher than Selenium. It is easy to see each tool has own features, so when we use them we should make their features suit the requirements.

VI. Conclusion

dynamic testing has many advantages like reduce total test time and number of tester of a program. In this part, we compare 5 dynamic test tools, two of those are chosen to test. We feel some limitations of dynamic test from that testing, limitations such as the coverage of test case is less than the one generated by tester.

Reference

[1] S. Wang and J. Offutt, “Comparison of Unit-Level Automated Test Generation Tools,” in International Conference on Software Testing, Verification and Validation Workshops, 2009. ICSTW ’09 , 2009, pp.210–219.

[2] L. Briand and Y. Labiche, “A UML-Based Approach to System Testing. Carleton University,” Softw. Syst. Model., vol. 1, no. 1, pp. 1–57, 2002.

[3] D. Talby, O. Nakar, N. Shmueli, E. Margolin, and A. Keren, “A process-complete automatic acceptance testing framework,” in IEEE International Conference on Software - Science, Technology and Engineering, 2005. Proceedings, 2005, pp. 129–138.

[4] P. Berander, L. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jönsson, S. Kågström, D. Milicic, F. Mårtensson, K. Rönkkö, P. Tomaszewski, L. Lundberg, M. Mattsson, and C. Wohlin, “Software quality attributes and trade-offs,” no. June, pp. 1–100, 2005.

[5] M.-F. Wendland, “Abstractions on test design techniques,” in 2014 Federated Conference on Computer Science and Information Systems (FedCSIS), 2014, pp. 1575–1584

Identifier of defec	Description of Defect	Assessmen	Description of values for assessment	
1	Warning 1	True positi	True positive (warning represents a real defect)	
2	Warning 2	True positi	False positive (wrong warning)	
3	Warning 3	Undecided	Undecided (do not know)	
4	Warning 4	False positive		
5	Warning 5	False positive		

Identifier of defec	Description of Defect	Assessmen	Description of values for assessment	
1	Warning 1	False positi	True positive (warning represents a real defect)	
2	Warning 2	True positi	False positive (wrong warning)	
3	Warning 3	Undecided	Undecided (do not know)	
4	Warning 4	False positive		
5	Warning 5	False positive		

Identifier of defec	Description of Defect	Assessmen	Description of values for assessment	
1	Warning 1	False positi	True positive (warning represents a real defect)	
2	Warning 2	Undecided	False positive (wrong warning)	
3	Warning 3	Undecided	Undecided (do not know)	
4	Warning 4	False positive		
5	Warning 5	False positive		

Identifier of defec	Description of Defect	Assessmen	Description of values for assessment	
1	Warning 1	True positi	True positive (warning represents a real defect)	
2	Warning 2	True positi	False positive (wrong warning)	
3	Warning 3	Undecided	Undecided (do not know)	
4	Warning 4	False positive		
5	Warning 5	True positive		

Identifier of defec	Description of Defect	Assessmen	Description of values for assessment	
1	Warning 1	False positi	True positive (warning represents a real defect)	
2	Warning 2	True positi	False positive (wrong warning)	
3	Warning 3	Undecided	Undecided (do not know)	
4	Warning 4	False positive		
5	Warning 5	False positive		

Identifier of defect	Description of Defect	Reviewer 1	Reviewer 2	Reviewer 3	Reviewer 4	Reviewer 5	Reviewer 6	Consensus (after group discussion)		
Feedback servlet	public class FeedbackServlet extends ServletBase { class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.	True positive	False positive	False positive	True positive	True positive	False postive	True positive		
Guics Repository Context Listener	public class GuiceRepositoryContextListener extends GuiceServletContextListener { class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.	True positive	True positive	False positive	True positive	True positive	True positive	True positive		
Rest Controller	public class FeedbackStatusC	True positive	True positive	True positive	True positive	True positive	True positive	True positive		
Importing java.io file	import java.io.File This functionality acts as an entry point for external data and the code should be manually checked to ensure the data obtained is correctly validated and/or sanitised. Additionally, careful checks/sanitisation should be applied in any situation where the user may be able to control or affect the filename.	high	False positive	False positive	False positive	False positive	False postive	False positive		

Importing java.io.FileInputStrea	import java.io.FileInputStream;	True positive	True positive	True positive	True positive	True positive	True positive	True positive		
File controler	public class FileController {	True positive	True positive	True positive	True positive	True positive	True positive	True positive		
Constructor	InputStream is = new FileI	True positive	True positive	True positive	True positive	True positive	True positive	True positive		
A.L.U.	offset += numRead; The code appears to be carrying out a mathematical operation on a primitive data type. In some circumstances this can result in an overflow and unexpected behaviour. Check the code manually to determine the risk.	True positive	True positive	True positive	True positive	True positive	False postive	True positive		
Controller	public class StatusOptionController extends RestController<StatusOption>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.	True positive	False positive	True positive	False positive	False positive	True positive	Undecided		

Repository	public class DataProviderIntegratorRepository { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Integrator repository	public class MdmIntegrator									
Mail service	public class MailService Th									
SUSPICIOUS COMMENT	Comment Indicates Potenti	low								
Repository mail configuration	public class RepositoryMail									
Attachment Feedback	public class AttachmentFee									
Audio feedback	public class AudioFeedbacl	high								
Category feedback	public class CategoryFeedb									

Context Information	public class ContextInformation extends ItemBase<ContextInformation> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Itembase Feedback	public class Feedback extends ItemBase<Feedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

Feedback comment	public class FeedbackComment { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
FileFeedback ItemBase	public abstract class FileFe									
RatingFeedback	public class RatingFeedback extends ItemBase<RatingFeedback>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Screenshot feedback	public class ScreenshotFeedback									

Status	public class Status extends ItemBase<Status> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Status options	public class StatusOption extends ItemBase<StatusOption>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

TextAnnotation	public class TextAnnotation extends ItemBase<TextAnnotation> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
TextFeedback	public class TextFeedback extends ItemBase<TextFeedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

FeedbackSerializationServic	public class FeedbackSerializationService extends RepositorySerializationService<Feedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
File Output Stream	import java.io.FileOutputStream; This functionality acts as an entry point for external data and the code should be manually checked to ensure the data obtained is correctly validated and/or sanitised. Additionally, carefull checks/sanitisation should be applied in any situation where the user may be able to control or affect the filename.	high								

FileStorageService	public class FileStorageService { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
File Name Of Stored File	String fileNameOfStoredFile = Integer.toString(fileSize) + "_" + String.valueOf(new Date().getTime()) + "." + feedback. getFileExtension(); The code appears to be carrying out a mathematical operation on a primitive data type. In some circumstances this can result in an overflow and unexpected behaviour. Check the code manually to determine the risk.									
InputStream read	while ((read = inputStream.									
Failure To Release Resource	There appears to be no rele	high								

Repository Serialization Service	public abstract class RepositorySerializationService<T> extends DefaultSerializer<T> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.		high							
Status Option Serialization Service	public class StatusOptionS									
Attachment Feedback result parser	DbResultParser<AttachmentFeedback>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final. public class AttachmentFeedbackResultParser extends									

Attachment feedback service	public class AttachmentFeedbackService extends ServiceBase<AttachmentFeedback>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Context Information result parser	public class ContextInformationResultParser extends DbResultParser<ContextInformation> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

Context information service base	public class ContextInformationService extends ServiceBase<ContextInformation>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Feedback result parser	public class FeedbackResultParser extends DbResultParser<Feedback>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Feedback service extends	public class FeedbackServi									

Rating feedback parser	public class RatingFeedbackResultParser extends DbResultParser<RatingFeedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.	high								
Screenshot feedback	public class ScreenshotFeedbackResultParser extends DbResultParser<ScreenshotFeedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.	High								

Screenshot feedback Service	public class ScreenshotFeedbackService extends ServiceBase<ScreenshotFeedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.	high								
Status options result parser	public class StatusOptionR	high								
Status option	public class StatusOptionService extends ServiceBase<StatusOption> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.	high								

status result parser	public class StatusResultParser extends DbResultParser<Status> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Status service base	public class StatusService extends ServiceBase<Status> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

Text annotation result parser	public class TextAnnotationResultParser extends DbResultParser<TextAnnotation> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Text annotation service base	public class TextAnnotationService extends ServiceBase<TextAnnotation>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Text Feedback result parser	public class TextFeedbackF	high								

Text Feedback service base	public class TextFeedbackService extends ServiceBase<TextFeedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Repository database configuration	public class RepositoryDatabaseConfiguration extends DatabaseConfiguration { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.	high								
Attachment validator	public class AttachmentVali									

Audio Validator	public class AudioValidator extends ValidatorBase<AudioFeedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Category validator	public class CategoryValidator extends ValidatorBase<CategoryFeedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

Context validator	public class ContextValidator extends ValidatorBase<ContextInformation> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Feedback validator	public class FeedbackValidator extends ValidatorBase<Feedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

Rating validator	public class RatingValidator extends ValidatorBase<RatingFeedback> { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Screenshot validator	public class ScreenshotValidator extends ValidatorBase<ScreenshotFeedback>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Status option vallidator	public class StatusOptionVa									
Status validator	public class StatusValidator									

Text annotation validator	public class TextAnnotationValidator extends ValidatorBase<TextAnnotation>{ The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Text feedback validator	public class TextFeedback\									
Data provider integratoe repository	public class DataProviderIntegratorRepositoryTest { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

Mail service test	public class MailServiceTest { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									
Suspicious comment	TODO ATTENTION: canno	low								
Integrator Repository Test	public class MdmIntegratorRepository Test { The class is not declared as final as per OWASP recommendations. It is considered best practice to make classes final where possible and practical (i.e. It has no classes which inherit from it). Non-Final classes can allow an attacker to extend a class in a malicious manner. Manually inspect the code to determine whether or not it is practical to make this class final.									

defect type	defect	Sai Chand	Mahesh	pengyang	feng	chandan	anand	consenses	no. of warnings
potentially unsafe code	Public class Not declared as final	true positive	true positive	true positive	true positive	true positive	true positive	true positive	128
potentially unsafe code	operation on primitive data type	false positive	false positive	false positive	false positive	true positive	false positive	false positive	8
suspicious comment	comment indicates potentially unfininished	false positive	false positive	false positive	false positive	false positive	false positive	false positive	4
potentially unsafe code	java.io.File	true positive	undecided	true positive	true positive	false positive	true positive	undecided	8
potentially unsafe code	java.io.FileOutputStream	true positive	true positive	true positive	true positive	undecided	true positive	truepositive	2
potentially unsafe code	Failure to release resources In All Cases	false positive	undecided	true positive	true positive	true positive	false positive	false positive	2
potentially unsafe code	FileInputStream	true positive	true positive	undecided	false positive	true positive	true positive	true positive	4