AVISCA: A Static Code Analyzer

Alpesh M. Patil[1] Aniket V. Pataskar[2] Vandana P. Tonde[3]
[1,2,3]Department of Information Technology
[1,2,3]Sinhgad Institute of Technology, Lonavala

Abstract— Security and performance are crucial aspects of software quality measurement. Static code analysis is one of the techniques used for quality assurance of software. Here we propose a system which can be used on the salesforce platform for static code analysis of the apex code. The three main component of tool will evaluate the apex source code step by step and produce final results which can be used to determine the code quality. This paper goes through various methods that can be used to develop static code analysis tool.
Key words: AVISCA, Apex, Salesforce, Static Code Analysis, Software Quality Assurance

I. INTRODUCTION

Salesforce is a cloud application platform. It allows developing and hosting applications. The business logic for salesforce app can be derived from apex programming. Apex is a strongly typed and an object-oriented programming language for use on Salesforce platform. Using syntax that looks like Java, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages.

Nowadays the security of the software is more important. Sometimes we may code the software correct but in some situation it behaves abnormally. We should detect the vulnerabilities in the software. Here vulnerabilities means small errors that can crash the software. The static code analyzer scans all the code and detects runtime vulnerabilities in the software. The use of static code analyzer in order to avoid bugs is one of the main pillars of the software development.

Static Code Analysis is usually performed as part of a Code Review i.e. white-box testing and is carried out at the Implementation phase of a Security Development Lifecycle. Static Code Analysis refers to the running of Static Code Analysis tools that attempt to highlight possible vulnerabilities within the 'static' i.e. non-running source code. Static analysis allows us to reason about all possible executions of a program. It gives assurance about any execution, prior to deployment.

The Static code analysis can be done in two ways:

- Manually.
- Using automated Tool.

In order to perform manual code analysis we need skilled professionals. They will check the source code and create detail reports. The professionals since are human being are prone to errors and may give inaccurate results. Moreover it takes huge time for manual inspection of the code. Hence the overall cost incurred is more.

To perform static code analysis using an automated tool we need a tool that will scan the code and report all vulnerabilities. The accuracy of the tool can be better than the manual inspection. The cost of the tool is less and it will take less time to show errors.

II. LITERATURE SURVEY

A. Static Code Analysis

1) Author: Panagiotis Louridas

This paper discusses how common static code analyzers work. It describes various strategies involved in static code analysis. It explains abstract representations that can be used for analysis. Further the paper explains how dataflow analysis is used to monitor and track values of variables. At the end the paper gives comparative evaluation of static code analysis tools like FindBugs Checkstyle, Klocwork K7 and PMD.

B. JUnit: Unit Testing and Coding in Tandem

1) Author: Panagiotis Louridas

These paper discuses on comparison of various testing frameworks for Java such as TestNG, Jtest and JUnit. It describes their feature set, functionality and ability to produce detailed results. During the coding and testing process, we must make sure that our tests do not leave important parts of the code untested. JUnit helps write tests and create test sets. There are various metrics for test coverage, such as statement coverage or the line coverage, decision coverage (also known as branch coverage), path coverage, and others.

C. Static Analysis Tools for Security: A Comparative Evaluation

1) Author: Hanmeet Kaur Brar, Puneet Jai Kaur

This paper differentiates between three open source tools used for static analysis viz are Cppcheck, RATS, Flawfinder. The comparative evaluation is done based upon different parameters analyzed on executing demo codes with intentionally introduced vulnerabilities. Depending upon the time of execution of each tool, we found that RATS was the fastest among Flawfinder and Cppcheck. Cppcheck was slowest in execution of almost every application. Another comparison on the basis of category of vulnerabilities a tool can detect was done and it was found that Cppcheck was able to detect maximum categories of vulnerabilities introduced by us.

III. STUDY OF PROPOSED SYSTEM

By scrutinizing an application's source code without having to actually execute it, it is possible to find errors prior to production early in the development cycle. Static code analysis tool can help with the code review process by

1) Detecting areas in the code that need to be refactored and simplified.
2) Finding areas of the code which may need deeper review or more testing.
3) Identify design issues and help reduce the code complexity so as to improve maintainability.
4) Identify potential software quality issues before the code moves to production.

The proposed system can be used to scan the apex code and it will report vulnerabilities present in the software. The proposed system is helpful for the developer to scan their code and improve the code quality.

IV. SYSTEM ARCHITECTURE

There are 3 main components in our proposed system.

- Modeller
- Analyzer
- Reporter

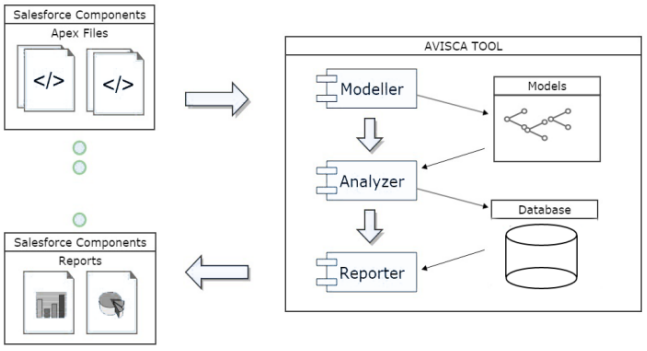

Fig. 1: System Architecture

A. Modeller

The modeller component reads the source code and creates intermediate analysis models such as symbol table, parse tree, Control flow graph, etc. An apex language parser is used to parse the source code. Here parsing involves identification of tokens and accordingly generating a parse tree. The parse tree is a tree representation of hierarchy of tokens. Furthermore this parse tree can be used to detect best practice violation, for instance if there's a SOQL statement nested in a loop structure. Also the parse tree is used to construct the symbol table.

The symbol table is a data structure that holds all the variables and methods declared in the source code. It helps to monitor status of variables. The symbol table also provides access specifiers of method and variables which can be further used for security checks. Another intermediate analysis model that we use is control flow graph. The control flow graph is a dynamic data structure where each of the method defined in the source code is represented as a graph node. The edges in the control flow graph represent a call made to another method from one of the method. This graph helps to identify loops in execution flow of the software. Similarly it also determines dependency of one method over another.

A more detailed control flow graph can be used which operates on blocks of code. This gives more deeper control over the flow analysis. It also helps in determining dead sections of code block.

B. Analyzer

Once the system is done with modeling of source code into intermediate models, the process of analysis starts. The analyzer component traverses each of the generated intermediate analysis models and identifies violation of best coding practices. The best coding practices may be hard coded into the code logic or they might be represented through use of regular expressions. These regular expressions will correspond to patterns in the analysis models. Accordingly whenever a match for a particular pattern is found, a violation is reported. Each of these violations are stored in a database for later use. The database entry of each violation consists of a violation code and the line number and column number where the violation was detected in the source code. Each of these violations are stored in a database for later use.

C. Reporter

Finally the reporter component generates reports from the database. The report may be a summarized visualization of database in the form of charts and graphs. These reports are used for statistics purpose and quality analysis of software product. So far we have all the violations listed in the database. The reporter component will fetch all entries from database and subsequently apply summarization logic to deduce high level conclusion of code quality or else it may be as simple as highlighting the line number in the source code where violation for a particular best coding practice was detected. This information can be used by software developer which will help him to optimize the code and correct it as per standards defined in best coding practices. In addition to normal reports, this component may also suggest solutions to detected violation thereby assisting the developer in software development process.

The summarized reports feature pie charts and bar graphs that show comparative analysis of particular subject where the source code is weaker. These reports can also help to get a total new perception of the problem statement so as to identify various strategies to solve the problem in terms of software code development.

V. MATHEMATICAL MODEL

Let S be the system object such that
S= {I, P, O}
I= Input
P= Process
O= output
I= {SRC}
SRC = Apex Source code
P= {MOD, IAM, ANL, DB, REP}
MOD = Modeller component
IAM = Intermediate Analysis Models
IAM = {SYM, PT, CFG}
SYM = Symbol Table
PT = Parse Tree
CFG = Control Flow Graph
ANL = Analyzer component
DB = Database
DB = {VCODE, LINE, TOKEN}
VCODE = Violation Code
LINE = Line Number
TOKEN = Token String
REP = Reporter component
O= {RES}
RES = get expected results.

VI. CONCLUSION AND FUTURE WORK

The purpose for this concept is to motivate developers to follow the best coding practices. Therefore using this system we can develop the software in better way. We can handle the abnormal runtime situation earlier at the time of coding.

Consequently a static code analysis tool may often produce false positive results where the tool reports a possible vulnerability that in fact is not. This often occurs because as the data flows through the application from input to output, the tool cannot be sure of the integrity and security of data. The use of static code analysis tools can also result in false negative results where vulnerabilities result but the tool does not report them. This might occur if a new vulnerability is discovered in an external component or if the analysis tool has no knowledge of the runtime environment and whether it is configured securely.

At current situation we just handle small number of violations. In future we will come with a large number of violations for detection. We will come with Dynamic rule violation detection where the developer can specify his own custom rule set of best coding practices by defining in terms of a regular expression.

ACKNOWLEDGEMENT

We want to thank the analysts and also distributers for making their assets accessible. We additionally thank commentator for their significant recommendations furthermore thank the school powers for giving the obliged base and backing.

REFERENCES

[1] Panagiotis Louridas, "JUnit: Unit Testing and Coding in Tandem"
[2] Panagiotis Louridas, "Static Code Analysis"
[3] Hanmeet Kaur Brar and Puneet Jai Kaur, "Static Analysis Tools for Security: A Comparative Evaluation"
[4] Suzanna Schmeelk, Bill Mills and Leif Hedstrom, "STANDARDIZING SOURCE CODE SECURITY AUDITS"
[5] Pixy: A Static Analysis Tool for Detecting Web Application

