



Analyze vulnerabilities of source codes published on open forums

A dissertation submitted for the Degree of Master of
Science in Information Security

S.T.S.T. Desapriya
University of Colombo School of Computing
2017



Declaration

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: S.T.S.T. Desapriya

Signature:

Date:

This is to certify that this thesis is based on the work of Mr. S.T.S.T. Desapriya under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name: Dr. Kasun De Zoysa

Signature:

Date:

Abstract

Web applications and mobile applications are extremely popular in the society and also became a part of the human lives. These applications are used by different institutions including governments for different purposes allowing them to access sensitive information and perform critical operations. Software developers are using many development languages to develop these applications by writing thousands of lines of code, with or without security in mind. Common practice among software developers is that they use open forums to share, suggest code examples and also to look for a suggestion for a problem they face or situation they need to address. Since these open forums are extremely popular among developer community, they tend to use those source examples, for the development of their applications. Because of that source code examples in open forums make direct impact on real world software application, for developers, it is important to have a method of verifying these source code samples and make sure they are free of security vulnerabilities before using.

Project aims to solve this problem by developing a simple, user friendly tool, which is capable of analyzing the security vulnerabilities of the source code samples published on open forums. The methodology used is, download large set of source code samples from an open forum, perform a static analysis using a reliable commercial tool, extract the results and create a knowledge-base of vulnerable source snippets, which can be used by the developed tool, to detect vulnerabilities of a particular source code block. Stackoverflow is selected as the open forum and five widely used programming languages, CSharp, Java, PHP, Python and JavaScript were selected for the analysis. Checkmarx is the static analysis tool selected. Over twenty-seven thousand source code samples used for the analysis and over thousand four hundred vulnerabilities detected by Checkmarx.

The Project delivers five main components. Python based crawler used to crawl through Stackoverflow and download source code samples. Data importer component, developed using csharp, used to import the results given by Checkmarx in to the knowledge base. Dashboard with various graphs and charts to show the results of the analysis is also developed using csharp. Chrome browser plugin, which is capable of analyzing a selected source code block, for potential vulnerabilities by referring the knowledge base, is developed as the tool. Finally, MS SQL server used to create the knowledge base which holds all the vulnerability data provided by Checkmarx.

The solution can influence the developers to write more secure code during the development of the project and also make them aware about the security vulnerabilities, which will ultimately make the software rugged. Project would be much more interest for those who involve in software development related areas and also for application security analysts who are interested and very keen on static analysis.

Acknowledgment

First and foremost, I would like to express my sincere thanks to my supervisor Dr. Kasun De Zoysa, senior lecture at University of Colombo School of Computing, for the enormous support, guiding and most importantly the encouragement given for the dissertation work.

Also, I would like to thank my panel members for giving me valuable comments and improvements points to make the dissertation a better one.

Specially, I would like to thank my family for the great support, encouragement, guidance and love given to me during this research work.

Special thank goes to Aaron Weaver, for spending his valuable time helping me and the support given with the commercial tool.

Finally, I would like to thank my working place, Pearson Lanka (pvt) Ltd. for all the opportunities given and making me a strong character in the industry.

Table of Contents

Declaration.....	i
Abstract.....	ii
Acknowledgment.....	iii
Table of Contents.....	iv
List of Figures.....	vii
List of Tables	x
Chapter 1 : Introduction.....	1
1.1 Motivation	1
1.2 Objective.....	4
1.3 Scope	5
Chapter 2 : Literature Review	6
2.1 Introduction	6
2.2 Application Security Vulnerabilities.....	6
2.2.1 OWASP Top 10.....	7
2.2.2 Cigital Top 20	15
2.2.3 SANS Top 25.....	18
2.3 Built-In Security	20
2.3.1 Microsoft Security Development Life cycle	21
2.3.2 Seven Security Touch Points Proposed by Gary McGraw	21
2.4 Static Analysis	22
2.4.1 Static Analysis Tools	26
2.4.2. Advantages and Disadvantages	46
2.5 Vulnerable Programming Languages	47
2.5.1. Top 10 Vulnerable Programming Languages	48
Chapter 3 : Design	51
3.1 Design Overview	51
3.2 System Overview.....	52
3.2.1 Web Crawler	53
3.2.2 Process Vulnerabilities and Store	54
3.2.3 Dashboard.....	55
3.2.4 Code Analysis Tool.....	56
3.2.5 Relational Database	57
3.2.6 Static Analysis Tool	58
3.2.7 Open Forum.....	58
Chapter 4 : Implementation	60
4.1 Implementation Overview	60

4.2 Source Samples	60
4.3 Programming Languages to Select	62
4.4 Source Samples Crawler.....	62
4.5 Static Source Analysis Tool	66
4.6 Vulnerability Importer	70
4.7 Dashboard.....	71
4.8 Code Analysis Tool.....	72
4.9 Vulnerability Database	78
Chapter 5 : Results, Testing and Evaluation.....	79
5.1 Introduction	79
5.2 Results	79
5.2.1 JavaScript	80
5.2.2 Python.....	81
5.2.3 PHP.....	82
5.2.4 Java	84
5.2.5 C-Sharp.....	86
5.3 Source Code Testing.....	87
5.4 Functional Testing	87
5.4.1 Testing Web Crawler	87
5.4.2 Vulnerability Exporter	88
5.4.3 Testing Dashboard	88
5.4.4 Testing Code Analyzer Tool	89
5.5 Usability Testing.....	94
Chapter 6 : Conclusions and Future Work.....	97
6.1 Introduction	97
6.2 Summary.....	97
6.3 Problems Faced	98
6.3.1 Crawling StackOverflow	98
6.3.2 Commercial Tool	99
6.3.3 False Positives	99
6.3.4 Browser Plugin	99
6.3.5 Usability Testing.....	99
6.4 Limitations.....	100
6.5 Extensions and Further Work	100
6.5.1 Fully Automate	100
6.5.2 Expand the Knowledge Base.....	101

6.5.3 Enhancements.....	101
6.5.4 Future Work.....	102
6.6 Critical Appraisal of the System.....	102
6.7 Final Conclusion.....	103
Appendix A : Development and Testing Environment.....	104
A.1 Hardware Requirements.....	104
A.2 Software Requirements.....	104
Appendix B : General Information.....	105
B.1 Execution of Web-Crawler.....	105
B.2 Browser Usage.....	106
B.3 Checkmarx Manual Verification.....	108
B.4 Checkmarx Reports.....	109
B.5 Chrome Extension.....	110
B.6 CodePlex FuzzyString.....	111
B.7 Google Trends.....	112
Appendix C : Project Source Code.....	113
C.1 Project Structures.....	113
C.2 Helpful Comments.....	113
Appendix D : Testing the Application.....	115
Appendix E : Dashboard Options.....	118
References.....	119

List of Figures

Figure 1.1 : Monthly Stack Overflow Visits - Geographically.....	2
Figure 1.2 : Most Popular Technologies - Full Stack	3
Figure 1.3 : Age Groups	3
Figure 1.4 : Job Titles	4
Figure 2.1 : OWASP Top 10 Vulnerabilities.....	8
Figure 2.2 : Sample Attack - Injections	9
Figure 2.3 : Sample Attack - Broken authentication & Session management.....	10
Figure 2.4 : Sample Attack - Cross site scripting (XSS)	11
Figure 2.5 : Sample Attack - Insecure direct object reference	12
Figure 2.6 : Sample Attack - Missing functional level access control	13
Figure 2.7 : Sample Attack - Cross site request forgery.....	14
Figure 2.8 : OWASP Top 10 Vulnerabilities - Mobile.....	15
Figure 2.9 : Cigital Top 20 Vulnerabilities	16
Figure 2.10 : Cigital - Data collection methodology	17
Figure 2.11 : Comparison - Cigital Top 20 vs OWASP Top 10.....	18
Figure 2.12 : Insecure Interaction Between Components - Vulnerabilities.....	19
Figure 2.13 : Risky Resource Management - Vulnerabilities.....	19
Figure 2.14 : Porous Defenses - Vulnerabilities	20
Figure 2.15 : Microsoft Security Development Life cycle.....	21
Figure 2.16 : Software Security Touch points	22
Figure 2.17 : Traditional quality assurance model vs Attacker	23
Figure 2.18 : Data flow Analysis	23
Figure 2.19 : Control flow Graph	24
Figure 2.20 : Taint Analysis.....	25
Figure 2.21 : Lexical Analysis	25
Figure 2.22 : VisualCodeGrepper V2.1.0	28
Figure 2.23 : YASCA.....	29
Figure 2.24 : OWASP LAPSE+.....	30
Figure 2.25 : RIPS	31
Figure 2.26 : DevBug	32
Figure 2.27 : Flawfinder	33
Figure 2.28 : CPPCheck	34
Figure 2.29 : Brakeman	35
Figure 2.30 : Brakeman users	35
Figure 2.31 : IBM AppScan source	37
Figure 2.32 : Fortify Static Code Analyzer.....	38
Figure 2.33 : Veracode application security platform.....	39
Figure 2.34 : Languages supported by Veracode.....	39
Figure 2.35 : WhiteHat Sentinel Source	40
Figure 2.36 : WhiteHat Security Customers.....	41
Figure 2.37 : Checkmarx Source Code Analysis Tool.....	42
Figure 2.38 : Checkmarx Supported Programming Languages	43
Figure 2.39 : Open Source Components getting in every Angle.....	44
Figure 2.40 : Black Duck Hub.....	45

Figure 2.41 : WhiteSource.....	45
Figure 2.42 : WhiteSource, Managing Policies.....	46
Figure 2.43 : Policy compliance by programming language.....	49
Figure 2.44 : Comparison of critical vulnerability types.....	49
Figure 2.45 : Dynamic vs. static application security testing.....	50
Figure 3.1 : System Overview.....	53
Figure 3.2 : Sample crawler using Scrapy.....	54
Figure 3.3 : Process Vulnerabilities and Store.....	55
Figure 3.4 : Dashboard - Process Vulnerabilities and Display Charts.....	56
Figure 3.5 : Code Analysis Tool.....	57
Figure 3.6 : Database diagram.....	57
Figure 3.7 : Table used for graphs and charts.....	58
Figure 3.8 : Static code analysis tool.....	58
Figure 3.9 : Crawl Open Forum and store source samples locally.....	59
Figure 4.1 : StackOverflow sample question.....	61
Figure 4.2 : StackOverflow sample answer.....	61
Figure 4.3 : StackOverflow Posted question URL format.....	62
Figure 4.4 : Sample source code file.....	66
Figure 4.5 : Running the web crawler.....	66
Figure 4.6 : Sample source code files are ready to upload.....	67
Figure 4.7 : Checkmarx upload zip file for scan.....	67
Figure 4.8 : Checkmarx scan queue.....	67
Figure 4.9 : Project overview.....	68
Figure 4.10 : Issue viewer.....	68
Figure 4.11 : Import vulnerabilities to a report.....	69
Figure 4.12 : Imported vulnerabilities to XML documents.....	69
Figure 4.13 : Vulnerability report importer.....	70
Figure 4.14 : Top 5 Vulnerabilities.....	71
Figure 4.15 : Vulnerabilities by Platform.....	72
Figure 4.16 : Source Analyzer Chrome Plugin.....	73
Figure 4.17 : Source Analyzer tool.....	77
Figure 4.18 : Database implementation.....	78
Figure 5.1 : Risk level indicator of each language.....	80
Figure 5.2 : Vulnerability categories - JavaScript.....	80
Figure 5.3 : Top 5 vulnerability categories – JavaScript.....	81
Figure 5.4 : Vulnerability categories - Python.....	81
Figure 5.5 : Top 5 vulnerability categories - Python.....	82
Figure 5.6 : Vulnerability categories - JavaScript within PHP.....	83
Figure 5.7 : Top 5 vulnerability categories - PHP.....	83
Figure 5.8 : Vulnerability categories - PHP.....	84
Figure 5.9 : Vulnerability categories - Java.....	85
Figure 5.10 : Top 5 vulnerability categories - Java.....	85
Figure 5.11 : Vulnerability categories - CSharp.....	86
Figure 5.12 : Top 5 vulnerability categories - CSharp.....	86
Figure 5.13 : Open vulnerabilities by severity.....	88
Figure 5.14 : SQL query results.....	89
Figure 5.15 : Unit test method 001.....	90
Figure 5.16 : Unit test method 002.....	90
Figure 5.17 : CSharp vulnerabilities summery of the Knowledge-base.....	91

Figure 5.18 : Checkmarx results of new source samples	93
Figure 5.19 : Usability feedback form.....	95
Figure 5.20 : Expert feedback on accuracy of source analyzer	96
Figure B.1 : Above to run the web crawler	105
Figure B.2 : After running the web crawler.....	106
Figure B.3 : Browser Usage 2009 - 2016	107
Figure B.4 : Browser market map - 2015	107
Figure B.5 : Mark vulnerabilities as false positive.....	108
Figure B.6 : Manually set the severity of a vulnerability	109
Figure B.7 : Checkmarx imported XML report.....	109
Figure B.8 : Chrome extension files.....	110
Figure B.9 : Source Analyzer Chrome extension	110
Figure B.10 : FuzzyString algorithms	111
Figure B.11 : FuzzyString compare two strings	111
Figure B.12 : Interest over time.....	112
Figure B.13 : Interest by region.....	112
Figure C.1 : Project structures	113
Figure C.2 : Project comments	114
Figure E.1 : Chart options of the Dashboard.....	118

List of Tables

Table 2.1 : Well-known security research organization.....	7
Table 2.2 : SANS vulnerability categories	18
Table 2.3 : Selecting an open-source static analysis tool - Points to Consider.....	27
Table 2.4 : VisualCodeGrepper - Advantages and Disadvantages	27
Table 2.5 : YASCA - Advantages and Disadvantages	28
Table 2.6 : OWASP LAPSE+ - Advantages and Disadvantages	29
Table 2.7 : RIPS - Advantages and Disadvantages.....	30
Table 2.8 : DevBug - Advantages and Disadvantages.....	31
Table 2.9 : Flawfinder - Advantages and Disadvantages.....	32
Table 2.10 : CPPCheck - Advantages and Disadvantages.....	33
Table 2.11 : Brakeman - Advantages and Disadvantages.....	34
Table 2.12 : Points to consider with open source static analysis tools - Checkmarx	36
Table 2.13 : IBM AppScan Source capabilities	37
Table 2.14 : Fortify Static Code Analyzer benefits	38
Table 2.15 : Checkmarx static analysis tool benefits.....	42
Table 2.16 : Black Duck Hub capabilities	44
Table 3.1 : Components of proposed system.....	52
Table 5.1 : False Positive and Negative Percentages.....	89
Table 5.2 : Expected outcome the Tool should provide.....	91
Table 5.3 : Expected results vs Actual results	92
Table 5.4 : Expected outcome vs Actual outcome - CSharp	92
Table 5.5 : Results comparison - Unknown source samples	93
Table 5.6 : Test results summery	94
Table D.1 : Detailed results comparison.....	116
Table D.2 : Ideas to Improve	117

Chapter 1 : Introduction

1.1 Motivation

Web and mobile applications are part of human lives in the present days which are dealing with highly sensitive data and operations. Because of the critical and the integrated nature of those applications, they have become the primary targets of attacks. New security vulnerabilities are discovered every day in those commonly used applications which creates a huge security threat on end users. Which makes the point that these applications must be rugged and should be able to stand against malicious attacks.

Each and every software created using thousands of lines of code, which called as source code, to make it functional. It is very clear that the source code plays important role in terms of application security and need to make sure the source code followed security best practices in order to assure application security.

Industry is extremely focused on S-SDLC (Secure Software Development Life cycle) which is focusing on building security into the development life cycle which can assure the security of end product. Analyzing source code which is called static code analysis, commonly referred to, scanning the source code using a static analysis tool to identify potential security vulnerabilities is a part of S-SDLC [10]. Static analysis of the source code can identify potential security vulnerabilities during the development phase of the software, so that the developer can take necessary actions to eliminate these issues then and there.

Software developers involved in writing these source codes, according to the functional specifications of the software. During the day to day development, developers tend to surf web and refer freely available source code examples to solve their problems or to enhance their source code. These open forums are extremely popular among developer community and thousands of developers around the world refer these contents and also share their knowledge and views on published answers. The beauty of these open forums is, anyone can freely join and start sharing suggestions to a particular problem with their own source code samples.

By analyzing some of the open forum statistics, it's very clear how popular and how actively developers are using them. According to StackOverflow, one of the most famous open forum among developers, about 32 million people visit them every month and more than 25 million are return visitors. And return visitors visits the site 6 times every month. In January 2016, 46 million visited StackOverflow and they believe 16 million of those are professional developers [15]. Below chart shows the monthly visits of StackOverflow, geographically.



Figure 1.1 : Monthly Stack Overflow Visits - Geographically

It is also important to see what are the popular programming languages among the developer community. This information can greatly help to understand what are the highly demanded programming languages and also it is critical to make sure that the source code samples under these highly demanded programming languages are secured or security best practices are followed. So according to the StackOverflow statistics, JavaScript, CSharp .Net, PHP and Java has a high demand and all these are famous web development programming languages.

Analyzing the usage based on the job title also very important, since developers or programmers are the people who actually do code to implement the organization's product. According to StackOverflow developers used the site most and it is clear that developers spend more time looking for source samples to solve the issues or looking for new ideas. Below charts shows the most popular technologies among the community.

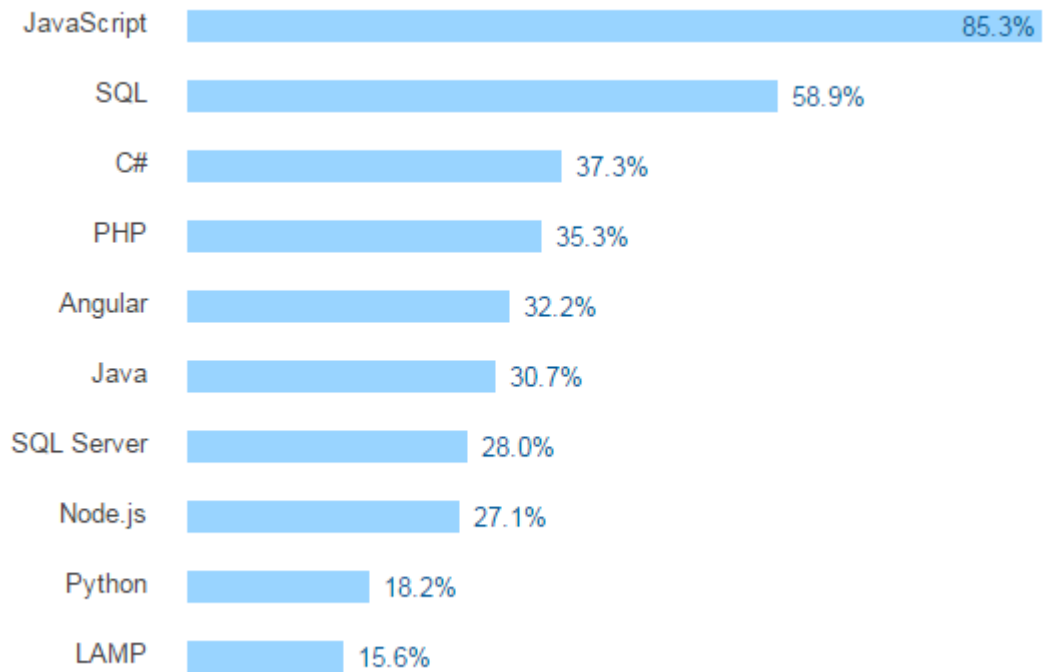


Figure 1.2 : Most Popular Technologies - Full Stack

Also, it is important to understand what are age groups which heavily using StackOverflow, since it is possible to understand, what sort of experienced developers asking questions and posting answers. Understand the job roles of the people who frequently access the site is also important, to get a clear understanding on who are really using the resources for what purpose. Below charts shows the StackOverflow usage statistics by age-groups and the job titles.

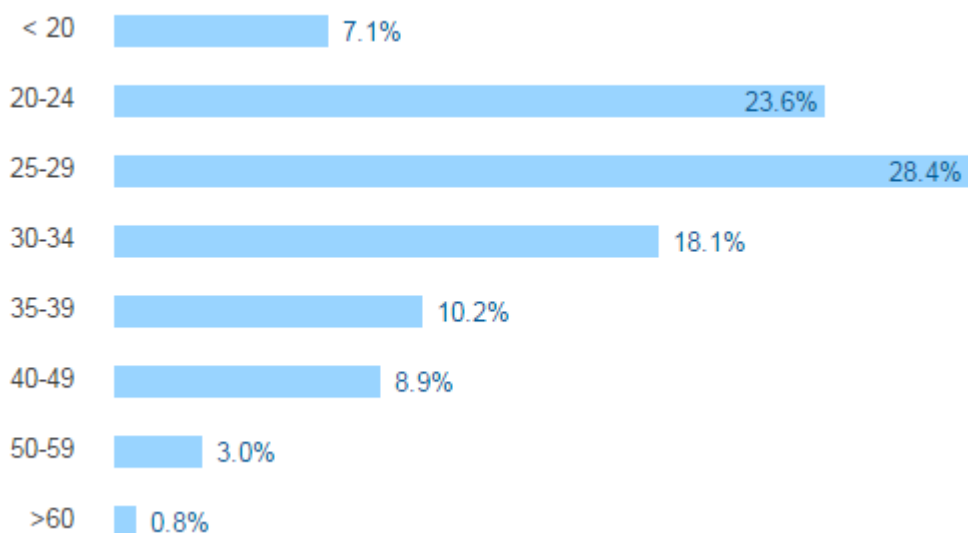


Figure 1.3 : Age Groups

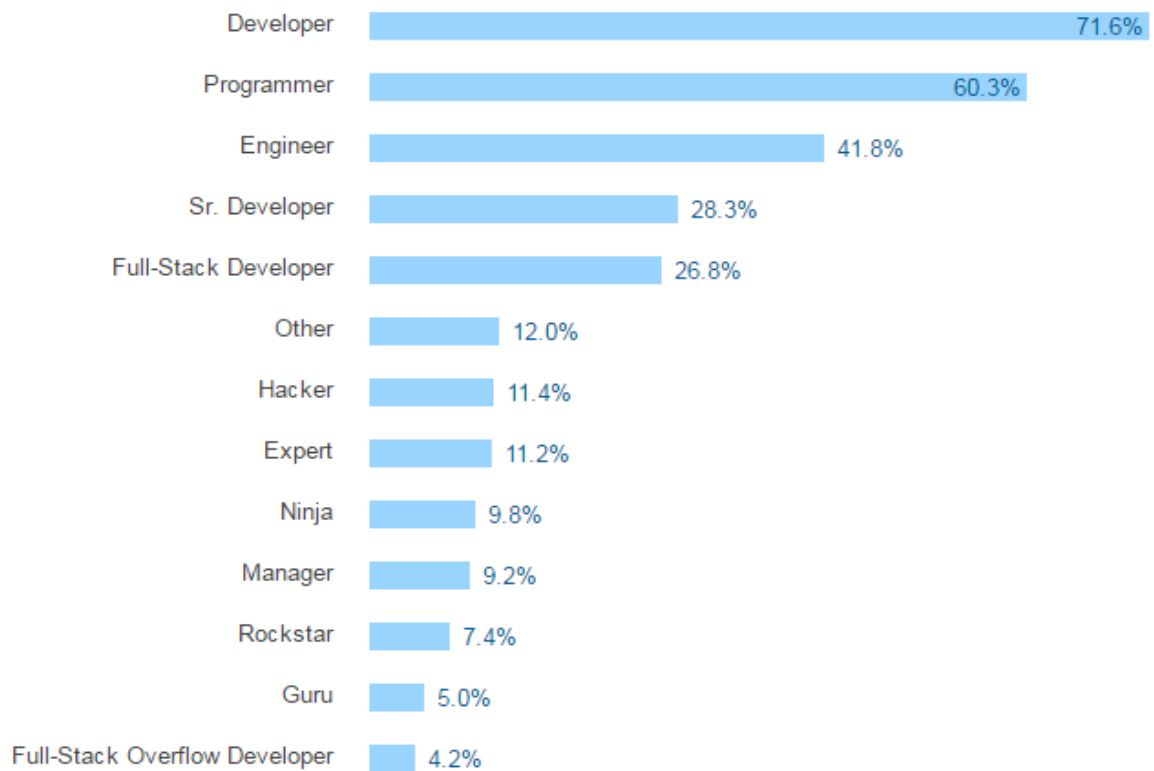


Figure 1.4 : Job Titles

By looking at these statistics, it is pretty clear that, almost all the developers are using open forums during their development work. And these open forums provide solutions for almost all the problems and situations that any developer could face during their developments, they tend to use these source code samples directly or indirectly for the software that they develop. This is a critical situation, since the source code samples in open forums can make an impact on enterprise level software and there is a need to make sure that these source code samples do not introduce any security vulnerabilities or to make sure these code samples follow recommended security best practices by the industry.

1.2 Objective

Currently, there is no easy way of verifying whether the source code samples in open forums are vulnerability free or followed security best practices. This will make the developer use these samples without verifying and indirectly making the software vulnerable. Manual verification is a possibility but it is time consuming. The number of developers using these open forums are increasing day by day and there is an indeed requirement to have an easy solution to quickly verify these source code examples before using within the software that the developers are developing.

The objective of this project is to provide a solution for this requirement by providing an easy and convenient solution. With this project source code samples published in open forums will be gathered using a software crawler, perform a static code analysis using a sophisticated commercial tool and store the results as a knowledge base. Also, a user friendly and easy to use tool will be developed for developers to analyze a particular source code block which is published in an open forum, using the gathered knowledge base. A sophisticated dashboard with a holistic view of the analysis of open forum source samples, also will be developed, so that the development community can get a better understanding of what sort of security vulnerabilities are exists with source codes published in open forums. Also, ultimately the solution of the project will support to make the developing software more secured.

1.3 Scope

Based on the usage statistics, among professional developers, StackOverflow is the most famous and highly used open forum among the available forms. This project will be focused on source code samples published on StackOverflow forum only. Also, project will be only focused on source code samples written in most famous back end development languages. To perform the static code analysis, to create the knowledge base, commercial static analysis tool named Checkmarx will be used. Also, the vulnerabilities will be limited to most critical vulnerabilities related to web and mobile application. To develop the web crawler to gather the source code sample, python will be used and a python based web crawling framework will be used. The planned implementation has two main components, the dashboard and the source analysis tool. Microsoft CSharp .Net, will be used to develop the dashboard and the back-end of the source analysis tool. A browser plugin will be developed for a one particular browser to implement the front-end of the source analysis tool.

Chapter 2 : Literature Review

2.1 Introduction

In recent years, web and mobile application become very close to human lives and started playing important role of their day to day life. Attractiveness and easiness of these application made them so popular and commercial industries and governments start to leverage those within their respective areas. At present, there are thousands of web and mobile application used by almost all commercial industries and governments, maintaining highly sensitive information including government secrets, trade secrets, and also performing critical operations such as stock market activities and online money transfer. There is a definite need to make sure that these applications are fully secured and capable of stand against any malicious activity.

Although that there is a definite need to make sure these software applications are holistically secured, it is difficult because these applications are, by definition, exposed to the general public, including malicious users [1]. Every application developed with thousands of lines of code using one or many available technologies by human developers. To achieve better security for an application, organizations need to make a considerable investment to make sure the required level of security is achieved. Due to the highly competitive nature of the business, organizations are reluctant to invest money and time on security, because most of the time, security professionals cannot justify the investment or cannot define a clear return on investment.

Historically, applications security considers as an afterthought, and industries gave priority to user friendliness and the performance of the application. Over the time this makes a huge and complicated application with many security vulnerabilities, which make it extremely expensive, difficult or impossible to address, mainly because the application is already in production and has a large customer base.

This chapter explain and analyze the top and critical security vulnerabilities identified by the industry, which are exists in most of the common web applications. And explain the importance of built in security and available frameworks. Also discuss about the importance of analyzing the source code for potential security vulnerabilities, what are the available tools and techniques and common advantages and disadvantages of using the tools.

2.2 Application Security Vulnerabilities

It is the nature that any software contains issues or defects. These can be functional flaws, architectural defect, performance issues, usability issues and so on. Functional issues can be verified and easily detectable based on the required use-cases. And most of the time these issues are identified and fixed without any hassle. Security vulnerabilities on the other hand, very hard to detect, because none of the functional use cases will cover those scenarios or the steps. Use cases have become popular for demonstrating, communicating and defining the software

requirements. They demonstrate the functional requirements of the application well, but provide less support for extra-functional requirements, such as security requirements. With the increase of the usage of e-commerce and m-commerce applications, such requirements are growing in importance [3].

Web and mobile applications are facing various attacks each and every day. When considering the top critical web application vulnerabilities, it is clear that, somewhat poor programming approach which leads to these vulnerabilities [2]. That make the developers are responsible for these vulnerabilities. There are various web and mobile applications related vulnerabilities exists in the present. Also new vulnerabilities are discovered by attackers very frequently. New technologies like cloud infrastructure, new programming languages changes the threat landscape and create new attack vectors. This situation make security more complicated and bizarre for the organizations and make it easier to the attackers. Since the situation is getting worst day by day, it would be nice to have independent body or organization who can invest on researching on new threats, vulnerabilities, define the severity of the vulnerabilities and define guidelines and best practices to avoid, address these vulnerabilities. Also, they can suggest required and best security solutions, providers and necessary tools. Then the organizations can get a clear idea about the top vulnerabilities exists and take necessary actions like, educate the engineers, focus on test cases to cover necessary scenarios. This will be a great advantage since it can save considerable resources for an organization. Couple of well-known independent foundations or organizations are exists, performing security related researches and doing a great help for businesses as well as the community. Below are some of them.

Open Web Application Project (OWASP)
Cigital
SANS

Table 2.1 : Well-known security research organization

All the above organizations define their identified top vulnerabilities, root cause of those vulnerabilities and set of guideline and best practices to fix or avoid those vulnerabilities.

2.2.1 OWASP Top 10

OWASP, the Open Web Application Security Project is an unbiased, independent foundation came online in December 2001 and also it is a not for profit organization with variety of security experts from around the world. They perform researches to identify various threats, vulnerabilities, risks related to web applications and provide state of the art solutions to address them. They also implement application verification standard which will help for an organization to understand the security risk level or the compliance level of a web or mobile application. With application verification standards, OWASP defines three levels under application verification standard and each level has number of criteria that the application needs to fulfill in order to achieve a particular level. Each criterion defines certain test or verification scenarios

to see whether the application is meeting that criteria. When considering the application security, OWASP application verification standards can create a considerable impact.

OWASP is doing a great help for the community by providing various learning materials and implementing security libraries to protect applications from vulnerabilities. Among the tools, Zed Attack Proxy, a tampering proxy which can analyze security vulnerabilities, is very famous among security professionals. OWASP doing researches on web applications as well as mobile application and they came with top 10 most critical issues that the web applications and the mobile applications are facing by doing an independent research. Many organizations refer the OWASP recommendations because they give unbiased, practical and cost-effective solutions for application security. Below are the top ten issues for web applications identified by OWASP [4].



Figure 2.1 : OWASP Top 10 Vulnerabilities

When analyzing these top ten issues, it is clear that more than six issues are related to poorly written source code and developers are responsible to those vulnerabilities. Some of the most critical vulnerabilities related to poorly written source codes are as follows.

A1. Injections

Injection flaws, such as SQL, OS, and LDAP injection occur when malicious user input is taken as a parameter by the application plug it as part of a command or query. The attacker's malicious input can trick the interpreter into executing unwanted commands or providing access to the

application without proper authorization [4]. It is clear that this issue exists due to the source code fails to validate the data sent by the user or attacker.

Example Attack Scenarios

Scenario #1: The application uses untrusted data in the construction of the following **vulnerable** SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in her browser to send: ' or '1'=1. For example:

```
http://example.com/app/accountView?id=' or '1'=1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify data or even invoke stored procedures.

Figure 2.2 : Sample Attack - Injections

A2. Broken Authentication & Session Management

Authentication and session management is a key and fundamental concept to manager the user access properly with in the application. Sometimes application fails to manage the authentication properly and attacker will be able to by-pass the login or impersonate another a user by hijacking or predicting session tokens. [4]. Once again, source code fails to validate whether the user is authenticated and has a valid session and also whether the user is allowed to perform a particular action.

Example Attack Scenarios

Scenario #1: Airline reservations application supports URL rewriting, putting session IDs in the URL:

```
http://example.com/sale/saleitems?  
sessionid=268544541&dest=Hawaii
```

An authenticated user of the site wants to let his friends know about the sale. He e-mails the above link without knowing he is also giving away his session ID. When his friends use the link they will use his session and credit card.

Scenario #2: Application's timeouts aren't set properly. User uses a public computer to access site. Instead of selecting "logout" the user simply closes the browser tab and walks away. Attacker uses the same browser an hour later, and that browser is still authenticated.

Scenario #3: Insider or external attacker gains access to the system's password database. User passwords are not properly hashed, exposing every users' password to the attacker.

Figure 2.3 : Sample Attack - Broken authentication & Session management

A3. Cross Site Scripting (XSS)

XSS attack is all about application fails to validate or properly escape user input data and echoed the malicious user inputs into the browser. Attacker can enter malicious JavaScript code to get execute on the victim's browser to steal some sensitive data or even possible to install a key-logger to record everything and send back to the attacker [4]. Another example of not validating or escaping the user supplied content with the source code. This is a classic example of poorly written source code and a critical also a very common issue.

Example Attack Scenarios

The application uses untrusted data in the construction of the following HTML snippet without validation or escaping:

```
(String) page += "<input name='creditcard' type='TEXT'  
value='" + request.getParameter("CC") + "'>";
```

The attacker modifies the 'CC' parameter in their browser to:

```
'><script>document.location= 'http://www.attacker.com/cgi-  
bin/cookie.cgi ?foo='+document.cookie</script>'
```

This causes the victim's session ID to be sent to the attacker's website, allowing the attacker to hijack the user's current session.

Note that attackers can also use XSS to defeat any automated CSRF defense the application might employ. See A8 for info on CSRF.

Figure 2.4 : Sample Attack - Cross site scripting (XSS)

A4. Insecure Direct Object References

If the application allows to access any internal resources such as files, objects or data belongs to other users, usually using an internal reference key, without any authorization check, then the application is vulnerable for direct object reference attacks [4]. With this issue attacker will be able to access or even destroy unauthorized data. This a classic example of an extremely poor coding practices and failed validation in the source code, and the consequences of this issue is much severe.

Example Attack Scenarios

The application uses unverified data in a SQL call that is accessing account information:

```
String query = "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt = connection.prepareStatement(query
, ... );
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

The attacker simply modifies the 'acct' parameter in their browser to send whatever account number they want. If not verified, the attacker can access any user's account, instead of only the intended customer's account.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Figure 2.5 : Sample Attack - Insecure direct object reference

A7. Missing Function Level Access Control

Authorization is critical for any application to make sure the particular user is allowed to perform the requested action. Most of the time application verifies the access levels during the start of the application for the main UI, but sometimes misses these necessary access verifications for some features inside the application. Always the application should make sure it verifies the required access verification and authorized the user correctly. [4]. Common mistake that most of the software developers and other related professionals have their mind is, only the UI validations are enough and those cannot be bypassed. But in reality, attacker can simply alter and by pass UI validations with a tampering proxy. It is always necessary to have and in this case, it is failed to implement server side validations within the source code.

Example Attack Scenarios

Scenario #1: The attacker simply force browses to target URLs. The following URLs require authentication. Admin rights are also required for access to the `admin_getappInfo` page.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, that's a flaw. If an authenticated, non-admin, user is allowed to access the `admin_getappInfo` page, this is also a flaw, and may lead the attacker to more improperly protected admin pages.

Scenario #2: A page provides an 'action' parameter to specify the function being invoked, and different actions require different roles. If these roles aren't enforced, that's a flaw.

Figure 2.6 : Sample Attack - Missing functional level access control

A8. Cross-Site Request Forgery (CSRF)

If the victim logged in to a particular application, then the attacker can forcefully send forged HTTP request to the application, using victim's browser, if that application is vulnerable for CSRF attacks. The forged request will automatically include session and authentication tokens, since the victim is already logged in and application will trust the request [4]. With this issue, the application is failed to validate a particular request made by the user's browser is legitimate or not. Bit tricky to launch an attacker using this issue, but if the application is vulnerable for CSRF, it is a lethal weapon an attacker can used against the application.

Example Attack Scenarios

The application allows a user to submit a state changing request that does not include anything secret. For example:

```
http://example.com/app/transferFunds?  
amount=1500&destinationAccount=4673243243
```

So, the attacker constructs a request that will transfer money from the victim's account to the attacker's account, and then embeds this attack in an image request or iframe stored on various sites under the attacker's control:

```

```

If the victim visits any of the attacker's sites while already authenticated to example.com, these forged requests will automatically include the user's session info, authorizing the attacker's request.

Figure 2.7 : Sample Attack - Cross site request forgery

According to these OWASP Top vulnerabilities, most of the issues are exits because of the poorly written source codes. Leveraging these vulnerabilities, an attacker can make severe damage to the application, which may destroy the organization as well. In order to make sure the application is secure enough to stand against malicious attacks, it is necessary to make sure the source code is well written and followed all the required security best practices.

Below are the top ten issues for mobile applications identified by OWASP [19].

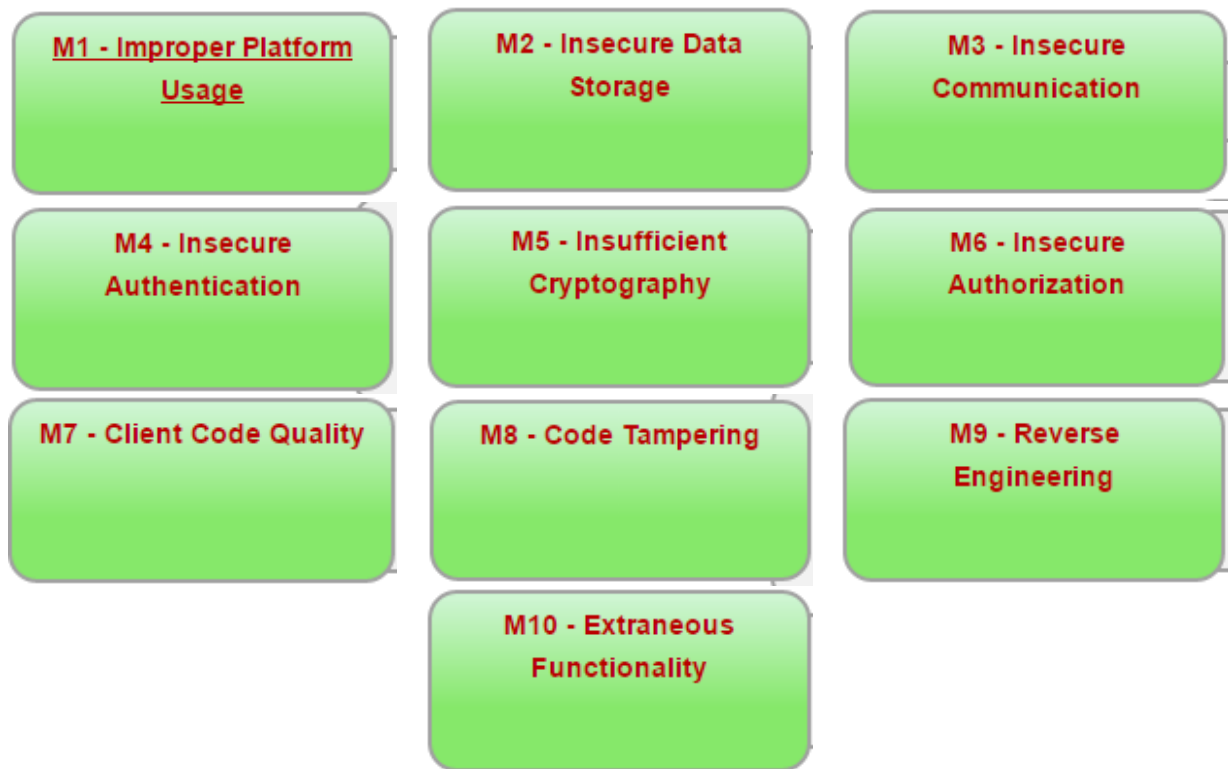


Figure 2.8 : OWASP Top 10 Vulnerabilities – Mobile

2.2.2 Cigital Top 20

Cigital is one of the largest application security firm in the world and helps to identify the application related security vulnerabilities. Cigital experts also provide guidelines, best practices to re-mediate the application security vulnerabilities and most importantly they provide user training on application security and related areas for developer, quality engineer and other related positions like business analysts, architects and project managers. They have identified the top twenty vulnerability list that they think which are more important to pay attention by the organizations and engineers. Below is the list of top twenty vulnerabilities identified by Cigital [17].

1	Verbose server banner	8%
2	Weak SSL ciphers	6%
3	Hidden directory detected	6%
4	Clickjacking (aka UI Redressing)	5%
5	Weak password policy	5%
6	Secure cookie attribute not set	5%
7	Cacheable SSL pages	4%
8	SSL/TLS beast information leakage	4%
9	Username enumeration through password reset	3%
10	Reflected cross-site scripting (XSS)	3%
11	HttpOnly cookie attribute not set	3%
12	Verbose error messages	2%
13	Unencrypted viewstate	2%
14	Cross-site request forgery (CSRF)	2%
15	TLS/SSL not enforced	2%
16	Sensitive information leaked via query string parameter	2%
17	TLS/SSL not enabled	2%
18	Application error	2%
19	No account lockout policy	2%
20	Session identifier set prior to authentication	2%

Copyright © 2016, Cigital



Figure 2.9 : Cigital Top 20 Vulnerabilities

The methodology of identifying the top vulnerabilities used by Cigital also very interesting one. Below diagram shows the methodology following by Cigital [17].

Data collection methodology

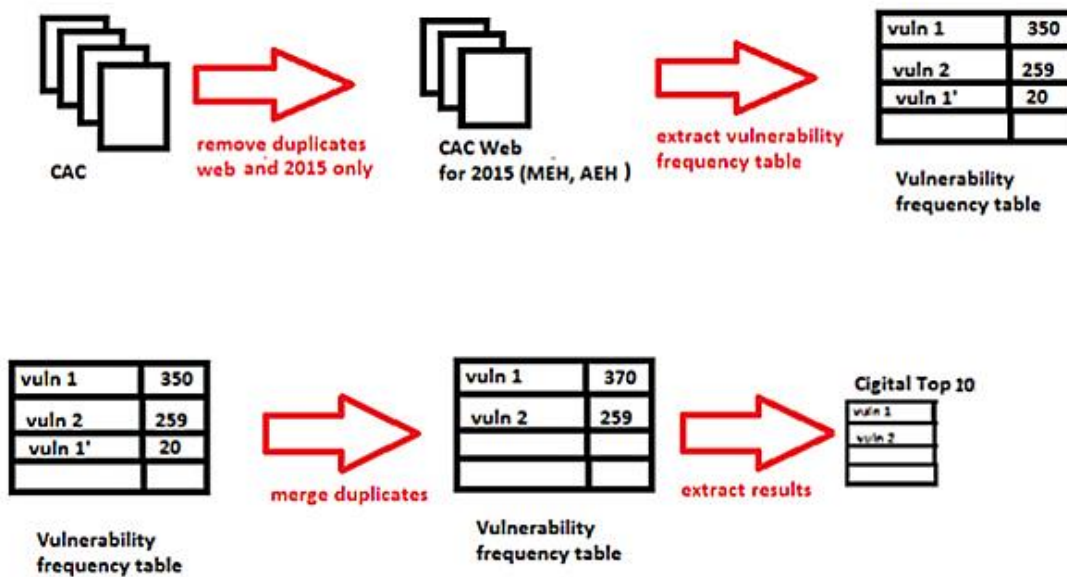


Figure 2.10 : Cigital - Data collection methodology

Below table shows the comparison of OWASP top 10 and the Cigital top 20 vulnerabilities [17].

Comparison to OWASP Top 10

OWASP Top 10	Cigital Top 20 Web	Comparable OWASP Ref.
A1-Injection	Verbose server banner	A5-Security Misconfiguration
A2-Broken Authentication and Session Management	Weak SSL ciphers	A6-Sensitive Data Exposure
A3-Cross-Site Scripting (XSS)	Hidden directory detected	A4 Insecure Direct Object References
A4-Insecure Direct Object References	Clickjacking (aka UI Redressing)	(none)
A5-Security Misconfiguration	Weak password policy	A2-Broken Authentication and Session Management
A6-Sensitive Data Exposure	Secure cookie attribute not set	A6-Sensitive Data Exposure
A7-Missing Function Level Access Control	Cacheable SSL pages	A6-Sensitive Data Exposure
A8-Cross-Site Request Forgery (CSRF)	SSL/TLS beast information leakage	A6-Sensitive Data Exposure
A9-Using Components with Known Vulnerabilities	Username enumeration through password reset	A2-Broken Authentication and Session Management
A10-Unvalidated Redirects and Forwards	Reflected cross-site scripting (XSS)	A3-Cross-Site Scripting (XSS)



The Next 10

Cigital 10-20	Comparable OWASP Ref.
HttpOnly cookie attribute not set	A6-Sensitive Data Exposure
Verbose error messages	A5-Security Misconfiguration
Unencrypted viewstate	A5-Security Misconfiguration
Cross-site request forgery (CSRF)	A8-Cross-Site Request Forgery (CSRF)
TLS/SSL not enforced	A6-Sensitive Data Exposure
Sensitive information leaked via query string parameter	A6-Sensitive Data Exposure
TLS/SSL not enabled	A6-Sensitive Data Exposure
application error	A5-Security Misconfiguration
No account lockout policy	A2-Broken Authentication and Session Management
Session identifier set prior to authentication	A2-Broken Authentication and Session Management



Copyright © 2016, Cigital



Figure 2.11 : Comparison - Cigital Top 20 vs OWASP Top 10

2.2.3 SANS Top 25

SANS is a well-known organization for cooperative research and education, which was established in year 1989. They have range of individuals from each and every job category in information security industry and also from members from around the globe. Also, SANS is an award-winning security research firm holding more than 1200 award winning research papers. On the other hand, SANS is the most trusted information security training and certifications provider in the world. SANS came up with list of twenty-five security vulnerabilities named as SANS TOP 25 Most Dangerous Software Errors. SANS identified these list under three categories as below [18].

Software Error Category: Insecure Interaction Between Components (6 errors)
Software Error Category: Risky Resource Management (8 errors)
Software Error Category: Porous Defenses (11 errors)

Table 2.2 : SANS vulnerability categories

Insecure Interaction Between Components

This category talks about the insecure way of sending data between application components, modules or systems.

CWE ID	Name
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
CWE-434	Unrestricted Upload of File with Dangerous Type
CWE-352	Cross-Site Request Forgery (CSRF)
CWE-601	URL Redirection to Untrusted Site ('Open Redirect')

Figure 2.12 : Insecure Interaction Between Components - Vulnerabilities

Risky Resource Management

This category covers the vulnerabilities related to, not managing the life cycle of the application including creation, transfer and destruction.

CWE ID	Name
CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
CWE-494	Download of Code Without Integrity Check
CWE-829	Inclusion of Functionality from Untrusted Control Sphere
CWE-676	Use of Potentially Dangerous Function
CWE-131	Incorrect Calculation of Buffer Size
CWE-134	Uncontrolled Format String
CWE-190	Integer Overflow or Wraparound

Figure 2.13 : Risky Resource Management - Vulnerabilities

Porous Defenses

This category covers the vulnerabilities related to the misuse of the protective activities of the application like encryption or authorization.

CWE ID	Name
CWE-306	Missing Authentication for Critical Function
CWE-862	Missing Authorization
CWE-798	Use of Hard-coded Credentials
CWE-311	Missing Encryption of Sensitive Data
CWE-807	Reliance on Untrusted Inputs in a Security Decision
CWE-250	Execution with Unnecessary Privileges
CWE-863	Incorrect Authorization
CWE-732	Incorrect Permission Assignment for Critical Resource
CWE-327	Use of a Broken or Risky Cryptographic Algorithm
CWE-307	Improper Restriction of Excessive Authentication Attempts
CWE-759	Use of a One-Way Hash without a Salt

Figure 2.14 : Porous Defenses - Vulnerabilities

2.3 Built-In Security

Web and mobile applications are in a critical state where attackers are primarily targeting them mainly due to its nature. Every day the risk is increasing, new attack vectors are identified, new vulnerabilities are discovered and organizations has to be alert all the time and monitor their web application for anomalies. Historically, the software development life cycle did not consider about the security. Everyone believed that the security is something we can add as a feature, when the application is ready and it is all about tools like firewalls, Intrusion detection or prevention systems, for example. But because of the growing demand, there is a need of build the security into the development life cycle, where the necessary controls added and actions have been taken in every phase of the development life cycle to make sure the end product is secured and rugged. The new methodology is named as Secure Software Development Life Cycle - SSDLC. Industry came up with couple of methodologies for secure software development life cycle. Microsoft Security Development Life cycle [5] and Cigital Seven Security Touch Points Proposed by Gary McGraw are the most recognized methodologies within the software industry.

2.3.1 Microsoft Security Development Life cycle

Today's cyber security threats are complex, sophisticated, and ever-changing. They require an ongoing, multifaceted response from the information technology industry for development solutions that optimize software security and provide for safer computing experiences for people around the world. The Microsoft Security Development Life cycle (SDL) is Microsoft's security assurance process for software development that introduces security and privacy at every step of the way. It offers a holistic and practical approach to addressing evolving security threats and increasingly sophisticated cyber-crime [5].



Figure 2.15 : Microsoft Security Development Life cycle

Model includes seven phases which added two new phases for the classic software development life cycle. It is interesting to see that the model recognizes the need of core security training to the people who ever involved with the development life cycle. Also, the model state that during the implementation phase, where the developers do coding, it is necessary to perform static analysis. This is to make sure that the developed source code does not contain potential security vulnerabilities and followed the required security best practices.

2.3.2 Seven Security Touch Points Proposed by Gary McGraw

Most organizations have a well-oiled machine with the sole purpose to create, release, and maintain functional software. However, the increasing concerns and business risks associated with insecure software have brought increased attention to the need to integrate security into the development process. Implementing a proper Secure Software Development Life Cycle (SSDLC) is important now more than ever [6].

Software Security Touchpoints

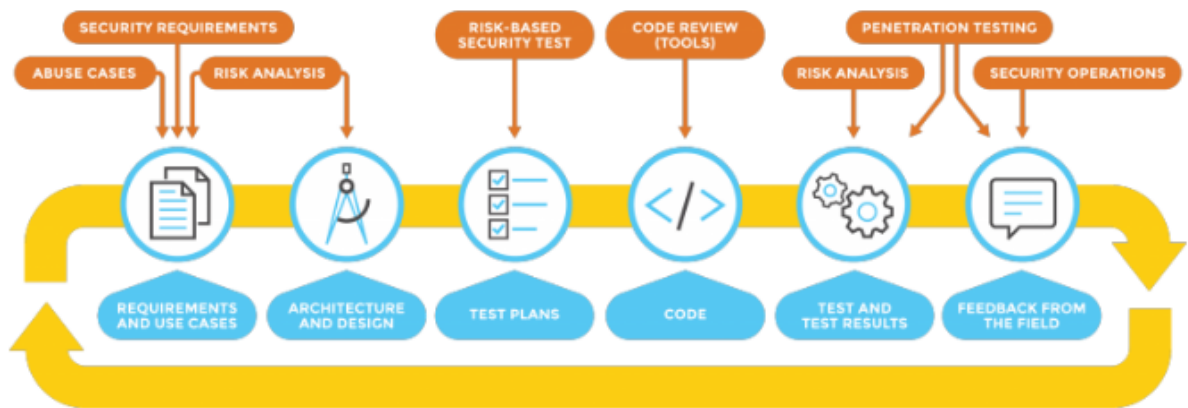


Figure 2.16 : Software Security Touch points

Model proposed seven touch points, which are necessary actions should be taken at each and every phase of the software development life cycle. This model mentioned about the abuse cases, which helps to understand the mindset of an attacker, which will be a great help to secure the application. Also in the coding phase, the model has mentioned the needs of code reviews using tools. This is where it looks for the source code to see whether there is any weakness in the code which can be leveraged by an attacker. Both these Secure Software Development Life Cycle models clearly mentioned that the need of reviewing or performing static analysis to the source code, during the development phase itself. This will greatly help on addressing security weakness in the code if exists, during the development. For the organization it is great benefit, because of the final product will be rugged with less issues and the cost of fixing a critical vulnerability which the application is in production can be eliminated.

2.4 Static Analysis

To implement built in security for the software development life cycle, it is necessary to have static code analysis performed during the coding phase of the development life cycle. Microsoft and Cigital models for Secure Software Development Life Cycle are clearly mentioned static analysis requirement. Static analysis is the process of scanning the source code and identifying the intended functionality of the source code to predict the potential security vulnerabilities. This is very useful because the quality-oriented approach to security leaves many opportunities for attackers [11], especially because only the functional use case is considered.

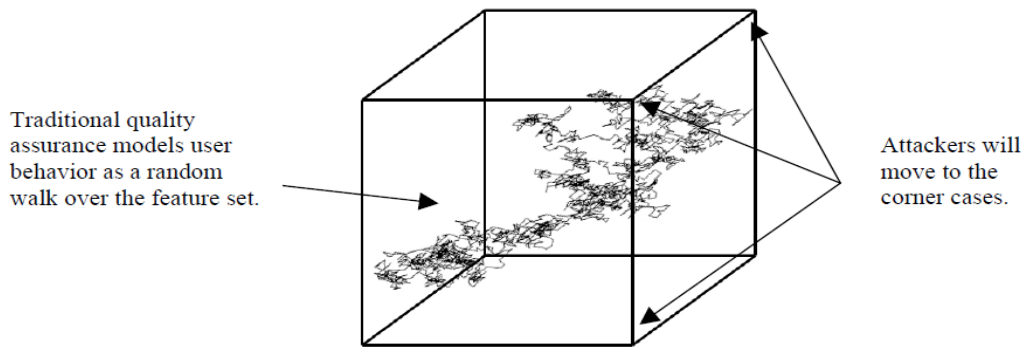


Figure 2.17 : Traditional quality assurance model vs Attacker

It is impossible to perform static analysis manually. Simple because the code base is huge and possible to have human errors. Modern software products typically contain millions of lines of code. Precisely locating the source of bugs in that code can be very resource consuming [9]. Because of that, Static Code Analysis usually means running a particular Static Code Analysis tools which will attempt to discover and highlight possible vulnerabilities within the (non-running) source code by using techniques such as Taint Analysis and Data Flow Analysis [10]. Development teams commonly turn to third-party software to incorporate particular functionality, such as communications or graphics, into their applications [12]. This is another area where the static analysis can help to discover the potentials security vulnerabilities. Also, because many software security weaknesses are introduced at the implementation phase, using a source code security analyzer should help reduce the number of security vulnerabilities in software [8]. There are numerous techniques to perform static analysis to discover potential vulnerabilities. Most of the time combination of couple of techniques are used. Couple of popular techniques are as follows [10].

Data Flow Analysis

With this technique, it tries to understand the run-time behavior of the data using the static source code.

```

1. $a = 0;
2. $b = 1;
3.
4. if ($a == $b)
5. { # start of block
6.   echo "a and b are the same";
7. } # end of block
8. else
9. { # start of block
10. echo "a and b are different";
11.} # end of block

```

Figure 2.18 : Data flow Analysis

Control Flow Graph

This technique tries to represent the software by nodes which represent the blocks of the software. Normally it has entry block and exist block and arrows are used to represent flows from one node to another.

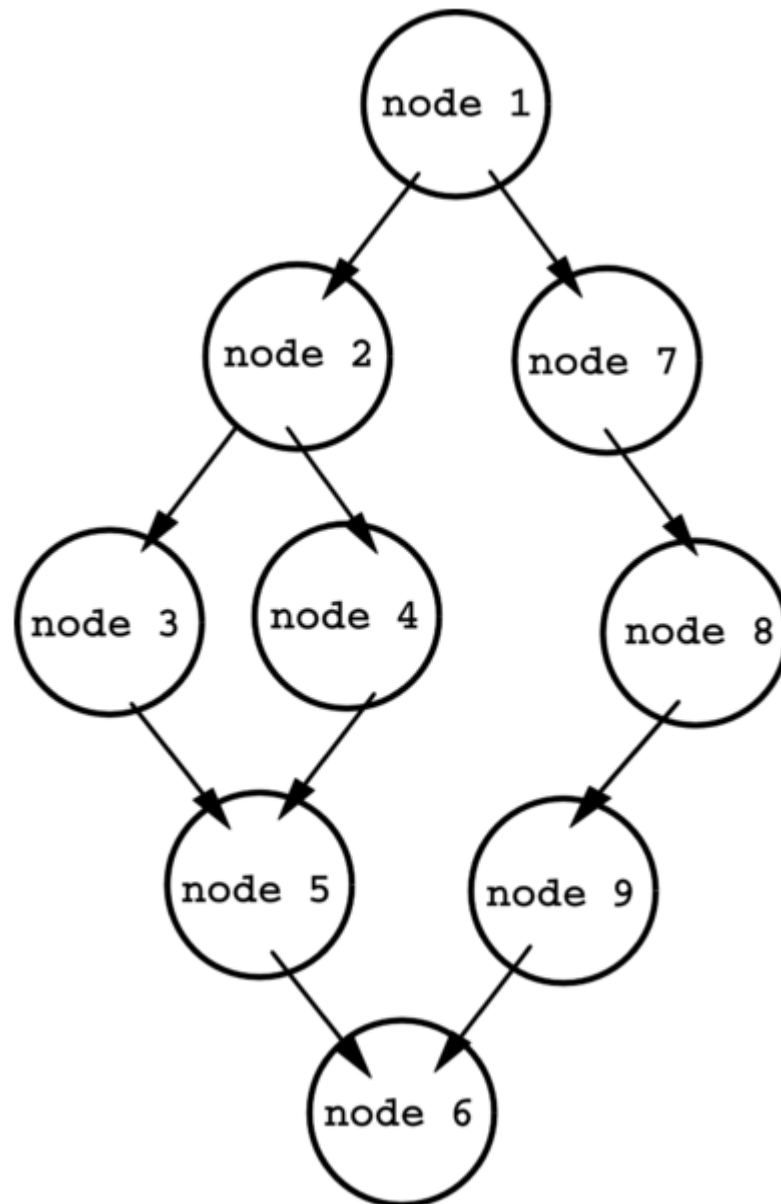


Figure 2.19 : Control flow Graph

Taint Analysis

Taint analysis is about analyzing the variable used in the source code. It basically attempts to identify the used variable in the source code that can be changed by the user input and then analyze to see whether those variables are used for some purpose without proper validation or sanitization. If the user controllable variables are passed in to some other functionalities directly without proper neutralization, then it will mark it as a vulnerability.

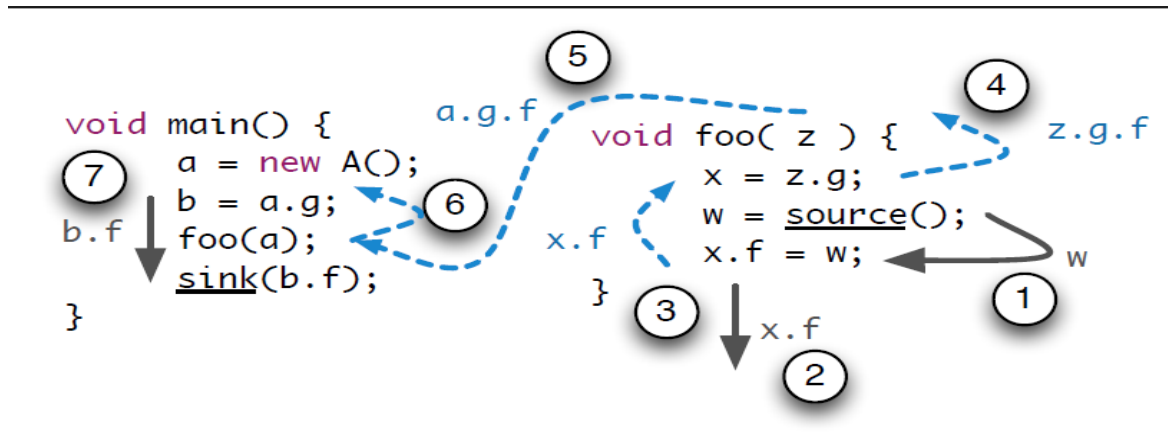


Figure 2.20 : Taint Analysis

Lexical Analysis

During the lexical analysis, it will convert syntax of the source code into token and after that it will be easy to understand and identify the source to manipulate and see for the vulnerabilities.

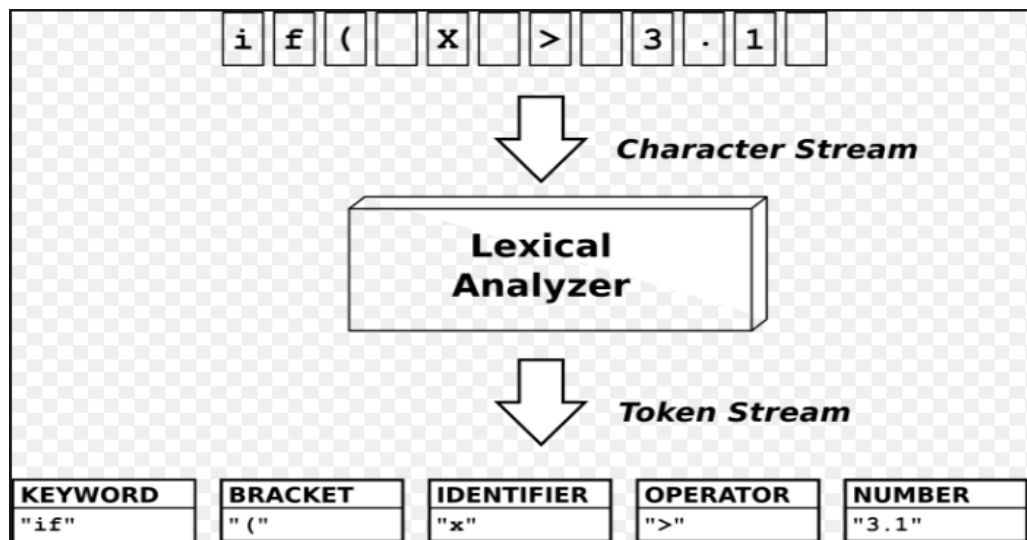


Figure 2.21 : Lexical Analysis

The expected outcome of a static analysis tool is to run through the source code and predict the potential security vulnerabilities. The tool should be fast, accurate, user friendly and should provide easy to understand meaningful reports. Since most of the top security vulnerabilities are introduced during the development phase, it is a great advantage to find these vulnerabilities during the development phase itself and address them immediately, rather than waiting till the last moment. To fulfill this requirement some tools support integration with Integrated Development Environment, and these tools are capable of finding the potential vulnerabilities during the development of the application and highlight them to the developer with suggestions to fix the issues [10].

2.4.1 Static Analysis Tools

The true power of static analysis tool is, it can analyze the entire source code, without executing it. Which means the static analysis tool can cover complete application, without missing anything. This is something difficult to achieve with dynamic testing as humans may miss some scenarios or because of the way the source code is written, the application may have unknown, unexpected scenarios that no knows they are exists. It is mandatory to select a dependable static analysis tool to achieve better or correct results. One common issue is most of the commercial static analysis tools are highly expensive which is something a small or medium scale organization may not be able to afford. Because of this reason most of the developers as well as organization tend to use open source static analysis tools. Obviously, there are limitations with open source static analysis tools and also can the organizations depend on the results provided by these tools, is also an important question, which everyone should consider.

2.4.1.1 Open Source Static Code Analysis Tools

There is huge list of open source static analysis tools available in the market, but only few are dependable and which are capable of performing an accurate static analysis results. Checkmarx, which is a well-known company who owned one of the most famous static analysis tool, has done an analysis on available open source static analysis tools and provide some interesting results and recommendations. Below are some of the points to consider, provided by Checkmarx, when the organization is selecting an open source tool to perform the static analysis.

Points to consider - Selecting an open-source static code analysis tool

Development language or languages supported by the tool
Types of vulnerabilities and code issues can be found by the tool
Type of IDE supported and time to get the feedback
Required learning curve for the tool
Customization and the support for automation of the tools
Any support provided by the tool for the organization
How much support provided for integration and automation
Other tools should use with the tool to get the maximum output

Table 2.3 : Selecting an open-source static analysis tool - Points to Consider

Also, Checkmarx came up with a list of open source static analysis tools, which are promising to provide required results and organizations can depend on. Below are the names with few details [20].

VisualCodeGrepper

Multiple languages, Java, C++, C#, VB and PHP supported by the tool and provide a detailed report and tool has easy to use, user friendly interface [21].

Advantages	Disadvantages
Tool makes it possible to customize the configurations as for the requirement.	Though the tool support for multiple programming languages, it cannot automatically detect the programming language and scanner has to select it for the tool to perform the scan.
Tool indicates the severity levels of the identified vulnerabilities.	
Focuses on OWASP top vulnerabilities and recommendations.	The vulnerability list that he tools support is fixed and cannot be modified.
Owner is updating the tool and it is active software.	Tool is not fully automated.

Table 2.4 : VisualCodeGrepper - Advantages and Disadvantages

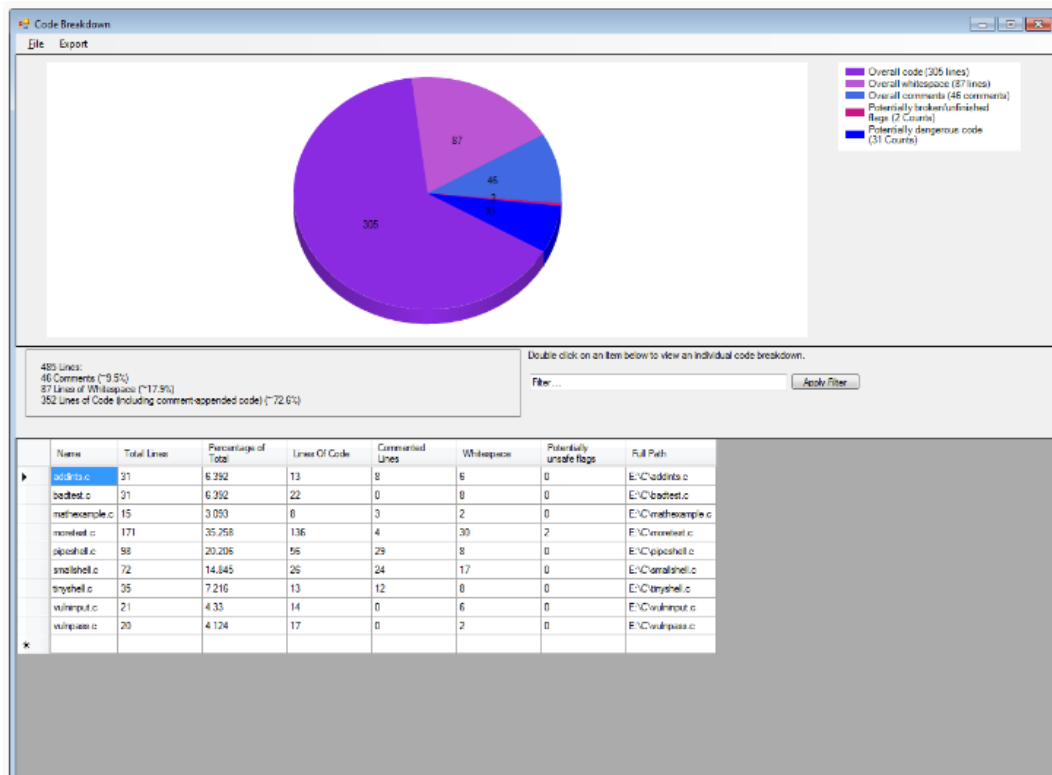


Figure 2.22 : VisualCodeGrepper V2.1.0

YASCA (Yet Another Source Code Analyzer)

YASCA is a static analysis tool, mainly target Java, C/C++, HTML, JavaScript, ASP, ColdFusion, PHP, COBOL, .NET and some other programming languages. Interesting feature it has is, tool made it possible to integrate with other related tools like FindBugs, PMD, JLint, JavaScript, Lint, PHPLint, CppCheck, ClamAV, RATS, Pixy. Also, the tool provides commercial support to the users, including custom development, integration and rules [22].

Advantages	Disadvantages
Possible to integrate with other powerful and related tools.	Capable only for finding straight forward, low-hanging fruits and Cross-Site scripting and SQL injections attacks
Possible to integrate with other powerful and related tools.	

Table 2.5 : YASCA - Advantages and Disadvantages

```

# .\yasca.exe
Yasca 1.3 - http://www.yasca.org/ - Michael U. Scovetta

Usage: yasca [options] directory
Perform analysis of program source code.

--debug          additional debugging
-h, --help       show this help
-d "QUERYSTRING" pass the expanded query string to Yasca's components
-i, --ignore-ext EXT,EXT ignore these file extensions
                  (default: exe,zip,jpg,gif,png,pdf,class)
--ignore-file FILE ignore findings from the specified xml file
--source-required only show findings that have source code available
-f, --fixes FILE include fixes, written to FILE (default: not included)
                  (EXPERIMENTAL)
-l, --level LEVEL show findings at least LEVEL (1-5) (default: 5)
-p, --plugin DIR|FILE load plugins from DIR or just load FILE (default: ./plugins)
-px PATTERN,PATTERN... exclude plugins matching PATTERN or any of "PATTERN,PATTERN"
                  (multiple patterns must be enclosed in quotes)
-o, --output FILE write output to FILE (default: unique file on
                  desktop in Yasca directory)
--log FILE        write log entries to FILE
-r, --report REPORT use REPORT template (default: HTMLGroupReport). Other options
                  include HTMLGroupReport, CSUReport, XMLReport, SQLReport,
                  and DetailedReport.
-s, --silent      do not show any output
-v, --version     show version information

Examples:
yasca c:\source_code
yasca /opt/dev/source_code
yasca -px FindBugs,PMD,Antic,JLint /opt/dev/source_code
yasca -o c:\output.csv --report CSUReport "c:\foo bar\quux"
yasca -d "SQLReport.database=./ny.db" -r SQLReport /opt/dev/source_code

```

Figure 2.23 : YASCA

Above two tools are supporting for multiple programming languages. Most of the open source tools support for only one programming language and below is the list mentioned by Checkmarx analysis report [20].

OWASP LAPSE+

The tool is developed by OWASP to detect security vulnerabilities of Java EE applications and it developed as an eclipse integrated development environment plugin, so it is easy for software developers to use the tool. LAPSE+ can detect Parameter Tampering, URL Tampering, Header Manipulation, Cookie Poisoning, SQL Injection, Cross-site Scripting (XSS), HTTP Response Splitting, Command Injection, Path Traversal, XPath Injection, XML Injection and LDAP Injection vulnerability categories [23].

Advantages	Disadvantages
Possible to integrate with integrated development environment and perform the source validation without compilation.	Only support for eclipse integrated development environment
Tool handles the testing with three steps, which are identifying the vulnerability source in the source code, identifying the vulnerability sink in the tool and examine to see whether we can use vulnerability sink to reach the vulnerability source.	No new versions after 2012

Table 2.6 : OWASP LAPSE+ - Advantages and Disadvantages

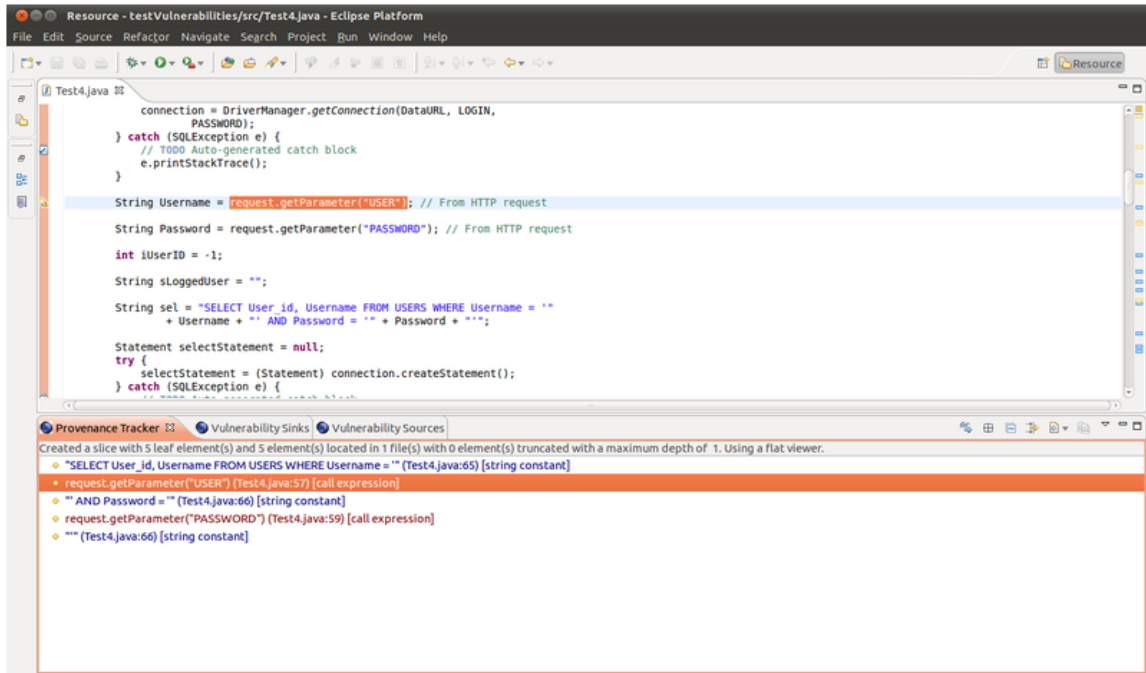


Figure 2.24 : OWASP LAPSE+

RIPS

Tool developed using PHP to discover security vulnerabilities of PHP applications. The tool can discover basic vulnerabilities including Cross-Site scripting, Remote code execution and SQL injection attacks. Tool also provides a framework for further manual analysis [25].

Advantages	Disadvantages
Fast processing and finding range of security vulnerabilities.	RIPS is abandoned the development and planning to come up with a re-write, but still not available.
Informative reports with visualization which is easy to understand for developers	

Table 2.7 : RIPS - Advantages and Disadvantages

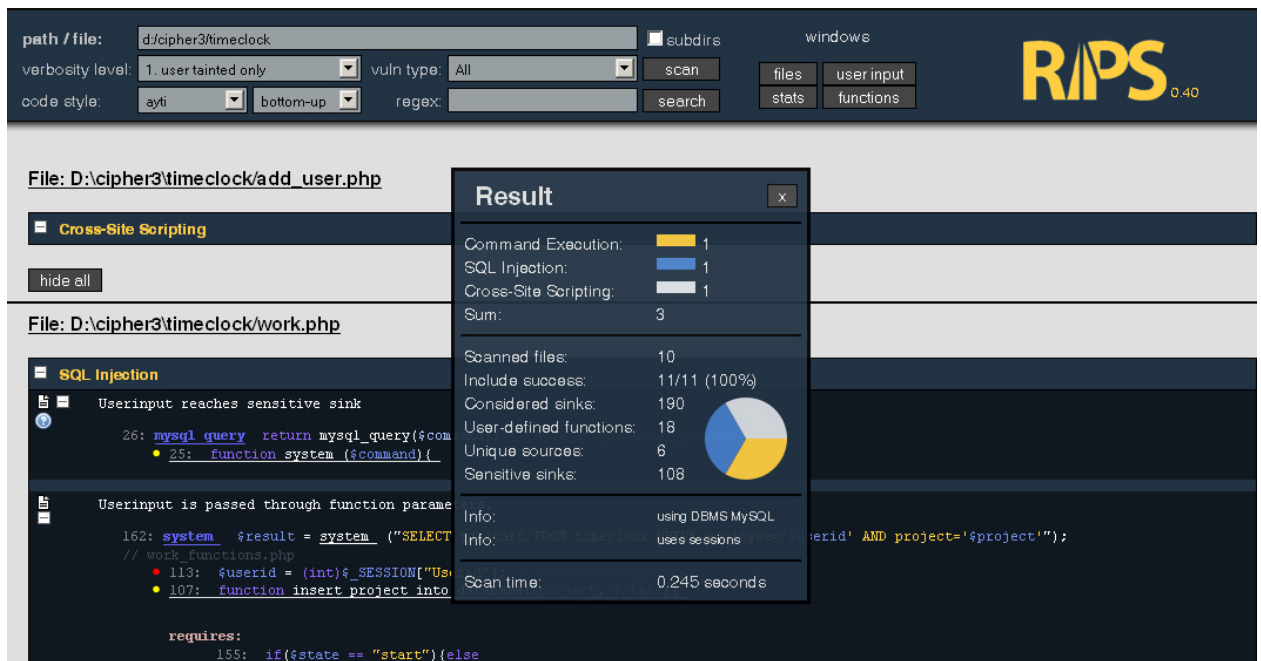


Figure 2.25 : RIPS

DevBug

DevBug is a free online tool to analyze security vulnerabilities of PHP code, mainly developed with JavaScript and the tool is getting support from RIPS and few other available tools [26].

Advantages	Disadvantages
Available online and very easy to use.	Very simple and very light analysis
Linked OWASP guideline for more information about the vulnerabilities.	

Table 2.8 : DevBug - Advantages and Disadvantages

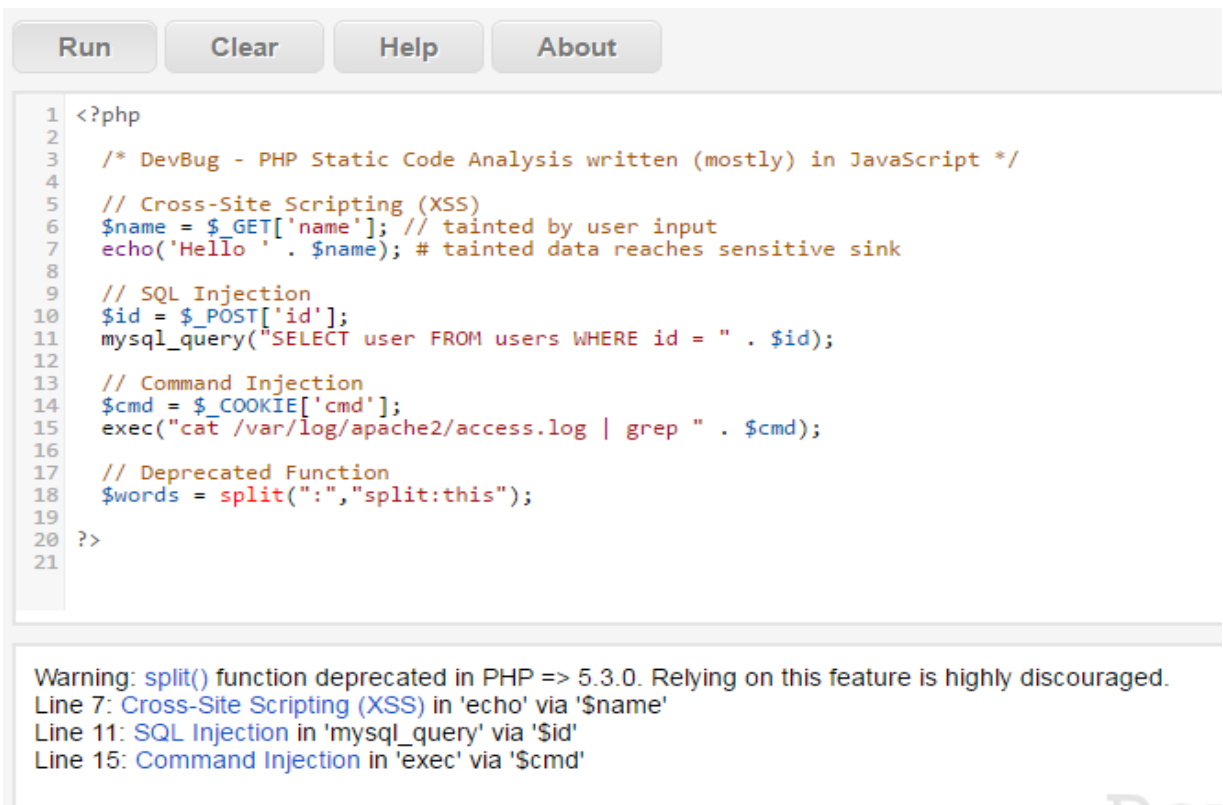


Figure 2.26 : DevBug

Flawfinder

Main purpose of the tool is to be simple and user friendly and it reports security vulnerabilities that are well known in applications which are written in C programming language. The tool is written a simple command line execution using a powerful language named python. Also, the tool is CWE compatible [27].

Advantages	Disadvantages
Tool can detect only the code changes and quickly verify only the changes to find security vulnerabilities.	High number of false positives.
Tool has a long history and well maintained with regular updates.	Require python 1.5 version to run the tool.

Table 2.9 : Flawfinder - Advantages and Disadvantages

```

Flawfinder version 1.29, (C) 2001-2014 David A. Wheeler.
Number of dangerous functions in C/C++ ruleset: 160
Examining test.c
Examining test2.c

```

FINAL RESULTS:

```

test.c:32: [5] (buffer) gets:
  Does not check for buffer overflows (CWE-120). Use fgets() instead.
test.c:56: [5] (buffer) strcat:
  Easily used incorrectly (e.g., incorrectly computing the correct maximum
  size to add) (CWE-120). Consider strcat_s, strlcat, or automatically
  resizing strings. Risk is high; the length parameter appears to be a
  constant, instead of computing the number of characters left.
test.c:57: [5] (buffer) _tcsncat:
  Easily used incorrectly (e.g., incorrectly computing the correct maximum
  size to add) (CWE-120). Consider strcat_s, strlcat, or automatically
  resizing strings. Risk is high; the length parameter appears to be a
  constant, instead of computing the number of characters left.
test.c:60: [5] (buffer) MultiByteToWideChar:
  Requires maximum length in CHARACTERS, not bytes (CWE-120). Risk is high,
  it appears that the size is given as bytes, but the function requires size
  as characters.
test.c:62: [5] (buffer) MultiByteToWideChar:
  Requires maximum length in CHARACTERS, not bytes (CWE-120). Risk is high,
  it appears that the size is given as bytes, but the function requires size
  as characters.
test.c:73: [5] (misc) SetSecurityDescriptorDacl:
  Never create NULL ACLs; an attacker can set it to Everyone (Deny All
  Access), which would even forbid administrator access (CWE-732).

```

Figure 2.27 : Flawfinder

CPPCheck

CPPCheck is a tool under GNU license, developed to detect issues in C/C++ applications, which are normally not detected by the compilers. The tool offers both command line and a GUI options and also support integration with number of popular integrated development environments [28].

Advantages	Disadvantages
Supported integration with Eclipse, Hudson, Jenkins and Visual Studio integrated development environments.	Tool can detect only very limited issues.
Frequent updates.	Difficult to customize and comparatively slower than other tools

Table 2.10 : CPPCheck - Advantages and Disadvantages

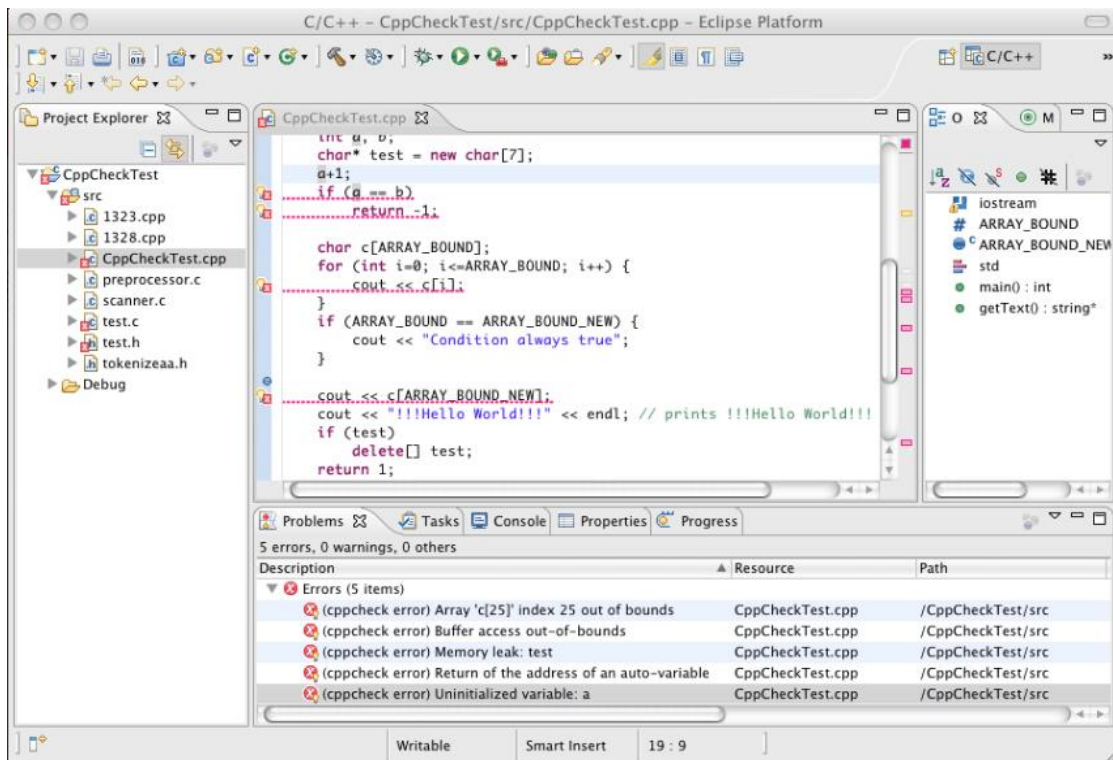


Figure 2.28 : CPPCheck

Brakeman

Purpose of this tool is to find potential security vulnerabilities of an application developed by ruby on rails, during the development life cycle. Also, the tool is used by some well-known commercial organization to do their static code assessments including twitter and GitHub [29].

Advantages	Disadvantages
Tool is faster and easy to setup and configure.	Only limited for ruby on rails.
Highly effective for ruby on rails.	High number of false positive and false negatives.
Well maintained and regular updates.	

Table 2.11 : Brakeman - Advantages and Disadvantages

```

== Brakeman Report ==
Application Path: /home/justin/work/brakeman/test/apps/rails5
Rails Version: 5.0.0
Brakeman Version: 3.3.5
Scan Date: 2016-09-05 19:01:09 -0700
Duration: 0.403342327 seconds
Checks Run: ContentTag, FileAccess, UnsafeReflection

== Overview ==
Controllers: 3
Models: 3
Templates: 12
Errors: 0
Security Warnings: 4

== Warning Types ==
Cross Site Scripting: 3
Remote Code Execution: 1

== Warnings ==
Confidence: High
Category: Cross Site Scripting
Message: Rails 5.0.0 content_tag does not escape double quotes in attribute values (CVE-2016-6316). Upgrade to 5.0.0
File: Gemfile.lock
Line: 115

Confidence: High
Category: Cross Site Scripting
Message: Unescaped parameter value in content_tag
Code: content_tag(:div, "hi", :title => params[:stuff].html_safe)
File: app/views/widget/content_tag.html.erb
Line: 1

```

Figure 2.29 : Brakeman

It is very clear that there are many static analysis tools available and some tools are performing really well, so commercial organization kept faith on those. Brakeman a good example and according to the Brakeman team [29], many commercial organization are using the tool and which means the tool must be providing the expected outcome, because after all, the companies are commercial and they should take everything very seriously.

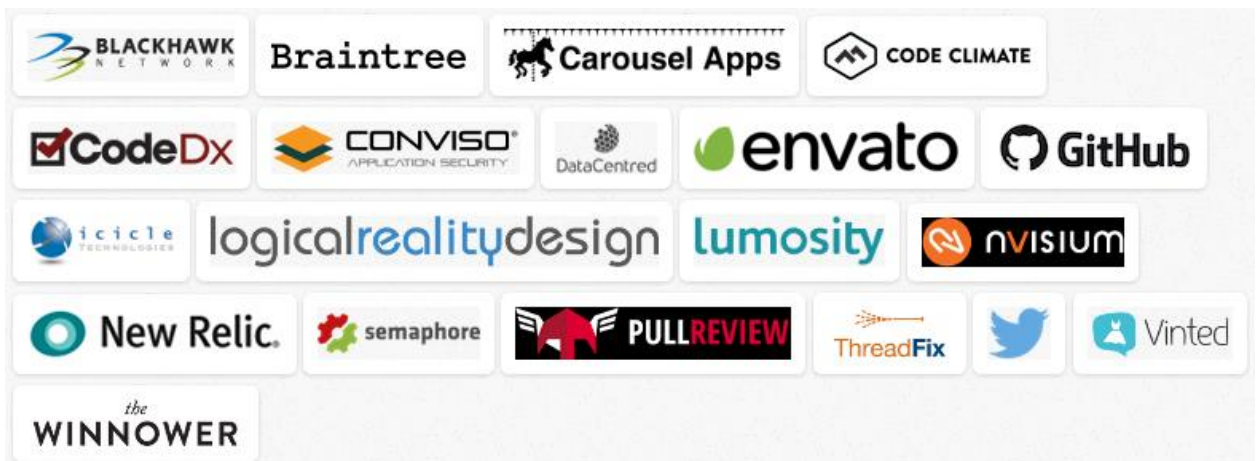


Figure 2.30 : Brakeman users

By considering the analyzing done by Checkmarx on open source static analysis tools, below are some point that can be extracted.

Tools can support only limited programming languages.
Can discover limited number of security vulnerabilities, mostly the very common ones.
No frequent updates for most of the tools.
Most of the tools are isolated and limited support for integration with other tools.
Limited or no customization and user support.

Table 2.12 : Points to consider with open source static analysis tools - Checkmarx

Commercial organization are highly sensitive environments and accuracy, dependability, automatability, maintainability, user friendliness and the support provided are the main factors they consider when they select a tool or software solution for the organization. This is the main reason why commercial organization would like to consider commercial tools, because they provide state of the art solutions which can handle all modern technologies, with customizability and interpretability, most importantly with great support service. Also, some tools provide the necessary features and can be integrated to with the integrated development environment, so the developer can get the instant feedback from the tool. So, the organizations can depend on these tools and focus on whatever the business goals and this is the ultimate goal of the organization. Like other commercial products, static analysis tools are also a very competitive product and some of the information technology and software development giants are building static analysis tools to the market with great features and superb after sales support for end users and they managed to make huge revenue of out these tools. Below are some famous and widely used commercial static analysis tools.

IBM Security AppScan Source

AppScan source is the static code analysis solution provided by well-known IT company called IBM. The plan is to help software development organizations to identify potential security vulnerabilities of the web and mobile applications, by analyzing the source code of those applications for lower cost. The tools can be integrated to the software development life cycle and include support for Java, Objective C, JavaScript, Cordovo and HTML5 [30].

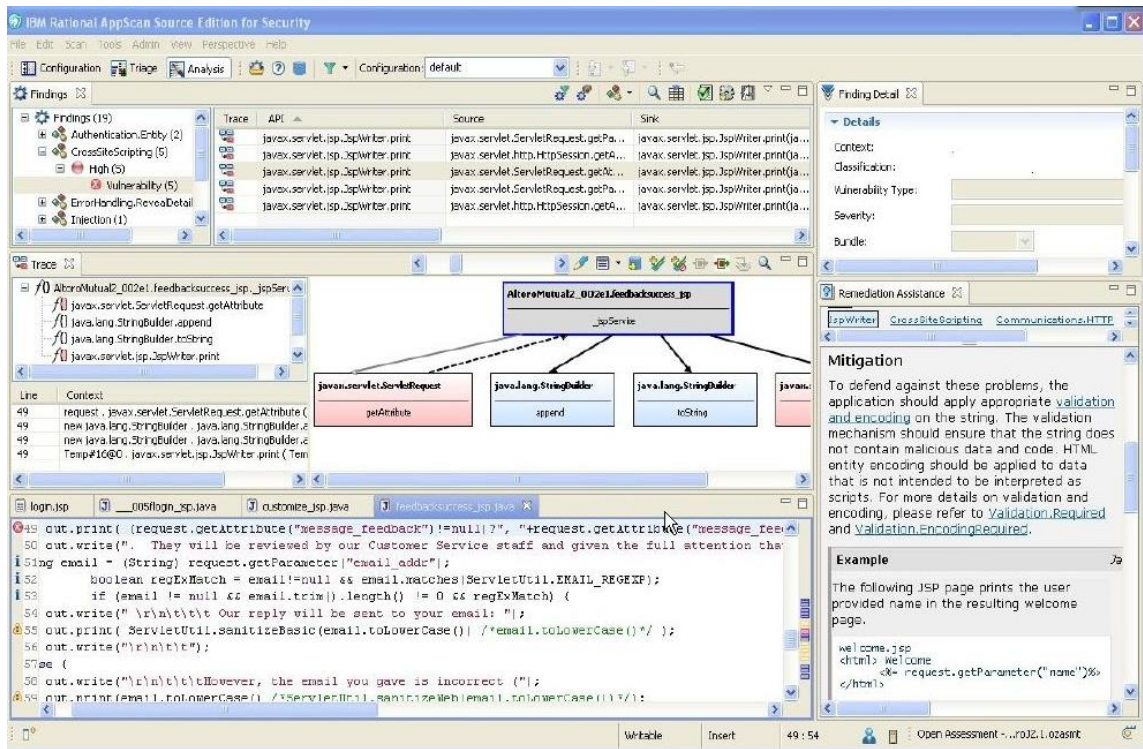


Figure 2.31 : IBM AppScan source

IBM AppScan Source also has below capabilities

Cost-effective source code analysis tool.
Great support for integration with other existing tools such as development related tools, build tools and monitoring tools.
Management of best practices and policies of security.
Support for governance and compliance.

Table 2.13 : IBM AppScan Source capabilities

Fortify Static Code Analyzer

The tool is developed by well know IT company named Hewlett Packard Enterprise and it is developed by highly skilled groups of security professionals the tool can identify the security vulnerabilities in the source code with appropriate risks and guidelines to address the vulnerability. One of the great feature that the tool has is, incremental scans which allows to perform a scan faster and which directly helps to improve the productivity of the organization [31].

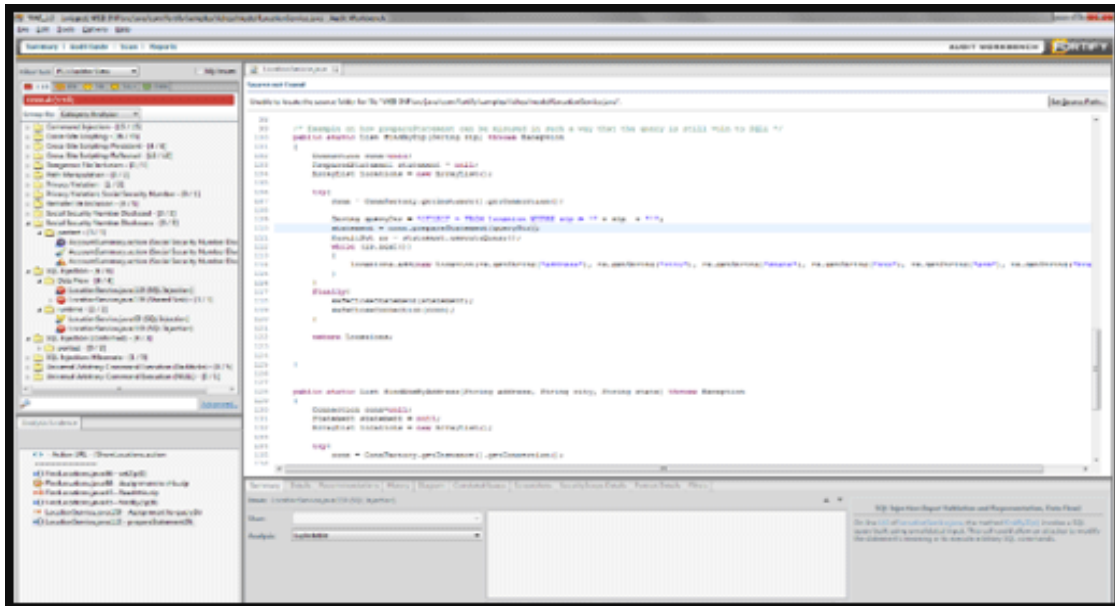


Figure 2.32 : Fortify Static Code Analyzer

Below are some of the major benefits provided by the Fortify Static Code Analyzer

Increase efficient with incremental scanning and providing results faster and reduce the time to wait for the security review.
Wide variety of programming languages, frameworks and development environments are supported and also mixed development environment are supported.
Provide accurate results with sophisticated rules engine which is frequently updated by the research team.
User friendly and easy to use and it is easy to integrate with other tools.
Most of the programming languages are supported and capable of supplying to the growing demand.

Table 2.14 : Fortify Static Code Analyzer benefits

Veracode

Veracode is also a highly respected static analysis tool among the software development organizations. This tool is heavily used and one of the best commercial static code analysis tool in the industry. One of the unique feature with Veracode is, the tool does not need the source code to analyze the vulnerabilities, instead the debug enabled compiled version of the source is enough using their own analyzer framework.

Since the Veracode proprietary analyzing framework can assess binaries for security vulnerabilities, customers can analyze third party components also to determine the security risks and which is a huge advantage. Also, it supports all kind of applications, including Web,

Mobile, Desktop and back-end applications. According to Veracode statistics, the tool has scanned 1.8 trillion lines of source code of 15 different programming languages belongs to 50 different frameworks. Which means the tool is really mature enough to do a static analysis for an organization to discover the security vulnerabilities successfully. Since Veracode provide a SaaS based security platform, organization can reduce the operational overhead. Organizations does not keep or spend money on in-house hardware or any other additional resources for the tool [32].

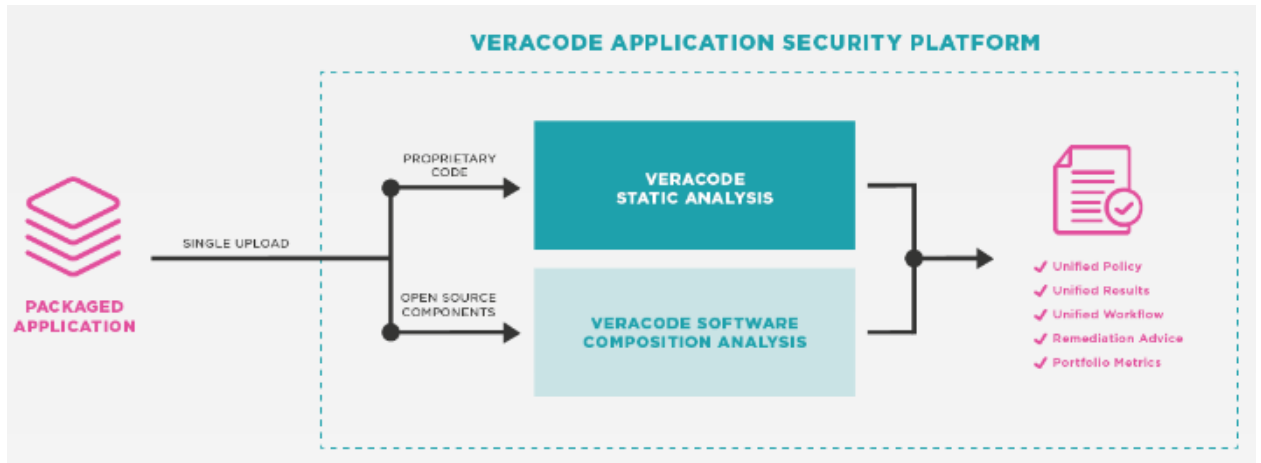


Figure 2.33 : Veracode application security platform

Veracode Static Analysis supports all widely-used languages for desktop, web and mobile applications including:

- Java (Java SE, Java EE, JSP)
- .NET (C#, ASP.NET, VB.NET)
- Web Platforms: JavaScript (including AngularJS, Node.js, and jQuery), TypeScript, Python, PHP, Ruby on Rails, ColdFusion, and Classic ASP
- Mobile Platforms: iOS (Objective-C and Swift), Android (Java), PhoneGap, Cordova, Titanium, Xamarin
- C/C++ (Windows, RedHat Linux, OpenSUSE, Solaris)
- Legacy Business Applications (COBOL, Visual Basic 6, RPG)

Figure 2.34 : Languages supported by Veracode

Veracode support various type integration. It allows the developer to integrate the tool with the integrated development environment, so the developer can get the results immediately. Other than that, tool allows to integrate it with build environments like Jenkins, so the build environment can initiate scans periodically and get the potential security vulnerabilities. With this it is possible to fully automate the scanning process where developers submit their changes and build environment make the build and if the build is successful then, upload the binaries to Veracode for the security analysis automatically. This will reduce the effort tremendously and

improve the productivity of the company. Also, Veracode provide detailed user-friendly reports, so developers can easily understand the issues and apply the recommended fixes immediately.

WhiteHat Sentinel Source

WhiteHat is a well-recognized and trusted security organization providing vast variety of security products and solutions and Sentinel Source is the static analysis tool provided by WhiteHat. Tool can scan the source code written in commonly used programming languages and can discover common vulnerabilities and provide a vulnerability report. Also, capable of providing recommended fixes for certain vulnerabilities. Analyzing binary files also possible for software written in certain programming languages. Possible to integrate with continuous integration tools and also with integrated development environments, so it is possible to identify vulnerabilities during the early stage of the development life cycle. Cloud option is available and also local installation is also possible in-case company have any issues with the intellectual property rights. Also, it is possible to get a help from WhiteHat technical team [33].

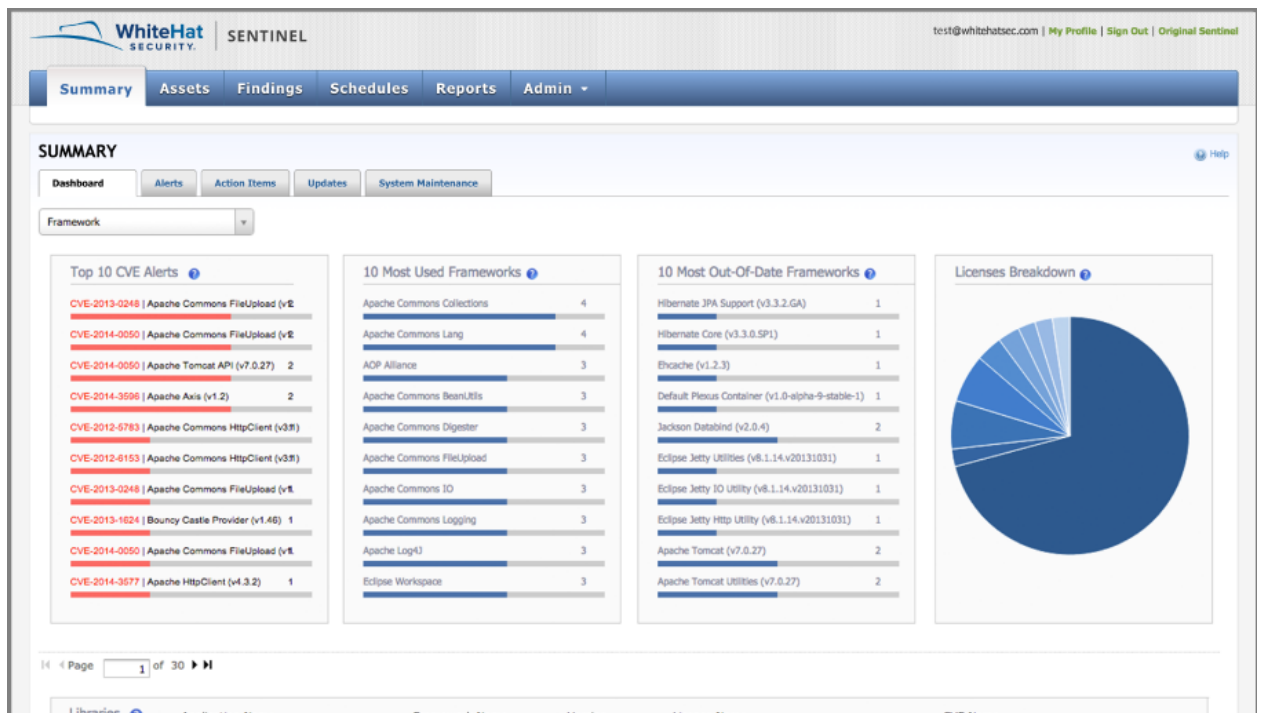


Figure 2.35 : WhiteHat Sentinel Source

Many organizations from almost all the business sectors are getting service from WhiteHat to make sure that what they developed are secured. The business sectors include financial, education, healthcare, government, software development, retail and many more. Below list shows some of the happy customers of WhiteHat.



Figure 2.36 : WhiteHat Security Customers

Checkmarx

Checkmarx is one of the best and most reliable tool in the world. The tool is very user friendly and easy to configure as well as integrate with continuous integration tools and build environments. They way that the tool shows the vulnerabilities to the users is very interesting and it is super easy to navigate step by step to the vulnerable point of the source code. Tool show what are the reasons for the vulnerability and sophisticated guide line to fix the vulnerability.

Tool can be locally installed and easy to maintain. Admin user of the tool can provide login accounts to the development and quality engineering teams so that the teams can perform source scanning and identify the security vulnerabilities. This is one of the major advantage where the development teams do not need to request reviews from the security team or wait for the security team to perform the scan and provide the results. Since the tool can remove the dependency between development teams and the security team, it can increase the productivity significantly.

Checkmarx can be configured easy to grab source code from almost all the well-known source code repositories including perforce, git and SVN. Tool provide regular updates to make sure it handles the latest security vulnerabilities and technologies [34].

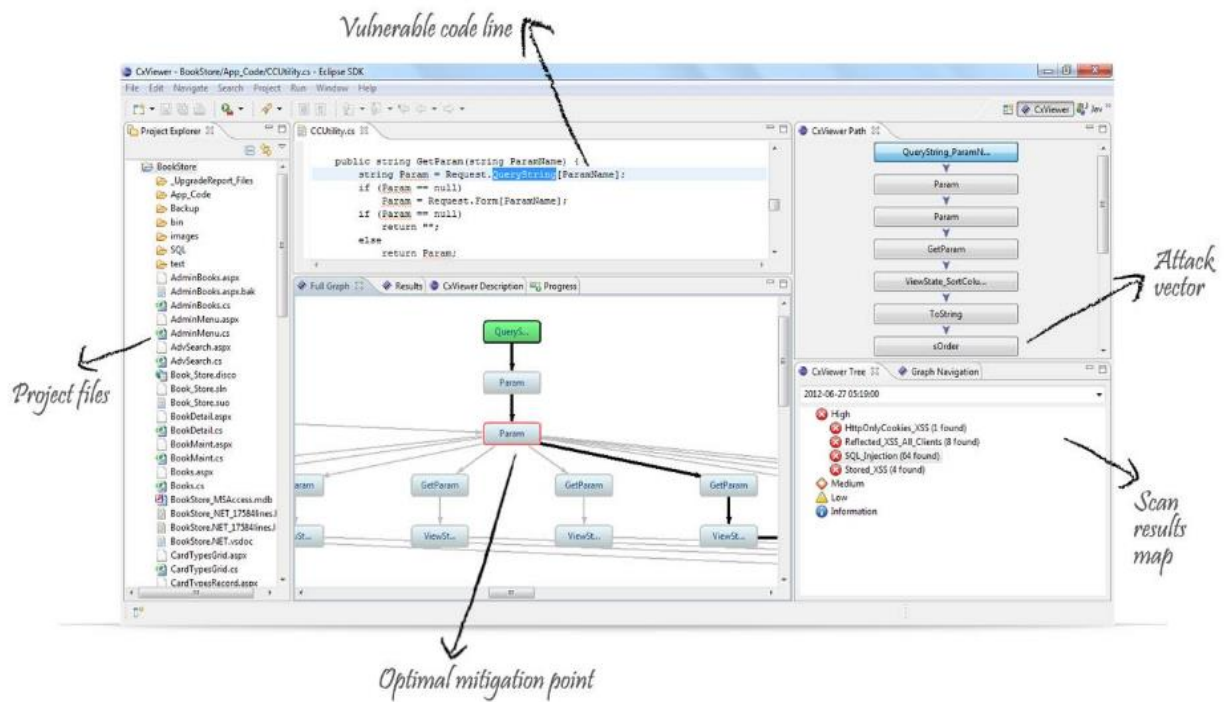


Figure 2.37 : Checkmarx Source Code Analysis Tool

Below are some of the feature and capabilities of the Checkmarx static analysis tool.

Tool supports 20 programming and scripting languages and the frameworks which covers the latest technologies.
No configurations needed from language to language.
Covers OWASP Top10, OWASP Mobile Top10, SANS Top 25, PCI DSS, HIPAA and other security standards.
Incremental code scanning capability which scan only modified or newly added source codes.
Strong integration capability with build environments and integrated development environments.
Supports hundreds of vulnerabilities including all the common vulnerabilities like SQL injections, Cross-Site scripting, Session issues and all.
Engineers can mark a particular vulnerability as false positive and tool has the capability to remember it between scans.
Tool can point out the best fix location which can save lots of remediation time.
User friendly vulnerability dashboard which shows the path and the exact location of the vulnerability.
Tool allows for custom rule creation.

Table 2.15 : Checkmarx static analysis tool benefits

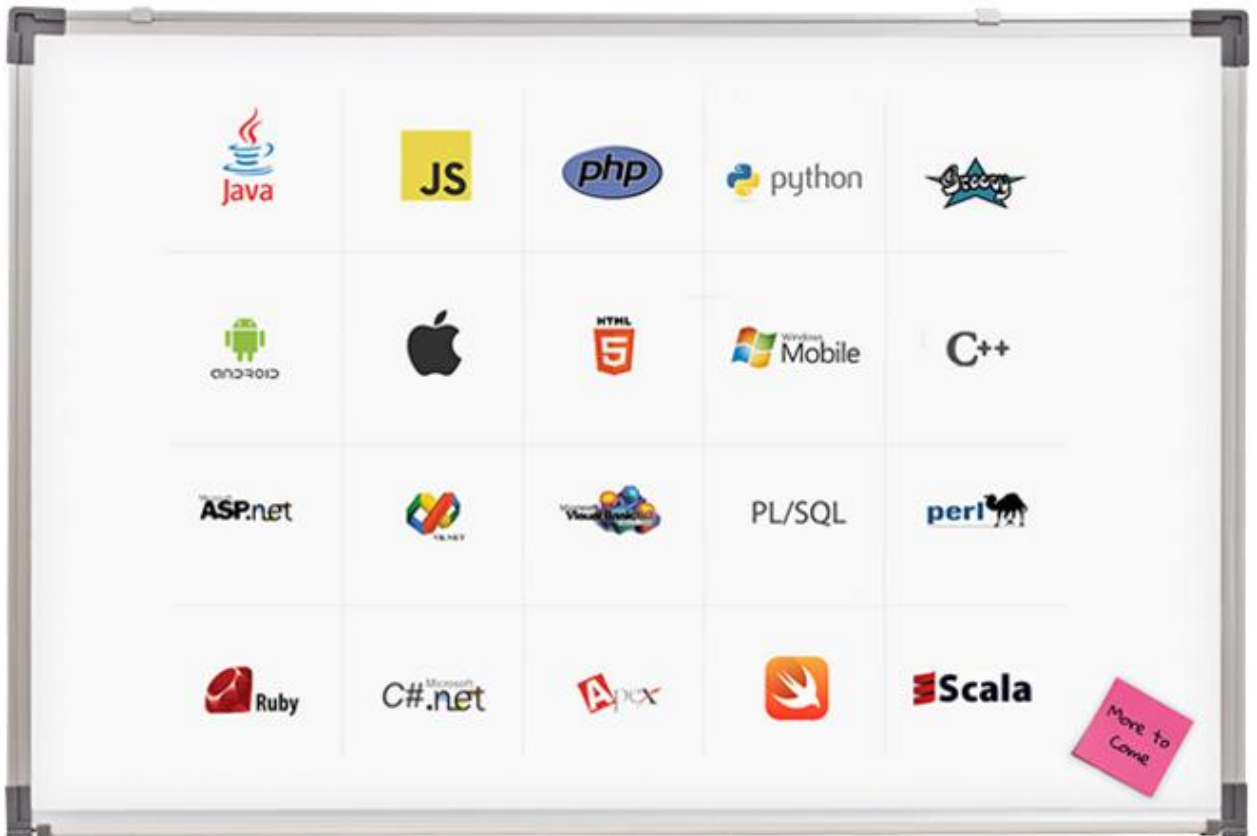


Figure 2.38 : Checkmarx Supported Programming Languages

There is another area of source code analysis that most of the organizations does not even aware of, which is the open source libraries. Most of the software tend to use open source, readymade software libraries due to many reasons, like cost saving, time saving and all. Open-source libraries are the foundation of most of the modern software these days. It is necessary to make sure that these open source components are also secured or follow security best practice and also well maintained with regular updates. To make sure these things, there are tools available and below are some of them. Because open source software is freely available and used everywhere, it can enter into any product, knowingly or without knowingly. Which will create an additional risk to the product and most probably no one is aware about, especially because no one is considering these open source software libraries.

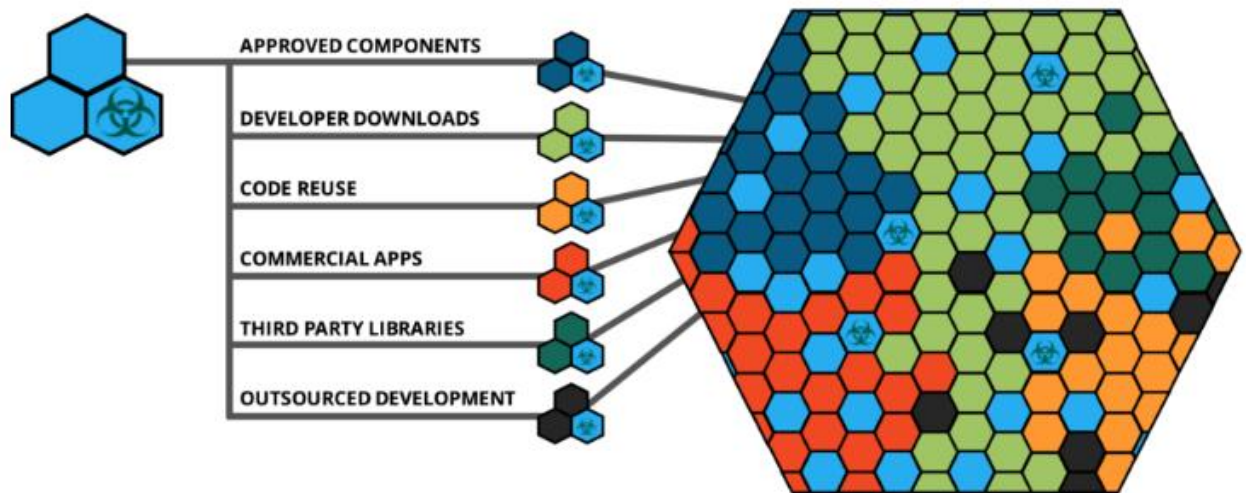


Figure 2.39 : Open Source Components getting in every Angle

As an organization, it is a must to do a comprehensive analysis and figure out about this open source software which are used within the product. Below are some of major points that the organization should consider [36].

- What are the open source components used within the product.
- Does the product use the latest version of those components
- Are those components being vulnerable.
- Are those components are well maintained.
- Are those components adhering to the required security policies and best practices.

Black Duck Hub

Black Duck Hub is providing a solution for the open source components issue state above. The tool can identify the used open source components with the application and asses the risk of it. Tool is a lightweight scanner with tracking and monitoring solution, which is also user friendly and support for integrating with other tools like build and continuous integration. Below are some of the main capabilities of the tool.

Scan the code base and identifies open source components used or referred in the source code.
Automatically maps the discovered open source components in use to known open source security vulnerabilities.
Flags policy violations, triage and tracks remediation progress.
Continuously monitors for newly identified open source vulnerabilities.

Table 2.16 : Black Duck Hub capabilities

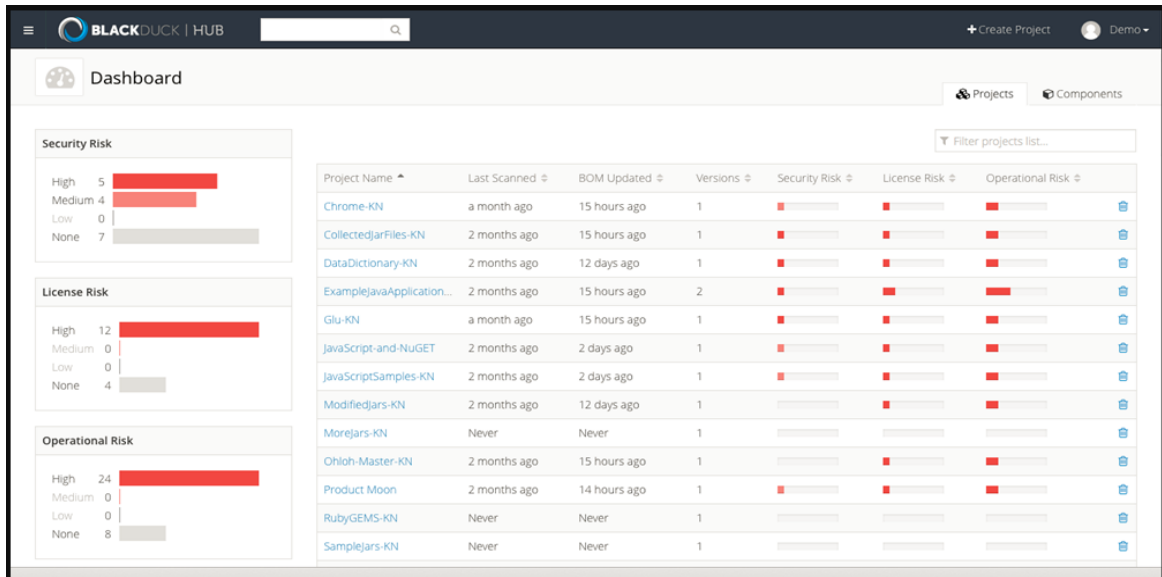


Figure 2.40 : Black Duck Hub

WhiteSource

WhiteSource is also a tool where it can scan the product source code and identify the used or referred open source components and capable of mapping relevant vulnerabilities and security risks. Tool is capable of integrating with build tools and continuous integration tools, also providing real time alerts on detected vulnerabilities. WhiteSource also has a browser plugin which can help developers, when they want browse and select some components, by suggesting better recommendations. Also, the tool provides a comprehensive report on open source inventory, so that the organization is aware of the current risks of the product [37].

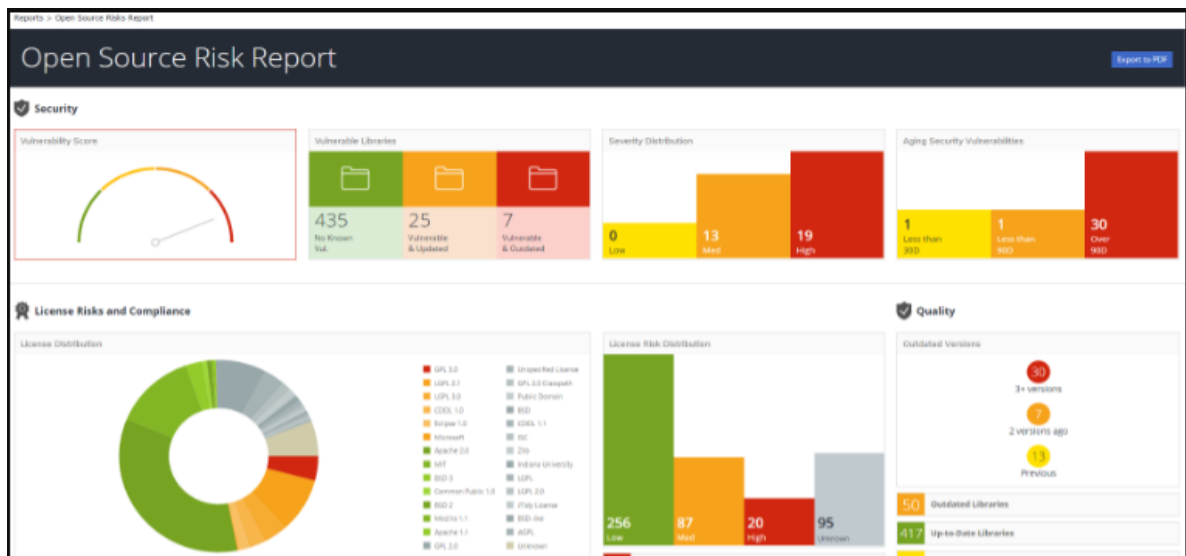


Figure 2.41 : WhiteSource

Tool helps to the organization defining a process of approving or rejecting open source components which are request by engineering teams, on the basis of each component’s license, vulnerabilities, newer version updates, how well the component is maintained and more.

Organizational Policies Help

Policies					
Policy name	Match	Action	Creator	Creation Date	
<input type="checkbox"/> Reject Eclipse 1.0	By License Group	Reject	Rami Sass	Sep 05, 13	details
<input type="checkbox"/> Catalog approval	If Exists in Product's Inventory	Approve	Rami Sass	Jun 11, 14	details
<input type="checkbox"/> Re-assign BSD-like	By License Group	Reassign	Rami Sass	Mar 24, 16	details
<input type="checkbox"/> Approve Apache	By License Group	Approve	Rami Sass	Sep 05, 13	details
<input type="checkbox"/> Reject GPLs	By License Group	Reject	Rami Sass	Sep 19, 13	details
<input type="checkbox"/> Reject high severity security issues	By Security Vulnerability Severity	Reject	Rami Sass	Nov 04, 15	details
<input type="checkbox"/> Workflow	By Regex on Resource Name	Conditions	Idan	May 24, 16	details
<input type="checkbox"/> Assign Medium Vulnerability to Security Officer	By Security Vulnerability Severity	Reassign	Gal Yaffe	Jun 27, 16	details

Figure 2.42 : WhiteSource, Managing Policies

In summary below points can be extracted by analyzing and considering all the facts about commercial source code analysis tools.

- Commercial tools are highly created and well maintained with frequent updates.
- Capable of providing accurate results.
- Support for almost all the modern programming languages and frameworks.
- User friendly, customizable and easy to integrate with all the other tools.
- End user support and organization can depend on the tool.
- Tool development companies do research and development to identify new vulnerabilities and root causes for those vulnerabilities.

2.4.2. Advantages and Disadvantages

Every tool irrespective of whether it is a commercial tool or an open-source tool, has issues, limitations and bottlenecks. Most importantly no tools can be used out of the box as it is, with default settings and configurations. There is customization, configuration changes, optimizations need to be done in order to get the better performance of the tool and to cater specific requirements of the organization. There are two kinds of issues that every tool has and organizations have to deal with those and put some level of manual effort to rectify those errors.

False Positive

This is a situation where the particular tool, detect and indicate a vulnerability, but in the vulnerability, is not exists in reality. Basically, it is a false alarming situation. Almost of the tools suffering from these issues and it is also acceptable. This is where the organization needs

to engage their engineers to go through the detected vulnerabilities and verify the issues and get rid of false positives.

False Negative

False negative is the opposite of false positive, where the tool is unable to detect a potential issue. Considering the definition, it self, false negative is more severe than the false positive, since the organization misses the vulnerabilities that are actually exists in the application. Most of the time the reason for this issue is that the organization is trying to tune the tool to reduce the false positives. Organization must configure and fine tune the tool before use it and it is always recommended to have a trial run period. The tool should be fine-tuned to make sure tool does not provide any false negative scenarios and also to make sure that the tool will generate lesser number of false positives. Static code analysis adds a great value for an organization, when it tries to implement secure software development life cycle. But as always, there are pros and cons with the static analysis. Some of the main important advantages are as follows

- Static analysis tools are faster, easy to use and can cover complete source code repository to find potential security vulnerabilities.
- Possible to integrated with development life cycle.
- Supported for almost all the development languages and frameworks.
- Engineering teams can perform the static analysis and no need to wait for security team to do it.
- Improve the productivity by saving lots of time and producing more secure software.

Some common disadvantages are as follows

- Static analysis is not instant, it will take some time. Analyzing every change then and there is practically impossible.
- Cannot find configuration issues since those are not in the source code.
- Unable to predict issues such as authentication and authorization issues.
- High number of false positives will take considerable effort to remove those.
- Technically, tool cannot detect all the vulnerabilities.
- There can be programming languages not supported by the tool.
- Commercial tools are highly expensive.

2.5 Vulnerable Programming Languages

There are many programming languages and frameworks are available in the present that are capable of building web and mobile applications. Most of the modern programming languages are very easy to learn and use, and because of that the beginners also can use these languages to develop complex state of the art software applications. Some frameworks provide all the components build in, like front end, middle components and the back-end database, like MEAN (Mongo, Express, AngularJS, NodeJS), Django (Python, MySQL), which makes the development of a web or mobile applications even more easier. This is however a good thing,

where the technology manages to reduce the learning curve for a particular programming language, so that the organizations can develop and deploy the applications to the market quickly.

However, there is a huge risk also attached to this. Because using and learning it very easy, even a beginner can develop a commercial application and also, since the framework is providing most of the features, developers are tending to totally rely on the framework. When considering about the application security, this is a major area that the organizations should focus. Typically, when selecting a programming language or a framework, organizations are focusing on, availability of developers or engineers, learning curve of the technology, performance and all. But the other most important factor is whether the language or the framework provides ways and means to develop a secured or rugged application. Organization should carefully look in to the matter and analyzed the matter, before they select a particular technology.

Veracode, the well-known application security firm, which also owns the very famous static code analysis tool, successfully conducted a research and publish a paper name “*State of Software Security: Focus on Application Development*”, by analyzing 200,000 different software applications from October 1, 2013, through March 31, 2015. Veracode engaged their best security professionals crawl well known and popular web development and scripting languages including PHP, Java, JavaScript, Ruby, .NET, C and C++, Microsoft Classic ASP, Android, iOS, and COBOL, by scanning hundreds of thousands of available applications during one and half years long. One highly important factor they discovered is that, non-popular languages like Classic ASP and ColdFusion and modern language, PHP are more vulnerable and the riskiest programming languages and also the .Net and Java are the safest programming languages. Flow density per MB is the metric used by Veracode in the report where it indicates the numbers of security vulnerabilities per one MB of source code [38].

2.5.1. Top 10 Vulnerable Programming Languages

Below is the top programming language list provided in the report by Veracode.

- Classic ASP – 1,686 flaws/MB (1,112 critical)
- ColdFusion – 262 flaws/MB (227 critical)
- PHP – 184 flaws/MB (47 critical)
- Java – 51 flaws/MB (5.2 critical)
- .NET - 32 flaws/MB (9.7 critical)
- C++ – 26 flaws/MB (8.8 critical)
- iOS – 23 flaws/MB (0.9 critical)
- Android – 11 flaws/MB (0.4 critical)
- JavaScript - 8 flaws/MB (0.09 critical)

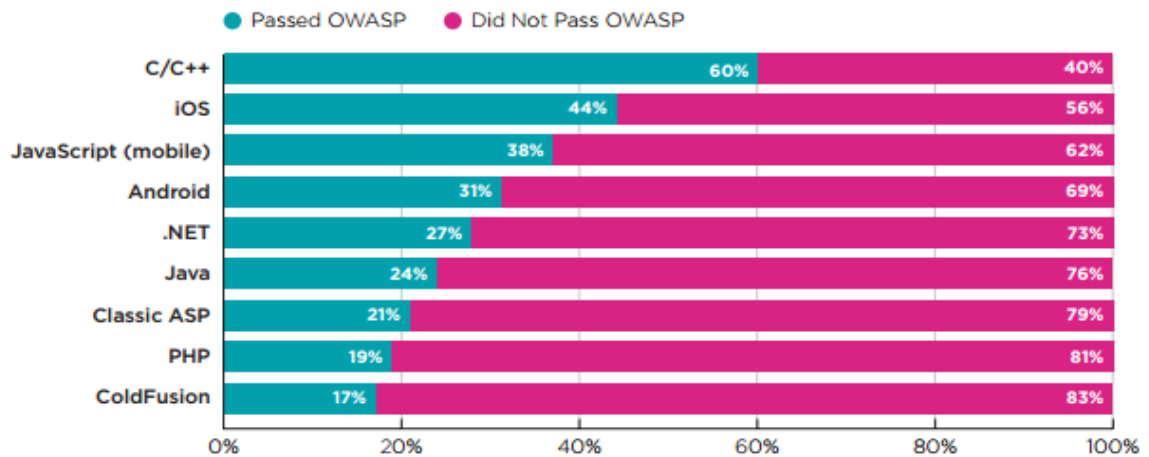


Figure 2.43 : Policy compliance by programming language

Surprisingly PHP language, even though it is a modern and heavily used programming language, becomes the third most vulnerable application development language. Almost all the very famous content management applications are written in PHP.

Veracode report provided justifiable, logical reasons for the PHP issues as below.

- 86% of applications written in PHP contained at least one cross-site scripting (XSS) vulnerability.
- 56% of apps included SQLi (SQL injection), which is one of the dangerous and easy-to-exploit web application vulnerabilities.
- 67% of apps allowed for directory traversal.
- 61% of apps allowed for code injection.
- 58% of apps had problems with credentials management
- 73% of apps contained cryptographic issues.
- 50% allowed for information leakage.

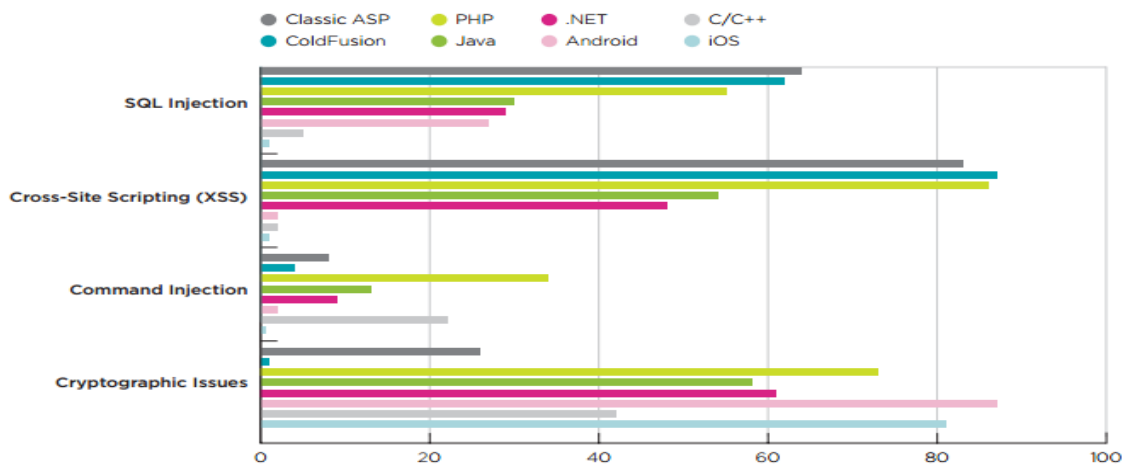


Figure 2.44 : Comparison of critical vulnerability types

Other than these above factors, there is one other important result included in the Veracode report, which is the vulnerability comparison which can be found with dynamic security testing and static security testing. This is also highly important for the organizations because they can understand the difference and what are the risks they have if they do not perform a one type of test. The difference between these two testing methodologies is that, dynamic application security testing also called DAST is using the running application and perform a black box test, whereas static application security testing also called SAST, focuses on the source code and perform a white box test. It is understandable that these two methodologies can detect different types of vulnerabilities and for an organization both these methodologies are important to make sure the product is vulnerability free and secured. One advantage of the static analysis is that, it can be leveraged during very early stages of software development life cycle.

Vulnerability Category	Dynamic	Static
Code Quality	n/a	63%
Cryptographic Issues	53%	58%
Information Leakage	80%	56%
CRLF Injection	n/a	49%
Deployment Configuration	55%	n/a
Server Configuration	16%	n/a
Cross-Site Scripting (XSS)	27%	47%
SQL Injection	6%	29%
Credentials Management	12%	25%
Code Injection	1%	2%
Time and State	n/a	23%
Directory Traversal	2%	47%
Insufficient Input Validation	4%	37%

Figure 2.45 : Dynamic vs. static application security testing

Every day there are new applications coming into market and organizations are start developing new applications and also new software developing companies also coming in. During the selection of programming languages, methodologies and frameworks, organizations should employ security professional to analyze the risks of these technologies and then selected a proper technology wisely to make sure the product, the organization deploy to the market is well secured.

Chapter 3 : Design

3.1 Design Overview

Ultimate goal of the project is to create a tool which is free and capable of analyzing security vulnerabilities of the source codes published in open forums. The project selected the StackOverflow as the open forum, since the stats shows that, StackOverflow is highly famous among the development community. Even though the project is aiming to create a static analysis tool, it is actually trying to address a different problem. When considering getting source code sample from the open source forums scenario, average develop will typically follow below steps.

- Developer search a solution for a particular problem.
- Refer couple of source code samples, that are available on the open forum like StackOverflow
- Directly copy the sample or part of it, or get influenced by the sample and follow the same to develop a solution.

What is missing here is, there is no way to make sure those source code samples are not having any security vulnerabilities or they followed the required security best practices. Even if the organization uses a commercial static analysis tool, it is practically not possible to analyze each and every sources samples to check the vulnerabilities before using them, mainly due to the time that the scan is taking and also the scans are very costly. If there is a pre-scanned knowledge base of these source samples and an easy tool to access it, then it will be a great help for the development community, because it is possible to analyze the security vulnerabilities quickly and easily, then and there, before they are using or implementing those source code samples. into their production source code.

This project is trying to address the above described issue by analyzing open forum published source code samples and create a vulnerability knowledge base. Also create an easy access, user friendly tool, where the developers will be able to use it to analyze the security vulnerabilities of a particular source code sample, by accessing the knowledge base created. The proposed system is to gather source code examples published in open forums and create a vulnerability knowledge base by analyzing the potential vulnerabilities of those source code examples, using a professional static analysis tool. Then implement a tool which can be used by the developer to analyze the security vulnerabilities of a particular source code example. Also, to create a dashboard to show case all the vulnerabilities exists in open forum source code examples. Proposed system has six main components state as below.

Web Crawler to read and grab source code blocks, published in the open forum.
Professional, commercial static analysis tool.
Vulnerability processor to read the vulnerabilities and store into a database.
Dashboard to showcase the findings and stats.
Tool which can be used by developers to identify the potential vulnerabilities of a selected source code block in the open forum.
Database management system to store the vulnerability data.

Table 3.1 : Components of proposed system

Other than the commercial static analysis tool and the database management system, all the other components are planned to develop using an appropriate programming languages. Primary target is to provide a simple, user friendly solution that can process the user request and return the results faster.

3.2 System Overview

As mentioned above the system is divided into independent components, so that the implementation can be done parallelly. More focus and the weight given for designing Dashboard components and the Developer tool, since those two components are providing a high value for the end users. Building the vulnerability knowledge base of the analyzed source codes, is the major part of the proposed system and that is not completely automated process. Some manual work also exists to continue the workflow of the building vulnerability knowledge base, such as, after web crawler crawled the source code samples, those need to be uploaded to the static analysis tool to perform the analysis. Also, when the tool completed the analysis, need to perform a false positive removal to make the result set accurate. And after that, result set need to be imported to a report in a particular format, where the database importer component can read the report and store the vulnerability results into the database. In summary below activities will be performed manually.

- Upload the crawled source code sample to the static analysis tool
- False positive analysis
- Import the vulnerabilities found by the static analysis tool, to a report

Below is the high-level overview of the complete system. The diagram shows all the components of the proposed system and how each component is going to integrate with other components to provide the necessary output of the proposed system.

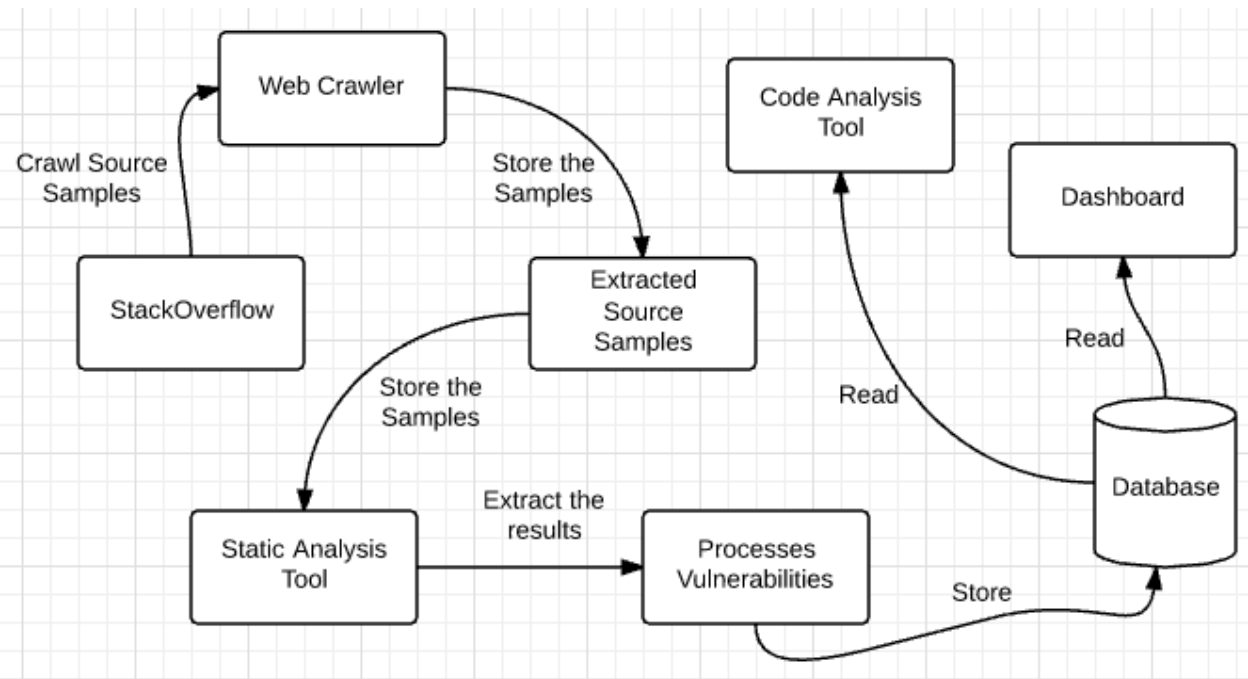


Figure 3.1 : System Overview

3.2.1 Web Crawler

To start the process of the application, it needs collected source codes from the open forum. Manually browsing the open forum and copying and saving the code sample to the local hard-disk is time consuming and practically not possible when the required number of source samples are high. Best way to solve this and automate the process is to create a web crawler. The web crawler component is used to read the source code samples from the stack exchange and store into a file in the local hard disk. There is no requirement of writing a web crawler from scratch, since the readymade crawlers are available and can be used without paying for it. Scrapy is a python based application framework for crawling web sites and scrapy is the crawling framework used for the application. The framework is simple and fast, which used for many purposes including data mining. Main advantage of scrapy is that it sends and processes requests asynchronously, which means it can do crawling very fast. Also, it allows customizations such as delaying between requests, limiting request to a particular ip address and auto throttling [39].


```

import scrapy

class QuotesSpider(scrapy.Spider):
    name = "quotes"
    start_urls = [
        'http://quotes.toscrape.com/tag/humor/',
    ]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text': quote.css('span.text::text').extract_first(),
                'author': quote.xpath('span/small/text()').extract_first(),
            }

        next_page = response.css('li.next a::attr("href")').extract_first()
        if next_page is not None:
            next_page = response.urljoin(next_page)
            yield scrapy.Request(next_page, callback=self.parse)

```

Figure 3.2 : Sample crawler using Scrapy

3.2.2 Process Vulnerabilities and Store

Next important step is to read the output report, process the vulnerability data and store the data into a database. This component should provide couple of features like, it should allow the user to point the report file generated by the static analysis tool, it should be able read the output report given by the static analysis tool, process it and also convert the data to a format where the data can be saved to a relational database and finally save the data into the database to create the knowledge base. This is a very important part of the project, because creating the knowledge database is the most important phase of the project and base for end user components as well.

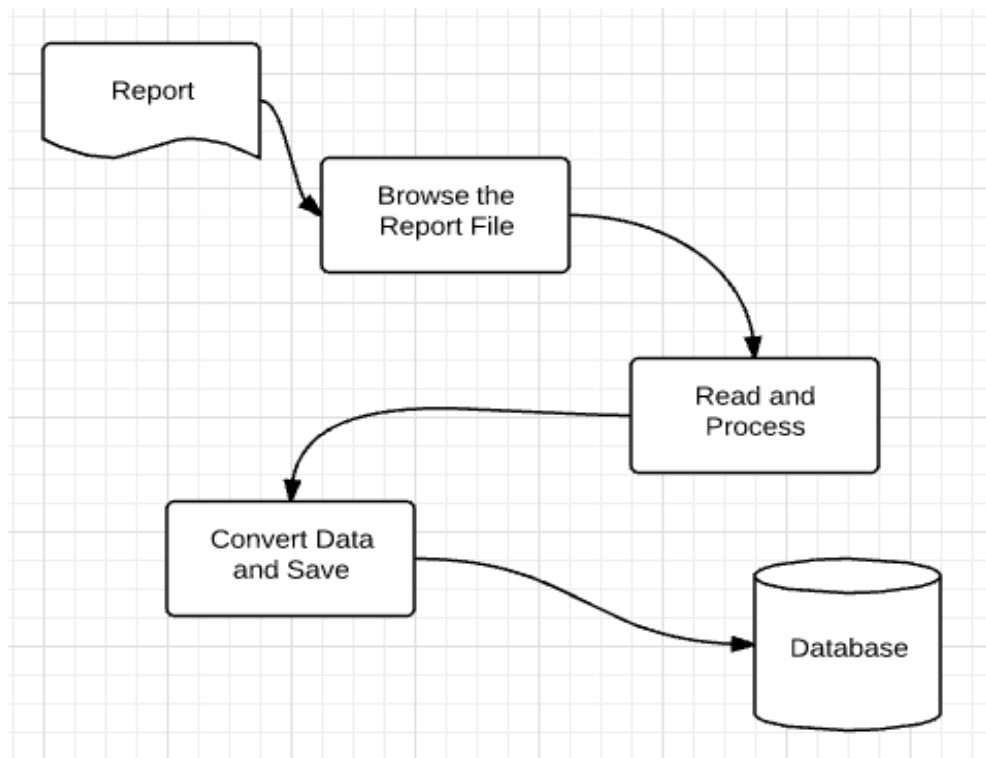


Figure 3.3 : Process Vulnerabilities and Store

3.2.3 Dashboard

Dashboard is a graphical component with various graphs, which mainly focuses on senior engineers, development managers, software architectures and also project managers. This component will showcase all the security vulnerabilities related to the scanned source code examples using the created knowledge base. Various graphs and charts will be used to give the information to the community, so they can take the actual benefit from the system. Couple of major advantages provided by this component are as follows.

- Developers can use the data to understand what are the common issues with each technology and what are the things and areas need to be considered to develop a secure software.
- Senior engineers and software architects can take an advantage of these statistics when performing manual code reviews and peer reviews. They can be decided which areas need to more focus and attention.
- Software architects and technical managers can use these data when selecting a particular technology for a product development.
- Quality engineers can use these data to decide what are the areas that needed more focus and also to create test cases and misuse cases.
- Project managers and development managers can refer the statistics do get an idea about how much effort the testing and peer reviews needed.
- The organization can leverage these data to decide what are the training that the developers and quality engineers needed to deliver secure product.

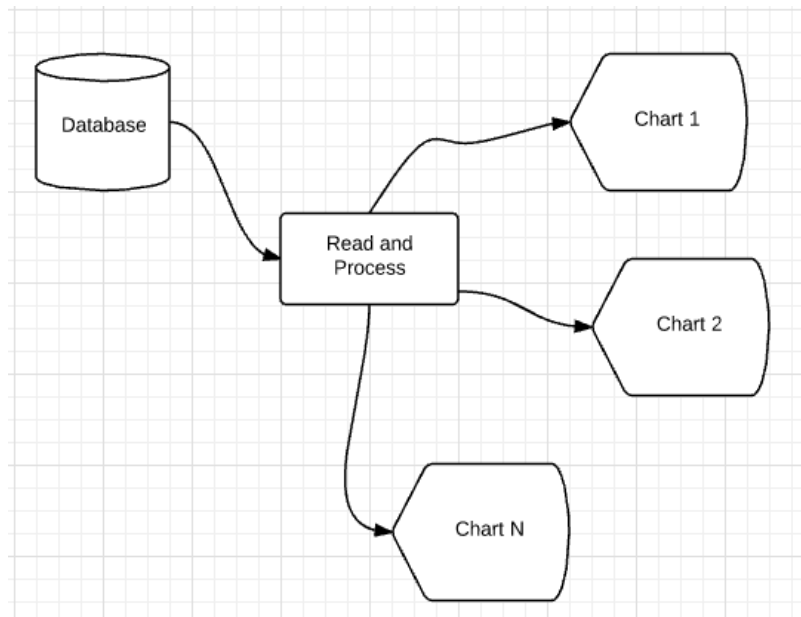


Figure 3.4 : Dashboard - Process Vulnerabilities and Display Charts

3.2.4 Code Analysis Tool

This is the most important component of the project and the interface that the developer can use to analyze the potential vulnerabilities of a particular piece of source code. This is where the developer and the community get the actual advantage of the project and this component is the solution for the problem that the project is trying to address. When a developer wants to use an entire or part of the source code published in the open forum, there is no easy way of making sure that the particular source sample is secured or vulnerability free, or if it is vulnerable what are the vulnerabilities, risks and what are the ways to address those vulnerabilities. The code analysis tool is the component that addresses this issue. The tool can help during whenever the developer wants to check whether the source sample published in the open forum is having any security vulnerabilities or not. The tool can connect to the created knowledge base to analyze the potential security vulnerabilities of the selected source code sample and give the feedback to the developer in a user-friendly manner.

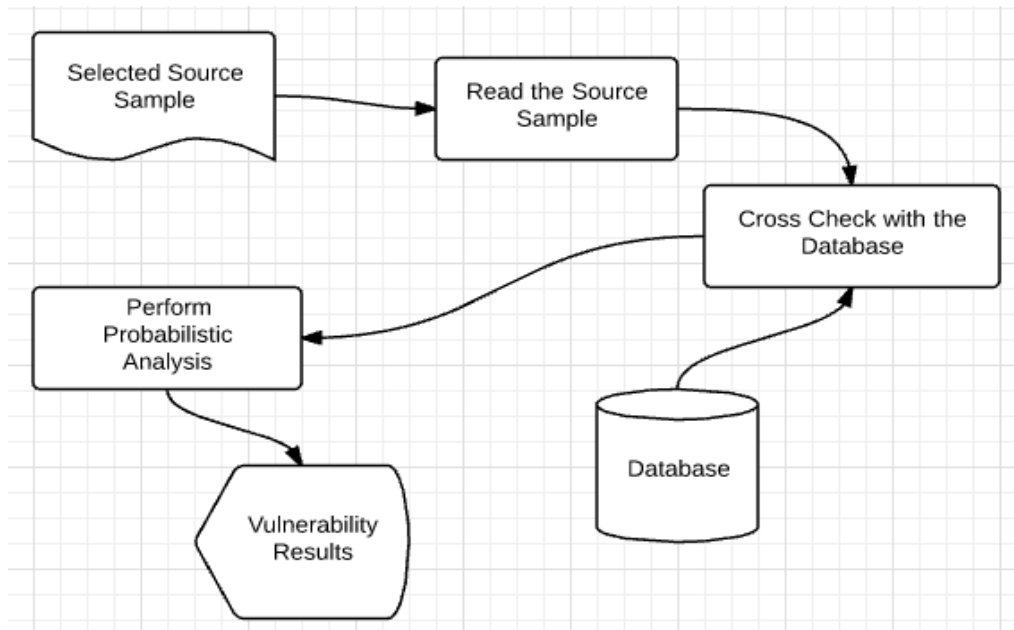


Figure 3.5 : Code Analysis Tool

3.2.5 Relational Database

This is where the application stores the vulnerability data and also the application itself uses it as the knowledge base. Because of the data model is relational, during the project design, it is decided to use a relational database system to store the required data. During the design of the database, the main considerations were, how easy is to store the processed vulnerability data and how efficient is to retrieve data during the end user is accessing the data. Also, to make the dashboard faster, separate table is used with all the vulnerability data.

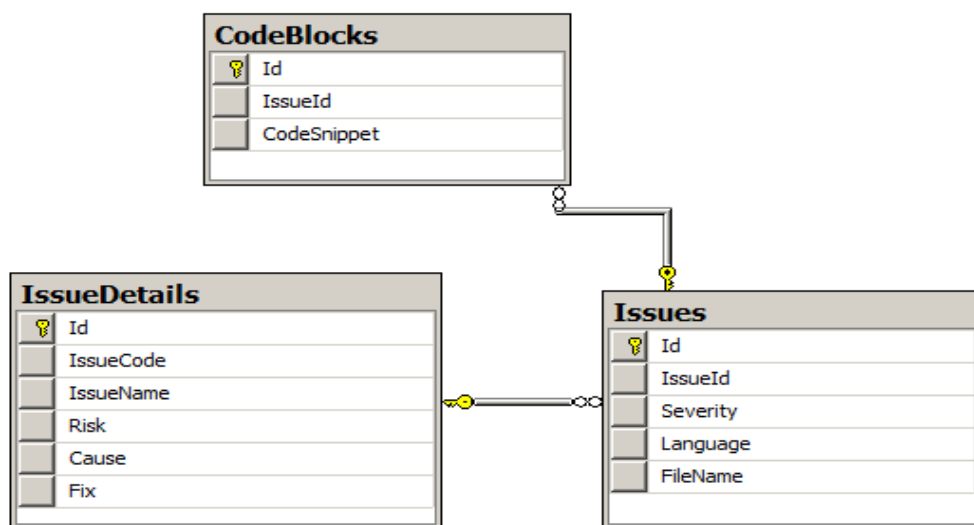


Figure 3.6 : Database diagram

Backlog	
🔑	Id
	Severity
	VulnerabilityType
	AppPlatform

Figure 3.7 : Table used for graphs and charts

3.2.6 Static Analysis Tool

Project required a commercial static analysis tool to perform vulnerability assessment of the crawled source code samples. Also, it is practically impossible to purchase a commercial tool for the project because of these tools are very expensive. For example, static analysis tool named Checkmarx is 1500 US Dollars. During the static analysis tool selection process, mainly considered the analysis done as a part of the project to understand the features, capabilities and the differences of the static analysis tools and easiness of use and possibility of getting a sponsor from an organization. Specially looked for a sponsorship, that someone can allow the source code samples to be scanned and get the vulnerability results. Open Source analysis tools were the last option because of the language support is limited and the accuracy level also not that satisfactory.

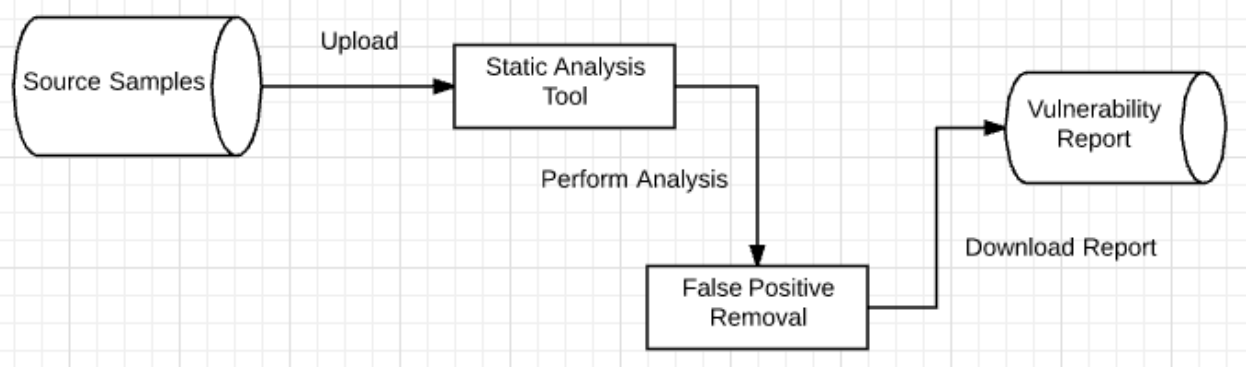


Figure 3.8 : Static code analysis tool

3.2.7 Open Forum

Project needs to select an open forum among the available open forums to extract the source sample to perform the vulnerability assessment. Analysis done on open forms and mainly considered how popular the open forum among the development community, what are the programming languages discussed within the open forum, how frequently users post questions

and answers, how many users visit the forum during an hour and what the experiences of the users who visit the forum. Also, the availability of the forum and whether any restrictions imposed by the open forum for crawling the source samples also considered, because if there is a restriction of crawling and downloading the source code, project cannot use that particular open forum

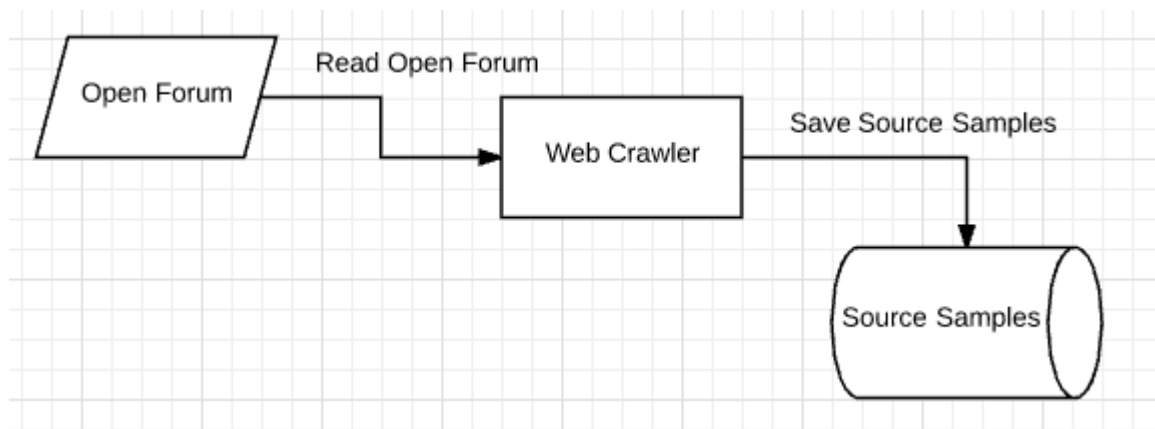


Figure 3.9 : Crawl Open Forum and store source samples locally

Chapter 4 : Implementation

4.1 Implementation Overview

After the design of the project is completed, next challenge will be to implement the project and also make sure implementation will achieve all the project requirements, specially the user friendliness and the efficiency. Most importantly, Implementation should not limit or completely restrict the required features of the project and implementation should enhance or facilitate to enrich the project features. Certain decisions need to be made to achieve the successful implementation of the project, including underlying technology, frameworks need to be used and back-end technology which is going to use. Primary focus should be, and it was to implement the project successfully rather than using the best or cutting-edge technologies in the industry.

4.2 Source Samples

Project needs an open forum with huge number of source code samples with all the technology categories. It should be used by developer in all technology categories, in all ages and in all experience levels. Also, the forum should be heavily used one. By considering all the factors during the analysis, found that stack overflow is a potential open form which is the site target for developer community under stack exchange umbrella. According to the static overflow statistics, 46 million people visited Stack Overflow in 2016 January and 16 million believed to be professional developers. Also, developer is posting a question in every 8 second, which is very high usage and indication of stack overflow extremely famous among development community and heavily used by the development community as well [15]. By considering these strong reasons, project decided to select the stack overflow as the open forum to crawl source samples (<http://stackoverflow.com>). Also decided to crawl the most recent source code samples to analyze the vulnerabilities to build the knowledge base. Below is how the static overflow publishing the developer questions and relevant answers.

c# regex matches example

▲ Am trying to get values using following text, any thoughts this can be done with Regex?

24 **Input:** Lorem ipsum dolor sit %download%#456 amet, consectetur adipiscing %download%#3434 elit. Duis non nunc nec mauris feugiat porttitor. Sed tincidunt blandit dui a viverra%download%#298. Aenean dapibus nisl %download%#893434 id nibh auctor vel tempor velit blandit.

★ **Output:**
3
456
3434
298
893434

Thanks in advance.

c# regex

Figure 4.1 : StackOverflow sample question

Every question is tagged with the particular technology, in this case it is c-sharp and with the area of technology the question is belongs to, it is regex in this case. Which makes it easy for the developers to find the details.

▲ All the other responses I see are fine, but C# has support for named groups!

21 I'd use the following code:

```
const string input = "Lorem ipsum dolor sit %download%#456 amet, consectetur adipiscing %dow\n\nstatic void Main(string[] args)\n{\n    Regex expression = new Regex(@"%download%#(?<Identifier>[0-9]*)");\n    var results = expression.Matches(input);\n    foreach (Match match in results)\n    {\n        Console.WriteLine(match.Groups["Identifier"].Value);\n    }\n}
```

The code that reads: `(?<Identifier>[0-9]*)` specifies that `[0-9]*`'s results will be part of a named group that we index as above: `match.Groups["Identifier"].Value`

Figure 4.2 : StackOverflow sample answer

4.3 Programming Languages to Select

The programming languages or the technologies that the project needs to focus on is another important factor to consider. Also, how many sample codes that the project is going to consider from each programming language is another important factor. Because it is practically impossible to consider all the available programming languages and all the available samples. So, it is necessary to set expectations for these two parameters first. By considering the popularity of the modern web and mobile application development languages and also considering the top vulnerable programming languages, project decided to consider five programming languages and consider at least 5000 source code samples from each language to perform the vulnerability analysis. Below is the list of considered programming languages.

- Python
- Java
- C-Sharp
- PHP
- JavaScript

4.4 Source Samples Crawler

Web crawler is actually the second most important supporting components for the project. There are many web crawler frameworks available for free and there is no requirement of writing specific one for the project. Because of that, a python based, well known framework named Scrapy is used to develop a crawler to read the source samples from StackOverflow and save the samples locally. StackOverflow has a URL format for each language to list down the questions posted by developers, and using that URL to crawl is very easy. URL format is simple and language can be specified as python, php or java. Also, the page number and the page size can be specified. Using the page number, it is possible to navigate through the pages and using the sort parameter, it allows to retrieve the latest posted questions into the first page and so on. Below is the sample URL format.

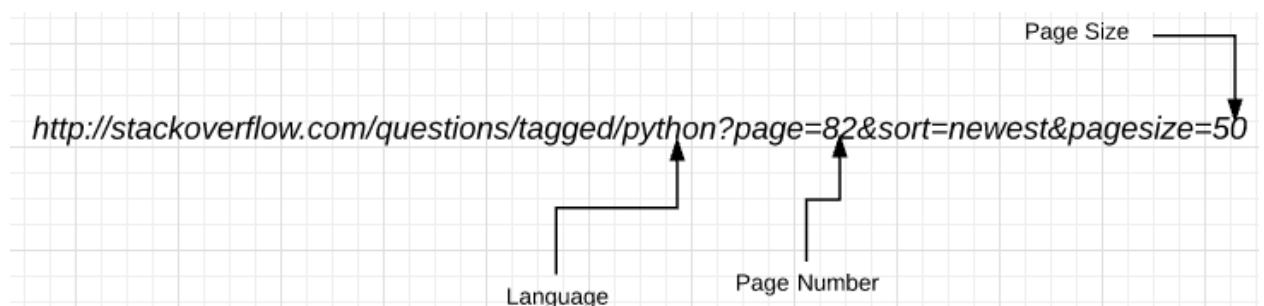


Figure 4.3 : StackOverflow Posted question URL format

Below is the python code written implement the crawler using Scrapy framework. Script has a separate section to identify the language and save the source code sample with the correct extension. And the script crawl the URL assign to start URLs and save the source code samples to the folder name assigned to code directory variable with the correct file extension.

```
import os
import scrapy
from scrapy.selector import Selector

class Language():

    def __init__(self, language=None, extension=None, comment=None):
        self.language = language
        self.extension = extension
        self.comment = comment

    def code_type(self, language):
        if language == "python":
            self.language = "python"
            self.extension = ".py"
            self.comment = "#"
        elif language == "java":
            self.language = "java"
            self.extension = ".java"
            self.comment = "//"
        elif language == "node.js":
            self.language = "nodejs"
            self.extension = ".js"
            self.comment = "//"
        elif language == "perl":
            self.language = "perl"
            self.extension = ".pl"
            self.comment = "#"
        elif language == "c++":
            self.language = "c++"
            self.extension = ".cpp"
            self.comment = "//"
        elif language == "c#":
            self.language = "csharp"
```

```

        self.extension = ".cs"
        self.comment = "//"
    elif language == "php":
        self.language = "php"
        self.extension = ".php"
        self.comment = "//"
    elif language == "javascript":
        self.language = "javascript"
        self.extension = ".js"
        self.comment = "//"

class StackOverflowSpider(scrapy.Spider):
    name = 'stackoverflow'
    code_directory = None
    start_urls =
['http://stackoverflow.com/questions/tagged/javascript?page=93&sort=newest&pagesize=50']

    def __init__(self):
        self.code_directory = "source_code"
    def parse(self, response):
        print "*****\n\n"
        print "Executing\n\n"
        print "*****\n\n"
        #Create the language directory if it doesn't exists
        try:
            os.stat(self.code_directory)
        except:
            os.mkdir(self.code_directory)
        for href in response.css('.question-summary h3 a::attr(href)':
            #Parse out the URL's to request
            full_url = response.urljoin(href.extract())
            yield scrapy.Request(full_url, callback=self.parse_question)

    def parse_question(self, response):
        rep = response.css('.accepted-answer')

        base_url = response.url

```

```

print "***#####"
print base_url
print "Parsing"

#Select the code language for each of the coding samples
code_sample_lang = response.css('.post-
tag').xpath('text()').extract_first()
lang = Language()
lang.code_type(code_sample_lang)

#Create the language directory if it doesn't exists
try:
    os.stat(self.code_directory + "/" + lang.language)
except:
    os.mkdir(self.code_directory + "/" + lang.language)

answers = 1
for s in response.css('.answercell pre code'):
    mycode=s.extract()
    mycode=mycode.replace('<code>','')
    mycode=mycode.replace('</code>','')

    mycode=mycode.replace('&gt;','>')
    mycode=mycode.replace('&lt;','<')

    id = base_url.split("/")
    filename = self.code_directory + "/" + lang.language + "/" + id[4] +
    "-" + str(answers) + lang.extension
    answers = answers + 1
    with open(filename, 'w') as f:
        #Comment the URL in the code
        f.write(lang.comment + "URL: " + base_url + "\n\n")
        f.write(mycode)

```

File name is generated using the id of the question and the number of the answer give to that particular question. Also in each save source sample file, there is a comment line added to with the full URL of the source sample. Below is a c-sharp code sample file.

```
24130650-2.cs x
1 //URL: http://stackoverflow.com/questions/24130650/scraping-data-dynamically-generated-by-javascript-in-html-document-using-c-sharp
2
3 private void webBrowserControl_DocumentCompleted(object sender, WebBrowserDocumentCompletedEventArgs e)
4 {
5     HtmlElementCollection divs = webBrowserControl.Document.GetElementsByTagName("div");
6
7     foreach (HtmlElement div in divs)
8     {
9         //do something
10    }
11 }
12
```

Figure 4.4 : Sample source code file

This script need to be run in the command line using below command.

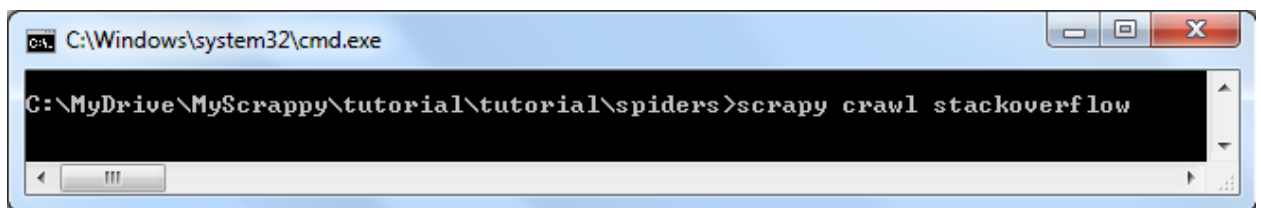


Figure 4.5 : Running the web crawler

One practical issue faced was that, after crawling couple of hundred code samples, StackOverflow blocked the ip of the computer for 10 or 15 minutes. So, had to patiently wait and slowly crawl the source codes and save to the local hard disk.

4.5 Static Source Analysis Tool

After studying several available commercial static analysis tools, by considering the scenario of the project it was clear that the project need a tool to analyze the raw source codes. So, due to that, Veracode is not usable with the project. When consider the features of the available commercial tools, Checkmarx was a better fit with lots of support, but it was a very expensive tool. Project managed to find a sponsor for the Checkmarx tool and decided to use that as the static code analysis tool to perform the static analysis against the downloaded source code samples. Source codes needed to be compressed to a zip file and uploaded to the Checkmarx for it to perform the analysis. There is a possibility that the uploading process also can be automated, but for the project, decided to upload it manually. Separate zip file is created for each programming language to make it convenient. Below are the zip files created and ready to upload to the tool.

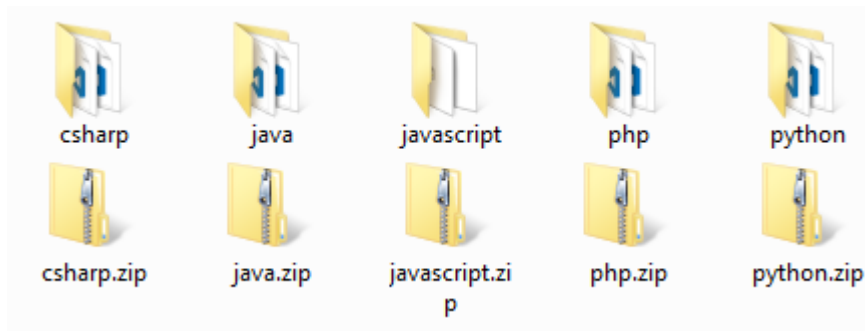


Figure 4.6 : Sample source code files are ready to upload

With Checkmarx, first thing is to create a project for the scan, and then after navigating into the project, there is an option called full scan. By clicking on that option Checkmarx will allow the user to upload the created zip file. Below is the screenshot of upload zip file for full scan option. There is another option called incremental scan, which means the tool will scan only the changed or newly added files to discover vulnerabilities.

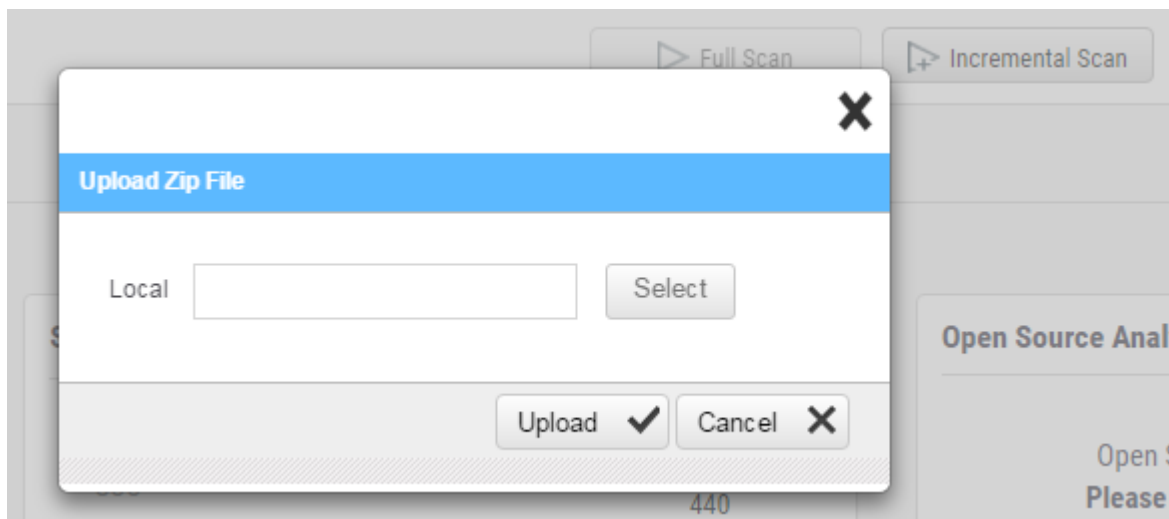


Figure 4.7 : Checkmarx upload zip file for scan

After uploading the zip file, Checkmarx will queue the scan job and perform it based on the availability of its resources.

QUEUED DATE ▼	INITIAT...	ORIGIN	PROJEC...	SERVER...	LOC	STATUS	ACTIONS
2/21/2017 10:11:57 PM	Thushar...	Web Po...	Test_Th...		108676	Queued	🔄 🗑️

Figure 4.8 : Checkmarx scan queue

As mentioned above, Checkmarx is a very expensive tool and had to use it very carefully without interrupting other projects. So, this was a bottleneck for the project, since Checkmarx gave extremely low priority for the project related scans and also had to divide 5000 samples into to 200 chunks and perform the scan to reduce the stress to the Checkmarx. Checkmarx has couple of interesting viewers. Current status of a particular project is very important, where it showcase all the vulnerabilities discovered within that project. The issues viewer helps to navigate through all the discovered issues and also it is possible to mark the vulnerability as false positive after studying about the issue. Also, there is an option available to import the Checkmarx discovered vulnerabilities to a report and couple of formats are supported including PDF, csv and xml.

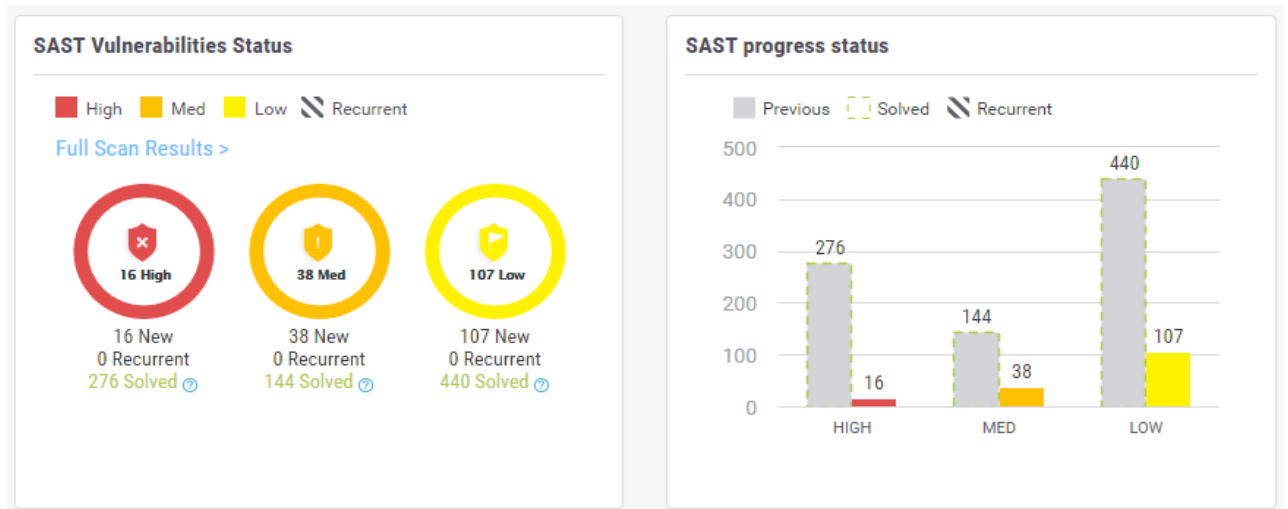


Figure 4.9 : Project overview

Below is the issue viewer component of Checkmarx which can be used to view issues and issue details.

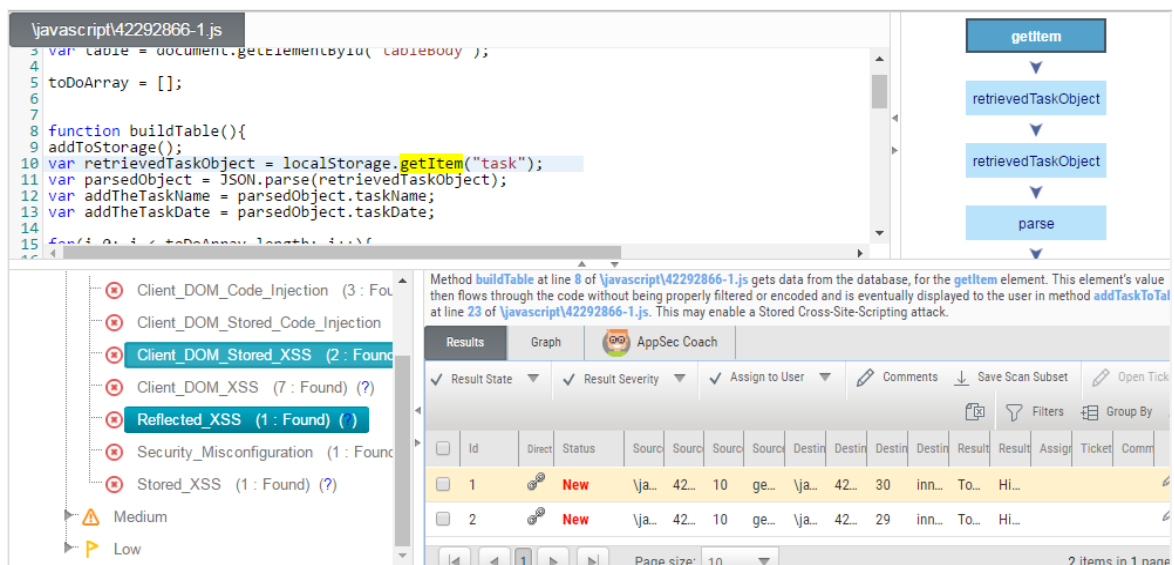


Figure 4.10 : Issue viewer

Below is the report generator component of Checkmarx and it supports couple of report formats and the user can export the reports to any supported format.



Figure 4.11 : Import vulnerabilities to a report

For the project, Checkmarx vulnerabilities list was imported to a XML report. Programmatically it is extremely easy to handle or process XML documents and that was the primary reason to choose XML format. Below is the exported list of Checkmarx xml reports.

Name ^	Date modified	Type	Size
csharp.xml	3/2/2017 3:14 PM	XML Document	74 KB
csharp_init.xml	12/12/2016 10:07 AM	XML Document	10 KB
Java.xml	3/2/2017 1:10 PM	XML Document	710 KB
JavaScript.xml	3/2/2017 12:25 PM	XML Document	360 KB
php.xml	3/2/2017 1:08 PM	XML Document	1,808 KB
Python.xml	3/2/2017 12:27 PM	XML Document	173 KB

Figure 4.12 : Imported vulnerabilities to XML documents

4.6 Vulnerability Importer

This component will read the XML report, which is imported from Checkmarx tool and process it and then save the vulnerability data to the relational database. Another important supporting component of the project to achieve its goal. To develop this component Microsoft csharp, a powerful programming language within the .net family is used and a windows forms application is created. Even though, this is not an external user facing component, developed it with a simple and easy user interface. Also implemented proper error handling and informational messages to make it user friendly. Since the component uses entity framework with the importer, it can roll back the changes, so, there will not be any harm for the data stored in the database, in case of an error.

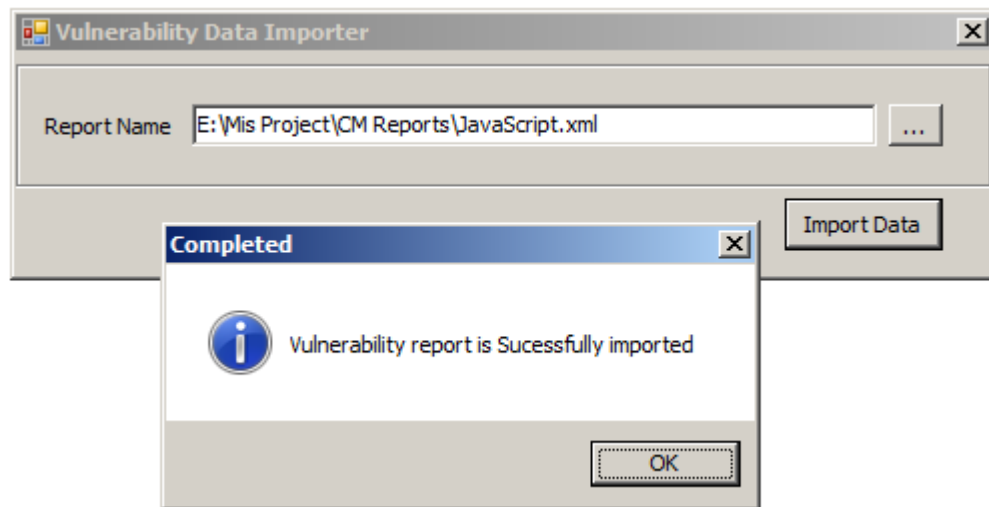


Figure 4.13 : Vulnerability report importer

Below is the code block to read the XML report and serialize to relevant CSharp classes.

```
using System.Xml.Serialization;  
using System.Xml;  
  
XmlSerializer serializer = new XmlSerializer(typeof(CxXMLResults));  
CxXMLResults resultingMessage = (CxXMLResults)serializer.Deserialize(new  
XmlTextReader(filePath));
```

4.7 Dashboard

This component is defined as the second most important component of the project. This is where the project utilizes the discovered vulnerabilities by reading the knowledge base, to project with various charts and graphs. Main target is to make the dashboard very informative and user friendly. Again, a powerful language and easy to user web framework, Asp.Net MVC with csharp is used to develop the dashboard. Asp.Net MVC is easy to use web framework with less learning curve, which was ideal to develop and was an ideal framework for the project. For the charts and graphs, google chart API is used, since it is very convenient, powerful and contains all the required chart types. Microsoft solution for object relational mapping, name entity framework is used with Aps.Net MVC application to access the database where the vulnerabilities are stored. By using the entity framework, managed to cut down the development time significantly and also managed to develop the dashboard component with clean source code.

Below are some of the charts included in the dashboard.

Top 5 Application Security Vulnerabilities

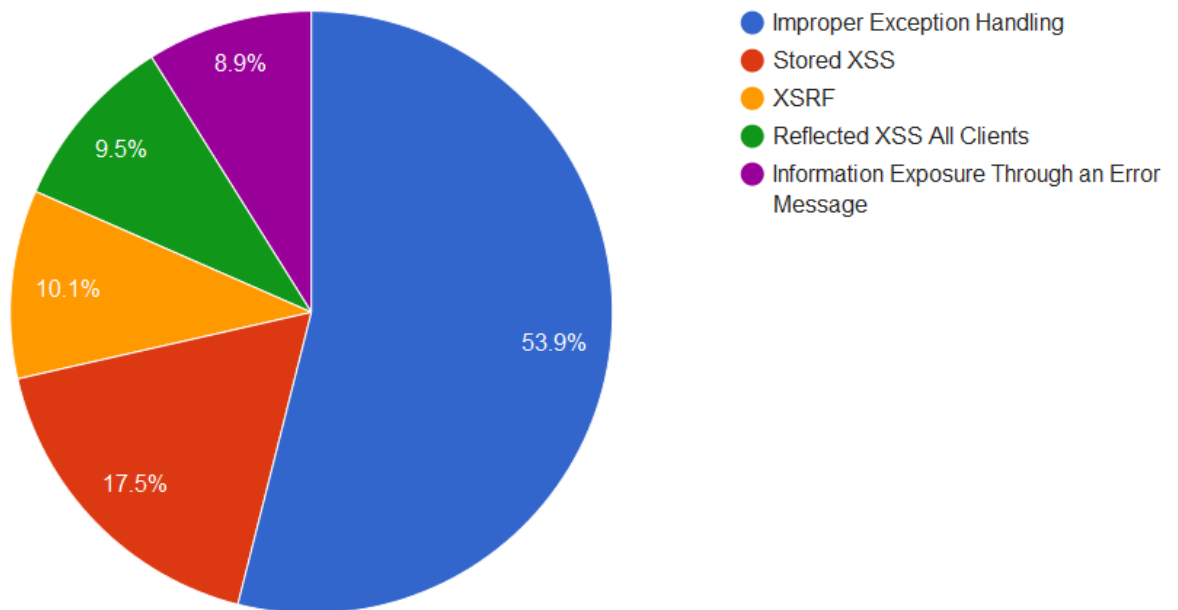


Figure 4.14 : Top 5 Vulnerabilities

Application Security - Vulnerabilities by Platform

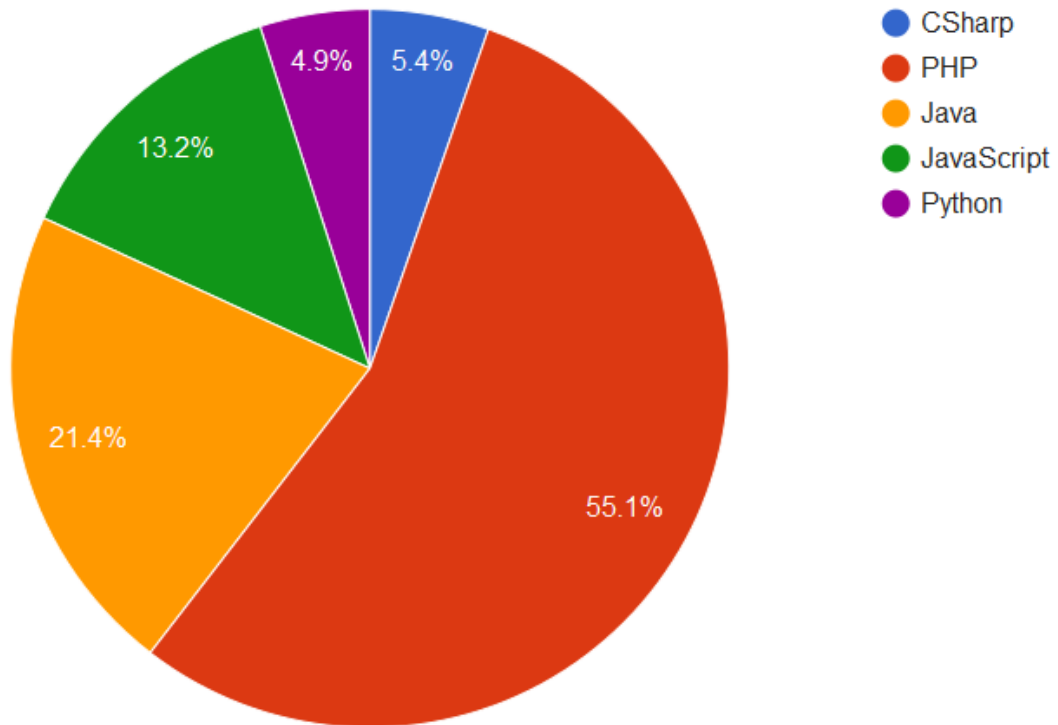


Figure 4.15 : Vulnerabilities by Platform

4.8 Code Analysis Tool

Code analysis tool is the most important component of the project. This component allows the developer or any other user to verify, whether there are any security vulnerabilities with a particular source code block in StackOverflow. User should be able to select the particular source code block and analysis tool should be able to read the user selected source code block, use the knowledge base to analyze it, and show the potential vulnerabilities to the user. This is main workflow the component needs to cover. When analyzing the scenario, it was clear that, StackOverflow is a web site and user has to use a browser to access it and see the sample source codes published in it. By considering the situation, it is very much clear that best solution is to develop a browser plugin to capture the user input. After analyzing the situation more and considering the user friendliness, decide to add a right click menu option, so the user can highlight a particular source code block and use the right click menu to send it for the analysis.

Also decided to choose a one particular browser and develop a browser plugin only for the selected browser. Project decided to select the google Chrome as the browser because it is famous among the development community and even with normal users and also Chrome provides a great support for developing plugins and it is reasonably easy with the help portal provided. Initially wanted to automatically get the entire source code block, but chrome plugin does not have a straightforward way of reading an entire html text inside a particular tag. So, decided to go with the text highlight and user can highlight the source code block and get the

right click menu. In a way, this method has an advantage, where the user can select a portion of the source sample and perform the analysis.

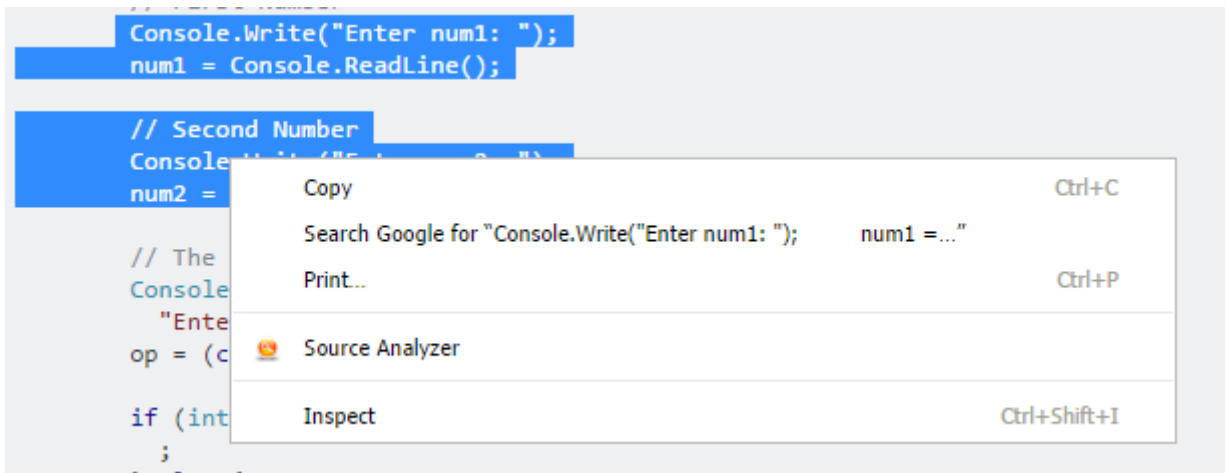


Figure 4.16 : Source Analyzer Chrome Plugin

There should be a service endpoint listening all the time to accept the user requested, collect the user selected source code sample and response back to the user with the potential vulnerabilities exists within the user selected source code block. Again, Asp.Net was used to develop a service endpoint with entity framework to access the knowledge base. Chrome plugin is issuing a cross origin XML, http request (*XMLHttpRequest*) using JavaScript code to send the user selected source code and retrieve results. Web servers do not allow cross origin calls by default and project needed to do the necessary changes make it possible. Below is the JavaScript code written to make the *XMLHttpRequest* call and process the results, with in the chrome plugin. <http://localhost:49362/Home/Analyze> is the endpoint implemented to accept the call from chrome plugin.

```
function getIssues(sourceCode)
{
    var xhr = new XMLHttpRequest();
    var url = "http://localhost:49362/Home/Analyze";
    var params = "CodeSelected="+encodeURIComponent(sourceCode);
    xhr.open("POST", url, true);

    xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

    xhr.onreadystatechange = function() {
        if(xhr.readyState == 4 && xhr.status == 200) {
            var resp = JSON.parse(xhr.responseText);
            document.getElementById("testsource").innerHTML =
                resp.ResultText;
        }
    }
    xhr.send(params);
}
```

From the Asp.Net side had to implement the below code to inform the web server to allow and accept cross origin client calls.

```
public class AllowCrossSiteJsonAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext
filterContext)
    {
        filterContext.RequestContext.HttpContext.Response.AddHeader("Access -
Control-Allow-Origin", "*");
        base.OnActionExecuting(filterContext);
    }
}
```

After reading the user selected source code block to analyze, source analyzer should cross validate it with the vulnerable code samples in the database. This is sort of a fuzzy string matching and after searching for available libraries for CSharp, powerful library found with CodePlex, which can perform approximate string comparison and decided to use with the application [42]. Below is the implementation, which uses the FuzzyString library from CodePlex to verify the user selected source code against the database.

```
public static bool IsCodeEqual(string userInput, string vulnerableCode)
{
    List<FuzzyStringComparisonOptions> options = new
        List<FuzzyStringComparisonOptions>();
    options.Add(FuzzyStringComparisonOptions.UseLongestCommonSubstring);
    return userInput.ApproximatelyEquals(vulnerableCode, options,
        FuzzyStringComparisonTolerance.Strong);
}
```

Source analyzer defines a confidence level of a detected potential vulnerability by analyzing the relevance between selected code block and the actual vulnerable code. For example, actual vulnerability may need more than one code lines, but the user may select only one line out of them for analysis. So, the tool should be able to detect it and shows the vulnerability with an appropriate confidence level. Simple logic has implemented to achieve this capability of the source analyzer and it is a great benefit for the user.

```

public Result AnalyzeSource(string selectedSource)
{
    List<Vulnerability> lst = new List<Vulnerability>();

    foreach (var issue in analyzeDb.Issues)
    {
        decimal fndCnt = 0;

        foreach (CodeBlock c in issue.CodeBlocks)
        {
            if(UtilitySvc.IsCodeEqual(selectedSource,c.CodeSnippet))
            {
                fndCnt++;
            }
        }

        if (fndCnt > 0)
        {
            Vulnerability v = new Vulnerability();
            v.CodeIssue = issue;
            v.Likelihood = Math.Round(((decimal)fndCnt /
                (decimal)issue.CodeBlocks.Count) * 100, 2);

            lst.Add(v);
        }
    }

    Result vulnerabilities = new Result();
    vulnerabilities.ResultText = GenerateView(lst);

    return vulnerabilities;
}

```

After calculating a percentage of likelihood, below logic will decide the confidence level and assign to the potential vulnerability.

```

public enum ConfidenceLevels
{
    Certain = 0,
    Firm = 1,
    Tentative = 2
}

public class Vulnerability
{
    public Issue CodeIssue { get; set; }
    public decimal Likelihood { set; get; }
    public string Confidence
    {
        get
        {
            string conf = string.Empty;

            if (Likelihood >= 80)
                conf = ConfidenceLevels.Certain.ToString();
            else if (Likelihood >= 40)
                conf = ConfidenceLevels.Firm.ToString();
            else if (Likelihood < 40)
                conf = ConfidenceLevels.Tentative.ToString();

            return conf;
        }
    }
}

```

Primary target was to develop the chrome plugin, user-friendly, convenient and self-understandable manner. Plugin view include below points with the vulnerability analysis to make it come convenient.

- Severity of the vulnerability
- Confidence level
- Vulnerable source code snippet
- Risk of the vulnerability
- Reason or the cause of the vulnerability
- General recommendations to fix the vulnerability

Below is the chrome plugin, which actually shows the potential vulnerability to the user with severity and confidence level above other information.

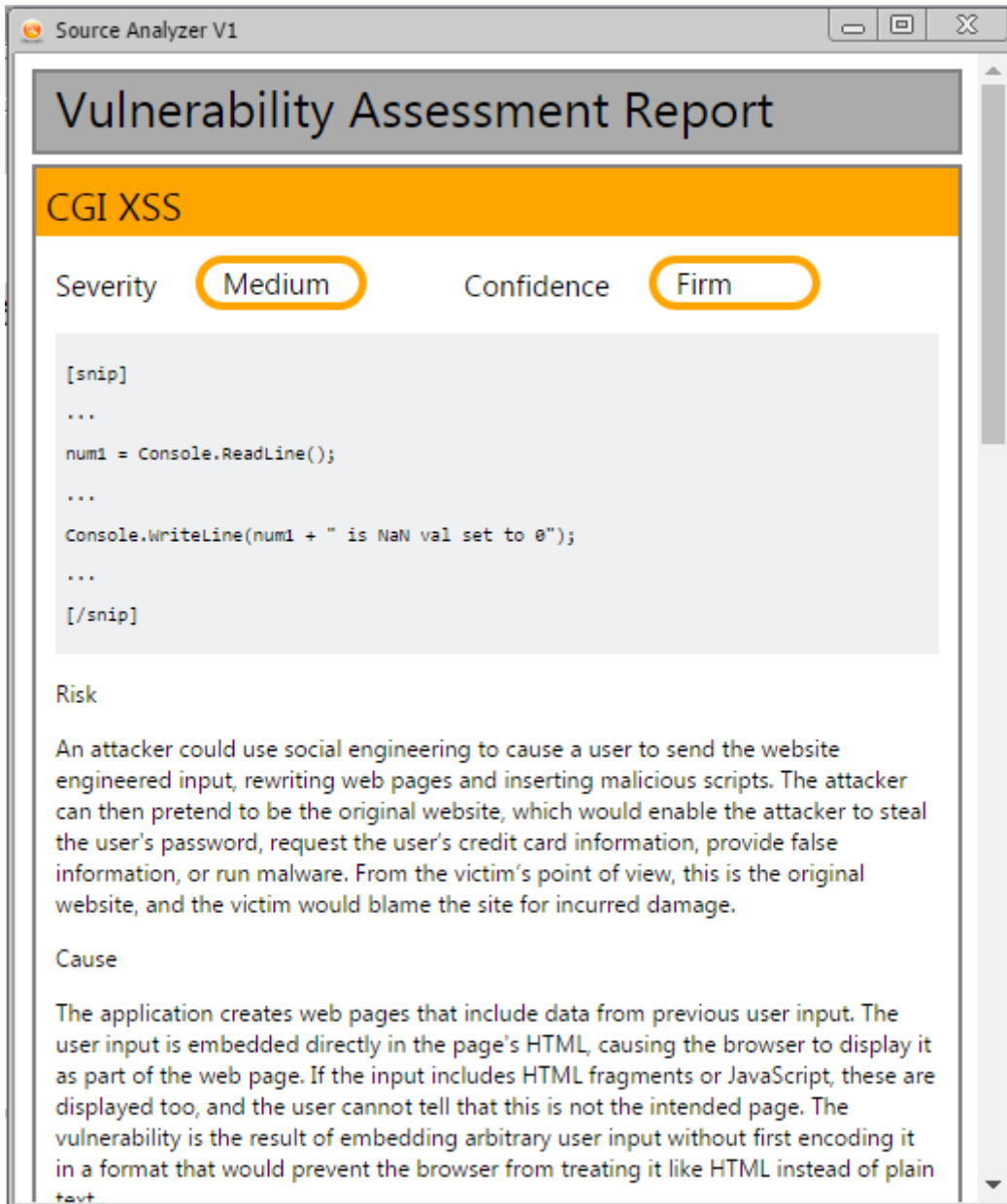


Figure 4.17 : Source Analyzer tool

4.9 Vulnerability Database

Project needed a relational database system to store the vulnerability data, which is provided by the Checkmarx. Considering other technologies used to develop the application, Microsoft SQL server is the best option and it is supporting with Microsoft technologies seamlessly. Also, Microsoft provides easy and very user-friendly studio which can be used to create and manage the database.

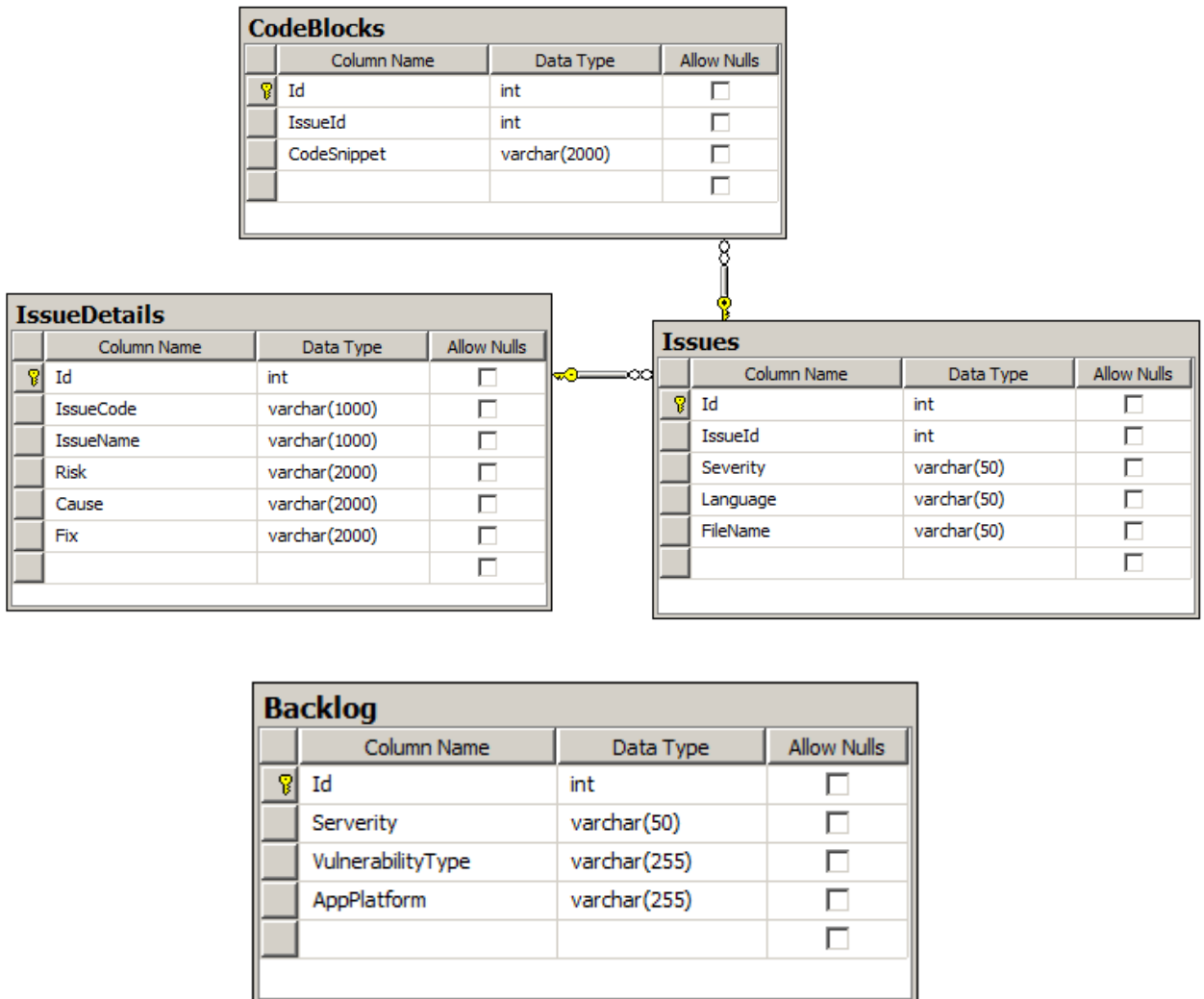


Figure 4.18 : Database implementation

Chapter 5 : Results, Testing and Evaluation

5.1 Introduction

This chapter will describe the results of the project, application features, functionalities and capabilities to evaluate the project, also possible approaches to test and verify the application functionality to make sure it provides the expected quality output. Below steps will be used to test and evaluate the developed application.

- Source code testing to make sure there are no errors and logically it is implemented as expected.
- Functionality testing.
 - Functionality testing of web crawler.
 - Functionality testing of the vulnerability exporter.
 - Functionality testing of the dashboard.
 - Functionality testing of the code analyzer.
- Usability testing.
 - Functionality testing of the dashboard.
 - Functionality testing of the code analyzer.

5.2 Results

Selected five different programming languages, that are c-sharp, java, php, python and JavaScript and analyzed 5000+ samples from each language using static code analysis tool, Checkmarx for this project. And this section is focusing on discussing the findings in couple of different angles. Altogether, 346131 lines of code had been uploaded to the Checkmarx and assessed for security vulnerabilities. In total tool managed to discover 1489 vulnerabilities belongs to various vulnerability categories. According to the Checkmarx, PHP is the language with highest risk and this is matching with the most vulnerable programming language research done by Veracode. Below is the risk level summary of each language provided by Checkmarx.






	RISK LEVEL SCORE		LOC
JavaScript	 (88)		108676
Python	 (61)		49840
PHP	 (100)		55336
Java	 (40)		63994
C-Sharp	 (38)		68285

Figure 5.1 : Risk level indicator of each language

Below is some more vulnerability information for each programming language, discovered by the Checkmarx tool. This information directly extracted from Checkmarx.

5.2.1 JavaScript



Figure 5.2 : Vulnerability categories - JavaScript

Above are the vulnerability categories discovered by Checkmarx and grouped by the severity of those categories. Checkmarx manage to find High, Medium and Low severity issues with JavaScript. Below diagram shows the top 5 vulnerability categories discovered by Checkmarks.

Top 5 Vulnerabilities

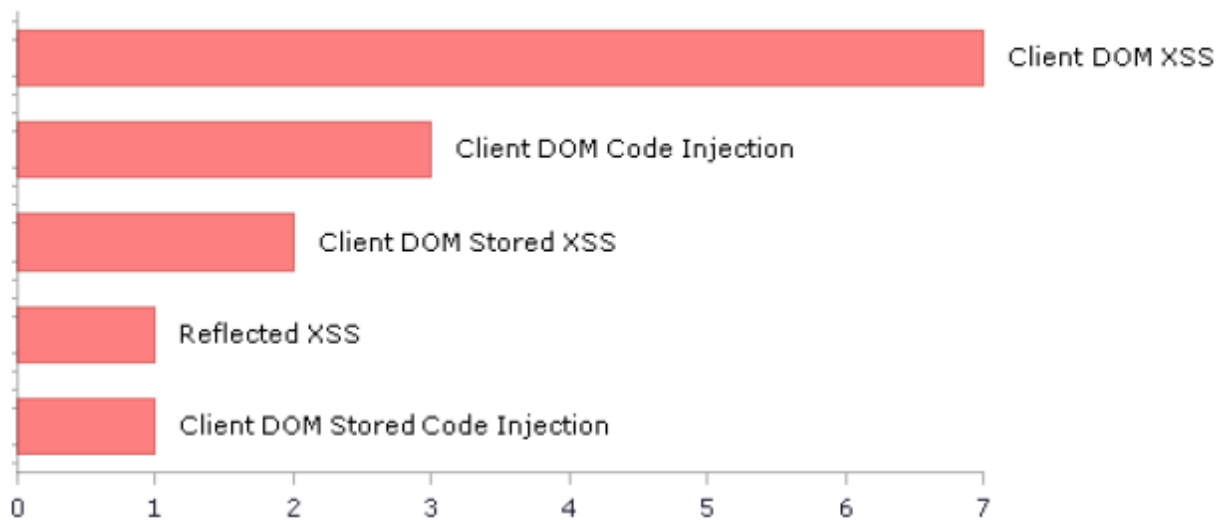


Figure 5.3 : Top 5 vulnerability categories – JavaScript

5.2.2 Python

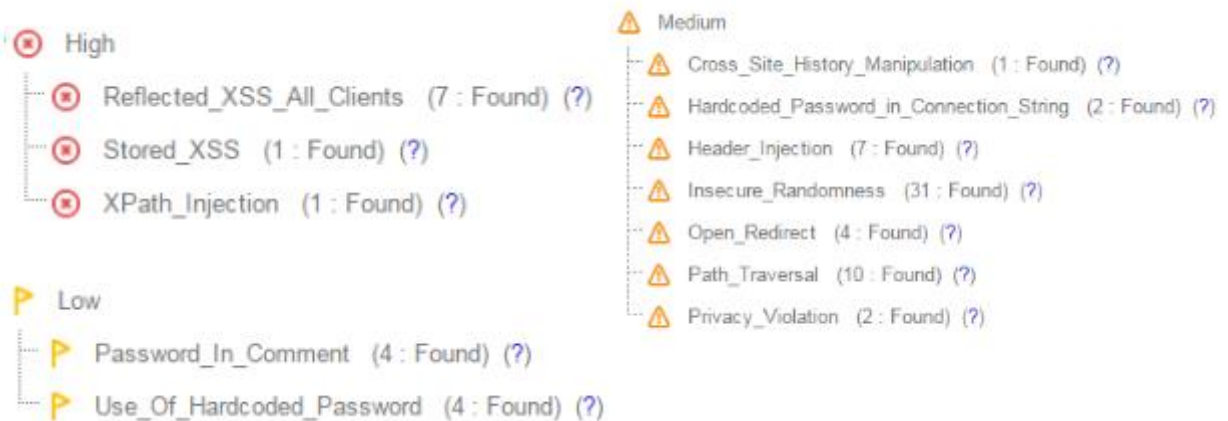


Figure 5.4 : Vulnerability categories - Python

Checkmarx manage to discover High, Medium and Low severity issues with uploaded python source code sample as well. Above chart shows the discovered vulnerability categories grouped by severity. Below diagram shows the top 5 vulnerability categories discovered by Checkmarks.

Top 5 Vulnerabilities

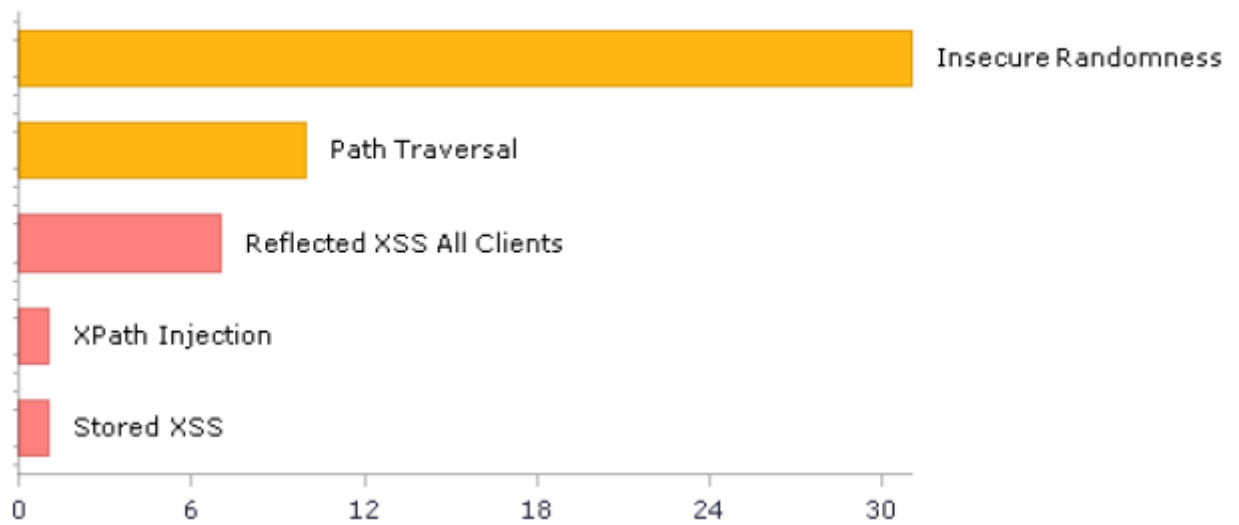


Figure 5.5 : Top 5 vulnerability categories - Python

5.2.3 PHP

According to the source code analysis done for the project, Checkmarx marked PHP as the highest risk programming language. During the analysis Checkmarx manage to discover some JavaScript code issues also, which were written inside PHP code. This is another advantage of Checkmarx because JavaScript are essential ingredient for web development and can be plug into any development language. With Checkmarx, developer does not need to worry about, because tool can automatically detect the programming language and perform the vulnerability assessment for using the appropriate rule set. Below is the summary of the vulnerability categories, Checkmarx discovered with JavaScript written inside PHP code, grouped by risk level.

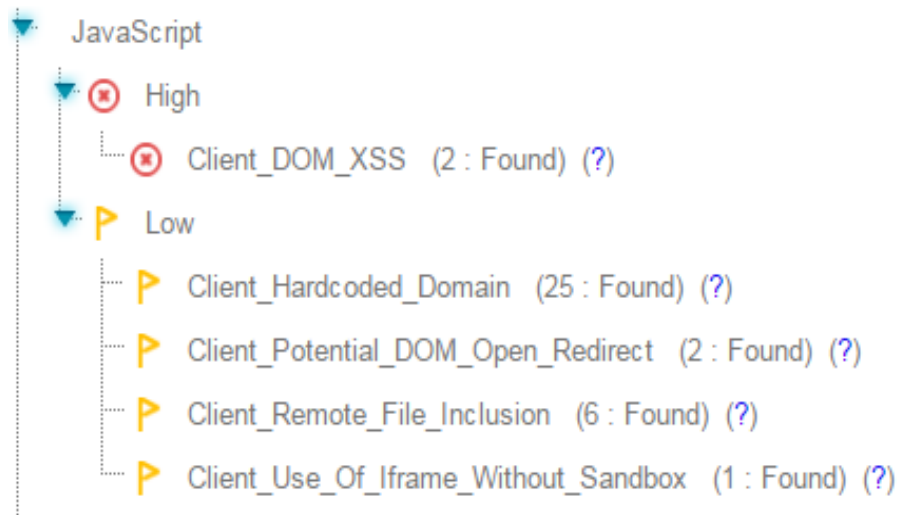


Figure 5.6 : Vulnerability categories - JavaScript within PHP

Below are the Top 5 vulnerability categories discovered with PHP.

Top 5 Vulnerabilities

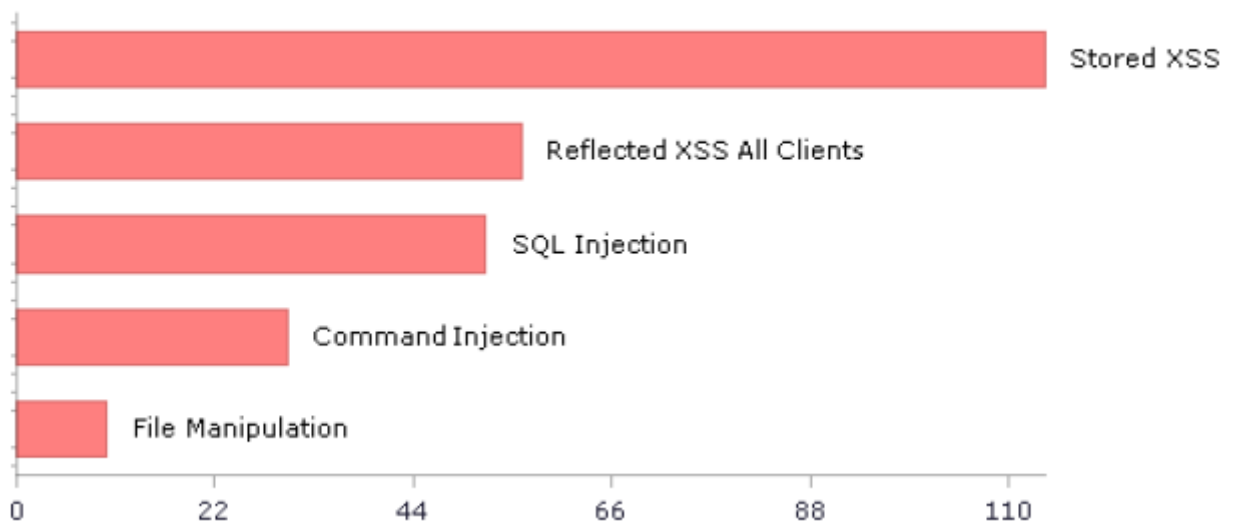


Figure 5.7 : Top 5 vulnerability categories - PHP

Below are the vulnerabilities found by Checkmarx and grouped by severity. With PHP, also the tools managed to discover High, Medium and Low severity issues.

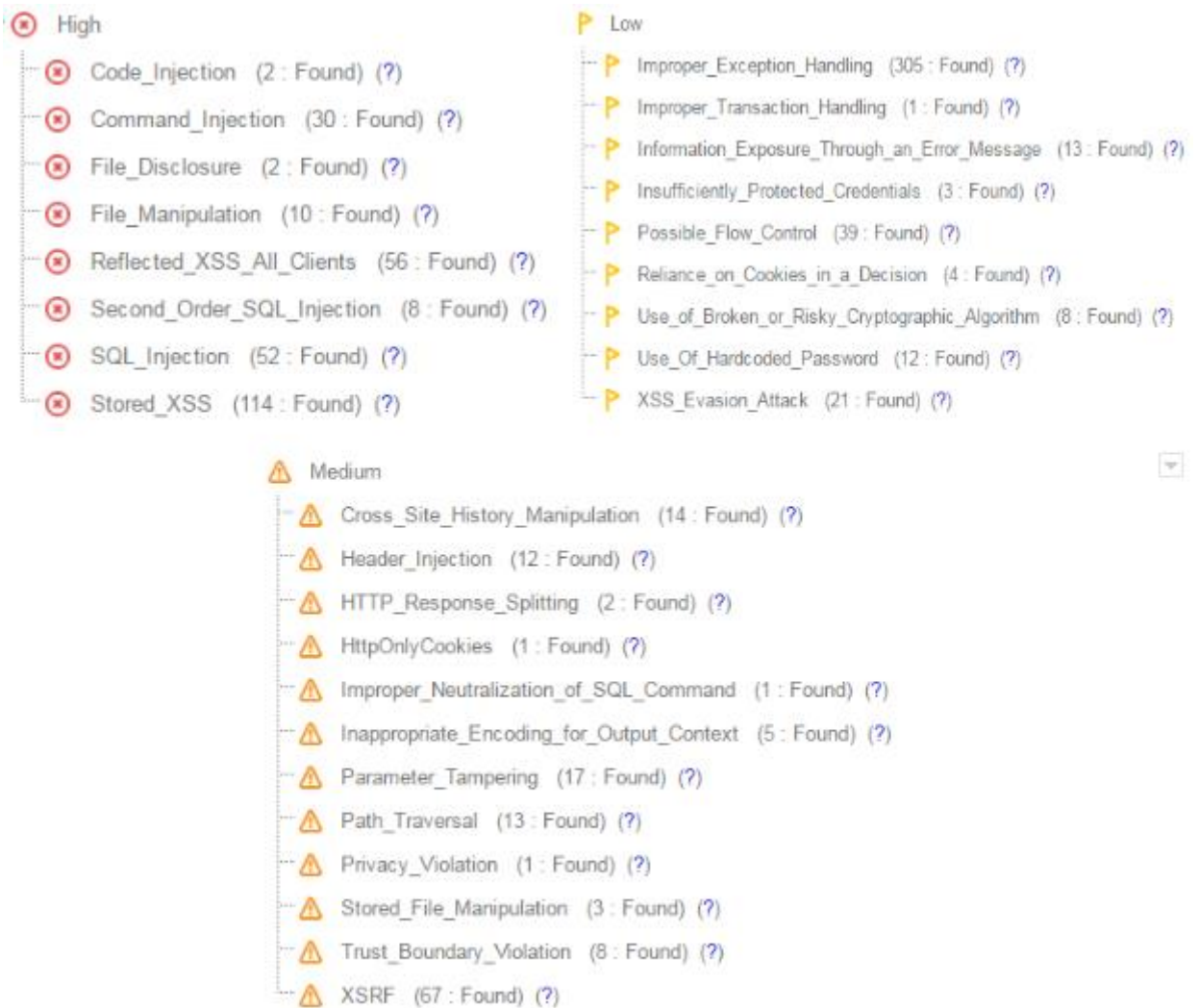


Figure 5.8 : Vulnerability categories - PHP

5.2.4 Java

Java is also widely used language for web and mobile development and Checkmarx managed to discover only Medium and Low vulnerability categories. This is sort of a medication to say that Java language code samples are safe when comparing to JavaScript or PHP samples. Veracode research report also mentioned that Java is a comparatively safe programming language. Below are the discovered vulnerability categories grouped by severity levels.

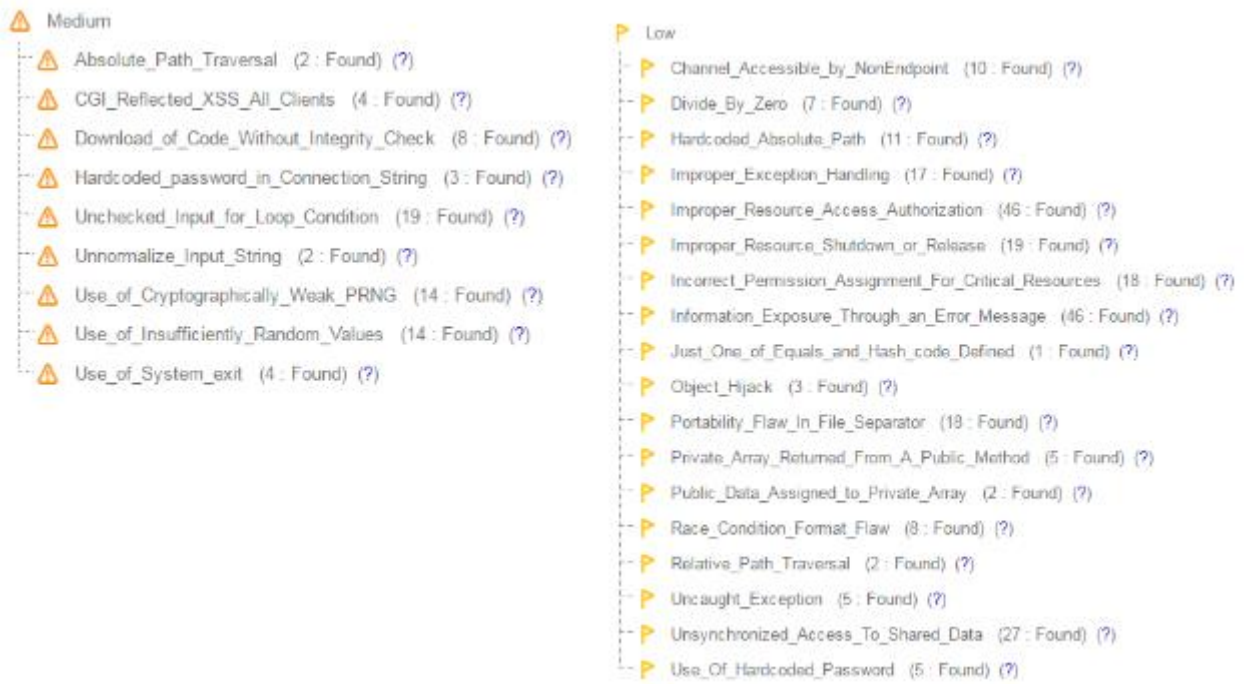


Figure 5.9 : Vulnerability categories - Java

Discovered top 5 vulnerability categories are as follows.

Top 5 Vulnerabilities

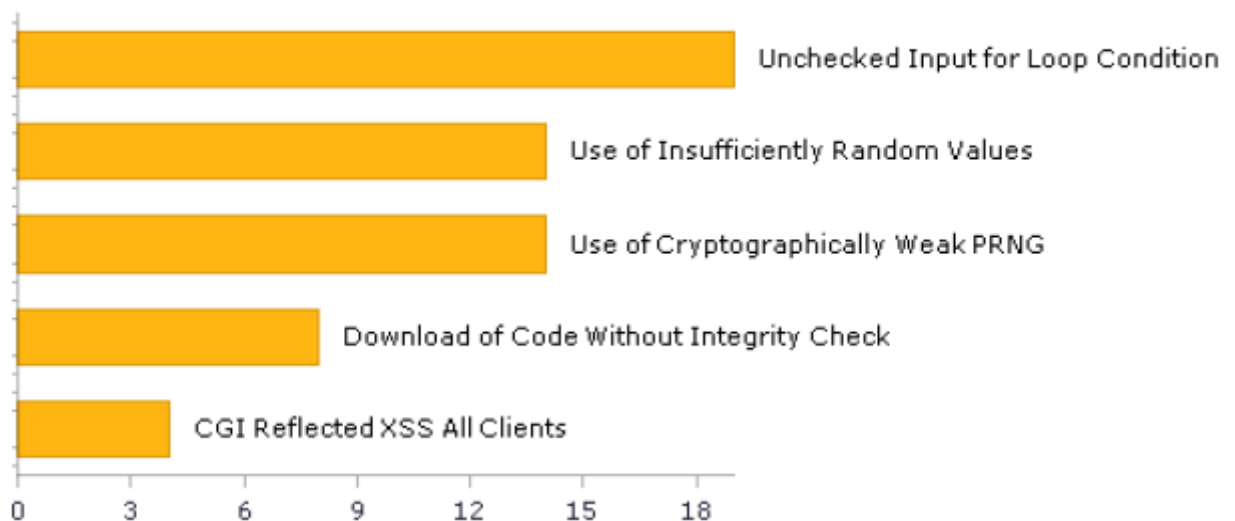


Figure 5.10 : Top 5 vulnerability categories - Java

5.2.5 C-Sharp

Veracode mentioned that .net languages are safe languages to develop software in their research report. Checkmarx also able to discover only Medium and Low severity issues with C-Sharp and comparatively lesser issues than other four languages. Below are the discovered vulnerability categories grouped by severity levels.

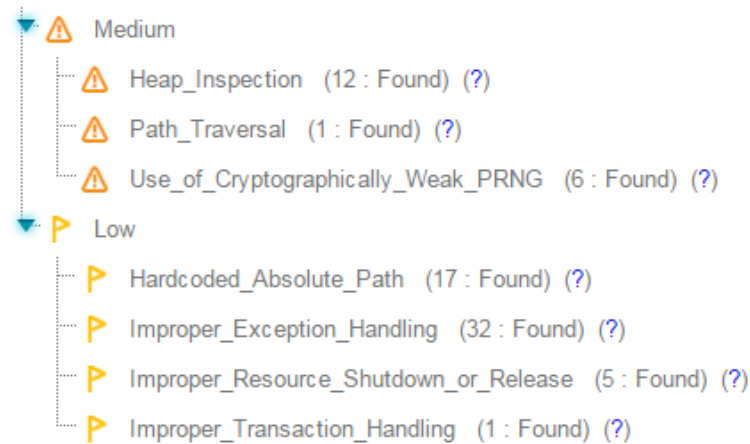


Figure 5.11 : Vulnerability categories - CSharp

Discovered top 5 vulnerability categories are as follows.

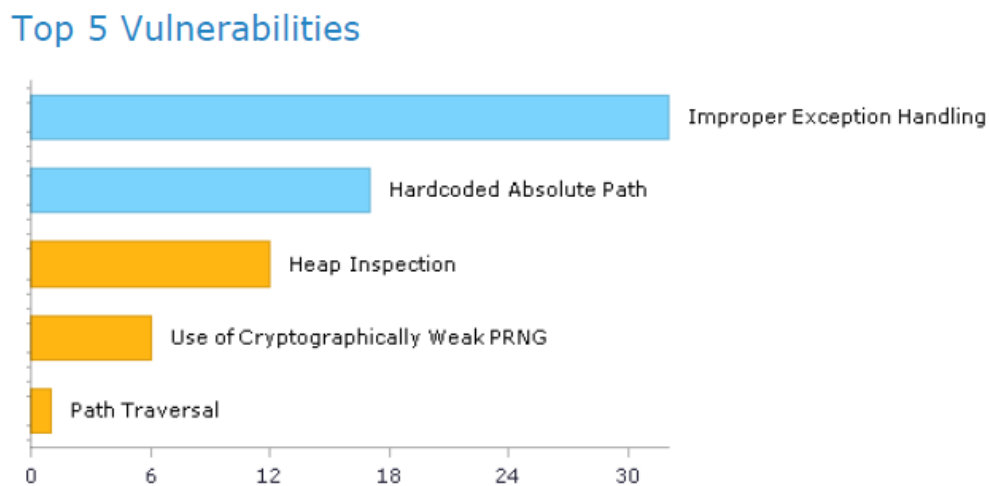


Figure 5.12 : Top 5 vulnerability categories - CSharp

According to the source code analysis results, it is clear that all the selected five languages has security vulnerabilities. Tool manage to detect even High severity vulnerabilities with some languages. With these findings it is proven that, is it highly important to discover the security vulnerabilities of these source code samples published in open forums.

5.3 Source Code Testing

It is important as well as mandatory to make sure the source code written to develop the software is logically correct and meet the expected quality without any unexpected error and all. Also, it is required manually map and verify the logical paths and branching of the source code is actually achieving what is described in the design phase. All the developed components were manually verified by going through the source code paths. Also, all the possible paths were manually verified to see whether it is an expected scenario. Mainly two development language were used, which is CSharp and python and also application has a SQL server database. Since both the languages used to develop the project, are compiler based, so, during the compilation it can detect all the syntax issues. So, during the run time, project will have less surprises. As the first step iron, out all the syntax error to make sure the application components are compiling successfully. Then went through code to verify whether the proper error handling is in place to avoid getting run time errors.

Also, the database is verified manually going through all the field level, verified whether all the required data is storing to build the knowledge base and datatype of the fields. Couple of sample records are manually entered into the database to verify whether everything is defined as expected. Also verified the normalization of the database and keys defined as well as indexes. Peer reviews are also important and it is a must to implement an application with great quality and expected behavior, because of an outsider can see potential issues in the code which cannot be seen by the developer who developed the application. Source code is given to couple of experience developers and to review and got their feedback. Also made some necessary changes to improve the source code based on the peer review comments.

5.4 Functional Testing

Functional testing is really important to make sure the application behaves correctly as expected in a normal scenario as well as another unusual scenario. Both these scenarios application should not crash or destroy the database. Because, creating as well as maintaining the database is the key of the project and also specially as discussed, creating the database is very expensive. For functional testing, followed the actual workflow to verify everything is working as expected.

5.4.1 Testing Web Crawler

The first component which is required to make sure it is working, is the source code crawler, which is written in python and it is a command line execution, without any UI. So, ran the crawler targeting the selected programming language paths and download around 100+ samples from each language. Two things verified here with the crawler, which are, whether the sample code files saved to the correct folder and whether it is saving the complete source code sample

into the files as it is published in StackOverflow. In Order to make sure the second point decided to take 25+ files from each programming language and cross verify the code saved in the files with relevant code published in StackOverflow. Since the source URL is saved as a comment into the file, this verification is quite easy.

5.4.2 Vulnerability Exporter

To verify the functionality of the vulnerability exporter, ran the exporter against the downloaded source code files. Expected functionality is to read the vulnerability data from the saved files and insert into database. So, after import and saving vulnerability data, again selected around 25+ files from each programming language and verified the vulnerability data in those files against the data inserted to the database.

5.4.3 Testing Dashboard

To verify the dashboard, examined all the statistics showed in the dashboard against the data in the database. Examined the logic written to retrieve the data for various charts in the dashboard and verified the same data using SQL queries against the database.

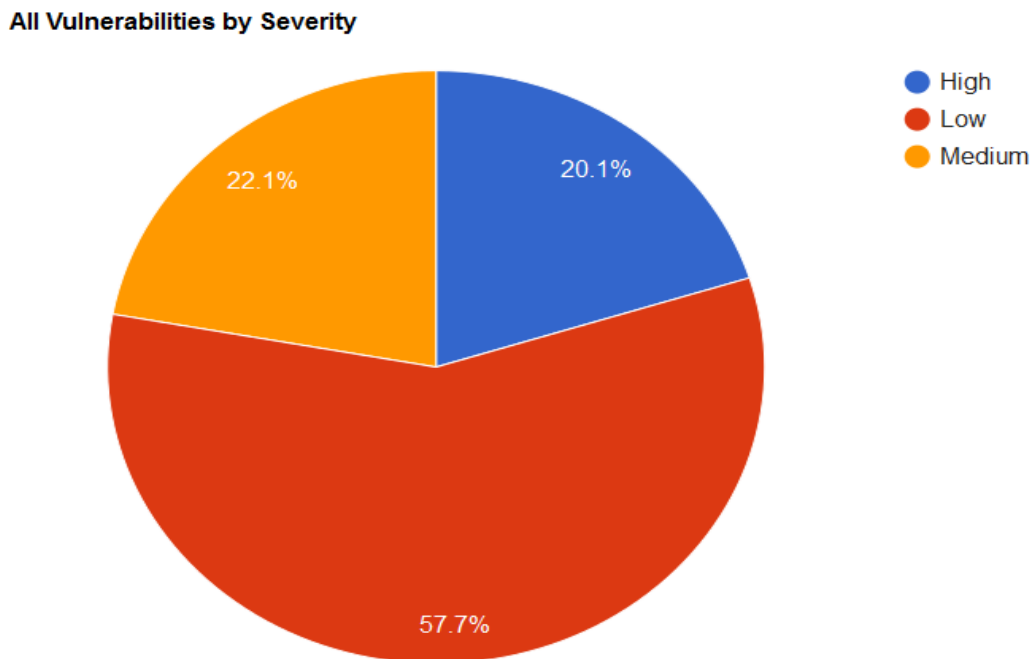


Figure 5.13 : Open vulnerabilities by severity

For example, to verify the above vulnerabilities by severity graph, below SQL queries can be used. These SQL queries are very simple and can be verifies the actual data against the plotted graphs.

```

Select COUNT(Id) TotalCnt from dbo.Issues;
Select ROUND(CAST(A.Cnt as decimal)*100/1495,1) from
(Select Severity, COUNT(Id) Cnt from dbo.Issues Group By Severity) A

```

Below is the output result of the above SQL query, which can be verified against the above chart, Open Vulnerabilities by Severity.

	TotalCnt
1	1495

	VulPercentages
1	20.100000
2	57.700000
3	22.100000

Figure 5.14 : SQL query results

5.4.4 Testing Code Analyzer Tool

Code analyzer tool is the most important piece of the project and verifying the functionality of the component is also critical. Most importantly need to make sure it is providing the correct results, since developers and others are referring the results of it to get an understanding of a particular source code sample.

5.4.4.1 Manual Verification

First step was to manually verify the correctness of the tool by testing and analyzing false positive rate and false negative rates. The strategy used is, select around 250+ sample code files from all the programming languages, analyze the vulnerabilities using the tool and manually verify whether the tool managed to identify the issues correctly or not. With the initial version of the tool, below are the received results of the false positive and negatives rates.

False Positive	70.00%
False Negative	0.00%

Table 5.1 : False Positive and Negative Percentages

The results were extremely disappointing, because the false positive rate was way over the expected level, which was around 70%. On the other hand, tool did not have false negatives and in a way, this is a great achievement, which means, tool manage to correctly detect available issues. However, the results clearly indicate that there is a significant loophole in the

vulnerability analyzing logic and definitely needs a fine-tune. After performing deep analysis with more testing, managed to successfully reduce the false positive rate also to zero percent. Visual studio unit test project supports greatly on achieving this significant improvement.

```
[TestMethod()]
public void IsCodeEqual_001()
{
    string userInput = "Console.ReadLine()";
    string vulnerableCode = "num1 = Console.ReadLine()";
    bool expected = true;
    bool actual;

    actual = UtilitySvc.IsCodeEqual(userInput, vulnerableCode);
    Assert.AreEqual(expected, actual);
}
```

Figure 5.15 : Unit test method 001

```
[TestMethod()]
public void IsCodeEqual_002()
{
    string userInput = "WriteLine";
    string vulnerableCode = "num1 = Console.ReadLine()";
    bool expected = false;
    bool actual;

    actual = UtilitySvc.IsCodeEqual(userInput, vulnerableCode);
    Assert.AreEqual(expected, actual);
}
```

Figure 5.16 : Unit test method 002

During the manual testing, also verified whether the tool manages to identify the severity as well as the confidence level of these vulnerabilities correctly.

5.4.4.2 Automated Verification

Decided to perform an automated unit test to evaluate the accuracy of the source analyzer. With automation, it is possible to evaluate larger sample of test source codes in a very short time and possible to refine the accuracy of the source analyzer tool. Below scenarios were considered during the automated testing.

- Source code samples which are used to build the knowledge-base
 - Vulnerabilities detected
 - Vulnerabilities not detected
- Source code samples which are not used to build the knowledge-base

Below is the expected outcome of these testing scenarios, assuming the tools is behaving logically as expected.

Test Scenario	False Positives	False Negatives
Known source codes with vulnerabilities	0.00%	0.00%
Known source codes without vulnerabilities	0.00%	0.00%
Unknown source codes	> 0.00%	> 0.00%

Table 5.2 : Expected outcome the Tool should provide

For the first scenario, project already has the downloaded source code samples and based on the knowledgebase developed, identify the vulnerable and non-vulnerable code sample files of each and every programming language, then extracted a sample for the testing. Below SQL queries were used to extract data from the knowledge-base for the verification and the examples shows the verification performed against the source code files of CSharp programming language.

```
Select s.FileName,d.IssueName,COUNT(d.IssueName) Cnt from dbo.Issues s,
dbo.IssueDetails d, dbo.CodeBlocks c where s.IssueId=d.Id and s.Id=c.IssueId and
s.Language='CSharp' group by s.FileName, d.IssueName order by s.FileName
```

```
Select d.IssueName,COUNT(d.IssueName) Cnt from dbo.Issues s, dbo.IssueDetails d,
dbo.CodeBlocks c where s.IssueId=d.Id and s.Id=c.IssueId and s.Language='CSharp'
group by d.IssueName
```

	IssueName	Cnt
1	CGI XSS	4
2	Hardcoded Absolute Path	17
3	Heap Inspection	13
4	Improper Exception Handling	35
5	Improper Resource Shutdown or Release	5
6	Path Traversal	11
7	Use of Cryptographically Weak PRNG	6

Figure 5.17 : CSharp vulnerabilities summery of the Knowledge-base

Below is the comparison of expected results and the actual outcome of the testing.

Issues Name	Expected Count	Actual Count	False Positives	False Negatives
CGI XSS	4	4	0	0
Hardcoded Absolute Path	17	17	0	0
Heap Inspection	13	11	0	2
Improper Exception Handling	35	34	0	1
Improper Resource Shutdown or Release	5	7	2	0
Path Traversal	11	11	0	0
Use of Cryptographically Weak PRNG	6	6	0	0
Improper Transaction Handling	0	1	1	0

Table 5.3 : Expected results vs Actual results

After getting the results, specially focused on false negatives and manually verified and found that the issue count was reduced due to the optimization logic, where when the user selects a code block with two classes and both the classes have the same issue, then the analyzer optimized the situation and show an aggregated result to the user. After manually verified the scenario, false negatives were ruled out. With this testing, project covered True Positive scenario, where is there is an issue analyzer should detect it correctly, after summarizing all the results, manage to discover that the analyzer has 6.5% of false positive rate. As the next step selected a sample of the source code files with zero vulnerabilities detected and ran the automation test against those sample. Technically the source analyzer should not detect any issues with this test. Below is the comparison of expected results and the actual outcome of the testing performed against the source code sample of CSharp language.

Issue Name	Expected Count	Actual Count	False Positives
Improper Exception Handling	0	3	3

Table 5.4 : Expected outcome vs Actual outcome - CSharp

After completing the True Negative scenario, which means analyzer should correctly reject the source codes, which does not have vulnerabilities, discovered that the source analyzer can have a 4% of false positives. To analyze the behavior of the source analyzer tool in a critical manner, decide to use source code samples, which are not considered for building the knowledge-base and evaluate the behavior of the source analyzer tool. This way it is possible to get a clear understanding of what are the enhancements, modifications required for the source analyzer. The strategy followed was as follows.

- Use the crawler to download source code from each programming language, which not downloaded before.
- User the Checkmarx tool to analyze those source code files and get the results.
- Analyze the source code samples using the Source Analyzer components.
- Compare the Checkmarx results verses Source Analyzer results.

After analyzing the source code sample using Checkmarx, the received results were as follows.

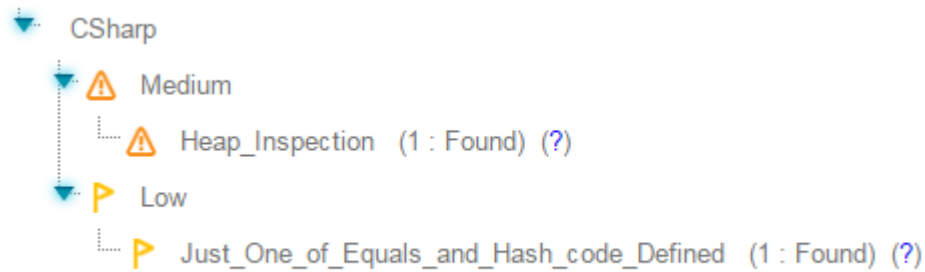


Figure 5.18 : Checkmarx results of new source samples

After analyzing the same set of source code samples with source analyzer, below is the comparison of the results.

Issue Name	Checkmarx	Source Analyzer	False Positives	False Negatives
Heap Inspection	1	1	0	0
Just One of Equals and Hash Code Defined	1	0	0	1
Improper Transaction Handling	0	1	1	0

Table 5.5 : Results comparison - Unknown source samples

After analyzing the results of test scenario of unknown source code samples, discovered that the source analyzer can have 2% of false positives and 2% of false negatives. Below is the summary of all the automated unit test performed for the project with the sample size used for the testing. According to the results, source analyzer tool managed to achieve the required level of accuracy and also the tool can provide a dependable result.

Test Scenario	False Positive %	False Negative %	Sample Size
Known source samples with vulnerabilities	6.50%	0.00%	500+
Known source samples without vulnerabilities	4.00%	0.00%	500+
Unknown source samples	2.00%	2.00%	500+

Table 5.6 : Test results summery

Test scenarios automation is implemented using CSharp.Net and it is tied to the source analyzer components of the project. Below is the source code implemented to read the known source code samples with vulnerabilities and perform the analysis.

```

System.Text.StringBuilder sb = new System.Text.StringBuilder();

public void TestKnownCodeWithVulnerabilities()
{
    sb.Clear();

    var lst = analyzeDb.Issues.ToList().FindAll(n =>
n.Language.Equals("CSharp")).GroupBy(k =>
k.FileName).Select(lt => new { fname = lt.Key });

    foreach (var s in lst)
    {
        string p = "E:\\Mis Project\\used\\" +
s.fname.ToString().Replace("/", "");
        if (File.Exists(p))
        {
            string txt = File.ReadAllText(p);
            AnalyzeSource_UnitTest(txt, s.fname.ToString());
        }
    }
    System.IO.StreamWriter file = new System.IO.StreamWriter("E:\\Mis
Project\\vulstats.txt");
    file.Write(sb.ToString());
    file.Close();
}

```

5.5 Usability Testing

Usability testing is also important because it can assess how user friendly or how much required information provided by the developed system or tool. Software systems can be developed using best or cutting-edge technologies using latest methodologies and best tools can be used to test those systems, but if the system does not meet the required usability, no one will use those software applications. Best method to assess the usability of the application is to, provide the beta version to end users and ask them to use the software for some time and then provide

the honest feedback. Web crawler and the vulnerability exporter are not developed for end users and there is no reason to perform usability testing against those two components. Dashboard and Code analyzing tool are the two components developed to end user and focus on performing usability testing only for those two components.

Following above described strategy, testing version of the dashboard and code analyzer components given to set of developers, quality engineers and technical specialist and ask them to use these components for some time. Google form is used to collect the feedback and feedback collected anonymously, because the target is to get genuine feedback from those set of professionals. Also incorporated some of the important comments into the components to make them more usable. Below is the feedback form created to set of end users to give their feedback about the Dashboard and the Code analyzer.

The image shows a Google Form titled "Assessment of Code Analyzer & Dashboard". The form includes a title, a request for honest feedback, a list of required questions, a 5-point Likert scale for accuracy, and a text box for improvement ideas. A blue "SUBMIT" button is at the bottom, and a warning about passwords is at the very bottom.

Assessment of Code Analyzer & Dashboard

Please take some of your valuable time to fill the form. Please make sure to provide honest feedback

* Required

What do you think about analyzing source sample published in open forums *

Your answer

What are the issues/benefits of Dashboard *

Your answer

What are the issues/benefits with code analyzer *

Your answer

What do you think about the Accuracy of the code analyzer *

	1	2	3	4	5	
Poor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very High

Ideas to improve *

Your answer

SUBMIT

Never submit passwords through Google Forms.

Figure 5.19 : Usability feedback form

Even though, it sounds like a great idea and a definite way of improving the project, it is also really hard to collect the feedback. This is even harder, when the targeted audience is

professionals and totally engaged with day to day work. Had to spend huge effort on collecting feedback from developers, architects and managers. Planned to collect around 50+ feedback, but after spending weeks following up those professionals, managed to collect 20+ feedback and its decent enough for the analysis. Analysis mainly focus on getting feedback on user friendliness and point to improve the application and also it was focused on getting an accuracy measure of the source analyzer tool. Below are the results received for the accuracy of the tool.

What do you think about the Accuracy of the code analyzer (22 responses)

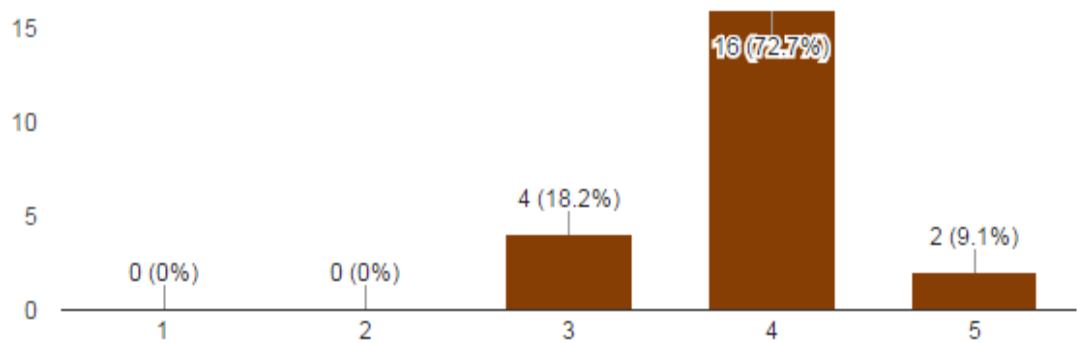


Figure 5.20 : Expert feedback on accuracy of source analyzer

Chapter 6 : Conclusions and Future Work

6.1 Introduction

Static analysis of the source code is an important and essential activity to make sure the developed is secured and rugged to stand against malicious attacks. Vulnerabilities that can be discovered during the static analysis will help developers to iron out them during very early stages of the development life cycle. Also, static code analysis is mandatory for an organization to implement secure software development life cycle, which has the security built into the development life cycle. Even though, many static analysis tools are available, including very expensive commercial tools, technically the developers or the development team does not analyze each and every code block they write or even before they write. Simply this is because analysis takes time and the tool is costly, when it comes to commercial tools. Instead of that they perform weekly or biweekly scans to discover the vulnerabilities. Then the discovered issues will be added to the detect backlog and will be addressed during same sprint or next sprint. This is again a problem because developers have to spend time on fixing issues, where they could have addressed them during the initial development, if they got to know about those issues.

When it comes to developers referring source code samples from open forums like StackOverflow, situation is getting worst, because no one assess these source code samples for security standards and vulnerabilities. Considering all these scenarios, it is required to have a method to verify these open forum source codes quickly before those are used into the production source code of the organization's product. This chapter will summarize the project work and discuss about the findings, problems, challenges, learning and limitations. Also, this chapter will discuss about the possible future work of the project.

6.2 Summary

The project aims to address the issue of, developers using source codes samples, published in open forums, without assessing security vulnerabilities of those source codes. The plan of the project is to develop a tool which is easy, convenient, efficient and most importantly user friendly, for developers, which is capable of identifying and visualizing the potential vulnerabilities of the source code samples published in open forums. Solution also aim to provide some insight to the developers, architects and managers about the vulnerabilities exists with the source samples with some other useful statistics like what are the most common vulnerabilities, which language has the most number of vulnerabilities.

After analyzing the situation, project decided to select one open forum, grab source code samples published in the selected open forum, under selected programming languages and then analyze the vulnerabilities using a commercial static analysis tool. After that, import the vulnerability results from the static analysis tool to create a knowledge base and then develop

a tool, which can be used by the developers to assess the vulnerabilities in the open forum by referring the knowledge base created.

First step was to select an open forum and after doing a study, decided to select StackOverflow as the open forum. Went through all the published user statistics data by StackOverflow, before selecting it as the open forum. Due to the project time limitation, had to decide what are the programming languages and the number of source code samples from each programming language, which are going to select for the vulnerability analysis. Totally five programming language were selected purely based on the popularity among the development community which includes, CSharp .Net, Java, PHP, Python and JavaScript. Decided to analyze at least 5000 source code samples from each programming language. To grab the source codes from the StackOverflow, implemented a web crawler using python programming language. Also, python based famous web crawling framework names scrapy is used to developed the web crawler.

After performing a study on commercially available static code analysis tools, to select an appropriate tool for the project. According to the analysis, Checkmarx was the most suitable tool for the project. Since the tool is highly expensive, manage to find a sponsorship to use the tool to perform the analysis. Implemented a software component to read the vulnerability results from Checkmarx report and save the data to the knowledge base.

Most convenient and user-friendly method of implementing the developer tool to analyze the source codes is to develop a browser plugin. Because, developer need to use the browser to visit the open forum to see the source codes. Project decided to stick to one particular browser and develop a plugin for that browser only. After doing a study and see which browser is the famous among all, selected the browser as google chrome for the project, since it is famous among the developer community. Also, to help with the vulnerability statistics, decided to develop a dashboard with various charts related to vulnerability data. For implementation of vulnerability imported, dashboard and source code analyzer for developer are developed using CSharp .Net and for the knowledge base, Microsoft SQL server database is used.

6.3 Problems Faced

Couple of issues were faced during the implementation of the project and had perform workarounds and sometimes some components got delayed due to these problems. And some of the issues could not resolve technically, but did not harm the final output of the project.

6.3.1 Crawling StackOverflow

The issue arises when crawling source codes from StackOverflow. After crawling certain number of source code samples, StackOverflow detect large number of source code request from the public ip address and block the ip for some time, like 5 to 10 minutes. Had to work with this issue because there is no way to get rid of the issue. The issue slow down the source code crawling speed significantly, but manage to achieve the required number of source code samples by putting an extra effort.

6.3.2 Commercial Tool

Project needed a commercial static analysis tool and purchasing a tool is impossible due to the high cost and find a sponsorship for the tool is really challenging. It took considerable effort and time to make the tool available. Since the tool is already engaged with the day to day scans of the organization, it was really challenging to allocate time for the project related source code scans and also it was not possible to upload huge number of sample codes at once, because it will make the tool stressed. This was the greatest challenge faced during the project and had to spend lots of time to manage time and upload small chunks of source codes to analyze them and get the results from the tool.

6.3.3 False Positives

This was a problem as well as a huge challenge. Like discussed above any tool, irrespective of whether the tool is a commercial one or not, can provide false alarms. The only way of getting rid of detected false positives is to manually verify and eliminate them. Had to spend considerable amount of time and effort, going through all the discovered vulnerabilities and verify whether those are false positives or not. This process is a must to perform to have an accurate and quality output.

6.3.4 Browser Plugin

It was easy to implement a browser plugin with chrome browser, but could not find an easy way to reading the entire source code block, when the user right clicks on it. Technically could not solve this issue and had to go with select source code block and the right click, so the browser plugin was able to read the selected text. This is not very convenient and user friendly for the developer. But in a way, it was an advantage, because there may be cases where the developer needs only a portion of the source code block to asses. Not so convincing but finally it was an advantage for the project also.

6.3.5 Usability Testing

Usability testing was not a problem, but it was bit of a challenge, since project needed feedback from professional developer, architects and managers. Usually these professionals are extremely busy and it was hard to buy some of their time for evaluation of the project. Also had to spent considerable time to demonstrate the tool for the developers and managers. Another problem was, these professionals were bit lazy to fill the feedback form, so had push them little bit on filling it, and also had to get verbal input and proceed as well.

6.4 Limitations

Even though the project aims to provide a fair solution for a specific problem, like any other project, it has limitations. Some limitations arise due to the time limitation of the project and some limitations are technical constraints. Since the code analyzer is the most used and the primary component of the project, limitation of the code analyzer component will be noticeable. Below are the major limitations of the source analysis component.

- Tool can detect or predict the issues only using the knowledge base which is previously created by analyzing the source samples.
- Limited only for the source codes published in StackOverflow.
- Tool is doing a text based matching to verify with the knowledge base.
- Only support for CSharp, Java, PHP, Python and JavaScript programming languages.
- Only support for google chrome browser.
- Tool is only indicating the potential vulnerabilities to the user, but it cannot prevent the user from using the vulnerable code. So, the tool is just a helper only.
- Building knowledge base is not fully automated.

6.5 Extensions and Further Work

After identifying the limitations of the project, it is required to plan for improvements and enhancements for the project to make it better and serve the users better. Mainly focused on the limitations of the current implementation and also the original problem which needs to be solved, during coming up with future work for the project.

6.5.1 Fully Automate

Current implementation of crawling the source codes from the open forum and get it analyzed using the commercial tool and then build the knowledge base is not fully automated. With the current implementation, below tasks should be manually performed to successfully build the knowledge base.

- Execute the web crawler to grab the source code samples.
- Upload the source samples to the commercial tool.
- Download the report of potential vulnerabilities from the commercial tool.
- Execute the vulnerability importer to import data to the database.

Plan is to fully automate this process and also handle the StackOverflow restriction by adding proper time delay when the ip is restricted. Then project can keep on building the knowledge base automatically, without much of a human interaction.

6.5.2 Expand the Knowledge Base

With the current implementation project is limited to five programming languages, CSharp, Java, PHP, Python and JavaScript and only limited for source codes published in StackOverflow. Also, current implementation considered 5000+ samples from each programming language. Another point is, source samples are analyzed using only one commercial tool. Need to expand this to other programming languages also and increasing the number of samples considered also. Consider the source code samples published in other available open forums also required. Most importantly use other commercial tools to analyze the code samples and integrated the vulnerability results can make the knowledge base sophisticated and more accurate.

6.5.3 Enhancements

Application needed to be fine-tuned to make it more efficient, so that the application can handle user requests fast and accurate. Specially need to optimize the vulnerability analyzing code to make it efficient. Also, fine tune the UI also important to make the users are comfortable with the tool, it is user friendly and also to make sure the tool provides necessary information with its feedback to the user. Optimizing the database also important to make sure the application can achieve the required level of efficiency. Using a text matching with the vulnerability identification logic is not so effective and also it is slowing down the process. Also, the tool cannot identify potential vulnerabilities effectively because some code lines are logically same but text comparison is different. Required to com-up with symbolic representation of the source texts and need to perform the validation based on these symbols. With that tool can improve both efficiency and accuracy.

Currently the implementation is done using CSharp .Net and the database is implemented using Microsoft SQL server. There is no issue with these technologies, but better to move with a python framework like Django and MySQL which is more robust, scalable and with high maintainability. With MySQL, the application will not be having any license issues as well. Testing is performed by limited number of known developers and managers, which is not enough for an open tool, targeted a large audience. Required to host the application in a publicly accessible production environment and make it available for the development community and invite them to perform testing and send feedback for fine tune and improve the tool. Also need to make the source code available for the community using a public repository like GitHub and get support from the development community to fine tune the source code.

6.5.4 Future Work

Most important assets or the output of the project work is the knowledge base, which includes all the vulnerable source codes and related data. There is a great opportunity to perform a data mining activity against the knowledge base and identify patterns hidden within the knowledge base. This can greatly help to the developer and related community. Also, there is an opportunity where the source analysis tool can be enhanced to use the discovered pattern to identify potential vulnerabilities in an unknown source code sample.

Another future work will be to assess the impact created by these source code samples published in open forums, on open source products. To assess this, needs to verify whether the vulnerable source codes are existing in know open source products by assessing the GitHub source code of those known open source products. By analyzing the impact created by source code sample published in open forums, it is possible to alert the community so the community itself will be more careful when publishing source code sample in future.

6.6 Critical Appraisal of the System

Goal of the project was to create a tool which can help developers to analyze the source codes published in open-forum, for security vulnerabilities and also give an indication of the vulnerability statistics of those source code samples to the developers as well as managers. Implementation of the project manages to successfully meet all the expected requirements with required quality. So, the project managed to achieve its goal. The project is highly designed and not tied to a particular technology or a framework. Also, the design is capable of adopting future enhancements, changes as well as the required expansion to the project.

During the implementation, followed the recommended coding guidelines and best practices to improve the maintainability. Required and useful comments were added into the source code and the source code is well tested and reviewed by couple of senior developers. Final implementation is very easy and convenient for developers to use, since it is a browser plugin. During the browsing of open forum source samples, developers can easily use the browser extension to analyze the vulnerabilities of a particular code block in a matter of seconds. Project implementation is tested and evaluated by developers and managers and they are very much satisfied with the implementation and the idea behind the project. Also, the evaluators are confident that the idea and the tools will add great value to the community as well as the commercial organizations.

6.7 Final Conclusion

The author is confident that the project idea is very much valid and the implementation of the project can greatly help developers to write more secure code and ultimately make the final product more secured and rugged. Implementation of the project can be used as a vulnerability assessment tool as well as a learning tool for the developers. Also using the project implementation an organization can get to know about the vulnerabilities exists with each programming language and they can define guideline and best practices to avoid those vulnerabilities. Also, the tool can help to perform the peer review effectively and also it can help to come up with an effective testing strategy. Since the tool helps to assess and understand the vulnerabilities of a particular source code block, which is published in an open forum, before it is used or implemented into the organization's production code, author believes that the project ultimately helps to have a successful secure software development life cycle within an organization.

Appendix A : Development and Testing Environment

A.1 Hardware Requirements

To perform the development of the application, personal computer is used with below configurations.

- Intel Core i7 processor with 2.60 GHz.
- 8.00 GB internal RAM.
- 500 GB SSD hard disk.
- Microsoft windows 7, 64-bit operating system.

A.2 Software Requirements

For the development, virtual environment is used to make it more convenient, since the virtual environment can be managed easily. To develop the main components CSharp .Net is used. Below mentioned software were used for the project implementation.

- Microsoft Visual Studio 2010 IDE.
- CSharp .Net with MVC 3 framework.
- Microsoft SQL Server 2008.
- Python 2.7.
- Visual Studio Code IDE.
- Google chrome browser.
- Oracle VM VirtualBox - To create and run the VirtualBox.

Appendix B : General Information

B.1 Execution of Web-Crawler

Web crawler is a python based script and can be executed as a normal python script. Expected behavior is, when the crawler ran pointing to a particular URL in StackOverflow, it should read all the published source code samples, which are user's answers to the published questions, and save those samples into a local folder with the correct extension of the related programming language.

In the below example, crawler is pointed to JavaScript related questions, so the crawler will grab the published JavaScript code samples.

(<http://stackoverflow.com/questions/tagged/javascript?page=1&sort=newest&pagesize=50>)

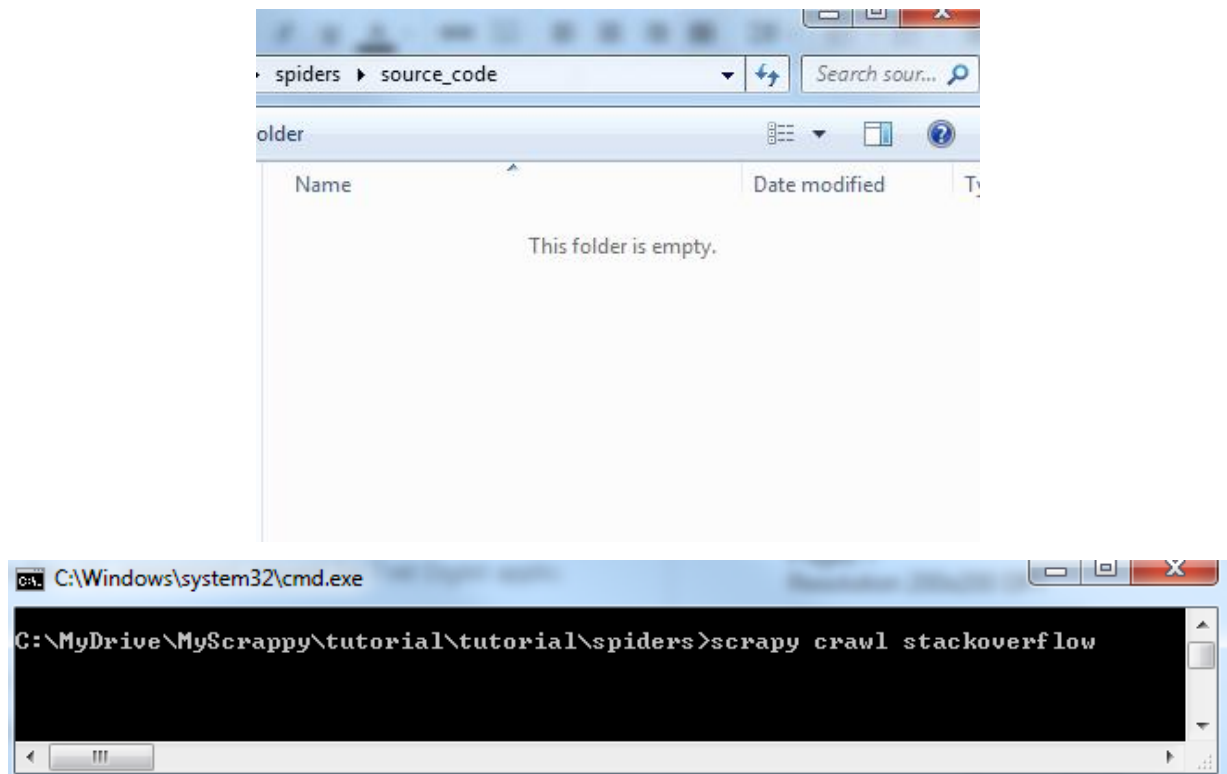


Figure B.1 : Above to run the web crawler

```
C:\Windows\system32\cmd.exe
'downloader/request_method_count/GET': 53,
'downloader/response_bytes': 1035660,
'downloader/response_count': 53,
'downloader/response_status_count/200': 52,
'downloader/response_status_count/301': 1,
'finish_reason': 'finished',
'finish_time': datetime.datetime(2017, 2, 28, 7, 13, 21, 488000),
'log_count/DEBUG': 54,
'log_count/INFO': 7,
'request_depth_max': 1,
'response_received_count': 52,
'scheduler/dequeued': 52,
'scheduler/dequeued/memory': 52,
'scheduler/enqueued': 52,
'scheduler/enqueued/memory': 52,
'start_time': datetime.datetime(2017, 2, 28, 7, 13, 16, 497000)}
2017-02-28 12:43:21 [scrapy.core.engine] INFO: Spider closed (<finished>)

C:\MyDrive\MyScrapy\tutorial\tutorial\spiders>
```

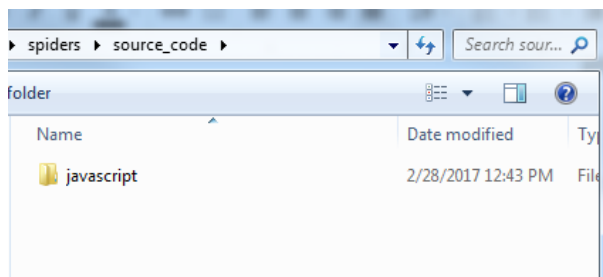


Figure B.2 : After running the web crawler

B.2 Browser Usage

To make it easy and convenient, project decided to implement a browser plugin for developers. After analyzing the easiness to develop a plugin as well as how famous is the browser among the community, decided to select the browser as google chrome and implement a chrome extension for the developers. Below is some browser usage information referred for the project, to select a browser.

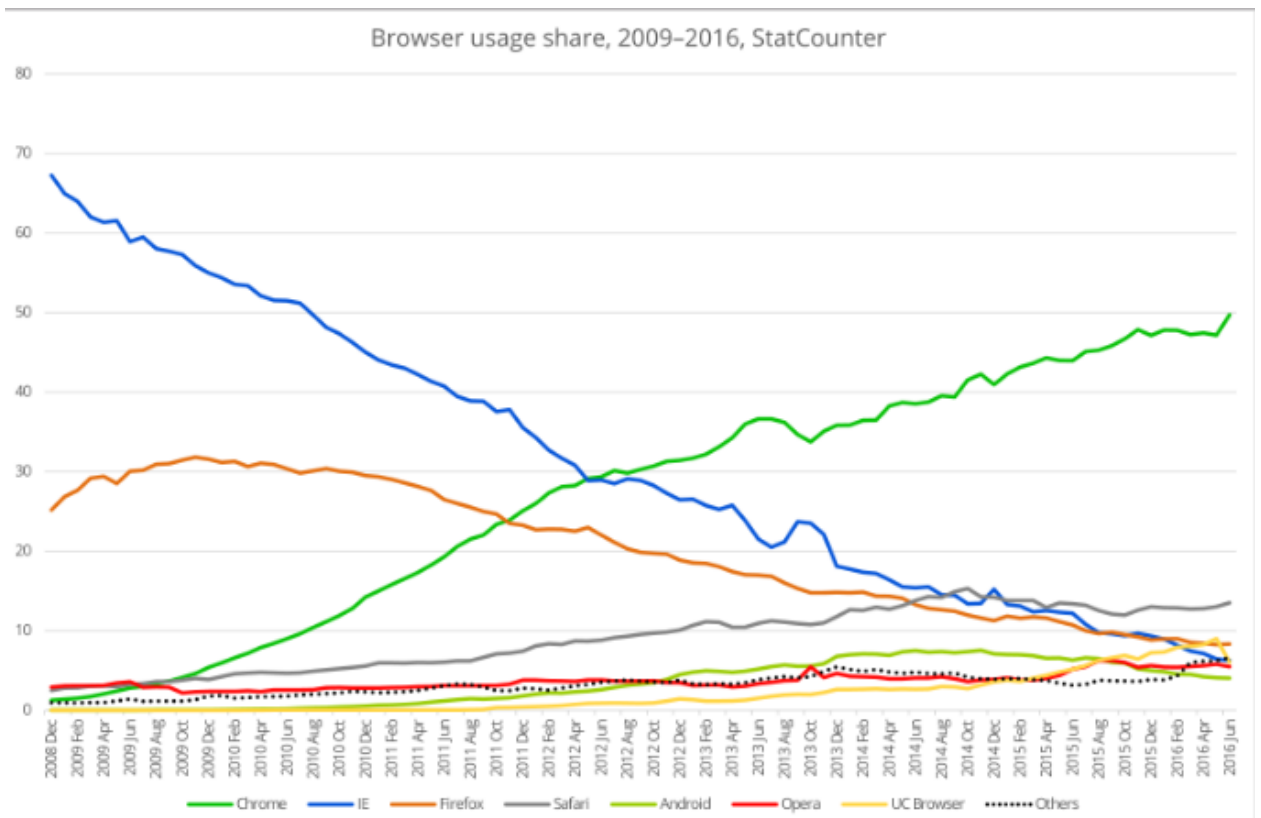


Figure B.3 : Browser Usage 2009 - 2016

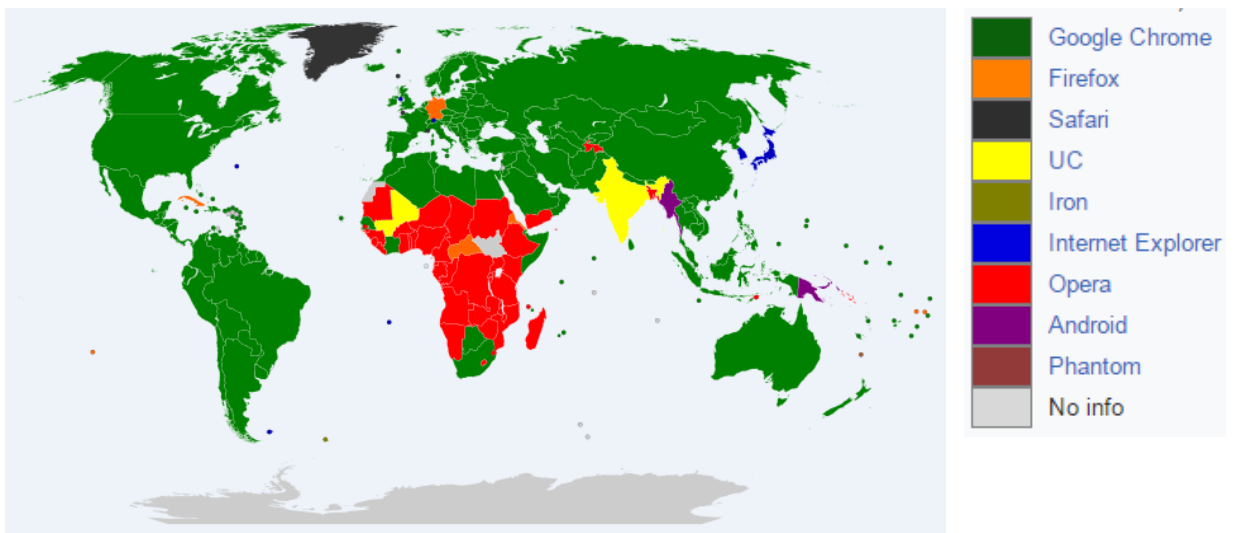


Figure B.4 : Browser market map - 2015

B.3 Checkmarx Manual Verification

Checkmarx is the commercial tool used for the project to perform static analysis. After Checkmarx is completed with the static analysis, it is a must to perform a false positive analysis to clean up the results and keep only the actual issues. Checkmarx allows to perform a manual verification against the discovered vulnerabilities and mark and eliminate false positives issues.

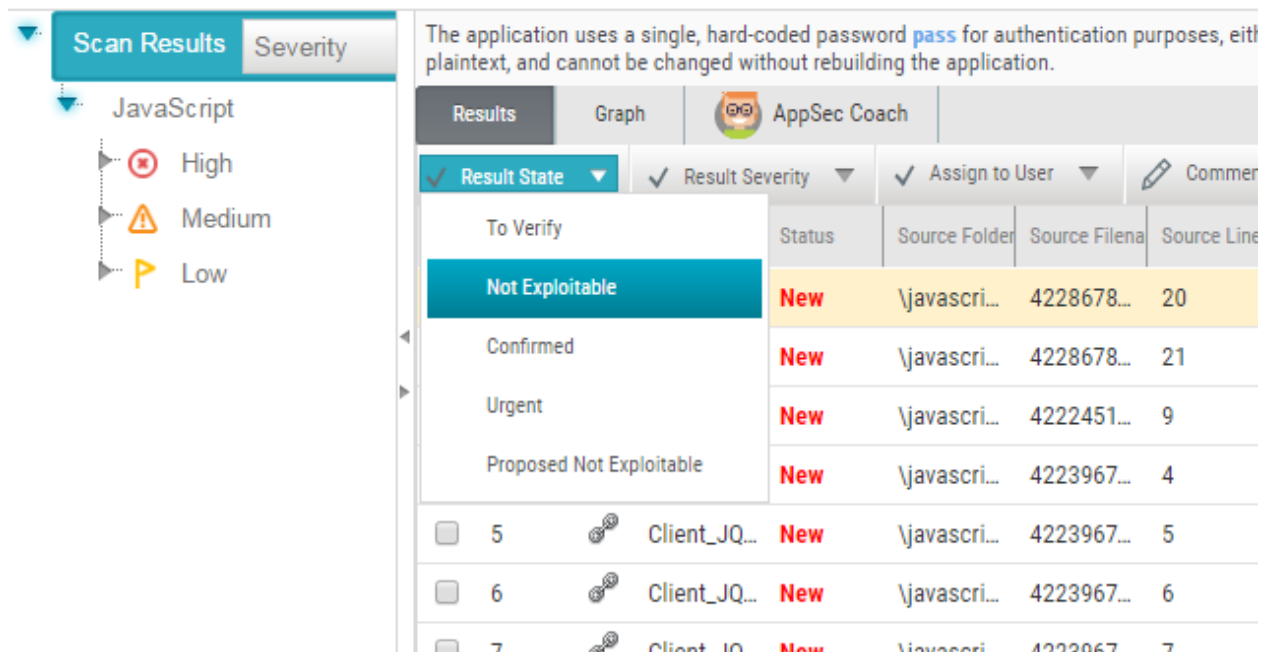


Figure B.5 : Mark vulnerabilities as false positive

Also, there can be situation where the Checkmarx marked a vulnerability with a particular severity, but after the manual verification, it seems the severity should be something else. If the detected severity of a vulnerability is not the correct figure, Checkmarx allows to set the correct severity for the vulnerability. Below screenshot show the menu option to set the severity, manually.

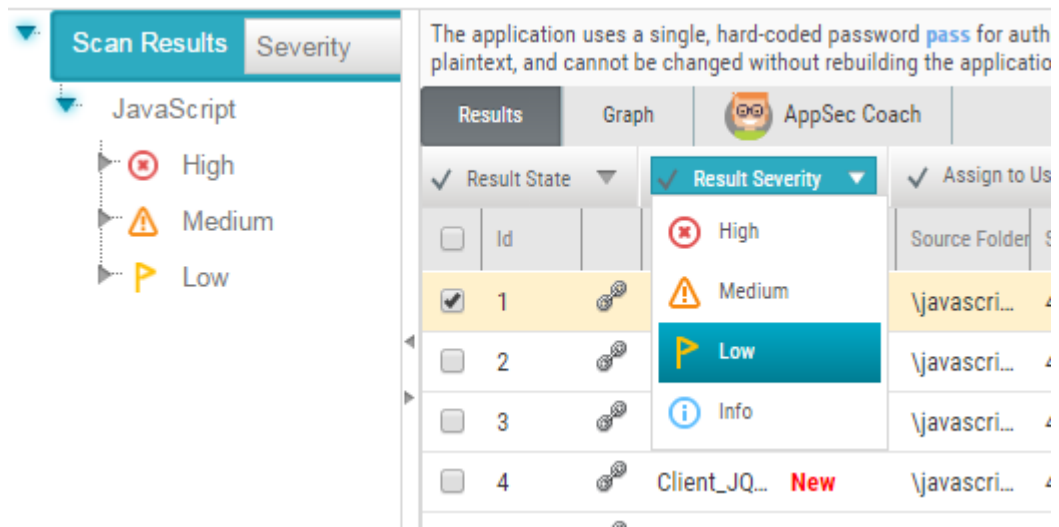


Figure B.6 : Manually set the severity of a vulnerability

B.4 Checkmarx Reports

Checkmarx allows and support several report formats to import the vulnerabilities, including PDF and csv. For the project XML report format is used, since it is really easy to process and import issues from a XML report to the database.

```
<Query id="463" categories="PCI DSS v3.1;PCI DSS (3.1) - 6.5.7 - Cross-site scripting (XSS),OWASP Top 10
<Query id="3772" cweId="244" name="Heap Inspection" group="CSharp Medium Threat" Severity="Medium" Langua
<Query id="443" categories="PCI DSS v3.1;PCI DSS (3.1) - 6.5.5 - Improper error handling" cweId="248" nam
  <Result NodeId="276930002" FileName="/csharp/39749136-2.cs" Status="New" Line="15" Column="40" FalsePos
    <Path ResultId="27693" PathId="2" SimilarityId="-2041245357">
      <PathNode>
        <FileName>/csharp/39749136-2.cs</FileName>
        <Line>15</Line>
        <Column>40</Column>
        <NodeId>1</NodeId>
        <Name>ReadAllText</Name>
        <Type></Type>
        <Length>11</Length>
        <Snippet>
          <Line>
            <Number>15</Number>
            <Code>          string dataFromRead = File.ReadAllText(filePath);</Code>
          </Line>
```

Figure B.7 : Checkmarx imported XML report

B.5 Chrome Extension

Chrome developer center giving a great help and detailed documentation for developers and chrome extension can be easily created referring the documentation. It has certain set of files and specific format expected from the developer. Below is the set of files required by chrome.







 background.js	JScript Script File
 glass_icon.png	PNG image
 info.js	JScript Script File
 manifest.json	JSON File
 results.html	Firefox HTML Document
 styles.css	Cascading Style Sheet Document

Figure B.8 : Chrome extension files

Manifest file is the one which defines the chrome extension with the general information, like extension name, description, permissions and icons of the plugin. Below is the manifest.json file created for the project.

```
{  
  "name" : "SourceAnalyzer",  
  "version" : "1.0.1",  
  "description" : "Retrieve Potential Vulnerabilities of the Selected Code Sample",  
  "background" : { "scripts": ["background.js"] },  
  "permissions" : [  
    "contextMenus",  
    "tabs",  
    "http://**/*",  
    "https://**/*"  
  ],  
  "minimum_chrome_version" : "6.0.0.0",  
  "icons" : {  
    "16" : "glass_icon.png",  
    "48" : "glass_icon.png",  
    "128" : "glass_icon.png"  
  },  
  "manifest_version": 2  
}
```

Figure B.9 : Source Analyzer Chrome extension

B.6 CodePlex FuzzyString

FuzzyString is an open source project and a library, developed using CSharp .Net to verify the equality of two strings approximately. Library includes well known approximation algorithms and below is the full list of algorithms supported.

- » Hamming Distance
- » Jaccard Distance
- » Jaro Distance
- » Jaro-Winkler Distance
- » Levenshtein Distance
- » Longest Common Subsequence
- » Longest Common Substring
- » Overlap Coefficient
- » Ratcliff-Obershelp Similarity
- » Sorensen-Dice Distance
- » Tanimoto Coefficient

Figure B.10 : FuzzyString algorithms

Below is an example of how to compare two strings approximately using the library and it returns Boolean value indicating whether the two strings are matched or not.

```
string source = "kevin";
string target = "kevyn";

List<FuzzyStringComparisonOptions> options = new List<FuzzyStringComparisonOptions>();

// Choose which algorithms should weigh in for the comparison
options.Add(FuzzyStringComparisonOptions.UseOverlapCoefficient);
options.Add(FuzzyStringComparisonOptions.UseLongestCommonSubsequence);
options.Add(FuzzyStringComparisonOptions.UseLongestCommonSubstring);

// Choose the relative strength of the comparison - is it almost exactly equal? or is it
just close?
FuzzyStringComparisonTolerance tolerance = FuzzyStringTolerance.Strong;

// Get a boolean determination of approximate equality
bool result = source.ApproximatelyEquals(target, options, tolerance);
```

Figure B.11 : FuzzyString compare two strings

B.7 Google Trends

CodeProject also an open forum which is very famous among the development community. It is important to analyze and compare the CodeProject with the StackOverflow to see the current trend. Below are some trends provided by Google.

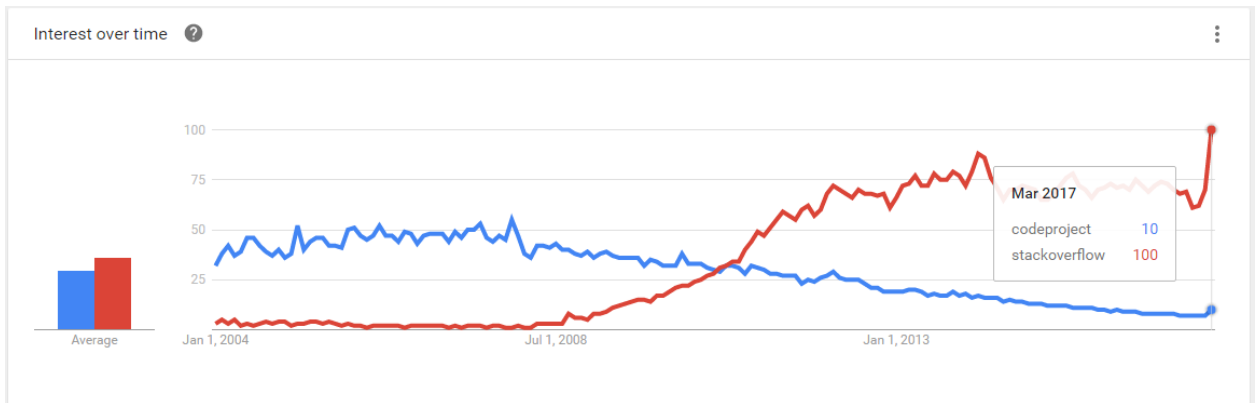


Figure B.12 : Interest over time

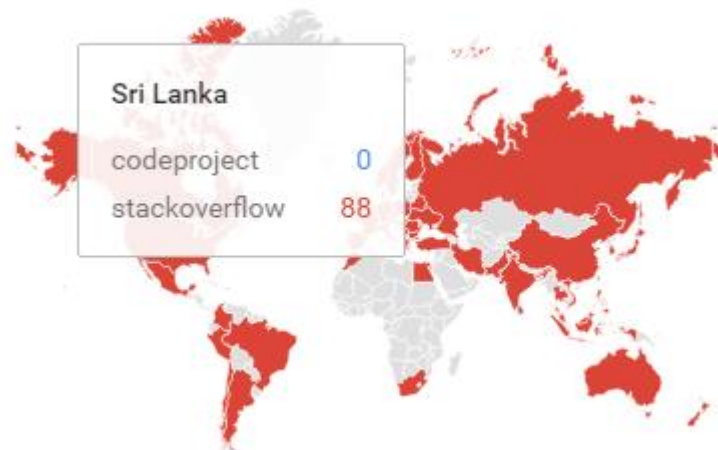


Figure B.13 : Interest by region

Appendix C : Project Source Code

C.1 Project Structures

Application has two projects developed with CSharp .Net and other one is the chrome plugin. To develop both CSharp .Net projects, Visual Studio is used as the integrated development environment. Below are the project structures of these two projects.

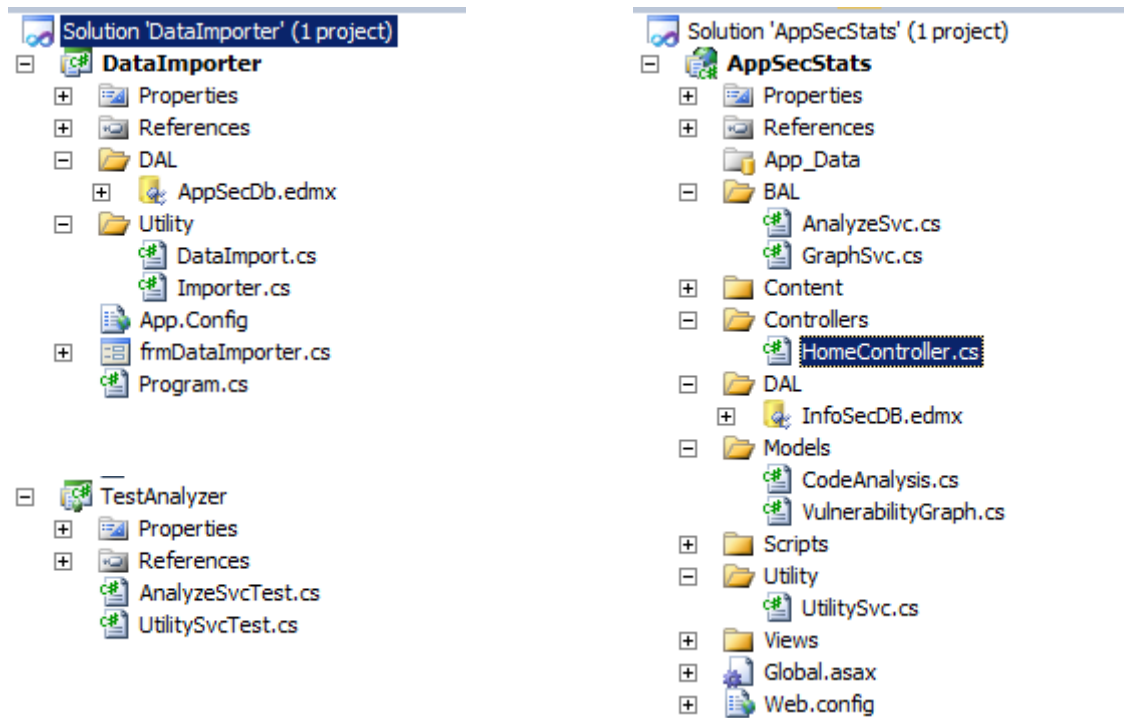


Figure C.1 : Project structures

C.2 Helpful Comments

Appropriate comments were used within the source code, which can describe the source and its functionality. Comments can improve the maintainability and greatly help for the future developments of the project.

```

/// <summary>
/// enum used to define the vulnerability confidence levels
/// </summary>
public enum ConfidenceLevels
{
    Certain = 0, //Issues is Certainly exists
    Firm = 1, //High confidence, but manual verification needed
    Tentative = 2 //Not sure
}

/// <summary>
/// Import vulnerability data and save it into the database.
/// </summary>
/// <param name="filePath">Path of the vulnerability file (.xml report)</param>
/// <returns></returns>
public static bool ImportData(string filePath)
{
    try

        /// <summary>
        /// Holds the data related to a particular vulnerability
        /// </summary>
        public class Vulnerability
        {
            //Actual issue of the vulnerability
            public Issue CodeIssue { get; set; }

            //Likelihood of the vulnerability
            public decimal Likelihood { set; get; }

            //Confidence level os the vulnerability
            public string Confidence
            {
                get

```

Figure C.2 : Project comments

Appendix D : Testing the Application

During the automated testing, project selected 500+ source samples to verify the accuracy of the source code analyzer. Results were collected aggregated and compared with the expected numbers one by one manually. With this project managed to discover the deviations and identify false positives, negatives to fine-tune the application. Below is the data table used to compare the CSharp results with data.

File Name	Vulnerability	Expected Count	Actual Count
/39711899-1.cs	CGI XSS	4	4
/1716447-3.cs	Hardcoded Absolute Path	1	1
/41755542-3.cs	Hardcoded Absolute Path	1	1
/41791595-2.cs	Hardcoded Absolute Path	2	2
/41804185-6.cs	Hardcoded Absolute Path	1	1
/41815058-1.cs	Hardcoded Absolute Path	2	2
/41816733-6.cs	Hardcoded Absolute Path	1	1
/41819241-1.cs	Hardcoded Absolute Path	1	1
/41822147-3.cs	Hardcoded Absolute Path	1	1
/41825522-9.cs	Hardcoded Absolute Path	1	1
/41834241-3.cs	Hardcoded Absolute Path	2	2
/41853886-6.cs	Hardcoded Absolute Path	1	1
/41868930-1.cs	Hardcoded Absolute Path	1	1
/41870998-5.cs	Hardcoded Absolute Path	1	1
/462270-4.cs	Hardcoded Absolute Path	1	1
/39730960-1.cs	Heap Inspection	1	1
/41764683-1.cs	Heap Inspection	1	1
/41764683-5.cs	Heap Inspection	1	1
/41783872-1.cs	Heap Inspection	3	3
/41793082-2.cs	Heap Inspection	2	2
/41846572-4.cs	Heap Inspection	1	1
/41883072-1.cs	Heap Inspection	2	1
/41907955-4.cs	Heap Inspection	2	1
/18757097-10.cs	Improper Exception Handling	1	1
/2876616-7.cs	Improper Exception Handling	1	1
/39668236-1.cs	Improper Exception Handling	1	1
/39749136-2.cs	Improper Exception Handling	2	2
/41755542-3.cs	Improper Exception Handling	1	1
/41769399-3.cs	Improper Exception Handling	3	3

/41788661-3.cs	Improper Exception Handling	1	1
/41791595-2.cs	Improper Exception Handling	3	3
/41793534-3.cs	Improper Exception Handling	1	1
/41803707-1.cs	Improper Exception Handling	2	1
/41813610-1.cs	Improper Exception Handling	1	1
/41816147-1.cs	Improper Exception Handling	2	2
/41834241-2.cs	Improper Exception Handling	2	2
/41834241-3.cs	Improper Exception Handling	2	2
/41840827-2.cs	Improper Exception Handling	1	1
/41842148-2.cs	Improper Exception Handling	1	1
/41853886-6.cs	Improper Exception Handling	1	1
/41854338-5.cs	Improper Exception Handling	2	2
/41854338-6.cs	Improper Exception Handling	2	2
/41870998-5.cs	Improper Exception Handling	1	1
/41894232-1.cs	Improper Exception Handling	1	1
/41907955-4.cs	Improper Exception Handling	2	2
/41907959-1.cs	Improper Exception Handling	1	1
/41755542-3.cs	Improper Resource Shutdown or Release	1	1
/41853886-6.cs	Improper Resource Shutdown or Release	1	1
/41870998-5.cs	Improper Resource Shutdown or Release	1	1
/41890156-2.cs	Improper Resource Shutdown or Release	1	1
/41894232-1.cs	Improper Resource Shutdown or Release	1	1
/41788661-1.cs	Path Traversal	5	5
/41788661-2.cs	Path Traversal	3	3
/41788661-3.cs	Path Traversal	3	3
/41786555-5.cs	Use of Cryptographically Weak PRNG	1	1
/41824277-5.cs	Use of Cryptographically Weak PRNG	2	2
/41910525-1.cs	Use of Cryptographically Weak PRNG	1	1
/767999-5.cs	Use of Cryptographically Weak PRNG	2	2

Table D.1 : Detailed results comparison

It is important to check the usability of the application to measure the user-friendliness as well as the accuracy. Decided to distribute the application among professionals to collect their feedback. Below is the row feedback data received from those experts regarding the ideas to improve the application.

Ideas to improve
needs to fine tune
need to analyze more samples
improve please
fine tune, customization charts
should be able to identify issues based on the source
fine tune
consider latest source codes
focus on something important. may be super hero
keep going. Need more surprises
optimize and make it faster
fine tune
automate analyze part also
code project is better option
continue
provide some details of the issues
need to fine tune more
need to enhance it by analyzing more code
need more charts
share the project so all can contribute
allow to enter issues manually to the database
make it open source and fine-tune it
implement using python

Table D.2 : Ideas to Improve

Appendix E : Dashboard Options

Dashboard included various graphs, charts to represent the vulnerabilities discovered during the source code assessment. This component can greatly help for developers and quality engineers as a learning tool. Also engineering teams can use these data to come up with a solid test strategy to perform required tests to discover these vulnerabilities. Another important point is, the dashboard can be used by architects and senior developers to perform peer reviews, effectively. Below is the available charts of the dashboard.

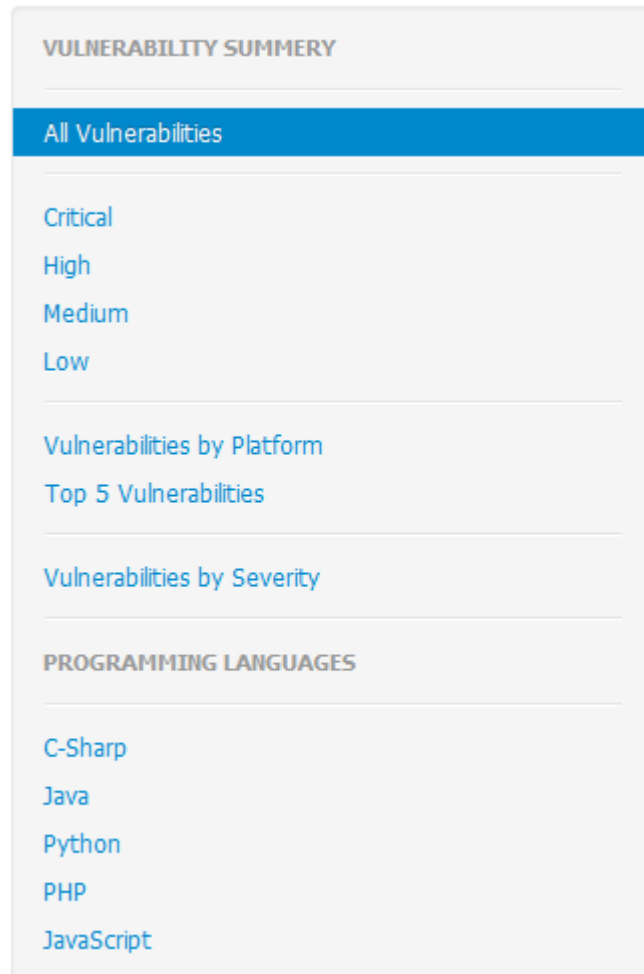


Figure E.1 : Chart options of the Dashboard

References

- [1]. Elizabeth Fong, Vadim Okun, "Web Application Scanners: Definitions and Functions," in Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8970
- [2]. Symantec (Apr, 2006) "Five common Web application vulnerabilities," [Online]. Available:<<http://www.symantec.com/connect/articles/five-common-web-application-vulnerabilities>> [Accessed on 17 May 2016]
- [3]. Guttorm Sindre, in Dept of Computer and Info. Sci. Norwegian Univ. of Sci. and Tech., Andreas L. Opdahl, in Dept of Information Science, University of Bergen, Norway, "Capturing Security Requirements through Misuse Cases"
- [4]. OWASP "2013 Top 10 List", [Online]. Available:<https://www.owasp.org/index.php/Top_10_2013-Top_10> [Accessed on 18 June 2016]
- [5]. Microsoft (Feb 2013) "Microsoft Security Development Lifecycle"
- [6]. Cigital "What Is the Secure Software Development Life Cycle", [Online]. Available:<<https://www.cigital.com/blog/what-is-the-secure-software-development-lifecycle/>> [Accessed on 17 June 2016]
- [7]. Cigital "Seven Touchpoints for Software Security", [Online]. Available:<<http://www.swsec.com/resources/touchpoints/>> [Accessed on 5 July 2016]
- [8]. Paul E. Black, Michael Kass, Michael Koo, Elizabeth Fong, "Source Code Security Analysis Tool Functional Specification Version 1.1" in NIST Special Publication 500-268 v1.1
- [9]. RTI "The Economic Impacts of Inadequate Infrastructure for Software Testing" in NIST, RTI Project Number 7007.011
- [10]. OWASP (Jul 2016) "Static Code Analysis", [Online]. Available:<https://www.owasp.org/index.php/Static_Code_Analysis> [Accessed on 28 July 2016]
- [11]. Brian Chess, "Metrics That Matter:Quantifying Software Security Risk" in Fortify Software,2300 Geng Road, Suite 102 Palo Alto, CA 94303 1-650-213-5600
- [12]. Grammatech (Dec 2013), "Eliminating Vulnerabilities in Third-Party Code with Binary Analysis Eliminating"
- [13]. Wikipedia, "List of tools for static code analysis", [Online]. Available:<https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis> [Accessed on 20 July 2016]

- [14]. Checkmarx, "Why Choose Us", [Online]. Available:<<http://lp.checkmarx.com/new-brand-general/>> [Accessed on 6 August 2016]
- [15]. StackOverflow, "Developer Survey Results 2016", [Online]. Available:<<http://stackoverflow.com/research/developer-survey-2016>> [Accessed on 25 March 2016]
- [16]. SANS, "SANS Top 25 Most Dangerous Software Errors", [Online]. Available:<<http://cwe.mitre.org/top25/>> [Accessed on 5 December 2016]
- [17]. Cigital "Top Web Application Security Vulnerabilities", [Online]. Available:<<https://www.cigital.com/blog/top-web-application-security-vulnerabilities/>> [Accessed on 5 December 2016]
- [18]. SANS "SANS TOP 25 Most Dangerous Software Errors", [Online]. Available:<<https://www.sans.org/top25-software-errors/>> [Accessed on 5 December 2016]
- [19]. OWASP "Mobile Top 10 2016-Top 10", [Online]. Available:<https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10> [Accessed on 6 December 2016]
- [20]. Checkmarx "The Ultimate List of Open Source Static Code Analysis Security Tools", [Online]. Available:<<https://www.checkmarx.com/2014/11/13/the-ultimate-list-of-open-source-static-code-analysis-security-tools/>> [Accessed on 6 December 2016]
- [21]. VisualCodeGrepper "Code security review tool for C/C++, C#, VB, PHP, Java and PL/SQL.", [Online]. Available:<<https://sourceforge.net/projects/visualcodegrepp/>> [Accessed on 8 December 2016]
- [22]. YASCA "Michael V. Scovetta", [Online]. Available:<<http://www.scovetta.com/yasca.html/>> [Accessed on 8 December 2016]
- [23]. OWASP LAPSE+ "OWASP LAPSE Project", [Online]. Available:<https://www.owasp.org/index.php/OWASP_LAPSE_Project> [Accessed on 8 December 2016]
- [24]. RIPS "A static source code analyser for vulnerabilities in PHP scripts", [Online]. Available:<<https://websec.files.wordpress.com/2010/11/rips-slides.pdf>> [Accessed on 10 December 2016]
- [25]. RIPS "A static source code analyser for vulnerabilities in PHP scripts", [Online]. Available:<<http://rips-scanner.sourceforge.net/>> [Accessed on 11 December 2016]
- [26]. DevBug "PHP Static Code Analysis (SCA) tool", [Online]. Available:<<http://www.devbug.co.uk/>> [Accessed on 22 November 2016]

- [27]. Flawfinder “Flawfinder”, [Online]
Available:<<https://www.dwheeler.com/flawfinder/>> [Accessed on 22 November 2016]
- [28]. CPPCheck “CPPCheck”, [Online]
Available:<<http://linux.softwsp.com/linux-development/miscellaneous-linux-development/cppcheck/>> [Accessed on 22 November 2016]
- [29]. Brakeman “Brakeman - Rails Security Scanner”, [Online]
Available:<<http://brakemanscanner.org/>> [Accessed on 25 November 2016]
- [30]. IBM “IBM Security AppScan Source”, [Online]
Available:<<http://www-03.ibm.com/software/products/en/appscan-source>> [Accessed on 22 December 2016]
- [31]. HP Enterprise “Fortify Static Code Analyzer”, [Online]
Available:<<http://www8.hp.com/us/en/software-solutions/static-code-analysis-sast/index.html>> [Accessed on 18 November 2016]
- [32]. Veracode “Veracode Static Analysis”, [Online]
Available:<<http://www.veracode.com/products/binary-static-analysis-sast>> [Accessed on 22 December 2016]
- [33]. Sentinel Source “WhiteHat Sentinel Source”, [Online]
Available:<<https://www.whitehatsec.com/products/static-application-security-testing/>> [Accessed on 27 December 2016]
- [34]. Checkmarx “Checkmarx Static Code Analysis (SAST)”, [Online]
Available:<<https://www.checkmarx.com/technology/static-code-analysis-sca/>> [Accessed on 27 December 2016]
- [35]. Security and Quality Software GmbH “Checkmarx STATIC APPLICATION SECURITY TESTING ”, [Online]
Available:<<http://sq-software.com/products/checkmarx/>> [Accessed on 23 December 2016]
- [36]. Black Duck “Open Source Application Security”, [Online]
Available:<<https://www.blackducksoftware.com>> [Accessed on 21 December 2016]
- [37]. WhiteSource “WhiteSource Secures Your Open Source Usage”, [Online]
Available:<<https://marketplace.visualstudio.com/items?itemName=whitesource.whitesource>> [Accessed on 28 December 2016]
- [38]. The Hacker News “These Top 10 Programming Languages Have Most Vulnerable Apps on the Internet”, [Online]
Available:<<http://thehackernews.com/2015/12/programming-language-security.html>> [Accessed on 12 December 2016]
- [39]. Scrapy “Scrapy at a glance”, [Online]
Available:<<https://doc.scrapy.org/en/latest/intro/overview.html>> [Accessed on 2 April 2016]

[40]. Browser Usage “Usage share of web browsers”, [Online]
Available: <https://en.wikipedia.org/wiki/Usage_share_of_web_browsers> [Accessed on 27 February 2017]

[41]. Chrome Extensions “What are extensions”, [Online]
Available: <<https://developer.chrome.com/extensions>> [Accessed on 22 May 2016]

[42] CodePlex “FuzzyString - Approximate String Comparison in C#”, [Online]
Available: <<https://fuzzystring.codeplex.com/>> [Accessed on 15 January 2017]

[43] Google Trends “Compare the code project vs stackoverflow”, [Online]
Available: <<https://trends.google.com/trends/explore?date=all&q=codeproject,stackoverflow>>
[Accessed on 27 February 2017]