International Journal of Computer Applications (0975 - 8887)
Volume 176 - No.32, June 2020

An Empirical Examination of the Relationship between Code Smells and Vulnerabilities

Aakanshi Gupta
ASET, GGSIPU, Delhi

Bharti Suri
USICT, GGSIPU, Delhi

Vijin Vincent
ASET, GGSIPU, Delhi

ABSTRACT

The quality of software is a crucial issue as a software system evolves. Managing source code smells and vulnerabilities contributes to software quality. In general, metrics have been used to classify code smells in source code, and an empirical examination is being considered in this paper on the correlation of code smells and vulnerabilities. For continuous inspection of code quality, Sonar Cloud has been used to conduct automated assessments with static code analysis to detect code smells and vulnerabilities with web scrapping technique. Web scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. In web scrapping technique, Selenium library provides sufficient tool to scrap data from Sonar Cloud. A statistical correlation approach is used to create a relationship between code smell and vulnerability that takes both dependent and independent values to measure coefficient of correlation. The conclusion of the study is, there exist vulnerabilities and code smells pair whose correlation coefficient is up to 0.93, which is quite sufficient to justify the results.

Keywords

Software Quality, Code Smell, Vulnerability

1. INTRODUCTION AND MOTIVATION

Vulnerability in the software system is the instructions in source code that can lead to major issues like security breaches and system crashes. Krsul [15] defined software vulnerability as an instance of an error in the specification, development, or configuration of software such that its execution can violate the security policy. Business systems managing confidential data and systems accessing the public networks (e-bank, e-commerce) are mostly affected by this problem. Controlling vulnerability and taking adequate precautions can help mitigate the effect. It has various types of problem related, which can intercept the development process as well as the maintenance phase of the software life cycle. Marija Kati [13] stated this fact in his paper. Moreover, the redesign is supposed to reduce software vulnerabilities. Research concerning this topic is still in infancy, and redesigning is done in two steps, code smell detection and refactoring. Code smells are another factor, that can adversely affect the maintenance phase. To remove a vulnerability inside a project, the first step is to detect the same, so here, finding a valid relation between code smell and the vulnerability, also the analysis of its behavioral change throughout different projects taken. For finding a valid relation between vulnerability and code smell using a correlation approach, a series of count values of both code smells and vulnerabilities of different projects are required. To obtain this, required a tool capable enough to find both code smells and vulnerabilities together of some software systems that is considered in this paper. Several automated tools already exist to detect vulnerable source code statements like Splint and Pixy etc. which can detect vulnerabilities by analyzing the static source code for this research work SonarCloud. This online platform, which suffices the analysis requirement.

Sonar Cloud is an open-source platform developed by Sonar source to analyze project source code saved in GitHub or BitBucket. Its analysis includes detection of the number of bugs, amount of code smells, amount of vulnerabilities, and calculations of technical debts of a project, etc. In this case, code smell and vulnerability attributes are considered for the study and analysis. SonarCloud uses its own set of rules for detecting code smell and vulnerability. SonarCloud is an online platform for analysis where SonarQube is the PC plugin used with Eclipse. Both use the same rules that are found in SonarSource. SonarSource has categorized the analysis metrics like code smell and vulnerability language wise i.e. each language has its own sets of rules defining code smells and vulnerabilities, which is present in SonarSource rule definition. There are 332 code smells and 49 vulnerabilities defined under Java language on SonarSource.

To perform correlation; there is required many instances that are having code smells and vulnerabilities so that can give a better result. Hence, 300+ projects are considered in this study. Analyzing each project individually and storing it manually might be a time consuming task. So, SonarCloud has an explore tab [6] in their site, which lists down all the projects analyzed through their online platform and the analyzed results are open for all visitor to be viewed. So, instead of manually analyzing all of the projects, scraping those data for this empirical study that is publicly available. Web scraping is a technique to extract huge amount of data from websites. It is mostly used when the respective sites from which; want to extract data do not provide specic API to acquire those data or when coping data manually is a bit cumbersome. Here, Selenium library is used for this purpose. Selenium is a python library, that is used to automate functional tests on the website but also can be used to extract web data. Selenium library provides functionality to select

Table 1. : Vulnerabilities and their Description

ID	Vulnerability	Description
V1	The "public static" fields should be constant	This rule states that a variable or functions defined as public and as well static should also be defined as final.
V2	Basic authentication should not be used	This rule suggest not to use basic authentication like Base64, whose encoding is simple and only offer thinnest veil of protection to user.
V3	Cross-document messaging domains should be carefully restricted	Html provides a functionality to send messages to document which is served from other domains. This rule raises issue whenever a "Window.postMessage" is used.
V4	Defined filters should be used	Filters may not invoke if filters defined under web.xml are not used in a $< filter - mapping >$ element.
V5	Function constructors should not be used	Function constructors can also be dangerous i.e. constructor's string arguments work similar to the way eval works, which can be both slow and a security risk.

Table 2. : Code Smells and their Description

ID	Code smell	Description
C1	Static non-final field names should comply with a naming convention	Static non-final fields should follow a naming convention; hence allow team to collaborate efficiently. This rule checks if the static non- final field match the provided regular expression. By default the regular expression is [a-z][a-zA-Z0-9]*$
C2	Exceptions should be either logged or rethrown but not both	In multi-threaded applications, it is a bad practise to log an exception and then rethrow it. This will end up in huge amount log statements that may contain multiple instance of the same exception. Hence, debugging of the same will be cumbersome.
C3	Extra semicolons should be removed	Places in the source code, containing extra semicolons causing error
C4	Variables and functions should not be redeclared	Redeclaration of variables and function should be avoided. Redeclaration can confuse maintainer and might also overwrite the previously used variables.
C5	Control structures should use curly braces	Use of curly brace is a good convention of coding even if it is not technically incorrect.
C6	Comma operator should not be used	In C language use of comma operator is normal but It could be detrimental to the readability and reliability of the code

HTML content and to get the same, the study aims at investigating the following research questions:

—RQ1:How the presence of vulnerability as well as code smell changes over various projects?

—RQ2:If there exist any pairs of code smell and vulnerability which has high correlation coefficient?

Motivation for work

—Code smells and vulnerabilities both degrade the software quality and make the software less maintainable.

—There is no study and analysis exist on the relationship of code smells and vulnerabilities as per our knowledge.

The paper is organized as follow: Section 2 presents the Background Theory follow by Section 3 describes the Experimental setup including Data Extraction and preparation. Section 4 describes the implementation steps with result analysis. Section 5, presents the conclusion and future work.

2. BACKGROUND THEORY

Code smells are the design defects that degenerate the software quality and maybe the reason for the software failure also. Refactoring is the technique for removing the code smells from the software as such that no impact on the external behavior of the software. On the other hand, Vulnerabilities are considered security issues in the software system.

Code smells are the symptoms of the presence of poor coding style in the source code. Code smells are a threat to software maintenance. Code smell detection is a very important step for the software in this new era. There are detection techniques like machine learning [7], association rule mining [3], change history information [18], cooperative parallel SBSE approach [14], weight-based distance metrics [5] and many more [10] available in the literature. Code smell prediction model also available in the published documents [9]. Vulnerabilities are the loophole in the source code that empowers an attacker to circumvent the security. Vulnerabilities are xed by the testing team for the reliability and security of the software. Camilo et al. [1] showed that both vulnerability, and bugs are empirically different and more research is needed for detecting vulnerability. Gopalakrishna et al. [8] measured vulnerability as four artifacts of programming. Those are programming style, programming mistakes, error-prone constructs, and privileged lines of code. Chowdhury et al. [2] compared the performance of C4.5 Decision Tree, Logistic Regression, Random Forests and Naive-Bayes for predicting vulnerability. Zimmermann et al. [23] presented an empirical study to evaluate that complexity, churn, coverage, dependency measure, and organizational structure to predict vulnerability. To the extent of our knowledge, no research or study on the relationship between code smell and vulnerability has been conducted earlier. However, it is necessary to point out that relationship between code smells and faults, code smells and bugs has been proposed. Sonar cloud considered as violations of both the code smells and vulnerabilities [19]. Code smells affected the software maintainability negatively [12,21] and increased the number of faults. The studies investigated the correlation between faults and code smells in the software. They have also identied the names of specic code smells that were changed more frequently. W.Li and R. Shatnawi [16] have found a relationship between class error probability and code smells with the proof that the code smells will be increased if there is a high probability of class errors in the software. Yamashita [22] identied the relationship between problems in the software and code smells using binary logistic regression. He also identied the harmful code smells in respect to maintenance. Rahman et al. [20] established a correlation between duplicate code smell and bugs. Juergensetal [11] dened and analyzed a correlation between inconsistent clone code smell and faults. They have also developed an algorithm for the detection of inconsistent clone bad smells. DAmbros et al. [4] found a relationship between code smells and faults. They justied the results that smells and faults are proportionally related to each other. Monden et al. [17] claried the correlation between clone code smells and software quality with the result that classes with clones are less maintainable. Falessi and Voegel [6] studied for nding a relationship between Sonar cloud violations and faults and showed that classes with higher violations would have higher faults. Code smells are design aws in object-oriented systems [24], which can lead to maintainability issues in the evolution of software system. Also, in the paper [25], it is stated that negligence and inconsiderateness in the design of a software system can also lead to defects in software logical workow. These defects can also cause the software to crash as well as can be the reason for attackers to exploit the defects for their benet [2]. Hence, design aws or defects can also be the cause of maintainability issues and future security vulnerability. Hence, in this paper, a systematic approach is taken to nd the relationship between code smell and vulnerability those are particularly related to some design aw.

3. EXPERIMENTAL SETUP

Web scraping is a task especially when if the website has dynamic content. BeautifulSoup is another python library for web scraping but only for static websites. Here, in our case SonarCloud load data dynamically. Hence, BeautifulSoup is unable to extract data from it; for overcoming this dynamic loading problem, Selenium is used, which has the functionality to wait until the component it is searching for, has completely loaded into the browser.

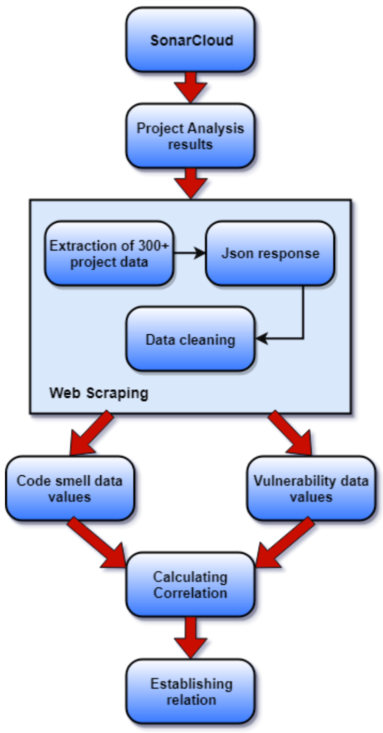
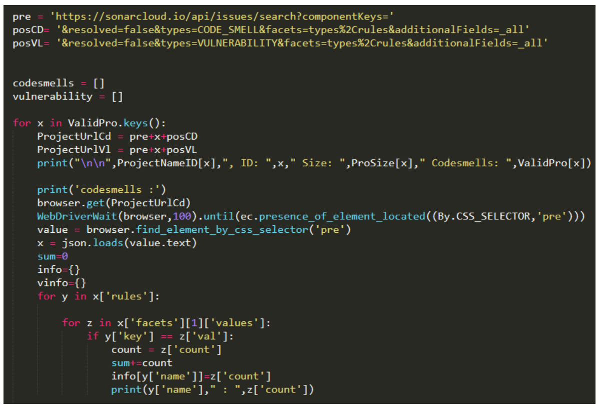


Fig. 1: Flow diagram of the methodology

As the study required a huge number of project data to be extracted from SonarCloud with specification of each and every code smell and vulnerability defined under SonarSource, a python script with Selenium library is created , which can get the appropriate projects for the study and then extract each projects code smell and vulnerability count data. The following fig. 1 shows the flow diagram of the methodology is being used in the experiment.

Data Extraction and Preparation

This section will describe the process taken to extract data for evaluating the correlation between code smell and vulnerability. SonarCloud support many languages including Java, PhP etc. Hence under explore option a mixture of project can be found of different languages. As our study only focuses on java projects, the filter option available on the site [7] is used to request all the projects under java language. Some of the projects analysis result revealed that huge projects with larger lines of code, analysis results could not be loaded completely into the site.



```python
pre = 'https://sonarcloud.io/api/issues/search?componentKeys='
posCD= '&resolved=false&types=CODE_SMELL&facets=types%2Crules&additionalFields=_all'
posVL= '&resolved=false&types=VULNERABILITY&facets=types%2Crules&additionalFields=_all'


codesmells = []
vulnerability = []

for x in ValidPro.keys():
    ProjectUrlCd = pre+x+posCD
    ProjectUrlVl = pre+x+posVL
    print("\n\n",ProjectNameID[x],", ID: ",x," Size: ",ProSize[x]," Codesmells: ",ValidPro[x])

    print('codesmells :')
    browser.get(ProjectUrlCd)
    WebDriverWait(browser,100).until(ec.presence_of_element_located((By.CSS_SELECTOR,'pre')))
    value = browser.find_element_by_css_selector('pre')
    x = json.loads(value.text)
    sum=0
    info={}
    vinfo={}
    for y in x['rules']:

        for z in x['facets'][1]['values']:
            if y['key'] == z['val']:
                count = z['count']
                sum+=count
                info[y['name']]=z['count']
                print(y['name']," : ",z['count'])
```

Fig. 2: Screen shot of the scripting for Webscrapping

Hence, a limit is introduced to the project size i.e. projects with total No. of code smell found should be less than 1000, and our second constraint was to select only those projects, which have at least one vulnerability issue. This constraint is achieved with SonarCloud helpful filter options, i.e. sorting out the project with their security issues. Since the data, which loads into the site are dynamically loaded, hence inspect the functionality of the browser is used to know its source. Under Network tab, the RequestURL can be found, which requests the data from the server and further loads it dynamically.

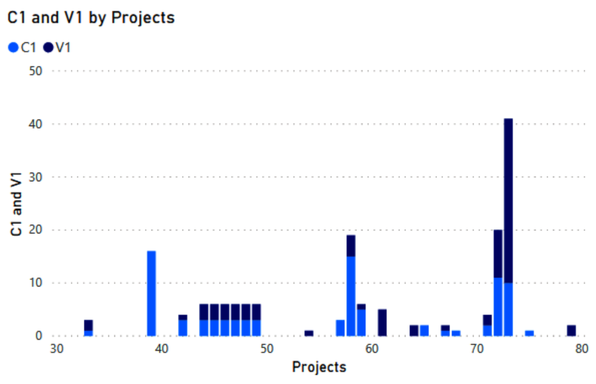


Fig. 3: The bar graph of both code smell and vulnerability across projects

This RequestURL is responsible for loading the project details into the site. This URL requests data in batches of 50, which further can be changed according to the need. Here in our case, it is set to 100, hence 100 project data can be loaded in an instance. It is achieved by changing the attribute ps=50 in the URL to ps=100. To extract this JSON data from this URL, Selenium is used. It is a library in python for web automation and its testing. Before developing the script with Selenium, the Chromedriver is installed, as needed by Selenium to execute the script. Selenium has its function to run the URL in Chrome driver. Driver.get(URL) Here Driver is a variable that represents the chrome driver and using its get function any URL can be executed in chrome browser. This function also returns the server response that it receives, which in our case are the JSON data. Here project ID is required for further extraction of code smell and vulnerability data specifically for each project respectively. Project detail could also be extracted using the same URL specified above. Hence using that URL useful details like project ID, project name, project size can be extracted in JSON format and stored in .csv file. After executing the get function 5 times, 500 projects are obtained and out of which. some of the projects are left as they do not follow the constraint previously defined. Finally, 312 projects ID is successfully obtained that fulfill the need. Now, the second most important step is to extract individual details about code smell and vulnerability for each project taken. In the Fig. 2 a snapshot of the screen has been shown for web scrapping scripting.

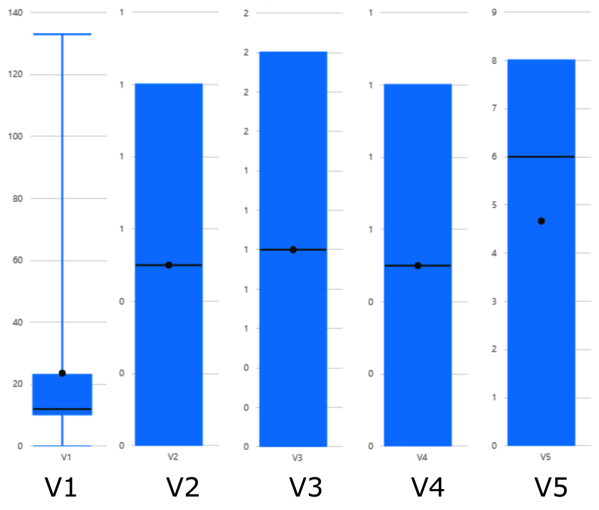


Fig. 4: Considered data set of Vulnerabilities over the projects

The code smell and vulnerability are two page sections in the site, which can be opened for inspection. SonarCloud provides an insight of the code smell and vulnerability present in the code and highlights its location with rule that it follows. There can be many similar code smell and vulnerability present in a project. To get those details with its frequency count, there is a rules section available under both the pages of code smell and vulnerability, which contains all the metric details and its respective frequency count. Those data can be extracted with its frequency in the same way as it has been done with the project detail extraction i.e. the data , which is loaded into the site also has a RequestURL as a

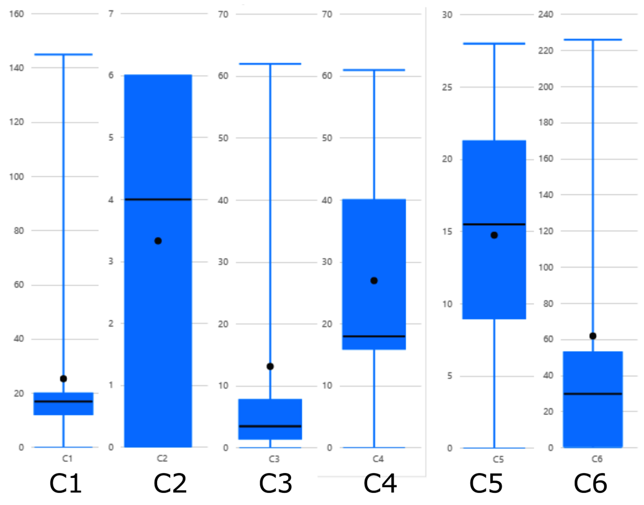


Fig. 5: Considered data set of Code Smells over the projects

source. Under the inspect option, the RequestURL for both code smell and vulnerability can be found inside the network section. Figure 3, shows the bar graph between vulnerability V1 and code smell C1 across the considered projects.

Hence, those data can also be request in JSON format and the values can be extracted in the same manner. In order to automate this process for all the projects, first a similarity is identified between varies RequestURLs of many different projects taken and found that the URL body differ only at the project ID part and everything else are similar. Their RequestURL follows the sequence as follow:
A + ProjectID + B1
RequestUrl for Vulnerability Values:
A + ProjectID + B2
where
A : `(https://sonarcloud.io/api/issues/search?componentKeys=)`
B1 : `(&resolved=false&types=CODE_SMELL&facets=types%2Crules&additionalFields=_all)`
B2 : `(&resolved=false&types=VULNERABILITY&facets=types%2Crules&additionalFields=_all)`

Hence, the already extracted project ID is iterated one by one to create a custom URL for each project and that can be used to extract the JSON data for both code smell and vulnerability respectively. After getting the data in JSON format, extraction of its name and its frequency count is done and saved those extracted data into python dictionary data variable, each for both code smell and vulnerability for every project. The key value of the dictionary variable represents the attribute name and the value part represents its frequency count. At the end of the iteration, a list of dictionary variable is obtained where each element represents code smell as well as vulnerability details for a specific project. Those list values are further converted into a Pandas Data frame variable. Pandas is a library in python , which provide many useful data processing functionality for data analysis. Hence, the converted list value to pandas data frame can further be used for correlation calculation. Figure 4 and Figure 5 are the box and plot graphs of code smell and vulnerability taken in this research. Box and plot graph usually tells the distribution of data, so here it tells the distribution of the amount of values of code smell and vulnerability over the projects.

4. IMPLEMENTATION AND RESULT ANALYSIS

Since, all the data has been extracted and stored, next step is to find the correlation coefficient between two distinct code smell and vulnerability. Here, Pearson Correlation Coefficient is evaluated between both the variables. Pearson Correlation Coefficient, also known as Pearson product-moment correlation coefficient (PPMCC), is measure of correlation between two variables.

Table 3. : Vulnerabilities and Code Smells Correlation Values

	Vulnerability	Code Smell	Correlation Values
1	public static fields should be constant	Static non-final field names should comply with a naming convention	0.92181146
2	Basic authentication should not be used	Exceptions should be either logged or rethrown but not both	0.92107494
3	Cross-document messaging domains should be carefully restricted	Comma operator should not be used	0.96249069
4	Cross-document messaging domains should be carefully restricted	Extra semicolons should be removed	0.98421813
5	Cross-document messaging domains should be carefully restricted	Variables and functions should not be redeclared	0.91758443
6	Defined filters should be used	Control structures should use curly braces	0.91634193
7	Function constructors should not be used	Comma operator should not be used	0.92167049
8	Function constructors should not be used	Extra semicolons should be removed	0.96510528
9	Function constructors should not be used	Variables and functions should not be re-declared	0.92946156

It has value between +1 and -1, where 1 represents positive correlation, 0 is no linear correlation and -1 represents negative

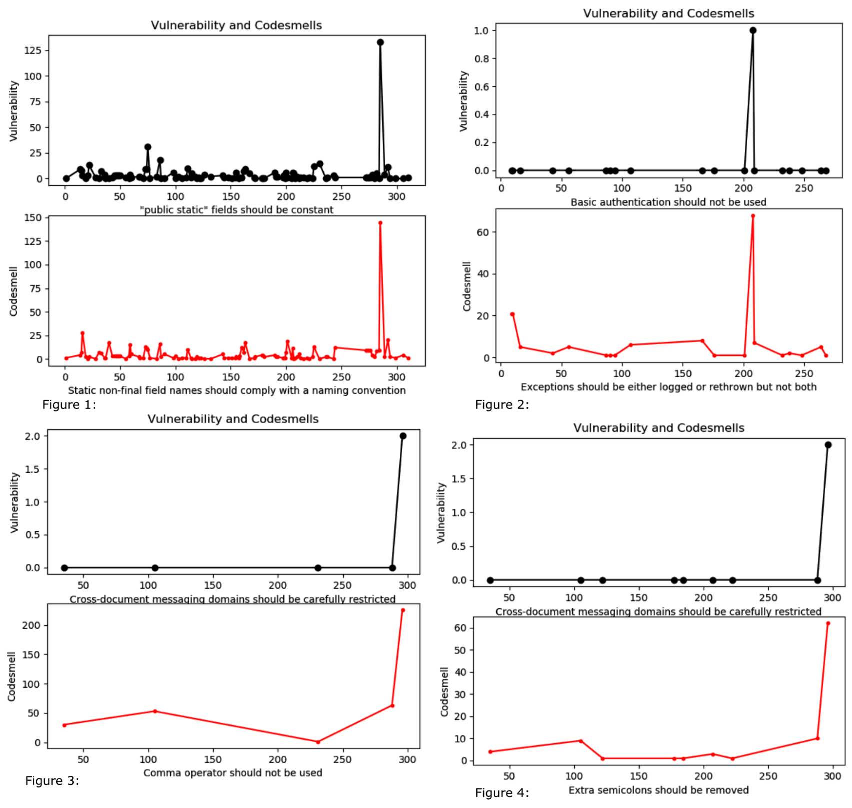


Fig. 6: Result analysis through graphs between Code smells and Vulnerabilities

linear correlation. Pandas is one of the great libraries available in python. It is an open source, data structure and data analysis tool for python language. Since the values has been already stored in python data frame, the correlation between two different attribute can easily be evaluated using data frames corr() function. There exist two data frames, as one for code smell and the other for vulnerability. Each row in both the data frames represents different code smell and vulnerability values representing a project and each column represents a code smell and vulnerability and its count values respectively. Hence, using data frames correlation functions, the correlation values are calculated between every attribute from vulnerability data frame to every attributes of code smell data frame.

There are many intermittent code smells and vulnerabilities defined under SonarSource, which occur occasionally. While taking the attribute values from data frame, there might be a case when each attribute contains zero values. This will affect the result as the long stream of zeros in between attribute value might astray the correlation coexistent with higher values. In order to reduce its affect, those rows are deleted where both the attribute values are null or zero. Hence, at the end of operation correlation values are obtained for each combination of code smell and vulnerability and the coefficient values higher than 0.8 are stored locally.
The result obtained from this study is the analysis taken place with multiple number of java projects extracted from SonarCloud and its analysis result respectively. 300+ projects are taken for this analysis where the project size is not the same. Using the python script, the results of projects available in SonarCloud website are extracted and stored in CSV format for further analysis. Further, Pearson correlation method has been applied between each vulnerability with every code smell and some graphs have been plotted shown in the Figure 6 and Figure 7 over those values. Here, public static fields should be constant(vulnerability) and Static non-final field names should comply with a naming convention (code smell). Finally, a table is created storing vulnerability and respective code smell name with its correlation coefficient values as shown in the table 3. This table only contains values higher than 0.8, all the other are left ignored as they does not represent a good relation.

RQ1: How the presence of vulnerability as well as code smell changes over various projects?
As visible from the graphs, that there are some vulnerability and code smell pairs whose frequency count changes in amount, in the same manner throughout the projects that have taken in this research. In Fig.6, V1, a vulnerability defined under Sonarsource rules[5] and C1, a code smell defined under Sonarsource rules[5] are changing in the same manner.
RQ2: If there exist any pairs of code smell and vulnerability which has high correlation coefficient?

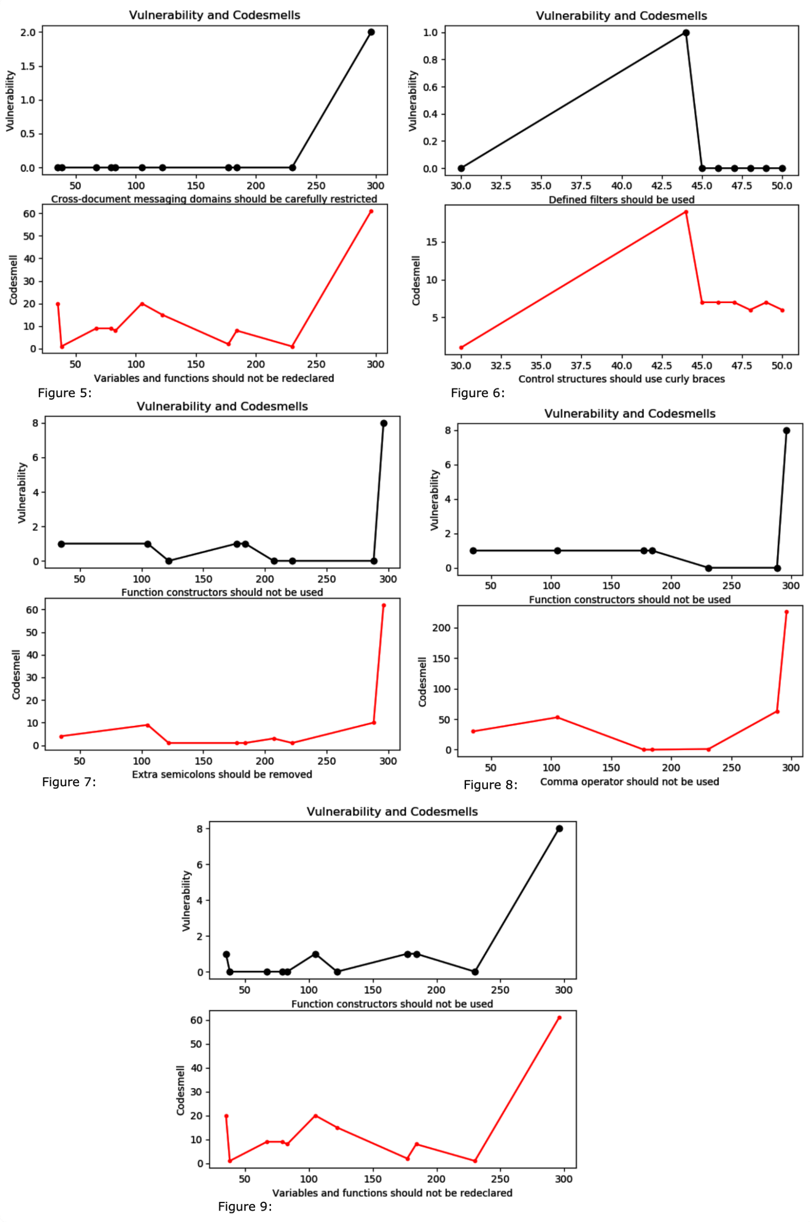


Fig. 7: Result analysis through graphs between Code smells and Vulnerabilities

After the calculation of correlation coefficient value between each vulnerabilities with every code smells, a significant number of code smell and vulnerability pairs has been found values higher than 0.8.

5. CONCLUSION

This research represents data collection and its analysis to calculate a relationship between code smells and vulnerabilities. Code smell can also be a cause of some rising vulnerability in software system. This study is carried out with an online tool called SonarCloud, which is used to analysis software source code of any bugs, code smells and vulnerabilities. Since, the study focuses on vulnerability and code smell, hence both the attributes have been considered in this paper. In order to find a valid relation among both the attributes, Pearson correlation coefficient methodology and various graphs are analyzed. To find correlation coefficient a sufficient amount of data set is essentially required. Here, instead of manually gathering the project data for code smell and vulnerability values, web scraping technique is employed for the same. Web scraping technique is used to extract 300+ project details that had already been submitted and analyzed by other users. Further, extracting the individual code smell and vulnerability values, they were stored in CSV file format locally for further correlation calculation. The

extracted data has been cleaned for better correlation calculation. After successful extraction of data from SonarCloud, correlation between each vulnerability with every code smell is calculate. For every relation with correlation value greater than 0.8, a graph has been plotted. The graphs, in the figure 5 and figure 6, shows a visible pattern reflecting the behavior of various pairs of vulnerability and code smell among various projects taken. As, throughout the 300+ project; there exist some pairs of vulnerability and code smell together whose correlation values is greater than 0.8 as in the table 3, a value supposed to be a good indication of better relation. The method used in the study can further be carried out with a greater number of projects taken for finding out new pairs of code smell and vulnerability.
Correlation coefficient can have higher positive or negative value if the instances of data values are less. Hence, greater number of instance values are required for better results. Since taking more projects can increase their occurrence, thus 300+ projects were taken to overcome that issue. Second issue can be the continues stream of zero values throughout the data values; this issue can also cause the correlation values to go higher. In order to overcome the same, the pair of rows having zero values is deleted while calculating the correlation coefficient. In this way, the research is been carried out for getting the satisfactory outcome.

6. FUTURE WORK

Ahmed et al. [26] analyzed 220 open source projects and their result tells that ignoring code smells leads to software decay. Here in this paper, code smell is taken as the decaying factor but with relationship with the. The combination of both code smell and vulnerability can help developers to find vulnerabilities before it is maliciously used by intruders. As both code smell and vulnerability are not good for a software life cycle. Code smell are meant to be detected and refactored earlier before it causes serious problem. Hence, finding any code smell causing vulnerability could be of great use. In this study a sequential process is taken to find these relations. The obtained result is small with some good correlation coefficient value. Still, the method used in this study can be used to find more valuable relation between code smell and vulnerability. May be different tools can be used, which gives code smell output in terms of Fowlers named code smell with the respective vulnerability, employed with the calculation of coefficient value between them. Different languages might have different vulnerabilities; hence the relation can be recalculated for those different language platforms of development. Using the same method more pairs of code smell and vulnerability relation can be found by taking more project into account. As in this study only java projects are considered, other languages can also be chosen for further study. The extracted data using web scraping can also be used for other statistical methods for find relations like Spearman correlation, MIC (Mutual Information Coefficient), Hoeffding D method, marginal and conditional distributions etc.

7. REFERENCES

[1] Felivel Camilo, Andrew Meneely, and Meiyappan Nagappan. Do bugs foreshadow vulnerabilities?: a study of the chromium project. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 269–279. IEEE Press, 2015.

[2] Istehad Chowdhury and Mohammad Zulkernine. Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities. *Journal of Systems Architecture*, 57(3):294–313, 2011.

[3] Gabriela Czibula, Zsuzsanna Marian, and Istvan Gergely Czibula. Detecting software design defects using relational association rule mining. *Knowledge and information systems*, 42(3):545–577, 2015.

[4] Marco D'Ambros, Alberto Bacchelli, and Michele Lanza. On the impact of design flaws on software defects, 2010.

[5] Jiang Dexun, Ma Peijun, Su Xiaohong, and Wang Tiantian. Detecting bad smells with weight based distance metrics theory. In *2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control*, pages 299–304. IEEE, 2012.

[6] Davide Falessi and Alexander Voegele. Validating and prioritizing quality rules for managing technical debt: An industrial case study, 2015.

[7] Francesca Arcelli Fontana, Mika V Mäntylä, Marco Zanoni, and Alessandro Marino. Comparing and experimenting machine learning techniques for code smell detection. *Empirical Software Engineering*, 21(3):1143–1191, 2016.

[8] Rajeev Gopalakrishna, E Spafford, and Jan Vitek. Vulnerability likelihood: A probabilistic approach to software assurance. *CERIAS, Purdue Univeristy Tech. Rep*, 6:2005, 2005.

[9] Aakanshi Gupta, Bharti Suri, Vijay Kumar, Sanjay Misra, Tomas Blažauskas, and Robertas Damaševičius. Software code smell prediction model using shannon, rényi and tsallis entropies. *Entropy*, 20(5):372, 2018.

[10] Aakanshi Gupta, Bharti Suri, and Sanjay Misra. A systematic literature review: Code bad smells in java source code. In *International Conference on Computational Science and Its Applications*, pages 665–682. Springer, 2017.

[11] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, and Stefan Wagner. Do code clones matter?, 2009.

[12] Marija Katić and Krešimir Fertalj. Challenges and discussion of software redesign, 2009.

[13] Marija Katić and Krešimir Fertalj. Challenges and discussion of software redesign. In *Proceedings of the 4th International Conference on Information Technology*, 2009.

[14] Wael Kessentini, Marouane Kessentini, Houari Sahraoui, Slim Bechikh, and Ali Ouni. A cooperative parallel search-based software engineering approach for code-smells detection. *IEEE Transactions on Software Engineering*, 40(9):841–861, 2014.

[15] Ivan Victor Krsul. *Software vulnerability analysis*. Purdue University West Lafayette, IN, 1998.

[16] Wei Li and Raed Shatnawi. An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution, 2007.

[17] Akito Monden, Daikai Nakae, Toshihiro Kamiya, Shin-ichi Sato, and Ken-ichi Matsumoto. Software quality analysis by code clones in industrial legacy software, 2002.

[18] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. Detecting bad smells in source code using change history information. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, pages 268–278. IEEE Press, 2013.

[19] Luca Pellegrini, Andrea Alexander Janes, and Davide Taibi. On the fault proneness of sonarqube technical debt violations. an empirical study, 2018.

[20] Foyzur Rahman, Christian Bird, and Premkumar Devanbu. Clones: What is that smell?, 2012.

[21] Dag IK Sjøberg, Aiko Yamashita, Bente CD Anda, Audris Mockus, and Tore Dybå. Quantifying the effect of code smells on maintenance effort, 2013.

[22] Aiko Yamashita. Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data, 2014.

[23] Thomas Zimmermann, Nachiappan Nagappan, and Laurie Williams. Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista. In *2010 Third International Conference on Software Testing, Verification and Validation*, pages 421–428. IEEE, 2010.

[24] Olbrich, Steffen and Cruzes, Daniela S and Basili, Victor and Zazworka, Nico. The evolution and impact of code smells: A case study of two open source systems. In *2009 3rd international symposium on empirical software engineering and measurement*, pages 390–400. IEEE, 2009.

[25] Ping, Liu and Jin, Su and Xinfeng, Yang. Research on software security vulnerability detection technology. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, pages 1873–1876. IEEE, 2011.

[26] Mannan, Umme Ayda and Ahmed, Iftekhar and Almurshed, Rana Abdullah M and Dig, Danny and Jensen, Carlos. Understanding code smells in Android applications. In *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 225–236. IEEE, 2016.