Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 3, Issue 9 , 2025

OPEN SOURCE SOFTWARE VULNERABILITY ASSESSMENT AND PREDICTION: USING DATA MINING AND MACHINE LEARNING TECHNIQUES

Muhammad Janas Khan[1], Ahmad Zia[2]
Faculty of Computer Science, Department of Software Engineering, Afghan International Islamic University Afghanistan
Email: muhammadjanaskhan@gmail.com, a.zia7544@gmail.com

DOI https://doi.org/10.5281/zenodo.17286046

Keywords
Software Vulnerabilities
Vulnerability Prediction
Vulnerability Trends
Open Source Software

Article History
Received on 26 Aug 2025
Accepted on 8 Sep 2025
Published on 29 Sep 2025

Corresponding Author: *
Muhammad Janas khan

Abstract
Software vulnerabilities are among some critical issues in system information security. Security defects in software cost millions of dollars to firms in terms of downtime, disruptions, and confidentiality breaches. There is a need for timely exposure and exclusion of vulnerabilities to improve computer systems' security and avoid the victims from exploiting various vulnerabilities. As technology has progressed, there have been many different ways to find zero-day flaws in software systems. Still, most of these techniques are suggested for detecting one or a limited number of vulnerability classes. We used data mining and machine learning techniques to address the scenario in this research work. The mining of trends and vulnerabilities patterns is useful. They can help software vendors prepare solutions before time for vulnerabilities that may occur in a software application. To predict future recurring vulnerabilities in OSS (Open Source Software) applications, we tried to investigate the use of historical patterns of vulnerabilities. We examined the software vulnerabilities taken from CWE (Common Weakness Enumeration) identifier/organization, CVE (Common Vulnerability Exposure) and NVD (National Vulnerability Database) in the duration from 2006-2016. Our findings revealed that the DOS (Denial of Services), 'code execution', 'overflow', 'memory corruption' and 'bypass something' are some critical vulnerabilities faced to OSS applications.

INTRODUCTION

Developing secure software is a time-consuming and complex activity. The vulnerabilities are the primary source of software insecurity. So the prediction of software vulnerabilities plays a key role in software engineering, specifically in web application development. Web applications have become more prevalent in the last two decades than traditional businesses. On one side, the internet is a vital source of useful information, but it also causes some dangerous issues like breaches of personal privacy

and security. In the present day, software systems require to make certain the protection of sensitive and confidential information along with providing the superiority of services to its end users. Frequently software is an aggressive big business that can be revolutionized quickly and takes action to changes in marketplace, hardware and software podiums.

A software vulnerability is a security error, flaw, fault, failure or weakness found in a software or in operating system that can lead to security concerns [1-3]. Software vulnerability is "an occurrence of a mistake in the specification, development, or configuration of software such that its execution can disobey the explicit or implicit security policy" [4, 5]. Software vulnerabilities are major concern in all software systems whether they are open sourced or close sourced. Both vendor/developer organizations and users require being capable to evaluate the security intensity of software systems in sort to advance the efficiency of software. The protection of a software system is impacted by the existence of unpatched vulnerabilities which are exposed from time to time and can be significantly exploited [6-8]. If we might foresee the probable vulnerability trends, patterns and the characteristics of the vulnerabilities revealed, the desirable resources might be owed at the correct time for remedial actions which would significantly diminish security threats. Such methods might also be utilized by end users to asses' risks and be equipped to get corrective action when possible security violations happen.

Quantitative techniques will formulate it probably for end users and developers to impartially evaluate the threat faced by vulnerabilities in software systems and their possible violation. Nowadays researchers have commenced to discover some of the quantitative distinctiveness of vulnerabilities as the vital datasets which are becoming suitably large to be analyzed [9-11]. Zero day vulnerabilities prolong to be a threat such that the vendors are unfamiliar with them and when attacks occurs, vendors have zero days to afford mitigations [12, 13]. Zero day vulnerabilities are generally difficult to prevent such that usually the information is only available for analysis after the attack has been completed [16, 17]. In some cases, it is very difficult to discover all the vulnerabilities simultaneously even if every security protocol was used at the time of launch [18]. It is still possible for hackers in the various types of security exploits to take advantage if something is left behind in the code by a code-developer [19-21].

In such context, this paper provides the following main contributions:

Exploring the trend of vulnerabilities over a period of time in software applications

Exploring the trend of vulnerabilities over a period of time in some popular open source software applications

Exploring the common patterns of vulnerability in open source software applications

Exploring the top most types of vulnerability across the selected open source software applications

Exploring the next possible vulnerability/vulnerability pattern recurring in some open source software applications

In order to get hold on the above mentioned contributions, we set the following RQs (Research Questions):

: Over time, how much of a rise or fall can we expect in the frequency of software vulnerabilities?

: Do popular OSS software have a long-term trend of vulnerabilities?

: Are there any patterns to the vulnerabilities that are found in different open source software applications?

: Do the most popular OSS applications have the most common sorts of vulnerabilities?

: Why does a particular vulnerability/vulnerability pattern keep reoccurring in an OSS application?

The paper is organized as follows. Related works are presented in Section 2. The research method used in our assessment is outlined in Section 3, where different aspects are presented to analyze our datasets. The results of this research and its analysis are described in Section 4. Limitations of the study are

discussed in Section 5. Finally, the conclusions and future work are presented in Section 6.

Here, we exploit the NVD from [22] and CWE from [23], for long term annual periodicity analysis of different OSS applications. We chooses these organizations because of their public access to most extensive and structured datasets. Further, the NVD project is supported by the US department of Homeland Security and sustained by the National Institute of Standard and Technology, it can be painstaking a standard foundation. The NVD uses CVE identifiers [24] and so any updates to CVE directly come into view on NVD.

2. Related Work

Open Source Software (OSS) is a software tool that operates under an open source label whose source code should be available and modifiable. Throughout the world an increasing number of people are using OSS with an open source code, where it generally operates securely and reliably in a stable and cost-effective manner. OSS use has moved from the fringes into the mainstream of software development, accelerating time to market and generating cost savings. But despites these benefits, a thoughtful approach to adoption are essential. Lack of visibility into the open source software in use - particularly to known open source security vulnerabilities that have found their way into the codebase – exposes organizations to exploitation that code result in financial losses, business disruption, customer defection, legal action and brand risk [25]. Currently, OSS products have started to become popular in the market as an alternative to traditional closed source software [26]. Government and organizations are beginning to adopt OSS on a large scale and several governmental initiatives have encouraged the use of OSS in the private sector. One major issue for the government and private sector is the selection of appropriate OSS [26].

In recent times, the impact of progression caused by consecutive software versions on the vulnerability innovation development have been scrutinized [27-29]. In this learning they have measured the impact of giving out code across succeeding versions. The rising accessibility and exploitation of OSS in individual and marketable environments makes OSS interesting for hackers, and others who are paying attention in exploiting software vulnerabilities [30, 31]. Alhazmi et. al. [32] uncover numerical proof for their model for both closed source software and OSS. They identified vulnerabilities, as provided by the US NVD [22] Version 2.0, and obtain these information significant to OSS: Red Hat Linux 6.2: 123 and Red Hat Linux 7.1: 163. They additionally discover that the standards of vulnerability concentrations fall within a variety of values, just like the frequently used measure of defect density for general defects. Pham et. al. [33] performed an empirical study on thousands of vulnerabilities and initiate that many of vulnerabilities are recurring in OSS due to recycle source code or libraries. Wijayasekara et. al. [34] hidden impact vulnerabilities, as the name implies, are vulnerabilities that appear long after the flaw has been made public. From 2006 to 2011, two OSS, the Linux Kernel and MySQL software application, have 32 and 62 percent hidden effect vulnerabilities, respectively, according to CVE [24] and bug databases.

An improved description of the AML (Alhazmi-Malayia Logistic)[35] security detection model were projected by taking into deliberation the superposition [36] of the discovery processes for multiple version software systems and demonstrated their model by exploratory two OSS systems.Neuhas and Zimmermann [37] to automatically divide vulnerabilities into different kinds, use CVE vulnerability documentation from before 2009. They claimed that there are a large number of CWE [23] vulnerability categories, roughly 700, that are beyond human comprehension. Zaman et al.[38] conducted an exploratory research on the assessment of security and performance problems in open-source software (OSS), such as the Firefox web browser, and discovered security bugs using CVE repositories. Security bugs are unchanging faster than design defects, and they can be reopened several times in a

bug repository, according to them. By probing the impact and incidence of vulnerabilities in NVD and guesstimating the mishandling incidence and misuse impact of vulnerabilities, Houmb and Franqueria[39] predicted the possible amount of vulnerabilities e.g. the severity of the punishment for attacks that exploit flaws, as well as an estimate of the likelihood level of vulnerabilities at different periods. Frei et al. [40] utilise NVD and other comparable repositories to measure the gaps between the dates of vulnerability invention, molest discovery, and patch accessibility. They additionally discovered that attacks of zero day vulnerabilities are vacant directly in NVD on the date of exposé of vulnerabilities but software vendors are sluggish to provide software patches.

Several VDMs (Vulnerability Discovery Models) to facilitate model the vulnerability discovery method have been projected in recent times. These incorporate Andersons thermodynamic mode [41], AML model [35] and Rescorlas linear/exponential models [42]. Various classical software consistency growth models have been used [43]. Chen et. al. [27] have obtainable a multi-cycle VDM integrating a sinusoidal performance. Their model endeavor to model the relationship involving the number of vulnerabilities and their release time and its compared with other VDMs using data sets from eight Windows operating systems.

3. Vulnerabilities Measuring Methods

The general idea about the study is presented in **Fig. 1**. The data was collected from CWE [23], CVE [24] and NVD [22]. The details about these terminologies are presented next:

CWE is a directory of software weakness types that is anticipated as a community developed absolute dictionary of software weakness.

CWE™ International in extent and liberated for public use which can afford a cohesive, measureable set of software weakness that enables more effective discussions.

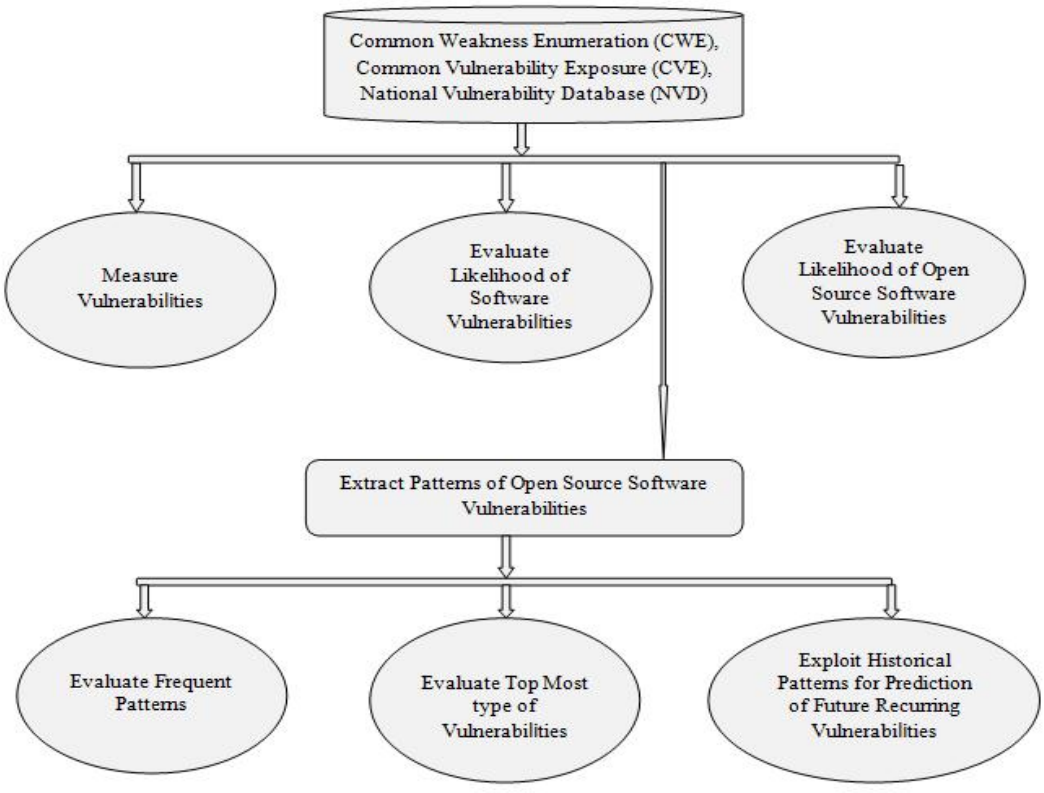

FIG. 1.OVERVIEW OF THE STUDY

Explanation, assortment and use of software safety tools and services that can discover these weaknesses in source code and operational systems along with better perception and management of software weaknesses associated to architecture and design [23]. It is sponsored by the US-CERT in bureau of Cyber Security and Communications at U.S. Department of Homeland Security.

- CVE is a chronicle trade name of the MITRE Firm. CVE is an open accessible, complimentary for use software weakness dictionary of standardized identifiers relating to general computer vulnerabilities and exposures. The CVE's widespread identifiers facilitate data exchange among security products and present a baseline catalog point for assessing coverage of tools and services [24].

NVD is the U.S government repository of standards based vulnerability calculated data represented using the SCAP (Security Content Automation Protocol) [22]. This data facilitates the computerization of vulnerability managing, safety dimensions, and observance. NVD incorporates databases of security checklists, security interrelated software defects and flaws, product names, and impact metrics.

We used the CWE, CVE and NVD repositories. Throughout these repositories, a total number of 88,477 vulnerabilities have been mined from January 2006 to October 2016 as shown in the Table 1.

Table 1. List of total Vulnerabilities by Types

No.	Types of Vulnerability	Years											
		2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	Total Vulnerabilities
1	Denial of Services	893	1100	894	1035	1102	1221	1425	1453	1597	1781	1660	14161
2	Code Execution	2719	2601	2310	2184	1714	1334	1457	1186	1573	1808	1159	20045
3	Overflow	662	952	699	700	678	770	842	859	850	1073	1062	9147
4	Memory Corruption	91	95	128	188	342	351	423	366	420	746	587	3737
5	Sql Injection	967	706	1101	963	520	294	242	156	304	213	53	5519
6	Cross Side Scripting (XSS)	1302	883	807	851	605	467	758	650	1105	757	357	8542
7	Directory Traversal	321	339	363	322	275	108	122	110	204	145	76	2385
8	Http Response Splitting	8	14	7	9	8	7	13	7	12	12	14	111
9	Bypass Some thing	267	267	288	338	234	197	343	352	457	577	373	3693

10	Gain Information	267	322	270	302	282	409	389	510	2104	745	650	6250
11	Gain Privileges	184	242	188	223	237	206	250	274	239	366	462	2871
12	Cross site request forgery (CSRF)	18	69	83	115	86	58	166	123	264	245	70	1297
13	File Inclusion	849	700	170	138	73	17	14	1	2	5	1	1970
14	Number of Exploits	97	1251	1936	2108	1469	564	615	202	390	117	0	8749
Total Vulnerabilities		8645	9541	9244	9476	7625	6003	7059	6249	9521	8590	6524	88,477

The following steps we followed in our research:

3.1 Step-1: Assessing the Significance of the Trends in Vulnerabilities Across Software Applications

We used the Equation (1) [18] to measure the likelihood of vulnerability in a specific year.

$$\text{Likelihood}\ (v_k, y) = \frac{\text{count}\ (v_k, y)}{\Sigma v \epsilon\ V\ \text{count}\ (v, y)} \quad (1)$$

"This equation tells that the likelihood of a vulnerability (v_k) in a specific year (y) is equal to counting the frequency of that vulnerability in a year and then dividing it by the total amount of the entire vulnerabilities in that year".

Where v_k ϵV and V is a set of the entire vulnerabilities types in a given year. We continued this process and measured the likelihood of all vulnerabilities in each year from 2006-2016. In order to measure that either there exists any significant differences in the likelihood of vulnerabilities types across different years, we applied the Chi-Square test from [19]. The test was also used to measure whether the trend of vulnerabilities over a period of time is rising or declining. The same test was used by the other studies in software engineering [44-48].

3.2 Step-2: Measuring the Significance of the Trends of Vulnerabilities of some popular OSS Applications:

"To measure the likelihood of vulnerabilities in an OSS application over a period of time, we used the Equation (2) from [18]".

$$\text{Likelihood}\ (o_k, V, y) = \frac{\text{count}\ (o_k,\ V,\ y)}{\Sigma o \epsilon\ O\ \text{count}\ (o,\ V,\ y)} \quad (2)$$

"Where o_k represents an OSS application for which we measured the likelihood of vulnerability in a particular year y, V is a set of all vulnerabilities in an application through a given year y, and O is a set of all applications through a given year y". "Equation (2) tells that measuring the likelihood of vulnerability of an application is such that counting the occurrences of all vulnerabilities in an application during a year and dividing it by total number of occurrences of all vulnerabilities in all the applications in a particular year". Through the help of such equation, we measured the likelihood of vulnerabilities in fifteen OSS applications. After then we again applied the Chi-Square test to measure that either there exists any significant differences in these OSS applications or not.

3.3 Step-3: Determining recurrent N-Itemset to measure frequent patterns in an OSS Applications:

To extract patterns of various vulnerabilities, we used N-Itemset pattern Apriori Algorithm from [49]. This algorithm is used to mine chronological patterns of vulnerabilities from a sequence of vulnerabilities by sliding a window of a given length. Through the use of such algorithm, we mined the N-Itemset pattern of vulnerabilities of each OSS application across each year. Let's take a look on the working process of this algorithm, for example, {O_1, O_2, O_3... O_k} is a

sequence of K vulnerabilities occurring consecutively within an OSS application in a year. And we using length 4-Itemset pattern of vulnerabilities for this OSS application in that year such as: {O_1, O_2, ..., O_4}, {O_2, O_3, ..., O_5}, {O_3, O_4, ...O_6}..... {O_{K-3}, O_{K-2}, ...O_K}. Further, In order to better mine the recurrent pattern of vulnerabilities in the selected OSS applications, we are using 2, 3 and 4-Itemset pattern of vulnerabilities for OSS applications across each year. The selection of N-Itemset are dependent on the data sets and the determination of the user.

For example, {Denial of services, Code execution, Overflow, Code execution, Overflow} are the various types of vulnerabilities that occur in a sequence on an OSS application. The 2-Itemset pattern from this sequence can be made as: {(Denial of services, Code execution), (Code execution, Overflow), (Overflow, Code execution) and (Code execution, Overflow)}. The last pattern, "Code execution, Overflow" could occurs twice in this sequence and we need to extract only this pattern. We further need to sum the frequencies of unique pattern occurred in the OSS applications. Table 2, presents the final 2-Itemset pattern along with their frequencies for this example.

Table 2. Example of 2-Itemset Pattern of Vulnerabilities

No.	2-Itemset Pattern	Frequency
1.	Code execution, Overflow	2
2.	Denial of Services, Code execution	1
3.	Overflow, Code execution	1

To answer RQ3, we mine the length 2, 3 and 4-Itemset pattern of vulnerabilities for each OSS application across each year. We also mine the most frequent recurrent Itemset pattern of vulnerabilities in the selected OSS application across all years. Suppose a 4-Itemset {Denial of service, Code execution, Overflow, Sql Injection} occurred 20 times in one OSS application and 30 times in an another OSS application across all the years. Then the total frequency of 4-Itemset occurrences will be 50. The most frequent 4-Itemset of vulnerabilities in OSS application will be that one which occurred mostly than the remaining ones. By this way the measurement of all 2, 3 and 4-Itemset pattern of vulnerabilities across all OSS applications will turn the results of RQ3.

3.4 Step-4: Using Historical Recurrent N-Itemset to measure future patterns in an OSS Applications:

For evaluation, validation and accuracy purposes, we used the machine learning and validation techniques. In order to predict the future vulnerabilities in OSS applications, we used the machine learning algorithm N-Itemset (n-gram) of historical vulnerabilities in various OSS applications from CWE, CVE and NVD repositories. For this purpose, we matched the first little vulnerability in an OSS application with the historical N-Itemset to predict the recurrent future vulnerability or pattern. To determine the N^{th} vulnerability in an OSS application from the historical collection of N-Itemset, we used N-1 vulnerabilities in an OSS application. For the determination of the second vulnerability in an OSS application, there was a need to use its first vulnerability in 2-Itemset. After a match found, we predicted about the next possible recurrent vulnerability in an OSS application. Similarly, for the determination of the third and fourth vulnerability in an OSS application, there was a need of first two vulnerabilities in 3-Itemset and first three vulnerabilities in 4-Itemset.

To find out the precision and recall, there was a need to split our data into training and testing set as mentioned in [18]. Fig. 2, gives the concept for the recall and precision procedures. In pattern recognition, information retrieval and binary classification, precision(also called positive predictive value) is the fraction of relevant instances among the retrieved instances, while recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant

instances. Both precision and recall are therefore based on an understanding and measure of relevance.

Training set was used for the calculation of N-Itemset pattern whereas the test case was used for the calculation of precision and recall values of each pattern. The experimentation was performed in two phases such that in the first phase, the training set data was taken from 2006-2011 and from 2012-2016. For the second phase, the training set data was taken from 2013-2014 whereas the testing set data was taken from 2015-2016. Further detail about the experimentation is coming next in section III. In order to find out the precision and recall fraction of patterns of vulnerabilities in OSS applications, we used the following two equations from [18].

$$\text{Precision} = \frac{\text{Number of patterns retrieved from the training set that also exist in the test set}}{\text{Total number of patterns retrieved from the training set}} \quad (3)$$

$$\text{Recall} = \frac{\text{Number of patterns retrieved from the training set that also exist in the test set}}{\text{Total number of relevant patterns in the test set}} \quad (4)$$

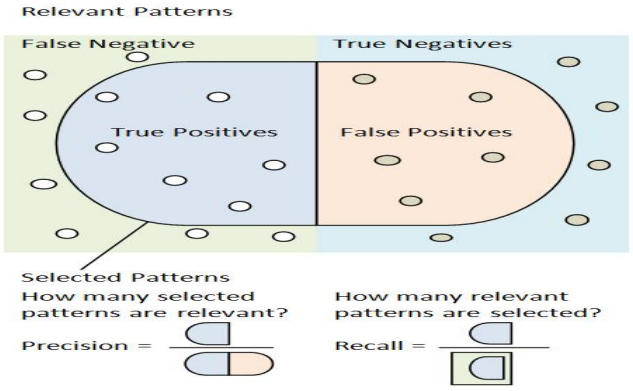

FIG. 2. CONCEPT OF PRECISION AND RECALL

4. Results and Analysis

As discussed earlier and shown in Table 1, a total number of 88,477 vulnerabilities were found in all software applications from January 2006 to October 2016. Through these vulnerabilities, it was also be noted that there were 8749 vulnerabilities of whose types were not declared and, therefore, we excluded them in our analysis.

The remaining vulnerabilities (79,728) are analyzed in the next subsections:

4.1 Analysis of the Trend of Software Vulnerability Rising or Declining over a Period of Time

The CVE and NVD repositories categorized the vulnerabilities in thirteen major types which are presented in Table 3. At this stage, once again we used the Equation (1) for the analysis of likelihood of these various types of vulnerabilities extracted from 2006-2016. In Table 3, "V" represents these various types of vulnerabilities such that V1 means Denial of Services, V2 means Code Execution and so forth. Table 3, also gives the detail about the frequency of these vulnerabilities across each and every year. This was helpful for us to determine the pre-dominant type of vulnerability occurred over a period of time. Through this data, we also concluded that if a vulnerability which occurs repeatedly across many years without any significant change may enhance the chance of attack from the attackers to damage a software application. For example, the frequency of Denial of services (V1) was 10% in 2006 which gradually rises in the next years and reached to 25% in 2016. Typically, we found that V1 was the most critical vulnerability type faced in all software applications.

Table 3. Likelihood of Vulnerabilities (in percentage) from 2006-2016

No.	Vulnerability Types	Likelihood/Percentage of each Vulnerability by Year										
		2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
V1	Denial of Services (DoS)	10	12	10	11	14	20	20	23	17	21	25
V2	Code Execution	31	27	24	23	22	22	21	19	16	21	18
V3	Overflow	8	10	8	7	9	13	12	14	9	12	16
V4	Memory Corruption	1	1	1	2	4	6	6	6	4	9	9
V5	SQL Injection	11	7	12	10	7	5	3	3	3	2	1
V6	Cross Side Scripting (XSS)	15	9	9	9	8	8	11	10	12	9	5
V7	Directory Traversal	4	4	4	3	4	2	2	2	2	2	1
V8	Http Response Splitting	0.1	0.1	0.08	0.1	0.1	0.1	0.2	0.1	0.1	0.1	0.2
V9	Bypass something	3	3	3	4	3	3	5	6	5	7	6
V10	Gain Information	3	3	3	3	4	7	5	8	22	9	10
V11	Gain Privileges	2	3	3	3	3	3	4	4	3	4	7
V12	Cross site request forgery (CSRF)	0.2	1	1	1	1	1	2	2	3	3	1
V13	File Inclusion	10	7	2	1	1	0.2	0.2	0.02	0.02	0.08	0.02

This data provided answer to our RQ1: that how the frequency of various vulnerabilities were rising or declining over a period of time. **Fig 3**, shows the frequency for various types of vulnerabilities over a period of time for this data.

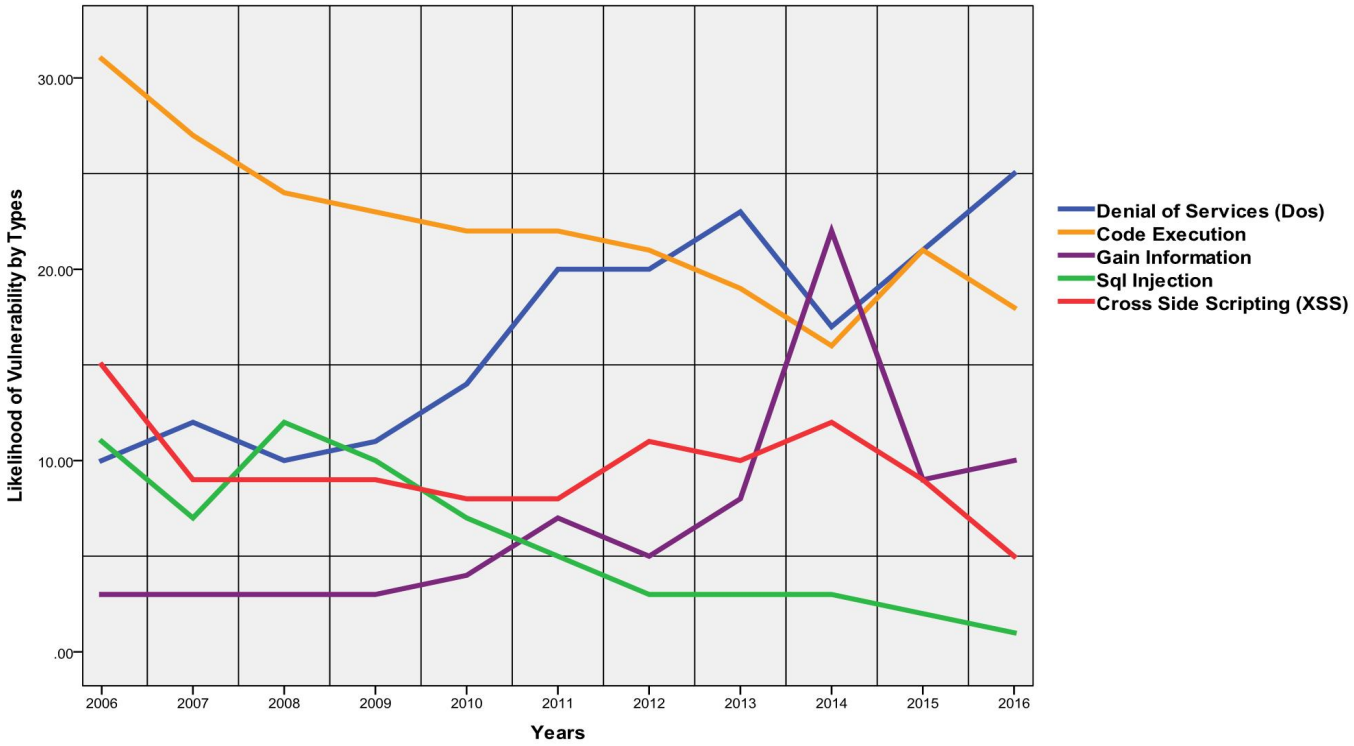

FIG 3. Selected Vulnerabilities and their Likelihood over a Period of Time

We used the Chi-Square from [19] to further analyze and to find out any significant difference among the various types of vulnerabilities. The Table 4 data presents such difference across each and every year. We also used Linear-by-Linear association Chi-Square test for the identification of statistically significant difference among the various vulnerabilities on the bases of each year.

To find out any significant difference amongst these vulnerabilities, we examined the following two hypotheses:

- Null Hypothesis (H_0): To expose the vulnerabilities, there exists no significant difference across each year for a particular vulnerability.

To conclude the above discussion for RQ1, we can say that there exists sufficient proofs for the Alternative Hypothesis (H_1): To expose the vulnerabilities, there exists a significant difference across each year for a particular vulnerability.

We considered H_0 for the case when the value of "p" surpassed 0.05 for a specific vulnerability or H_1 otherwise. Table 4 shows that there exists astounding significant differences amongst maximum vulnerabilities types across each year. According to Table IV, ten vulnerabilities V1, V2, V3, V4, V5, V7, V9, V10, V11 and V13 followed H_1, which means that they were not constant over a period of time. The remaining three vulnerabilities V6, V8 and V12 were found constant at time.

likelihood of software vulnerabilities changed drastically with the passage of time, regardless of their rise and fall per year.

Table 4. Significant Difference of Vulnerabilities on the Base of Years

No.	Vulnerability Types	Chi-Square Test (Linear-by-Linear Associations) α= 0.05		
		X^2	Df	P
V1	Denial of Services (DoS)	7.938	1	0.005
V2	Code Execution	7.793	1	0.005
V3	Overflow	5.115	1	0.024
V4	Memory Corruption	8.298	1	0.004
V5	Sql Injection	8.143	1	0.004
V6	Cross Side Scripting (XSS)	1.820	1	0.177
V7	Directory Traversal	8.127	1	0.004
V8	Http Response Splitting	0.094	1	0.759
V9	Bypass something	7.456	1	0.006
V10	Gain Information	4.640	1	0.031
V11	Gain Privileges	5.565	1	0.018
V12	Cross site request forgery (CSRF)	2.059	1	0.151
V13	File Inclusion	6.694	1	0.010

4.2 Analysis of the Trend of OSS Applications Vulnerabilities Rising or Declining over a period of Time

As discussed earlier in Section 3.1, where we analyzed the frequency for the likelihood of all software applications vulnerabilities across each year, still does not make public the likelihood inside individual software applications. In this section, we analyze the trend of likelihood of vulnerabilities rising or declining within the fifteen popular OSS applications across each year. After using Equation (2) for this analysis, we found some OSS applications with more vulnerabilities than other OSS applications. In Table 5, "OSS" represents an open source software like OSS1 means Mozilla Firefox, OSS2 means Google Chrome and so forth. Such table presents the frequency of likelihood of OSS applications vulnerabilities across each and every year. We analyzed the frequency of vulnerabilities of various OSS application and this was helpful for us in determining the major types of vulnerabilities occurred over a period of time in a specific OSS. Table V results show that, comparing to other open source software applications, the Mozilla Firefox (OSS1) has faced the major types of vulnerabilities during 2006-2009.

Table 5. Likelihood (in percentage) of some popular OSS Applications sorted from 2006-2016

No.	Open Source Software's	Likelihood/Percentage of each Vulnerability by Year										
		2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
OSS1	Mozilla Firefox	1.06	0.8	0.9	1.2	1.2	1.5	2.3	2.2	1	1.8	2
OSS2	Google Chrome	0	0	0.02	0.4	1.8	4	3	2.5	1.3	1.8	2.4
OSS3	Linux Kernel	1	0.6	0.7	1	1.4	1.4	1.5	3	1.3	0.8	2.3
OSS4	Mozilla Thunderbird	0.7	0.2	0.4	0.4	0.7	1	2	1.7	0.6	0.3	0.2
OSS5	SeaMonkey	0.7	0.4	0.6	0.5	1	1	2	1.5	0.5	0.1	0
OSS6	MYSQL	0.2	0.07	0.06	0.06	0.08	0.3	1.7	1.7	1	1	1
OSS7	Moodle	0.2	0.03	0.06	0.07	0.1	0.02	1.3	0.7	0.5	0.3	0.5
OSS8	Webkit	0	0.02	0.01	0.02	1.2	2	0.1	0	0.03	0	0.2
OSS9	Wordpress	0.2	0.4	0.3	0.1	0.01	0.2	0.3	0.3	0.3	0.1	0.3
OSS10	Adobe Air	0	0.01	0.01	0	0.1	0.5	0.7	0.7	1	0.1	0
OSS11	ClamAV	0.1	0.2	0.2	0.1	0.1	0.05	0.07	0.03	0	0.1	0.05
OSS12	Joomla!	0.01	0	0	0.04	0.05	0.2	0.5	0.2	0.1	0.1	0.02
OSS13	SAMBA	0.02	0.1	0.02	0.06	0.1	0.1	0.6	0.1	0.07	0.02	0.2
OSS14	BIND	0.05	0.06	0.03	0.04	0.08	0.1	0.1	0.1	0.03	0.06	0.2
OSS15	VLC Media Player	0	0.05	0.1	0.02	0.04	0.2	0.1	0.1	0.03	0.05	0.02

For the betterment of analysis, we further categorized the likelihood (percentage) of some popular OSS applications into three categories as shown in Table 6. In "Category 0-10" we put those OSS applications whose vulnerability percentage lie between 0-10 in a particular year. In "Category 11-20" we put those OSS applications whose vulnerability percentage lie between 11-20 in a particular year. Similarly, in "Category 21-30" we put those OSS applications whose vulnerability percentage lie between 21-30 in a particular year.

TABLE 6. Categorization of Likelihood of some Popular OSS Applications

Categories	Likelihood (%) Value of OSS Applications
0-10	0, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.1, 0.2
11-20	0.3, 04, 0.5, 0.6, 0.7, 0.8, 1, 1.2, 1.3, 1.4
21-30	1.5, 1.6, 1.7, 1.8, 2, 2.2, 2.3, 2.5, 3, 4

Fig 4, shows the frequency for various types of vulnerabilities in some major OSS applications disclosed each year. It can be seen that for each OSS application, there exists ups and downs for various vulnerabilities as compare to others.

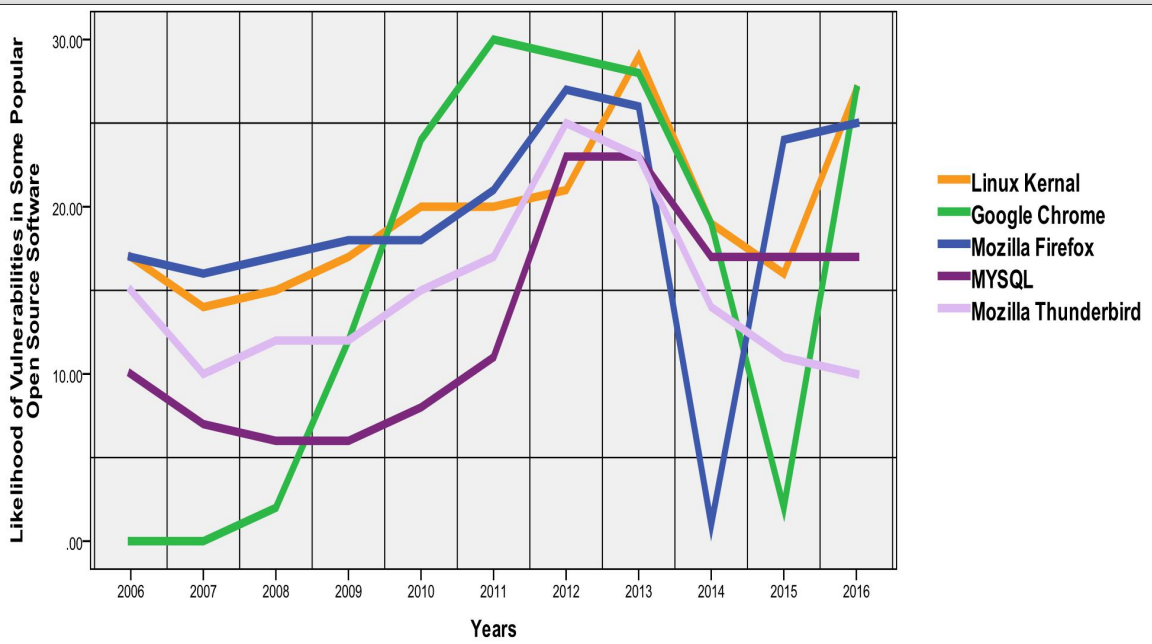

FIG. 4. Selected Popular OSS Applications and their Likelihood of Vulnerabilities over a Period of Time

We used the Chi-Square from [19] to further analyze and to find out any significant difference between the likelihood of an OSS application. The Table 7 presents the significant difference of vulnerabilities in all OSS application across each year. The Linear-by-Linear association Chi-Square test was used for the identification of statistically significant vulnerability difference among the various OSS applications on the basis of each year

.Table 7. Significant difference of some Popular OSS Applications Vulnerabilities on the base of years

No.	OSS Applications	Chi-Square Test (Linear-by-Linear Associations) α= 0.05		
		X^2	Df	P
OSS1	Mozilla Firefox	121.893	1	0.000
OSS2	Google Chrome	50.491	1	0.000
OSS3	Linux Kernel	78.186	1	0.000
OSS4	Mozilla Thunderbird	.867	1	0.352
OSS5	SeaMonkey	23.862	1	0.000
OSS6	MYSQL	101.154	1	0.000
OSS7	Moodle	54.733	1	0.000
OSS8	Webkit	2.537	1	0.111
OSS9	Wordpress	1.355	1	0.244
OSS10	Adobe Air	25.787	1	0.000
OSS11	ClamAV	69.333	1	0.000
OSS12	Joomla!	44.850	1	0.000
OSS13	SAMBA	21.582	1	0.000
OSS14	BIND	82.860	1	0.000
OSS15	VLC Media Player	2.604	1	0.107

To find out any significant difference among the likelihood of vulnerabilities in OSS applications, we examined the following two hypotheses:

- Null Hypothesis (H_0): To expose the likelihood of vulnerabilities in OSS applications, there exists no significant difference across each year for a particular vulnerability in OSS application.
- Alternative Hypothesis (H_1): To expose the likelihood of vulnerabilities in OSS applications, there exists a significant difference across each year for a particular vulnerability in OSS application.

We considered H_0 for the case when the value of "p" surpassed 0.05 for a specific vulnerability or H_1 otherwise.

Table 7 shows that there exists astounding significant differences amongst maximum OSS applications for the likelihood of vulnerabilities across each year.

According to Table 7, eleven OSS applications such as OSS1, OSS2, OSS3, OSS5, OSS6, OSS7, OSS10, OSS11, OSS12, OSS13 and OSS14 followed H_1, which means that the likelihood of vulnerabilities in these OSS applications was not constant over a period of time. The remaining three OSS applications likewise OSS4, OSS8 and OSS15 were found constant at time.

4.3 Analysis of common Patterns of Vulnerabilities across different OSS Applications

The Apriori Algorithm was used for the extraction of N-Itemset pattern of vulnerabilities in all OSS applications. The detail about the automatically extraction of sequential patterns are given in section 2.3. Further, we also extracted different length N-Itmesets of vulnerabilities for each OSS application and counted the frequency i.e. the number of occurrence of each one. Then we added the frequency of each individual N-Itemset, and determined the frequency of N-Itemset of vulnerabilities across all OSS applications. Table 8 shows the top 7 most frequently occurred 4-Itemset pattern of vulnerabilities in the descending order across top 15 popular OSS applications from 2006 to 2016.

In the following tables "V" represents vulnerability.

Table 8. Frequent 4-Itemset pattern of vulnerabilities in OSS Applications in the period of 2006-2016

4-Itemset Pattern	Total	Occurrences of 4-Itemset Pattern of Vulnerabilities in the selected Open Source Software's														
		OSS1	OSS2	OSS3	OSS4	OSS5	OSS6	OSS7	OSS8	OSS9	OSS10	OSS11	OSS12	OSS13	OSS14	OSS15
V1, V2, V3, V4	328	68	10	3	38	29	0	0	114	0	63	0	0	1	0	2
V1, V3, V11, V4	14	1	0	13	0	0	0	0	0	0	0	0	0	0	0	0
V1, V3, V9, V10	3	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
V1, V2, V3, V10	2	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
V1, V2, V4, V10	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
V1, V11, V4, V10	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
V1, V3, V4, V10	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

The table data proves that the first 4-Itemset pattern i.e. "V1, V2, V3 and V4" has occurred maximum time than others with the score of 328 in all OSS applications.

The same pattern has also occurred maximum time with the score of 114 in OSS8 (Webkit) application and through many other applications such as OSS1, OSS10, OSS4, OSS5 and OSS2 applications. The second 4-Itemset pattern of vulnerabilities which occurred many times was "V1, V3, V11 and V4" with the total score of 114. Such pattern occurred 13 times only in OSS3 (Linux Kernal), but not occurred so many times in other applications. The occurrence of other remaining 4-Itemset pattern of vulnerabilities was very low in all OSS applications**.** Fig. 5 plots the 4-Itemset pattern of vulnerabilities faced to all OSS applications. Such Fig. 5 clearly depicts that the graph of "V1, V2, V3 and V4" pattern moves with maximum value.

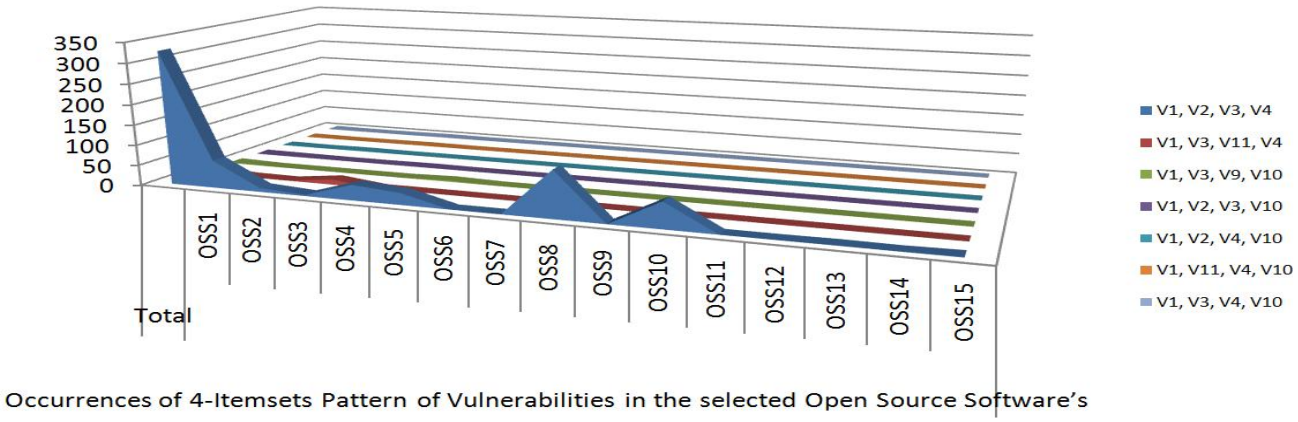

FIG. 5. Occurrences of 4-Itemset Pattern of Vulnerabilities in the selected OSS Applications

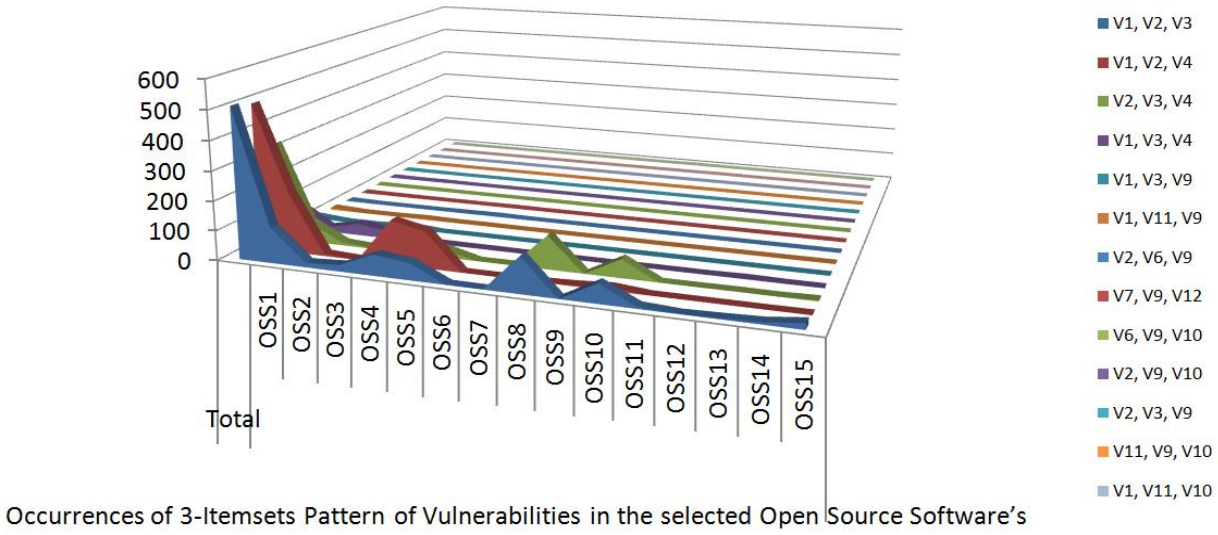

FIG. 6. Occurrences of 3-Itemset Pattern of Vulnerabilities in the selected OSS Applications

The same process was repeated for frequently 3-Itemset pattern of vulnerabilities by ordering them in the descending order of their maximum occurrence in an OSS application. At this stage, the first 3-Itemset pattern "V1, V2, and V3" has occurred maximum time with the total score of 511 as shown in Table IX. Which only occurred 119 and 122 times in OSS1 (Mozilla Firefox) and OSS8 (Webkit) respectively. The further occurrences of such pattern with the highest score were 71 times in OSS4, 57 times in OSS5, 65 times in OSS10 and 21 times in OSS15.

Fig. 6 plots the 3-Itemset pattern of vulnerabilities faced to all OSS applications. Such Fig. 6 clearly depicts that the graph of "V1, V2 and V3" pattern moves with maximum value.

Similarly, the same process was repeated once again for 2-Itemset pattern of vulnerabilities in all OSS application as shown in Table 9. Table 9 shows the list of top 30 most frequently occurred 2-Itemset pattern of vulnerabilities in the top 15 popular OSS applications from 2006 to 2016. The 2-Itemset pattern "V1 and V2" was found maximum with 1187 times occurrence in all OSS applications. The highest occurrences of such pattern found only in OSS1 (Mozilla Firefox) application was 376 times, shows that Mozilla Firefox faced the highest probability for the occurrences of this pattern. The

next highest occurrences of "V1 and V2" pattern of vulnerability were 252, 211, 161 and 76 in OSS4, OSS5, OSS8 and OSS10 applications, respectively. The second maximum 2-Itemset pattern of vulnerability was "V2 and V3", occurred 764 in all times in all OSS applications.

Table 9. Frequent 3-Itemset pattern of vulnerabilities in OSS Applications in the period of 2006-2016

3-Grams Pattern	Total	Occurrences of 3-Itemset Pattern of Vulnerabilities in the selected Open Source Software's														
		OSS1	OSS2	OSS3	OSS4	OSS5	OSS6	OSS7	OSS8	OSS9	OSS10	OSS11	OSS12	OSS13	OSS14	OSS15
V1, V2, V3	511	119	16	22	71	57	1	0	122	0	65	7	0	3	7	21
V1, V2, V4	495	201	7	1	148	123	0	0	0	3	11	0	0	0	0	1
V2, V3, V4	332	69	10	3	39	30	0	0	114	0	63	0	0	1	0	3
V1, V3, V4	88	17	41	24	2	1	1	0	0	0	0	0	0	2	0	0
V1, V3, V9	9	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0
V1, V11, V9	7	1	1	3	1	1	0	0	0	0	0	0	0	0	0	0
V2, V6, V9	3	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0
V7, V9, V12	2	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
V6, V9, V10	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
V2, V9, V10	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
V2, V3, V9	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

V11, V9, V10	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
V1, V11, V10	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
V1, V6, V10	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
V2, V5, V6	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

The highest occurrences of such pattern for Mozilla Firefox was 187. For other applications such as OSS8, OSS10, OSS4 and OSS5, the number of occurrences of such pattern were 128, 122,111 and 98, respectively.

The third most frequent 2-Itemset pattern of vulnerability was "V2 and V4", occurred 499 times in all OSS applications. This pattern showed the highest number of occurrence such that 201 in OSS1 (Mozilla Firefox), 149 in OSS4 (Mozilla Thunderbird) and 125 in OSS5 (SeaMonkey) application.

The fourth most frequent 2-Itemset pattern of vulnerabilities was "V1 and V3" with a total of 454 occurrences in all OSS applications. The further highest occurrences of such pattern were such that 210 in OSS2 (Google Chrome), 141 in OSS3 (Linux Kernel) and 55 in OSS1 (Mozilla Firefox) application.

Some other 2-Itemset pattern of vulnerabilities such that "V1 and V4", "V3 and V11", "V9 and V10", "V6 and V9", "V1 and V11", "V2 and V9", "V3 and V10" and "V2 and V5" showed the average number of occurrences in all OSS applications as shown in Table 10. The rest of 2-Itemset pattern of vulnerabilities showed less number of occurrence in all OSS applications.

Table 9. Frequent 2-Itemset pattern of vulnerabilities in OSS Applications in the period of 2006-2016

2-Grams Pattern	Total	Occurrences of 2-Itemset Pattern of Vulnerabilities in the selected Open Source Software's														
		OSS1	OSS2	OSS3	OSS4	OSS5	OSS6	OSS7	OSS8	OSS9	OSS10	OSS11	OSS12	OSS13	OSS14	OSS15
V1, V2	1187	376	35	30	252	211	1	0	161	0	76	7	0	4	7	27
V2, V3	764	187	25	28	111	98	4	0	124	0	122	13	0	7	13	32
V2, V4	499	202	7	1	149	125	0	0	3	0	11	0	0	0	0	1
V1, V3	454	55	210	141	16	11	4	0	1	0	1	3	0	5	3	4
V1,	96	19	20	28	12	11	1	0	1	0	0	2	0	0	2	0

V4																
V3, V11	60	5	0	52	3	0	0	0	0	0	0	0	0	0	0	0
V9, V10	59	23	4	1	13	13	0	0	1	1	3	0	0	0	0	0
V6, V9	53	23	5	0	7	13	0	1	2	2	0	0	0	0	0	0
V1, V11	53	1	0	51	1	0	0	0	0	0	0	0	0	0	0	0
V2, V9	41	16	3	0	9	10	1	0	0	0	1	0	1	0	0	0
V3, V10	38	13	0	10	9	3	0	0	1	0	1	0	0	1	0	0
V2, V5	36	0	0	0	0	0	3	7	0	14	0	0	12	0	0	0
V1, V10	19	2	1	13	1	1	0	0	0	0	0	0	0	1	0	0
V2, V6	19	6	2	0	4	4	0	0	0	0	3	0	0	0	0	0
V9, V12	10	4	0	0	1	3	0	0	0	2	0	0	0	0	0	0
V6, V10	9	1	6	0	0	1	0	0	0	1	0	0	0	0	0	0
V11, V9	7	3	0	2	1	1	0	0	0	0	0	0	0	0	0	0
V7, V9	7	2	1	0	1	2	1	0	0	0	0	0	0	0	0	0
V11, V10	7	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0
V2, V10	3	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0
V1, V6	3	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0
V1, V9	3	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0
V6, V12	3	0	0	0	0	0	0	1	0	2	0	0	0	0	0	0
V2, V11	2	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
V2, V13	2	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
V1, V12	2	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
V2,	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

V7																
V11, V12	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
V1, V5	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
V5, V10	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

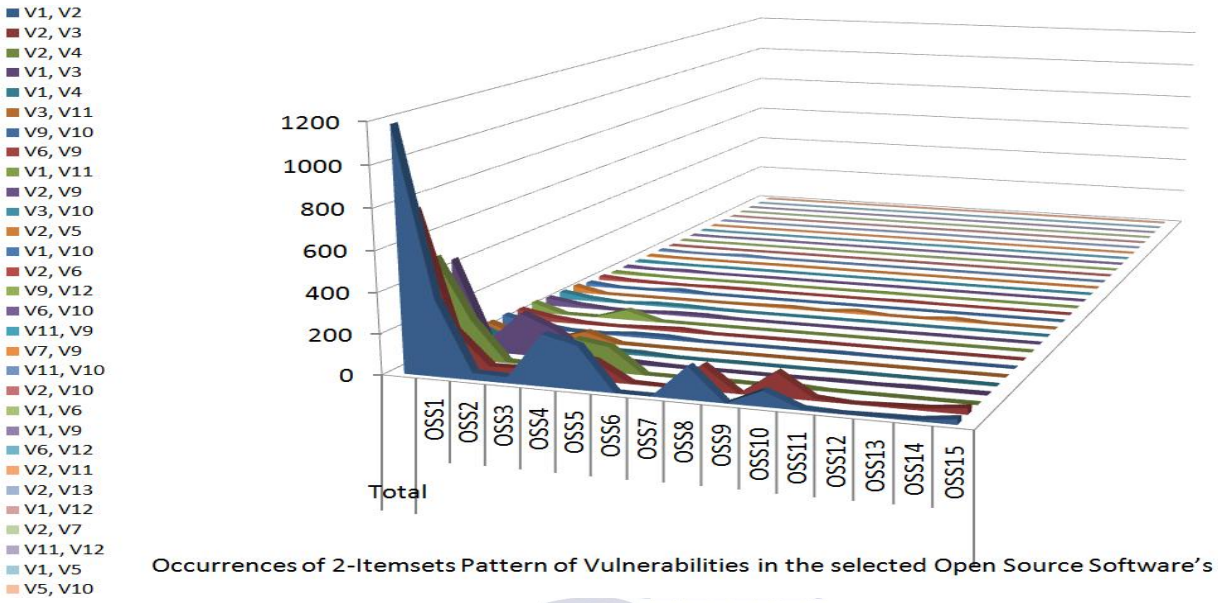

FIG. 7.Occurrences of 3-Itemset Pattern of Vulnerabilities in the selected OSS Applications

Fig. 7 plots the 2-Itemset pattern of vulnerabilities faced to all OSS applications. Such Figure clearly depicts that the graph of "V1 and V2" pattern moves with maximum value.

Fig. 7 also shows that such pattern rises dominantly in OSS1, OSS4, OSS5 and OSS8 applications comparatively to other OSS applications. Similarly, the graph of "V2 and V3" has astonishing increase in OSS1, OSS4, OSS5, OSS8 and OSS10 applications comparatively to other OSS applications. Upon concluding our discussion, we can say that "Denial of Service, Code Execution", "Code Execution and Overflow", "Code Execution, Memory Corruption" and "Denial of Service and Overflow" were the most frequently 2-Itemset patterns of vulnerabilities in our results.

4.4 Analysis of Top most types of Vulnerabilities across different OSS Applications

In the previous sections, we discussed different length of patterns of vulnerabilities across all OSS applications. However, some OSS applications may have not faced the 4, 3 and 2-Itemset pattern of vulnerabilities but still faced a single type of vulnerability from successive years. In this section, we identified that which type of vulnerability, not a vulnerability pattern, has the maximum occurrence in a specific OSS application. We found out a total of 10,058 vulnerabilities faced to the selected 15 OSS applications from 2006-2016 as shown in Table XI. We found 'V1' .i.e. Denial of Services with the highest frequency of 3764 across all OSS applications. This was the unique vulnerability faced to almost each OSS applications. The maximum occurrence of such vulnerability was 973 in a single OSS3 (Linux Kernel) application. The same vulnerability reoccurred in almost all OSS applications.

Another predominant type of vulnerability was V2 (Code Execution) which occurred 1944 times across all OSS applications. Such type of vulnerability almost occurred in all these OSS applications but one step down than VI. Table 11 shows that the maximum occurrences of V2 for OSS1 (Mozilla Firefox) was 623. The other occurrences of such vulnerability were 393, 357, 167, 143 and 42 in OSS4, OSS5, OSS8, OSS10 and OSS15 applications, respectively. Importantly, it can also be noticed from Table 11 results that V3 (Overflow) was found with the highest ranked total occurrences of 1288 across all OSS applications. Also, its highest occurrence for OSS1 (Mozilla Firefox) was found 253. The other occurrences of V3 were 238 and 205 in OSS2 (Google Chrome) and OSS3 (Linux Kernel) applications, respectively.

Table 9. List of Vulnerabilities by Types across all the OSS Applications

Open source Software's	Occurrences of Vulnerabilities in the selected Open Source Software's													Total
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
OSS1	640	623	253	309	0	96	9	1	135	98	31	8	0	2203
OSS2	837	71	238	78	0	40	8	0	129	60	5	3	0	1469
OSS3	973	40	205	78	0	0	2	0	62	207	158	0	0	1725
OSS4	353	393	135	203	0	49	2	1	60	45	21	3	0	1265
OSS5	307	357	111	168	0	60	5	1	71	47	10	5	0	1142
OSS6	42	9	10	2	6	1	1	0	10	3	3	0	0	87
OSS7	97	11	0	0	9	61	1	1	46	67	4	18	1	316
OSS8	186	167	126	118	0	14	1	0	11	14	0	0	0	637
OSS9	52	26	0	0	15	78	7	0	26	25	1	12	1	243
OSS10	88	143	123	74	0	5	0	0	26	9	1	2	0	471
OSS11	50	17	19	2	0	0	1	0	11	1	0	0	0	101
OSS12	22	13	0	0	13	24	0	0	6	18	0	1	0	97
OSS13	27	15	13	3	0	1	1	0	6	6	3	2	0	77
OSS14	49	17	19	2	0	0	1	0	11	1	0	0	0	100
OSS15	41	42	36	4	0	1	0	0	1	0	0	0	0	125
Total	3764	1944	1288	1041	43	430	39	4	611	601	237	54	2	10058

V4 (Memory Corruption) was the next astonishing vulnerability faced to the selected OSS applications with total occurrences of 1041 across all OSS applications. Its highest occurrences values were 309, 203, 168, 118 and 74 in OSS1, OSS4, OSS5 and OSS8 applications, respectively.

It has been identified that V6, V9 and V10 occurred almost maximum time in the selected OSS applications. There were some vulnerabilities likewise V5, V7, V8, V11, V12 and V13, whose occurrences were seen very less often in all OSS applications.

Fig. 8 plots the entire vulnerabilities faced by all OSS applications. The graph of V1 has astonishing increase in OSS1, OSS2, OSS3, OSS4, OSS5 and OSS8 applications comparatively to other OSS applications. Likewise, the graph of V2 (Code Execution) has unforeseen raise in OSS1, OSS4, OSS5, OSS8 and OSS10 applications comparatively to other OSS applications. In order to provide answer to RQ 4, we summarize our discussion and say that "Denial of Services", "Code Execution", "Overflow", "Memory Corruption", "Bypass Something" and "Gain Information" were the most frequently types of vulnerabilities faced to the OSS applications.

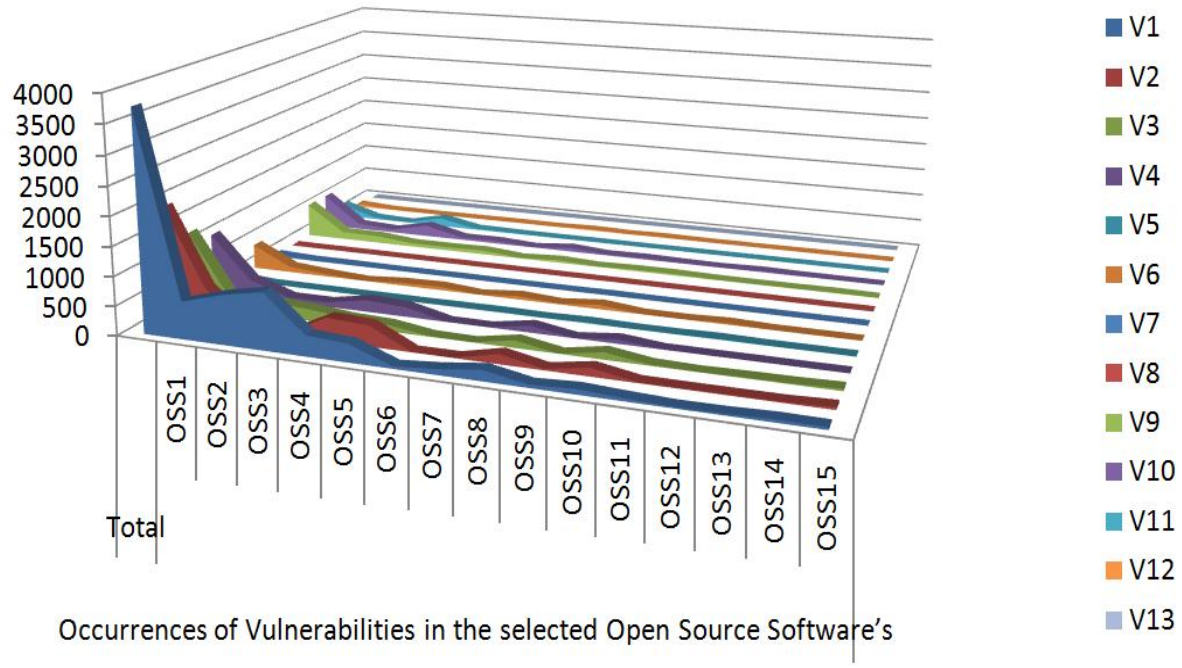

FIG. 8. Frequent 1-Gram Pattern across all the selected OSSs

4.5 Analysis of Prediction of Recurring Pattern of Vulnerability in OSS Applications

To predict the next possible recur vulnerability or vulnerability pattern in future, we are going to highlight the use of N-Itemset pattern of historically vulnerabilities in some OSS applications in this section. From the experimental results, the same predictions were determined for some popular OSS applications such as, Mozilla Firefox, Google Chrome, Linux Kernel, Mozilla Thunderbird and SeaMonkey. Typically, the selection was based on the maximum occurrences of vulnerabilities to these OSS applications.

To highlight the process for clarity, suppose we considered an OSS application with the denial of service vulnerability recently been exposed in it and perform the historical collection of 2-Itemset pattern of vulnerabilities. Similarly, suppose we considered an OSS application recently been discovered with the denial of services along with code execution vulnerabilities, and performed the historical collection of 3-Itemset pattern of vulnerabilities. We searched for all those 3-Itemset pattern whose first and second vulnerabilities were denial of service and memory execution, and used the third following vulnerability to predict the next possible recur vulnerability to that OSS application. We prolonged this tendency for upper N-Itemset by discovering the match of N-1 vulnerabilities in a collection of N-Itemset patterns to guess the nth possible recur vulnerability in future to a particular OSS application.

We carried out an information retrieval experiment to measure the prediction of N-Itemset pattern of the next possible vulnerability to an OSS application. Then we divided our data sets of N-Itemset pattern of vulnerabilities into training and testing sets as explained in Section 2.4. In order to bring accuracy in the results, we performed our experiment in two phases. For the first phase, the training set data was taken from 2006-2011 and from 2012-2016. For the second phase, the training set data was taken from 2013 to 2014 whereas the testing set data was taken from 2015-2016.The main objective behind the two phases of the experiment was such that to test the predication power with

older and innovative data. We carried out these phases of experiment for single vulnerability and three types of N-Itemset i.e. 2-Itemset, 3-Itemset and 4-Itemset pattern of vulnerabilities in the selected five OSS applications.

We used for evaluation the N-1 vulnerability order from N-Itemset in the test set data to search for the next possible vulnerability from the N-Itemset in the training set data. We then concluded that how many N-Itemset patterns that initiate N-1 vulnerability in the test set data are really there in the retrieved list of patterns in the training set data, and counting it by means of the precision and recall procedures (Section 2.4). Tables12-13 shows the results of the experiment by using various combinations of training and testing sets data. Table 12 presents the precision and recall for the prediction of the next possible vulnerability or vulnerabilities pattern in Mozilla Firefox, Google Chrome, Linux Kernel, Mozilla Thunderbird and SeaMonkey OSS applications. It means that the vulnerabilities in these OSS applications have been predicted using prior vulnerabilities in these applications. In order to measure the predication of possible vulnerability or vulnerability pattern recur to these applications, the data set from 2006-2011 was used as a training set whereas the data set from 2012 to 2016 was used as a testing set as shown in Table 12. Next, a search for a single possible vulnerability, 2-Itemset, 3-Itemset and 4-Itemset pattern of vulnerabilities recur in future was performed to these 5 OSS applications. From Table 12, we can see that a single vulnerability like V1 or V6 in Mozilla Firefox has almost the same precision and recall in both the training and testing data sets. The search for 2-Itemset (say "V1 and V3", "V1 and V2",) and 3-Itemset (say V1, V2 and V4", and "V1, V2 and V3) pattern of vulnerabilities, gives almost the same precision and recall in both data sets. In such table, we can see 50% precision and 5% recall for V1, 50% precision and 10% recall for V6, 100% precision and 33% recall for "V1 and V3", 68% precision and 10% recall for "V1 and V2",26% precision and 90% recall for "V1, V2 and V4", 80% precision and 20% recall for "V1, V2, and V3" which tells about the recur of them only in Mozilla Firefox.

Similarly, in Google chrome, the precision of vulnerability V1 was 54% whereas its recall was 100%, gives recur of V1 in future with these values. Similar was the case for the 2-Itemset pattern of vulnerabilities "V1 and V3", with 64% precision and 11% recall, gives the detail about the future recur of such pattern of vulnerabilities to this application. In this way, we evaluated the whole table data and finally plotted the results as shown in Fig 9. Here, we can see that the precision and recall values goes high when searched for the subsequent vulnerability, based on a single vulnerability or 2-Itemset pattern of vulnerability, performed. However, these values reduces with higher N-Itemset when used to look up for the next possible vulnerability. This was the reason that the number of sequential (ordered) collections of vulnerabilities gives similar results to that of pattern searched.

Table 12. Prediction of Recurring Pattern of Vulnerability in OSS Applications, with Training-Set "2006-2011" and Testing-Set "2012-2016"

Open Source Software's	Vulnerabilities Pattern	Training Set (2006-2011)		Testing Set (2012-2016)	
		Precision	Recall	Precision	Recall
Mozilla Firefox	V1	0.47	0.30	0.50	0.05
	V6	0.57	0.21	0.50	0.10
	V1, V3	1	0.33	1	0.04
	V1, V2	0.60	0.26	0.68	0.10
	V1, V2, V4	0.28	0.61	0.26	0.90
	V1, V2, V3	1	0.13	0.80	0.20

Google Chrome	V1	0.49	1	0.54	1
	V1, V3	0.54	0.21	0.64	0.11
Linux Kernel	V1	0.50	1	0.43	0.96
	V10	0.8	0.13	0.51	0.27
	V1, V3	0.8	0.17	0.84	0.10
	V1, V11	1	0.10	1	0.50
Mozilla Thunderbird	V2	0.54	0.61	0.48	0.43
	V6	0.40	0.20	0.62	0.37
	V2, V3	0.68	0.33	0.68	0.10
	V1, V2, V4	0.34	1	0.38	1
	V1, V2, V3, V4	0.68	0.14	0.59	0.30
SeaMonkey	V9	0.83	0.36	0.68	0.24
	V1	0.60	0.25	1	0.20
	V6	0.55	0.30	0.64	0.53
	V2, V3	0.71	0.24	1.	0.11
	V1, V2, V4	0.3	0.69	0.39	0.93

To predict the next possible vulnerability in the top 5 popular OSS applications, once again we performed the similar experimentation process for the single vulnerability and for various N-Itemset pattern of vulnerabilities. This time, we considered the data from 2013-2014 as a training set, whereas, the data from 2015-2016 as a testing set as shown in Table 13. The reason behind such setup was to know 'what type of vulnerability' mostly reoccurring from the last 4 years.

Moreover, training older data has reduced the recall of predicting innovative patterns nevertheless can still predict the incidence of identified vulnerabilities with high precision.

Table 13 shows that a single vulnerability V2 has maximum occurrence of 30% precision and 56% recall in next possible vulnerability which may recur to Mozilla Firefox. Similarly, the search for the 2-Itemset pattern of vulnerabilities "V1 and V2" showed 30% precision and 56% recall for the next possible recur 2-Itemset pattern of vulnerabilities in Mozilla Firefox.

The search for the 3-Itemset pattern of vulnerabilities "V1 V2 and V4" showed 30% precision and 84% recall for the next possible recur 3-Itemset pattern of vulnerabilities for this application. The only 4-Itemset pattern "V, V2, V3 and V4" which showed 30% precision with 100% recall for the next possible recur 4-Itemset pattern of vulnerabilities in Mozilla Firefox. In Google Chrome, the vulnerability V1 noted with 50% precision and 100% recall which tells recur of V1 in future to this application. Other vulnerabilities likewise V1 and V9 in Linux Kernel noted with precision of 48% and 100% whereas the recall values for them noted with 100% and 16%, respectively. In Mozilla Thunderbird, the V1 was noted with astonishing values of 100% for both precision and recall which means that it may recur with this ratio in future. Similarly, the 4-Itemset pattern of vulnerability "V1, V2, V3 and V4" was noted with 100% values both for precision and recall, gives the 100% recur of such pattern in future to this application.

Likewise, the vulnerability "V2" and 3-Itemset pattern "V1, V2 and V4" were noted with 100% precision and recall values in SeaMonkey application.

Table 13. Prediction of Recurring Pattern of Vulnerability in OSS Applications, with Training-Set "2013-2014" and Testing-Set "2015-2016

Open Source Software's	Vulnerabilities Pattern	Training Set (2013-2014)		Testing Set (2015-2016)	
		Precision	Recall	Precision	Recall
Mozilla Firefox	V2	1	0.25	0.30	0.56
	V9	1	0.07	0.42	0.28
	V1, V2	1	0.25	0.30	0.56
	V1, V3	1	0.33	0.25	0.39
	V2, V3	1	0.09	0.50	0.17
	V1, V2, V4	0.3	1	0.30	0.84
	V1, V2, V3	0.75	0.20	0.60	0.20
	V1, V2, V3, V4	1	0.25	0.30	0.56
Google Chrome	V1	0.58	1	0.50	1
	V9	1	0.10	0.61	0.37
	V1 , V3	0.58	0.28	0.56	0.28
Linux Kernel	V1	0.44	0.92	0.48	1
	V9	1	0.07	1	0.16
	V10	0.48	0.44	0.40	0.24
Mozilla Thunderbird	V11	1	0.25	1	1
	V1,V2, V3	0.80	0.29	1	0.33
	V1, V2, V4	0.35	0.95	0.35	1
	V1, V2, V3, V4	1	0.11	1	1
SeaMonkey	V2	0.64	0.60	1	1
	V1, V2, V4	0.36	1	1	1

5. Limitation of the Study

In this section, threats of validity to the research are presented. The construct validity threat may possibly be present due to the selection of attributes from CWE, CVE and NVD. In order to tackle the definite research questions, the study extremely focuses on the vulnerability types and vulnerabilities patter in Open Source Software applications.

However, there exists a lot of attributes such as CVSS score report and distribution, CWE-Ids, Vendors, Software Versions, Vendor CVSS scores, Product CVSS scores and other various types of software's vulnerability information excluded from the study. Further, we plan to include them in future studies in the same field. Another threat towards the validity of the study is such that the datasets are publicly obtainable and could be mimic easily. More often, every vulnerability corresponds to a probable molest through an exploit but we were like tacit during the study which means that there exist a conclusion threat to validity.

These were the threats that may influence the statistical conclusions. However, there exists theoretical possibilities that molests exists for each vulnerability reported in the mentioned repositories, but the real exploits capacity does not exist for some vulnerabilities in these repositories. Though, this risk is diminished by the information that maximum vulnerabilities comprise probable exploits. However, we alleviated this threat by using standard statistical test like a Chi-Square test for these purposes.

The possible threat to internal validity might rise due to the need of available information concerning the vulnerabilities types. In such cases, we ignored these vulnerabilities

types, and it might have distorted the figures of our analysis. However, these cases are uncommon and do not have an effect on our results.

The threat to external validity may rise due to the quantity of vulnerabilities chosen from the CWE, CVE and NVD repositories. For analysis, we selected specific amount of 79,728 vulnerabilities from 2006 to 2016. However, the selected amount of vulnerabilities were chosen from well known public repositories.

6. Conclusion and Future Work

Nowadays, lot of innovative technologies developed and several security mechanisms were introduced, but still new software attacks are rising and enduring to spoil the security of software systems. So, in this paper, we mined the vulnerabilities and different patterns of vulnerabilities of OSS applications in order to assists the OSS vendor organizations to acquire decisions about vulnerabilities occurrence.

Table 1 shows the data we retrieved from CWE, CVE, and NVD archives concerning different types of vulnerabilities in software applications from 2006 to the 8th of October 2016. A total of 88,477 vulnerabilities were discovered across all software programmes. Furthermore, we discovered that there were 8749 vulnerabilities for which the vulnerability category was not specified, thus we excluded these vulnerabilities from our research. We further analyzed the remaining 79,728 vulnerabilities. The CVE and NVD repositories categories the vulnerabilities into thirteen major types as presented in Table 3. We used Equation (1) for the analysis of the likelihood of these different types of vulnerabilities form 2006-2016.According to Table 3, the Denial of services (V1) is the most critical type of vulnerability faced to all software applications according to our results. V2 (Code execution) vulnerability also creates critical problems to all software applications in all years, but it slightly falls down and stops at 18% in 2016. Some vulnerability such as, V5, V7, V8, V12 and V13 are almost falls each year. To finding some significance difference between the likelihood of the types of vulnerabilities, we use the Chi-Square test [44]. Table 4 presents the significance difference of all vulnerabilities across each year. In order to answer for RQ1, we determine that presently there are sufficient proofs that the likelihood of software vulnerabilities has changed drastically with the passage of time regardless of their rise and fall in each year.

In Section 4.1, we analyzed the trend of likelihood of vulnerabilities whether rising or declining within fifteen popular OSS applications across each year. We used Equation (2) for this analysis and also analyzed that some OSS applications could have more vulnerabilities than other OSS applications. From Table 5, we conclude that Mozilla Firefox (OSS1) has faced the most types of vulnerability in 2006, 2007, 2008 and 2009 by comparing to other OSS applications in these years. We further noted that Google Chrome has not faced any vulnerability in 2006 and 2007, but faced the most critical type of vulnerabilities in 2011 which scored the maximum vulnerabilities faced to all software applications according to our results. In 2016 Google Chrome had faced the most type of vulnerabilities as compared to other OSS applications. From overall results we noted that the top five OSS applications faced any type of vulnerability in every year. The other remaining OSS application also faced vulnerabilities across each but not so much as compare to the first five. Further, we also concluded from this section that the vulnerabilities faced to these applications rising from 2008 to 2013, but in 2014-2015 these application gained control on all vulnerabilities. In 2016, again all OSS applications faced huge among of vulnerabilities except Mozilla Thunderbird. We used Linear-by-Linear

association Chi-Square test for the identification of statistically significant vulnerability difference among the various OSS applications on the base of each year in Table 7. Table 7 shows that in our analysis maximum OSS applications have the significant differences of likelihood of vulnerabilities. These results shows that the rise and fall in the likelihood of OSS applications vulnerabilities in each year is presently by possibility and the software vulnerabilities are sustained to be exploited in related likelihood. In order to get conclusion from this section and answering for RQ2, we determine that presently is sufficient proofs that the likelihood of OSS applications vulnerabilities has changed drastically with the passage of time regardless of their rise and fall in each year.

In Section 4.3, we used Apriori Algorithm for the extraction of N-Itemsets pattern of vulnerabilities in all OSS applications. We extracted 7 different 4-Itemsets pattern of vulnerabilities. Overall, we can conclude that "Denial of Service, Code Execution, Overflow and Memory Corruption" is the most high 4-Itemsets frequent pattern of vulnerabilities façade to the selected OSS application. We also identified 15 frequently 3-Itemsets pattern of vulnerabilities and their maximum occurrence in all OSS applications. Generally, we can conclude that "Denial of Service, Code Execution, and Overflow", "Denial of Services, Code Execution and Memory Corruption" and "Code Execution, Overflow and Memory Corruption" are the most high 3-Itemsets frequent pattern of vulnerabilities in our results. Similarly, we also extracted 30 most frequently 2-Itemsets pattern of vulnerabilities in all OSS applications as shown in Table 10. On the whole, we can conclude that "Denial of Service, Code Execution", "Code Execution and Overflow", "Code Execution, Memory Corruption" and "Denial of Service and Overflow" are the most high 2-Itemsets frequent pattern of vulnerabilities in our results. The conclusion of Section 4.3 gives the answer to RQ3.

In Section 4.4, we identified that which type of vulnerability has the maximum occurrence in a specific OSS application. We find out a total of 10,058 vulnerabilities faced to the selected 15 OSS application over the period of 2006-2016 as shown in Table 11. In Table 11 DOS (Denial of Services) has the highest frequency of 3764 across all OSS applications. Another predominant type of vulnerability in our results is the V2 (Code Execution), which has occurrence of 1944 across all OSS applications. Table 11, also represents a leading vulnerability such as V3 (Overflow), which has occurrence of 1288 across all OSS applications. V4 (Memory Corruption) is another astonishing vulnerabilities facade to the selected OSS applications, which has occurrence of 1041 across all OSS applications. We identified that V6, V9 and V10 has almost a maximum occurrence in the selected OSS applications. There are some vulnerability such as V5, V7, V8, V11, V12 and V13 which has very less occurrence in all OSS applications. Overall, we can conclude from section 4.4 that "Denial of Services", "Code Execution", "Overflow", "Memory Corruption", "Bypass Something" and "Gain Information" are the most top types of vulnerabilities façade to the OSS applications and obviously this is the answer of our RQ4.

In Section 4.5, we discussed the use of N-Itemsets pattern of historically vulnerabilities in some OSS applications to predict the next possible recur vulnerability or vulnerability pattern in future to these OSS applications. We divided our data sets of N-Itemsets pattern of vulnerabilities into training and testing sets as explained in Section III. In order to get accurate results, we carried out two experiments on our collection. Initially, we use the N-Itemsets pattern of vulnerabilities from 2006-2011 as training set data and N-Itemsets pattern of vulnerabilities from 2012-2016 as testing set data. The main objective behind these experiments was that to test the predication power with older and innovative data. We carried out these experiments

for single vulnerability and three types of N-Itemsets: 2-Itemset, 3-Itemsets and 4-Itemsets pattern of vulnerabilities in the selected five OSS applications. We conclude that the precision and recall are high when we are searching for the subsequently vulnerability based on a single vulnerability or 2-Itemsets pattern of vulnerability. However, the precision and recall measures values reduce when higher N-Itemsets are used to look for next possible vulnerability. This is for the reason that the number of sequential (ordered) collections of vulnerabilities similar to the pattern searched. We again carried out the similar experimentation on single vulnerability and different N-Itemsets pattern of vulnerabilities to predict the next possible vulnerability in top 5 popular OSS applications. Moreover, training older data have reduced the recall of predicting innovative patterns nevertheless can still predict the incidence of identified vulnerabilities with high precision and obviously it is the answer of RQ5.

Using all our study questions' responses the OSS application vendor organizations can improve the security of their systems. In future, we are planning to analyze the other attributes of CVE, CWE and NVD repositories. We also planning to develop an open source software vulnerability detection model, which can automatically detected the vulnerabilities.

ACKNOWLEDGEMENT

Authors are grateful to the team of Software Engineering Research Group, University of Malakand, Pakistan, (SERG_UOM) for giving the valuable time to the thoroughly review of the paper.

7. Reference

[1] F. Lomio, E. Iannone, A. De Lucia, F. Palomba, and V. Lenarduzzi, "Just-in-time software vulnerability detection: Are we there yet?," *Journal of Systems and Software*, vol. 188, p. 111283, 2022/06/01/ 2022.

H. Hanif, M. H. N. Md Nasir, M. F. Ab Razak, A. Firdaus, and N. B. Anuar, "The rise of software vulnerability: Taxonomy of software vulnerabilities detection and machine learning approaches," *Journal of Network and Computer Applications*, vol. 179, p. 103009, 2021/04/01/ 2021.

K. R. Ahmad and k. S. Ullah, "A preliminary structure of software security assurance model," in *Proceedings of the 13th Conference on Global Software Engineering*, Sweden, 2018, pp. 137-140.

Weizhong. Qiang, Yuehua. Liao, Guozhong. Sun, Laurence. T. Yang , Deqing. Zou, and H. Jin, "Patch-Related Vulnerability Detection Based on Symbolic Execution," *IEEE Access* vol. 5, pp. 20777-20784, 2018.

K. S. U. Khan. Rafiq. Ahmad, Idris. Mohd. Yazid, "Systematic Mapping Study Protocol For Secure Software Engineering," in *AIMC 2018Asia International Multidisciplinary Conference*, UK, 2019, pp. 367-374.

Z. Dazhi, L. Donggang, C. Christoph, K. David, and L. Yu, "A distributed framework for demand-driven software vulnerability detection," *Journal of Systems and Software*, vol. 87, pp. 60-73, 2014.

H. Kekül, B. Ergen, and H. Arslan, "A multiclass hybrid approach to estimating software vulnerability vectors and severity score," *Journal of Information Security and Applications*, vol. 63, p. 103028, 2021/12/01/ 2021.

H. Yan, S. Luo, L. Pan, and Y. Zhang, "HAN-BSVD: A hierarchical attention network for binary software vulnerability detection," *Computers & Security*, vol. 108, p. 102286, 2021/09/01/ 2021.

A. Romanov, H. Tsubaki, and E. Okamoto, "An approach to perform quantitative information security risk assessment in it landscapes," *Journal of Information Processing*, vol. 18, pp. 213-226, 2010.

S. Luciano and G. Alessandro, "Exploring context-sensitive data flow analysis for early vulnerability detection," *Journal of Systems and Software*, vol. 113, pp. 337-361, March 2016.

[11] G. Lars and J. David, "Quantitative risk-based security prediction for component-based systems with explecity modeled attack profiles," *Journal of Systems and Software*, vol. 81, pp. 1327-1345, 2008.

[12] Murtaza. Syed. Shariyar, Khreich. Wael, Hamou-Lhadj. Abdelwahab, and B. A. Basar, "Mining trends and patterns of software vulnerabilities," *Journal of Systems and Software*, vol. 117, pp. 218-228, July 2016.

[13] Johson. Pontus, Gorton. Dan, Lagerstrom. Robert, and E. Mathias, "Time between vulnerabiluty disclosures: A mesure of software product vulnerability," *Computer & Secuirty*, vol. 62, pp. 278-295, September 2016.

[14] S. S. Murtaza, A. Hamou-Lhadj, M. Couture, and W. Khreich, "TotalADS: automated software anomly detection system," presented at the Proceedings of the 14th International Working Conference on Source Code Analysis and Manipulation, 2014.

[15] H. Chien-Chang and L. Feng-Yi, "A novel approach to evaluate software vulnerability priotization," *Journal of Systems and Software*, vol. 86, pp. 2822-2840, 2013.

[16] Kumar. Pratap and S. R. K, "A Review on 0-day Vulnerability Testing in Web Application," presented at the International Conference on Information and Communication Technology for Competative Strategies ICTCS'16, New York, USA, 2016.

[17] R. A. Khan, S. U. Khan, H. U. Khan, and M. Ilyas, "Systematic Literature Review on Security Risks and its Practices in Secure Software Development," *IEEE Access*, vol. 10, pp. 5456-5481, 2022.

[18] R. A. Khan, S. U. Khan, H. U. Khan, and M. Ilyas, "Systematic Mapping Study on Security Approaches in Secure Software Engineering," *IEEE Access*, vol. 9, pp. 19139-19160, 2021.

[19] R. Vibhandik and K. Arijit, "Vulnerability Assessment of Web Application-A testing Approach," presented at the Forth International Conference on e-Technologies and Networks for Development, Banglore, India, 2015.

P. Nushafreen, G. Deepa, K. F. Ahmed, P. S. Thilagam, and A. R. Pais, "Securing native XML database-driven web application from XQuery injection vulnerabilities," *Journal of Systems and Software*, vol. 122, pp. 93-109, Dec 2016.

R. A. Khan and S. U. Khan, "A preliminary structure of software security assurance model," presented at the Proceedings of the 13th International Conference on Global Software Engineering, Gothenburg, Sweden, 2018.

N. V. D. (NVD). National Vulnerability Database (NVD) [Online]. Available: https://nvd.nist.gov/

C. W. E. (CWE). Common Weakness Enumeration (CWE) [Online]. Available: https://cwe.mitre.org/

C. V. a. E. (CVE). Common Vulnerability and Exposure (CVE) [Online]. Available: http://cve.mitre.org/

P. Mike, "Know your open source code," *Network Security*, pp. 1-5, May 2016.

Sarrab. Mohamed and O. M. H. Rehman, "Empirical study of open source software selection for adoption, base on software quality characteristics," *Advances in Engineering Software*, vol. 69, pp. 1-11, January 2014.

K. Chen, D-G. Feng, P-R. Su, C-J. Nie, and X.-F. Zhang, "Multi-cycle vulnerability discovery model for predicition," *Journal of Software*, vol. 21, pp. 2367-2375, 2010.

A. Nadeem, K. Ahsan, and M. Sarim, "Illustration, Detection & Prevention of Sleep Deprivation Anomaly in Mobile Ad Hoc Networks," *Mehran University Research Journal of Engineering and Technology*, vol. 36, pp. 233-242, 2017-04-01 2017.

R. A. Khan, S. U. Khan, M. Ilyas, and M. Y. Idris, "The State of the Art on Secure Software Engineering: A Systematic Mapping Study," presented at the Proceedings of the Evaluation and Assessment in Software Engineering, Trondheim, Norway, 2020.

Schryen. Guido and K. Rouven, "Open source vs. Closed source software: Towards measuring security," presented at the SAC'09, Honolulu, Hawaii, U.S.A, 2009.

[31] V. Vasilyev, A. Vulfin, V. Gvozdev, and A. Nikonov, "Semantic Text Analysis Technology Application in Assessing Current Threats and Software Vulnerabilities," *IFAC-PapersOnLine*, vol. 54, pp. 599-604, 2021/01/01/ 2021.

[32] Alhazmi. Omar, Y. Malaiya, and R. Indrajit, "Security Vulnerabilities in Software Systems: A Quantitative Perspective," *IFIP International Fedration for Information Processing*, pp. 281-294, 2005.

[33] Nam H. Pham, Nguen. Tung. Thanh, Nguen. Hoan. Anh, and N. Tien, "Detection of recurring software vulnerabilities," presented at the IEEE/ACM international conference on Automated Software engineering, Antwerp, Belgium, 2010.

[34] D. Wijayesekara, M. Manic, J. L. Wright, and M. McQueen, "Vulnerabilitiy Identification and Classification Via Text Minning Bug Database," presented at the 40th Annaual Conference of the IEEE Industrial Electronic Society, 2014.

[35] O. H. Alhazmi and Y. K. Malayia, "Application of vulnerability discovery models to major operating systems," *IEEE Transaction Reliability*, vol. 57, pp. 14-22, 2008.

[36] S.G. Eick, T.L. Graves, A.F. Karr, and J. S. Maroon, "Does code decay? Assesing the evidence from change management data," *IEEE Transaction on Software Engineering*, vol. 27, pp. 1-12, 2001.

[37] S. Neuhaus and T. Zimmermann, "Security Trend Analysis with CVE Topic Mode;," presented at the 21st International Symposium on Software Reliability Engineering (ISSRE), 2010.

[38] S. Zaman, B. Adams, and A. E. Hassan, "Secuity versus performance bugs: a case study on Firefox," presented at the 8th Working Conference on Mining Software Repositoroes, 2011.

[39] S. H. Houmb and V. Nunes, "Estimating ToE Risk Level using CVSS," presented at the 4th International Conference on Availability, Relaiablity and Secuirty, 2009.

[40] S. Frei, M. May, U. Fielder, and B. Plattner, "Large scale vulnerability analysis," presented at the SIGCOMM Workshop on Large-scale attack defence (LSAD), 2006.

R. Anderson, "Security in open versus closed systems-the dance of boltzmann, coase and moore," presented at the Open Source Software, Economics, Law and Policy, 2005.

E. Rescorla, "Is finding security holes a good idea?," *IEEE Security Privacy*, vol. 3, pp. 14-19, 2005.

A. Ozment and S. E. Schechter, "Milk or wine: does software security improve with age?," presented at the 15th USENIX-SS'06 Security Symposium, Berkeley, CA, USA, 2006.

M. Bland, *An Introduction to Medical Statistics*, 3rd ed.: Oxford Medical Publishers, 2000.

Khan. Rafiq. Ahmad and K. S. Ullah, "A Survey Based Study on Communication and Coordination Challenges in Offshore Software Development Outsourcing Relationships from Vendors' Perspective," presented at the Proceedings of the Fourth International Multi-topic Conference (IMTIC'15), Mehran University, Jamshoro, Pakistan, 2015.

Khan. Siffat. Ullah and A. M. Ilyas, "Intercultural Challenges in Offshore Software Development Outsourcing Relationships: An Exploratory Study Using a Systematic Literature Review," *IET Software*, vol. 8, pp. 161-173, 2014.

A. Sikandar and K. S. Ullah, "Software outsourcing partnership mode: An evaluation framework for vendor organizations," *The Journal of Systems and Software*, vol. 17, pp. 402-425, 2016.

R. A. Khan, M. Y. Idris, S. U. Khan, M. Ilyas, S. Ali, A. U. Din, *et al.*, "An Evaluation Framework for Communication and Coordination Processes in Offshore Software Development Outsourcing Relationship: Using Fuzzy Methods," *IEEE Access*, vol. 7, pp. 112879-112906, 2019.

Ian H. Witten, Eibe. Frank, and M. A. Hall, *Data Mining Practical Machine Learing Tools and Techniques*, 3rd ed. 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA Morgan Kaufmann, 2011.