ISSN:0975-9646

Ranking Vulnerability for Web Application based on Severity Ratings Analysis

Nitish Kumar#1, Kumar Rajnish#2 Anil Kumar#3

1,2,3*Department of Computer Science & Engineering, Birla Institute of Technology, Mesra Ranchi, Jharkhand, India*

Abstract.-Vulnerability in web application persistently challenges the security in the web application software products. Now a day, Security is paramount way to protect software against wicked attacks and other hacker risk so that the software continues to function correctly under such potential risks. In recent year, the communication through the internet has increased in lapse and bounds. But the security in web application has compromised because of lots of vulnerabilities found in the web applications. Knowing these vulnerabilities and their consequences, the software developer can get some clue to develop the patches for most common vulnerabilities. The main objective is to find the vulnerabilities causing more adverse affect and rank those vulnerabilities. We propose a mathematical model to rank the vulnerability for the web application which will be based on vulnerability severity rating.

Keywords- vulnerability,

I. Introduction

Software security is an idea implemented to guard software against wicked attack and other hacker risks so that the software continues to function correctly under such potential risks. Most software assaults happen because of exploitation of multiple vulnerabilities in software products, which are vital to the operation of any business, association or even to the security infrastructure of a country. Vulnerability in software can expose intellectual property, user trust, and industry operations and services. However, ensuring security is challenging because software becomes more complex day by day now. It is continuously reported to be vulnerable to attacks and compromises despite of using most recent security techniques and protocols. That is why software is one of the root causes of all common computer security problems.Many experts in the field of security such as SANS, provide practices to control the security risks which consist of system configuration sensitive data protection through encryption and eliminating flaws from software applications.

Tom DeMarco stated that, “You can’t control what you can’t measure.” This clearly states the importance of metrics in software engineering. Metrics are quantifiable measurement. Security metrics are quantitative pointer for all the security attributes of an information system or technology. Metrics help us to comprehend quality and consistency of software. Metrics also provides a universal way to exchange thoughts, to measure product and service quality, and to improve a process. Significant attempts have been put on quality control in the software development industry, but, there are still a lot of vulnerabilities and weaknesses that remain in software products. Software vulnerabilities exist due to flaws and errors in design, coding, testing, and maintenance of software. These vulnerabilities could be exploited by attacker to compromise the computing system where the software is running on. Therefore, the number of vulnerabilities and the severity of those vulnerabilities should be important indicators for software security and trustworthiness.

We propose an approach to rank the vulnerability in web application based on severity ratings analysis. The tools used for this purpose are : the Common Vulnerability and Exposures(CVE) standard [7], an industry standard for vulnerability and exposure names; the Common Weakness Enumeration (CWE) [6], which lists software weaknesses; the Common vulnerability Scoring System (CVSS) [8], a vulnerability scoring system design to provide open and standardized method that rate software vulnerability; Common Attack Pattern Enumeration and Classification (CAPEC) [9], a list of attack patterns. These tools help us to find the vulnerability ranking.

The vulnerabilities found are generally disclosed by the finders using some of the common reporting mechanisms that have been developed. Some of them use a scoring system such as Common Vulnerability Scoring System (CVSS). The databases for the vulnerabilities and defects are maintained by organizations such as National Vulnerability Database (NVD) [4], Open Source Vulnerability Database, US-CERT, Secunia, etc. We have used the NVD database because it provides the most extensive datasets. NVD is the U.S. government repository of vulnerability management data collected and organized using specific standards. It includes databases of security checklists, security related software flaws, misconfigurations, product names, and impact metrics. NVD is synchronized with Common Vulnerabilities and Exposures (CVE), which is a list of information security vulnerabilities and exposures that aims to provide common names for publicly known problems, so that any updates to CVE appear immediately on NVD. Since detected and qualified vulnerabilities take some time become an official CVE entry, the database does not reflect all the vulnerabilities. Each CVE listed in NVD has a corresponding CVSS score, and according to the types of vulnerabilities, there are 23 categories of vulnerabilities in NVD; however, nine of them have not been mapped to the attack patterns of CAPEC till now. To improve the accuracy in our approach, we decided to work with only the

14 types of vulnerabilities that have been mapped to attack patterns of CAPEC.

The rest of the paper is organized as follows: Section II is review work, Section III discusses the ranking algorithm, Section IV presents two examples to illustrate our approach, Section V compares the existing and proposed work and Section VI includes conclusion and future scope.

II. Review Of Existing Work

JuAn Wang et al., [1] proposed a mathematical method for ranking attacks based on vulnerability analysis. In this paper they calculated the severity level of a potential attack patterns against a software product and ranking them based on the software's vulnerability information. The tools used for this purpose are: the Common Vulnerabilities and Exposures (CVE) [7], an industry standard for vulnerability and exposures names; the Common Weakness Enumeration (CWE) [6] , which lists software weaknesses; the Common vulnerability Scoring System (CVSS) [8], a vulnerability scoring system design to provide open and standardized method that rate software vulnerability, and the Common Attack Pattern Enumeration and Classification (CAPEC) [9], which is a list of attack patterns. In this paper, they considered only 14 types of software vulnerabilities out of 23 known vulnerabilities. Divided these vulnerabilities into three time intervals: present , recent and past. Also three coefficients for three time interval are assumed. These coefficients are 0.5, 0.3,and 0.2 for present, recent, and past respectively. Finding weight using these coefficients for each weakness and all the CAPEC-ID of the respective weaknesses. Then higher value of weight of attack pattern is taken with corresponding CAPEC-ID to rank the vulnerabilities. They have ranked the top ten vulnerabilities attack patterns for two different browsers namely *Mozilla Firefox 3 and Internet Explorer 7.*

III. Proposed Ranking Algorithm

The proposed vulnerability ranking algorithm is a new approach for finding the rank of vulnerability for web application software products. The algorithm is based on Severity rating. The Severity rating is the rating given by the National Institute of Standard and Technology, according to the severity level of the vulnerability found.The Severity levels are categorized in three types: Low Severity, Medium Severity, and High Severity. These severities are having their range to categorize the vulnerabilities in these levels. The Common Vulnerability Scoring System (CVSS), is a standard vulnerability scoring system for rating software vulnerabilities. This CVSS score is categorized according to their severity level ranges. The Range of the different Severity Levels are:

Severity	Range
Low	0.0 – 3.9
Medium	4.0 – 6.9
High	7.0 – 10.0

The vulnerability information is retrieved from the National Vulnerability Database. The data is collected for a specific period of one year, and for specific software products, namely: Google Chrome and Mozilla Firefox.All the vulnerabilities collected from NVD are classified in 14 types, there are total of 23 types of vulnerabilities, according to NVD, we have taken only 14 because the rest nine are not mapped to attack pattern of CAPEC (Common Attack Pattern Enumeration and Classification). The algorithm has following steps:

Step 1: Group the retrieved vulnerabilities in three different severity levels: low, medium, and high.

Step 2: Calculate the sum of CVE-ID's scores of individual vulnerabilities in high, medium, and low severity CVSS scores, using the following equations:

$$CVSS_{HIGH} = \sum_i^m CVSS_{\mathrm{i}} \quad (1)$$

$$CVSS_{MEDIUM} = \sum_i^m CVSS_{\mathrm{i}} \quad (2)$$

$$CVSS_{LOW} = \sum_i^m CVSS_{\mathrm{i}} \quad (3)$$

where, m is the number of CVE-ids found in each CWE vulnerability.

Step 3: Calculate the sum of $\mathrm{CVSS}_{\mathrm{HIGH}}$, $\mathrm{CVSS}_{\mathrm{MEDIUM}}$, and $\mathrm{CVSS}_{\mathrm{LOW}}$of each vulnerabilities, using the equation:

$$CVSS_{TOTAL} = \sum(CVSS_{HIGH} + CVSS_{MEDIUM} + CVSS_{LOW}) \quad (4)$$

where, $CVSS_{TOTAL}$is the summation of all the three severity category CVSS scores of low, medium, and high.

Step 4: There is possibility that in $CVSS_{TOTAL}$column , we get zero scores in one or multiple places, this is because there is no such vulnerabilities found in the given time period. So in this algorithm we have taken only those vulnerabilities which are actually causing problem. For this, we have to find the number of non-zero vulnerabilities columns which is represented by K. The equation for finding K is :

$$K = (14 – \text{No. of zeros in } CVSS_{TOTAL}\text{column}) \quad (5)$$

were '14' represents the total number of types of vulnerabilities out of 23 known vulnerabilities.

Step 5: Using the K, find the rank of vulnerabilities. The equation for finding rank is:

$$\text{Rank} = (\lfloor (CVSS_{TOTAL})_{\mathrm{i}} \rfloor \,\%\, K) + 1 \quad (6)$$

The rank of each vulnerability is calculated by taking modulo of floor value of $CVSS_{TOTAL}$and one is added to it. In our algorithm, lesser the rank value (1 is less than 2) higher is the vulnerability.

Step 6: Finally, the ranking of top ten vulnerabilities is decided by taking the maximum frequency of occurrences of CAPEC-ID according to the rank value.

IV. EXAMPLES

Example 1: Google Chrome v 49.

1. *Categorizing the vulnerabilities.*

Collected vulnerabilities and grouped them in accordance with the 14 fixed types in CWE. Second, we categorized the vulnerabilities of each type according to the high, medium, and low severity.

Weakness (CWE)	**CWE-ID**	**HIGH**	**MEDIUM**	**LOW**
Authenti-cation Issues	CWE-287	None	None	None
Buffer Errors	CWE-119	CVE-2015-8480(10.0) CVE-2015-8479(7.5) CVE-2015-6778(7.5) CVE-2015-6764(7.5) CVE-2015-1360(7.5)	CVE-2015-6776(6.8) CVE-2015-1273(6.8) CVE-2015-1271(6.8) CVE-2015-1240(5.0) CVE-2015-1225(5.0)	None
Code Injections	CWE-94	None	None	None
Cross-Site Request Forgery (CSRF)	CWE-352	None	None	None
Cross-Site Scripting (XSS)	CWE-79	None	CVE-2015-1286(4.3) CVE-2015-1275(4.3) CVE-2015-1264(4.3)	None
Format String Vulnerability	CWE-134	None	None	None
Information Leak/ Disclosure	CWE-200	None	CVE-2015-6759(5.0) CVE-2015-1285(5.0) CVE-2015-1247(5.0) CVE-2015-1244(5.0)	None
Input Validation	CWE-20	CVE-2015-1302(7.5) CVE-2015-1303(7.5) CVE-2015-1284(7.5)	CVE-2015-6790(4.3) CVE-2015-6784(4.3) CVE-2015-1261(5.0) CVE-2015-1241(4.3)	None
Link Following	CWE-59	None	None	None
OS Command Injections	CWE-78	None	None	None
Path Traversal	CWE-22	None	None	None
Permissions, Privileges And Access Control	CWE-264	CVE-2015-6770(7.5) CVE-2015-6768(7.5) CVE-2015-6755(7.5) CVE-2015-1293(7.5) CVE-2015-3335(7.5)	CVE-2015-6786(4.3) CVE-2015-6779(4.3) CVE-2015-1292(5.0) CVE-2015-1291(6.4) CVE-2015-3336(4.3)	None
Race Conditions	CWE-362	CVE-2015-6789(9.3)	CVE-2015-6761(6.8) CVE-2015-1234(6.8)	None
SQL Injection	CWE-89	None	None	None

2. *Calculate $CVSS_{HIGH}$, $CVSS_{MEDIUM}$, $CVSS_{LOW}$ and $CVSS_{TOTAL}$ using the following equations:*

$$CVSS_{HIGH} = \sum_{i}^{m} CVSS_{i} \quad (1)$$

$$CVSS_{MEDIUM} = \sum_{i}^{m} CVSS_{i} \quad (2)$$

$$CVSS_{LOW} = \sum_{i}^{m} CVSS_{i} \quad (3)$$

$$CVSS_{TOTAL} = \sum(CVSS_{HIGH} + CVSS_{MEDIUM} + CVSS_{LOW}) \quad (4)$$

Weakness	$\text{CVSS}_{\text{HIGH}}$	$\text{CVSS}_{\text{MEDIUM}}$	CVSS_{LOW}	$\text{CVSS}_{\text{TOTAL}}$
Authenti-cation Issues	0	0	0	0
Buffer Errors	40	30.4	0	70.4
Code Injections	0	0	0	0
Cross-Site Request Forgery	0	0	0	0
Cross-Site Scripting	0	12.9	0	12.9
Format String Vulnerability	0	0	0	0
Information Leak/ Disclosure	0	20	0	20
Input Validation	22.5	17.9	0	40.4
Link Following	0	0	0	0
OS Command Injections	0	0	0	0
Path Traversal	0	0	0	0
Permission, Privileges, and Access control	37.5	24.3	0	61.8
Race Condition	9.3	13.6	0	22.9
SQL Injection	0	0	0	0

3. *Calculate K, the number of non-zero entries in $CVSS_{TOTAL}$ column using the equation*:

K = (14 – No. of zeros in $CVSS_{TOTAL}$column) (5)

Here, K= 6.

4. *Rank the vulnerabilities with CAPEC-ID, using the equation:*

$$\text{Rank} = (\lfloor (CVSS_{TOTAL})_{i} \rfloor \% \text{K}) + 1 \quad (6)$$

Vulnerability/ Weakness	**Rank**	**CAPEC-ID**
Cross-Site Scripting	1	18,19,32,85,86,91
Permission, Privileges and Access Control	2	5,17,35,58,69,76
Information leak/ Disclosure, Race Condition	3	13,22,59,60,79, 26,29
—	4	—
Buffer Errors, Input Validation	5	3,7,8,9,10,13,14,22,24,31,32,42, 43,44,45,46,47,52,53,63,64,66, 67,71,72,73,78,79,80,81,83,85, 86,88,91,99
—	6	—

5.*Top ten vulnerability of Google Chrome.*

The ranking of top ten vulnerabilities is decided by taking the maximum frequency of occurrences of CAPEC-ID according to the rank value.

CAPEC-ID	**Description**
CAPEC-13	Subverting Environment Variable Values
CAPEC-18	Embedding Scripts in Non-Script Elements
CAPEC-19	Embedding Scripts within Scripts
CAPEC-22	Exploiting Trust in Client
CAPEC-32	Embedding Scripts in HTTP Query Strings
CAPEC-35	Leverage Executable Code in Non-executable File
CAPEC-79	Using Slashes in Alternate Encoding
CAPEC-85	AJAX Fingerprinting
CAPEC-86	Embedding Script (XSS) in HTTP Headers
CAPEC-91	XML Parser Attack

Example 2: Mozilla Firefox v 44.

1. *Categorize vulnerabilities.*

As with the first example, retrieve vulnerabilities and grouped them in accordance with the 14 fixed types in CWE and categorized the vulnerabilities of each type according to the high, medium, and low severity.

Weakness (CWE)	**CWE-ID**	**HIGH**	**MEDIUM**	**LOW**
Authenti-cation Issues	CWE-287	None	None	None

Buffer Errors	CWE-119	CVE-2015-7221(10.0)
		CVE-2015-7220(10.0)
		CVE-2015-7203(10.0)
		CVE-2015-7194(7.5)
		CVE-2015-7176(7.5)
			CVE-2015-7217(4.3)
			CVE-2015-7189(6.8)
			CVE-2015-4512(6.4)
			CVE-2015-4511(6.8)
				None

Code Injections	CWE-94	CVE-2014-8636(7.5)	None	None

Cross-Site Request Forgery (CSRF)	CWE-352	None	CVE-2015-0807(6.8)
			CVE-2014-8638(6.8)
				None

Cross-Site Scripting (XSS)	CWE-79	None	CVE-2015-7191(4.3)
			CVE-2015-4518(4.3)
			CVE-2015-4490(4.3)
				None

Format String Vulnerability	CWE-134	None	None	None

Information Leak/ Disclosure	CWE-200	None	CVE-2015-7215(5.0)
			CVE-2015-7208(5.0)
			CVE-2015-7186(4.3)
			CVE-2015-4515(4.3)
				None

Input Validation	CWE-20	None	CVE-2015-7216(6.8)
			CVE-2015-7211(5.0)
			CVE-2015-0799(4.3)
				None

Link Following	CWE-59	None	None	None
OS Command Injections	CWE-78	None	None	None
Path Traversal	CWE-22	None	None	None

Permissions, Privileges and Access Control	CWE-264	CVE-2015-0804(7.5)
		CVE-2015-0803(7.5)
		CVE-2015-0818(7.5)
		CVE-2015-8021(6.8)
		CVE-2015-8643(7.1)
			CVE-2015-7223(4.0)
			CVE-2015-7197(5.0)
			CVE-2015-4505(6.6)
			CVE-2015-4483(4.3)
			CVE-2015-0821(6.8)
				CVE-2015-2714 (2.1)

Race Conditions	CWE-362	None	CVE-2015-7189(6.8)
			CVE-2015-4510(6.8)
			CVE-2014-8640(5.0)
				CVE-2015-4481 (3.3)

SQL Injection	CWE-89	None	None	None

2. *Calculate $CVSS_{HIGH,}CVSS_{MEDIUM,}CVSS_{LOW}$and $CVSS_{TOTAL}$ using the following equations:*

$$CVSS_{HIGH} = \sum_i^m CVSS_{\mathrm{i}} \quad (1)$$

$$CVSS_{MEDIUM} = \sum_i^m CVSS_{\mathrm{i}} \quad (2)$$

$$CVSS_{LOW} = \sum_i^m CVSS_{\mathrm{i}} \quad (3)$$

$$CVSS_{TOTAL}= \sum(CVSS_{HIGH} + CVSS_{MEDIUM} + CVSS_{LOW}) \quad (4)$$

Weakness	$CVSS_{HIGH}$	$CVSS_{MEDIUM}$	$CVSS_{LOW}$	$CVSS_{TOTAL}$
Authenti-cation Issues	0	0	0	0
Buffer Errors	45	24.3	0	69.3
Code Injections	7.5	0	0	7.5
Cross-Site Request Forgery	0	13.6	0	13.6
Cross-Site Scripting	0	12.9	0	12.9
Format String Vulnerability	0	0	0	0
Information Leak/ Disclosure	0	18.6	0	18.6
Input Validation	22.5	16.1	0	38.6
Link Following	0	0	0	0
OS Command Injections	0	0	0	0
Path Traversal	0	0	0	0
Permission, Privileges, and Access control	36.4	26.7	2.1	65.2
Race Condition	0	18.6	3.3	21.9
SQL Injection	0	0	0	0

3. *Calculate K, the number of non-zero entries in $CVSS_{TOTAL}$column using the equation*:

K = (14 – No. of zeros in $CVSS_{TOTAL}$column) (5)

Here, K= 8.

4. *Rank the vulnerabilities with CAPEC-ID, using the equation:*

Rank = (⌊ $(CVSS_{TOTAL})_i$⌋ % K) + 1 (6)

Vulnerability/Weakness	Rank	CAPEC-ID
__	1	__
Permission, Privileges, and Access control	2	5,17,35,58,69,76
Information Leak/ Disclosure	3	13,22,59,60,79,
__	4	__
Cross-Site Scripting	5	18,19,32,85,86,91
Buffer Errors, Cross-Site Request Forgery, Race Condition	6	8,9,10,14,24,26,29,42,44,45,46,47,62
Input Validation	7	3,7,8,9,10,13,14,18,22,24,28,31,32,42,43,88,45,46,47,52,53,63,64,66,67,71,72,73,78,79,80,81,83,85,86,91,99
Code Injections	8	35,77

5.*Top ten vulnerability of Mozilla Firefox.*

The ranking of top ten vulnerabilities is decided by taking the maximum frequency of occurrences of CAPEC-ID according to the rank value.

CAPEC-ID	Description
CAPEC-13	Subverting Environment Variable Values
CAPEC-32	Embedding Scripts in HTTP Query Strings
CAPEC-35	Leverage Executable Code in Non-executable File
CAPEC-45	Buffer Overflow via Symbolic Links
CAPEC-46	Overflow Variables and Tags
CAPEC-47	Buffer Overflow via Parameter Expansion
CAPEC-79	Using Slashes in Alternate Encoding
CAPEC-85	AJAX Fingerprinting
CAPEC-86	Embedding Script (XSS) in HTTP Headers
CAPEC-91	XML Parser Attack

V. COMPARISON WITH EXISTING WORK

Comparing the purposed work with the existing work [1], will give some relationship between these two methodologies. The tools used in these methods are CWE, CVE, CVSS, and CAPEC. These are the standard tools used to represent a good security metrics. The browsers which are taken by the existing work are Mozilla Firefox 3 and Internet Explorer 7 and the internet browsers taken for analysis in the proposed work are Google Chrome v 49 and Mozilla Firefox v 44. The top ten attack pattern vulnerability of Mozilla Firefox and Internet Explorer of the existing work is shown in the following table:

Mozilla Firefox 3		Internet Explorer 7	
CAPEC-ID	Description	CAPEC-ID	Description
CAPEC-13	Subverting Environment Variable Values	CAPEC-8	Buffer Overflow in an API Call
CAPEC-17	Accessing, Modifying or Executing Executable Files	CAPEC-9	Buffer Overflow in Local Command-Line Utilities
CAPEC-22	Exploiting Trust in Client	CAPEC-10	Buffer Overflow via Environment Variables
CAPEC-32	Embedding Scripts in HTTP Query Strings	CAPEC-14	Client-Side Injection-induced Buffer Overflow
CAPEC-35	Leverage Executable Code in Non-executable File	CAPEC-24	Filter Failure through Buffer Overflow
CAPEC-76	Manipulating Input to File System Calls	CAPEC-35	Leverage Executable Code in Non-executable File
CAPEC-79	Using Slashes in Alternate Encoding	CAPEC-42	MIME Conversion
CAPEC-85	AJAX Fingerprinting	CAPEC-45	Buffer Overflow via Symbolic Links
CAPEC-86	Embedding Script (XSS) in HTTP Headers	CAPEC-46	Overflow Variables and Tags
CAPEC-91	XML Parser Attack	CAPEC-77	Manipulating User-Controlled Variables

The top ten attack pattern vulnerability of Google Chrome and Mozilla Firefox of the proposed work is shown in the table given below:

Google Chrome v 49.		**Mozilla Firefox v 44.**	
CAPEC-ID	**Description**	**CAPEC-ID**	**Description**
CAPEC-13	Subverting Environment Variable Values	CAPEC-13	Subverting Environment Variable Values
CAPEC-18	Embedding Scripts in Non-Script Elements	CAPEC-32	Embedding Scripts in HTTP Query Strings
CAPEC-19	Embedding Scripts within Scripts	CAPEC-35	Leverage Executable Code in Non-executable File
CAPEC-22	Exploiting Trust in Client	CAPEC-45	Buffer Overflow via Symbolic Links
CAPEC-32	Embedding Scripts in HTTP Query Strings	CAPEC-46	Overflow Variables and Tags
CAPEC-35	Leverage Executable Code in Non-executable File	CAPEC-47	Buffer Overflow via Parameter Expansion
CAPEC-79	Using Slashes in Alternate Encoding	CAPEC-79	Using Slashes in Alternate Encoding
CAPEC-85	AJAX Fingerprinting	CAPEC-85	AJAX Fingerprinting
CAPEC-86	Embedding Script (XSS) in HTTP Headers	CAPEC-86	Embedding Script (XSS) in HTTP Headers
CAPEC-91	XML Parser Attack	CAPEC-91	XML Parser Attack

Analyzing the top ten vulnerabilities of existing and proposed methodologies on different browsers, the attack which are mostly common in these are Buffer Overflow, Cross-site scripting, Input Validation and others are same in both the methodologies. This shows a good relationship between proposed and existing work.

VI. CONCLUSION AND FUTURE SCOPE

Security in web browsers is one of the prime importance in the Internet world. Web browser allow user to view information on Internet by retrieving data from remote servers and displaying in the user's system. Web browsers are the common target for attacker to hack the information about user's system and other files. The other security threat is that hackers like to execute malicious code by exploiting a buffer overflow, cross-site scripting or injecting code or other application code in order to make system function abnormally. The Buffer Overflow is occurs when a browser permits a read and write operation on memory outside the allocated space. Buffer Overflow is severe attack pattern found in both the methodologies as it is very common vulnerability or weakness in the web browsers. Cross-site scripting (XSS) occurs when an application fails to block the data which is executable by a browser like add-ons [6]. Input Validation is occurs when an application fails to validate the input data properly [5].Ranking the vulnerability is helpful for the software developer to pay more attention on these vulnerabilities so that developing the patches for the most common vulnerabilities will be easy. In future, more statistical analysis will be possible to validate the reliability and accuracy in our metrics. Also, a precise mathematical model is needed to uplift our security metrics.

REFERENCES

[1] Ju An Wang, Hao Wang, MinzheGuo, Linfeng Zhou, and Jairo Camargo, "Ranking Attacks Based On Vulnerability Analysis", proceeding of the 43rd Hawaii International Conference on System Sciences, 2010, pp.1-10.

[2] M. Gegick, L. Williams, J. Osborne, and M. Vouk, "Ranking Attack-porne Components with a Predictive Model": proceeding of 19th International Symposium on Software Reliability Engineering, 2008.

[3] AnshuTripathi and Umesh K. Singh, "On prioritization of vulnerability categories based on CVSS scores", 2011, 6thInternational Conference on Computer Science and Convergence Information Technology, pp. 692-697.

[4] HyunChulJoh, and Yashwant K. Malaiya, "Seasonal Variation in the Vulnerability Discovery Process": 2009 International Conference on Software Testing Verification and Validation, pp. 191-200.

[5] National Institute of Standard and Technology, National Vulnerability Database.[Online] Available: http://nvd.nist.gov , accessed on January 2016.

[6] J. Whittaker, H. Thompson, "How to break software security", June 2003.

[7] Common Weakness Enumeration (CWE), The MITRE Corporation, [Online] Available: http://cwe.mitre.org/, accessed on January 2016.

[8] Common Vulnerability and Exposures (CVE), the MITRE Corporation. [Online] Available: http://cve.mitre.org/ , January 2016.

[9] Common Vulnerability Scoring System (CVSS), the MITRE Corporation. [Online] Available: http://www.first.org/cvss/ , January 2016.

[10] Common Attack Pattern Enumeration and Classification (CAPEC), the MITRE Corporation. [Online] Available: http://capec.mitre.org/ , January 2016.