International Journal of Hybrid Intelligent Systems 14 (2017) 1–11
DOI 10.3233/HIS-170237
IOS Press

Estimating influence of threat using Misuse Case Oriented Quality Requirements (MCOQR) metrics: Security requirements engineering perspective

Chitresh Banerjee[a,*], Arpita Banerjee[b] and S.K. Sharma[c]
[a]*Pacific Academy of Higher Education and Research University, Udaipur, India*
[b]*Department of Computer Science, St. Xavier's College, Jaipur, India*
[c]*Modern Institute of Technology & Research Centre, Alwar, India*

Abstract. Security is an aspect which contains varied classification and dimensions. One such classification of security is software security and it's facet is metrics. Software security metrics provides an estimation of how secure a software could be and indicates that where the loophole might occur while it is being developed. The realization of security implementation should occur during the initiation of software development, i.e. the requirements elicitation phase among the software development team. Misuse Case Oriented Quality Requirements (MCOQR) Metrics framework provides an easy and comprehensive way of identifying security loopholes in software much before it is developed. It provides 6 dimensional security indicators and estimators so that security team can have an insight into areas which needs further improvement and for proper drafting of security requirements. This research paper takes into account influence of threat predicted using the misuse case modeling for estimating the security aspect of software much before it is developed and implemented practically. In this paper an empirical study is provided that shows how security team may identify core areas where security could be enhanced further. The research work proves that if MCOQR metrics framework is applied during software development the outcome is more secure software.

Keywords: Security, software security, security metrics, software security metrics, misuse case, security requirements engineering

1. Introduction

Security is an intangible entity and cannot be measured directly through any means [11]. It requires an amalgamation of technical expertise and managerial insight for its proper measurement and subsequent prediction [14]. In Computer Science field, security may have varied classification like data security, network security, computer security, hardware security, information security, cyber security, web security, application security, and software security to name a few. All the mentioned classification may have some overlapping aspects and some unique aspect associated with the concerned security type [9].

Software security is core to all other types of security because everything is centered around it and operated, managed and controlled by it [24]. Researchers have advocated that if security could be built in and around the software during its initial development phase then a more comprehensive and secured software could be imagined. It increases the extent and dimension of security of that software when put to practical use. Incidents of exploitation of the software could be minimized if not 100% guaranteed [7,15].

A good number of guidelines, standards, and protocols have been laid out by national and international

*Corresponding author: Chitresh Banerjee, AP-III, AIIT, AUR, 14, Gopal Bari, Near Ajmer Pulia, Ajmer Road, Jaipur, Rajasthan 302004, India. Tel.: +91 9928446600; E-mail: chitreshh@yahoo.com.

agencies from time to time [17]. Some methods, tools, and techniques have also been proposed by the research organization and industry personnel's [2]. Still, considering the facts, the vulnerable side of software is getting exposed and is increasing in magnitude day by day. This is due to the fact that software development teams pay less attention to the security aspect of software during its development due to the time and cost constraint [12,13]. Another important factor is that the available guidelines, standards, and protocols are too broad in nature to be implemented and the methods, tools, and techniques available are too technical in nature and also lack a lay man's approach [17].

Metrics is a potential aspect of security which can provide measurement in form of indicators and estimators using which the security team can analyse those areas of an entity which can be further improved to strengthen the security. In software development, security is in the form of non-functional aspect and indicates an indirect measure [4]. Metrics can be used to quantify the security attribute of software while it is in the development phase and the indicators and estimators thus obtained may be used by the security analysis team to specify security requirements [18].

Researchers have suggested that security should be implemented '*right from the beginning*' during the software development process i.e., the requirements engineering phase so that it propagates further into the design phase and subsequent phases and adopts a comprehensive approach [3]. There are many techniques which could be used during the requirements engineering phase but considering the lay man's approach of security implementation, use cases and misuse cases which are object oriented (OO) concept are a potential contender in this case [23,25].

Using this OO concept, Misuse Case Oriented Quality Requirements (MCOQR) Metrics Framework has been proposed which may be used by the security team involved in the software development process to quantify security during the requirements elicitation phase. The outcome of MCOQR metrics framework is in the form of 6 indicators and estimators which could be further analysed to improve the misuse case modeling so that pragmatic security requirements could be specified based on it resulting in more secured software [5].

In this research paper, one of the 6 indicators and estimators, i.e. the influence of threat aspect has been taken which if implemented and anlaysed properly may help the security team of software development process to estimate the influence of threat of the proposed software during the requirements elicitation phase. Using this estimation, the areas which could be improved further are taken into consideration and a ranking of the software in terms of security could be done much before it is developed. It also contributes to the specification of security requirements of the said software.

Apart from the introduction section, the rest of the paper is organized as follows: Section 2 showcases the related work in this area, Section 3 highlights a brief of the Misuse Case Oriented Quality Requirements (MCOQR) Metrics framework, Section 4 presents the proposed MCOQR metrics for estimation of influence of threat of the software in development process, whereas results and discussions are covered in Section 5, and further conclusion and future work are given in Section 6.

2. Related work

Herrmann and Paech proposed Misuse-Oriented Quality Requirements Engineering (MOQARE) which explores the quality requirements and provides a general conceptual model. The relationships among the requirements are modeled using a Misuse Tree. It is a four step process, i.e. firstly, quality goals are identified based on business goals, business damages, and quality deficiencies, secondly, misuse case are described which includes threat, misuser, vulnerabilities, and their consequences, thirdly, countermeasures are defined, lastly, using the countermeasures, step 2 is repeated in a cyclic manner [10].

Chung and do Prado Leite in their research work provided a review of Non Functional requirements (NRF) and provided a classification in form of six areas viz., software variability, requirements analysis, requirements elicitation, requirements reusability, requirements traceability and aspect-oriented development. The researchers also suggested future prospects and emphasized on the empirical research on the use of NRFs during requirements engineering [8]. Okubo et al. proposed an extension of misuse case for eliciting security requirements. The researchers divided the process into two sub process, i.e. the business requirements elicitation and system requirements elicitation each consisting of 5 sub process respectively [16].

Santhosh Babu et al. proposed a security designers benchmark entitled 'Surksha' to facilitate designer to build security right from the requirements analysis phase. The proposed work is a eight step process which is well synchronized with STRIDE and

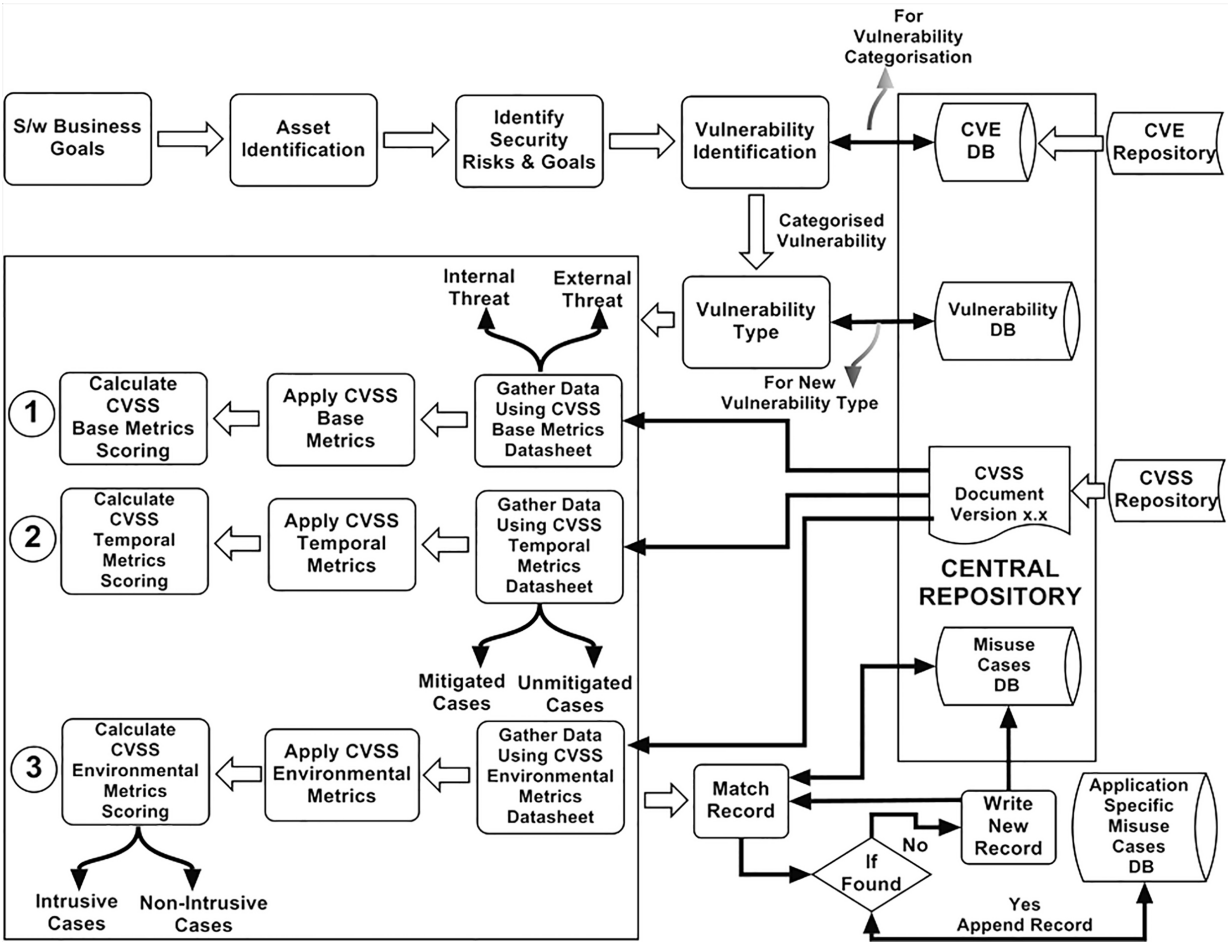

Fig. 1. Proposed framework of vulnerability identification and misuse case classification [5,6,19].

DREAD. The STRIDE is used to identify threats and DREAD is used to calculate the risk of the threat. Using the DREAD process, the threats are categorized as considered or neglected. The proposed work was applied to an e-commerce application and the results were discussed [22].

Raspotnig et al. proposed a combined process for elicitation and analysis of safety and security requirements. In the research work, eleven risk identification techniques were used, of which, five were used for the safety field and six were used for the security field. These eleven techniques were assessed for each criteria identified. The researchers in their work created artefacts Failure Sequence Diagrams (FSD), Combined Harm Assessment for Safety and Security of Information Systems (CHASSIS) and the Security Conceptual Model (SeCM) to compare the safety and security techniques for achieving the objectives. The evaluation of the proposed process was done on Air Traffic Management domain [20].

Salini and Kanmani proposed Model Oriented Security Requirements Engineering (MOSRE) framework for web applications. The researchers applied this framework on an E-Voting system to capture the security requirements and domain knowledge after identifying assets, threats, and vulnerabilities. The proposed framework is a sixteen step process for identifying security requirements. The researchers advocated that the proposed system is better than the traditional approach [21].

Yahya et al. proposed Essential Use Cases (EUC) for capturing and specifying the security requirements. The researchers further developed Security Essential Use Cases (Se-cEUC), Security Essential Interactions (SecEI), and Security Control Patterns (SecCtrl) library which assisted in capturing the security requirements. The researchers also developed a prototype

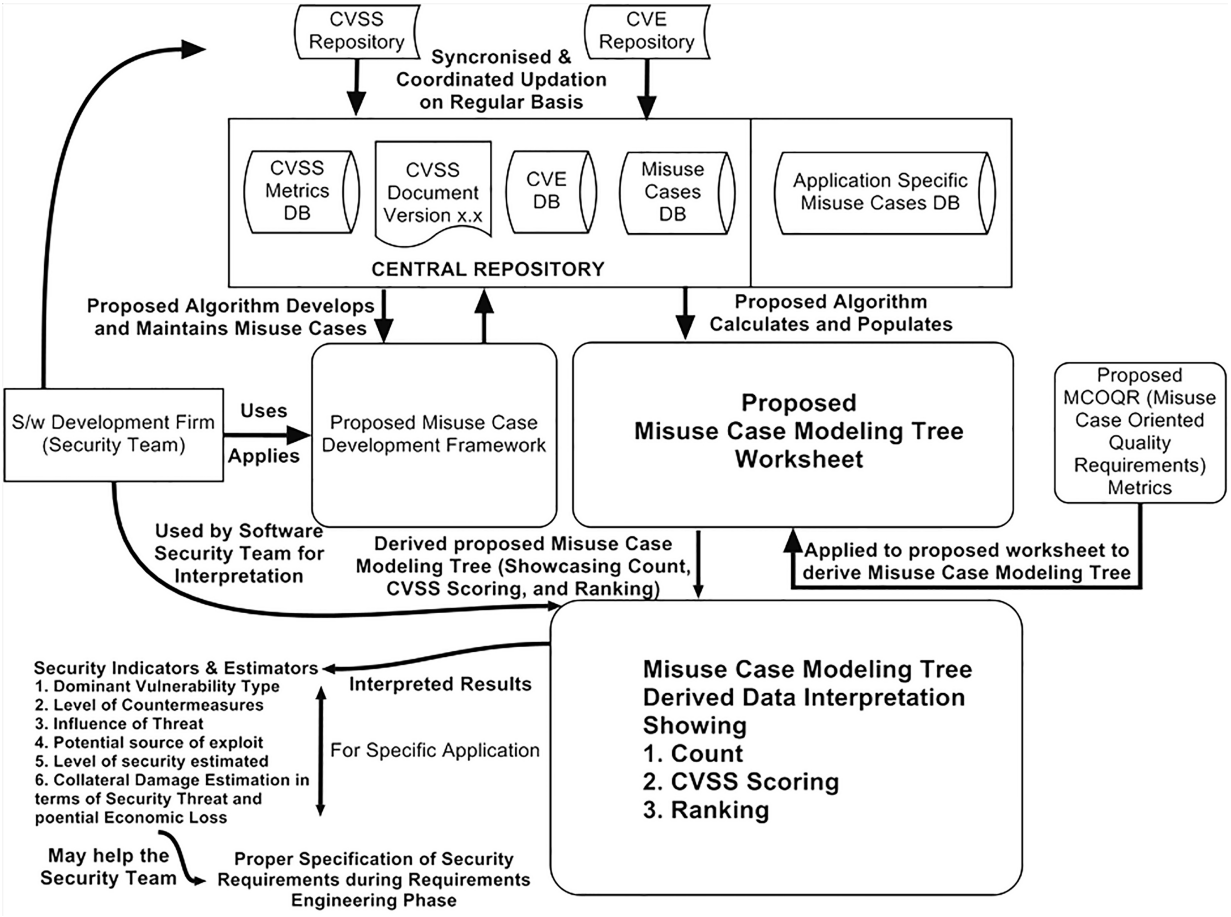

Fig. 2. Proposed Misuse Case Oriented Quality Requirements (MCOQR) framework [5,6,19].

tool called SecMEReq which is an extended version of MEReq for the automation of the proposed process [25].

Abdulrazeg et al. proposed a secure V model which integrates the security requirements engineering activities with requirements engineering activities of V model to for specifying security requirements for a web application. The proposed model is a nine step process. The researchers also highlighted the importance of SCRUM into V model for execution in an iterative manner [1].

The limitations of the study carried out so far in this field is that the framework or model proposed by researchers does have any synchronization and coordination with industry accepted standards like CVSS and CVE. The research work though helps in specifying the security requirements but lacks quantitative approach and hence the security of the software cannot be measured. Further, the study also lacks empirical study on large base of data and varied software applications.

3. Misuse Case Oriented Quality Requirements (MCOQR) metrics framework – an overview

In the implementation of the proposed Misuse Case Oriented Quality Requirements (MCOQR) Framework, first of all, the vulnerability is identified and associated misuse case classification takes place as show in Fig. 1. The process is synchronized with Common Vulnerability Enumeration (CVE) and Common Vulnerability Scoring System (CVSS) which are industry accepted standards. There are two major outcome of the process, i.e. first the classification of misuse cases based on the type of vulnerability and second the databases consisting of information related to misuse cases, CVE, and CVSS document [5].

Figure 2 demonstrates the workflow of proposed Misuse Case Oriented Quality Requirements (MCOQR) Metrics framework and the outcome of the framework when implemented during the requirements elicitation phase of software development is in the

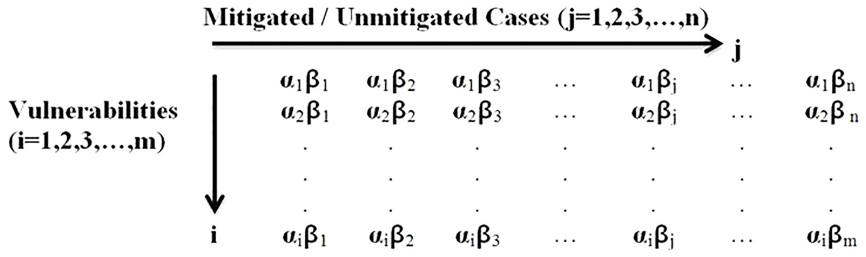

Fig. 3. MCOQR metrics representation.

form of 6 indicators and estimators, viz., dominant vulnerability type, level of countermeasure, influence of threat, level of security estimated, collateral damage estimation in terms of security threat and potential economic loss [5].

This paper highlights the research work carried out for the estimation of influence of threat which is one of the 6 indicators and estimators predicted before the software is developed. It showcases the proposed metrics for the estimation purpose and subsequent results and discussions are carried out to prove the importance and viability of the proposed work. The dataset created for the analysis purpose has been taken from CVE repository consisting of vulnerability reporting and their exploit since 1999 to till date software product wise.

4. Proposed mcoqr metrics for estimation of influence of threat

The proposed Misuse Case Oriented Quality Requirements Metrics (MCOQR) representation is given in Fig. 3.

The base metrics equation for the estimation of influence of threat during the requirements elicitation phase of software development process by the security team is show as follows:

$$\langle Case_M \rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_i \beta_j \tag{1}$$

Where '$Case_M$' represents Misuse cases which are Intrusive Mitigated (MC_{IM}), Intrusive Unmitigated (MC_{IUM}), Non-Intrusive Mitigated (MC_{NIM}), Non-Intrusive Unmitigated (MC_{NIUM}), Intrusive (MC_I), Non-Intrusive (MC_{NI}).

Following metrics may be used by the security team of software development process to estimate the influence of threat during the requirements elicitation phase:

From Eq. (1) following can be derived

$$MC_{IM} = MC_{IIM} + MC_{EIM} \tag{2}$$

where 'MC_{IM}' represents Misuse cases which are Intrusive in nature and Mitigated here MC_{IIM} are misuse cases which are internally exploitable, intrusive in nature and mitigated, and MC_{EIM} are misuse cases which are externally exploitable, intrusive in nature and mitigated.

From Eq. (1) following can be derived

$$MC_{IUM} = MC_{IIUM} + MC_{EIUM} \tag{3}$$

where 'MC_{IUM}' represents Misuse cases which are Intrusive in nature and Unmitigated here MC_{IIUM} are misuse cases which are internally exploitable, intrusive in nature and unmitigated, and MC_{EIUM} are misuse cases which are externally exploitable, intrusive in nature and unmitigated.

From Eqs (2) and (3) following can be derived

$$MC_I = MC_{IM} + MC_{IUM} \tag{4}$$

where 'MC_I' represents Misuse cases which are Intrusive in nature.

From Eq. (1) following can be derived

$$MC_{NIM} = MC_{INIM} + MC_{ENIM} \tag{5}$$

where 'MC_{NIM}' represents Misuse cases which are Non-Intrusive in nature and Mitigated here MC_{INIM} are misuse cases which are internally exploitable, non-intrusive in nature and mitigated, and MC_{ENIM} are misuse cases which are externally exploitable, non-intrusive in nature and mitigated.

From Eq. (1) following can be derived

$$MC_{NIUM} = MC_{INIUM} + MC_{ENIUM} \tag{6}$$

where 'MC_{NIUM}' represents Misuse cases which are Non-Intrusive in nature and Unmitigated here MC_{INIUM} are misuse cases which are internally exploitable, non-intrusive in nature and unmitigated, and MC_{ENIUM} are misuse cases which are externally exploitable, non-intrusive in nature and unmitigated.

From Eqs (5) and (6) following can be derived

$$MC_{NI} = MC_{NIM} + MC_{NIUM} \quad (7)$$

where 'MC_{NI}' represents Misuse cases which are Non-Intrusive in nature Metrics MC_{IIM}, MC_{EIM}, MC_{IIUM}, MC_{EIUM}, MC_{INIM}, MC_{ENIM}, MC_{INIUM} and MC_{ENIUM} explore the source of threat aspect of MCOQR metrics framework and are beyond the scope of this research paper. These metrics could be calculated using Eq. (1) mentioned above.

5. Result and validation

Using the process specified in Misuse Case Oriented Quality Requirements (MCOQR) Metrics framework synchronized with CVSS 3.0 document which contains the base metrics, temporal metrics and environmental metrics and data from CVE repository the information was collected in various tables of database. The data collected and analysed was using Microsoft 2008 Server and Linux Kernel for the year 2008 to year 2017. During the collection and analysis process, two vulnerabilities were taken viz., Denial of Service (DoS) and Execution Code (Exec Code). It was assumed that the vulnerability reported during an individual period (year) was well predicted be the security team during the requirements elicitation phase of the software development process. Hence, the vulnerabilities identified during 2008 were assumed to be predicted well in advance during the development of the said software and so on.

The entire process of collection and analysis of the vulnerabilities of the three said software is beyond the scope of this research paper, hence, this research paper is restricted only to demonstrate the intrusive and non-intrusive nature and the analysis of software during the individual years of the said software development process. It shows the vulnerable side of the software in development in terms of intrusive and non-intrusive nature and also highlights areas where improvisation is required or necessary to secure the software from within so that a comprehensive security requirement can be drafted. Of course, 100% security and future vulnerabilities introduced due to new technology and environment cannot be predicted well in advance, hence, this framework advocates for a cyclic process in which the various breaches and their associated vulnerabilities of the said software after implementation should be updated on a regular basis.

MCOQR Metrics Framework is suggested to be used by the software development industry for three main significant reasons. Firstly, it induces a sense of responsibility among the organization from security implementation point of view thereby advocating a separate security team. As per the statistics available, the academicians, researchers, as well as industry do advocate for a separate security team but very few, almost negligible portion of the industry have it and properly implements it during the software development process specially during the requirements engineering phase.

The reason being the cost factor which adds towards the selling cost of the software and the technical and operational infeasibility in terms of security team and security framework which as per the statistics available worldwide is a misconception and underweights the two factor cost aspect after the software is implemented. One, being the cost which is incurred to make the software secure after its implementation, and second, being the cost in terms of asset and reputational loss incurred during the breach and attack on the software.

Secondly, it is applicable for any kind of software and hence it is software neutral security framework and also it is irrespective of the type of vulnerabilities and the software development team can customize it as per their own classification of vulnerabilities (which could be 'n' in nos.) but driven by the foundational laws of CVSS and CVE. Lastly, it not only motivates the software firms to have their own set of vulnerability databases but also supports the importance of CVSS and CVE implementation and whenever a new vulnerability is identified which has no inclusion in CVSS or CVE, the when identified using the process of proposed framework can be reported back to the concerned agencies. Based on the information from the vulnerability database of the three individual software mentioned above (which is collected from the data available in http://www.cvedetails.com) the following dataset(s) was created as shown in Table 1. Here 'I' is for Intrusive Cases, 'NI' for Non-Intrusive Cases, 'C' for Vulnerability Count, 'S' for Vulnerability Scoring, and 'R' for Vulnerability Rating.

As per the dataset created and the analysis done using Fig. 4 as well as Fig. 5, it is shown that in the year 2008 for Microsoft 2008 Server software the count of pure Denial of Service (DoS) vulnerability was '2' with a scoring of '14.2' and a cumulative rating of '7.1'. No reporting of non-intrusive vulnerability was made. For Linux Kernel 2.6 ad 3.2 subversions, the count of pure Denial of Service (DoS) vulnerability was '8' with a scoring of '61' and a cumulative rating of '7.62'. No reporting of non-intrusive vulnerability

Table 1
Influence of threat dataset created using MCOQR metrics framework

Software/Application	Vulnerabilities	I			NI			I			NI		
		C	S	R	C	S	R	C	S	R	C	S	R
	Year →	2008						2009					
Microsoft 2008 Server	DoS	2	14.2	7.1	0	0	0	3	22.7	7.57	0	0	0
	Exec Code	3	26.3	8.77	0	0	0	12	114	9.5	0	0	0
Linix Kernel 2.6 and	DoS	8	61	7.62	20	89.9	4.49	14	104.4	7.44	15	70.1	4.13
3.2 sub versions	Exec Code	1	6.9	6.9	0	0	0	0	0	0	0	0	0
		2010						2011					
Microsoft 2008 Server	DoS	3	22.7	7.57	5	24	4.8	5	37.6	7.52	2	9.4	4.7
	Exec Code	5	44	8.8	0	0	0	4	38.6	9.65	0	0	0
Linix Kernel 2.6 and	DoS	14	105.6	7.56	21	98.6	4.66	10	66.6	6.79	21	106	5.04
3.2 sub versions	Exec Code	1	7.2	7.2	0	0	0	0	0	0	0	0	0
		2012						2013					
Microsoft 2008 Server	DoS	0	0	0	5	23.6	4.72	0	0	0	3	12	4.67
	Exec Code	12	113	9.42	0	0	0	9	80.3	8.92	0	0	0
Linix Kernel 2.6 and	DoS	8	61	7.68	23	105.5	4.56	1	7.2	7.2	23	108.9	4.73
3.2 sub versions	Exec Code	0	0	0	0	0	0	0	0	0	0	0	0
		2014						2015					
Microsoft 2008 Server	DoS	1	7.1	7.1	3	15	5	1	9.3	9.3	2	8.3	4.15
	Exec Code	7	65.8	9.4	0	0	0	11	102.3	9.3	0	0	0
Linix Kernel 2.6 and	DoS	3	20.5	6.92	28	126.9	4.5	8	64	7.96	22	100.1	4.35
3.2 sub versions	Exec Code	1	10	10	0	0	0	1	10	0	0	0	0
		2016						2017					
Microsoft 2008 Server	DoS	1	6.8	6.8	1	3.6	3.6	1	7.2	7.2	4	9.8	2.45
	Exec Code	8	74.8	9.35	0	0	0	3	24.1	8.03	0	0	0
Linix Kernel 2.6 and	DoS	10	78.7	8	16	71.6	4.53	11	80.3	7.3	14	64.5	4.61
3.2 sub versions	Exec Code	1	10	10	0	0	0	48	357.7	7.82	0	0	0

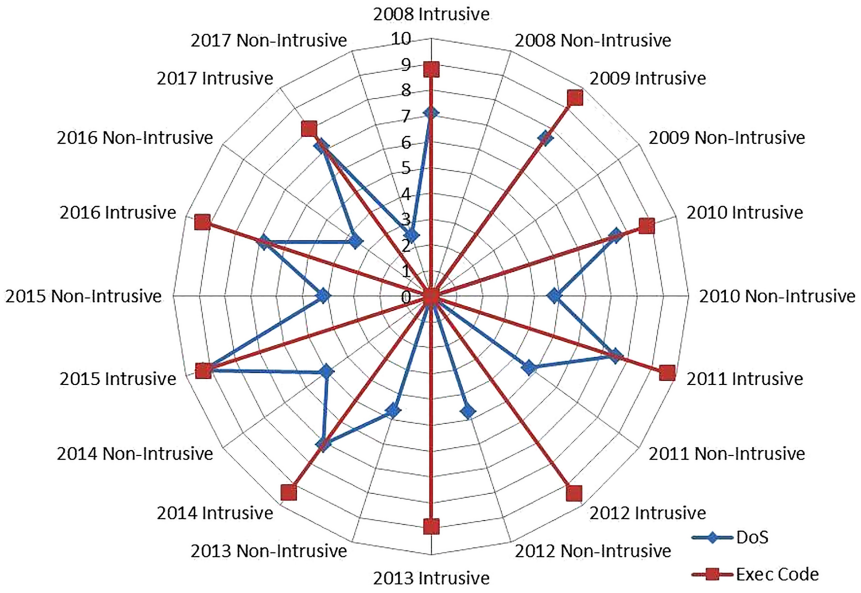

Fig. 4. Showing the trend of predicted intrusive and non-intrusive vulnerabilities as influence of threat (Microsoft 2008 Server, Year 2008–2017).

was made for Microsoft 2008 server but for Linux Kernel 2.6 ad 3.2 subversions, the count of pure Denial of Service (DoS) vulnerability was '20' with a scoring of '89.9' and a cumulative rating of '4.49'.

Here, the cumulative rating of '7.1' for Microsoft 2008 Server and '7.62' for Linux Kernel 2.6 & 3.2 subversions are on the higher side and needs further treatment to improvise the security aspect of the software for the specific vulnerability. The cumulating rating of non-intrusive vulnerability for Microsoft 2008 Server

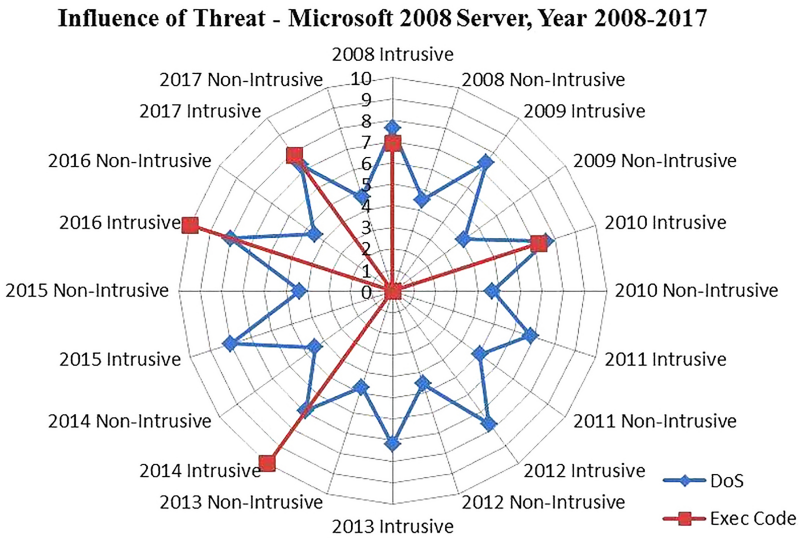

Fig. 5. Showing the trend of predicted intrusive and non-intrusive vulnerabilities as influence of threat (Linux Kernel 2.6 & 3.2 sub versions, Year 2008–2017).

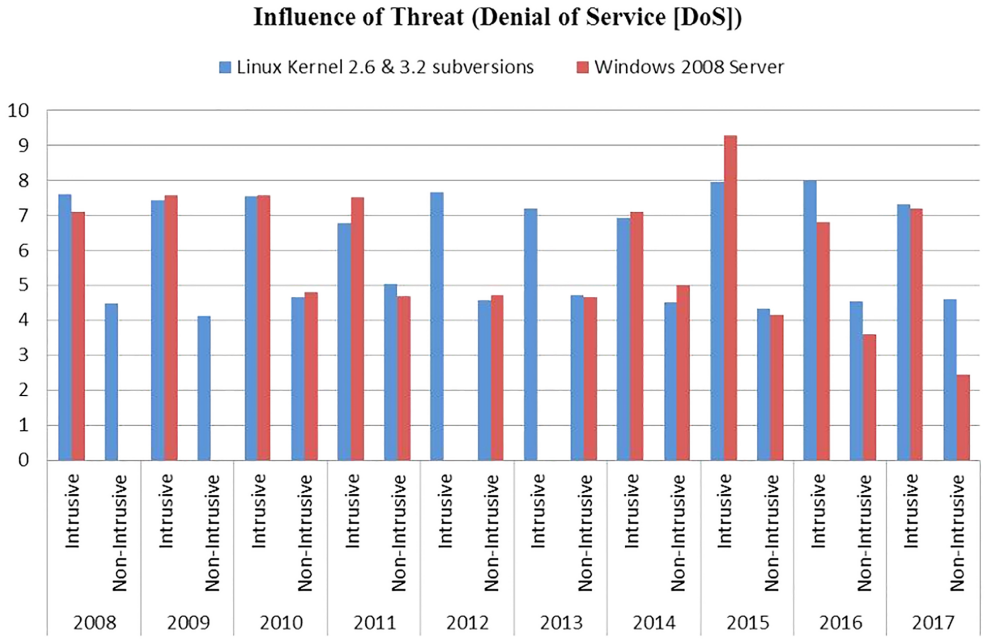

Fig. 6. Influence of threat (Denial of Service [DoS]).

is 'Nil' and for Linux Kernel 2.6 & 3.2 subversions it is '4.49' which is well within the acceptable limits as 100% security cannot be implemented and guaranteed.

For the year 2008 for Microsoft 2008 Server software the count of pure Exec Code vulnerability was '3' with a scoring of '26.3' and a cumulative rating of '8.77'. No reporting of non-intrusive vulnerability was made. For Linux Kernel 2.6 ad 3.2 subversions, the count of pure Exec Code vulnerability was '1' with a scoring of '6.9' and a cumulative rating of '6.9'. No reporting of non-intrusive vulnerability was made for Microsoft 2008 Server as well as for Linux Kernel 2.6 ad 3.2 subversions.

Here, the cumulative rating of '8.77' for Microsoft 2008 Server and '6.9' for Linux Kernel 2.6 & 3.2 subversions are on the higher side and needs further treat-

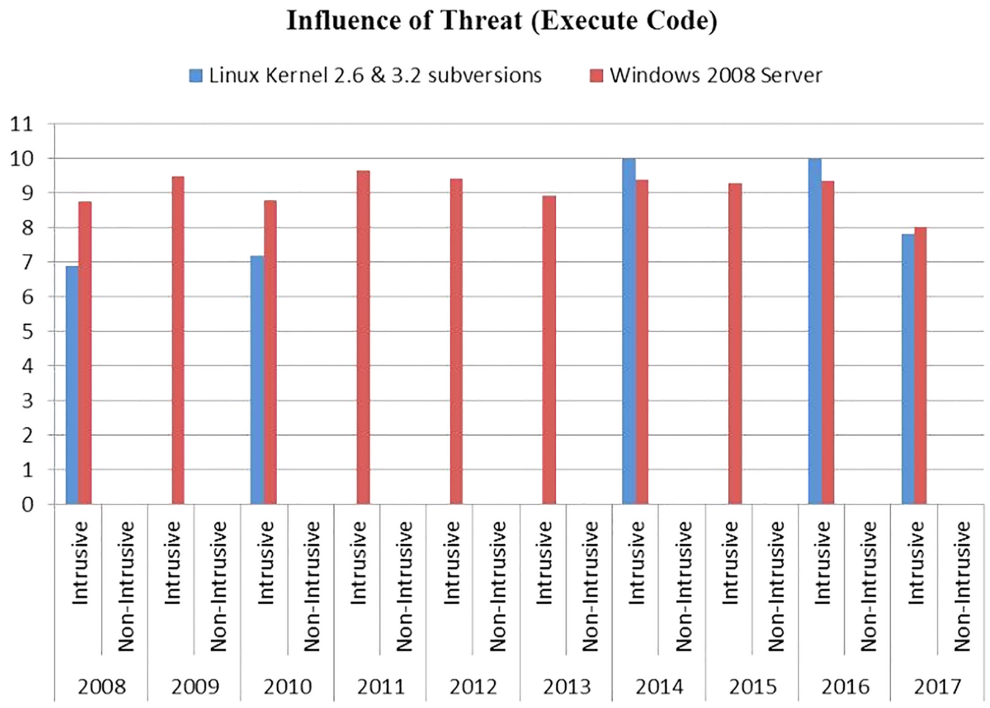

Fig. 7. Influence of threat (Exec Code).

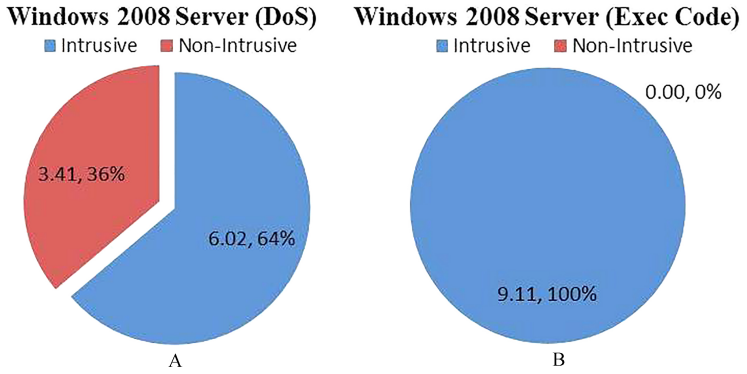

Fig. 8. (A) Windows 2008 Server (DoS) Year 2008–2017 Consolidated; (B) Windows 2008 Server (Exec Code) Year 2008–2017 Consolidated.

ment to improvise the security aspect of the software for the specific vulnerability. The cumulating rating of non-intrusive vulnerability for Microsoft 2008 Server and Linux Kernel 2.6 & 3.2 subversions is 'Nil'.

Still, for improvement, the security team involved in the analysis can further get an insight in to the level of mitigation/countermeasure applicable, i.e. whether it is mitigated or unmitigated, if it is mitigated whether it has an official fix, temporary fix, or work around solution. Thereafter, the team may have an understanding what is the potential source of threat, i.e. whether the vulnerability may be exploited internally or has more scope for external exploitation. The collateral damage estimated can also be examined to see the potential economic loss if the vulnerability is exploited and which assets may be harmed.

The above mentioned factors could be worked upon and corrected to bring down the level of vulnerability rating so that the software could be directed towards more security. Similar analysis could be done on the data collected for the year 2009 till 2017 and the same process can be applied for other type of vulnerabilities.

Figures 6 and 7 highlight the comparison of Influence of Threat in terms of vulnerability types for the

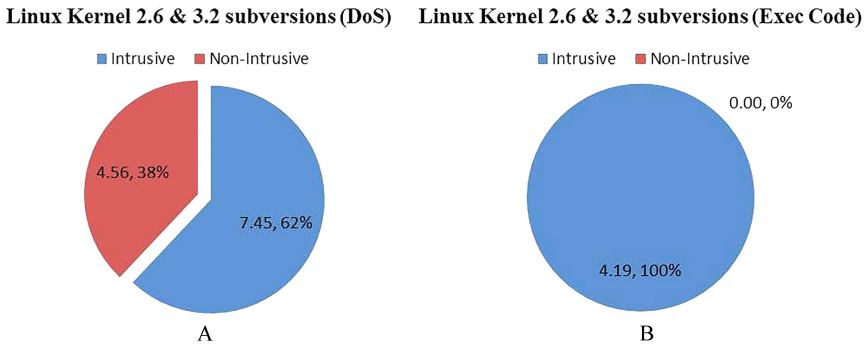

Fig. 9. (A) Linux Kernel 2.6 & 3.2 subversion (DoS) Year 2008–2017 Consolidated; (B) Linux Kernel 2.6 & 3.2 subversion (Exec Code) Year 2008–2017 Consolidated.

year 2008 till 2017 for both Microsoft 2008 Server and Linux 2.6 & 3.2 subversions. It highlights which software is more intrusive in nature with the associated rating. It also showcases the non-intrusive side of the vulnerability for both the software.

Figures 8 and 9 show the consolidated influence of threat in terms of vulnerability type for Microsoft 2008 Server and Linux Kernel 2.6 & 3.2 subversions. It highlights the intrusive and non-intrusive nature of vulnerability types in percentage with cumulative rating of the said software over a period of time.

As per the statistics shown, Windows 2008 Server exposes less vulnerable nature with 64% intrusive cases with a rating of 6.02 reported as compared to 62% intrusive cases with a rating of 7.45 reported for Linux Kernel 2.6 & 3.2 subversion, from year 2008 till 2017 for the vulnerability Denial of Service (DoS). For the vulnerability Exec Code, Microsoft 2008 Server has 100% intrusive nature with a rating of 9.11 as compared to Linux Kernel 2.6 & 3.2 subversion which has equally 100% intrusive nature with a rating of 4.19 which is comparatively on lower side.

So the conclusion is that for Denial of Service (DoS) vulnerability, Linux Kernel 2.6 & 3.2 subversion has to be worked upon more than Microsoft 2008 Server and the reverse applies for Exec Code vulnerability.

6. Conclusion and future work

The research work carried out shows that the security team may very well apply the Misuse Case Oriented Quality Requirements (MCOQR) metrics framework to estimate the influence of threat. For comparison purpose, two different types of operating system were chosen and the data was collected as per the proposed process specified in Vulnerability Identification and Misuse Case Framework.

Further as per the work flow of proposed MCOQR metrics framework, the data was gathered from the various tables creating and populated during the proposed process specified in Vulnerability Identification and Misuse Case Framework. The analysis was shown in a graphical form and the explanation and discussion was carried out to justify the purpose and objective of the proposed MCOQR Metrics framework, from influence of threat point of view which is one among the six parameters defined.

Future work may include collecting and testing data for more types of vulnerabilities. Another future works includes testing the influence of threat factor on a large sample of dataset. Another future work may include empirical testing on varieties of software applications to justify the claim of the proposed MCOQR metrics framework. For a comprehensive view of proposed MCOQR metrics framework, all the six parameters identified and defined needs to be dealt with at the same time, which is beyond the scope of this research paper. Further discussion shall be carried out in subsequent research papers.

Acknowledgments

First of all we would like to thank almighty GOD for everything we have and are today. We would like to thank Dr. Tarun Kumar Sharma, Amity University Rajasthan, India & Dr. Ajeet S Poonia, Cyber Security Expert, GCET, Bikaner, India for his positive approach towards the problems and their constructive suggestions. Special thanks to the experts from industry and academics for their contribution & approval about the concept and implementation of the proposed work. Last but not the least, we would like to thanks our parents and our children for their support and patience.

References

[1] A.A. Abdulrazeg, N.M. Norwawi and M. Basir, Extending V-model practices to support SRE to build secure web application, *Advanced Computer Science and Information Systems (ICACSIS), IEEE* (2014), 213–218.

[2] B. Bailey, Addressing software security, *NASA's IV&V Program Safety and Mission Assurance (SMA) Office Information Assurance/Cybersecurity Support*, 2015.

[3] C. Banerjee and S.K. Pandey, Software security rules, SDLC perspective, *International Journal of Computer Science and Information Security (IJCSIS)* **6**(1) (2009), 123–128.

[4] C. Banerjee, A. Banerjee and P.D. Murarka, Evaluating the relevance of prevailing software metrics to address issue of security implementation in SDLC, *International Journal of Advanced Studies in Computers, Science and Engineering* **3**(3) (2014), 18–25.

[5] C. Banerjee, A. Banerjee and S.K. Pandey, MCOQR (misuse case-oriented quality requirements) metrics framework, Problem Solving and Uncertainty Modeling Through Optimization and Soft Computing Applications, *IGI Global* (2016), 184–209.

[6] C. Banerjee et al., Proposed algorithm for identification of vulnerabilities & associated misuse cases using cvss, cve during sre phase, in: *International Conference on Soft Computing: Theories and Applications (SoCTA)*, AISC Series of Springer, 2017 (in publication).

[7] M. Busch, N. Koch and M. Wirsing, evaluation of engineering approaches in the secure software development life cycle, *Engineering Secure Future Internet Services and Systems*, Springer International Publishing (2014), 234–265.

[8] L. Chung and J. do Prado Leite, On non-functional requirements in software engineering, *Conceptual Modeling: Foundations and Applications* (2009), 363–379.

[9] W.C. Easttom II, Computer security fundamentals, Pearson IT Certification, 2016.

[10] A. Herrmann and B. Paech, MOQARE: misuse-oriented quality requirements engineering, *Requirements Engineering* **13**(1) (2008), 73–86.

[11] P.J. Houngbo and J.T. Hounsou, Measuring information security: understanding and selecting appropriate metrics, *International Journal of Computer Science and Security (IJCSS)* **9**(2) (2015), 108–120.

[12] S.R. Jan et al., Issues in global software development (communication, coordination and trust) a critical review, *International Journal of Scientific Research in Science, Engineering and Technology* **2**(2) (2016), 660–663.

[13] E. Letier, D. Stefan and E.T. Barr, Uncertainty, risk, and information value in software requirements and architecture, in: *Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014, pp. 883–894.

[14] J. Luftman, K. Lyytinen and T. ben Zvi, Enhancing the measurement of information technology (IT) business alignment and its influence on company performance, *Journal of Information Technology* (2015), 1–21.

[15] G. McGraw, Software security: building security, in: *Addison-Wesley Professional*, 2006.

[16] T. Okubo, K. Taguchi and N. Yoshioka, Misuse cases+ assets+ security goals, *Computational Science and Engineering, 2009. CSE'09. IEEE* (2009), 424–429.

[17] T.R. Peltier, Information security policies, procedures, and standards: guidelines for effective information security management, *CRC Press*, 2016.

[18] B. Penzenstadler et al., Safety, security, now sustainability: The nonfunctional requirement for the 21st century, *IEEE Software* **31**(3) (2014), 40–47.

[19] A.S. Poonia et al., Vulnerability identification and misuse case classification framework, Proceedings of Soft Computing: Theories and Applications SoCTA, Advances in Intelligent Systems & Computing Series of Springer, 2017. (in publication)

[20] C. Raspotnig, P. Karpati and V. Katta, A combined process for elicitation and analysis of safety and security requirements, *Enterprise, Business-process and Information Systems Modeling*, Springer Berlin Heidelberg (2012), 347–361.

[21] P. Salini and S. Kanmani, Model oriented security requirements engineering (MOSRE) framework for web applications, *Advances in Computing and Information Technology* (2013), 341–353.

[22] G. Santhosh Babu et al., Suraksha: A security designers' workbench, *Hack.in*, IIT Kanpur (2009), 59–65.

[23] G. Sindre and A.L. Opdahl, Eliciting security requirements with misuse cases, *Requirements Engineering* **10**(1) (2005), 34–44.

[24] R. Sinn, Software security technologies, *Cengage Learning*, 2015.

[25] S. Yahya et al., Capturing security requirements using essential use cases (EUCs), *Requirements Engineering*, Springer Berlin Heidelberg (2014), 16–30.