# A Phase-wise Review of Software Security Metrics

**Syed Anas Ansar, Alka and Raees Ahmad Khan**

**Abstract** Integrating security at each phase of the software Development Life cycle (SDLC) has become an urgent need. Moreover, security must not be overlooked at early phases of SDLC. This helps to minimize cost and efforts for later phase of the life cycle. In addition, software security metrics are the tools to judge level of security of software. Without the use of the metrics, no one can ensure the usefulness of any approach which claims to improve security of the software. The paper presents a phase-wise review of security metrics and the issues in their adaptation. Though there are security metrics available for each phase of the software development life cycle, their usefulness in the software industry or in research is in question without their validation. In addition, a concrete research is needed to develop security metrics at early phases of software development life cycle.

**Keywords** Software security · Security metrics · Software development life cycle

## 1 Introduction

Security refers to ensuring confidentiality, integrity, authenticity, availability, non-repudiation etc. [1]. It protects the system from unauthorized use, access, disclosure, and modification [2]. Nowadays news headlines are frightening us about data and information theft. This raises question about the status of security of data

S.A. Ansar (✉) · Alka · R.A. Khan
Department of IT, Babasaheb Bhimrao Ambedkar University,
Vidya Vihar, Raebareli Road, Lucknow 226025, India
e-mail: syed000anas@gmail.com

Alka
e-mail: alka_cs@jmuyahoo.co.in

R.A. Khan
e-mail: khanraees@yahoo.com



G.M. Perez et al. (eds.), *Networking Communication and Data Knowledge Engineering*, Lecture Notes on Data Engineering and Communications Technologies 4, https://doi.org/10.1007/978-981-10-4600-1_2

stored, processed through computers, and shared through Internet. The hackers/attackers cannot only be blamed for these incidences, designers and developers of the software are equally responsible. Even after so many life threatening security incidents, it is still treated as an afterthought while developing the software. Security features are often sprayed on the completely developed software [1, 3]. As a result, ensuring software security has become a battle. A hacker tries to find and exploit security holes present in the software. He does not create security holes on his own. Hence, presence of even single security hole may be responsible to exploit the software's security completely. The irony is that the security practitioners can never be confident that they have found and patched all the security holes. Hence, security has become a great challenge.

Instead of spraying security features on the security holes found during penetration, security must be addressed during each phase of software development life cycle (SDLC) [1]. Addressing security at each phase of SDLC is termed as software security. Hence, it is the idea of developing software so that it can provide the required function even when it is attacked [3]. In addition, while improving security of the software under development, it is very important but difficult to judge the level of security. Now, the role of security metrics comes into the picture. Without the help of the good security metrics and systematic approaches, it is difficult to assure security level of software. Metrics is a measurement standard which defines what is to be measured and how and helps the security practitioners to manage the product efficiently [4]. Security metrics is the powerful tool that helps security practitioners to integrate security features into their system [5–7]. Nowadays metrics are gaining much consideration because with the help of the data obtained from them decision can be taken accordingly. They assist practitioners to meet their goal for secure software development [1, 7, 8]. Rest of the paper is organized as: next section discusses about the related work in the area. Section 3 presents phase-wise systematic review of security metrics. Section 4 discusses about major findings, and finally, the paper is concluded at Sect. 5.

## 2 Related Work

The research in the area of software security has been going on worldwide. Following is some pertinent research work in the area. B. M. Alshammari et al. (2016) have reviewed different procedures for building more secure systems. They have firstly reviewed various design principles. The authors have also explained some of the existing work on software quality including software security metrics and have also compared different security metrics for building secure systems. Finally, they have discussed about refactoring and its influence on security with the demonstration showing that the refactoring is used for enhancing the quality of program. In addition, they have suggested that these all can be used as components of secure system architecture. This study may be used as guidance for building and recognizing secure software [9].

D-E. Lim and T-S. Kim (2014) have modelled the discovery and removal of software vulnerability based on queuing theory. Vulnerability has been classified into groups on the basis of their severity calculated by Common Vulnerability Scoring System (CVSS). In this approach, three parameters have been used. For each class, the queuing model has been used for obtaining waiting time and number of vulnerabilities present in a queue. They applied the Takagi equation for obtaining average waiting time of an arbitrary vulnerability, and then, they finally applied Little's Law to Takagi equation for obtaining the number of unfixed vulnerabilities. The system risk is measured by counting the number of unfixed vulnerabilities [10]. A. A. Abdulrazeg et al. (2012) have developed security metrics to improve misuse case model for discovering and fixing defects and vulnerabilities. The proposed security metrics indicates the possibility of security defects. The metrics have been developed using goal question metric approach (GQM). The presented metrics consist of two main goals. For achieving first and second goal, they have developed security metrics on the basis of anti-pattern and web application security risk OWSAP top 10-2010 respectively. The proposed work is significant to remove modelling defects and to improve security use cases before these defects move to the next stage of development life cycle [11].

S. Islam and P. Falcarin (2011) have identified security requirements for measuring software security goals with the help of risk management process. They evaluated these software security requirements for measuring software security on the basis of GQM approach. GQM approach is a procedure that confers a framework for defining and explaining metrics. The approach includes: purpose (Why), object, issue, perspective, viewpoint, environment (context), and when. They partially followed security quality requirement engineering (SQUARE) methodology for identifying requirements. SQUARE methodology provides a means of eliciting, classifying, and emphasizing security requirements for information technology system and application. They followed ISO 17799:2005 standard as a baseline for developing the metrics [2]. H. C. Joh and Y. K. Malaiya (2010) proposed a framework in which they combine stochastic model (as vulnerability life cycle) and Common Vulnerability Scoring System (CVSS) metrics for the evaluation of risk. They defined risk from the point of view of the vulnerability life cycle, considering the probabilities, exploitation of software vulnerability in a system, and the impact of its exploitation. They have first considered the evaluation of the risk induced by a single vulnerability. They have also generalized the approach to include all the potential vulnerabilities in software [12].

R. M. Savola (2009) introduced a modern method for the development of security metrics based on threat, security requirements, and decomposition of security goals. In the proposed method, he has used some process for security metric development. Firstly, threat and vulnerability analysis has been carried out, and then, prioritized security requirements have been declared. By identifying basic measurable components (BMC), the author has generated measurement architecture

and had selected the BMCs to be used as the basis for detailed metrics. Finally, he validated the security metrics. The core activity of the proposed method is to decompose the security requirements [13]. J. A. Wang et al. (2009) developed security metrics on the basis of representative weaknesses of the software [14]. M. A. Hadvi et al. (2008) proposed a method for early mitigation of software vulnerabilities for the secure software development. They have selected the most common 23 vulnerabilities and defined them. They have analyzed reasons for their presence in a phase. They provided countermeasures for their avoidance and mitigation through design level activities (13 design activities) as well as implementation level activities (19 activities). They finally mapped these vulnerabilities to the given activities. The mapping would mitigate the specific vulnerabilities and would provide a better insight of introduction of vulnerabilities. This may help security practitioners to develop secure software [15].

Shirley C. Payne (2007) has worked on developing a security metrics program and has proposed a seven-step methodology that may guide development of simple metrics programs. They have advised managers to take help from existing easy, cheap, and fast measures. The important thing they have concluded is that the metrics generated should be useful enough to make advancement in the overall security program. The purpose of this guide is to provide an overview of the current state of security metrics [16]. O.H. Alhazmi et al. (2007) have proposed a new metric called vulnerability density. It can be used to compare the software systems. They have defined vulnerability density metric as the number of vulnerabilities per unit size of code. By using vulnerability density as a parent metric, they have coined a set of metrics called as known vulnerability density, residual vulnerability density. They defined known vulnerability density as the number of known vulnerabilities in the unit size of code and the residual vulnerability density metric as the vulnerabilities density minus the known vulnerabilities density [17].

## 3 Phase-wise Security Metrics

The field of security metric is comparatively new [18]. Many software industries, researchers, and practitioners have developed security metrics. These metrics are related to different phases of the software development life cycle. This paper presents a systematic review of the security metrics available for different phases of software development life cycle. The research has performed phase-wise review of available security metrics. The security metrics for requirement Phase, design phase, coding/implementation phase, testing phase, and maintenance phase are presented in the Tables 1, 2, 3, 4, and 5, respectively.

**Table 1** Security metrics for requirement phase

| Security metrics | Definition/purpose |
|---|---|
| Security Requirements Recorded Deviations (SRRD) | This metric is used to provide the number of deviations from security requirements [19] |
| Security Requirements stage Security Errors (SRSE) | It provides the number of security errors that are the result of incomplete or incorrect security requirements [19] |
| Security Requirements gathering Indicators (SRI) | It provides indicators on requirements gathering and analysis phase, which explains the impact of security requirements on the number of security breaches/violations [19] |
| Total number of security requirement (Nsr) | It aims to measure the number of security requirements identified/found during analysis phase [20] |
| Ratio of security requirements (Rsr) | This provides the ratio of requirement which has direct impact on security to the total number of requirement. $R_{sr} = \frac{\lvert SR \rvert}{\lvert R \rvert}$ Where SR is the set of security requirement and R is the set of all the requirement of the system [20] |
| Number of omitted security requirements (Nosr) | It measures the number of security requirement that has been not considered during the analysis phase [20] |

**Table 2** Security metrics for design phase

| Security metrics | Definition/purpose |
|---|---|
| Vulnerable Association of an Object Oriented Design(VA_OOD) | It is calculated as the ratio of summation of vulnerable association of each class to the total number of vulnerable classes in the design [21] $VA\_OOD = \frac{\sum Vulnerable\ Association\ of\ Each\ Class}{Total\ Number\ of\ Vulnerable\ Classes\ in\ the\ Design}$ |
| Security Requirements Statistics ($SR_s$) | If NSRD is number of security requirements considered for design, then $SR_s$ is given as $SR_s = \frac{NSRD}{NSRG}$ Where NSRG is the number of security requirement gathered in that design [19] |
| Number of Design stage Security Errors (NDSE) | It calculates the number of security errors due to the design stage [19] |
| Composite-Part Critical Classes (CPCC) | $CPCC(D) = 1 - \left(\left\lvert \frac{CP}{CC} \right\rvert\right)$ Where CC is critical classes in design D and CP is composed-part critical classes in the same design [22] |
| Critical Class Coupling (CCC) | It is calculated as the ratio of the number of all classes' linked with classified attributes to the total number of possible links with classified attributes in a given design [22]. $CCC(D) = \frac{\sum_{j=1}^{ca} \alpha(CA_j)}{(\lvert C \rvert - 1) \times \lvert CA \rvert}$ |
| Critical Class Extensibility (CCE) | It can be calculated as the ratio of the number of non-finalized classes in a design to the critical classes in that design [22]. $CCE(D) = \frac{\lvert ECC \rvert}{\lvert CC \rvert}$ |

(continued)

**Table 2** (continued)

| Security metrics | Definition/purpose |
|---|---|
| Classified Methods Extensibility (CME) | It is the ratio of the number of non-finalized classified method to the total number of classified methods in a design [22]. $CME(D) = \frac{\lvert ECM \rvert}{\lvert CM \rvert}$ |
| Critical Super-classes Proportion (CSP) | It measures the ratio of the number of critical super classes to the total number of critical classes in an inheritance hierarchy [22]. $CSP(H) = \frac{\lvert CSC \rvert}{\lvert CC \rvert}$ |
| Classified Methods Inheritance (CMI) | This aims to measure the ratio of the number of classified methods which can be inherited in a hierarchy to the total number of classified methods in that hierarchy [22]. $CMI(H) = \frac{\lvert MI \rvert}{\lvert CM \rvert}$ |
| Critical Design Proportion (CDP) | It is calculated as the ratio of the number of critical classes to the total number of classes in a design [22]. $CDP(D) = \frac{\lvert CC \rvert}{\lvert C \rvert}$ |
| Coupling Induced Vulnerability Propagation Factor(CIVPF) | It is defined as the summation of induced vulnerability propagation from a root vulnerable class to others in a design to the total number of classes in that design. $CIVPF = \frac{\sum_{i=1}^{p} Li}{N}$ Where N is the total number of classes, and Li (i=1,…..p) is the total number of coupling induced vulnerability propagation from a root vulnerable class Ci to the others [23] |
| Classified Instance Data Accessibility (CIDA) | It is the ratio of the number of classified instance public attributes to the total number of classified attributes in a class [24]. $CIDA(C) = \frac{\lvert CIPA \rvert}{\lvert CA \rvert}$ |
| Classified Class Data Accessibility (CCDA) | It is computed as the ratio of the number of the classified class public attributes to the number of classified attributes in a class [24] $CCDA(C) = \frac{\lvert CCPA \rvert}{\lvert CA \rvert}$ |
| Classified Operational Accessibility (COA) | It is defined as the ratio of the number of classified public methods to the number of classified method in a class [24]. $COA(C) = \frac{\lvert CPM \rvert}{\lvert CM \rvert}$ |
| Classified Methods Weight (CMW) | It is calculated as the ratio of the number of classified methods to the total number of methods in a given class [24]. $CMW(C) = \frac{\lvert CM \rvert}{\lvert M \rvert}$ |
| Number of design decisions related to security (Ndd) | It aims to measure the number of design decisions that describes security requirements of the system [20] |
| Ratio of design decisions (Rdd) | This can be computed as the ratio of design decisions related to security to the total number of design decisions [20]. $\text{Rdd} = \frac{Ndd}{Nd}$ |

**Table 3** Security metrics for coding/ implementation phase

| Security metrics | Definition/purpose |
|---|---|
| Percent of Security Coding Aspects (PSCA) | This indicates the percentage of security aspects considered during coding according to design [19] |
| Percent use of Coding Standard (PCS) | It indicates the use of coding standards for secured development and shall be supported in identifying the consideration of security standards during code implementation [19] |
| Number of Security Errors (NSE) | This metric is used to indicate the flaws expressed as the sum of coding errors and also the errors from other library code [19] |
| Stall Ratio (SR) | This metric aims to measure the ratio of the number of lines of non-progressive statements in the loop to the total number of lines in a loop [25] $SR = \frac{Lines\ of\ non-progressive\ statements\ in\ a\ loop}{Total\ lines\ in\ the\ loop}$ |
| Coupling Corruption Propagation (CCP) | It is defined as the number of child methods called with the parameter(s) that are based on the parameter(s) of the original invocation [25] |
| Critical Element Ratio (CER) | This aims to provide the ratio of critical data elements in an object to the total number of elements in the object. It measures the ways a program can be infected by malicious inputs [25]. $CER = \frac{Critica\ Data\ Elements\ in\ the\ Object}{Total\ Number\ of\ Elements\ in\ the\ Object}$ |
| Precision | It relates the true defective components to the total number of components predicted as defective. $Precission = \frac{TP}{TP + FP}$ Where TP is true positive and FP is false positive [26] |
| Recall | It is defined as the ratio of true defective components to the total number of defective components. $Recall = \frac{TP}{TP + FN}$ Where TP is true positive and FN is false negative [26] |
| F-measure | It is used to combine the precision and recall as a harmonic mean [26]. $F - measure = \frac{2 \times Recall \times Precision}{Recall + Precision}$ |
| Accuracy | It is used to measure the overall accuracy of the prediction [26]. $Acc = \frac{TP + TN}{TP + TN + FP + FN}$ |
| Ratio of implementation errors that have direct impact on security(Rserr) | This provides the ratio of the number of errors that have a direct impact on security to the total number of errors in the implementation of the security [20]. $\text{Rserr} = \frac{Nserr}{Nerr}$ |

**Table 4** Security metrics for testing phase

| Security metrics | Definition/purpose |
|---|---|
| Security Requirements Considered for Testing (SRT) | It can be indicated by the ratio of the security requirement tested, and the number of security requirement gathered (NSRG) [19] |
| Process Effectiveness (PE) | It can be represented by the ratio of number of security vulnerability discovered ($N_{VD}$) to the number of modules undergone security testing ($M_{ST}$) [19]. $PE = \frac{N_{VD}}{M_{ST}}$ |
| Security Testing Ratio (STR) | This indicates the ratio of modules undergone security testing to the total number of modules [19]. $STR = \frac{M_{ST}}{M}$ |
| Ratio of security test cases that fail (Rtcp) | This aims to provide the ratio of numbers of test cases that fails to detect implementation errors to the number of test cases specially designed to detect the security issues [20]. $Rtcp = \frac{\lvert TF \rvert}{\lvert TP \rvert + \lvert TF \rvert}$ |

**Table 5** Security metrics for maintenance phase

| Security metrics | Definition/purpose |
|---|---|
| Mean Time to Complete Security Changes (MTCSC) | It is estimated by the number of security failures and mean time taken to repair the flaws [19]. MTCSC = MTTSF + MTTR |
| Percent of Changes with Security Exceptions (PCSE) | It is measured by the ratio of counts of completed changes with security exceptions and completed changes multiplied by 100 [19] |
| Ratio of patches issued to address security vulnerability (Rp) | This aims to provide the ratio of a number of patches that are released to address security vulnerability to the total number of patches of the system [20]. $Rp = \frac{Nsp}{Np}$ |
| Number of security incidents reported (Nsr) | It aims to measure the number of incidents that are concerned with security [20] |

## 4 Major Findings

A literature survey of software security metrics indicates the immaturity of the area. Following are some finding obtained during the survey:

- It is found that no comprehensive evaluation scheme is devised for these metrics.
- Most of the metrics developed for the early phases of SDLC are vague in nature. Hence, there is a need to develop security metrics for successful implementation of security in those phases.
- Almost all the metrics have been implemented only on small data sets.
- There is need to conduct more experiments to extract concrete conclusion about the implication of the proposed metric values.

- Proper validation of most of these metrics has not been done.
- There is no real-life implementation of these metrics.
- There is a need to examine the role of CASE tools while using these metrics.
- Proper evaluation of usefulness of these security metrics is also required.
- There is a need to develop a security metric development framework for unified development and evaluation of metrics.

In SDLC, design phase is the most convenient phase for consolidating security decisions. Unfortunately, there is no efficient methodology or tool exists to address security issues at this phase. Almost negligible work has been reported to address security at this phase. So there is need to develop an appropriate framework for metric development at this phase. The framework may assist in developing and validating security metric.

## 5 Conclusion

Rapid development in issues related to security has facilitated the development of security metrics. Today, software security measurements have become a genuine demand in software industry. Metric is a measurement standard which defines what is to measure and how and helps the security practitioners to manage product efficiently. It also provides quantitative as well as objective basis for security assurance. The metrics are supposed to be the foundation of secure development of software. In this paper, a number of software security metrics for different phases of SDLC have been reviewed. The developers of the metrics have claimed that the metrics not only allow the security practitioners to ensure security of the software, but also indicate where the security issue or vulnerability occurs. The available approaches also ensure security hole-free software design and coding. Overall, actual usefulness of the metrics is always in question until it is validated or implemented on industrial data. In absence of a pertinent ready to use framework for development of security metrics to be used in early stage of software development life cycle, there is need to develop a security development framework to guide the development of a minimal set of the metrics or integrated metrics.

**Acknowledgements** This work is sponsored by UGC-MRP, New Delhi, India under F. No. 43-391/ 2014 (SR)

## References

1. McGraw, G.: “Software Security”:Building Security In. (Addison-Wesly, 2006)
2. Islam, S. Falcarin, P.: Measuring Security Requirements for Software Security. In 10th International Conference on Cybernetic Intelligent Systems (CIS), ISBN 978-1-4673-0687-4, DOI 10.1109/CIS.2011.6169137, pp. 70-75, IEEE, (2011)

3. McGraw, G., Potter, B.: Software Security Testing [J]. IEEE Security & Privacy, 2(5):81–85, (2004)
4. Herrmann, D.S.: Complete Guide to Security And Privacy Metrics. Auerbach Publications, ISBN: 0-8493-5402-1. (2007)
5. Swanson, M., Bartol, N., Sabato, J., Hash, J., and Graffo, L.: Security Metrics Guide For Information Technology Systems. NIST Special Publication 800–55, National Institute Of Standards And Technology, (2003)
6. Chaula, J. A., Yngstrom, L., and Kowalski, S.: Security Metrics And Evolution Of Information Systems Security. In Proc. of the 4th Annual Conference on Information Security For South Africa, (2004)
7. Payne, S. C.: A guide To Security Metrics. (2001)
8. Goodman, P.: Software Metrics: Best Practices For Successful IT Management. (2004)
9. Alshammari, B., Fridge, C., Corney, D.: "Developing Secure System: A Comparative Study of Existing Methodologies". Lecture Notes on Software Engineering, vol.2, no.2, may 2016, pp: 139–146, doi: 10.7763/LNSE.2016.V4.239
10. Lim, DE., Kim, TS.: Modelling Discovery and Removal of Security Vulnerabilities in Software System Using Priority Queuing Models. Journal of Computer Virology and Hacking Techniques, Springer, 10: 109–114,DOI 10.1007/s11416-014-0205-z, (2014)
11. Abdulrazeg, A. A., Norwani, N. Md., Basir, N.: Security Metrics to Improve Misuse Case Model. International conference on Cyber Security, Cyber Warfare and Digital Forensic, ISBN 978-1-4673-1425-1, Doi 10.1109/CyberSec.2012.6246129, pp. 94–99, IEEE, (2012)
12. Joh, HC., Malaiya, Y. K.: A Framework for Software Security Risk Evaluation Using the Vulnerability Lifecycle And CVSS Metrics. Proc. International Workshop on Risk and Trust in Extended Enterprises, pp. 430–434 (2010)
13. Savola, R. M.: A security Metrics Development Method for Software Intensive Systems. Advances in Information Security and its Application, Communications in Computer and Information Science, 2009, Volume 36, pp. 11-16,Springer, (2009)
14. Wang, J. A., Wang, H., Guo, M., Xia, M.: Security Metrics for Software Systems. In the Proc. Of ACMSE, March 19–21, Clemson, SC, USA, (2009)
15. Hadvi, M. A., Sangchi, H. M., Hamishagi, V. S., Shirazi, H.: Software Security; A Vulnerability-Activity Revisit. Third International conference on Availability, Reliability, and Security, ISBN 978-0-7695-3102-1, Doi10.1109/ARES.2008.200 IEEE, (2008)
16. Payne, S. C.: "A Guide to Security Metrics". SANS Institute 2007. Available at: www.sans.org/reading_room/whitepapers/auditing/55.php. Last visit Aug. 22 2016.
17. Alhazmi, O. H., Malaiya, Y. K., Ray, I.: Measuring, Analysing, and Predicting Security Vulnerabilities in Software Systems. Computers and Security Journals, pp. 219–228, (2007)
18. Manadhata, P. K and Wing, J. M.: An Attack Surface Metric. Technical Report. School of Computer Science, Carnegie Mellon University (CMU). CMU-CS-05-155, (2005)
19. Jain, S., Ingle, M.: Security Metrics and Software Development Progression. Journal of Engineering Research and Applications, ISSN: 2248–9622, Vol. 4, Issue 5 (Version 7), pp. 161–167, (2014)
20. Sultan, K., En-Nouaary, A., H-Lhadj, A.: Catalog for Assessing Risks of Software Throughout the Software Development Life Cycle. In the Proc. of International Conference on Information Security and Assurance, pp. 461–465, IEEE, (2008)
21. Agarwal, A., Khan, R. A.: Assessing Impact of Cohesion on Security- An object Oriented Design Perspective. vol 76, No. 2, pp. 144–155, Pensee Journal, (2014)
22. Alshammari, B., Fridge, C., Corney, D.: Security Metrics for Object-Oriented Designs. Proc. 21st Australian software Engineering Conference, IEEE Press, pp. 55–64, Doi:ieeecomputersociety.org/10.1109/ASWE(2010)
23. Agarwal, A., Khan, R. A.: Role of Coupling in Vulnerability Propagation Object Oriented Design Perspective. Software Engineering: An International Journal (SEIJ), Vol. 2, No. 1, pp. 60–68, (2012)

24. Alshammari, B., Fridge, C., Corney, D.: Security Metrics for Object-Oriented Class Designs. In proceedings of the Ninth International Conference on Quality software (QSIC), IEEE, (2009)
25. Chowdhury, I., Chan, B., Zulkerine, M.: Security Metrics for Source Code Structures. In Proceedings of the Fourth International Workshop on Software Engineering For Secure Systems, ACM, pp. 57–64. (2008)
26. Nguyen, V. H., Tran, L.M.S.: Predicting Vulnerable Software Components with Dependency Graphs.In Proceedings of the 6th International Workshop on Security Measurements and Metrics, ISBN: 978-1-4503-0340-8, Doi: 10.1145/1853919.1853923, (2010)

http://www.springer.com/978-981-10-4599-8

Networking Communication and Data Knowledge Engineering
Volume 2
Perez, G.M.; Mishra, K.K.; Tiwari, S.; Trivedi, M.C. (Eds.)
2018, XX, 262 p. 79 illus., Softcover
ISBN: 978-981-10-4599-8