

ANALYSES THE PROACTIVE SECURITY METRICS FOR SOFTWARE DEVELOPMENT PHASES

Padala Kavitha,

Research Scholar, Computer Science and Engineering, OPJS University, Churu, Rajasthan.

Dr. Vijay Pal Singh,

Assistant Professor, Computer Science and Engineering, OPJS University, Churu, Rajasthan.

Abstract

To understand the significance of security metrics, we reviewed the available product and process metrics for various stages of SDP. Further, we have proposed security metrics and effectiveness factor for software development stages that may help in controlling the security flaws. The process metrics based on different software development stages and the product metrics are illustrated, We propose proactive security metrics required during the software development process that focuses on the security issues of its various stages. The effectiveness factor of security metrics on various development stages are presented. Three case studies illustrating the security metrics and effectiveness factors are presented.

1. Introduction

Most of the security vulnerabilities are result of flaws that are introduced inadvertently during design and development of a software system. To decrease software vulnerabilities, the overall defect content must be reduced. It is important to measure the vulnerabilities in order to mitigate the security defects and produce more secured product. The primary goal of metrics is to quantify data so as to facilitate insight and provide degree of trustworthiness. Metrics help the project management team to effectively manage the software development process as well as judge the product. It expedites decision support, especially assessment and prediction of security aspects of the software. Metrics aims in early detection and correction of the security flaws of the software.

Security metrics are quantitative indicators of the security aspect of the software that assesses the quality related imperfections of the software. The security of the software can be assessed through product and process level metrics. Most of the security metrics analyzes security at the system level. A number of security metrics evaluate security consideration during the various stages of software development. Several other security metrics are aimed towards specific stage of development process of software such as design, coding and testing. It has been observed that the metrics describe either the security aspect of the software or assesses security risks during Software Development Process (SDP) but the realization is not at a glance.

2. Product Metrics

Product metrics describe the product quality characteristics such as security. Security metrics have been defined on the basis of vulnerabilities and have been proposed on the basis of Common Vulnerabilities and Exposures (CVE), and Common Vulnerability Scoring System (CVSS). CVE is an industry standard for vulnerability and exposure names, and CVSS is a vulnerability scoring system designed to provide an open and standardized method for rating software vulnerabilities. Metrics defined is representative weakness of software i.e. those weaknesses that lead most of the vulnerabilities to be exploited by the attackers and are given below:

Security metrics, SM(s), for software s is given as

$$SM(s) = \sum_{i=1}^m (P_i \times W_i) \quad \text{----- 1}$$

where W_i ($i=1, 2, \dots, m$) is the severity of weakness in the software s and P_i ($i = 1, 2, \dots, m$) is risk of the corresponding weakness. W_i is the severity of the weakness and is given by

$$W_i = \frac{V_i}{k} \quad \text{----- 2}$$

$$W_a = \sum_{i=1}^K$$

Where K is number of vulnerabilities of weakness W with corresponding base scores as V_i . In Equation 1, P_n can be expressed as percentage and is given by occurrence of each weakness in the overall weakness.

$$P_n = \frac{R_n}{\sum_{i=1}^m R_i} \quad \text{----- 3}$$

In Equation 3, R_n is the frequency of occurrences of each representative weakness. If K is the number of weaknesses during M months, R_n can be expressed as

$$R_n = \frac{K}{M} \quad \text{----- 4}$$

SM(s) can range from 0 to 10 if following formula hold true for P_n .

$$P_n = \frac{1}{m} \quad \text{----- 5}$$

$$\sum_{i=1}^n$$

Thus, software security metrics have been defined on the basis of representative weakness and can be calculated using Equations (1-5). Common Criteria (CC) defines seven assurance levels for a product varying from EAL1 to EAL7 that specifies how thoroughly the product is tested and ranks assurance on evaluated products. Higher assurance level only means that the product has undergone more number of security tests. NIST presents system level controls addressing the information security program which includes access control, awareness and training, audit and accountability, etc. It provides some of the measures for audit processing failures that include software/ hardware errors, failures in the audit capturing mechanisms, and audit storage capacity being reached or exceeded. CIS security outcome and practice metrics define 20 metrics under six business functions. For example, Risk Assessment Coverage and Security Testing Coverage metrics assess the application security. The change management metrics include Mean time to complete changes, Percent of changes with security reviews, Percent of changes with security exceptions. Vulnerability management metrics include Mean-time to mitigate vulnerabilities, Number of known vulnerability incidences etc.

3. Proposed Security Metrics

It can be revealed that most of the researchers have developed metrics specific to some software development stage, while others addressing SDP have not provided enough details regarding implementation of the metrics. Also, the metrics developed have not considered the reasons for insecurity. In this section, we propose security metrics that analyzes the security efforts during various software development stages. We denote development stages in terms of phases that stand for requirements gathering and analysis, software design, coding, system integration and testing, operations and maintenance, and documentation respectively.

Requirements Gathering and Analysis

Gathering security requirements acts as foundation for secured software. Requirements gathering stage focuses on gathering security requirements from various stakeholders along with the other functional requirements. The security requirements can be gathered using Software Security Requirements Gathering Instrument (SSRGI) from the various stakeholders by the help of misuse cases, attack trees etc. as mentioned in Chapter 6. Most of the metrics defined for this stage are direct measures and can be considered as internal performance indicators for the requirements gathering team. The various metrics proposed are discussed below:

Number of Security Requirements Gathered (NSRG)

The number of security requirements gathered during the requirements gathering phase can be measured by metric *NSRG*. It consists of security requirements gathered from stakeholders as well as through the use of various tools. Requirements can be gathered using misuse cases, attack trees, best practices, security drivers etc. Some of the security drivers include Sarbanes Oxley, HIPPA etc. The requirements may relate to authentication and authorization, password, inactive sessions etc. and can be gathered from managers and other stakeholders (refer Chapter 6). *NSRG* may indicate the importance of security requirements during Phase 1. Let *TR* and *SR* denote security requirements be gathered with the help of tools and from stakeholders respectively, then

$$NSRG = SR + TR$$

For example, *SR* and *TR* be 6 and 2 respectively for a web-based application, it implies that most of the requirements have been gathered with the help of stakeholders. The value of *NSRG* focuses on the number of security requirements gathered with the help of tools and stakeholders.

Security Requirements Recorded Deviations (SRRD)

It describes the deviations from security requirements. The deviations can be measured after designing the software system based on the requirements specification. If *SRRD* is 0, then all the security requirements have been incorporated in the software product.

Security Requirements stage Security Errors (SRSE)

It specifies the security errors encountered during requirements gathering. It measures errors as a result of incorrect or incomplete security requirements gathered during this stage. If for a web-based project the value of *SRSE* is 1, it indicates the number of security related errors due to security requirements.

Security Requirements Gathering Indicators (SRI)

Indicators on Requirements gathering and analysis stage or *SRI* explain the impact of security requirements on number of security breaches. For example, in a client/ server system, the value of *SRI* as 3 provides an indicator regarding number of security breaches in first year of deployment.

4. Software Design

The design identifies the work that software can perform. It mainly illustrates the various components of the software and its interrelationships with each other and the surroundings. Even when the designers are proactive, the software may not be fully secured. The design should be able to reduce the attack surface of the software making it attack resistant and tolerant. In the software design stage, the requirements gathered must be considered for design thereby helping to produce software as per specifications. Requirements and design are indispensable phases of software development. Requirements specifications are considered for design proposal whereas analysis of design generates the need for further requirements.

4.1. Security Requirements Statistics (SRS)

Security Requirements Statistics or *SRS* indicates the percent of security requirements gathered reflected in the design stage. If *NSRD* is Number of Security Requirements considered for Design, *SRS* can be defined as the ratio of *NSRD* and *NSRG* expressed as percentage. The value of *SRS* as 100% indicates that all of the gathered security requirements have been considered for the design. If value of *NSRD* and *NSRG* is 5 and 8 respectively for a web-based system, then *SRS* implies 62.5% security consideration during design.

5. Effectiveness Factor Of Security

In this section, we attempt to develop effectiveness factor using the metrics as proposed at each phase of software development. The effectiveness factor can help to judge the security efforts during various development phases. Based on security metrics, we collected data for fifteen live projects. These projects have been developed using platforms such as Java, PHP, .NET, MySQL, C#, AJAX, Perl, Visual Basic, ASP .NET 4.0, Java Script, SQL Server, Apache, Red Hat Linux, MSDE, JQuery, Microsoft SSRS and Unix shell script and are based on various network design such as client/ server system, web-based, static web sites, mobile and desktop based systems. Also, the size of projects here range from small to very large.

Table 1: Dependency of Metrics at Different Phases of SDP

S.No	Phases Metrics	I	II	III	IV	V	VI
		1	NSRG	✓			
2	SRRD	✓					
3	SRSE	✓					
4	SRI	✓					
5	SRs	✓	✓				
6	DTTE		✓				
7	NDSE		✓				
8	PSCA		✓	✓			
9	PCS			✓			
10	NSE			✓			
11	SRT	✓			✓		
12	PE				✓		
13	STR				✓		
14	MTCSC					✓	
15	PCSE					✓	
16	RVA					✓	
17	Rsc					✓	
18	NSCM						✓

Phase I: Factors

The security efforts at requirements gathering phase can be judged in two stages. At first stage, we identify the existence of relationship between each pair of metrics of the all development phases. At second stage, the effectiveness factor for the software development stages is being developed.

Stage I – Relationship among metrics

We propose null hypothesis stating that each pair of metrics, do not belong to the same sample unit. The null hypothesis can be stated as follows:

H01: *SRRD* is unrelated to *NSRG*. H02: *SRSE* is unrelated to *NSRG*.

H03: *SRI* is unrelated to *NSRG*. H04: *SRRD* is unrelated to *SRSE*. H05: *SRRD* is unrelated to *SRI*. H06: *SRSE* is unrelated to *SRI*.

The null hypothesis is evaluated using paired t-test as follows:

$$t = \frac{X1 - X2}{S \sqrt{\frac{1}{n1} + \frac{1}{n2}}} \text{-----6}$$

Where,
$$S_{x_1, x_2} = \sqrt{\frac{(n_1 - 1)S_{x_1}^2 + (n_2 - 1)S_{x_2}^2}{n_1 + n_2 - 2}}$$

Here, X_1 and X_2 represents the means of the metrics of sample units. The number of sample units are $n_1 = n_2 = 15$. The paired t-test identifies if the sample means differ significantly or not. The level of significance is evaluated at 5%.

We applied paired t-test among the metrics of requirements gathering stage to test the hypotheses. The t-test results reject the null hypotheses H_{01} , H_{02} and H_{03} with t-values 3.2248, 3.1873 and 3.0854 respectively indicating that the metrics belong to the same sample unit and may be related to each other. Thus, *SRRD*, *SRSI*, and *SRI* are related to *NSRG*.

Stage II - Effectiveness Factor

Let us consider that there exists independent metric X and n number of dependent metrics as y_1, y_2, \dots, y_n . Let Y be the sum of all n metrics representing total effect by independent metric X . We use Least Square Method to identify the relationship.

$$Y = C - \alpha \cdot X \tag{7}$$

where, C is constant and is calculated as the sum of all intercepts generated for each pair of metrics indicating relationship. α represents the effectiveness factor of X on Y and can be computed using Equation 7.7 as

$$\alpha = (C - Y) / X \tag{8}$$

The domain of metrics may consist of independent and dependent metrics. Based on the practicability, dependent metrics may be *SRRD*, *SRSE* and *SRI* where as *NSRG* may be considered as independent metric X .

Applying Equation 7, the value for constant C for metrics pair *NSRG* and *SRRD*, *NSRG* and *SRSE*, and *NSRG* and *SRI* are 1.3671, 1.8423, and 2.0327 respectively. Thus, using Equation 7.8, the combined effect of independent variable X (i.e. *NSRG*) on dependent variable Y (i.e. *SRRD*,

SRSE and *SRI*) can be given as $\alpha = (5.2421 - Y) / NSRG$ $\tag{9}$

Where $NSRG \neq 0$

In Equation 9, the constant C is evaluated as combined effect of X on Y . Special Cases of Effectiveness Factor

$\alpha = -1$: Effectiveness -1 implies that number of dependent metrics are large enough such that $Y - C = X$. This also shows that the entire software development process may have not considered security.

$\alpha = 0$: Effectiveness 0 implies that sum of dependent metrics equals C while X can assume any value. It shows that even though security requirements have not been gathered, the security deviations from requirements and errors are recorded, leading to trivial situation.

$\alpha = 1$: Effectiveness can be 1 when $Y = C - X$. This may lead to trivial case when the numbers of security requirements gathered are high as Y may acquire negative value.

$\alpha > 1$: The values greater than 1 implies that it may be difficult to judge the effectiveness of the security.

Phase II: Factors

The null hypothesis H_{07} states that the metrics *DTTE* is unrelated to *NDSE* i.e., there is no significant difference between the means of *DTTE* and *NDSE*. The null hypothesis is evaluated using t-test (Equation 7.6) and is rejected at 5% level of significance ($t = 3.268$) thereby indicating relation between the two. The metrics pair is further evaluated resulting in negative correlation of -0.7.

As evident from the discussion in Section 7.3.2, the metrics *DTTE* and *NDSE* are inversely proportional to each other.

$$NDSE = \beta / DTTE$$

$$\beta = NDSE \cdot DTTE \text{ ----- } 10$$

Here, β denotes the effectiveness of design stage where $\beta \geq 0$. $\beta = 0$ implies that design stage security errors are nil. From Equation 10, it is evident that β shall lie between 0 and 1. The value for β can be interpreted as highly effective (0-0.25), moderately effective (0.26-0.50), effective (0.51-0.75), ineffective (0.75-1.00) and very ineffective (>1.00) security considerations during design phase.

Phase III: Factors

The secure coding may reduce the number of errors if the coding standards and secure design are looked upon. The null hypothesis H08 states that sum of *PSCA* and *PCS* is unrelated to *NSE*. Using t-test (Equation 6), the null hypothesis is rejected at 5% level of significance ($t=3.37$) indicating that *NSE* is result of consideration of *PSCA* and *PCS* during coding.

Effectiveness Factor

Let there exists one independent variable X and one dependent variable Y as *NSE*. To establish the relation between X on Y, we apply Least Square Method to identify the effect of X on Y. Thus,

$$Y = C - \gamma \cdot X$$

where C is constant and γ is effectiveness of X on Y. γ can be calculated as

$$\gamma = (C - Y) / X \text{ ----- } 11$$

where $X > 0$

Using Equation 11, the combined effect of *PSCA* and *PCS* (independent variable) on *NSE* (dependent variable) is given by

$$\gamma = (8.4022 - NSE) / (PSCA + PCS) \text{ ... } 12$$

Special cases – $\gamma = -1$: Using Equation 12, the sum of the value of independent metrics and the constant C equals *NSE*. It may also result in significantly large *NSE*.

$\gamma = 0$: If $\gamma = 0$ then $NSE = 8.4022$. It indicates *NSE* to be moderately high (approx. 8.4) and is not dependent on sum of *PACS* and *PCS* metrics. The value may not possible as *NSE* cannot be a real number.

$\gamma = 1$: Effectiveness 1 indicates that $Y = C - X$, i.e. number of independent metrics should be less than 8.4. It also tells that if X increases, then Y decreases. Lower value of Y may reveals that security is considered from design stage as well as security coding standards.

Phase IV: Factors

Security testing uncovers the security flaws of the software product. Phase IV metrics helps in determining the effectiveness of security testing. The null hypotheses for implementation and testing stage can be stated as follows:

H09: *PE* is unrelated to *STR*

H010: *PE* is unrelated to *SRT*

Using Equation 6, the hypotheses H09 and H010 are evaluated using t-test. The null hypotheses H09 and H010 are accepted at 5% level of significance having t-values as 0.6346 and 0.2878 respectively, indicating that *PE* is not

determined by *STR* and *SRT*.

Phase V: Factors

The null hypothesis H_{011} states that the metric *PCSE* is unrelated to *Rsc*. H_{011} is analyzed at 5% level of significance using paired t-test using Equation 6. H_{011} is accepted revealing that the two are unrelated ($t = 1.045$) and hence effectiveness of the security efforts of this stage cannot be judged.

6. RESULTS

In this we first tried to study the available product and process security metrics for the software development stages. Next, process metrics for secured software development process have been established for various stages. Lastly, the effectiveness factors for each of the development stages have been realized to judge the effectiveness of security efforts of the development team.

We reviewed the available metrics for different life cycle activities of SSDP. It has been observed that most of the process metrics do not provide sufficient implementation details considering SDP stages. Moreover, the process metrics do not focus on the security issues of SDP phases. Most of the metrics access security late in the software development process making it difficult to fix the problems early.

The various security metrics have been established on the basis of the security needs of the different development lifecycle activities. The metrics instituted are *NSRG*, *SRRD*, *SRSE*, and *SRI* for requirements gathering stage while design stage metrics are *SRs*, *DTTE*, and *NDSE*. The coding stage metrics comprise of *PSCA*, *PCS* and *NSE*, and testing stage metrics are *SRT*, *PE*, and *STR*. The metrics established for operations and maintenance stage are *MTCSC*, *PCSE*, *RVA* and *Rsc*, whereas document stage metrics constitute of *NSCM*.

The trend of metrics for the various projects is depicted in Table 2. Based on the trend, following observations can be made:

1. *SRRD* is zero for web-based and desktop based systems indicating that there are no deviations from the security requirements during development process. For a client/ server based system, *SRSE* implies more deviations from security requirements in a client/ server based system as compare to the other two systems..
2. The value of *SRS* illustrates moderately high security requirements aforethought during design of client/ server and desktop based systems. During the design phase of a web- based system, *DTTE* indicates moderate to low use of design tools and testing effectiveness. It also illustrates moderate to low consideration of *SADT*, *NTSSD* and *NASD*.

Table 2: Trend of Metrics for Different Projects

S.No	Software Development Stages	Projects			
		Metrics	Web-based (JPS)	Client/ Server (PMS)	Desktop (LS)
1	Requirements Gathering and Analysis	NSRG	8	22	5
2		SRRD	0	2	0
3		SRSE	1	6	0
4		SRI	1	3	0
5	Software Design	SRs	62.5%	72.7%	80%
6		DTTE	0.41	0.39	0.38
7		NDSE	1	2	0
8	Coding	PSCA	70.59%	45.5%	30.8%
9		PCS	20%	25%	20%
10		NSE	7	15	4
11	System	SRT	1	0.81	1
12	Integration and Testing	PE	0.5	1.11	0
13		STR	0.40	0.75	0.6
14	Operations and Maintenance	MTCSC	17 days	16 days	NA
15		PCSE	30%	55.5%	0
16		RVA	4/qtr.	3/qtr.	NA
17		Rsc	0.2	0.33	0
18	Documentation	NSCM	6	5	5

3. Low *PSCA* and *PCS* have resulted in high *NSE* in client/ server based software. *NSE* may also indicate security errors due to low experienced software professionals in secured coding leading to fifteen security errors although security requirements gathered are high.
4. For a web-based system and desktop based system, *SRT* indicates that all the security requirements being gathered have been tested for security. *PE* reveals high to low errors per module for web-based and client/ server based system. *STR* shows that 40% modules have undergone security testing for a web-based system.
5. Operations and maintenance phase metric *PCSE* implies that the out of all changes expected in a web-based system, only 30% have been related to security policies. It shows that moderately high security consideration during earlier stages of software development resulting in less security exceptions. *Rsc* reveals 20% and 33.3% changes in *JPS* and *PMS* are due to security related issues.
6. *NSCM* indicates low security controls that do not have much importance during documentation for all types of systems.
7. Although *NSRG* is high for client/ server based system, security is given low consideration during all the stages of software development.

On the basis of measures and metrics of varied industrial projects we developed the Effectiveness Factors (EFs) for requirements gathering, design and coding stages. The effectiveness factors for the case studies discussed are illustrated in Table 3. Based on Table 3, following observations can be made:

The α values for web-based, client/ server based and desktop based systems illustrates that security considerations are moderately high and low for web-based and client/ server based systems respectively during requirements gathering stage. The α value for desktop based systems implies that the effectiveness of security considerations cannot be judged. This is in line with the fact that security is a major issue in web-based systems and low for desktop based systems.

Table 3: Effectiveness Factors

S.No	Projects EF	Web- based (JPS)	Client/ Server (PMS)	Desktop
1	α	0.405	-0.26	1.048
2	β	0.41	0.78	0
3	γ	0.0155	-0.101	0.087

1. The β values (Table 3) for web-based and client/ server based system indicates moderately effective and ineffective security considerations during design stage. The β for desktop based system signifies that no security errors have been detected in the design.
2. The γ values for web-based, client/ server and desktop based systems indicate that the coding errors are low in web and desktop based system as compared to client/ server based system. The values may be the result of the coding errors.

Conclusion

The metrics can assess security considerations more efficiently when measured during software development process. We analyze various security metrics available in the literature for product as well as process. Further, we developed a set of metrics in view of the security issues addressing the stages of software development process. The security metrics developed aims to evaluate the efforts of the various software development stages regarding security consideration. The metrics can help the development team to judge its performance for security and may focus on gathering more security requirements, consider more of implicit security aspects, make use of standards, increase input validation etc. by providing security training to the development team. Metrics may also act as a checklist for increasing security aspects of the software product. It may support the developers to enhance the software development

process. It shall help predict the phases that require more security forethought during development. It may be concluded that the metrics and effectiveness factors provide a way to assess and visualize security during the software development process.

References

1. Wingkvist, A., Ericsson, M., Lincke, R. and Lowe, W., —A Metrics-Based Approach to Technical Documentation Quality, In Seventh International Conference on the Quality of Information and Communications Technology, IEEE Computer Society, 2010.
2. Wallace, K., —Common Criteria and Protection Profiles: How to Evaluate Information, SANS Reading Room, 2013, [Online] Available: http://www.sans.org/reading_room/whitepapers/standards/common-criteria-protection-profiles-evaluateinformation_1078.
3. J. Wilander and J. Gustavsson, —Security requirements – A Field Study of Current Practice, Symposium on Requirement Engineering for Information Security (SREIS' 2005), Paris, France, Aug. 29, 2015.
4. Wilson, I., Lin, X. and Craske, N. —Client Server Security Issues, PC Update Online, The Magazine of Melbourne PC User Group, Australia, 1999, [Online] Available: <http://www.melbpc.org.au/pcupdate/9908/9908article8.htm>
5. Xiao, L., Lewis, P. and Gibb, A., —Developing a Security Protocol for a Distributed Decision Support System in a Healthcare Environment, ICSE'08, ACM, May 10–18, 2018.
6. Yee, K. P., —User Interaction Design for Secure Systems, Springer, Berlin Heidelberg, 2012, pp. 278-290.
7. Ying, R., —Building Systems Using Software Components, Software Tech, Vol. 9, No. 1, Mar.2016.
8. Jeff Zadeh, J., and DeVolder, D., —Software Development and Related Security Issues, IEEE, 2007, pp. 746-748.
9. Zhang, C.T. Designing Security into Software, Massachusetts Institute of Technology, University of New Hampshire, 2006.
10. Zipkin D. S., Using STAMP to Understand Resent Increases in Malicious Software Activity, S. M. Thesis, Technology and Policy Program, Massachusetts Institute of Technology, Cambridge, MA., 2005.
11. Zhang, X., Parisi-Presicce, F. and Sandhu, R., —Formal Model and Policy Specification of Usage Control, ACM Transactions on Information and System Security, Vol. 8, No. 4, Nov. 2015, pp. 351–387.