Modern Applied Science; Vol. 9, No. 8; 2015
ISSN 1913-1844 E-ISSN 1913-1852
Published by Canadian Center of Science and Education

# Mills Model Based Evaluation of Security of Software Systems

Valeriy Valentinovich Gurov[1] & Grigory Grigoryevich Novikov[1]

[1] National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Russia

Correspondence: Valeriy Valentinovich Gurov, National Research Nuclear University MEPhI (Moscow Engineering Physics Institute), Russia.

Received: January 3, 2015 Accepted: February 3, 2015 Online Published: July 30, 2015

doi:10.5539/mas.v9n8p213 URL: http://dx.doi.org/10.5539/mas.v9n8p213

**Abstract**

The presented paper discusses a possibility for an implementation of software reliability models for an evaluation of security of software. Mills model is proposed as the most suitable one. In order to use it more effectively it is proposed to implement a division of vulnerabilities into groups. Classification of vulnerabilities based on a method of their connection with features of a process's execution on a level of interaction of system resources and operating system is presented. Security of system in a context of a specific application can be evaluated more precisely by means of assigning of a certain weight to each group of vulnerabilities. The suggestions on an evaluation of vulnerability of software taking into account a division of vulnerabilities into groups according to the presented classification are made.

**Keywords:** software vulnerabilities, unauthorized access, software reliability models, Mills error seeding model, software security analyses

## 1. Introduction

Security is an inherent property of software, which is currently being given more and more attention. The various sources give slightly different definition of that property. In the presented paper, we will adhere to (ISO/IEC 9126-93 Information technology), which defines security as a set of software properties, characterizing its ability to prevent unauthorized access, both accidental and deliberate, to software and data, as well as a degree of convenience and completeness of a detection of results of such access or actions aimed for a destruction of software and data.

Unauthorized access may be carried out if there are certain vulnerabilities in software code. The aim of a detection of vulnerabilities in software is developing of tests for a detection of security breaches in software's security system and determining an effective testing procedure (Zhang, Liu, Csallner, Kung &Yu, Lei, 2014). For quantitative evaluation of software security in a phase of software development various methods are used, including hierarchical methods of modeling, stochastic Petri networks, discrete Markov chains, Bayesian network models, etc. (Holm, Korman, & Ekstedt, 2015; Zhu, Qian, Zhang, Yanlong, Zhang, 2014; Yang, Yu, Qian, Sun, 2012; Vibhu, Saujanya, Sharma, Kishor, 2007). However, such methods are often too complicated and significantly depend on volume and accuracy of initial data, which obtainment is, in turn, a separate complicated task.

To a large extent concept of security coincides with concept of reliability of software, one of the main properties of which is a resistance to errors, i.e., the ability to sustain a certain level of quality of the functioning in cases about software errors or failures of a specific interface (IPX over IEEE Std 610.12-1990).

Issues of evaluation and provision of software's reliability are discussed for a long time and possess a large set of models and methodologies. Its presentation is given in the work (Myers, 1976). An increase of number models and a development of methods of their implementation is continuing until the present time (see, for example, the study of Lohmor, & Sagar, 2014). At the same time, analysis of software security and provision of their security, as an aim, appeared much later. However, nowadays security issues had become leading for software development of a variety of systems, ranging from simple applied problems to cloud computing (Wright, McQueen, & Wellman, 2013; Younis, Kifayat, & Merabti, 2014). Therefore, it is interesting to consider a possibility of an implementation of software reliability models in relation to an evaluation of its security.

The paper presents a procedure for an application of Mills model of software reliability for an evaluation of software security. In order to increase effectiveness of initial data preparation process and to increase accuracy of

obtained results it is proposed to use the developed classification of low-level vulnerabilities.

## 2. Methodology

### *2.1 Basic Characteristics of Mills Model*

Software debugging is performed until intensity errors flow is reduced to an acceptable value for the given application, i.e., until mean time between failures is not large enough. Therefore, many reliability models focus on analysis of time of appearance of next error.

At the same time, it can be stated that, if software falls into a criminal's hands for a sufficiently long time, it will be trespassed. Therefore, it is reasonable to carry out an evaluation of its security by a degree of its resistance, i.e., by length of time during which a software can resist against unauthorized access. That parameter is determined by a number of vulnerabilities, which remained in a software after its testing.

Therefore, in that application it is unreasonable to use software reliability models, which are based on temporary characteristics of error flow. For that purpose one should use another class of models, for example, Mills model (Mills, 1972).

An implementation of Mills model in relation to an evaluation of software reliability consists of that, firstly, software is «obstructed» by a certain number of known errors. These errors are randomly made in a software, then it's presumed that for its own and introduced errors detection probability for a subsequent testing is the same and depends only on quantity. By means of testing of software for a certain period of time and sorting its own and introduced errors, one can estimate parameter $N$, which is an initial number of errors in a software, as well as to set confident level of that forecast.

Let's presume that, $S$ error were introduced into a software, after which testing was initiated. Let's presume that $n$ own errors and $v$ introduced errors were detected during the test. Then, an evaluation for $N$ according to maximum likelihood method will be as follows:

$$N = \frac{n}{v} * S \quad .$$

That model is simple and results obtained by means of that model are clear. However, a mechanism for an introduction of errors is currently the weakest part that model, because it is assumed that, a detection probability is identical (but unknown) both for its own and introduced errors. It results in the fact, that introduced errors must be «typical» sample errors. That, in turn, is quite hard to ensure, because software developers and persons, who introduce deliberately known errors, have usually have different programming style, skills and other aspects.

### *2.2 Classification of Low-Level Vulnerabilities*

Simplification of «typification» of introduced errors can be achieved by means of their classification. In that case, it is easier to introduce similar errors into software, while knowing features of typical errors of a certain class.

Turning to an applicability Mills model for an evaluation of software's security, first of all, let's specify that the criterion for an absence of security vulnerabilities in software is the fact that, during testing of software no violations of security requirements for studied software in a system of its presumable application were found (Kazarin, 2012). Probability of all vulnerabilities depends on completeness of coverage of all possible conditions of a software. At the same time, it is presumed, that there are means of vulnerabilities' detection for a given run of a software. It is necessary to create a test for a detection of a vulnerability of each type, the test is formed depending on developer ideas about how a certain vulnerability appears in a certain situation, and, thus, a way of its detection.

Thus, we have the following situation. A multiplicity of vulnerabilities $W=\{w_0, w_1, ..., w_m\}$, their appearance in a software in a given test impact is unknown. A multiplicity of test impacts $T=\{t_0, t_1, ..., t_m\}$ exists, each of which is designed to detect a specific vulnerability. It is necessary to minimize a test's length, which with a certain probability detects all vulnerabilities of software. At the same time it worth noting that a detection of vulnerabilities always has a probabilistic nature.

For a solution of that problem is reasonable to divide all vulnerabilities into groups, and inside those groups by means of a certain test (presumably, it can also consist of a several test impacts) all vulnerabilities can be detected.

In that case, on the one hand, it is necessary to reduce a number of groups, in order to reduce a number of

conducted tests. However, in that case, a situation can take place, in which there is no common attributes among all vulnerabilities, selected into a certain group. As a result, within each group for each vulnerability it will be necessary to generate its own test impact. That, in turn, will not produce any advantage for a test procedure, which leads us an original option.

At the same time, an excessive increase of number of groups can lead to the fact that a total number of groups will be equal to an original number of identified vulnerabilities.

Because of that, it is necessary to develop such a classification of vulnerabilities, which would make it possible to bring together in separate groups vulnerability, which are of common appearance in a certain test impacts.

A large number of approaches for a division of vulnerabilities into classes is suggested. Such approaches are offered and used both by various companies developing anti-virus software and other companies, conducting their own analysis of their security software systems (Classification is detected objects, 2014; Classification of viruses' names according to «Doctor Web», Engle and Bishop, 2008; Knight, 2000).

However, in the discussed case, the most effective measure is use of a classification of vulnerabilities related with characteristics of a process's execution at a level of interaction of system resources (memory, processor) and the operating system -*low-level vulnerabilities* (Gurov, Gurov, Ivanov and Shustova, 2010). Areas of appearance of vulnerabilities and mechanism of their use for carrying out an attack on a computer systems are taken as a parameters for that classification.

According to the classification, vulnerabilities are divided into 6 classes. Those are buffer overflow, integer overflow, format string error, double free and race conditions. Buffer overflow vulnerability, in turn, is divided into two subclasses: stack overflow and heap overflow.

Also, data interpretation vulnerabilities can be attributed to a separate class. Low-level vulnerabilities are united by a cause of an appearance, which is an inadequate monitoring of input values.

Vulnerabilities inside classes are divided into types.

Stack overflow subclass is characterized by an overwrite of return address from a vulnerable function and is divided into 3 types:

- executable code is placed in stack;
- executable code is placed in environment variable;
- data, overflowing buffer, is placed in stack in such a way that they interpreted as input parameters of functions from system library (e.g., system( )), which is called, when a next command, which is indicated by a return address, is executed. This is «ret scheme attack» (the name is based on a name of assembler's command).

A vulnerability, which is also connected to those main types of vulnerabilities, is single-byte overflow, which appears because of an incorrect calculation of a number of elements in an array. As a result, it is impossible to rewrite return address, but it is possible to rewrite lower byte register *EBP*, which is in a case of *x*86 architecture stores an address of stack of software, which is called a function with a vulnerability. As a result, in a case of a return of a function with a vulnerability, stack frame can be modified and a next execution of ret and reti commands will pass a control to malicious code, which is the same as for another types of stack overflow (via environment variable, return to system library or putting of malicious code directly into stack).

«Heap buffer overflow» subclass is different from stack overflow in that an aim of a criminal is not return address but, in the first place, data in working memory. The subclass is divided into 4 types according to a type of rewritten data:

- -first type is rewriting of local function data, containing vulnerability. In a case of passing through borders of overflown array, adjacent set internal variables are deleted. In many cases it is enough to obstruct authorization system, which leads to authorization without a correct verification of user rights.
- -in the second type of vulnerability, which is «heap overflow», a function is rewritten. As a result, a criminal is becoming able to pass control to an arbitrary point in memory and execute introduced code.
- -the third type of vulnerability consists of rewriting of virtual functions table, which, in a case of use of gcc compiler, is placed after variables of a class, which is using a virtual function. Risk factor is the same, i.e. passing of control to arbitrary code.
- -the forth vulnerability type is based on use of mechanism of dynamic memory allotment. Blocks of allocated memory are united in doubly-connected list, and, as a result of overflow in data area of that block,

service information is rewritten. It leads to a situation, when during an execution of free() command, which is emptying an overflown block, a substitution of address of one of system functions in global offset table takes place, which allows a criminal to pass control to an arbitrary code in a case of calling of a rewritten function.

Integer overflow is a next class of vulnerabilities. Vulnerabilities of that class are related with an interpretation passing of results of arithmetic operation through limits of format (as a rule, ignoring of such overflow), and is divided into 3 types:

- overflow in a case of setting a variable's value bigger, than it is capable to enclose (e.g., in a case of setting integer type value to byte type variable a truncation of higher high-order bits takes places, digit of initial number is unchanged).
- overflow as a result of arithmetic operation, if a result is enclosed into unsigned variable (e.g., if in a case a result of an addition of two positive numbers is bigger than format, bit, which is passing out of its limits, is truncated);
- overflow in a digit in a case of executing of a arithmetic operation, if the result can be enclosed in character variable (that kind of overflow takes place, if a result, which is unsigned number with one in high-order bit can be enclosed in a character variable, in that case, that bit is interpreted as sign and the number becomes negative).

A change of a variable inside a software can lead to a situation, in which normal run of a software will be compromised, e.g. not enough space can be allocated for operations with string, which leads to buffer overflow, authorization mechanism logic will be changed, as a results, a criminal can, for example, conduct unauthorized access into a system with the highest user rights (superuser rights).

A vulnerability of string format takes place because of an absence of functions' specificators, which are conducting formatted input, or because of a specification of smaller number of specificators, as compared to a number of inputted variables. Nature of vulnerability is in the mechanism of data transfer into a function during a variable input of parameters. If a specificator is set as one of parameter's values, a function will search in memory to find data itself for processing with that specificator. As a result, a criminal can read any byte of memory or (by means of using «%n» specificator) rewrite any byte by means of a transfer of various number and specificators as a parameter of a function of a formatted input. Thus, knowing a structure of address space of an attacked process, a criminal can obtain parameters of authorization of legit users or ever rewrite return address or addresses of system functions.

The last vulnerability class, which is directly related with features of compiler's operation and placement of service information during an execution of software, is double free. Mechanism of realization of that vulnerability is close to «heap overflow» type, which is related to mechanism of dynamic memory allotment. If a software empties the same block twice, service data after a first emptying can be modified in a such way, that repeated calling of free() function leads to a situation, when a function's address in a global table offset table will be replaced by an address of introduced code.

Race conditions is a next class of vulnerabilities. That class of vulnerabilities is based on a transparency of resources of operating system for processes, each of which is working in a way like it has an exclusive time-constant accesses to OS resources. In that vulnerabilities class file race condition type, which is related with a verification of a file's existence and its following opening for witting. If a file doesn't exist, but in time between a verification and opening external process is able to create a file (which can be a link to a certain system file), an attacked application will send its output into system file, destroying of changing information in a file.

Signal race condition vulnerability can be used in a same manner, in that case a software after receiving a signal of completion stops its operation and after repeated launch continues it from «the same point» without carrying out a verification for unchanged external conditions (in particular, its temporary files). A presence of that kind of vulnerability can lead to an interruption of running of software with a vulnerability. In the same time, a criminal, depending on user rights, for example, can overwrite system file, change a system's configuration or even seize control with superuser rights.

Finally, the last class of discussed vulnerabilities is data interpretation vulnerabilities. Those vulnerabilities consist of a transfer of input data to an external interpreter without necessary verifications and restrictions. Vulnerabilities of that class can be described as follows. Logic of software implies, that data, entered into software, will be connected with some kind of statistic block and transferred to interpreter. It can be a value for a certain field of SQL-query (e.g., a software gives interpreter request «select permits from users where name =

*user_input*») or name of a catalog, which contents are produced by a software (shell-command «ls –al *user_input*» is formed). However, if not just number or name is passed into software, but text, which is separated by special signs (e.g., semicolon), it can be processed by interpreter as other command. In a case of processing of such a inputed fragment unauthorized reading or modification of data can take place.

*2.3 Implementation of Mills Model for An Evaluation of Security of Software*

In respect to an evaluation of security of software, an implementation of Mills model with a consideration of use of introduced errors will be as follwos (Gurov, Gurov, & Ivanov, 2012). Let's introduce S vulnerabilities into a software, including $s_i$ vulnerabilities of i type ( $S = \sum_{i=1}^{m} s_i$ ), where m is a number of types of vulnerabilities. Then, if during testing it is detected that n own vulnerabilities, including $n_i$ vulnerabilities of i type ( $n = \sum_{i=1}^{m} n_i$ ), and v introduced vulnerabilities, including $v_i$ introduced i type ( $v = \sum_{i=1}^{m} v_i$ ), then a number of own vulnerabilities of i type, which were initially in a software, is $N_i = \frac{n_i}{v_i} * s_i$, and a total number of vulnerabilities N, which were in a software before beginning of testing will be as follows: $N = \sum_{i=1}^{m} N_i = \sum_{i=1}^{m} (\frac{n_i}{v_i} * s_i)$ .

Number of own vulnerabilities of i type, which remained after testing, will be following: $k_i = N_i - n_i = \frac{n_i}{v_i} * s_i - n_i$

A total number of own vulnerabilities, remained in a software after testing, will be as follows: $K = N - n = \sum_{i=1}^{m} (\frac{n_i}{v_i} * s_i) - n$

That value can be used for solving of the second problem, which is an evaluation of the model's confidence level. Let's presume that software has not more that k own vulnerabilities and introduce S vulnerabilities into it. Testing will continue until all vulnerabilities will be found. After testing is over, let's calculate number of detected own vulnerabilities, which are contained in a software n. Significance level C, which defines the model's confidence level and a probability, that the model will correctly reject a false presumption can be calculated by means of the following equations: $C = \begin{cases} 1, & \text{if } n > k \\ \frac{S}{S+k+1}, & \text{if } n \le k \end{cases}$ .

Thus, if we presume, that after first step of testing in a software there is no more than 2 vulnerabilities (k = 2), than, in order to achieve significance level of C=0.9, it is necessary to introduce 27 vulnerabilities and find all of them.

However, testing before all introduced vulnerabilities will be found is a quite complicated task. For a situation, when at testing phase only j from S of introduced vulnerabilities are found ($j \le S$), for an evaluation of validity of the result the equation can be used, which was introduced in the work

(Teichroev, 1972):

$$C = \begin{cases} 1, & \text{if } n > k \\ \binom{S}{j-1} / \binom{S+k+1}{k+j} & \text{if } n \le k \end{cases},$$

where expression of $\binom{a}{b}$ type is calculated as follows:

$$\binom{a}{b} = \frac{a!}{b!*(a-b)!}.$$

By means of obtained relationship, for example, a number of vulnerabilities, that should be introduced to achieve a desired reliability, or any other parameters, necessary for subsequent analysis, can be defined.

In the discussed case, a task is complicated by a fact that it is necessary to calculate $j_i$, which is a number of vulnerabilities found during testing of initially introduced $S_i$ vulnerabilities of each of $m$ types. In this case, the reliability of an evaluation of received number of own vulnerabilities of $i$ type, which were initially present in a software, should be evaluated individually.

$$C_i = \begin{cases} 1, & \text{if } n_i > k_i \\ \binom{S_i}{j_i - 1} / \binom{S_i + k_i + 1}{k_i + j_i}, & \text{if } n_i \le k_i \end{cases},$$

where $n_i$ is a number of found own vulnerabilities of $i$-type, $k_i$ is an estimated number of vulnerabilities of $i$ -type.

## 3. Results and Discussion

In the presented paper is suggested to use Mills model for an evaluation of software reliability for an assessment of degree of security of software systems. In order to improve quality of the work of the model, it is proposed to use a division of all vulnerabilities into separate groups in a case of reduced introduction of vulnerabilities into a software. Such an approach would make it possible, on the one hand, to minimize a number of tests to verify security of software system, and, on the other hand, will allow to simplify a process of a creation of tests for vulnerabilities for each group. Classification of vulnerabilities based on a method of their connection with features of a process's execution on a level of interaction of system resources (memory, processor) and operating system is presented. That classification allows to detect typical attributes of vulnerabilities and, consequently, mechanisms for their appearances in a certain test impacts.

The expressions obtained in the presented study allow to define a number of vulnerabilities, remaining in the program after a test phase, as well as to determine a number of vulnerabilities of each type, that should be artificially to introduced into software before start of a test, to obtain a specified level of a reliability of results. Presented relationships allow to evaluate that level even for a case, when not all artificially made vulnerabilities were detected during testing, which, as a rule, is actual test conditions.

Using the obtained relationships, it is possible, in particular, to introduce vulnerabilities of a certain type before testing phase in order to obtain the most reliable results for vulnerabilities that pose the greatest treat for a given application. Level of that risk is defined either by an expert or according to data by organizations authoritative in that area (see, for example, CWE/SANS Top 25, 2011).

Thus, the method proposed in the presented study allows to eliminate the main disadvantage of Mills model, i.e. uncertainty, related with an introduction of artificial errors, while maintaining simplicity of Mills model.

Application of the proposed methodology implies the following steps:

- definition of danger level for a given application for all possible types of vulnerabilities presented in Figure 1;
- on that basis, calculation of a number of vulnerabilities of each type, which should be introduced in an initial software in order to obtain desired reliability of testing results;
- testing of software until a moment, when a number of detected vulnerabilities of each type will allow to provide specified level of security.

General issues of applicability of certain models for software reliability for an evaluation of security were discussed before in the works (Kazarin, 2012; Markov, 2011; Sharma, & Kishor, 2007). However, the approach proposed in the presented paper shows that it is reasonable to use exactly Mills model, it also allows to improve accuracy of obtained results by means of differentiated approach for a selection of initial data on a basis of the classification presented in the paper.

## 4. Conclusion

Currently, the security of software against unauthorized access is a topical subject. Criminals do not always want only to steal information. Often their purpose is a violation of a computing system, seizing control over it, distortion or deletion of information stored in it. Appearance of vulnerabilities can occur in a wide variety of situations and for a variety of reasons.

In a stage of a development of a software a topical problem is search for vulnerabilities and their elimination, which are based on features of interaction between processor and memory, order of an execution of applications in a processor, placement of data in memory, an organization of access of processes to resources and so on Such vulnerabilities has do not lead to an interruption of system security during normal operation, i.e. as it was planned by a programmer. They are, as a rule, are not detected during normal testing, as they do not violate logic of the operation of a software. However, in a case of use of specific input data, for example, data of a very large size, various non-standard situations can take place, which in some circumstances can lead to a system's compromise and, in particular, a criminal obtaining full access to a system. Thus, for a developer, in order not to allow presence of any vulnerabilities in a final software product, it is necessary to take additional special measures for their detection and localization.

Detection of vulnerabilities in software is a complex and challenging task, especially, because a problem of ensuring security of software is relatively new. In the presented work it is suggested to use approaches for a detection of vulnerabilities, which are analogous to those, that are used for an evaluation of software reliability for a long time. Among a large number of models of software reliability Mills model is marked out as the most suitable for that case. In order to minimize the least formalized part of the model, which is artificial introduction of inconsistencies in tested software, the classification of vulnerabilities based on principle of interaction system resources and operating system is proposed.

It is demonstrated that, in that case, an evaluation of large number of vulnerabilities, remained in a software after its testing, can be carried out, as well as recommendations for additional testing of software with a view to detect those remaining vulnerabilities, which are the most critical for that specific application of software system, are given.

Further studies in that direction are planned to comprise formalization of description of vulnerabilities of different classes and testing impacts, which are ensuring their detection, as well as verification of approaches, presented in the paper, in the widest possible range of specific software systems.

**Acknowledgments**

The authors would like to express gratitude to Doctor of Technical Sciences Professor Mikhail Ivanov, as a specialist in a security of software systems, for careful attention to our attempts to use non-traditional software reliability models to assess state of security of software and valuable comments made by him in that regard.

**References**

Classification of detected objects (2014, May 20). Securelist. Retrieved October 20, 2014, from http://www.securelist.com/ru/threats/detect?chapter=32

Classification of viruses' names according to «Doctor Web». (n.d.). *Website of Dr. Web*. Retrieved October 20, 2014, from http://vms.drweb.com/classification/

Engle, S., & Bishop, M. (2008). *A Model for Vulnerability Analysis and Classification.* Retrieved October 20, 2014, from http://www.cs.ucdavis.edu/research/tech-reports/2008/CSE-2008-5.pdf

Gurov, V., Gurov, D., & Ivanov, M. (2012). Implementation of software reliability models for an evaluation of security of software. *Security in information technologies, 1*, 88-91. ISSN 2074-7128

Gurov, V., Gurov, D., Ivanov, M., & Shustov, L. (2010). Secure programming technology and its teaching features in a case of higher educational institutions. *Remote and virtual training, 9*, 35-49. ISSN 1561-2449.

Holm, H., Korman, M., & Ekstedt, M. (2015, February). A Bayesian network model for likelihood estimations of acquirement of critical software vulnerabilities and exploits. *Information and Software Technology, 58*, 304-318. http://dx.doi.org/10.1016/j.infsof.2014.07.001

IEEE Std 610.12-1990. (1990). *IEEE Standard Glossary of Software Engineering Technology (ANSI).*

ISO/IEC 9126-93. (1993). *Information technology. Software product evaluation. Quality characteristics and guidelines for their use.*

Jason, L. (2013, May). Wright, Miles McQueen, Lawrence Wellman Analyses of two

end-user software vulnerability exposure metrics (extended version). *Information Security Technical Report, 4*(17), 173-184. ISSN 1363-4127.

Kazarin, O. (2012). *Security of computer systems' software*. Moscow: Moscow State Forest University, 2003. Retrieved October 20, 2014, from http://citforum.ru/security/articles/kazarin/2.shtml#2

Knight, E. (nd.). *Computer Vulnerabilities.* Retrieved October 20, 2014, from http://www.ussrback.com/docs/papers/general/compvuln_draft.pdf

Lohmor, S., & Sagar, B. (2014). Overview: Software Reliability Growth Models. *International Journal of Computer Science and Information Technologies – IJCSIT, 5*(4), 5545-5547.

Markov, A. (2011). Model of an evaluation and planning of software testing program according to information security requirements. *Bulletin of Bauman Moscow State Technical University. Series «Instrument engineering».* Retrieved October 20, 2014, from http://www.npo-echelon.ru/doc/software_reliability_models.pdf. ISSN 0236-3933

Mills, H. (1972). *On the Statistical Validation of Computer Programs, FSC-72-6015.* IBM Federal Systems Div., Gaithersburg, Md.

Myers, G. (1976). *Software reliability. Principles and practices*. New York, London, Sydney, Toronto: A Willey Inter science Publication, John Willey & Sons.

Teichroev, D. (1972). Survey of Languages for Stating Requirements for Computer-Based Information Systems. *Proceedings of the 1972 Fall Joint Computer Conference. Montvale, N.J.: AFIPS Press*, 1203-1224.

Top 25 Most Dangerous Software Errors. (2011). *CWE/SANS – Common Weakness Enumeration*. Retrieved October 20, 2014, from http://cwe.mitre.org/top25/

Vibhu, S., & Kishor S. (2007, April). Trivedi Quantifying software performance, reliability and security: An architecture-based approach. *Journal of Systems and Software, 4*(80), 493-509. http://dx.doi.org/10.1016/j.jss.2006.07.021

Yang, N., Yu, H., Qian, Z., & Sun, H. (2012, January). Modeling and quantitatively predicting software security based on stochastic Petri nets. *Mathematical and Computer Modeling, 1-2*(55), 102-112.

Younis, A., Kifayat, K., & Merabti, M. (2014, February). An access control model for cloud computing. *Journal of Information Security and Applications, 1*(19), 45-60. http://dx.doi.org/10.1016/j.jisa.2014.04.003

Zhang, D., Liu, D., Csallner, C., Kung, D., & Lei, Y. (2014, January). A distributed framework for demand-driven software vulnerability detection. *Journal of Systems and Software, 87*, 60-73. http://dx.doi.org/10.1016/j.jss.2013.08.033

Zhu, H., Zhang, Q., & Zhang, Y. (2014). Chapter 5 – HASARD: A Model-Based Method for Quality Analysis of Software Architecture. *Relating System Quality and Software Architecture*, 123-156.