# An outline of the method for predicting IT vulnerabilities

*Mariusz* Zieja[1,*], *Mirosław* Zieja[1] and *Artur* Stachurski[2]

[1] Department of IT Support for Logistics, Air Force Institute of Technology, Warsaw, Poland
[2] Institute of Computer and Information Systems, Military University of Technology, Warsaw, Poland

**Abstract.** Majority of the currently known quantitative models for vulnerability analysis do not allow for a comprehensive vulnerability prediction process for a selected software. The article presents the outline of the method for predicting software vulnerabilities. The presented solution is based on probabilistic properties that allow to reflect external and internal factors affecting software and determining its vulnerabilities. Also, a possible direction of further method development was described, indicating the way of improving the method with elements representing preventive measures, as a result of which it may be possible to limit or eliminate potential software vulnerabilities.

## 1 Introduction

Recent advances in information technology and spread of Internet services result in new threats emerging in cyberspace. In many cases, their natural consequences may lead to the execution of cyberattacks, which are most often outcomes of conscious exploitations of IT vulnerabilities. Among many dictionary definitions of IT vulnerability it can be generally understood as a weak point in the applied security procedures of systems, hardware, software, as well as bad system administration and usage that may cause damage to its operations or may be used by cyber-aggressors to gain unauthorized access to information or disruption of the system [1, 2]. IT Vulnerability is most often defined in relation to the cyber threat, which is commonly described as any potential mechanism, event, random or intentional action that may lead to vulnerability exploitation and consequently to a potential loss [1, 3].

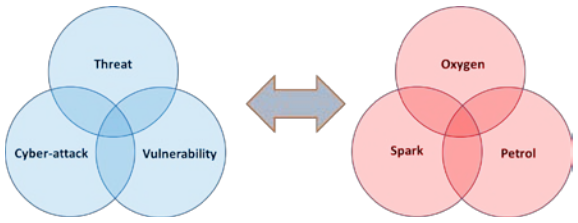

**Fig. 1** Cyber threat, cyber attack, IT Vulnerability overview scheme. Source: Own elaboration.

In the last few years, many works regarding the provision of software security have been done, taking into account the fact of software vulnerabilities presence [4-8]. In most cases, these works focused on a qualitative approach which aims at improving vulnerability detection techniques and preventing their occurrence in the software. In contrast to those already mentioned, there are less examples in the literature describing the use of quantitative methods for vulnerability analysis, which are used in the study of software reliability in an effective and reliable manner (e.g. Software Reliability Growth Models - SRGM) [9]. Therefore, it can be assumed that a similar approach can be implemented for measuring software security.

The article presents the outline of the method for software vulnerability prediction. The solution is based on probabilistic properties which provide great versatility. These can reflect software security, which for the purposes of this work, can be understood as resistance to vulnerabilities including external and internal factors affecting the software. An example of the further development of the method was also proposed, which will allow to adapt elements reflecting the use of preventive measures and, as a consequence, reduce the number of vulnerabilities in the software.

## 2 State of the art

A quantitative vulnerability analysis process involves using methods developed to provide a useful insight to assess software security. One approach to measuring vulnerabilities quantitatively focuses on the study of a single vulnerability. It can be implemented using mechanisms which provide standardized vulnerability scores. These can be obtained on the basis of calculations performed using individual vulnerability characteristics, ultimately indicating vulnerability severity [4]. The following standards are examples of measuring single vulnerabilities depending on their type:

- CVSS (*Common Vulnerability Scoring System*) - a standard for measuring the impact of software flaw vulnerabilities and a format for communicating vulnerability characteristics [4];
- CWSS (*Common Weakness Scoring System*) - a mechanism for testing the vulnerability severity

[*] Corresponding author: mariusz.zieja@itwl.pl

resulting from software defects. Conceptually similar to CVSS, but there are differences in construction and the type of information processed on vulnerability [10];

• CCSS (*Common Configuration Scoring System*) - a mechanism for measuring the severity of vulnerabilities, resulting from the use of inappropriate system configuration [5];

• CMSS (*Common Misuse Scoring System*) - a mechanism for measuring software feature misuse vulnerabilities [6].

Other ways of measuring software vulnerabilities focus on particular vulnerability sets [9]. These include analyses covering either whole software products or their particular modules. For this purpose, Vulnerability Discovery Models (*VDMs*) have been developed. The general purpose of VDM is to indicate trends in vulnerability detection. Most VDM models are time function based and were created on the basis of current and historical data obtained during software operation process. With this approach, it is possible to predict the number of potential vulnerabilities that may be detected in the future. Such quantitative approach to measuring vulnerabilities of a given system can be used to determine software security metrics such as e.g. *Vulnerability Density* or *Vulnerability Discovery Rate* [9]. As a result, it can be a reasonably good instrument for software producers and users to understand security trends, allowing for an efficient security management. A brief description of selected VDM models is presented in subsections 2.1-2.3.

## 2.1 Alhazmi-Malaiya Logistic (AML) model

The Alhazmi-Malaiya Logistic Model (AML) is an S-shaped, time logistic VDM [9]. The whole process of software vulnerability discovery in AML model is divided into three distinct phases: learning phase, linear phase and saturation phase, as shown in Fig 2.

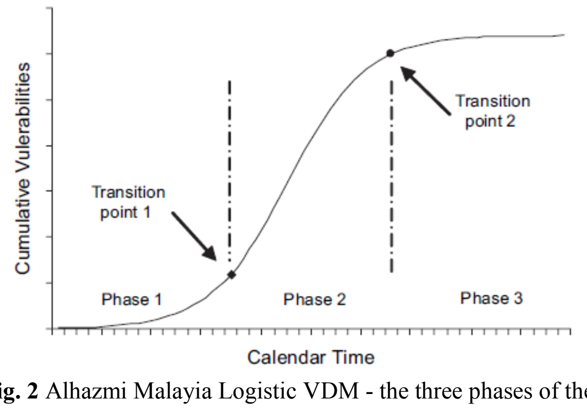

**Fig. 2** Alhazmi Malayia Logistic VDM - the three phases of the vulnerability discovery. Source: [9]

In the first phase, when the examined software starts being used, there is a gradual increase in the number of software users, including security testers and potential attackers. This way, a relatively small increase in the number of detected vulnerabilities can be observed. After some time (passing through Transition point 1), the number of users and testers increases significantly, which results in a linear increase in detected vulnerabilities - the software is then in the so-called linear phase. After that, when a newer version of the examined software is released (passing through Transition point 2), it enters a saturation phase, during which the vulnerability detection rate decreases.

The AML model is based on the assumption that the rate of change of the cumulative of all vulnerabilities $N(t)$, detected to time $t$, depends on the two factors: the first decreases with the decrease in the number of still undiscovered vulnerabilities in the system, while the second factor increases with time from the release of the system. Using the above assumptions, the *Vulnerability Discovery Rate* is given by the following differential equation:

$$\frac{dN(t)}{dt} = bN(t)(a - N(t)) \tag{1}$$

where $N(t)$ is a cumulative number of vulnerabilities to the testing time $t$, $a$ and $b$ are constants, determined empirically using the recorded data. By solving differential equation (1) we obtain formula for cumulative number of vulnerabilities to the time $t$, as follows:

$$N(t) = \frac{a}{ace^{-abt} + 1} \tag{2}$$

where: $c$ is a constant introduced while solving differential equation (1).

## 2.2 VDM for a software system using stochastic differential equation

Another VDM - model by A.K. Shrivastava, R. Sharma and P.K. Kapur, can be considered as an improvement of the ALM model, to which authors introduced additional parameters and assumed that the vulnerability discovery process in the system can be modelled as a stochastic process with a continuous state space, based on the following assumptions [11]:

1. As the research progresses, the number of remaining vulnerabilities in the system shows a downward trend.
2. There may be a system failure during its operation as it has vulnerabilities.
3. The process vulnerability elimination is ideal and during that, no new vulnerabilities arise, as a side effect.

In the approach, authors have extended differential equation (1) describing the *Vulnerability Discovery Rate* through randomisation [12] to a stochastic differential equation, that is given by the following formula:

$$\frac{d}{dt}N(t) = \left(b(t) + \sigma \cdot \gamma(t)\right) \cdot \left(a - N(t)\right) \tag{3}$$

where $b(t)$ is the result of transforming equation (2), $\sigma\gamma(t)$ is a standardized Gaussian white noise, $\gamma(t) = dW(t)/dt$, $W(t)$ is called a Wiener or a Brownian process [11].

The extension of the equation and using Îto formula for further transformations allowed authors to obtain the following formula for the cumulative number of software vulnerabilities to the testing time $t$:

$$N(t) = a - (a-k)\cdot\left[ \mathrm{e}^{-\int_0^t b(t)dt - \sigma W(t)} \right] \tag{4}$$

### 2.3 Rescorla Exponential Model

The next example of VDM - Rescorla Exponential Model [13] is an adaptation of a Software Reliability Growing Model by Goel-Okumoto [14] which can be described as follows:

$$\omega(t) = N\cdot\lambda\cdot e^{-\lambda\cdot t} \tag{5}$$

where $N$ is a maximum amount of vulnerabilities, which can be discovered in the examined software. $\lambda$ is a constant rate, representing *Vulnerability Discovery Rate*. By integrating equation (5) it is possible to determine the formula for the cumulative number of vulnerabilities in the software to the testing time $t$. which can be given as follows:

$$N(t) = N\cdot(1-e^{-\lambda\cdot t}) \tag{6}$$

## 3 A dynamic approach to measuring vulnerabilities

The idea of software security level assessment is presented in Fig. 3.

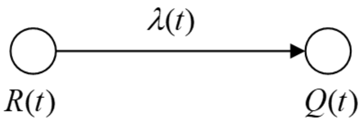

**Fig. 3.** Graph depicting software security level assessment

where:

| | | |
|---|---|---|
| $R(t)$ | - | software security state (no vulnerabilities discovered); |
| $Q(t)$ | - | vulnerable state (vulnerabilities discovered in the examined software); |
| $\lambda(t)$ | - | intensity of transition from state $R(t)$ to $Q(t)$. |

By using the above symbols, software security can be defined as follows:

$$R(t) = 1 - Q(t) \tag{7}$$

According to literature [15,17,18,19], the intensity of vulnerability occurrence in the software is probability density of vulnerability occurrence at time $t$ on condition that there were no vulnerabilities discovered in the software by the time $t$. The intensity of vulnerability occurrence in the software is described by the following formula:

$$\lambda(t) = \lim_{\Delta t\to 0}\frac{P\{t < T \le t+\Delta t \mid t < T\}}{\Delta t} \tag{8}$$

Let function $f(t)$ be the density of vulnerability occurrence in the software at time $t$. By using the conditional probability definition, after performing transformations, the function $f(t)$ can be written as follows:

$$f(t) = \lambda(t)R(t) \tag{9}$$

Further transformations allow to obtain the following system of differential equations [16,17,18,19]:

$$\begin{cases} R'(t) = -R(t)\lambda(t) \\ Q'(t) = \lambda(t)R(t) \end{cases} \tag{10}$$

By assuming initial conditions $R(0) = 1$ and $Q(0) = 0$ one obtains the following solution to a system of differential equations (10):

$$\begin{cases} R(t) = e^{-\int_0^t \lambda(t)dt} \\ Q(t) = \int_0^t \lambda(t)R(t)dt \end{cases} \tag{11}$$

For the purposes of this work, it can be assumed that the probability of a cyberattack resulting from the exploitation of vulnerabilities in a certain time interval is directly proportional to this time interval and to the number of all working software instances, within which the vulnerability can be exploited. Based on the above assumption, it can be defined as follows:

$$P(t, t+\Delta t) = \lambda N(t)\Delta t \tag{12}$$

gdzie:

| | | |
|---|---|---|
| $P(t,t+\Delta t)$ | - | Probability of vulnerability occurrence in the software in a $\Delta t$ time interval; |
| $N(t)$ | - | number of software instances in operation, in which the vulnerability can be exploited; |
| $\lambda$ | - | Factor of proportionality (Intensity of vulnerability occurrence); |
| $\Delta t$ | - | time interval of software operation (cumulative time interval of software operation) |

Let $P_n(t)$ be the probability that there were $n$ vulnerabilities discovered during time interval $(0,t)$. By using postulates of a Poisson process, the system of equations can be written as follows/in the following way:

$$\begin{cases} P_0(t+\Delta t) = P_0(t)(1-\lambda N(t)\Delta t) \\ P_n(t+\Delta t) = P_n(t)(1-\lambda N(t)\Delta t) + P_{n-1}(t)\lambda N(t)\Delta t \end{cases} \tag{13}$$

for $n > 0$

By using definition of the derivative, solution to the system of equations (13) takes the following form:

$$\begin{cases} P_0(t) = e^{-\lambda\int_0^t N(t)dt} \\ \vdots \\ P_n(t) = \dfrac{1}{n!}\left[\lambda\int_0^t N(t)dt\right]^n e^{-\lambda\int_0^t N(t)dt} \end{cases} \tag{14}$$

The probability that there may be $n$ vulnerabilities discovered in time interval $(0,t)$ is described by the Poisson distribution, whereby $\lambda t$ expression is replaced by $\lambda\int_0^t N(t)dt$.

Integral $\int_0^t N(t)dt$ can be replaced with the following sum:

$$\int_0^t N(t)dt \leftrightarrow \sum_{i=1}^{N} t_i \tag{15}$$

where:

$N$ - number of software instances in operation during time interval $(0,t)$;

$t_i$ - time of single software instance operation.

Thus, probability of vulnerability discovery in a single software instance is given by:

$$q = 1 - e^{-\lambda t} \tag{16}$$

where:

$q$ - Probability of vulnerability discovery in a single software instance;
$t$ - time of single software instance operation in time interval $(0,t)$.

Further transformations allow to determine estimator of the unknown intensity of discovered vulnerabilities - parameter $\lambda$**.** For this purpose, the maximum likelihood method was used, which gives the result that can be written as follows:

$$\hat{\lambda} = \frac{n_1 + n_2 + \ldots + n_i}{T_1 + T_2 + \ldots + T_i} \tag{17}$$

That allows to describe estimator $\hat{\lambda}$ as the average number of mutually independent vulnerabilities per time of operation of tested software instances.

## 4 Vulnerability assessment

To assess security by considering the impact of the potential vulnerabilities in a tested software, it should be verified whether a difference in intensities of vulnerability discovery obtained for two successive time intervals has changed significantly. For this purpose, let's use a statistical hypothesis test. Let's assume that software was in operation in time $t_{(t_0,t_2)} = t_{(t_0,t_1)} + t_{(t_1,t_2)}$. In the first step, intensities of vulnerability discovery in a tested software for time intervals $t_{(t_0,t_1)}, t_{(t_1,t_2)}$ are calculated:

$$\lambda(t_{(t_0,t_1)}), \lambda(t_{(t_1,t_2)}) \tag{18}$$

Then, it is possible to formulate the following hypotheses:
- $H_0$: The difference $\lambda(t_{(t_1,t_2)}) - \lambda(t_{(t_0,t_1)}) > 0$ is significant and a high upward trend in vulnerability intensity is noticeable. This is an alarming sign and further analysis of direct causes of undermining software security.
- $H_1$: The difference $\lambda(t_{(t_1,t_2)}) - \lambda(t_{(t_0,t_1)}) < 0$ is significant and a downward trend in vulnerability intensity is marked. This may be for example the result of applying effective preventive measures.

To examine hypotheses $H_0$ and $H_1$ a statistic test $S$, was defined as follows:

$$S = n\frac{\left(\lambda(t_{(t_0,t_1)}) - \lambda(t_{(t_1,t_2)})\right)^2}{\lambda(t_{(t_1,t_2)})} \tag{19}$$

where:
$n$ - number of discovered vulnerabilities in time interval $t_{(t_1,t_2)}$.

Conclusions about too frequent vulnerability discovery in the examined software (hypothesis $H_0$) or significant improvement of its security (hypothesis $H_1$) can be validated if obtained value of statistic $S$ is between $(a,b)$, where parameters $a$ and $b$ are determined on the basis of vulnerability data recorded during the process of software operation. Otherwise, security of the examined software has not changed significantly.

## 5 Future development – a proactive approach

An example of improving the process of predicting vulnerability discovery in the software as described in sections 3 and 4 may be achieved by the implementation of a proactive approach that allows for the use of preventive measures in this process.

When the process of increasing discovery of vulnerabilities in the software is random, it can be assumed that the intensity of vulnerability discovery is directly proportional to the product of the vulnerability discovery intensity and inversely proportional to the number of discovered vulnerabilities which determination included learning rate, applied prevention measures, etc.. Thus, the intensity of discovery of vulnerability can be given by the following formula:

$$\gamma(t) = \frac{\lambda}{1+\beta t} \tag{20}$$

where:

$t$ - time of software operation;

$\lambda$ intensity of vulnerability discovery (vulnerability indicator);

$\beta$ an indicator of the effectiveness of preventive measures implementation in a software;

$$\beta = \begin{cases} 1+S & \text{for effective countermeasures application} \\ 0 & \text{for no countermeasures application} \end{cases}$$

where:

$S$ - value of statistic (19).

What is more, the probability of an increase in the number of vulnerabilities in the examined software in a specified time interval can be described as follows:

$$\gamma(t)\Delta t \le 1 \tag{21}$$

By implementing such a defined vulnerability discovery intensity function, the proposed method may allow for predicting vulnerabilities discovery, which as a result may enable to assess the software security level.

## 6 Conclusions

The ability to prevent vulnerability occurrence in software or minimizing the level of their negative impact can have a significant influence on the broadly understood security of cyberspace elements. The use of quantitative models for vulnerability discovery can provide useful knowledge to assess the security of information systems. For example, it may allow for a more accurate determination of the time necessary to obtain the required level of security as well as an indication of personal, hardware and software resource allocation to achieve this. The method proposed in the article aims to improve the ability to predict IT vulnerabilities in the examined software. As a consequence, it can be a good tool for software end-users and vendors, enabling to forecast software security trends, as well as planning a broadly understood security management process.

## References

1. ISO/IEC, Information technology - Security techniques-Information security risk management" ISO/IEC FIDIS 27005:2008
2. Joint Task Force Transformation Initiative, Guide for Conducting Risk Assessments, *NIST*, [Online] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf, (2012)
3. Network Working Group, Internet Security Glossary, [Online] https://tools.ietf.org/html/rfc2828, (2000)
4. P. Mell, K. Scarfone, S. Romanosky, A Complete Guide to the Common Vulnerability Scoring System, *NIST*, [Online] http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=51198, (2007)
5. P. Mell, K. Scarfone, The Common Configuration Scoring System (CCSS): Metrics for Software Security Configuration Vulnerabilities, NIST Int. Rep. **7502**, (2010)
6. E. LeMay, K. Scarfone, P. Mell, „The Common Misuse Scoring System (CMSS): Metrics for Software Misuse Vulnerabilities," NIST Int. Rep. **7864**, (2012)
7. P. S. Anton, R. H. Anderson, R Mesic, M. Scheiern, *Finding and Fixing Vulnerabilities in Information Systems: The Vulnerability Assessment and Mitigation Methodology*, RAND, Pittsburgh, (2003)
8. R. Kasprzyk, A. Stachurski, A concept of standard-based vulnerability management automation for IT systems, Comp. Sc. and Math. Mod., **3**, 33-38, (2016)
9. O. H. Alhazmi, Y. K. Malaiya, I. Ray, Measuring, analyzing and predicting security vulnerabilities in software systems, Comp. & Sec., **26**, 219-228, (2007)
10. The MITRE Corporation, *Common Weakness Scoring System (CWSS™)*, [Online] https://cwe.mitre.org/cwss/cwss_v1.0.1.html, (2014)
11. A. K. Shrivastava, R. Sharma, P. K. Kapur, Vulnerability Discovery Model for a Software System Using Stochastic Differential Equation, Proc. of 2015 1st Int. Conf. on Fut. tr. in Com. An. and Kn. Man. *(ABLAZE-2015)*, IEEE, Amity University Greater Noida, 199–205, (2015)
12. R. Hoffmann, Vulnerability Discovery Models for a Software System Using Stochastic Differential Equations, Rocz. Koleg. An. Eko. / S.G.H, **45**, 177-187, (2017)
13. E. Rescorla, Is Finding Security Holes a Good Idea?, IEEE Sec. and Pri., **3,** 1, 14-19, (2005)
14. A.L. Goel, K. Okumoto, Time-Dependent Error Detection Rate Model for Software and Other Performance Measures, IEEE Tr. on Rel., **R-28** , 3, 206–211, (1979)
15. S. A. DeLurgio, *Forecasting principles and applications*, University of Missouri-Kansas City, Irwin/McGraw-Hill, 1998.
16. H. Tomaszek, M. Wróblewski, *Podstawy oceny efektywności eksploatacji systemów uzbrojenia lotniczego*, Bellona, Warsaw (2001).
17. M. Zieja, A method of predicting reliability and lifetime of aeronautical hardware with characteristic function applied. Transport Means – Proceedings of the International Conference, Kaunas, 22-23 October 2015. Kaunas Univ. Technol.
18. M. Zieja, M. Ważny, S. Stępień, Distribution determination of time of exceeding permissible condition as used to determine lifetimes of selected aeronautical devices/systems. Eksploatacja i Niezawodnosc-Maintenance and Reliability, **18(1)**, 57-64 (2016)
19. J. Żurek, Z. Smalko, M. Zieja, Methods applied to identify causes of air events. *Reliability, Risk and Safety: Theory and Applications*. CRC Press-Taylor and Francis Group, 1817-1822, (2010).