Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/080228

An Analytical Security Model for Existing Software Systems

Ayaz Isazadeh[1], *Islam Elgedawy*[2,*], *Jaber Karimpour*[1] *and Habib Izadkhah*[1,*]

[1] Department of Computer Science, Faculty of Mathematical Sciences, University of Tabriz, Tabriz, Iran
[2] Department of Computer Engineering, Middle East Technical University, Northern Cyprus Campus, Mersin 10, Turkey

Received: 1 Apr. 2013, Revised: 2 Aug. 2013, Accepted: 4 Aug. 2013
Published online: 1 Mar. 2014

Abstract: Nowadays, evaluation of software security, as one of the important quality attributes, is of paramount importance. There are many software systems have not considered security in their design; this makes them vulnerable to security risks. Architecture is the most important consideration in software design that affects final quality of software. Quality attributes such as efficiency and reliability have been studied at software architecture level; however, no report has ever been provided about the effect of software architecture on security. The purpose of this paper is to propose a mathematical-based method for evaluating and quantifying software security using the coupling aspects of the software architecture. To achieve this goal, first, we show the relationship between coupling types and vulnerability using an empirical-based software engineering technique that adopts Mozilla Firefox Browser vulnerability data. Then, we propose a mathematical weighted relationship between coupling types and vulnerability, where regression statistical analysis and Mozilla Firefox vulnerability data are used to predicate the relationship coefficients. Finally, we extract software architecture using DAGC tool and then convert the extracted architecture into Discrete Time Markov chains, which are used to predict and compute the system over all vulnerability.

Keywords: Security, Software Architecture, Mozilla Firefox Browser, Markov chains

1 Introduction

Our ongoing research area is focused on extracting dynamic and adaptive software system architecture from source code. Dynamism and adaptability with the environment are among the features of the next generation of software systems. A software system must be able to adapt itself with a continuously changing environment and modify its performance as a reaction to such changes. Dynamic software systems have dynamic architectures; therefore, software architecture has a close relationship with the dynamism and adaptability of the software system. Software architecture specifies the performance of the software system in different situations and different environmental interactions. Dynamic software system architecture, generally, has been defined as the ability of the software system to change its performance, in various interactions with the environment. Specifically, in our work, dynamic software architecture is considered as the design of the software system in a way that it can contain security, efficiency, power awareness, and distribution capability, where it should be able to increase its efficiency when it is important, improve its security when the security is the matter of significance, decrease its power consumption when the issue is the lack of energy (e.g., mobile systems and laptop computers,) and distribute itself in a distributed environment for deployment. Existing methods on dynamic software architectures are merely performable in primary phases of the software development. No method has been developed, so far, to extract the dynamic architecture of a software system from its source code. Existing methods have only provided a general framework to define the architecture under consideration; they have not presented an efficient pattern, which can provide the features of our concern (efficiency, security, distribution, and power aware energy assumption.)

In our view, there is a significant relationship between internal metrics of the software, such as cohesion and coupling, and external characteristics of the software system, such as efficiency, security, distribution, and power aware energy assumption. We propose, and intend,

* Corresponding author e-mail: elgedawy@metu.edu.tr, izadkhah@tabrizu.ac.ir

to find a mathematical relation between the internal software metrics and the features of our concern based on empirical software engineering methods. We further plan to develop a programming language independent toolset for extracting a suitable architecture from source code. This paper explores the security aspect of our ongoing research, as outlined above, which deals with extracting security based architecture from source code.

Secure software is a core requirement of the modern world, because, many of critical processes, such as air-traffic control and online banking, are software-related work to do. Unfortunately, there is an increasing rate of occurrence of software security failures. Most security specialists believe that threats are originated from network platform (which includes network hardware, network operating systems and protocols), whereas some global documented statistics available in authentic institutes state the contrary. According to the study conducted by Gartner Incorporation, 75% of cyber attacks are done at web application level and only 25% focused on network and network services, while of the total budget spent on security, 10% was allocated to the programs and 90% to network.

Software vulnerability is a weakness in a software system that allows to an attacker to use the system for a malicious purpose. Ivan Krsul [1] defined software vulnerability as an instance of a fault in the specification, development, or configuration of software such that its execution can violates an implicit or explicit security policy. Hogland and McGraw [2] define software security as the ability to defend attackers exploitation of software problems by building software to be secure throughout the development cycle. We use the term fault to denote any software fault or defect, and reserve vulnerability for those exploitable faults which might lead to a security failure.

Coupling is one of the internal software quality attributes that affect overall software quality [3]. Coupling in software architecture refers to the level of interconnection and dependency among software modules. Modules are said to be highly coupled when they depend on each other to such an extent that a change in one requires changes in others dependent upon it. High coupling hinders program comprehension [4]. There are five types of couplings [5]: (1) content, (2) common, (3) control, (4) stamp, and (5) data coupling. These five levels are considered as severity level from high to low, where the highest severity level, content coupling, is the worst in the software engineering principles. Data coupling is the type of coupling where data is the only factor of dependence among the modules. Stamp Coupling is the type of coupling when an entire data structure is shared between modules. In the Control Coupling one module passes parameters to control the activity of another module. Common coupling occurs when two modules share the same global data (e.g., a global variable or a complete DB table). In the Content Coupling one module modifies the internal data item in another module.

It is difficult that a software system can be developed without considering one or more of these types of coupling. However, it is well accepted that content coupling, be avoided as much as possible. Coupling shows the relationship between various modules, therefore, when information is passed among modules there is high potential that it is also exposed.

Numerous studies [6,7,8,9,10,11,12] show that coupling is a feature that has a significant impact on some critical software quality attributes such as reliability, portability, reusability, operability, flexibility, testability and maintainability, and, as a side effect, may introduce faults in software systems. In [13,14], is shown that high coupling tends to increase damage propagation when a vulnerability is exposed. Ayanam [15] has also shown that many forms of SQL injection or buffer overflow attacks are based on exploitations of certain types of coupling.

The relationship between coupling and vulnerability is shown in reference [14] using the empirically-based software engineering on Mozilla Firefox browser. Machine learning was used for this purpose; no mathematical relation was used to show this relationship. In fact, the tool presented by them receives number of couplings, cohesions and complexities and predicts security level of software. Here, we continue their research and create a formula for specifying and evaluating software security using the vulnerability data collected by them from Mozilla Firefox Browser. The difference between our research and their research is that they assume coupling as a general concept and assume the relationship between one entity and the other entity as a coupling, whereas there are different types of coupling and the results obtained from our experiments showed that the effect of each types of coupling on security is different from the others.

Our intuition is that coupling may, as well, lead to introduction of vulnerabilities weaknesses that can be exploited by malicious users to compromise a software system. In fact, we think that coupling should be considered as a factor that can affect software security. Although the effect of coupling is shown empirically and successfully in software faults, however, no mathematical relation was presented to show how coupling affects security. No discussion was also made on the effect of type of coupling. Here, we will propose a mathematical relation based on the types of coupling (such as Data, Stamp, Control, Common and Content) and vulnerability. Then, using the obtained mathematical relation, we will predict software security from software architecture.

1.1 The Problem and the claim

The overall problem addressed in this paper is to find mathematical relation for predicting existing software security upon software architecture. A solution to this problem should be the following specific characteristics:

1. Studying the possibility of relationship between type of coupling and vulnerability
2. In case there is a relationship, a mathematical relation is offered showing the relationship between type of coupling and vulnerability
3. Studying the security of the existing software system from the source code with respect to the mathematical relation obtained in item 2

To perform the items mentioned above, we will show the correlation between coupling types and vulnerability in section 3 using the statistical analysis concepts based on the vulnerability data collected from Mozilla Firefox software. After specifying correlation, we will use statistical regression for proposes a mathematical weighted relationship, which presents the relationship between type of coupling and vulnerability in each module. With respect to proposed mathematical weighted relationship, we will determine the expected security problems in a software system in section 3. Finally, to evaluate security of a software system, first, we will extract its architecture from source code using DAGC tool and then we will convert the extracted architecture into Discrete Time Markov Chains (DTMCs) for evaluation. We use DTMCs to model software systems and provide expressions for predicting the overall behavior of the system based on its architecture as well as the characteristics of individual modules. We will evaluate the software system using its DTMCs, and the expected security problems formalism offered in section 3.

1.2 Paper Outline

Other sections of this paper have been organized as follows: section 2 provides background on Software Architecture, Mozilla Firefox, Discrete Time Markov Chains and Correlation and Regression techniques. In section 3, first, the relationship between different types of coupling and vulnerability is shown using the empirically-based software engineering; then, we propose a mathematical relationship between different types of coupling and vulnerability. Case study is discussed in sections 4. At the end, section 5 deals with conclusions, study limitations and future works.

2 Background

This section provides background on Software Architecture, Mozilla Firefox Browser, Discrete Time Markov Chains, Correlation and Regression techniques used in this study to detect vulnerabilities.

2.1 Software Architecture

The software architecture of a system is the structure or structures of the system, which comprises software elements, the externally visible properties of those elements, and the relationships among them [16]. Modularization is a key activity in reverse engineering to extract software architecture. The goal of the software modularization process is to automatically partition the classes of a system into modules (or subsystems, i.e. a number of interrelated classes) so that minimizes connections between the classes of two different modules (called coupling), as well as maximizing connections between the classes of the same module (called cohesion.) The domain experts, cohesion and coupling are two features that have a significant impact on some critical software quality attributes such as reliability, portability, reusability, operability, flexibility, testability and maintainability [5,17]; therefore, to reduce costs and software design, cohesion and coupling management is very crucial. Therefore, we apply two criteria, namely cohesion and coupling for modularizing to extract software architecture.

The well-known and most popular tools for extracting software architecture are DAGC [18], Bunch [19]. One of the important issues related to modularization algorithms is largeness of search space. Search space in Bunch algorithm is nn; this large search space decelerates speed of this algorithm to find appropriate architecture. By introducing an efficient coding, DAGC algorithm managed to reduce this search space from n^n to n!. Therefore, we used DAGC, in this paper, for extracting software architecture.

For modularization process, to extract software architecture by these tools, they as input need an analysis tool to convert source code to Call Dependency Graph (CDG). The CDG vertices include system classes, and edges show the dependency between classes. We use the Ndepend tool for generation CDG from source code as input for DAGC.

2.2 Overview of Mozilla Firefox

Since, the Mozilla Firefox is the source of vulnerability data for our study; therefore we provide a description of that. The Mozilla Firefox, an open source browser, is one of the most popular web browsers with an approximate user-base of 320 million (SciTools Inc. Blog, http://scitools.com/blog/2008/10/tip-understand-the-countpath-metric.html). The Mozilla Firefox project is large in terms of source code (each release of the browser includes more than 2 million lines of code). Moreover, it has a rich history of publicly available vulnerability fixes over a period of four years (January 26, 2005 to April 27, 2009) [20]. We have conducted case study on different releases of Mozilla Firefox (releases are analogous to versions). Until March 1, 2009, fifty-two releases of Mozilla Firefox have had the vulnerability fixes, from Release 1.0 (R1.0) to Release 3.0.6 (R3.0.6). To validate this study, we have collected vulnerability information

from all fifty-two releases as the vulnerabilities are distributed amongst all the releases.

Over the aforementioned period of four years and fifty-two releases, 718 of the total of 11,139 files have had vulnerability fixes. In total, these 718 files suffered from about 1450 vulnerability fixes ranging from one vulnerability fix per file to more than five vulnerability fixes. In the Mozilla Firefox 454 files have had one vulnerability fix; 141 files have had two vulnerability fixes; 46 files have had three vulnerability fixes; and so on.

2.3 Overview of Discrete Time Markov Chains

In this section, we discuss Discrete Time Markov Chains (DTMCs) [21,22], which we use to model the software architecture. In a DTMC, the state space is discrete, and the parameter space, is also discrete. A DTMC is described by its states and transition probabilities among the states. The one-step transition probability is the probability that the process, when in state i at time n, will next transition to state j at time n + 1. We write:

$$P_{ij}(n) = P(X_{n+1} = j | X_n = i) \tag{1}$$

Definition: one-step transition probability matrix, $P = [p_{i,j}]$, is formed by arranging the one-step transition probabilities into a $N \times N$ matrix: Note that all the

$$P \stackrel{\text{def}}{=} \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1N} \\ p_{21} & p_{22} & \cdots & p_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ p_{N1} & p_{N2} & \cdots & p_{NN} \end{pmatrix}$$

elements in a row of P add up to 1 and each of the $p_{i,j}s$ lie in the range [0, 1]. For our purposes, we use absorbing DTMC. One DTMC is called absorbing if at least one state has no outgoing transition. Each DTMC with several final states can be converted into an absorbing DTMC. It is performed by adding a final state to DTMC. Next, a transition is drawn to the added absorbing state from all the final states available in DTMC. We can partition the transition probability matrix of an absorbing DTMC as:

$$P = \begin{bmatrix} 1 & 0 \\ C & Q \end{bmatrix} \tag{2}$$

If the DTMC has n states with m absorbing states, Q would be a $(n-m) \times (n-m)$ sub-stochastic matrix (with at least one row sum < 1) describing the probabilities of transition only between transient states, 1 being a $m \times m$ identity matrix, 0 would be an $m \times (n-m)$ matrix of zeros, and C would be an $(n-m) \times m$ matrix describing the probabilities of transition between transient states and absorbing state. The k-step transition probability matrix, given by P^k has the form:

$$P' = \begin{bmatrix} 1 & 0 \\ C' & Q' \end{bmatrix} \tag{3}$$

The (i, j)th entry of Qk denotes the probability of arriving in state s_j after exactly k steps, starting from state s_i . Hence the inverse matrix $(I-Q)^{-1}$ exists. This is called the fundamental matrix F:

$$F = (I-Q)^{-1} \tag{4}$$

Let $X_{i,j}$ represent the number of visits to state j starting from state i before the process is absorbed. It can be shown that the expected number of visits to a state j with starting from state i (i.e, $E[X_{i,j}]$), before entering an absorbing state is given by the (i, j)th entry of the fundamental matrix F [21,22]. So

$$E[X_{i,j}] = m_{i,j} \tag{5}$$

$m_{i,j}$ is the (i, j)th entry of the fundamental matrix F. The variance of the expected number of visits could also be computed using the fundamental matrix. Let $\sigma_{i,j}$ denote the variance of the number of visits to the state j starting from state i. Define $F_D = [md_{i,j}]$ such that:

$$md_{i,j} = \begin{cases} m_{i,j}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

In other words, F_D represents a diagonal matrix with the diagonal entries same as that of F. If we define $F_2 = [m^2_{i,j}]$, we have:

$$\sigma^2 = F(2F_D - I) - F_2 \tag{7}$$

Hence

$$Var[X_{i,j}] = \sigma^2_{i,j} \tag{8}$$

For an application consisting of a number of software modules, we can represent its software architecture using a DTMC such that DTMC states represent the software modules, and the transitions between states represent transfer of control from one module to another.

2.4 Correlation coefficient and Regression

Correlation coefficient is used to measure the correlation of related two variables. It is a statistical technique to measure the type and degree of a relation of a quantitative variable with another quantitative variable. A correlation coefficient shows the intensity of a relation and the type of a relation (direct or reversed). This coefficient is between 1 and -1 and if there is no relation between two variables, it is equal to zero.

We evaluate our claim about relationship between vulnerability and type of coupling by computing the correlations between the type of coupling and the number of vulnerabilities in Mozilla Firefox. The value of the correlation coefficient gives the strength of the relationship. However, the interpretation depends on the context of the usage of correlation. Cohen [23] suggest that a correlation of less than 0.3 means weak correlation, 0.3 to 0.5 means medium correlation, and greater than 0.5 means strong correlation. We interpret the strength of correlation as per Cohen et al. conventionally, the significance of a correlation is determined in terms of p-value, the probability of the t-statistic. The smaller the p-value, the higher is the confidence on the significance of the correlation. Traditionally, correlations with p-values of less than or equal to 0.05 are considered statistically significant [23]. A p-value of 0.05 means that we are 95% confident that the observed correlation is not by chance. The Pearson correlation coefficient and Spearman rank correlation coefficient are often used to measure the strength of correlations between two variables.

The Pearson correlation assumes normal distribution of data, while the Spearman rank correlation is a non-parametric test that does not assume any distribution. Spearman rank correlation is performed on the ranks of the values without considering the magnitudes of the values, and therefore, it is not sensitive to outliers. Spearman rank correlation is a commonly used and robust correlation technique because it can be applied even when the association between elements is non-linear [23]. For these reasons, we use the Spearman rank correlation for this study. We have used SPSS (www.spsstools.net), a statistical analysis tool, to compute the Spearman rank correlation and its corresponding p-value.

We analyze the correlation between type of coupling and vulnerabilities on fifty-two releases of Firefox developed till March 1, 2009 fall into four major releases: R3.0, R2.0, R1.5, and R1.0 (Mozilla Vulnerabilities, http://www.mozilla.org/projects/security/knownvulnerabilities). An independent variable is the variable that affects other variables. In an experimental research, an independent variable is the variable, which is manipulated by a researcher so that its effect on the dependent variable is specified. A dependent variable is the variable whose value or quantity depends on an independent variable. A dependent variable is not under the control of a researcher and he/she is unable to manipulate it.

Regression analysis is a statistical technique for studying the relationships between the independent and dependent variables [23]. It provides an opportunity for a researcher to predict changes of a dependent variable through an independent variable and determine the share of each independent variable in explaining a dependable variable. The difference between regression and correlation coefficient is that regression looks for prediction, whereas correlation coefficient only compares level of dependency of two variables. When there is a correlation between two variables, value of a variable (Y) can be predicted or estimated by the other variable (X) using regression. The higher the correlation between variables is, the more accurate a prediction will be. The least squares method is used to construct the regression. In fact, the least squares method is a method for fitting data. In this method, the best model fitted on a series of data is the model in which sum of the squared residuals is minimum. Residuals mean the difference between the observed data and the value obtained from a model.

Coefficient of Determination (R^2) value is used as a guideline to measure the accuracy of the data model [23]. It used to describe how well a regression line fits a set of data. R^2 is most often seen as a number between 0 and 1, an R^2 near 1 indicates that a regression line fits the data well, while R^2 closer to 0 indicates a regression line does not fit the data very well. For example, if $R^2 = 0.850$, which means that 85% of the total variation in y can be explained by the linear relationship between x and y (as described by the regression equation). The other 15% of the total variation in y remains unexplained.

3 Vulnerability Detection

The only research carried out on the relationship between the internal software qualities attributes and security was issued in System Architecture Journal in 2011 [14]. The study using an empirical case study of Mozilla Firefox showed that there is a strong correlation between coupling and security (the type of coupling has not been specified). Alexander Ivanov [24] studied different software codes in his thesis and showed that there is a reversed relationship between software security and holes inside code. A research conducted in Queens University in Canada in 2011 [14] revealed that there is a relationship between holes inside code and software coupling (The type of coupling has not been specified). In his software engineering book, Pressman [5] categorized different types of couplings as Data Dependency- Control Dependency- Stamp Dependency- Common Dependency- Content Dependency.

It can be concluded from what stated above that there are probably reciprocal a relationships between five items specified by Pressman and software security. To show such relationship, we use Spearman rank correlation coefficient. The Spearman rank correlation coefficient (denoted by correlation in short) between type of coupling and the number of vulnerabilities in each file of the five major releases of Mozilla Firefox are presented in Table 1 and Fig. 1. Coupling metrics are collected using NDepend code analysis tool (www.ndepend.com). Mozilla vulnerability bug report is obtained from (Bugzilla, http://www.bugzilla.org) [25]. The correlations between type of coupling and vulnerabilities are slightly above 0.5 for all the releases. Therefore, generally type of coupling metrics are moderately correlated to vulnerabilities in Mozilla Firefox. The Content is strongly

correlated (0.6-0.7) to vulnerabilities. Therefore, it may be a good indicator of vulnerabilities. The overlapping lines (one line for each major release) in Fig. 1 illustrates that the correlation patterns are consistent across releases. The observed correlations are statistically significant as indicated by the small p-value of t-test ($p < 0.05$). We chose t-statistics over z-score because z-score is used when the population mean and population standard deviation are known for the original population. We can make two inferences from these observations. First, type of coupling may be useful in vulnerability prediction as they demonstrate moderate but significant correlations to vulnerabilities. Second, because the correlations are positive, highly coupled files are likely to have more vulnerabilities than less coupled files.

Table 1
Correlations between type of coupling and vulnerabilities in Mozilla Firefox.

Type of coupling	Correlations with vulnerabilities				
	R1.0	R1.5	R2.0	R3.0	R3.0.6
Data	0.471	0.462	0.458	0.458	0.454
Stamp	0.482	0.482	0.482	0.471	0.469
Control	0.510	0.514	0.515	0.510	0.509
Common	0.537	0.536	0.536	0.521	0.519
Content	0.641	0.641	0.634	0.631	0.631

For all the correlations, p<0.05.

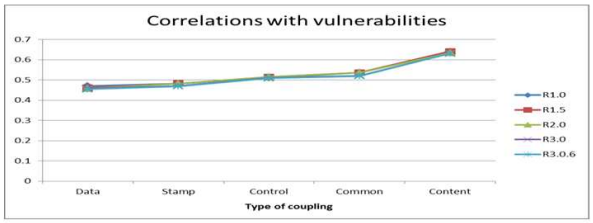


Fig. 1: Plot of correlations between types of coupling vs. the number of vulnerabilities in Mozilla Firefox

We observe from the table that all the coupling metrics are generally positively correlated to the number of vulnerabilities in Mozilla Firefox across all five studied releases. This unequivocally suggests that highly coupled files have higher number of vulnerabilities. Table 1 show that our intuition on the relationship between types of couplings and vulnerabilities is true and there is a significant relationship between them. We aim to establish a mathematical relationship between types of couplings and vulnerabilities. In general, coupling is directly related to vulnerability and vulnerability is inversely related to software security i.e.,

Coupling $\equiv$ Vulnerability $\equiv$ 1/Security

In this section, we devise a mathematical relation for studying the association between coupling and vulnerability. There are multiple types of coupling, and we capture their contributions to vulnerability by taking all of them into account through a weighted relation, called P(x).

We reviewed different types of relationships to propose a mathematical relationship between types of coupling and vulnerabilities. We calculated their R^2 and observed that a polynomial relationship can be a good estimation. Therefore, at first, we propose the Eq. (9) with maximum degree of three. We determined their coefficients using the vulnerability data collected from Mozilla Firefox and regression analysis of their coefficients. Finally, we will attempt to delete the statements with no effect on the relation using statistical analysis. We performed the regression using SPSS tool.

$$P_m(x) = c + \sum_{i=1}^{5} w_i C_i + \sum_{i=1}^{5} \alpha_i C_i^2 + \sum_{i=1}^{5} \beta_i C_i^3 + \sum_{i,j=1, i\neq j}^{5} \phi_{ij} C_i C_j \quad (9)$$

m = 1,..., Number of modules
C1 = number of Data Dependency in a module
C2 = number of Control Dependency in a module
C3 = number of Stamp Dependency in a module
C4 = number of Common Dependency in a module
C5 = number of Content Dependency in a module

$c, w_i, \alpha_i, \beta_i$ and ϕ_{ij} are weight and importance of coupling types, respectively. C_iC_j indicate the effect of type of coupling on each other. We use the vulnerability data of Mozilla Firefox software to determine the coefficients. $c, w_i, \alpha_i, \beta_i$ and ϕ_{ij} are obtained from the least squares method. These values are achieved by minimizing total squares of deviations. To do this, total squares of errors should be derived in ratio of W and be equal to zero. This will result in 26 equations, which are called Normal Equations. C_is are specified for each entity. In fact, there are 26 unknowns in the equations (as $c, w_i, \alpha_i, \beta_i$ and ϕ_{ij}).

We use vulnerability data of Mozilla Firefox Browser to determine C_is. Mozilla Firefox has 11139 files. Seven hundred and eighteen files out of 11139 files had vulnerabilities, which were fixed. Therefore, here, we use these 718 files to determine the coefficients of our formula. For each Mozilla Firefox file, we specify number and type of coupling using Ndepend tool and we specify number and type of vulnerability using the history of vulnerability (number of vulnerabilities in module i is shown by $P_i(x)$) and we put them in Eq. 9.

We used a commercial tool called NDepend to automatically calculate the metrics from the source code. This tool is used because it is user friendly and it has a good set of APIs to interact with programming languages such as C++, Perl, and Python. In this study, we compute the metrics at module-level granularity. Table 2 shows the obtained coefficients.

First, we calculated Coefficient of Determination or R^2, which is equal to 0.58 for our model. The bigger the value of R^2 is, the better the fitting of a model will be.

Table 2
Obtained coefficients for relation 9.

c	w_1	w_2	w_3	w_4	w_5	α_1	α_2	α_3	α_4	α_5
0.177608	0.011917	0.024287	0.100231	0.347872	0.428921	0.000911	0.000923	0.000981	0.100533	0.110542
β_1	β_2	β_3	β_4	β_5	$\varphi_{1,2}$	$\varphi_{1,3}$	$\varphi_{1,4}$	$\varphi_{1,5}$	$\varphi_{2,3}$	$\varphi_{2,4}$
0.000012	0.000034	0.000038	0.000124	0.000139	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
$\varphi_{2,5}$	$\varphi_{3,4}$	$\varphi_{3,5}$	$\varphi_{4,5}$							
0.000000	0.000000	0.000000	0.000000							

Number 0.58 indicates that there is a significant relationship between the independent variable (type of coupling) and the dependent variable (number of vulnerability). Now, we discuss each criterion (independent variable) with respect to their coefficient. We aim to study which one has a significant effect on the polynomial. To do this, we used t-Statistic. To study the significance of each coefficient, we need to compare the calculated t with the amount of tabulated t with degree of freedom of n-k (n is number of vulnerability data and k is number of unknown in equations). Approximately, for significance of each coefficient at the level of 95%, must to have $|t| \geq 2$. Table 3 shows the value of t-Statistic for all the obtained coefficients.

Table 3
Obtained t-Statistic for all the obtained coefficients for relation 9.

c	w_1	w_2	w_3	w_4	w_5	α_1	α_2	α_3	α_4	α_5
2.618319	2.076532	2.087455	5.12145	13.46926	24.54306	1.002316	1.074754	1.439125	5.000328	5.010423
β_1	β_2	β_3	β_4	β_5	$\varphi_{1,2}$	$\varphi_{1,3}$	$\varphi_{1,4}$	$\varphi_{1,5}$	$\varphi_{2,3}$	$\varphi_{2,4}$
0.363781	0.363841	0.473241	0.498342	0.843962	0.001280	0.025712	0.024329	0.876575	0.004759	0.700932
$\varphi_{2,5}$	$\varphi_{3,4}$	$\varphi_{3,5}$	$\varphi_{4,5}$							
0.039341	0.459121	0.042461	0.981241							

Therefore, after deleting the coefficients with no significant effect on vulnerability, the proposed general model is as follows:

$$P_m(x) = c + \sum_{i=1}^{5} w_i C_i + \alpha_4 C_4^2 + \alpha_5 C_5^2 \qquad (10)$$

m = 1,..., Number of modules

After deleting the variables with no significant effect on the dependent variable, we recalculated R^2, which was equal to 0.79. In fact, 0.79 indicates that our selected polynomial can show the relationship between type of coupling and vulnerability well. $P_m(x)$ can range in value from 0 to some very large number. The question is how large is very large vulnerability? We need some sense of the scale of vulnerability, and $P_m(x)$ does not provide that. Thus a little more elegant metric called vulnerability index, DV, is defined as follows.

$$DV_m = 1 - (1/P_m(x)) \quad where \quad 0 < DV_m < 1 \qquad (11)$$

m = 1,..., Number of modules

Where, $DV_m = 0$ implies possibly low or no coupling and possibly less vulnerable, and $DV_m = 1$ implies possibly high coupling and possibly highly vulnerable. We have described in Section 5.1, the limitations in this study.

3.1 Prediction Performance Measures

The performance of a predictor can be measured in several ways. Most frequently used measures are accuracy, recall, precision, false positive rate, and false negative rate. These performance measures are explained using a confusion matrix, shown in Table 4. The confusion matrix shows the actual versus the predicted results where:

- –True Negative (TN) = The number of files predicted as not being vulnerability-prone where no vulnerability is discovered in those files.
- –False Positives (FP) = The number of files incorrectly predicted as vulnerability-prone when they are not vulnerable.
- –False Negative (FN) = The number of files predicted as not being vulnerability-prone which turn out to have a vulnerability.
- –True Positives (TP) = The number of files predicted as being vulnerability-prone which are in fact vulnerable.

Table 4
Confusion matrix

Actual	Predicted as	
	Not Vulnerable	Vulnerable
Not Vulnerable	TN=True Negatives	FP=False Positives
Vulnerable	FN= False Negatives	TP= True Positives

From the confusion matrix, several of the prediction performance measures such as accuracy, precision, recall, F-measures, false positive rate, and false negative rate can be derived as follows:

Accuracy: Accuracy is also known as overall correct classification rate. It is defined as the ratio of the number of files correctly predicted to the total number of files as shown in Eq. (12).

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \qquad (12)$$

Precision: Precision, also known as the correctness, measures the efficiency of prediction. It is defined as the ratio of the number of files correctly predicted as vulnerability-prone to the total number of files predicted as vulnerability-prone, as shown in Eq. (13).

$$Precision = \frac{TP}{TP + FP} \qquad (13)$$

Recall: Recall is the vulnerable entity detection rate which quantifies the effectiveness of a predictor. It is defined as the ratio of the number of files correctly predicted as vulnerability-prone to the total number of files that are actually vulnerable. The formulae to calculate recall is given in Eq. (14).

$$Recall = \frac{TP}{TP+FN} \tag{14}$$

Both precision and recall are important performance measures. The higher the precision, the less effort wasted in testing and inspection, and the higher the recall, the fewer vulnerable files go undetected. Unfortunately, there is a trade-off between precision and recall. For example, if a predictor predicts only one file as vulnerability-prone and this file is actually vulnerable, the precision will be 100%. However, the predictor recall will be low if there are other vulnerable files. In another example, if a predictor predicts all files as vulnerable, the recall will be 100%, but its precision will be low. Therefore, a measure is needed which combines recall and precision in a single efficiency measure.

F-measure: F-measure can be interpreted as a weighted average of precision and recall [26]. For convenient interpretation, we also express it in terms of a percentage like our other performance measures so it reaches its best value at 100 and its worst at 0. The general formula for F-measure is given in Eq. (15), where F_β-measure "measures the effectiveness of prediction with respect to a user who attaches β times as much importance to recall as precision."

$$F_\beta - measure = \frac{(1+\beta^2) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall} \tag{15}$$

The traditional F-measure, denoted by F_1-measure, gives equal importance to both precision and recall by taking their harmonic mean. Two other commonly used F-measures are the F_2-measure and $F_{0.5}$-measure. F_2-measure weighs recall twice as much as precision whereas $F_{0.5}$-measure weighs precision twice as much as recall. Some researchers choose to use the false positive rate (FP rate) and the false negative rate (FN rate) instead of precision and recall. Ostrand and Weyuker[26] in particular argue that false positive rate and false negative rate are the most important measures. We also believe that these measures are effective in evaluating vulnerability prediction models. They are defined as follows:

$$FPrate = \frac{FP}{FP+TN} \tag{16}$$

$$FNrate = \frac{FN}{TP+FN} \tag{17}$$

A high FN rate indicates that there is a risk of overlooking vulnerabilities, whereas a high FP rate indicates effort may be wasted in investigating the predicted vulnerable entities. These notions are highly related to recall and precision. Therefore, it is redundant to use all of them to indicate prediction performance. In this study, we use accuracy, recall and FN rate as employed in [26]. All these measures are expressed in percentages. In addition, we use F_1-measure to quantitatively evaluate and compare the predictors.

We now examine the performance Eq. (11) in predicting of the modules vulnerability. To do this, we use the 20 version of Mozilla Firefox from R2.0.0.10 to R3.0.6. The goal is to determine the Eq. (11) to what extent can correctly predict the vulnerability in a component or module level. The results of performance evaluation of Eq. (11) in Table 5 are given.

Table 5
Performance the relation (11) in predicting vulnerability in Mozilla Firefox versions R2.0.0.10 to R3.0.6

Accuracy	Recall	FP rate	F_1-measure
72.51	69.22	28.12	68.00

For more investigation the Eq. (11) to predict the vulnerability of modules, ten modules has the greatest security holes found in the Apache Software has been studied. Therefore, the vulnerability of these modules by the Eq. (11) is calculated. The results in Table 6 are given. As shown in Table (6) is observed for these vulnerable modules, DV_m value is high (number is close to 1). This table shows that the Eq. (11) it can be used to show a vulnerability of the module.

Table 6
Relation between DV_m and ten most vulnerable component of the Apache Software

Component	Bug Count	DV_m
jsobj. cpp	24	0.937
nsCSSFrameConstructor.cpp	17	0.913
jsfun. cpp	15	0.891
nsScriptSecurityManager.cpp	15	0.881
NsGlobalWindow.cpp	14	0.891
jscsript. cpp	14	0.894
jsinterp. cpp	14	0.894
nsDOMClassinfo.cpp	10	0.892
nsGenericElement.cpp	10	0.892
nsDOCShell.cpp	9	0.832

3.2 Detecting the vulnerability Overall software system

After calculating vulnerabilities of a module, we examine the vulnerability of software system. If DV_i indicates

vulnerability index of module i and $X_{1,i}$ indicates the random variable corresponding to the number of visits of state i starting from state 1, the possibility of a module to be able to act without security failure during its run on the software system will be equal to $[(1-DV_i)^{X_{1,i}}]$. Therefore, the probability of i^{th} module with security problems during running of a software system is as follow:

$$1-[(1-DV_i)^{X_{1,i}}] \tag{18}$$

Consequently, for a software system with n modules (or components), the vulnerability index may be defined as follows:

$$DV = 1-\Pi_i^n(1-DV_i)^{X_{1,i}} \tag{19}$$

Where DV is the random variable which shows vulnerability index of a software system. Therefore, the expected of vulnerability index of software system is as follows:

$$\begin{aligned} E[DV] &= E[1-\Pi_i^n(1-DV_i)^{X_{1,i}}] \\ &= 1-E[\Pi_i^n(1-DV_i)^{X_{1,i}}] \\ &= 1-\Pi_i^n E[(1-DV_i)^{X_{1,i}}] \end{aligned} \tag{20}$$

Using the Taylor series expansion, $E[(1-DV_i)^{X_{1,i}}]$ in Eq. (19) can be written as Eq. (21).

$$\begin{aligned} E[(1-DV_i)^{X_{1,i}}] &= (1-DV_i)^{m_{1,i}} \\ &+ \frac{1}{2}(1-DV_i)^{m_{1,i}} \times (\log(1-DV_i))^2\sigma_{1,i} \end{aligned} \tag{21}$$

As number of visits of the last module is always 1. That is, E[$X_{1,n}$]=1 and Var[$X_{1,n}$]=0 as:

$$E[(1-DV_n)^{X_{1,n}}] = 1-DV_n \tag{22}$$

Therefore, expected of vulnerability index of second-order is as follows:

$$\begin{aligned} E[DV] &= 1-[\Pi_i^{n-1}[(1-DV_i)^{m_{1,i}} + \frac{1}{2}(1-DV_i)^{m_{1,i}} \\ &\times (\log(1-DV_i))^2\sigma_{1,i}]](1-DV_n) \end{aligned} \tag{23}$$

The bottleneck of security of a software system is a module with the maximum amount of $1-E[(1-DV_i)^{X_{1,i}}]$.

3.3 Sensitivity Analysis

Here, we intend to study the general impressionability of a software system on number of the parameters related to security of a module (for instance, changing vulnerability of a module). We use differential equation to study the effect of changes on the parameters related to security of module k on the general software system. To do this, we calculate the derivative of expected of vulnerability index in software on DV_k. Therefore:

$$\begin{aligned} \frac{dE[DV]}{dDV_k} &= [m_{i,k}(1-DV_k)^{m_{1,k}-1} + \frac{1}{2}\sigma_{1,k}^2(m_{1,k}(1-DV_k)^{m_{1,k}-1} \\ &\times (\log(1-DV_k))^2 + \frac{2\log(1-DV_k)}{(1-DV_k)}(1-DV_k)^{m_{1,i}})] \\ &\times [\Pi_{i=1,i\neq k}^{n-1}((1-DV_i)^{m_{1,i}} + \frac{1}{2}((1-DV_i)^{m_{1,i}}) \\ &\times (\log(1-DV_i))^2\sigma_{1,k}^2)](1-DV_n) \end{aligned} \tag{24}$$

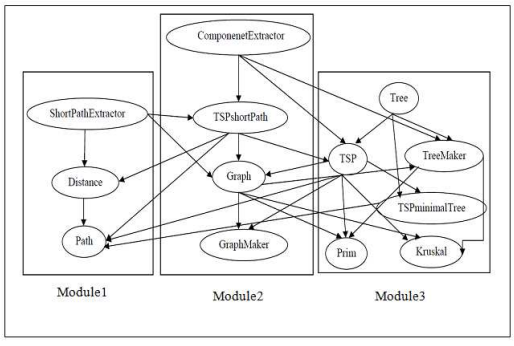
With respect to the modified vulnerability index of a software system as DV_{rev}, we define ΔDV_k as a change in security index in module k and if the original security index is DV_{org}, we have:

$$E[DV_{rev}] = E[DV_{org}] + (dE[DV]/dDV_n)\Delta DV_k \tag{25}$$

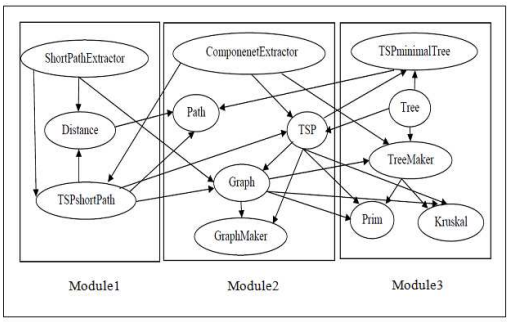
4 Case study

This section evaluates the proposed method. To do this, we use the well-known Travelling Salesman Problem (TSP). To evaluate the proposed method, we should extract TSP software architecture from its program using DAGC tool. The input and output of DAGC include call graph and software architecture respectively. To extract TSP call graph from its source code, we used a commercial tool called Ndepend. After extracting the call graph, it should be modularized to extract appropriate software architecture.

To evaluate TSP security from its architecture, we extract two architectures from its code with almost equal quality but with different modularizations. Figure 2 shows two extracted architectures for TSP. The edge between the two modules indicates that the dependency is of coupling type. In addition to the type of coupling, their numbers are also important.



(a)



(b)

Fig. 2: modularizations achieved through DAGC tool for TSP

After extracting architecture, we convert them to DTMC for evaluating security. The DTMC is as Fig. 3 for the architecture of Fig. 2(a) and it is as Fig. 4 for the architecture of Fig. 2(b). Numbers on edges indicate the probability of movement from one module to another module. In this paper the probability to go from module x to module y is computed as [number of method call from x to y/ total number of out method call of x (i.e. fan out)]. For example, in Fig. 2(a), totally 5 edges exited out of module 3, two edges went to Module 1, two edges to Module 2, and one edge to Module 4 (Module 4 is absorbing state and added automatically to DTMC.)

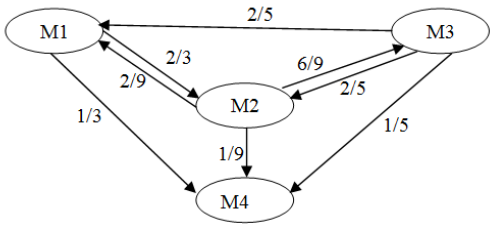


Fig. 3: Obtained DTMC for Fig. 2(a)

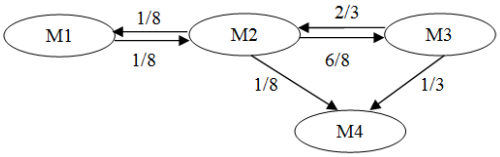


Fig. 4: Obtained DTMC for Fig. 2(b)

Transition probability matrix of an absorbing DTMC for Fig. 3 and Fig. 4 will be as Fig. 5. The fundamental matrix F for Fig. 5(a) and Fig. 5(b) is as Fig. 6. Table 7 shows DV_i for Modules 1, 2, and 3 of Fig. 2(a) and 2(b). Table 8 shows security index, that is, DV for both architectures with respect to Fig. 2.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1/3 & 0 & 2/3 & 0 \\ 1/9 & 2/9 & 0 & 6/9 \\ 1/5 & 2/5 & 2/5 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1/8 & 0 \\ 1/8 & 1/8 & 0 & 6/9 \\ 1/3 & 0 & 2/3 & 0 \end{bmatrix}$$

(a) (b)

Fig. 5: (a) Transition probability matrix for Fig. 3, (b) Transition probability matrix for Fig. 4

$$F = \begin{bmatrix} 1.766 & 1.584 & 1.045 \\ 1.161 & 2.400 & 1.584 \\ 1.175 & 1.594 & 2.052 \end{bmatrix} \qquad F = \begin{bmatrix} 1.028 & 0.227 & 0.150 \\ 0.227 & 1.822 & 1.202 \\ 0.150 & 1.202 & 1.793 \end{bmatrix}$$

(a) (b)

Fig. 6: (a) The fundamental matrix for Fig. 5(a), (b) The fundamental matrix for Fig. 5(b)

Table 7
DV_i for modularizations in Fig. 2(a) and Fig. 2(b).

	Modularization 1 (Fig. 2(a))			Modularization 2 (Fig. 2(b))		
	Module 1	Module 2	Module 3	Module 1	Module 2	Module 3
DV_i	0.81	0.89	0.91	0.47	0.81	0.65

Table 8
DV for architectures in Fig. 2(a) and Fig. 2(b).

	Architecture 1	Architecture 2
DV	0.75	0.46

5 Conclusion

This paper aims to propose a new method to determine level of vulnerability in the existing software systems from their source code. Since Mozilla Firefox is open source software and the history of all the vulnerabilities are specified, we used its vulnerability data to show the correlation between coupling types and vulnerabilities. Our experiments showed that there is a significant relationship between different coupling types and vulnerabilities. Then, a mathematical formalism was presented to show the relationship between number of couplings and number of vulnerabilities in a module and the whole software system. Finally, we extracted and retrieved the software architecture using DAGC tool. We converted the extracted architecture into Markov chains.

Then, we predicted and evaluated security of software system using these chains.

5.1 Limitations

We recognize that there are certain limitations to the results and conclusions we have presented in this paper, and we discuss several of them in the following paragraphs.

First, we know the fact that many factors are able to lead to vulnerability of software systems. As the this paper only aimed to emphasize on the issue of software architecture, we showed that coupling types could be used as the most important factor for evaluating security at the architectural level; however, we declare that coupling types, cannot be considered as a complete criterion by itself. Another criterion, which is used at the architectural level, is cohesion. Cohesion can also affect the vulnerabilities. However, the effect of coupling on vulnerability is greater than the effect of cohesion, as coupling is in contact with the external communications of a module while cohesion is in contact with the internal communications of a module.

Second, the formula proposed in this paper was based on vulnerability, which were detected and reported before collecting vulnerability data. The vulnerabilities, which were not detected or publicized, were disregarded in this research. As a result, it was impossible to know the true error rates in the research.

Finally, we have substantiated our findings over a wealth of vulnerability data by analyzing fifty-two releases developed over a period of four years.

5.2 Demonstration of the claim

Using Spearman rank correlation, it showed in section 3 that there is a significant relationship between coupling types and vulnerabilities. After showing this correlation, using regression and the least squares method, a mathematical formalism was presented between coupling types and vulnerabilities in a module. Next, a relation was presented in section 3-2 which predicted the expected number of vulnerabilities in a software system at the architectural level. To do so, we converted the architecture extracted from source code using DAGC tool into Discrete Time Markov Chains. Using the properties of DTMCs and the relation offered in section 3-2, we could predict the expected number of vulnerabilities at the architectural level.

5.3 Future works

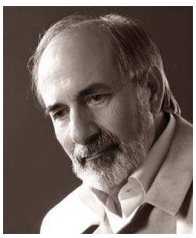
Following direction can be explored to extend and improve this work:

1. We have substantiated our findings over a wealth of vulnerability data by analyzing fifty-two releases developed over a period of four years. It is better to use the vulnerability data of other kinds of software to make the proposed formula more accurate.
2. Coupling and cohesion are the criteria used at the architectural level of software. In fact, the software architecture will be appropriate if its coupling is minimal and its cohesion is maximal. Therefore, if the proposed formula is able to be more useful at the architectural level in detecting vulnerability, it is better to add it to that formula.

References

[1] IV. Krsul, *Software Vulnerability Analysis*, PhD Thesis, Purdue University, West Lafayette, Indiana, USA, (1998).
[2] G. Hoglund, and G. McGraw, *Exploiting Software: How to Break Code*, Boston: Addison-Wesley, (2004).
[3] F. Tsui, and O. Karam, *Essentials of Software Engineering*, 2nd Edition, Jones and Bartlett Publishers, (2010).
[4] J. K. Kearney, R. L. Sedlmeyer, WB. Thompson, MA. Gray, MA. Adler, *Software complexity measurement*, ACM Communications, **29**, 1044-1050 (1986).
[5] R. S. Pressman, *Software Engineering: A Practitioners Approach*, 7th ed. McGraw-Hill, Inc, (2010).
[6] G. Koru, J. Tian, *An empirical comparison and characterization of high defect and high complexity modules*, Journal of Systems and Software, **67**, 153-163 (2003).
[7] M. Janes, W. Scotto, B. Pedrycz, M. Russo, G. Stefanovic, *Identification of defect-prone classes in telecommunication software systems using design metrics*, Journal of Systems and Software, **176**, 3711-3734 (2006).
[8] G. Succi, W. Pedrycz, M. Stefanovic, J. Miller, *Practical assessment of the models for identification of defect-prone classes in object-oriented commercial systems using design metrics*, Journal of Systems and Software, **65**, 1-12 (2003).
[9] K. O. Elish, M. O. Elish, *Predicting defect-prone software modules using support vector machines*, Journal of Systems and Software, **81**, 649-660 (2008).
[10] N. Nagappan, T. Ball, A. Zeller, *Mining metrics to predict component failures*, in: Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, 452-461 (2006).
[11] T. Menzies, J. Greenwald, A. Frank, *Data mining static code attributes to learn defect predictors*, IEEE Transactions on Software Engineering, **33**, 2-13 (2007).
[12] H. Zhang, X. Zhang, M. Gu, *Predicting defective software components from code complexity measures*, in: Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing, Melbourne, Australia, 93-96 (2007).
[13] I. Chowdhury, B. Chan, M. Zulkernine, *Security metrics for source code structures* , in: Proceedings of the Fourth International Workshop on Software Engineering for Secure Systems, Leipzig, Germany, 57-64 (2008).
[14] I. Chowdhury, and M. Zulkernine, *Using Complexity, Coupling, and Cohesion Metrics as Early Indicators of*

Vulnerabilities , Journal of Systems Architecture, **57**, 294-313 (2011).
[15] V. S. Ayanam, *Software Security Vulnerability vs Software Coupling: A Study with Empirical Evidence*, Masters Thesis, Southern Polytechnic State University, Marietta, Georgia, USA, (2009).
[16] L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, second ed., Addison-Wesley, Boston, (2003).
[17] C. J. Munson, *Software Engineering Measurements*, Auerbach Publications, ACRC Press Company, (2003).
[18] O. Bushehrian, *A New Encoding Scheme and a Framework to investigate Genetic Clustering Algorithms*, Journal of Research and Practice in Information Technology, **37**, 127-143 (2005).
[19] B. S. Mitchell, *A Heuristic Search Approach to Solving the Software Clustering Problem*, Ph.D Thesis, Drexel University, Philadelphia, (2002).
[20] Mozilla Vulnerabilities, http://www.mozilla.org/projects/security/knownvulnerabilities.
[21] U. N. Bhat, *Elements of Applied Stochastic Processes*, second ed. John Wiley & Sons, Inc, (1984).
[22] K. S. Trivedi, *Probability and Statistics with Reliability*, Queuing and Computer Science Applications. John Wiley and Sons, (2001).
[23] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences (2nd ed)*, Academic Press New York, (1988).
[24] I. Alexander, *Automatic Vulnerability Detection Using Static Source Code Analysis*, Ph.D thesis, Graduate School of The University of Alabama, (2005).
[25] Bugzilla, http://www.bugzilla.org.
[26] L. Kuang and M. Zulkernine, *An Anomaly Intrusion Detection Method Using the CSIKNN Algorithm*, in Proceedings of the 23rd Annual ACM Symposium on Applied Computing, Fortaleza, Brazil, 921-926 (2008).



Ayaz Isazadeh received a B.Sc. degree in Mathematics from University of Tabriz, Iran, in 1971, an M.S.E. degree in Electrical Engineering and Computer Science from Princeton University, USA, in 1978, and a Ph.D. degree in Computing and Information Science from Queen's University, Canada. Before returning to graduate school in 1992, he worked with Bell Laboratories for 10 years. He is currently a full professor of Computer Science at University of Tabriz.

Islam Elgedawy Middle East Technical University - Northern Cyprus Campus, Guzelyurt, Mersin 10, Turkey. Dr. Elgedawy is an Assistant Professor at the Computer Engineering Department. He received his B.Sc. and M.Sc. degrees in computer science from Alexandria University-Egypt in 1996, and 2000, respectively, and his Ph.D. degree in computer science from RMIT University-Australia in 2007. He has about 17 years of experience spread across industry and academia. His work is mainly focused on the areas of service-oriented computing, cloud computing, software engineering and semantic web. He is an author or coauthor of many technical papers published in well known international journals and conferences, more details could be found in the following web site (http://www.metu.edu.tr/~elgedawy/).

Jaber Karimpour received the B.Sc. degree in computer science and applied mathematics from Tabriz University (Iran) in 1998, the M.Sc. degree, specializing in the computer systems area of applied mathematics, from Tabriz University in 2000. He is currently an Assistance Professor of Computer Science at University of Tabriz, Iran. His research focuses primarily on the formal specification and compositional verification of component-based systems.

Habib Izadkhah received the B.Sc. degree in software engineering from University of PayamNour (Iran) in 2005, the M.Sc. degree in software engineering from Shabestar Islamic Azad University in 2008. He is currently a Ph.D. student of Computer Science at University of Tabriz, Iran. His current researches focus on view based software engineering and reverse engineering.