Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 165029, 11 pages
http://dx.doi.org/10.1155/2013/165029

*Research Article*

# A Quantitative Assessment Approach to COTS Component Security

**Jinfu Chen,[1] Yansheng Lu,[2] Huanhuan Wang,[1] and Chengying Mao[3]**

[1] *School of Computer Science and Telecommunication Engineering, Jiangsu University, Zhenjiang 212013, China*
[2] *School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
[3] *School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang 330013, China*

Correspondence should be addressed to Jinfu Chen; jinfuchen@ujs.edu.cn

Received 28 August 2012; Revised 26 December 2012; Accepted 31 December 2012

Academic Editor: Huaguang Zhang



The vulnerability of software components hinders the development of component technology. An effective assessment approach to component security level can promote the development of component technology. Thus, the current paper proposes a quantitative assessment approach to COTS (commercial-off-the-shelf) component security. The steps of interface fault injection and the assessment framework are given based on the internal factors of the tested component. The quantitative assessment algorithm and formula of component security level are also presented. The experiment results show that the approach not only can detect component security vulnerabilities effectively but also quantitatively assess the component security level. The score of component security can be accurately calculated, which represents the security level of the tested component.

## 1. Introduction

In the area of software engineering, component-based software engineering (CBSE) has currently become a research point. The various development technologies of components further improve the development efficiency and performance of components, but the problems on the reliability and security of components have not yet been resolved. Current component testing approaches have focused on component functionality testing, which, to some extent, guarantees the correctness and integrity of component functionality. However, component security testing is rarely researched as a special subject. Component security testing mainly involved in testing the vulnerabilities of component which can be exploited to crack software component. There are some general component vulnerabilities such as memory leakage, stack overflow, access beyond boundary, and access protected page. In addition, the special difficulties for the security testing of COTS component are as follows. Firstly, the source codes are unavailable. Secondly, The general security pattern is difficult to attain by analyzing the general classification model of software vulnerability. Finally, The COTS component is very independent, which results in some problems such as gaining the effective component information and running the independent component. On the other hand, the assessment of component security has not been solved. Research on the quantitative assessment of component security level is rather rare. In the field of software testing, quantitatively assessing the security level of the component system not only guarantees and enhances the system's reliability and security but also conducts a quantitative assessment level to software developers and facilitates users in selecting software products based on its security to reduce the risk of attack.

Nevertheless, component security vulnerabilities are usually studied, but the quantitative assessment of component security is not common. Most scholars only research on the theoretical description of security assessment, but practical implementation methods are not presented. A basic assessment approach was given in the literature [1]. Based on the knowledge of information security, the main approach is dividing the component security vulnerability into several ranks [2]. The security level of components is assessed using the software testing technology. The relevant experiments were conducted but were not studied in depth.

Enough internal information should be collected before assessing the security level for the software component. But,

it is usually very difficult to obtain the internal information for COTS (commercial-off-the-shelf) components, because its source codes are unavailable. In order to obtain more information about COTS component, it is necessary that some data are inputted the tested component to drive the component running. When the tested component is running, some internal information including some security information can be collected by monitoring the running state of the tested component. The data-driven method can obtain internal security information for COTS component. In the current research, we propose a feasible approach to assess the component security level quantitatively through using data-driven method and learning from relevant statistical and probabilistic methods. The exceptional component information monitored by the dynamic monitoring system can be obtained by the testing system of component vulnerability [3]. Relevant security factors are assigned to appropriate security exceptions according to the classification of the vulnerability in the Common Vulnerabilities and Exposures (CVE) vulnerability library. Each of the exceptional vulnerability factor values can be obtained from the triggering causes of software security vulnerabilities, which is a harmful degree combined with the characteristics of the component security vulnerabilities. Finally, six commercial components, in which security vulnerabilities exist, are collected for experimental analysis in the CVE Web site and are assessed by the proposed approach. The number of each interface and methods used for each interface can be obtained by dynamically executing the tested components. The executed probabilities of each interface and methods used for each interface can be determined by statistical method. At the same time, fault injection operators and the logical prediction rules of exceptional database (PRED) are proposed according to the component security exceptions commonly used. This approach not only can assess the component security level but also effectively assess the security level of each interface, thus providing better assessment capabilities.

## 2. Related Work

At present, commercial-off-the-shelf (COTS) components, such as military, mechanization, financial components, and so on, are used by more and more commercial software including some crucial security software. COTS security takes an important part in component systems because the security of the whole software system is determined by it. Therefore, assessing the security level of the component is used to test the security of the entire software system, which has become an essential part in component testing. To the best of our knowledge, research on assessing the software component security mainly focuses on component security requirements and assessment approaches, vulnerability assessment, and quantitative risk assessment.

Han and Zheng [4] and Jabeen and Jaffar-Ur Rehman [5] studied the security requirements and security assessment methods of components but did not give the specific relevant assessment methods. Md. Khan et al. examined the security testing description and component assessment [6–8]. Models of the user data security, component security framework, and security assessment model of software were proposed. Khan and Han developed an assessment scheme for the security properties of software components. The assessment scheme provided a numeric score that indicates the relative strength of the security properties of the component [8]. However, the approach consists of two prerequisites: (1) a security requirement specification of the candidate component and (2) a component-specific security rating. The approach requires the source codes of candidate component to be available. The security property descriptions of the components based on the security assessment standards of information technology were proposed by Khan et al. [9]. The main attributes of resource allocation, user data protection, and communications were included. The model was based on the active interface, which could provide the basis for reasoning and assessing a component's security to meet certain security requirements of a particular application. Common criteria (CC) provide a schema for evaluating information system and enumerate the specific security requirements for such systems [10]. The functional requirements in CC describe the security behavior or functions expected of an information system to counter threats, but not all the requirements in CC may be applicable directly to software components because of the distributed nature of software components and their compositional complexity.

Alhazmi and Malaiya proposed a time-based model for the total vulnerabilities and an alternative model which is analogous to the software reliability growth models [11]. However, this model cannot quantitatively assess the security level of software. Zhang et al. presented a vulnerability risk assessment method based on the attack tree model and Bayesian network model [12]. The analysis method can assess the risk probability of the software system but not its security level. The security risk assessment was introduced to the classical MDA (Model-Driven Architecture) framework [13]. The extension offers early assessment and early validation of security requirements. However, the improved framework still has not provided the quantitative mechanism of the software security. A systematic methodology was proposed for some software vulnerability assessment and security function verification [14]. A scalable and adaptable automatic test system was implemented to test hundreds of software releases, but the security assessment of software releases was not implemented. The state of the architecture-based approach to reliability assessment of component-based software was presented [15]. The common requirements of the architecture-based models were identified, and the classification was proposed. However, the approach only gave a framework for the reliability assessment of component-based software.

Idongesit proposed a quantitative risk assessment model based on annualized loss expectancy and the exposure factor, which is the percentage of asset loss due to the potential threat [3]. The model can assess the potential threats to a development effort and rank these threats based on the risk metric calculation. The Common Vulnerability Scoring System (CVSS) [16] provides an open framework for communicating the characteristics and effects of IT vulnerabilities. Its

quantitative model ensures repeatable accurate measurement while enabling users to see the underlying vulnerability characteristics used to generate the scores. However, CVSS measures software vulnerability based on the external factors of software system. Occasionally, the result cannot precisely represent the software vulnerability.

In summary, we propose a new quantitative assessment of component security (QACS) approach to represent more accurately the security level of the COTS component based on the internal factors of software component. The internal factors of software component are some important information inside the component instead of the outside enviornment. There are some main internal factors such as the component objects, component interfaces, component methods, and component attributes. These internal factors reflect more accurate security features for the tested component.

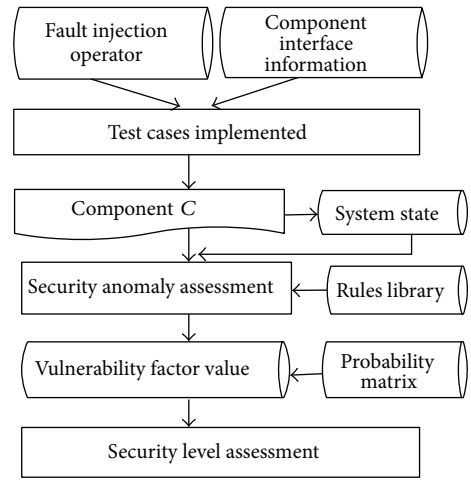


Figure 1: A quantitative assessment framework.

## 3. A Quantitative Assessment Framework for Component Security

With the further development of component-based software engineering, COTS components have become frequently used in software systems. The source codes of the COTS component are unavailable and highly independent. The traditional testing approach of interface mutation is suitable for system integration testing. That is, mutation operators are only effective at the interface between the module and the subsystem. Therefore, the traditional method is unsuitable for testing the COTS component system. Interface mutation testing, which is suitable for the COTS component, can inject faults at the component interface level by extending the mutation operators in the program mutation and increasing the mutation operators at the component interface level. The defects of components can be identified through observation and analysis of the testing result of the system. Figure 1 shows the quantitative assessment framework of the component security level.

The interface fault injection operators of the COTS component include irregular input parameter operators, mutation operators based on program language, mutation operators based on interface specifications [17], and mutation operators based on interface definition language (IDL) [18]. The fault injection operators, which are commonly used, are as follows: boundary value parameter, parameter beyond the length, parameter attribute replaced, parameter swap, parameter change, parameter assignment, parameter set null, and so on.

The test cases of fault injection can be generated through the selection of a suitable fault injection operator based on parameter types and number of component methods in the interface information. The test scripts can be generated according to the test cases. The component can be run with the test scripts. The running state can be monitored through the dynamic monitoring mechanism. Component vulnerability of the interface methods can be determined by the running state of the component, security prediction mechanism, prediction rules, and vulnerability factor in various methods. The vulnerable score can be obtained under certain running environments in terms of the vulnerability assessment method of component $C$. The vulnerability score, which can assess the security of component $C$, is called the security level of component.

### 3.1. Fault Injection Operator of the Component Vulnerability

*Definition 1.* Fault injection operator $\prod$ is the rule that generates a number of fault injection test cases, namely, $\prod \rightarrow \{\Psi_1, \Psi_2, \ldots, \Psi_n\}$. $\Psi_i$ denotes a fault injection test case. A class of fault injection test cases can be implemented by one fault injection operator.

*Definition 2.* Interface fault injection operation $O$ is a binary relation $O(\prod, P)$, where $P$ is a class of interface parameters, $O(\prod, P) = \{(\Psi_i, p_i) \in \{\Psi_1, \Psi_2, \ldots, \Psi_n\} \times P$. Fault injection test case, which is implemented by fault injection operator $\prod$, acts on the interface parameter $p_i$, where $O(\prod, P)$ can be short for $O$ only if it does not cause confusion.

Whether the generated test cases of fault injection are representative is the key to designing the fault injection operator, that is, whether the generated test cases of fault injection can trigger the component's security vulnerability as greatly as possible. Two errors, namely, environmental errors and interface parameter errors, can trigger component security vulnerabilities in a component system. The first error, which is discussed in the previous paper [1], does not repeat it again. The focus of the current paper is the interface parameter fault injection. The errors in the interface parameter include three types: IDL description errors, boundary value errors, and irregular input errors. In detail, they are the method errors defined by the IDL description, all boundary value, and irregular input value errors, respectively. Based on the cause of triggering software security vulnerabilities in practical experience, 18 fault injection operators of component vulnerability are specifically designed by comprehensively analyzing the interface mutation operators and the operators of parameter fault injection. These operators are PAS

Table 1: Exceptional predication rules and vulnerability factors.

| ID | Predication rules | Description | VC |
|---|---|---|---|
| 01 | General exception | The general exception code is detected, or throw an exception | 0.5 |
| 02 | Access out of range | Running a thread attempts to read or write the memory address, but has no corresponding access authority | 1 |
| 03 | Matrix visited beyond scope | A thread tries to access an array element beyond scope | 1 |
| 04 | Access the memory page not exiting | The file system returns a read error resulting in page fault which can-not meet the requirements | 0.8 |
| 05 | Visit the protected page | A thread tries to access memory page that has the attribute of PAGE_GUARD | 1 |
| 06 | Stack of thread is overflow | A thread runs out of all stack space assigned to it | 1 |
| 07 | Execute illegal instruction | A thread running an instruction that is not allowed in the current thread or executing an invalid instruction | 1 |
| 08 | Divisor is 0 | A thread tries to divide an integer or a float by 0 | 0.8 |
| 09 | Operation out of range | An operation result exceeds the specified range values | 0.8 |
| 10 | Floating-point stack overflow | Stack overflow or underflow because of floating-point operations | 1 |
| 11 | Buffer overflow | Written contents beyond the buffer length, resulting in written data covering the original return address and destroying the program stack | 1 |
| 12 | Memory leakage | Monitoring the input and output values of malloc(), free(), realloc() functions and making a statistical analysis to achieve a memory leakage check | 1 |
| 13 | Format string exceptions | Format or parameter mismatch when using printf() function, such as "%*n*%*n*%*s*%*d*" | 1 |

(Parameter Attribute Swap), PSN (Parameter Set Null), IPO (Insert Parameter Operator), PFB (Parameter Flip Bit), PRI (Parameter Reverse-probability Input), IIV (integer Irregular Value), FIV (Float Irregular Value), CIV (Char Irregular Value), BIV (Boolean Irregular Value), RSV (Random String Value), LSV (Long String Value), FSV (Format String Value), DSV (Directory String Value), USV (URL and string value of file directory), CSV (System Command String Value), SSV (SQL string injection), and CSS (Cascading Style Sheet). The new security vulnerabilities found in components can also be added.

In addition, the reverse distribution function of the parameter input domain is generated using the input operator PRI of the parameters of reverse probability distribution. The parameters of reverse probability distribution are then generated. The experimental result shows that component security exception can be effectively triggered through the irregular value of input domain. The input distribution function, which is usually difficult to obtain, is a probability density function. The one dimensional and multidimensional functions are the two commonly used kinds. $Q$ is assumed the input distribution function, of which the value is in the interval $[0, 1]$, and the value 0 indicates the probability of selection input field is 0. The algorithm of reverse probability distribution $Q'$ is given in the literature [19].

A number of models have been designed to facilitate the automatic management of the fault injection operator, such as the automatic learning model of the fault injection operator, adaptive model, topic model, intelligent vulnerable trees, among others. They can automatically increase and manage the operator libraries of fault injection.

### 3.2. Exceptional Predication Rules and Vulnerability Factors

*Definition 3.* The component exceptional predication rule is a set of all the possible exceptions that may arise under the exceptional circumstances.

*Definition 4.* The vulnerability factor of the component represents the degree of the component security vulnerabilities resulting from exceptions. The maximum value is 1, and the minimum value is 0. Table 1 shows the component exceptional predication rules and relative vulnerability factors used to assess the component security level. Each of the factor value of exceptional security vulnerability, which can objectively describe the component security level, is obtained from the triggering causes of software security vulnerabilities, which is a harmful degree combined with the characteristics of component security vulnerabilities based on experience software engineering.

## 4. Component Security Assessment Algorithm

Suppose $C$ is a component, and $I$ is a set of interface in $C$. Let $p_i$ represent the execution probability of interface $i$, and $M_i$ is a method properties set of interface $i$. Meanwhile, $p_{im}$ is the executable probability in method $m$ of interface $i$. Suppose $|I| = N$, $|M_i| = S$. $x$ is the input parameter of the method attribute in $C$. $T$ is vulnerable error test cases set and $T = \{t_1, t_2, t_3, \ldots, t_n\}$. $\Delta$ is all the input set of component $C$. $Q'$ is the reverse probability distribution of $\Delta$. PRED is the logical prediction rules of exceptional database. VC is the vulnerability factor of exceptional function. Before the

vulnerability assessment method of component is given, the following definitions should be introduced first.

*Definition 5.* Let $P$ represent an $N \times S$ probability matrix that can be described as $P = (p_{ij})_{n\times s}$, row $i$ represents the interface number of component $C$, and column $j$ represents the method number of interface $i$. The element $p_{ij}$ states the executable probability for function $j$ at interface $i$ of component $C$, where $p_{ij} = p_i \times p_j$.

*Definition 6.* Interface method coverage (MC), where MC = (the number of methods executed at least once/the total number methods $S$) $\times$ 100%, is used to measure the sufficient degree of the interface method during component testing.

*Definition 7.* Interface coverage (IC), where IC = (the number of interfaces executed at least once/the total number interface $N$) $\times$ 100%, is used to measure the sufficient degree of the interface test during component testing.

*Definition 8.* Component test adequacy CA can be measured by a two-tuple (i.e., MC and IC). Generally, the value is determined by the total score of MC and IC, that is, CA = $[\sum_{i=1,N}(\mathrm{MC}_i)]/N$+IC. A greater value of CA indicates a more comprehensive test; the relationship between MC and IC is parallel.

Suppose Sum $M$ represents the number of test methods, cnt represents the number of vulnerability methods in component $C$, and $\mathrm{VC}_i$ represents the vulnerability factors of exceptional method. The value of vulnerability factor $\mathrm{VC}_i$ is assigned to 0 in case the interface method has no security exceptions. The execution probability of the vulnerability method $i$ is labeled $P_i$. The component vulnerability assessment formula is given as follows:

$$\mathrm{Vul}_c = 1 - \prod_{i=1}^{\mathrm{cnt}} \left(1 - P_i \times \mathrm{VC}_i\right). \quad (1)$$

The value $1 - \mathrm{Vul}_c$, the security level of the component $C$, can be derived from the Formula (1). The vulnerability assessment formula of the component has the following properties.

*Property 9.* $(0 \le \mathrm{Vul}_c \le 1)$.

*Proof.* For any vulnerability method, $1 \ge P_i \ge 0$ and $1 \ge \mathrm{VC}_i \ge 0$ can be determined according to the definition of VC. Therefore, $1 \ge P_i \times \mathrm{VC}_i \ge 0$, $1 \ge 1 - P_i \times \mathrm{VC}_i \ge 0$ and $0 \le \prod_{i=1}^{\mathrm{cnt}}(1 - P_i \times \mathrm{VC}_i) \le 1$ are derived, that is, $0 \le \mathrm{Vul}_c \le 1$. □

Property 9 shows that the value of any component's vulnerability index is between 0 and 1, with a larger value indicating a higher vulnerability index.

*Property 10.* $(\mathrm{Vul}_c \ge \max(P_i \times \mathrm{VC}_i) \ge 0)$.

*Proof.* For any vulnerability method, $1 \ge P_i \ge 0$ and $1 \ge \mathrm{VC}_i \ge 0$ can be determined according to the definition of VC. Therefore, $\max(P_i \times \mathrm{VC}_i) \ge 0$. In addition, $1 \ge 1 - P_i \times \mathrm{VC}_i \ge 0$ and $1 - \mathrm{Vul}_c = \prod_{i=1}^{\mathrm{cnt}}(1 - P_i \times \mathrm{VC}_i)$ can be derived from Formula (1). $\prod_{i=1}^{\mathrm{cnt}}(1 - P_i \times \mathrm{VC}_i) = (1 - P_1 \times \mathrm{VC}_1) \times (1 - P_2 \times \mathrm{VC}_2) \times \cdots \times (1 - \max(P_i \times \mathrm{VC}_i)) \times \cdots \times (1 - P_{\mathrm{cnt}} \times \mathrm{VC}_{\mathrm{cnt}})$; therefore, $\prod_{i=1}^{\mathrm{cnt}}(1 - P_i \times \mathrm{VC}_i) \le (1 - \max(P_i \times \mathrm{VC}_i))$ and $1 - \mathrm{Vul}_c \le (1 - \max(P_i \times \mathrm{VC}_i))$ are derived, that is, $\mathrm{Vul}_c \ge \max(P_i \times \mathrm{VC}_i)$. Thus, $\mathrm{Vul}_c \ge \max(P_i \times \mathrm{VC}_i) \ge 0$ can proved correct. □

Property 10 shows that the vulnerability index of any one component is not less than the peak value of the methods' vulnerability indexes of the component. All situations of the vulnerability methods of component are synthetically considered by the index of the vulnerability of component.

Based on the component vulnerable assessment method based on the analysis of the interface, the algorithm of the component's security assessment level is given as follows.

The input probability matrix $P$ in Algorithm 1 is not only derived from the component requirement specifications or equal probability but is also statistically calculated after the component is practically executed. The vulnerability factor $\mathrm{VC}_{ij}$ is assigned to 0 given that the component has no security exceptional interface method. We usually set the value of $\delta$ and $\theta$ from 0 to 1 according to the complexity of tested component and the testing requirement. If the number of interfaces and methods of tested component is no more than 200, we usually let the value of $\delta$ and $\theta$ to be 1 to ensure a complete testing of COTS. In addition, the test cases of interface parameters are generated using the TGSM [20, 21] algorithm. First, the test cases of the interface parameter fault-injection, expressed in matrix form, are generated in terms of fault injection operator. The $K$-factor coverage solution matrix is generated by the TGSM algorithm, which is based on the parameter set of the matrix form. The fault injection test cases are composed of row data of the solution matrix. Test cases generated by the TGSM algorithm not only greatly reduce the scale of the test cases but also have certain coverage. The algorithm can trigger most of the security exceptions with fewer test cases.

Cnt is the number of vulnerability methods of the component $C$ in the algorithm results. $\mathrm{Vul}_{ij}$ is the vulnerability ratio of the method $j$ at the interface $i$ of the component $C$. $\mathrm{Vul}_c$ represents the vulnerability level of the component $C$. $1 - \mathrm{Vul}_{ij}$, the security level of the component $C$, represents the security of the method $j$ in the interface $i$ of component $C$.

## 5. Experiment Analysis

To validate the effectiveness of the proposed approach, the approach was implemented in our component security testing system (CSTS) platform [22]. The component security testing and assessing system based on interface fault injection (CSTASboIFT) was implemented using C # language in the Microsoft. NET platform. It is used as a subsystem of the CSTS platform. Figure 2 shows the function flow chart of CSTASboIFT.

In the experiment, six commercial components (Table 2), in which security vulnerabilities exist, were collected for experimental analysis in the CVE Web site. Several group

```
Input: Interface information XML file; Fault injection operator; Prediction rules PRED;
Probability matrix P; MC's threshold and IC's threshold value δ and θ.
Output: The security level of the whole component
01 {
02   Read XML file;
03   MC = 0, IC = 0, i = 0, cnt = 0; // cnt is the number of component vulnerability methods
04   While IC < δ do        // For each interface i in I
05   {
06   i = i + 1, j = 0;
07   While MC < θ do        // For each method m in M_i
08   {
09   j = j + 1;    // j is the number of the testing methods
10   Generate method parameter values set according to fault injection operators;
11   Call testing cases generating algorithm TGSM(); //call the generation algorithm of the
       minimum K factors combined cover test case based on solution matrix
12   Running C and M_ij;
13   If (the output after running C and M_ij satisfies PRED)
14      {
15           increment cnt;
16           Vul_ij = VC_ij× P_ij; // the vulnerability level of the method j in the interface i
17      }
18   MC = j/S;
19   }
20   IC = i/N;
21   }
22   Vul_c = 1 − ∏_{i=1}^{cnt}(1 − P_i × VC_i);
23 (Output) 1 − Vul_c;
24 }
```

Algorithm 1: QACS (quantitative assessment of component security) algorithm.

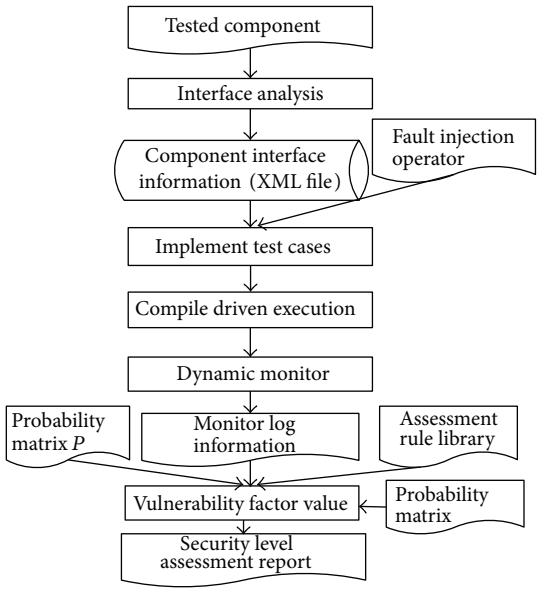


Figure 2: The function flow chart of CSTASboIFT.

experiments were conducted for these components. A component composed of six interfaces and each interface consisting of eight methods were assumed so that the component could be regarded as a 6 × 8 matrix. The first group of experiments tested the condition of $P_{ij}$ assigned to a fixed matrix and $VC_{ij}$ divided into interval descending. As $VC_{ij}$ changed, the $1 - Vul_c$ was affected. That is, only the level of vulnerability factors affected the component security level. The second group of experiments tested the condition of $VC_{ij}$ assigned to a fixed matrix and $P_{ij}$ divided into interval descending. As $P_{ij}$ changed, $1 - Vul_c$ was affected. That is, only the executed probability affected the component security level. The last group of experiments considered the effect on $1 - Vul_c$, as both vulnerability factor and executed probability simultaneously changed.

*5.1. Experiment One.* The vulnerability factor value is divided into intervals based on the knowledge of security vulnerability provided by the CVE Web site. $P_{ij}$ represents the executable probability of the method $j$ at the interface $i$, $VC_{ij}$ represents the vulnerability factors of the method $j$ at the interface $i$, and $Vul_c$ represents the vulnerability level of component $C$. $1 - Vul_c$ is the security level of component $C$.

The first experiment assumes that $P_{ij}$ is a 6 × 8 fixed matrix, which is made up of random values. For every interval, $P_{ij}$ is the same matrix. $VC_{ij}$ is divided into 10 intervals: [0.00, 0.09], [0.10, 0.19], [0.20, 0.29], [0.30, 0.39], [0.40, 0.49], [0.50, 0.59], [0.60, 0.69], [0.70, 0.79], [0.80, 0.89], and [0.90, 1.00]. The tests were done about ten times with different random values of $P_{ij}$ and $VC_{ij}$, respectively, in the experiment, and then the value $1 - Vul_c$ was got by calculating

Table 2: The six typical tested components.

| Component ID | Component name | Component information |
|---|---|---|
| 01 | ThunderAgent_005.dll | An ActiveX control in Thunder 5. Component version is 1.0.0.11. Thunder version is 5.5.2.252. |
| 02 | AcroPDF.dll | An AcroPDF ActiveX control in Adobe Reader and Acrobat. Component version is 7.0.8.0. |
| 03 | GLItemCom.dll | An ActiveX control in OURGAME WORLD game software. Component version is 2.7.0.8. |
| 04 | pdg2.dll | An ActiveX control in SUPERSTART READER software. Component version is 3.9.0.0. |
| 05 | GomWeb3.dll | An ActiveX control in GOM Player. GOM Player is a media player widely used in Korea. |
| 06 | Vuln.dll | A third-party tested component. Component version is 1.0.1.2. |

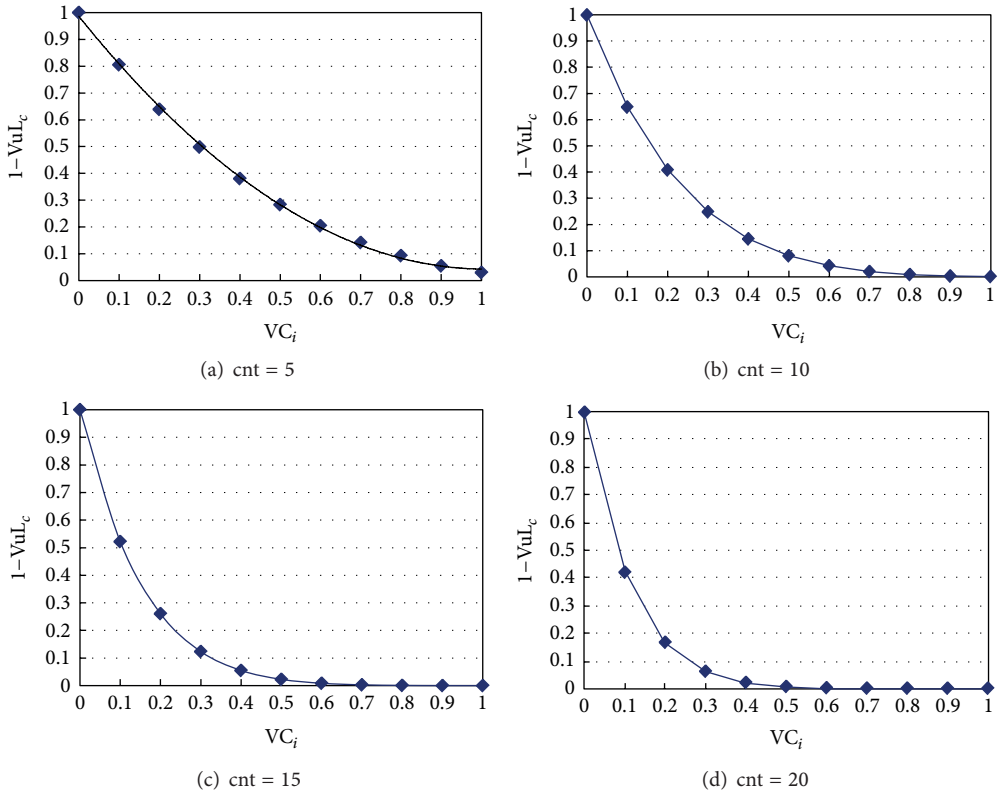


Figure 3: The comparison for different $VC_i$ and cnt.

the average result. Under the condition that $P_{ij}$ is regarded as a $6 \times 8$ matrix at every interval of $VC_{ij}$, $VC_{ij}$ affects the value $1 - Vul_c$.

When $Vul_{ij}$ is equal to $P_{ij} \times VC_{ij}$, as $P_{ij}$ is a fixed value, the values of $Vul_c$ and $Vul_{ij}$ are proportional to the value of $VC_{ij}$. The value of $1 - Vul_c$ is reversely proportional to $VC_{ij}$. A larger $VC_{ij}$ value indicates a lower security level. As the vulnerability number increases, the security level decreases slowly. The results are shown in Figure 3. The testing results show that the proposed assessment Formula (1) is effective.

*5.2. Experiment Two.* $VC_{ij}$ is assumed a $6 \times 8$ fixed matrix made up of random values in the second experiment. Then, $P_{ij}$ is divided into 10 intervals: $[0.00, 0.09]$, $[0.10, 0.19]$, $[0.20, 0.29]$, $[0.30, 0.39]$, $[0.40, 0.49]$, $[0.50, 0.59]$, $[0.60, 0.69]$, $[0.70, 0.79]$, $[0.80, 0.89]$, and $[0.90, 1.00]$. The tests were done about ten times with different random values of $P_{ij}$ and $VC_{ij}$, respectively, in the experiment, and then the value $1-Vul_c$ was got by calculating the average result. Under the condition that $VC_{ij}$ is regarded as a $6 \times 8$ matrix at every interval of $P_{ij}$, $P_{ij}$ affects the value of $1 - Vul_c$.

When $VuL_{ij}$ is equal to $P_{ij} \times VC_{ij}$, as the value of $VC_{ij}$ is fixed, the values of $Vul_c$ and $VuL_{ij}$ are proportional to the value of $P_{ij}$. The value of $1 - Vul_c$ is reversely proportional to $VC_{ij}$. A larger $P_{ij}$ value indicates a lower security level. That is, the component is more insecure. As the number of cnt increases, the decline rate of the security level is initially fast and then becomes slow. The results, as presented in Figure 4, show that the proposed assessment Formula (1) is effective.

*5.3. Experiment Three.* Both parameters $P_i$ and $VC_i$ are assumed to influence the value of $1 - Vul_c$ in Experiment three. The two parameters are randomly divided into 10

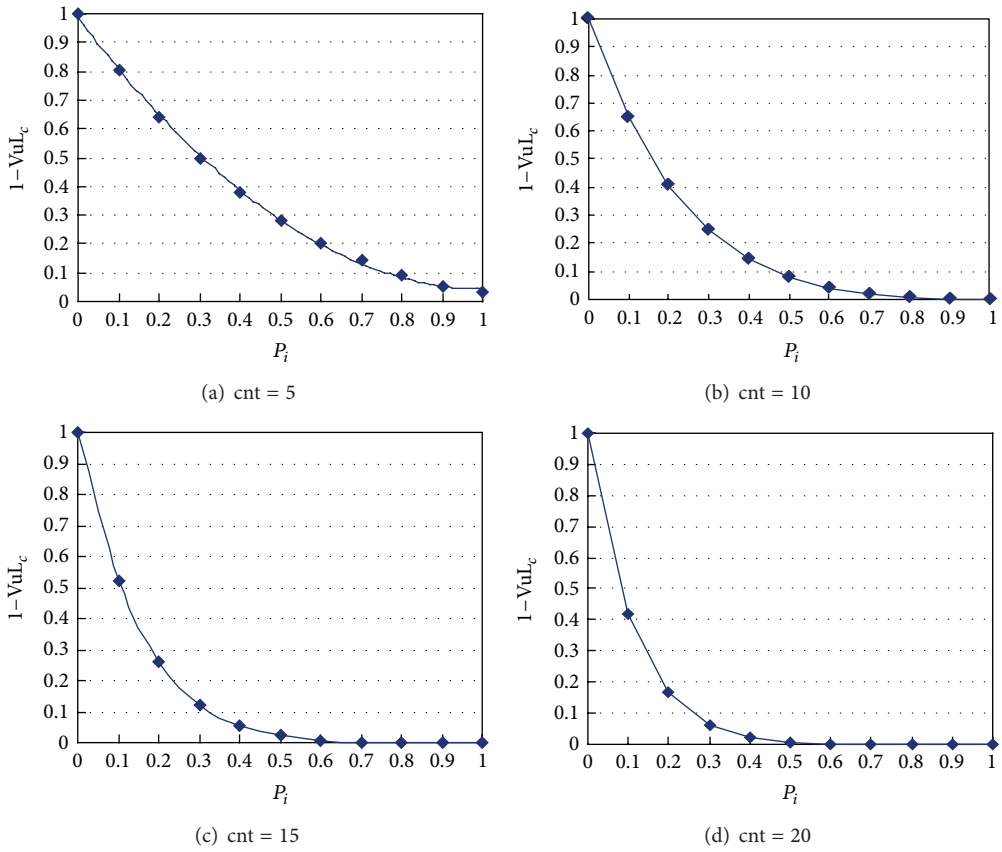


Figure 4: The comparison for different $P_i$ and cnt.

intervals. The effect on the value of $1 - \mathrm{VuL}_c$ when the two parameters changed simultaneously was observed. The results show that when $P_i$ is proportional to $\mathrm{VC}_i$, the value of $1 - \mathrm{Vul}_c$ changes significantly as shown in (a) and (b) of Figure 5. When $P_i$ is inversely proportional to $\mathrm{VC}_i$, the value of $1 - \mathrm{Vul}_c$ changes indistinctly as shown in (c) and (d) of Figure 5. As the vulnerability number cnt increases, the security level decreases. Results are shown in Figure 5. The values of $P_i$ and $\mathrm{VC}_i$ are based on the main $y$-axis values (the left axis), and the value of the $1 - \mathrm{Vul}_c$ is based on the secondary $y$-axis values (the right axis). The testing results show that the proposed assessment Formula (1) is suitable for assessing the COTS component.

*5.4. Experiment Four.* The components were assessed in Table 3 based on Algorithm 1. The value of IC and MC is set to 1. $P_i$ can be determined by executing the tested component 100 times or more.

During the system testing process, the test cases of fault injection can be implemented by testing the component information and fault injection operators. The testing state can be recorded through the dynamic monitoring mechanism after running the tested component. The vulnerability type of the component method and vulnerability factor can be derived from comparing the testing state with the component information record in PRED. The execution probability $P_i$ of each method in the component can be determined by running the tested component 100 times, respectively. Then, the component security level can be determined using Formula (1). The testing results are shown in Table 3. Because different components have different method parameters which have different parameter types such as integer, string, and boolean, we select suitable fault injection factors to test the component security based on the above presented 18 fault injection operators of component in Table 3.

A higher level of security indicates better relative security of the component. The component is secure if and only if the component security level is 1. Otherwise, it has security vulnerability. In contrast to the test results in Table 3, the level of security is closer to the vulnerability information published in the CVE Web site. The values of execution probability and vulnerability factors have been proved to affect the component security. The results agree with the previous experimental results, further confirming the feasibility and effectiveness of this approach.

*5.5. Experimental Comparison.* Table 4 shows the comparison with other assessment approaches. The related assessment approaches have various merits as shown in the analysis of Table 4. Our QACS approach mainly focuses on quantitatively assessing the security level of the COTS component. The source codes of the COTS component are

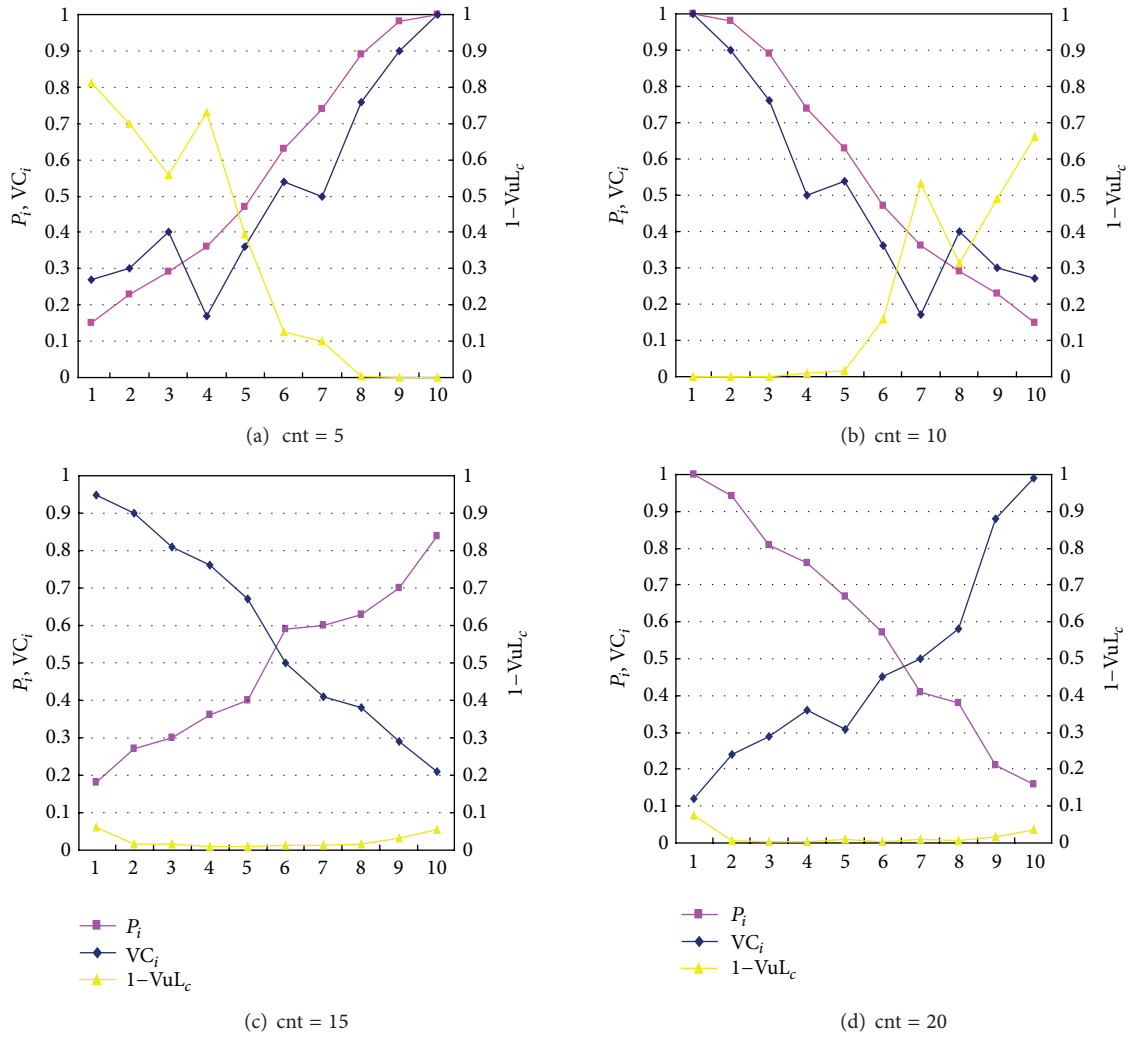


Figure 5: The comparison for different $P_i$, $VC_i$, and cnt.

generally unknown. The internal factors of the component are considered in QACS. The QACS approach is implemented in the CSTASboIFT that is integrated into our CSTS [22] system developed to test the security of the COTS component.

Aside from the above-mentioned approaches, CVSS is a widely used quantitative assessment approach for software component. Khan and Han developed an assessment scheme for the security properties of software components. The assessment scheme provided a numeric score that indicates the relative strength of the security properties of the component [8]. Results of the comparison with the CVSS approach and Khaled approach are shown in Table 5. In addition, the security level of six tested components is also listed according to the reported empirical data of security Web sites in Table 5. The initial CVSS Base Score is calculated by the CVSS formula [16].

Analyzing Table 5, the QACS approach is found to be close to the empirical data reported on some security Web sites regarding the security of the tested COTS components. The security score of the QACS approach is more accurate than that of others because the internal factors of the tested component are considered. The assessment process of the QACS approach is objective, without human intervention. Nevertheless, the CVSS approach is mainly focused on assessing the vulnerability of the general software component. If an assessor is not familiar with the assessed software, the vulnerability score becomes inaccurate because of the assessor's subjectivity in the assessment process. On the other hand, the Khaled approach is an approximate assessment approach based on the assumed component environment. In addition, presently there are no more effective assessment approaches for the security level of software component according to our studies.

## 6. Conclusion

In component-based software development and maintenance activities, the assessment of the component reliability and security level is essential in the third-party component. Research on an effective quantitative assessment method for component security level is valuable in the CBSE. The

Table 3: Testing results for six tested components.

| ID | Fault injection factor | The number of vulnerabilities and its $P_i$ | Test result | Security level |
|---|---|---|---|---|
| 01 | IIV | 2 (0.13, 0.13) | ThunderAgent_005.dll has two methods calling (GetInfoStruct(), GetTaskInfoStruc()) neglected the input parameter exception and generated security vulnerability. Then they resulted in program breakdown | 0.7569 |
| 02 | CIV, LSV, RSV | 5 (0.1, 0.2, 0.1, 0.15, 0.2) | AcroPDF.dll did not correctly deal with the string parameters sent to function setPageMode(), LoadFile(), setLayoutMode() and setNamedDest() and the attribute src. It triggered memory destroyed and resulted in security vulnerability for executing any instruction. | 0.4406 |
| 03 | LSV, RSV, FSV | 1 (0.64) | The function SetInfo() of GLItemCom.DLL ActiveX over trusted user input and did not check parameter's length. It resulted in the point the virtual function covered. | 0.36 |
| 04 | LSV, RSV, FSV | 1 (0.1) | Pdg2.dll did not deal with the sent parameter to Register (), if user sends the string over 256 bytes, it can trigger stack overflow and execute any code. | 0.9 |
| 05 | LSV, RSV, USV | 1 (0.15) | GomManager object did not collectively deal with the first parameter of OpenURL() in IGomManager interface. If transmitting the long parameters over 500 bytes, it can trigger stack overflow to result in visiting the memory exception. | 0.85 |
| 06 | PSN, CIV, PIV, LSV, USV | 5 (0.15, 0.15, 0.15, 0.15, 0.15) | 5 methods of component existed security exception. The test case parameters implemented by input fault injection operator would result in buffer overflow, visiting beyond the scope, and memory leakage. | 0.4437 |

Table 4: The comparison with related assessment approaches.

| Assessment approach | Assessment object | Assessment aspect | Is the internal factor considered? | Qualitative or quantitative? | Is there supporting tool? |
|---|---|---|---|---|---|
| Khan and Han [8] | Software component | Security | No | Quantitative | No |
| Alhazmi and Malaiya [11] | Operating system | Vulnerability | Yes | Quantitative | No |
| Zhang et al. [12] | Software system | Risk | No | Quantitative; qualitative | No |
| Goševa-Popstojanova and Trivedi [15] | Software system | Reliability | No | Quantitative | No |
| Mkpong-Ruffin [3] | Software | Risk | No | Quantitative | Yes |
| CVSS [16] | Software component | Vulnerability | No | Quantitative | Yes |
| QACS | COTS component | Security | Yes | Quantitative | Yes |

Table 5: The comparison with CVSS and Khaled approaches.

| Component ID | Initial CVSS base score | Converted security score of CVSS | Security level of QACS | The reported empirical data (security level) |
|---|---|---|---|---|
| 01 | 5.8 | 0.42 | 0.7569 | $B^+$ |
| 02 | 1.9 | 0.81 | 0.4406 | $C^-$ |
| 03 | 6 | 0.4 | 0.36 | $D^+$ |
| 04 | 7 | 0.3 | 0.9 | A |
| 05 | 3.6 | 0.64 | 0.85 | $A^-$ |
| 06 | 1.9 | 0.81 | 0.4437 | $C^-$ |

current paper proposes the QACS approach for quantitatively assessing component security, which is based on the interface fault injection of the component. The fault injection operators of the component vulnerability and PRED are defined and discussed. The assessment framework and algorithm for component security are proposed as well. The testing effect proves excellent in practical application. QACS has the following main advantages: (1) the security exceptional assessment rules and vulnerability factors of the component are given, providing the basis for security assessment; (2) QACS can assess the security level of the third-party component, and the score of component security can be accurately calculated; (3) QACS has better operability.

## Acknowledgments

We would like to thank Professor T. Y. Chen for his constructive suggestions and valuable discussion in this paper writing. In addition, this work was supported partly by the National Natural Science Foundation of China (NSFC) under Grants no. 61202110 and no. 61063013, Natural Science Foundation of Jiangsu Province under Grant no. BK2012284, and the Research Fund for the Doctoral Program of Higher Education of China under Grant no. 20103227120005.

## References

[1] J. Chen, Y. Lu, and X. D. Xie, "Component security testing approach by using interface fault injection," *Journal of Chinese Computer Systems*, vol. 31, no. 6, pp. 1090–1096, 2010.

[2] S. Gudder and R. Greechie, "Uniqueness and order in sequential effect algebras," *International Journal of Theoretical Physics*, vol. 44, no. 7, pp. 755–770, 2005.

[3] I. O. Mkpong-Ruffin, *Quantitative risk assessment model for software security in the design phase of software development, [Ph.D. thesis]*, Auburn University, Auburn, Ala, USA, 2009.

[4] J. Han and Y. Zheng, "Security characterization and integrity assurance for component-based software," in *Proceedings of the International Conference on Software Methods and Tools (SMT '00)*, pp. 61–66, IEEE Computer Society, Wollongong, Australia, 2000.

[5] F. Jabeen and M. Jaffar-Ur Rehman, "A framework for object oriented component testing," in *Proceedings of the IEEE International Conference on Emerging Technologies, (ICET '05)*, pp. 451–460, Islamabad, Pakistan, September 2005.

[6] K. Md. Khan, J. Han, and Y. Zheng, "Characterizing user data protection of software components," in *Proceedings of the Software Engineering Conference*, pp. 3–11, Canberra, Australian, 2000.

[7] K. M. Khan and J. Han, "A security characterisation framework for trustworthy component based software systems," in *Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC '03)*, pp. 164–169, November 2003.

[8] K. M. Khan and J. Han, "Assessing security properties of software components: a software engineer's perspective," in *Proceedings of the Australian Software Engineering Conference (ASWEC '06)*, pp. 199–208, Sydney, Australia, April 2006.

[9] K. Khan, J. Han, and Y. Zheng, "A framework for an active interface to characterise compositional security contracts of software components ," in *Proceedings of the Australian Software Engineering Conference (ASWEC '01)*, pp. 117–126, 2001.

[10] "Common criteria project/ISO. Common criteria for information technology security evaluation," version 2.1 (ISO/IEC International Standard 15408). NIST, USA and ISO, Switzerland, 1999, http://csrc.nist.gov/cc/.

[11] O. H. Alhazmi and Y. K. Malaiya, "Quantitative vulnerability assessment of systems software," in *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS '05)*, pp. 615–620, January 2005.

[12] Y.-K. Zhang, S.-Y. Jiang, Y.-A. Cui, B. W. Zhang, and H. Xia, "A qualitative and quantitative risk assessment method in software security," in *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering, (ICACTE '10)*, pp. V1534–V1539, Chengdu, China, August 2010.

[13] X. Tang and B. Shen, "Extending model driven architecture with software security assessment," in *Proceedings of the 3rd IEEE International Conference on Secure Software Integration Reliability Improvement, (SSIRI '09)*, pp. 436–441, Shanghai, China, July 2009.

[14] X. Wang, H. Shi, T. Y. W. Huang, and F. C. Lin, "Integrated software vulnerability and security functionality assessment," in *Proceedings of the 18th IEEE International Symposium on Software Reliability Engineering (ISSRE '07)*, pp. 103–108, Trollhattan, Sweden, November 2007.

[15] K. Goševa-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Performance Evaluation*, vol. 45, no. 2-3, pp. 179–204, 2001.

[16] "NVD Common Vulnerability Scoring System (CVSS)," http://nvd.nist.gov/cvss.cfm.

[17] Y. Jiang, G. M. Xin, J.-H. Shan, L. Zhang, B. Xie, and F.-Q. Yang, "Method of automated test data generation for web service," *Chinese Journal of Computers*, vol. 28, no. 4, pp. 568–577, 2005.

[18] M. E. Delamaro, J. C. Maldonado, and A. P. Mathur, "Interface mutation: an approach for integration testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 3, pp. 228–247, 2001.

[19] J. M. Voas and K. W. Miller, "Predicting software's minimum-time-to-hazard and mean-time-to-hazard for rare input events," in *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pp. 229–238, October 1995.

[20] J. Chen, Y. Lu, and X. Xie, "A fault injection model of component security testing," *Computer Research and Development*, vol. 46, no. 7, pp. 1127–1135, 2009.

[21] J. Chen, Y. Lu, W. Zhang, and X. D. Xie, "A fault injection model-oriented testing strategy for component security," *Journal of Central South University of Technology*, vol. 16, no. 2, pp. 258–264, 2009.

[22] J. Chen, Y. Lu, X. Xie et al., "Design and implementation of an automatic testing platform for component security," *Computer Science*, vol. 35, no. 12, pp. 229–233, 2008.