http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 7 (5): 769-775, 2008
ISSN 1812-5638

Testing Approach of Component Security Based on Dynamic Fault Tree

J. Chen, Y. Lu and X. Xie
School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei, 430074, China

Abstract: This study proposes a testing approach of component security based on dynamic fault tree and then specifies some related definitions of fault tree, fault injection model and attack pattern. A testing algorithm of component security based on dynamic fault tree and test-case generating approach are also proposed. The proposed testing approach generates fault injection cases which can trigger component vulnerabilities in maximum probability based on fault tree. At the same time, the fault tree can be improved according to the testing results after injecting faults. The proposed approach was implemented based on research projects CSTS (Component Security Testing System). The experimental results show that the approach is effective and can trigger lots of component exceptions by using fewer test-cases.

Key words: Component testing, component security, attack pattern, fault tree, test case generation

INTRODUCTION

Component-Based Software Engineering (CBSE) has been the research focus in the field of software engineering at present. All kinds of new component technologies are aimed to enhance the efficiency of component development and performance. However, problems with the component reliability and security have not yet been solved. Testing the component and component system is an important process which guarantees and enhances system correctness, reliability and security. Current component testing approaches have focused on component functionality testing, which, to some extent, ensures the correctness and integrity of component functionality (Mao and Lu, 2006). To the best of our knowledge, component security testing is rarely researched as a special subject and there are not feasible approaches or technologies in detecting component security vulnerabilities. The research on component security testing mainly focuses on component security characterization and assessment approaches, formalization analysis approaches and component deployment testing (Han and Zheng, 2000; Khan and Han, 2003; Jabeen and Jaffar-Ur-Rehman, 2005; Han and Khan, 2006).

Since, most component source codes are unavailable and the components are extremely independent, the traditional software testing approaches are not suitable for the component security testing (Zhong and Edwards, 1998; McGraw, 2004; McGraw and Allen, 2004). The reasons are as: (1) traditional software testing technologies focus on the entire software system, rather than on fault model, (2) they have no mature fault model, (3) the testing effects are uncertain, (4) it is very difficult to test the system environment-related errors and small probability failures and (5) some specific faults such as memory leakage and buffer overflow cannot be detected. The fault injection technique just can make up for the deficiencies of traditional testing technique. As a nontraditional testing technique, fault injection technique is that faults are purposely generated according to the specific fault model and then are imposed on the tested system to accelerate the system's errors and failures occurred (Hsueh *et al*., 1997; Voas, 1997). The dynamic monitoring mechanism can record the running state of tested component. Then, the component vulnerabilities can be evaluated by analyzing the monitoring log.

Fault injection technique including environment fault injection and interface fault injection is suitable for testing component security. We once proposed the testing approaches of component security based on dynamic monitoring and fault injection (Chen and Lu, 2007a, b). However, the approaches lack feasible fault injection strategy and effective generating mechanisms of fault case which may cause security vulnerabilities of component.

FAULT INJECTION MODEL AND ATTACK PATTERN

The testing technologies can effectively trigger component exceptions based on fault injection model to

Corresponding Author: Jinfu Chen, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei, 430074, China

expose the problem with component security. The fault injection model (Chen and Lu, 2008a) is firstly introduced below and then the attack pattern and the relationship which is between fault injection model and attack pattern are introduced.

Fault injection model: The component security testing is different from the traditional functionality testing. Fault injection is an effective testing approach of software security (Du and Mathur, 2002). Software fault injection is divided into environment fault injection and program fault injection. It is supposed that R and T, respectively represent the fault injection requirement set and fault injection test-case set of CUT (Component Under Testing). It is an assumption that the two sets are non-empty limited set. As for any subset R'($R' \subseteq R$) of fault injection requirement set, there exists a subset T'($T' \subseteq T$) which can cover the test requirement set. It is supposed that n represents the base of fault injection requirement set and m represents the base of test-case set, that is, $|R| = n$, $|T| = m$. The formal definitions of fault injection model are presented below:

Definition 1: FIM (fault injection model) is a three-tuple {R, T, S} and R represents testing requirement set of fault injection and T represents test-case set of fault injection and S represents binary operation $S(2^R,2^T)$, that is, $S(2^R,2^T)=\{(R', T') \in 2^R \times 2^T$, the test case set T' covers the testing requirement set R'$\}$. In the case of without causing confusion, the $S(2^R,2^T)$ can be abbreviated as S.

Application users include visible users (the human being) and invisible users such as API, OS (Operating System) and file system etc. (Whittaker, 2001). With the consideration of the security testing for tested component, invisible users must be taken into account in the fault injection testing. Based on the theory about software's invisible users, some definitions are presented below.

Definition 2: R = {IP, M, DF, PRS, NET, REG, EV} and IP, M, DF, PRS, NET, REG, EV refers to interface parameter, memory, disk file system, external component or process, network, registration information and environment variable, respectively. R shows the above-mentioned seven aspects injected into CUT. According to the different test objects and needs in practical application, R can be extended.

Definition 3: FIF = $\{fc_1, fc_2, \ldots, fc_i, \ldots, fc_n\}$, that is, the fault injection factor FIF includes some fault injection cases which are represented by some codes such as fc_i.

Definition 4: $\forall T' \in 2^T$, Regt(T') represents all the testing requirement set which is covered by the test-case set T'.

Table 1: A fault injection table

Fault injection types	Fault injection values	Fault injection codes
IP	Input parameters	10
	Parameter types	11
	Parameter sequence	12
	…	…
DF	File only read	20
	Invalid file	21
	Unfound file	22
	…	…
M	Lower memory	30
	Access control	31
	Invalid address	32
	…	…
REG	Access denied	40
	Corrupt registry	41
	Unfound key	42
	…	…
PRS	Unfound dll file	50
	Resource insufficiency	51
	Invalid type	52
	…	…
NET	Network congestion	60
	Invalid port	61
	Network discounect	62
	…	…
EV	Invalid access	70
	Racing condition	71
	Invalid value	72
	…	…

Definition 5: $\forall R' \in 2^R$, Test(R') represents all the test-case set which covers the testing requirement set R'.

Definition 6: $\forall R' \in 2^R$, T' = Test(R'), Expt(T') represents the exception number which is triggered by the test-case set T' when given the testing requirement set R'.

Given the fault injection testing requirement set R', there are two types of test-case in test-case set. One type is necessary test-case set and the other is redundant test-case set (Changhai and Baowen, 2003). As for test-case set $T' \in 2^T$, if $Regt(T-T') \neq R'$, T' is indispensable. On the contrary, if Regt(T-T') = R', T' is redundant.

Definition 7: If Regt(T) = R, T is the smallest test-case set, iff $\forall T_1 \subset T$, $Regt(T_1) \neq R \wedge Expt(T) > Expt(T_1)$.

If IP = {10,11,12,...}, DF = {20, 21, 22,...}
M = {30, 31, 32,...}, REG = {40, 41, 42,...}
PRS = {50, 51, 52,...}, NET = {60, 61, 62,...}

EV = {70, 71, 72,...}, a fault injection table is shown as Table 1 based on fault injection model.

Attack pattern: Attack pattern describes the general approach to crack software. Although, it is a good approach for exploiting software vulnerabilities, techniques for exploiting software tend to be few in number and fairly specific. This means that applying common techniques often results in the discovery of new

Table 2: The part of attack pattern

Attack pattern	Code	Type
File system function injection, content based	229	DF
Target programs that write to privileged OS resources	152	PRS
Client-side Injection, buffer overflow	231	M
User-controlled filename	217	DF
Direct access to executable files	162	PRS
Using escaped slashes in alternate encoding	270	IP
Alternative IP addresses	274	NET
Argument injection	169	IP
Command delimiters	172	IP
Parameter expansion	298	IP
User-supplied variable passed to file system calls	185	EV
Overflow binary resource file	293	PRS
Postfix, null terminate and backslash	186	IP
Relative path traversal	187	REG
Client-controlled environment variables	189	EV
User-supplied global variables	190	EV
Session ID, resource ID and blind trust	192	REG
HTTP query strings	216	NET
Buffer overflow in an API call	297	PRS
Simple script injection	214	IP
Overflow symbolic links	294	REG
XSS in HTTP headers	216	IP

software exploits. A particular exploit usually refers to the extension of a standard attack pattern to a new target. Classic bugs and other flaws can thus be leveraged to hide data, escape detection, insert commands, exploit databases and inject viruses. Clearly, the best way to learn to exploit software is to familiarize yourself with standard techniques and attack patterns and to determine how they are instantiated in particular exploits (Hoglund and McGraw, 2004).

An attack pattern is a blueprint for exploiting software vulnerability. In other words, an attack pattern describes several critical features of the vulnerability and arms an attacker with the knowledge required to exploit the target system. The general attack pattern can be attained by analyzing the general classification model of software vulnerability. The part of attack pattern is shown in Table 2 (Hoglund and McGraw, 2004).

The code column of Table 2 is attack code which is also considered as the fault injection code in accordance with FIM. The type column of Table 2 is fault type of FIM.

COMPONENT SECURITY TESTING APPROACH BASED ON DYNAMIC FAULT TREE

Approach framework: Fault tree has firstly been applied to analyze the hardware system. It is in application to qualitatively and quantitatively analyze the failure mode of critical system for a long time. Fault tree provides a graphic form and the logical framework for analyzing the system failure modes. The event combination of system failure is also provided by fault tree through the graphical and mathematic representation. On the one hand, the fault tree can illustrate the combination of event which can lead to the observed failure symptoms, thereby helping to diagnose failure symptoms; On the other hand, the fault tree indicates the probability of failure event to obtain the system failure probability for assessing system security and reliability. The definitions of fault tree and other related term are presented below in this study.

Definition 8: Fault tree is a tree. All events which trigger component vulnerabilities are considered as root node of tree, which is top event represented as TE. The root node weight represents the sum of fault factor. The branch node represents the logical operators such as OR DOOR and AND DOOR, which indicate OR relationship and AND relationship between children nodes. The branch node weight represents the fault factor sum of sub tree. The leaf node is basic event. The leaf node weight represents occurrence probability with which basic event triggers component vulnerabilities. Therefore, fault tree can show all kinds of logical combination of basic event which cause component vulnerabilities.

Definition 9: The basic events represent the fault factors which are injected into tested component.

Fault tree is derived from the actual model of system in component security testing. The basic event is the factor of triggering component vulnerabilities or attack pattern. The top event is the logical combination of basic event. The security testing is aimed at finding errors as much as possible which can trigger component vulnerabilities. The testing process of component security is shown in Fig. 1 based on fault tree.

The steps of creating fault tree are as follows:

- Generate a top event TE regarded as root node, which represents the sum of fault injection factors
- Generate a OR DOOR G regarded as the sub-node of TE
- Generate No.1 level faults such as M, DFS, IP, NET, REG, PRS and EV, which are regarded as the children nodes of G in accordance with the FIM
- Generate No. 2 level faults regarded as the children nodes of No. 1 level faults according to each No. 1 level fault
- Update fault tree according to attack pattern, published component vulnerabilities and experiential knowledge and then record the weight of branch node

Dynamically modify the weight of leaf node in accordance with the known causes which trigger the component vulnerabilities. If there are several factors which trigger component vulnerabilities at the same time,

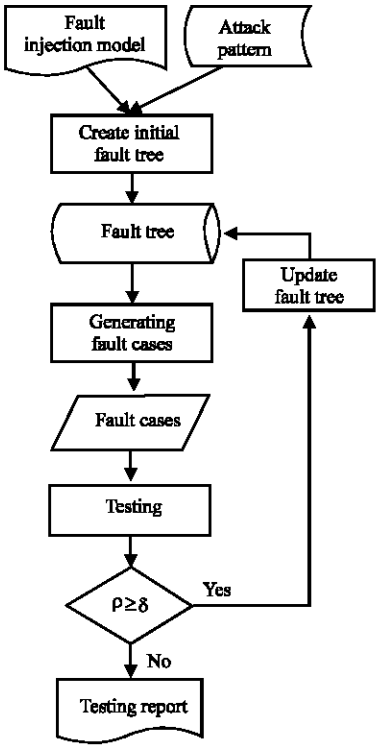

Fig. 1: Testing process based on fault tree

appending the AND DOOR Z regarded as the right brother of No. 1 level fault node and then add the multi-factor fault as the children node of AND DOOR Z.

A fractional fault tree, which is generated according to FIM and attack pattern, is shown in Fig. 2.

The nodes represent the fault types. The weight near the branch nodes and leaf nodes is fault number and the happened probability of fault, respectively.

Testing algorithm based on dynamic fault tree: The fault injection test-cases can be generated by the N factor coverage algorithm which is presented in the fault injection model of component security (Chen and Lu, 2008a). Although N factor coverage algorithm can achieve higher coverage, there are still redundant test cases. As a useful supplement to this method, this study presents a testing approach of component security based on dynamic fault tree. The fault test cases are also generated based on dynamic fault tree.

An initial fault tree is created in accordance with FIM and attack pattern. The root node represents all the fault types. The No. 1 level nodes represent first-class faults such as memory fault, disk file faults, register information fault, process fault, network fault and environment variable fault. The No. 2 level nodes represent the sub class of No. 1 level nodes. Followed by analogy, the last

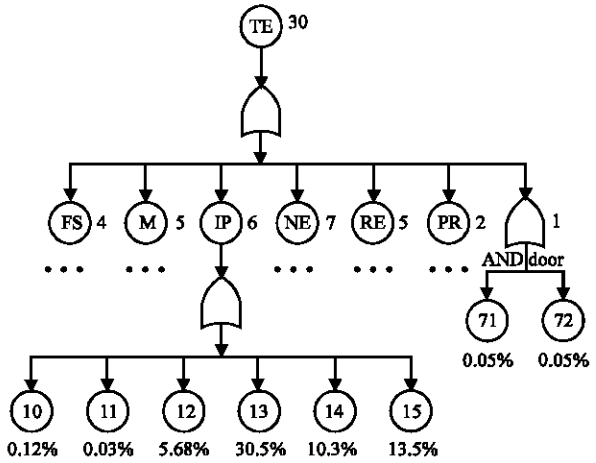

Fig. 2: A fractional fault tree based on FIM and attack pattern

leaf nodes are on behalf of specific factors which trigger component vulnerabilities. The weight of branch node represents fault number. The weight of leaf node represents the probability of happened vulnerability. With the initial fault tree, next to the existing vulnerabilities of some component are analyzed to mark the probability of triggered component vulnerability. A testing algorithm of fault injection based on dynamic fault tree named TADFT is presented in the Algorithm 1.

Algorithm 1: Testing algorithm based on dynamic fault tree (TADFT)
Input: Initial fault tree FT; probability threshold value δ; Dynamic monitoring log (DML); Security requirement specification (SRS).
Output: Testing report.
01 Initialize fault case set, let TG = Null, set triggering vulnerability probability threshold value δ;
02 Scan the fault tree and get the greatest uncompared probability ρ
03 while(ρ≥δ) do
04 {
05 Add the fault which is marked by ρ to TG;
06 Mark the greatest probability node(s) as compared node(s);
07 while (TG≠Null) do
08 {
09 Inject the faults of TG one by one, dynamically monitor testing result and then write into Security Log;
10 if (CVD(DML, SRS)) then //Call CVD algorithm
11 {
12 Dynamically update root node and branch node weight of fault tree according to injecting fault types;
13 Dynamically update leaf node weight of fault tree according to injecting fault types;

```
14      Write testing results into testing report;
15      }
16     Obtain next fault cases of TG;
17    }
18    TG = Null;
19      Get the greatest uncompared probability ρ in fault
tree;
20  }
21  if (Testing report is not null) then
22  {
23   Output testing report;
24 } // Algorithm over;
```

The algorithm of component vulnerability detection on component is shown in Algorithm 2.

Algorithm 2: Component Vulnerability Detection (CVD)
Stipulation: R expresses a record in dynamic monitoring log.
Input: Dynamic monitoring log (DML); Security Requirement Specification (SRS).
Output: True or False.

```
01  {
02   Scan records in security log one by one;
03  while(Security log has unprocessed R) do
04   {
05   if (Have the exception information in the R) then
06    {
07      Exist explicit security exception;
08      Return True;
09    }
10      if (Method invocation sequence in R violate the
security restraints ) then
11    {
12      Exist implicit security exception;
13      Return True;
14    }
15   Mark R as processed record;
16    Goto next unprocessed R in security log;
17  }
18  Return False;
19 }
```

The approach can inject corresponding faults into tested component according to the fault tree when the tested component is driven and then can further trigger component security vulnerabilities. At the same time, dynamic monitoring mechanism can record the invocation sequence of component method and the running state of component. When a testing task is finished, the testing system can analyze the testing report for component vulnerability by using the evaluation mechanism. Then, we can get the testing results.

In order to theoretically analyze the complexity of algorithm 2, we assume that the security log file records one tested component and each tested component averagely has n methods in a certain period of time. Each component method is compared with the relevant component method one time. The complexity of algorithm 2 is $O(n^2)$. As for algorithm 1, we assume that the node number of fault tree is m and then the complexity of visiting the tree is O(m). Thus, the algorithm 1 complexity is $O(m \times n^2)$.

EXPERIMENTAL ANALYSIS

The approach was implemented in our CSTS (Component Security Testing System) platform to test its effectiveness and performance (Chen *et al*., 2008b). Then, many experiments were carried out to verify the approach. We collected 38 components in which security vulnerabilities exist for experimental analysis in the CVE (Common Vulnerabilities and Exposures) and other security web sites. When initial fault tree is created, testing requirement set R of FIM is seven-tuple {IP, M, DF, PRS, NET, REG, EV} and the value of attack pattern is derived from Table 2. Each fault injection factor is expanded to choose several parameters (fault injection items) on the basis of Table 1. All item values of R and attack pattern are as follows: IP = { integer beyond the scope, string beyond the length, integer changed character, the character changed integer, real parameter number more than symbol parameter number, real parameter number less than symbol parameter number, the integer is prior to the character, the character is prior to the integer}; M = {segment locked, memory insufficiency, invalid address, invalid access, too small page file, memory space limit, memory read only, cannot allocate memory}; DF = {file only read, file already existence, file unfound, file locked, file data error, file access denied, file in use, file uncreated}; PRS = {process file unfound, library file insufficiency, invalid file type, access denied, invalid system dll, cannot register COM, cannot load system dll, corrupt process file}; NET = {network disconnect, no available ports, network is not connected, network API unfound, network latency, network congestion, denial service, invalid connecting}; REG = {access denied, registry destroyed, keys do not exist, registry IO failure, invalid values, invalid keys, cannot write registry, cannot create registration item}; EV = {access denied, racing condition, illegal modification, invalid value}; Attack pattern = {229, 152, 231, 217, 162, 270, 274, 169, 172, 298, 185, 293, 186, 187, 189, 190, 192, 216, 297, 214, 294, 216}.

Table 3: Testing result of system

Testing methods	Tested components	Exceptional components	Detection rate (%)
Random	38	11	28.95
Two-factor	38	28	73.68
Fault injection tree	38	32	84.21

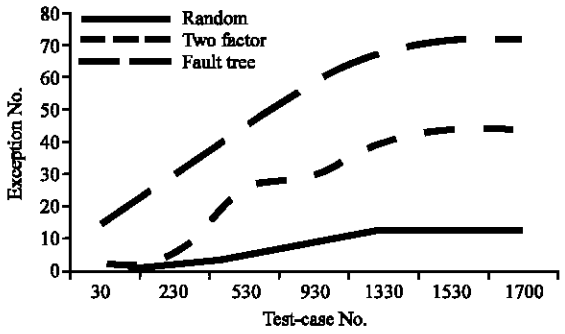

Fig. 3: The comparison of testing result for three approaches

We respectively adopt random testing method, two-factor testing method and the testing method based on fault tree for testing component vulnerabilities in the experiment. At first, some injected faults which are generated by above-mentioned three approaches are injected into the tested component. Then dynamic monitoring mechanism will record the security log and exceptional information. The experimental results are shown in Table 3.

The testing approach based on fault injection tree triggers the largest number of component exception with the detection rate of 84.21%. The exceptional number of the component triggered by the approach of two-factor injection is smaller than the exceptional number triggered by the testing approach based on fault injection tree with the detection rate of 73.68%. The component exceptions triggered by the random testing approach are the least with the detection rate of 28.95%.

We do contrastive experiments to further validate the relationship between different testing approaches, test-case number and triggered exception number in the CSTS. The results show that the exception number is rising along with the increase of test-case (Fig. 3). Under the same number of test case, test cases generated on the basis of dynamic fault tree trigger the largest exception number; test cases of two-factor combination trigger the second largest exception number; Test cases randomly generated trigger the least exception number. The exception number is stabile when test case number is at a certain level. Test cases, which are increased when the exception number is stabile, are redundant cases.

CONCLUSIONS

In the component-based software development and maintenance activities, it is necessary to test the reliability and security of the third-party component. However, the testing activity is always neglected. Thus, it is urgent to find an effective testing approach of component security. This research proposed a testing approach of component security based on dynamic fault tree. The paper defined and discussed fault injection model, attack pattern and dynamic fault tree in details. A testing algorithm based on dynamic fault tree named TADFT was also presented for detecting component vulnerabilities. The proposed approach mainly has the following advantages:

- The fault injection tree is comprehensive. It not only records the fault type and number, but also describes the probability of trigger component exceptions
- The function of fault injection is powerful. The faults are injected into tested component from static level to dynamic level to trigger component exceptions
- It has nice operability
- The detection rate is more than 84% for detecting published component vulnerabilities

The shortcoming of the approach is that there is not a good learning algorithm for automatically generating the fault tree. There will be much important work in future research. Firstly, the learning algorithm in which the fault tree are automatically generated and updated by the approach should be proposed. Secondly, some evaluation mechanisms after testing should be worked out to evaluate the security level in details. Finally, a vulnerability database of component should be established based on fault injection model.

ACKNOWLEDGMENT

This research is supported by the National Research Foundation under grant No. 513150601.

REFERENCES

Changhai, N. and X. Baowen, 2003. A minimal test suite generation method. Chin. J. Comput., 26: 1690-1695.

Chen, J. and Y. Lu, 2007a. Testing approach of component security based on dynamic monitoring. Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007), Aug. 13-15, Iowa City, IA, USA., pp: 381-386.

Chen, J. and Y. Lu, 2007b. Testing approach of component security based on fault injection. International Conference on Computational Intelligence and Security (CIS'2007), Dec. 15-19, Harbin, China, pp: 763-767.

Chen, J. and Y. Lu, 2008a. A fault injection model of component security testing. J. Comput. Res. Dev.,

Chen, J. and Y. Lu, 2008b. Design and implementation of an automatic testing platform for component security. Comput. Sci.,

Du, W. and A.P. Mathur, 2002. Testing for software vulnerability using environment perturbation. Qual. Reliab. Eng. Int., 18: 261-272.

Han, J. and Y. Zheng, 2000. Security characterisation and integrity assurance for component-based software. International Conference on Software Methods and Tools (SMT 2000), Nov. 6-9, Wollongong, NSW, Australia, pp: 61-66.

Han, J. and K.M. Khan, 2006. Assessing security properties of software components: A software engineer's perspective. Proceedings of the 2006 Australian Software Engineering Conference (ASWEC), Apr. 18-21, Sydney, NSW, Australia, pp: 199-210.

Hoglund, G. and G. McGraw, 2004. Exploiting Software How to Break Code. 1st Edn., Addison Wesley, Boston, USA., ISBN: 0-201-78695-8, pp: 512.

Hsueh, M.C., T.K. Tsai and R.K. Lyer, 1997. Fault injection techniques and tools. Computer, 30: 75-82.

Jabeen, F. and M. Jaffar-Ur-Rehman, 2005. A framework for object oriented component testing. International Conference on Emerging Technologies, Sep. 17-18, Islamabad, Pakistan, pp: 451-460.

Khan, M.K. and J. Han, 2003. A security characterisation framework for trustworthy component based software system. Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC), Nov. 3-6, Dallas, TX, USA., pp: 164-169.

Mao, C. and Y. Lu, 2006. Research progress in testing techniques of component-based software. J. Comput. Res. Dev., 43: 1375-1382.

McGraw, G., 2004. Software Security. IEEE Security Privacy, 2: 80-83.

McGraw, G. and B. Allen, 2004. Software security testing. IEEE Secur. Privacy, 2: 81-85.

Voas, J., 1997. Fault injection for the masses. Computer, 30: 129-130.

Whittaker, A.J., 2001. Software's invisible users. IEEE Softw., 18: 84-88.

Zhong, Q. and N. Edwards, 1998. Security control for COTS components. Computer, 31: 67-73.