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 10 (11): 2208-2213, 2011
ISSN 1812-5638 / DOI: 10.3923/itj.2011.2208.2213

An Improved Software Behavior Model in System Call Level and Trustworthiness Evaluation

[1]Zhen Li, [1]Junfeng Tian and [2]Liu Yang
[1]College of Mathematics and Computer, Hebei University, China
[2]Experiment Teaching Center of Economy and Management, Hebei University, China

Abstract: The traditional software behavior models are often based on system calls. This study proposes an improved software behavior model in system call level and the method of corresponding trustworthiness evaluation. The model overcomes some drawbacks of traditional software behavior models: low accuracy of system call arguments policies, inability to monitor the software behavior between adjacent system calls etc., First, several system call attributes are introduced and their deviation degrees are taken into account comprehensively for more overall and more accurate software behavior monitoring. Second, the policies of system calls and their arguments based on context are proposed which can be used to detect deviant software behavior aiming at control flow or data flow about system calls. Third, the method of clustering and quantifying time intervals between adjacent system calls is presented, which can decide whether the software behavior between adjacent system calls is trustworthy to some extent through monitoring the deviation degree of time interval. The experimental results show that our model can model the software behavior more accurately and has better deviation detection ability of software behavior than traditional models.

Key words: Software behavior, system call, time interval, trustworthiness evaluation

INTRODUCTION

With continuous deepening of the application of software in the sensitive fields such as finance, military affairs and economy, the requirement of software trustworthiness becomes more urgent. If the software behavior is always accordant with the expected behavior, we call the software is trustworthy (Wang *et al*., 2006). How to ensure the software trustworthiness during software development and running has become an important research direction of software theory and technology (Chen *et al*., 2003).

Because system call is the interface provided by the operating system to access system resources, the traditional software behavior models often use system calls as the monitoring points of software behavior. They focus on system call sequences or system calls in the software behavior trace, such as Forrest's model based on short sequences of system calls (Forrest *et al*., 1996), Var-gram model of variable-length audit trail patterns (Wepsi *et al*., 2000), automaton models (Wagner and Dean, 2001; Feng *et al*., 2003; Liu *et al*., 2005), anomaly detection model of program behavior combining system call with Markov chain (Frossi *et al*., 2009). In the aspect of software behavior monitoring and evaluation, Man *et al*. (2009) study the trust analysis and trend prediction of software interactive behavior. Zhang *et al*. (2010) propose a fuzzy-based method of evaluating the trustworthiness of software processes. Shanlin *et al*. (2009) study the evaluation mechanism of trustworthy software and propose an approach of trustworthy software evaluation using utility based evidence theory. The above software behavior models can be divided into two categories. One is to construct feature library of normal software behavior through statistical learning or rule mining. The other one is to model the software behavior by constructing the software behavior automaton. The first one determines whether the software behavior is abnormal or not according to the feature library of normal behavior. It cannot describe the running trace of entire software. The second one can describe the running trace of software but it is not enough to model the software behavior accurately because it is confined to system call and simple arguments policies.

However, although, system calls can be used to model the software behavior, there is still some information of software behavior that cannot be expressed just through system calls. For example, it is helpless to monitor the software behavior between adjacent system calls. Both Jones and Li (2001) and Pu and Lang (2007)

Corresponding Author: Zhen Li, College of Mathematics and Computer, Hebei University, China

collect all matched short system call sequences into groups called sequence clusters and train the sequence of time intervals for each sequence cluster. While, the time interval between two adjacent system calls in a sequence cluster can vary obviously because the path through the application code can vary which both Jones and Li (2001) and Pu and Lang (2007) don't distinguish. That is, although, two short system call sequences for normal software behavior are same, the time intervals of any two adjacent system calls among them may be different obviously because the program contexts of two short system call sequences or the software behavior trace between two adjacent system calls may be different.

For the above problems, the paper presents an improved software behavior model in system call level which can model the software behavior more accurately, and the method of corresponding trustworthiness evaluation.

- Several system call attributes are introduced and their deviation degrees are taken into account comprehensively for the deviation degree of system call and for the software trustworthiness evaluation further
- The policies of system calls and their arguments based on context are proposed which can be used to detect deviant software behavior aiming at control flow or data flow about system calls
- The method of clustering and quantifying time intervals between adjacent system calls is presented, which can decide whether the software behavior between adjacent system calls is trustworthy to some extent, through monitoring the deviation degree of time intervals

An improved software behavior model in system call level: This study proposes an improved software behavior model in system call level based on HPDA model (Liu *et al*., 2005). The model introduces several new system call attributes and the policies of them. It is defined as a 5-tuple: (S, Σ, T, s, A).

- S is a finite set of states
- Σ is a finite set of input symbols. In our model, the set of input symbols is defined as:

$$\Sigma = (X \times Addr) \cup SC \cup \varepsilon \quad (1)$$

where,

$$X = \{Entry, Exit\}, \{sc_1, sc_2, ...\}\varepsilon$$

is the empty string. Addr is the set of all possible addresses defined by the executable software and Entry and Exit are two new added system calls. Entry is added before each function call and Exit is added after each function call. Entry and Exit get the return address before and after the function call, respectively. SC is the set of system calls. For each system call, there is a group of attributes represented as Attr = <SN, CV, SA, TI>, where SN, CV, SA and TI are system call name, system call context, system call arguments polices and time interval, respectively.

- T is the set of transition functions: $(S \times \Sigma) \rightarrow S$
- s is the start state: $s \in S$
- A is the set of accept states: $A \subseteq S$

For system call sc_{i-1}, it is monitored involving the aspects of control flow and data flow and the behavior between last system call sc_{i-1} and current system call sc_i. We introduce the fuzzy theory and regard the deviation of software behavior trace as a fuzzy concept. The trustworthiness of software behavior depends on the deviation degree from normal trace.

r_{SN_i}, r_{CV_i}, r_{SA_i} and denote the deviation degree of system call name, system call context, system call arguments policies and time interval, respectively. Let universe $U = \{(SN_i, CV_{ij}, SA_{ik}, TI_{il})\}(1 \le i \le p, 1 \le j \le n, 1 \le k \le q, 1 \le l \le t)$, where p is the number of system calls, n, q and t denote the number of system call contexts, actual arguments and time intervals, respectively. The fuzzy set $\tilde{R}$ denoting the deviation of system call is the fuzzy set on universe U. The deviation of any attribute in SN_i, CV_i and SA_i can determine the deviation of system call, so r_{SN_i}, r_{CV_i}, $r_{SA_i} \in \{0,1\}$ and the deviation of system call is 1 if $r_{SN_i} = 1$ or $r_{CV_i} = 1$ or $r_{SA_i} = 1$. Considering the fuzziness of TI_i for the deviation of system call, $r_{TI_i} \in [0,1]$ and the deviation of system call depends on r_{TI_i} if $r_{SN_i}, r_{CV_i}, r_{SA_i}$ are all equal to 0. The membership function of fuzzy set $\tilde{R}$ is as follows:

$$\tilde{R}(r_{SN_i}, r_{CV_i}, r_{SA_i}, r_{TI_i}) = \max\{r_{SN_i}, r_{CV_i}, r_{SA_i}, r_{TI_i}\} \quad (2)$$

$$r_{SN_i}, r_{CV_i}, r_{SA_i}\ r_{TI_i}, \in \{0,1\}, \in [0,1]$$

The deviation degree of sc_i, represented as r_{sc_i}, is the membership degree for $\tilde{R}$. In the real-time detection, if r_{sc_i} is larger than or equal to the threshold τ, it is reported that the software behavior is untrustworthy.

System call arguments policies based on context: System call context is represented by calling stack of system call. If the calling stacks of two system calls are same, the system call contexts of two system calls are same; otherwise, the system call contexts of two system calls are different. For current call stack, if the return address of

function main is a_0, the return address of internal functions is a_1,....., a_{m-1} in sequence and the return address of last system call is a_m, then the system call context can be expressed as (a_0, a_1,…, a_{m-1}, a_m).

Le A^{sc_i} t =< $a_1^{sc_i}$,…, $a_n^{sc_i}$ > be the vector of formal arguments for system call sc_i. Each invocation sc_{ij} of sc_i has a concrete vector of values for A^{sc_i} defined as $A^{sc_{ij}}$ = < $a_1^{sc_{ij}}$,…, $a_n^{sc_{ij}}$ >. An argument set for system call sc_i in the system call context Cont is the set of all argument vectors $A^{sc_{ij}}$ observed for the chosen system call issued in the context Cont. This is denoted by $AS(Cont, sc_i)$.

We limit our attention to unary and binary relations on system call arguments in our model.

Unary relations are attributes of a single argument. They can be represented using the form. $A^{sc_{ij}} \in AS(Cont, sc_i)$ Such a relationship can be learnt from software behavior traces. The set $AS(Cont, sc_i)$ is represented using an enumeration by listing all elements of $A^{sc_{ij}}$ in the given system call context Cont.

Binary relations are relationships between two system call arguments or between one system call argument and another system call return value. Let's take equality relation represented as equal for example. The file descriptor fd_1 returned by open system call in context $Cont_1$ is equal to the operand fd_2 of a subsequent write system call in context $Cont_2$.

$$fd_1(Cont_1, open) \quad equal \quad fd_2(Cont_2, write) \tag{3}$$

Time interval

Definition 1 Time interval: Time interval is the difference between the beginning time of current system call and the beginning time of last system call. If the current system call is the first system call, its time interval is 0.

After removing various effects of the environment, most time intervals between adjacent system calls are approximately normally distributed (Jones and Li, 2001). Based on this, we introduce time interval to the traditional software behavior models.

Clustering of time intervals: The distribution of time intervals between adjacent system calls in some situations, described by Jones and Li (2001), has multiple peaks. The reason for such multiple-peak distribution of time intervals is likely because several different execution paths through the application code that exhibits the same system call sequence. The time intervals between adjacent system calls for the same system call sequences of normal software behavior may be different obviously because of different program context. Moreover, even if the program contexts of two adjacent system calls are same, the time interval of system call in normal trace may be also different obviously because of different software behavior trace between two adjacent system calls. Therefore, first of all, our model forms time interval samples of the same system call with the same program context, through running the software multiple times in given environment. Then cluster these time interval samples by a clustering algorithm based on entropy (Wang and Peng, 2007). For system call sc_i, the clustering process of m samples of time interval T_i is as follows:

Step 1: Number each sample as a class C_k, where $1 \le k \le m$ and C_k is composed of N_k samples

Step 2: Choose any sample, add it to another class and compute the entropy of C_k after adding the sample to it according to formula (4) h is the width of window or smoothing coefficient). The sample is allocated to the class which has the smallest added value of entropy

$$H(C_k) = -\log[\frac{1}{2\pi N_k^2 h^2} \sum_{i=1}^{N_k} \sum_{j=1}^{N_k} \exp(-\frac{(x_i - x_j)(x_i - x_j)^T}{2h^2})] \tag{4}$$

Step 3: Number the classes again after allocating samples

Step 4: Repeat step 2 and 3 until there is no class which has only one sample

Step 5: Compute the between-class entropy of any two classes according to formula (5) and merge two classes with smallest between-class entropy. The number of classes and samples for each class must be also modified

$$H(C_i, C_j) = -\log[\frac{1}{2\pi N_i^2 N_j^2 h^2} \sum_{i=1}^{m} \sum_{j=1}^{m} M(x_i, x_j) \exp(-\frac{(x_i - x_j)(x_i - x_j)^T}{2h^2})] \tag{5}$$

$M(x_i, x_j)$ in formula (5) is as follows:

$$M(x_i, x_j) = \begin{cases} 1, x_i \in C_i, x_j \in C_j \text{ or } \quad x_i \in C_j, x_j \in C_i \\ 0, \text{else} \end{cases} \tag{6}$$

Repeat step 5 until there are only two classes. If the changes of the smallest between-class entropy at a certain time are more significant than before, the classes at this time, represented as C_1, C_2 , …, C_M (M is the number of classes), is the result of clustering.

Deviation degree of time interval: We quantify each time interval based on the method of quantifying time series with SAX (Symbolic Aggregate Approximation) (Lin *et al*., 2003, 2007). The time interval T_i of system call sc_i is considered to obey normal distribution. W denotes the number of levels, that is, the number of characters mapped to.

Table 1: A lookup table containing breakpoints that divide a normal distribution in a number (3, 5, 7 and 9) of equiprobable regions

v β	3	5	7	9
β_{-4}				-1.22
β_{-3}			-1.07	-0.76
β_{-2}		-0.84	-0.57	-0.43
β_{-1}	-0.43	-0.25	-0.18	-0.14
β_1	0.43	0.25	0.18	0.14
β_2		0.84	0.57	0.43
β_3			1.07	0.76
β_4				1.22

Definition 2 breakpoints: Breakpoints are a sorted list of numbers such that the numbers $B=\beta_{-(w-1)},\cdots,\beta_{w-1}$ of areas under a N (0,1) normal curve is

$$\frac{1}{2w-1}$$

(β_{-w} and β_w are defined $-\infty$ as and ∞, respectively).

Considering the symmetry of time interval's deviation from 0, we select the symmetric intervals containing 0 from the lookup table containing the breakpoints that divide a normal distribution in an arbitrary number (from 3 to 10) of equiprobable regions in (Lin *et al*., 2003, 2007) and form our lookup table containing breakpoints. The number of equiprobable regions v is 3, 5, 7 and 9, as shown in Table 1.

When the software runs, our model captures the time interva T_i 1 of system call sc_i, computes the entropy of C_k after adding T_i to it according to formula (4) and allocates to the class which has the smallest added value of entropy. The time interval T_i of class C_e denotes T_{ie}. The calculative process of deviation degree is as follows.

First of all, time interval T_{ie} is normalized into, where μ_{ie} and σ_{ie} are the mean and the standard deviation of time intervals of class C_e, respectively. This makes the measurement of time intervals invariant to shifting and scaling.

$$T'_{ie}=\frac{T_{ie}-\mu_{ie}}{\sigma_{ie}}$$

Then quantify the time interval T'_{ie} by predefined breakpoints, that is, map the time interval into a level.

Definition 3 level: Level is an element of the set L = {0, 1, 2, 3, 4}, where 0 denotes normal, 1 denotes less normal, 2 denotes less abnormal, 3 denotes abnormal, 4 denotes very abnormal.

Time interval T'_{ie} is mapped into level:

$$\hat{T}_{ie}, \hat{T}_{ie} \in L$$

$$\hat{T}'_{ie}=\begin{cases}0, & \beta_{-j}\le T'_{ie}\le\beta_j\\ j, & 1\le j\le 4 \quad \text{and} \quad (\beta_{-(j+1)}\le T'_{ie}<\beta_{-j} \text{ or } \beta_j<T'_{ie}\le\beta_{j+1})\\ 4, & \text{else}\end{cases} \quad (7)$$

Note:

$$\beta_{-(j+1)}=-\infty,\beta_{-(j+1)}\le T_{ie}<\beta_{-j} \text{ in}\beta_{-(j+1)}<T_{ie}<\beta_{-j}$$

if in (7) should be: $\beta_{j+1}=\infty$ in $\beta_j<T'_{ie}\le\beta_{j+1}$ Equation 7 should be $\beta_j<T'_{ie}<\beta_{j+1}$.

When, $\hat{T}'_{ie}$ is 0, 1, 2, 3 and 4, τ_{TI} is 0, 0.25, 0.5, 0.75 and 1, respectively.

For the deviation degree of time interval, given upper threshold τ_{TI}, we call the time interval which satisfies $\tau_{TI}\ge\tau_{TI}$ the time interval with larger deviation. Our model can detect the deviation of software behavior which has time intervals with larger deviation through distinguishing time intervals of different program contexts or different software behavior traces between adjacent system calls.

Experiments and analyses: We have made experiments on a PC with Intel (R) Core (TM) 2 Duo E7500 2.93 GHz and 2 GB of main memory running Linux kernel 2.4.20. Each system call is intercepted by Loadable Kernel Module (LKM) and modified. The generation of the improved software behavior model in system call level adopts the similar method of HPDA (Liu *et al*., 2005). The differences between our model and HPDA are as follows:

- For system call, the time interval of should be recorded
- For two new system calls Entry and Exit, the return addresses before and after the function call should be recorded, respectively

We form the software behavior model of vi 6.1 in Red Hat 9 Linux after training and make some attacking experiments. Red Hat 9 Linux distribution includes vi 6.1 which exists potential TOCTTOU (Time of Check to Time of Use) vulnerabilities (Wei and Pu, 2005). Specifically, if vi 6.1 is run by root to edit a file owned by a normal user, then the normal user may become the owner of sensitive files such as /etc/passwd. The sequence of system calls that the vulnerability window <open, chown32> is related to is as follows:

..., open, write, close, chmod, stat 64, chown 32, chmod...

Table 2: A class of time intervals for each system call of vulnerability window (μs)

System call	Open	Write	Close	Chmod	Stat64	Chown32	Chmod
Sample mean	145.2	46.3	20.8	10.7	13.1	11.0	12.4
Sample deviation	15.4	17.9	7.1	6.4	15.2	6.8	10.6
Normal trace 1	148.0	49.0	22.0	11.0	15.0	10.0	11.0
Normal trace 2	145.0	50.0	21.0	10.0	13.0	11.0	12.0
Abnormal trace 1	147.0	50.0	22.0	11.0	12.0	18.0	12.0
Abnormal trace 2	146.0	45.0	20.0	12.0	11.0	32.0	11.0

Table 3: The deviation degrees of system calls

System call sequence			Open	Write	Close	Chmod	Stat64	Chown32	Chmod
Deviation	Normal	v=5	0	0	0	0	0	0	0
degree of	trace 1	v=7	0.25	0	0	0	0	0	0
system call	Normal	v=5	0	0	0	0	0	0	0
	trace 2	v=7	0	0.25	0	0	0	0	0
	Abnormal	v=5	0	0	0	0	0	0.50	0
	trace 1	v=7	0	0.25	0	0	0	0.50	0
	Abnormal	v=5	0	0	0	0	0	0.50	0
	trace 2	v=7	0	0	0	0.25	0	0.75	0

Trustworthiness evaluation of software behavior: We take software vi 6.1 as an example and discuss the trustworthiness of software behavior. Let threshold τ be 0.5.

Attack 1: The attack consists of a tight loop constantly checking whether the owner of the wfname file has become root. Once this happens, the attacker replaces the file with a symbolic link to /etc/passwd. When *vi* 6.1 exits, it should change the ownership of /etc/passwd to the attacker.

If the attack succeeds, when *vi* 6.1 runs to chown 32 system call, the abnormal can be detected by our model because the attribute is not satisfied with the following arguments policy:

$$\text{chown32}(\text{Cont}_{\text{chown32}}, \text{wfname}) \quad \text{equal} \quad \text{open}(\text{Cont}_{\text{open}}, \text{wfname})$$

$r_{SA_{chown32}} = 1$, so the abnormal value of chown 32 system call is 1 which is larger than. The software behavior deviates from the normal trace and is untrustworthy.

Attack 2: Some statements without system calls are added between stat64 and chown32 system call. In our experiments, vi 6.1 runs multiple times in the same environment. Table 2 shows a class of time intervals for each system call of above vulnerability window after clustering. The two abnormal traces listed in Table 2 are two examples of modification which executing two different codes between stat64 and chown32 system call.

For two normal traces and two abnormal traces in Table 2, the deviation degrees of system calls in system call sequence for v = 5 and v = 7 are shown in Table 3. For normal system calls, their deviation degrees are all less than or equal to 0.25 and most of them are 0. While, for abnormal chown32 system call, the deviation degrees are all greater than or equal to 0.5 which can be distinguished obviously. It is more sensitive to the variation of time interval for v = 7.

Let's take v = 7 as an example. For abnormal trace 1, $r_{\Pi_{chown32}} = 0.5$, so the deviation degree of chown32 is 0.5 ($\geq \tau$). For abnormal trace 2, $r_{\Pi_{chown32}} = 0.75$, so the deviation degree of chown32 is 0.75 ($\geq \tau$). Therefore, the deviation of software behavior for two abnormal traces can be detected accurately and the software behavior is untrustworthy.

Comparison of detection ability: We compare our model to some traditional software behavior models based on system call. System call time sequence model (Jones and Li, 2001; Pu and Lang, 2007) cannot detect the deviation of system call context, system call arguments and time interval for different program context or different software behavior trace between adjacent system calls. HPDA model (Liu *et al*., 2005) is helpless to monitor the software behavior between adjacent system calls. Our model can detect deviations according to system calls, such as system call name, system context, system call arguments and can decide whether to deviate from normal software behavior between adjacent system calls obviously including the deviation of time intervals for the same program context, different program context or different behavior trace between adjacent system calls. Our model's detection ability of software behavior is better than traditional models.

CONCLUSIONS

An improved software behavior model in system call level is proposed according to the deficiency of traditional software behavior model based on system call. Several

system call attributes are introduced and their deviation degrees are taken into account comprehensively for more overall and more accurate software behavior monitoring. The software behavior between adjacent system calls which cannot be monitored just through system call is monitored through time intervals. Our model's detection ability of software behavior is better than traditional models obviously. Moreover, our model has good extensibility, that is, new system call attributes can be introduced to our model conveniently. However, there is also some limitation. Our model cannot detect the deviation of software behavior with smaller time interval deviation. The research on whether the software behavior deviates from the normal one is very significant for applications such as intrusion detection, decision of software version update, etc.

ACKNOWLEDGMENT

This study is supported by the National Natural Science Foundation of China (60873203), the Foundation of Key Laboratory of Aerospace Information Security and Trusted Computing Ministry of Education (AISTC2009_03), the Outstanding Youth Foundation of Hebei Province (F2010000317), the Natural Science Foundation of Hebei Province (F2010000319, F2011201039).

REFERENCES

Chen, H.W., J. Wang and W. Dong, 2003. High confidence software engineering technologies. Acta Electron. Sin., 31: 1933-1938.

Feng, H.H., O.M. Kolesnikov, P. Fogla, W. Lee and W. Gong, 2003. Anomaly detection using call stack information. Proceedings of the IEEE Symposium on Security and Privacy, May 11-14, Berkeley, CA., pp: 62-76.

Forrest, S., S.A. Hofmeyr, A. Somayaji and T.A. Longstaff, 1996. A sense of self for Unix processes. Proceedings of the IEEE Symposium on Securiry and Privacy, May 6-8, Oakland, CA. USA., pp: 120-128.

Frossi, A., F. Maggi, G.L. Rizzo and S. Zanero, 2009. Selecting and improving system call models for anomaly detection. Proceedings of the 6th Detection of Intrusions and Malware and Vulnerability Assessment, (DIMVA'09), Springer-Verlag, Berlin, Heidelberg, Germany, pp: 206.

Jones, A. and S. Li, 2001. Temporal signatures for intrusion detection. Proceedings of 17th Annual Computer Security Applications Conference, Dec. 10-14, New Orleans, LA., USA., pp: 252-261.

Lin, J., E. Keogh, S. Lonardi and B. Chiu, 2003. A symbolic representation of time series, with implications for streaming algorithms. Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, June 13, ACM Press, San Diego, CA., USA., pp: 2-11.

Lin, J., E. Keogh, L. Wei and S. Lonardi, 2007. Experiencing SAX: A novel symbolic representation of time series. Data Mining Knowledge Discov., 15: 107-144.

Liu, Z., S.M. Bridges and R.B. Vaughn, 2005. Combining static analysis and dynamic learning to build accurate intrusion detection models. Proceedings of the 3rd IEEE International Workshop on Information Assurance, March 23-24, College Park, MD., USA., pp: 164-177.

Man, J.F., L.M. Yang, C.Y. Li and Z.C. Wen, 2009. Research on trust analysis and trend prediction of software interactive behavior. J. Chin. Comput. Syst., 30: 2003-2009.

Pu, S. and B. Lang, 2007. An intrusion detection method based on system call temporal serial analysis. Proceedings of the 3rd International Conference on Intelligent Computing, (ICIC'07), Springer-Verlag, Berlin, Germany, pp: 656.

Shanlin, Y., D. Shuai and C. Wei, 2009. Trustworthy software evaluation using utility based evidence theory. J. Comput. Res. Dev., 46: 1152-1159.

Wagner, D. and D. Dean, 2001. Intrusion detection via static analysis. Proceedings of the IEEE Symposium on Security and Privacy, May 14-16, Oakland, CA., USA., pp: 156-168.

Wang, H., Y. Tang, G. Yi and L. Li, 2006. The trustworthiness mechanism of network software. Sci. China (Series E): Inform. Sci., 36: 1-14 (In Chinese).

Wang, H.C. and H. Peng, 2007. A clustering algorithm based on entropy. Comput. Sci., 34: 178-180.

Wei, J. and C. Pu, 2005. TOCTTOU vulnerabilities in UNIX-style file systems: An anatomical study. Proceedings of the 4th USENIX Conference on File and Storage Technologies, Dec. 15, San Francisco, CA., USA., pp: 155-167.

Wepsi, A., M. Dacier and H. Debar, 2000. Intrusion detection using variable-length audit trail patterns. Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection, Oct. 2-4, Toulouse, France, pp: 110-129.

Zhang, H., F. Shu, Y. Yang, X. Wang and Q. Wang, 2010. A fuzzy-based method of evaluating the trustworthiness of software processes. Proceedings of the International Conference on Software Process, July 8-9, Paderborn, Germany, pp: 297-308.