Design and Implementation of Trusted Plug-in Based on Kylin Operating System Platform

Yong Gao\textsuperscript{1}, and Xuejun Yu\textsuperscript{1,*}

\textsuperscript{1} Faculty of Information Technology, Beijing University of Technology, Beijing, 100124, China

\textsuperscript{*}Corresponding author’s e-mail: yuxuejun@bjut.edu.cn

Abstract. Software behavior research on behavioral statements is mostly software behavior analysis for specific software, and the characterization of software behavior is only obtained by simple buried point monitoring or static analysis. In this regard, this paper proposes a model for credibility analysis of most software behaviors based on the detection sandbox environment and the actual operating environment. The research developed a trusted plug-in based on the Kylin operating system platform. It adopted the idea of “consistent words and deeds”, and used the actual behavior statement and the trusted behavior in the trusted behavior statement database for tested evaluation. In the aspect of software behavior tested measurement, a behavior statement judgment model based on cosine similarity algorithm is proposed to judge the reliability of software behavior. Experimental results: the trusted plug-in platform was able to perform credibility analysis on most software behaviors, and the introduced performance overhead was relatively low. It had a high ability to identify the credibility of software behaviors. It provides new ideas for studying the credibility of software behavior on the Kylin operating system platforms.

1. Introduction

Today’s world Internet technology is in the era of rapid development. The development of big data, cloud computing, artificial intelligence, machine learning, and the Internet of Things has promoted the improvement and progress of Internet technology in the world. In 2014, Microsoft closed down update and maintenance of Windows XP, which led to the security threat of system vulnerability for Chinese Windows XP users. In this regard, the Kylin operating system combined with the actual situation provides a practical feasibility plan, insisting on a smooth replacement of its system within a controllable range. This method ensures that national information security issues are resolved [1]. The Kylin operating system is an independently controllable security operating system developed in China. At present, it and its corresponding application ecosystem have been successfully used in defense, aerospace, government, power, and finance, energy, education and other industries, it will promote the vigorous development of China’s information industry and application ecosystem. In 2019, the 360 security brain issued a document indicating that the number of computers attacked by malware was as high as 4.125 million units, with an average daily attack volume of about 12,000 units. Therefore, for China’s government and enterprise departments, netizens, and even globally, the untrusted behavior of malware has become the most important security threat in the field of system security. In order to reduce the threat to the system, the credibility of the software in the operating system is getting more and more attention.
2. Related Research

At present, software behavior measurement is divided into static measurement and dynamic measurement. For the diversification of software code structure, program architecture, and writing language, there are many malwares that use some packaging or encryption technologies to evade security checks, which makes some software unable to perform complete static metric analysis [2]. For statically implanted monitoring code, there are also some incomplete behavior paths, and there are also problems with source code analysis of software. Most software source code is not available. In the dynamic measurement of software behavior, software may have untrusted behavior. The analysis process mostly relies on sandbox environment and some simulation technologies to simulate a real and reliable virtual environment [3]. With the emergence of various new technologies and the diversity of programming languages, software code structure, program architecture, and software functions have become more and more diverse. At present, there are many malwares that perform anti-sandbox operations against untrusted behavior in the sandbox environment - In the sandbox environment, the software performs normal behavior, but in the actual operation (non-sandbox environment), it will perform abnormal behavior, resulting in a very low detection rate of sandbox, resulting in system threats and security issues such as user data information disclosure [4].

Using the idea of “consistent words and deeds” to study the credibility of software, the key issue is how to define a structure of trusted behavior statement. The acquisition of existing behavior statements is mainly divided into two categories. The definition of the first type of behavior statement is to obtain the trusted requirements through the analysis of requirements analysis in the entire life cycle of software development. Finally, the definition of the behavior statement is given [5]. This method has certain limitations and poor applicability to software. In the face of non-participation in the entire software development process and no requirements documents, it is impossible to determine what kind of software behavior is difficult to implement the definition of behavior statement. The definition of the second type of behavior statement is to perform buried point monitoring or dynamic implantation of monitoring code during the software running process, and to integrate and analyze data through the attributes in the set behavior statement. Xiao et al. used the dynamically embedded monitoring code to obtain behavior statements, set monitoring points before the software runs, trigger monitoring points to report data and generate behavior statements when the software runs [6]. The monitoring codes implanted in this way will also some cases where the behavior path is not completely covered, and there are some differences in the monitoring points for different software to set the monitoring point. There is different embedded point monitoring with software. Because software writing languages and grammars are different, there are also differences in the acquisition of monitoring data for different software or using different programming languages. The software under study has certain limitations and cannot be applied to all software.

The operation of software in the operating system needs to call the kernel functions of the system for data interaction to achieve the purpose of executing software behavior. By studying the system user mode and kernel mode, the system call sequence can reflect the behavior characteristics of the software to a certain extent. Forrest first proposed a software behavior model N-gram based on a short sequence of system calls [7]. This model has certain defects. The model does not reflect its detection ability for system calls of indefinite length, and the behavior of actual software is different. System calls the length of the sequence is difficult to determine. Subsequently, in order to be able to characterize the indefinite-length sequence, Andreas et al. proposed a non-fixed-length system call sequence to characterize the software behavior to overcome the shortcomings of the fixed-length sequences [8]. Cai et al. classified system calls to construct system call vectors, and established a model for dynamically measuring software behavior of non-fixed-length system call sequences [9]. As the number of system call sequences increases, the efficiency of metric matching becomes slower. Li et al. proposed that the time interval attribute of system calls could improve the deviating ability of monitoring and characterizing software behavior trajectory to a certain extent, but it ignores the study of system call frequency, and should focus on monitoring system calls with high frequency [10]. Chen
et al. collected and processed system call information to build an integrated classifier to distinguish malicious programs from normal programs [11].

In view of the above problems, the credibility of software behaviors was studied in the Kylin operating system environment, and a trusted plug-in platform was constructed to realize the dynamic measurement of software behaviors. In the sandbox environment, software behaviors were divided into normal software behaviors and software that contains malware behaviors that could be detected by the sandbox, as well as untrusted behaviors in anti-sandbox detection malware. This had certain applicability for studying the behavior of application software installed in the operating system. Using normal program samples and abnormal program samples to test the function of the trusted plug-in platform, it adopted the “consistent words and deeds” idea to study the software behavior. By adding the obtained non-fixed-length system call sequence and calling frequency to the behavior in the statement, a credible behavior statement was constructed in a sandbox environment, and an actual behavior statement was constructed in an actual operating environment, then the cosine similarity algorithm model of the behavior statement could be used to determine the credibility of the software behavior.

3. Design of Trusted Plug-in Platform

The overall architecture of the trusted plug-in platform shown in figure 1. For the Kylin system, it is developed based on the microkernel and has a certain level of kernel structure, so it will not have the characteristics of frequent context switching and large system maintenance. It designed and implemented a trusted plug-in platform, and the trusted plug-in consists of a monitoring module (application environment monitoring and system call sequence monitoring), an information processing module, a trusted behavior statement library module, and a credibility analysis module. The sandbox has good independence and isolation, it can build an independent operating environment, and the programs running internally will not affect the hard disk. In the first stage, the monitoring module obtained system call sequence data and environmental data of the software running process in the sandbox environment. Then the information processing module performs processed, it was considered untrusted for unauthorized access or access to sensitive information during sandbox operation. Artificial analysis of environmental data excluded data with obvious problems and constructed an environmental data set to facilitate credibility analysis of environmental data. When the data information was correct, a trusted software behavior statement was formed, and then the software trusted behavior was directly added the software’s trusted behavior statement library. The second phase was that the data obtained through the monitoring module in the actual operating environment passes through the information processing module to generate the actual behavior statement. If there was unauthorized access and sensitive information access, it was regarded as untrusted behavior. The third phase passed the credibility analysis module dynamically measures the credibility of software behavior.
4. Trusted Plug-in Platform Specific Implementation

4.1. Design and implementation of monitoring module

4.1.1. Implementation of environmental monitoring module. The application environment monitoring module was written in the Java language. This module aims to obtain operating system information, memory information, network information, process information, etc. The required information cannot be completely obtained through the Java API, and the corresponding dependency package needs to be implemented. The function was used to obtain the required information. After obtaining the information, the module used the process number to classify, generated the corresponding data and the process number into a corresponding file, and passed it to the subsequent modules for further analysis.

4.1.2. System call monitoring module implementation. The kernel space in the Kylin operating system is to provide hardware programming and specific interfaces to interact with hardware devices. System calls are the bridge between user programs and the system kernel. The specific system call functions were implemented in the kernel subsystem. By viewing the Kylin system /usr/include/asm-generic/unixcl.h system function prototype files, the Kylin system provided 326 system calls, such as process management fork(), clone(), execve(), etc. Call mkdir(), open(), write(), umask(), etc. The Kylin system also defined the system call address table sys_call_table in the system kernel like the Linux system, and also assigned a unique identifier system call ID. Then it interrupted the mechanism transfers control from system user space from kernel space to system calls. First, it determined whether the system call was legal. If it was legal, it determined the corresponding system call number. It looked for the address from the system call table to find the trapped address, and carried out the corresponding kernel functions.

The behavior of the software is jointly completed by a series of system calls. After constructing KylinStrace in the Kylin system to obtain the software behavior system call sequence, KylinStrace obtained system calls for software behavior without modifying the system kernel and without disturbing the monitoring software. Then it will load the node information into the kernel, and deleted it immediately after collecting the data. KylinStrace set some data trace points such as system scheduling, file system operations, and software process information acquisition. KylinStrace relied on the kernel configuration macro to read the control information from it, obtains the system call sequence according to the trace of software process and subprocess by ptrace, sends the ptrace -
attach command to Galaxy Unicorn System through ptrace, requests the trace of software process, if the request obtains the execution right through kylinstrate.

KylinStrace relied on kernel configuration macros to read the control information, obtained system call sequence according to the trace of software process and subprocess by Ptrace, and sent the PTRACE_ATTACH command to the Kylin system through Ptrace to request the tracking of the software process. If the request passes, KylinStrace will get the execution right. Capturing the system call is a cyclic process waiting for the process to enter the next system call. Recording the system call number allows the system call to execute and wait for the return result. It can view the system call number stored in the register in the traced process. The stored system call number is passed through the register rax. If there is a child process in the process, it could use PTRACE_TRACEME to track system calls in the child process, read the system call number in the register, and count the corresponding system call through the count function. It could use system call number to record and restore system call information. According to the get_name (int n) function in the custom syscalls.h, it restored the system call number to the system call function name. It could use the pstree command to obtain the system process tree and generate software the process tree file. The information processing module and the behavior statement library combined information of each pid system call and the process tree information file to reorganize the information according to the trusted behavior statement structure.

4.2. Information processing module generates behavior statement
When executing software behaviors, the system assigns corresponding process schedules to software behaviors. Software behaviors and processes are not in a one-to-one relationship. A software behavior can be scheduled by multiple different processes to complete operations. There is a big difference between the normal software behavior call sequence and the abnormal untrusted system call sequence when a software process is running [12]. After the monitoring module, the environmental monitoring data can monitor the software process information and static data of memory information, network information, and file system information. The system call monitoring module monitored the behavior calling sequence and corresponding calling frequency of specific software, and describes the system call sequence of the process. The dynamic monitoring module can monitor the system call information of the processes in the system. The information processing module analyzed the process numbers of the parent process and the child process according to the information in the process tree. Then it sequentially analyzed and reorganized the system call information in the process and child process files. Finally, it generated a behavior declaration text with a certain structure.

There is also a certain permission mechanism for the system call. The systematic call first passed the permission protection mechanism. If the permission was insufficient, the addressing operation in the system call table would not be performed, the parameter operation would not be passed, and the specific type would not be returned. If it failed, it would return -1. The error code information was stored in errno. The information processing module would identify some software’s untrusted behaviors. It has tentatively determined that software-specific behavior in a sandbox environment is considered to be untrusted if there is an error code in the calling sequence, it records the statistics of untrusted system calls in noPermissionNum. When noPermissionNum is not 0, the system call returns that there is an error. The system kernel function is not completed normally, the system call is recorded in the untrusted behavior. If there is abnormal information in the static measurement environment information, such as abnormal memory allocation information, and unlimited applications for excessive memory occupation will also be stored in untrusted behaviors. It eliminated non-conforming data to ensure the credibility of the data in the trusted behavior statement database. In other cases, the behavior statement of software behavior in a sandbox environment was stored in a specific software trusted behavior statement library. The above is the acquisition of “words”, and the same operation was performed for the acquisition of “deeds” in the actual operating environment.

The trusted plug-in uses the idea of “consistent words and deeds” to perform credibility analysis. The definition of the behavior statement includes the path of the software running, the behavior name,
the user, the memory allocation size, the used memory, the process id, the main process system call sequence, the system call sequence table of the child process, and the environmental monitoring data. The trusted behavior statement library module is to monitor as many different behaviors of a software as possible, and obtain a specific system call sequence to generate the behavior statement through the information processing module.

It established a behavior statement library for each software, and stored the trusted behavior statements of different software behaviors obtained in the behavior statement library, which was convenient to compare and analyze the actual software behavior and the trusted behavior statements of the same software behavior in the behavior statement library. The credibility criterion is that the system call sequence information is correct in the sandbox environment and the actual running environment, and there is a certain error range for the allowed call frequency. The Kylin system call is similar to Linux, and there are many system calls. It is not a random combination. Some system calls can execute the kernel implementation function of the system call only after there is specific system call shared data. If the kernel state is not shared through the specific system call, the system call cannot be executed. The Kylin system and the Linux system are similar to system resources as file operations. System call sequence can be a combination of system calls according to different operations. It supposed that there are two system calls, their relationship could be sequential structure execution, branch structure, that is, there is a prerequisite execution when the system call is executed, and the loop structure system call requires multiple executions to complete the operation [9]. In the actual process, there is a certain randomness in the acquisition of the system call sequence, because the above three structures exist between system calls. There is a certain order difference in the sequence of system calls for the same software behavior. If it was a normal program, their system call kernel functions were consistent, allowing certain errors in frequency. Figure 2 shows the trusted behavior statement and the actual behavior statement of browser browsing web page. The trusted judgment index is that the information obtained by the environmental monitoring module in the obtained behavior statement allowing a certain error. Within the error tolerance range, it should also ensure that the system call sequence is for the same software behavior. The system call sequence was accurately represented on the process, then the software’s behavior could be determined to be trusted. For the untrusted behavior determination standard, first it need to determine that the data obtained by the environmental monitoring module is within the normal range. For example, the obtained network information is one piece of networked monitoring data and other piece of data obtained from non-networked monitoring, then the credibility analysis of the data obtained by the two pieces cannot be performed. Through the analysis of the data of a large number of environmental monitoring modules, the behavior is considered untrusted if the obtained data is not within the normal range. If the system call function is inconsistent in the behavior statement of the process using the system call sequence, the behavior is considered as untrusted.
There will be significant

4.3. Behavior analysis trusted analysis module

The trusted analysis module is a quantitative analysis of the credibility of software behaviors, including information measurement of environmental data and system call sequence information measurement.

The premise of environmental information measurement is the collection and processing of environmental monitoring data. The problematic data in the environmental monitoring data is eliminated to ensure the correctness of the environmental dataset data. Because the actual operating environment in the system is quite complex, as long as there is no obvious network information, irrational memory allocation, etc., so there may be some errors.

The premise of the system call sequence measurement is to obtain the system call sequence information multiple times, and to select the information of the system call frequency that stabilizes the call sequence. If there is untrusted behavior in the trusted behavior statement and the actual behavior statement, it will get different System call sequence information. There will be significant differences in the system call sequence for untrusted behavior in the experiment, and the effect of the call frequency on the trusted behavior metric is less than the impact of the new system call sequence on the trusted behavior metric.

The trusted analysis module analyzes the credibility of software behavior by constructing a mathematical model of the trusted attributes concerned in the behavior statement. The environmental data uses the SPSS to perform data similarity analysis, and the system call sequence is based on the cosine similarity algorithm’s behavior statement decision model for similarity comparison. The Cosine Similarity algorithm uses the cosine value of the angle between the two vectors in space as a measure
of the similarity of the individuals represented by the vector. Relative Euclidean according to the
distance measure, the algorithm pays more attention to the difference in direction of the vectors.

The cosine similarity algorithm uses the following formula (1) to express the similarity in
rectangular coordinates, which represents the cosine of the angle between the vector A and the vector
B. In the trigonometric function, the cosine value of the angle ranges from [-1,1]. The closer the cosine
value is to 1. The similarity of the two vectors is higher, the similarity is lower.

\[
\text{Sim} (A,B) = \cos (\theta) = \frac{A \cdot B}{||A|| \times ||B||},
\]

The cosine value between two vectors can be obtained in a rectangular coordinate system, and the
cosine value can also be obtained in three-dimensional space using the knowledge of the space vector.
The cosine similarity algorithm can also be applied to high-dimensional vectors as formula (2).

\[
\text{Sim} (A,B) = \cos (\theta) = \frac{\sum_{i=1}^{n} (A_i \times B_i)}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}.
\]

This experiment credibility analysis module mainly performs software behavior credibility analysis,
constructs the space vector of the system call sequence, determines the credibility of the system call
sequence based on the behavior declaration model of the cosine similarity algorithm, and builds an
environmental data set to determine using the SPSS similarity of environmental data. The trusted
measure of software behavior is represented by the trusted value of the system call sequence and the
trusted value of the environmental data. The construction process of its mathematical model is as
formula (3).

Mathematical representations of behavior statements and their environmental attributes and
software behavior call sequences, as well as trusted statement behavior libraries. The specific
expression is as formula (3).

\[
R_{\text{software}} = \{B(e,S) | B_1, B_2, \cdots, B_s\}
\]

\[
\text{Tru}(e) = \begin{cases} 
1 & e \in E \\
0 & e \notin E
\end{cases}
\]

Only when the environmental data is judged to be trusted can the credibility of the system call
sequence be compared. First, the system sequence in the trusted software behavior statement and the
actual behavior statement is sequentially converted into \(S\), trusted statement call sequence collection
and \(S\), the actual behavior declares a collection of calling sequences.
It obtain the union $S_{all} = S_1 \cup S_2$ of the trusted statement call sequence and the actual behavior statement, and then sequentially encode the system calls in the union to get $S_{code_{all}} = \{s_1:0,s_2:1,\cdots,s_{n-1}\}$.

Then, according to the $S_{code_{all}}$ obtained by encoding the system call sequence, number conversion is performed on $S_1$ and $S_2$, and the system call with the same name as the system call in $S_{code_{all}}$ is digitally converted to $S_{code_1}$ and $S_{code_2}$.

According to the sequence of the collection of system call sequences, $S_{code_1}$ and $S_{code_2}$ are converted into vectorized statistics of system call times, and $S_{codeVector_1}$ and $S_{codeVector_2}$ are obtained.

It used the cosine similarity algorithm to represent the high-dimensional space vector to obtain the system call sequence $S_1$ and $S_2$, system call sequence trusted value $Tru(S_1,S_2)$ as formula (5):

$$Tru(S_1,S_2) = Sim(S_{codeVector_1},S_{codeVector_2}) = \frac{\sum_{i=1}^{n} (S_{codeVector_1} \times S_{codeVector_2})}{\sqrt{\sum_{i=1}^{n} (S_{codeVector_1})^2 \times \sum_{i=1}^{n} (S_{codeVector_2})^2}}$$

The credibility of software behavior $Tru(B)$. The credibility value is the average value of the runtime environment credibility value and the credibility of the system call sequence as shown in the following formula (6). This formula is only when the system call sequence is credible. The software behavior credibility is $Tru(B)$. Because this model mainly studies the credibility of the system call sequence, the credibility of the environmental data is ignored, and the credibility of the software behavior is approximately equal to the credibility of the systemcall sequence $Tru(B) = Tru(S_1,S_2)$.

$$Tru(B) = (Tru(e) + Tru(S_1,S_2))/2$$

5. Experiments and Results Analysis

This article implemented a trusted plug-in platform based on the Kylin operating system and tests the platform functions. The system function test is mainly aimed at the system call sequence and environmental data acquisition and measurement analysis proposed in this article. The hardware of this experimental environment Configuration: Kylin operating system, Intel (R) Core (TM) i5-8250U with 1.80GHz CPU, 8.00GB of memory, and sandbox environment. The test software included normal software, abnormal software that could be detected by the sandbox and anti-sandbox detection software. Normal software selected the behavior of browsers and music players for monitoring, and found malware samples “tsuna” that could be detected by the sandbox in the sample library. In order to simulate the anti-sandbox software to implement the functional test, a malicious sample program “anti-sandbox” was written in Java. The software existed to determine whether the process includes a sandbox environment. The software function included normally generate, read and write files in a sandbox environment, but the software also uploaded text data in the actual operating environment. Through the above three kinds of software to test the function of trusted plug-in platform.

5.1. System call sequence credibility test

For the above software to be tested, the software behavior system call sequence was acquired, and the monitoring software behaviors were browsing the web, playing local songs, tsuna behavior, and anti-sandbox behavior. Obtaining the system call sequence requires repeated experiments to view the system call information whether it is accurate. Then, the cosine similarity algorithm represented by the system call sequence was used for credibility analysis and repeated credibility comparison experiments. Six experimental results were selected for each behavior, as shown in Table 1. From this table, it can know that for normal software the credibility of the system call sequence of behaviors is above 0.9. The credibility of tsuna’s behavior system call sequences is 0, and the software can detect
the existence of sensitive data access and does not participate in the credibility analysis of the system call information. There was a big difference in the behavior of anti-sandbox's system call sequence information, that was, an unexpected system call sequence had occurred, and the credibility was between 0.57-0.69.

Table 1. Six groups of reliability of software system call sequence in the experiment.

| Action       | Test1 | Test2 | Test3 | Test4 | Test5 | Test6 |
|--------------|-------|-------|-------|-------|-------|-------|
| browse the web | 0.91  | 0.90  | 0.96  | 0.99  | 0.95  | 0.97  |
| play music    | 0.96  | 0.98  | 0.93  | 0.97  | 0.91  | 0.95  |
| tsuna         | 0     | 0     | 0     | 0     | 0     | 0     |
| anti-sandbox  | 0.57  | 0.62  | 0.65  | 0.64  | 0.69  | 0.62  |

5.2. Environmental monitoring module credibility test

For the acquisition of environmental monitoring information data, a large number of experiments are used to eliminate untrusted data, and in the sandbox environment, trusted environmental data is obtained and stored in the data set. The environmental data of the above 6 groups of behaviors were also obtained for actual operating environmental data. Using the SPSS to perform similarity analysis of environmental data, according to the environmental data judgment model, the similarity is 1 and the dissimilarity is 0, which were shown in Table 2.

Table 2. Credibility of 6 sets of environmental data in the experiment.

| Action       | Test1 | Test2 | Test3 | Test4 | Test5 | Test6 |
|--------------|-------|-------|-------|-------|-------|-------|
| browse the web | 1     | 1     | 1     | 0     | 1     | 1     |
| play music    | 1     | 0     | 1     | 1     | 1     | 1     |
| tsuna         | 1     | 0     | 1     | 1     | 0     | 1     |
| anti-sandbox  | 1     | 1     | 1     | 1     | 1     | 1     |

5.3. Software behavior credibility analysis

Finally, the definition of software behavior credibility in the constructed mathematical model was used to calculate the software behavior credibility, which was shown in Table 3.

Table 3. Credibility of software behavior in experiments.

| Action       | Test1 | Test2 | Test3 | Test4 | Test5 | Test6 |
|--------------|-------|-------|-------|-------|-------|-------|
| browse the web | 0.96  | 0.95  | 0.98  | 0.50  | 0.98  | 0.99  |
| play music    | 0.98  | 0.49  | 0.97  | 0.99  | 0.96  | 0.98  |
| tsuna         | 0     | 0     | 0     | 0     | 0     | 0     |
| anti-sandbox  | 0.57  | 0.62  | 0.65  | 0.64  | 0.69  | 0.62  |

From the data shown in Table 2, it could be seen that the behavior credibility of the malicious samples monitored by the sandbox does not directly obtain the software behavior statement in the actual behavior statement. In the analysis module, no analysis is performed, and the software behavior credibility is defined as 0. In normal software behavior, if the environmental data is untrusted, it will directly lead to untrusted software behavior. This kind of data has certain contingencies, but it is still credible for normal software behavior in the case of repeated experiments. The credibility of the software behavior in anti-sandbox software is approximately equal to the credibility of the system call sequence. The credibility of the software behavior cannot be calculated by using the credible environment metric and the credibility of the system call sequence. The credibility of the data is higher than the system call sequence, which is not in line with the actual situation. Because the reliability of the system call is studied in this article, the credibility of the environmental data is only available
when the system call sequence is trusted. For the case where the system call information is not trusted, the credibility of the environmental data is not considered.

6. Summary
This article built a trusted plug-in platform in the Kylin operating system to conduct credibility analysis of the software behavior, and implemented a data monitoring module for the Kylin operating system. This module generated a behavior statement file with a certain structure by monitoring the process system call sequence information and environmental data. It utilized the “consistent words and deeds” to analyze the credibility of the software, built a mathematical model, and used the behavior statement determination model based on the cosine similarity algorithm to determine the credibility of software behavior. Experimental results showed that: effective testing of different software had certain identification ability for the credibility of software behavior.

References
[1] Z. X. Zhang, “Use and promotion of Kirin operating system,” Beijing:Science Press, 2014, pp. 2-3.
[2] Zolkipli, M. Fadli, and A. Jantan, "A Framework for Defining Malware Behavior Using Run Time Analysis and Resource Monitoring." 2011.
[3] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," ACM COMPUTING SURVEYS, vol. 44, 2012, pp. 1-42.
[4] J. Zhang, Y. Wang, and Z. Xue, "Research on sandbox detection method against malicious code based on angr," computer application and software, vol. 36. 2019, pp. 314-320.
[5] H. G. Lu, “Research on software credibility testing method based on behavior statement," Beijing University of Technology, 2016, pp. 1-30.
[6] X. J. Yu, R. Xiao, "Research on credibility testing method and credibility calculation based on behavior statement," Application of Computer Systems, vol. 27. 2018, pp. 17-26.
[7] R. Chinchani, E. V. D. Berg, “A Fast Static Analysis Approach to Detect Exploit Code Inside Network Flows,” Recent Advances in Intrusion Detection, 8th International Symposium, RAID 2005, Seattle, WA, USA, September 7-9, 2005, Revised Papers. Springer, Berlin, Heidelberg, 2005.
[8] A. Wespi, M. Dacier, and H. Debar, “Intrusion detection using variable-length audit trail patterns,” Lecture Notes in Computer Science, 2000, pp. 110-129.
[9] H. B. Cai, Z. Shan, C. Fan, B. L. Zhao, “Dynamic measurement method of program behavior based on non-fixed-length system call sequence,” Application Research of Computers, vol. 33, 2016, pp. 1154-1158.
[10] Z. Li, J. F. Tian, X. H. Yang, “Program behavior monitoring based on system call attributes,” Computer Research and Development, vol. 49. 2012, pp. 1676-1684.
[11] H. Chen, H. T. Jiang, J. Guo, et al., “Android malicious application detection method based on system call,” Journal of Nanjing University of Science and Technology, vol. 41. 2017, pp. 720-724.
[12] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. “A Sense of Self for Unix Process,” Proceedings of the 1996 IEEE Symposium on Security and Privacy, 1996, pp. 120-128.