Linux Auditing: Overhead and Adaptation

Lei Zeng, Yang Xiao*
Department of Computer Science,
The University of Alabama,
Tuscaloosa, AL 35487-0290 USA
Email: yangxiao@ieee.org
*Prof. Yang Xiao is the corresponding author

Hui Chen
Department of Mathematics and
Computer Science,
Virginia State University,
Petersburg, VA 23806 USA

***Abstract*— Logging is a critical component of Linux auditing. The experiments indicate that the logging overhead can be significant. The paper aims to leverage the performance overhead introduced by Linux audit framework under various usage patterns. The study on the problem leads an adaptive audit logging mechanism. The adaptive auditing mechanism reduces the overall system overhead and achieves a similar level of protection on the system and network security.**

Index items: logging, overhead, Linux, auditing;

I. Introduction

AUDITING in operating systems (OSs) is necessary to achieve network and system security. Linux auditing watches file accesses, monitors system calls, records commands run by users, and security events, such as authentication, authorization, and privilege elevation, which is achieved via logging the system activities to events. Attributes of an event include the date, time, type, subject identity, result, and sensitivity labels.

The book, The Trusted Computer System Evaluation Criteria, which was published by the U.S. Department of Defense in 1985 [1] defines the requirement of logging for auditing purpose. The criteria is often referred to as "the Orange Book," which has been regarded as the security requirements for the design and implementation of secure computer systems, and has a profound influence on the design of secure computer systems. The Orange Book divides its security criteria into four categories, D, C, B, and A, which are organized in a hierarchical manner such that category D has the lowest security requirements while category A has highest ones. Category D stands for the minimal protection (i.e., the computer systems that cannot meet the requirements of the higher category). Category C sets forth the requirements for discretionary protection. Category B defines the requirements for mandatory protection. In category A, formal verification methods are required to assure that sensitive or classified data processed or stored by the system can be protected by discretionary and mandatory security controls. Categories B and C are further divided into a number of sub classes, organized hierarchically. For example, class C1 requires the separation user and data enforcement of access limitation to the data on an individual basis while a more finely-grained discretionary access control characterizes class C2 so that actions of users become accountable via auditing of security-related events, login procedures, and resource isolation

The Common Criteria is closely related to the Orange Book. Its goal is to establish a guideline for developing and evaluating computer products in regards to its security features. The Common Criteria concentrates on the security threads from human activities. It reiterates the audit requirement for secure computer systems. Logging is essential to meet the requirement. Although these standards recognize the importance of auditing, the focus is obviously placed in bookkeeping audit trail/log, which is in fact the system activity log.

In regard to the logging functionality of computer systems, audit information must be recorded and protected selectively [1, 2, 3, 4]. It is the logging functionality that keeps and protects the audit information, which is often referred to as audit trails, audit data, logs, or logging data. The criteria also indicates that identifications are needed for individual subjects and access information, such as identities of accessing information, and their authorization of accessing the information is mediated [1]. Information of identification and authorization must be saved in computers, protected from attractors, and used when performing some security-relevant actions [1]. Thus, the responsible party can be traced via its actions related to security, and the outcome of the actions can be assessed, and this is essentially based on the logging data recorded on non-volatile memory.

Linux audit framework [5] helps Linux meet many government and industrial security standards, such as CAPP/EAL4+ [6] ,LSPP [7], RBAC [8], NISPOM [9], FISMA [10], PCI [11, 12], and DCID 6/3 [13]. It is important to the adoption of Linux in mission critical environments to meet the requirements of the security standards; otherwise, Linux cannot compete with other commercial operating systems, such as IBM, AIX, and Windows Server. They have already received certification in many government and industrial security standards.

However, to the best of our knowledge, an important question remains open: what is the performance overhead induced by Linux audit framework under various traffic and usage patterns, would recent development and development of high throughput/bandwidth networks further stress Linux audit framework?

In this paper, we first identify the important usage patterns of Linux operating systems, and then, we design experiments to measure the overhead induced by the Linux audit framework in these usage patterns. The experiments inform the design of an adaptive auditing mechanism, which uses a set of selected but important type of events as the vital sign of

system and network activity and uses the vital signs to adjust audit logging In order to change the type of events logged, the frequency of events logged and the time window intensive logging must be performed. The adaption achieves a low overhead in normal uses and at the same time provides the same amount of audit data sufficiently to accomplish an audit during critical events, such as system and network intrusion.

The rest of the paper is organized as follows. In Section II, we introduce Linux audit framework. In Section III, we study overhead of Linux logging. We propose an adaptive mechanism and study the performance of the proposed adaptive logging in Section IV. Finally, we conclude the paper in Section V.

II. Linux Audit Framework

Linux audit provides users a way to analyze system activities in great detail [5]. It does not, however, protect users from attacks [5]. Instead, Linux audit helps users to track these issues and take security measures to prevent them [5].

Linux audit framework includes several components: *auditd*, *auditctl*, *audit rules*, *aureport*, *ausearch*, *audispd*, and *autrace*, as shown in Fig. 1 [5].

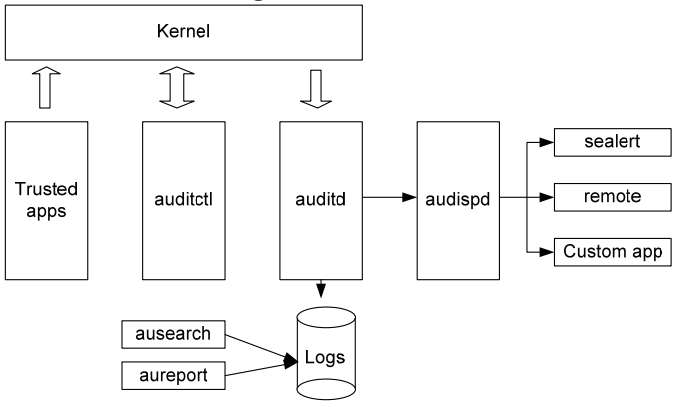

Fig. 1 Linux audit framework [5]

We explain the functions in Fig.1 as follows [5]:

- *Auditd*--The audit daemon writes the audit messages that collected in the Linux kernel to disk or transfers them to audispd. The configuration file, /etc/sysconfig/auditd, controls how the audit daemon starts, and the configuration file, /etc/auditd.conf, controls how the audit daemon functions once it starts.
- *Auditctl*--The auditctl utility is responsible for kernel settings, log generation parameters, and audit rules that determine which events are tracked.
- *Audit rules*--Audit rules are contained in the file /etc/audit.rules. The file is loaded when the system boots and starts audit daemon.
- *Aureport*--The aureport utility helps users to create a custom view of logged messages.
- *Ausearch*-- Users can use the ausearch utility to search some events with characteristics, such as keys, of the logged format in the log file.
- *Audispd*--The audit dispatcher daemon (audispd) dispatches event messages to other applications instead of writing them to a disk.

There are several keywords in an audit event record: type, msg, arch, syscall, success, exit, a0 to a3, items, ppid, pid, auid, uid, gid, euid, suid, fsuid, tty, comm., exe, subj, key, item, name, inode, dev, mode, and ouid. We explain them in details as follows [5]:

- The type field in the record denotes the type of events recorded.
- The msg field stores a message ID. The arch field refers to the CPU architecture.
- The syscall field stores the system call ID.
- The result of the system call, such as being invoked, success, or failure, is saved in the success field.
- The return value of the system call is saved in the exit field.
- The first four arguments to the system call are saved in the a0 to a3 field.
- The number of strings that is passed to the application is saved in the items field.
- Ppid, pid, auid, uid, gid, euid suid, fsuid, egid, sgid, and fsgid store the ID of the process, user, or group.
- The tty field refers to the terminal from which the application is initiated.
- The Comm field stores the application name appeared in the task list.
- The Exe field stores the pathname of the invoked program.
- The Subj field refers to the security context to which the process is subject.
- The key field stores the key strings assigned to each event.
- The item field stores the path argument of system call if there is more than one argument.
- As an argument, the pathname is saved in the name field.
- Inode refers to the inode number corresponding to the name field.
- The dev field stores the device and mode field stores the files access permission in numerical representation.
- The uid and gid of inode are saved in the ouid and ogid fields, respectively.

III. Overhead of Linux Audit Logging

A. Traffic Patterns and Types of Events

Linux distributions have been used as a server operating system for its stability, security, and pricing [14]. There are several types of servers in the general network environment: web server, mail server, file server, application server, catalog server, DNS server, data server, etc [15]**.** Moreover, Linux is a widely used platform for cloud computing, which provides storage service, software, data access, and computation, which users may not know, as well as configuration of the system and physical location that provide the services [16, 17, 18]. Besides, Linux distributions play a major role on scientific computing because of their efficiency [19] and stability. Last but not least, Linux can be used in workstation. Current

workstations uses sophisticated CPU such as Intel Xeon, AMD Opteron, or IBM Power and run Unix/Linux systems to provide reliable workhorse for computing-intensive tasks [20].

In addition, auditing is used widely in operating systems and Linux is used as a case study. More importantly, what we learned from this case study can be extended to auditing of other operating systems.

B. Experimental Design and Overhead Measurement

There are many free Linux benchmark tools available such as Phoronix Test Suite and Netperf [21]. The drawback is the benchmark is different from real situation [22].

Similar to the logging performance study, we will run a worker program (on the host, or from the network), measure wall-clock time, with/without audit logging, and with different granularity of audit logging.

We use a worker program, which invokes a number of system calls to simulate normal workload. The number of system calls and the frequency with which the worker program opens and closes files can be adjusted. In other words, the worker program simulates normal workload in a variety of usage patterns, such as web servers, file servers, and mail servers.

First, we run the worker program on a computer with Ubuntu 9.10 running, and we count the average running time (wall clock time). Then, we run the same worker program on the same computer with auditing function enabled, and we record the average running time as well. We denote the former average running time as Toff indicating that the auditing function is off and the later as Ton indicating that the auditing function is on. The performance overhead is a percentage expressed as $C = \left(T_{on} - T_{off}\right) / T_{off}$.

C. Performance Overhead Analysis of Linux Audit Logging

Configuration of the test and evaluation computer system is shown in Table 1.

TABLE 1 CONFIGURATION OF THE TEST AND EVALUATION COMPUTER SYSTEM

Component	Description
Linux distribution	Ubuntu 9.10
Motherboard	Dell 0G8310
Memory	1GB DIMM SDRAM PC 533 RAM
CPU	1 Intel(R) Pentium(R) 4 CPU 3.00GHz
Disk	82801FB/FW (ICH6/ICH6W) SATA Controller and 40GB WDC WD400BD-75JM

Since there are 347 system calls in arch/x86/kernel/syscall_table_32.S for x86 architecture, these primary system calls in [23] and [24] are configured to be audited. The performance overhead is shown in Fig. 2. In this figure, the X dimension denotes the action frequency in worker program from 1 times/s, 5 times/s, 10 times/s to 20 times/s. Because the action in worker program can only happen 23.8 times per second, the highest frequency recorded in Fig. 2 is 20 times/s. The Y dimension denotes the overhead that computed using $C = \left(T_{on} - T_{off}\right) / T_{off}$.

Significant performance overhead is observed when action frequency is 20 times/second. With the action frequency increasing, performance overhead increases as well. Since auditing these system calls incurred system overhead, the performance penalty will increase when the frequency of invoking system calls increases.

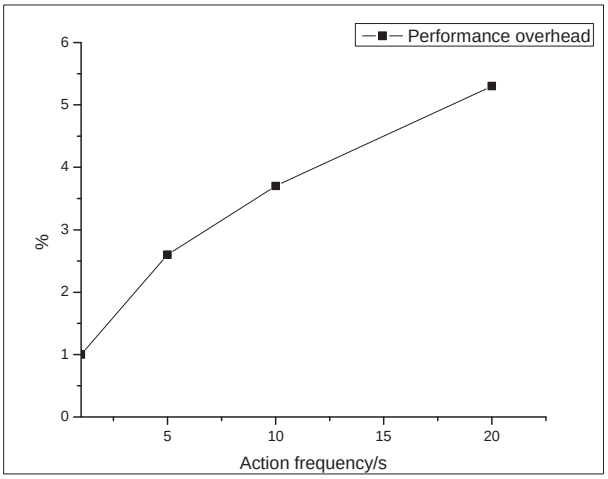

Fig. 2 Performance overhead when auditing 347 primary system calls

IV. ADAPTIVE LOGGING AND PERFORMANCE EVALUATION

A. Adaptive Auditing Mechanism

The study on the problem leads an adaptive audit logging mechanism. Many security incidents or other important events are often accompanied with precursory events. We identify important precursory events – the vital signs of system activity and the audit events that must be recorded. The adaptive auditing mechanism increases or reduces the type of events collected and the frequency of events collected based upon the online analysis of the vital sign events. The adaptive auditing mechanism reduces the overall system overhead and achieves a similar level of protection on the system and network security.

Two goals are 1) to identify critical events specific for Linux server traffic, 2) to estimate the performance overhead introduced by the auditing function.

Common tasks in the Linux server, uch as bash, aureport, crond and yum, etc., are shown in Fig. 3. These tasks will eventually invoke the connect system call or the accept system call, which are in fact are socket function calls. In order to adapt the auditing system to the server operating system and minimize the incurred system overhead, these two critical system calls, connect and accept, are chosen to be audited.

Operating systems are not actual user programs. The number of resources used by operating systems should be as small as possible. The auditing function would increase the resources used by the operating system. We would like to estimate the performance overhead introduced by the adaptive auditing function.

Audit rules in /etc/audit/audit.rules file are shown as

follows:

```
-a entry, always –S, connect
-a entry, always –S, accept
```

The first rule is to monitor the connect system call, and the second rule is to monitor the accept system call.

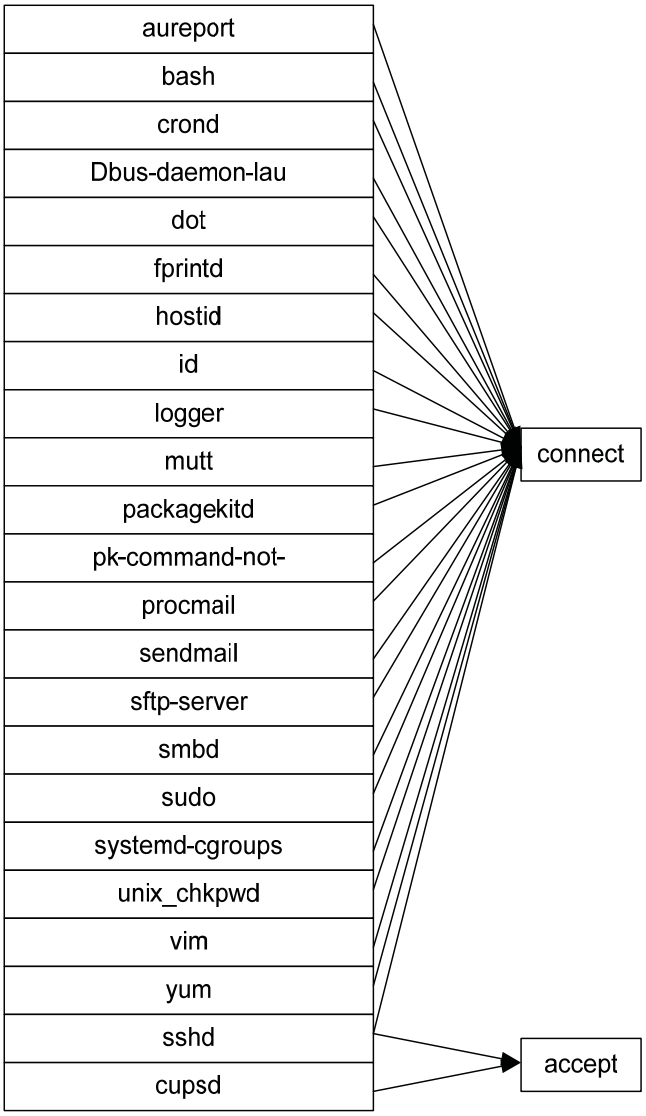

Fig. 3 Linux tasks and corresponding system calls

B. Performance Evaluation

This subsection shows that adaptive logging has reduced overhead.

Figs. 4, 5, and 6 are produced using aureport utility and two vital system calls, connect and accept, for web servers, are monitored.

The failed access statistics report is shown in Fig. 4. In the figure, /var/run/setrans/.setrans-unix is observed to have the highest failed access and /var/run/nscd/socket ranked the second. Because /var/run/setrans/.setrans-unix contains many system related parameters that are used by a variety of applications and because only the root user has access to this file, the other applications will fail to access the file if no root privilege is obtained. The third file in Fig. 4 is a log file owned by ssh utility and does not have a failed access recorded.

The system call statistics report is shown in Fig. 5. In this figure, two system calls, connect and accept, are recorded. The connect system call was invoked many times because a lot of the common tasks of server traffic will invoke this system call.

Event ranking is shown in Fig. 6. In the figure, the system call event ranked first among all events in the server operating system.

Performance overhead for adaptive logging is shown in Fig. 7. In this figure, the X dimension denotes the action frequency in the worker program from 1 times/s, 5 times/s, 10 times/s to 20 times/s. Because action in the worker program can only happen 23.8 times per second, the highest frequency recorded in Fig. 7 is 20 times/s. The Y dimension denotes the overhead computed using $C = \left(T_{on} - T_{off}\right) \Big/ T_{off}$.

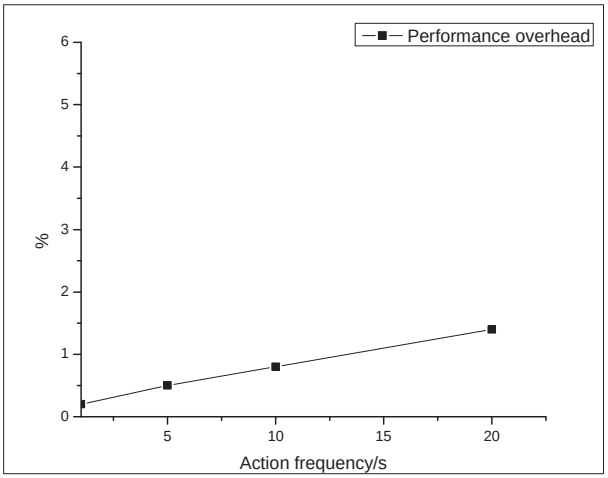

Fig. 7 Performance overhead

The performance overhead is observed as insignificant at a chosen frequency in the worker program, especially when compared with the previous experiment when all of the system calls are audited. This means that the performance overhead introduced by Linux audit function is acceptable when two system calls, connect and accept, are audited. In the case that all of the system calls are audited, the performance overhead will end up being unacceptable. Therefore, auditing should be adapted to different Linux security models to enhance the security with an acceptable, incurred overhead.

V. Conclusion

In this paper, we introduced Linux audit framework and measured system overhead when the auditing function was enabled. Then, we adapted the auditing function to the server traffic pattern and reevaluated system overhead. We observed that adaptive logging can dramatically reduce system overhead.

Acknowledgement

This work was supported partially by the Natural Science Foundation of China under grant #61374200.

REFERENCES

[1] US Department of Defense, “Department of Defense Trusted Computer System Evaluation Criteria,” DoD 5200.28-STD, Library No. S225, 711, December 1985.

[2] Y. Xiao, "Flow-Net Methodology for Accountability in Wireless Networks," IEEE Network, Vol. 23, No. 5, Sept./Oct. 2009, pp. 30-37.

[3] B. Fu and Y. Xiao, "Q-Accountable: A Overhead-based Quantifiable Accountability in Wireless Networks," Proceedings of IEEE Consumer Communications and Networking Conference (IEEE CCNC 2012), pp. 138-142.

[4] B. Fu and Y. Xiao, "Accountability and Q-Accountable Logging in Wireless Networks", Wireless Personal Communications, Vol. 75, No. 3, Apr. 2014, pp. 1715-1746.

[5] “SUSE Linux Enterprise—The Linux Audit Framework,” available: http://www.suse.com/documentation/sled10/pdfdoc/audit_sp2/audit_sp2.pdf, May 8, 2008

[6] “CAPP/EAL4+ compliant system overview,” see: http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.security/doc/security/capp_compliant_overview.htm.

[7] Information Systems Security Organization, "Labeled Security Protection Profile,” URL: http://www.commoncriteriaportal.org/files/ppfiles/lspp.pdf, October 8, 1999.

[8] Ravi S. Sandhu, Edward J. Coyne, “Role-Based Access Control Models,” URL: http://profsandhu.com/journals/computer/i94rbac(org).pdf, 1996, IEEE.

[9] “National Industrial Security Program—Operating Manual,” available: http://www.ncms-isp.org/NISPOM_200602_with_ISLs.pdf, February 2006.

[10] “Federal Information Security Management Act of 2002,” available: http://en.wikipedia.org/wiki/Federal_Information_Security_Management_Act_of_2002.

[11] “Conventional PCI,” available: http://en.wikipedia.org/wiki/Conventional_PCI.

[12] “Director of Central Intelligence Directive 6/3—Protecting Sensitive Compartmented Information Within Information Systems,” available: http://www.fas.org/irp/offdocs/DCID_6-3_20Manual.htm.

[13] J. Liu, Y. Xiao, H. Chen, S. Ozdemir, S. Dodle, and V. Singh, "A Survey of Payment Card Industry (PCI) Data Security Standard," IEEE Communications Surveys & Tutorials, Vol. 12, No. 3, pp. 287-303, Third Quarter 2010.

[14] “Wikipedia—Linux,” available: http://en.wikipedia.org/wiki/Linux.

[15] “Wikipedia—Server,” available: http://en.wikipedia.org/wiki/Server_(computing).

[16] “Wikipedia—Cloud Computing, ” available: http://en.wikipedia.org/wiki/Cloud_computing.

[17] Z. Xiao and Y. Xiao, “Security and Privacy in Cloud Computing,” IEEE Communications Surveys & Tutorials, Vol. 15, No. 2, Second Quarter 2013, pp. 843-859.

[18] Z. Xiao and Y. Xiao, "Achieving Accountable MapReduce in Cloud Computing," (Elsevier) Future Generation Computer Systems, Vol. 30, No.1, Jan. 2014, pp. 1–13.

[19] B. Saphir, “Linux for Scientific Computing,” available: http://www.lugod.org/presentations/linux4scientificcomputing.pdf.

[20] “Wikipedia—Workstation,” available: http://en.wikipedia.org/wiki/Workstation.

[21] “Linux Benchmarking HOWTO,” available: ftp://ftp.lyx.org/pub/sgml-tools/website/HOWTO/Benchmarking-HOWTO/t152.html

[22] “Evaluation assurance levels,” available: http://cygnacom.com/labs/cc_assurance_index/CCinHTML/PART3/PART36.HTM

[23] J. He, “Linux System Call Quick Reference,” available: http://www.digilife.be/quickreferences/qrc/linux%20system%20call%20quick%20reference.pdf.

[24] “Linux System Call Reference,” available: http://syscalls.kernelgrok.com/.

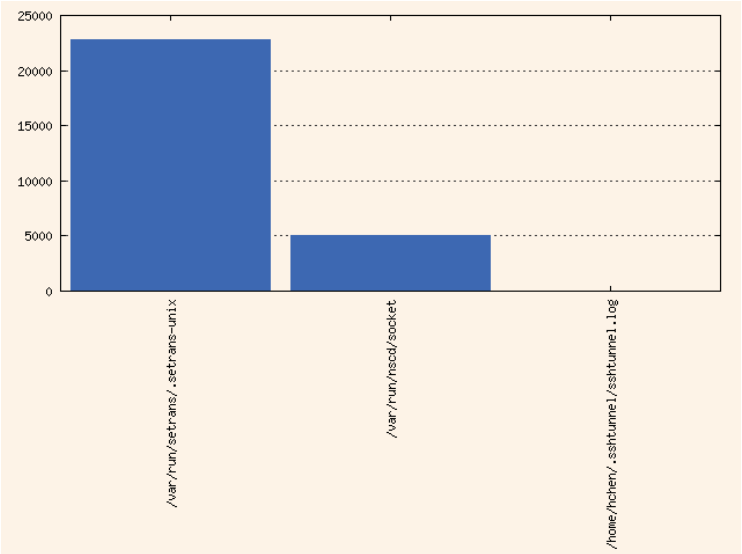

Fig. 4 Failed Access Statistics Report

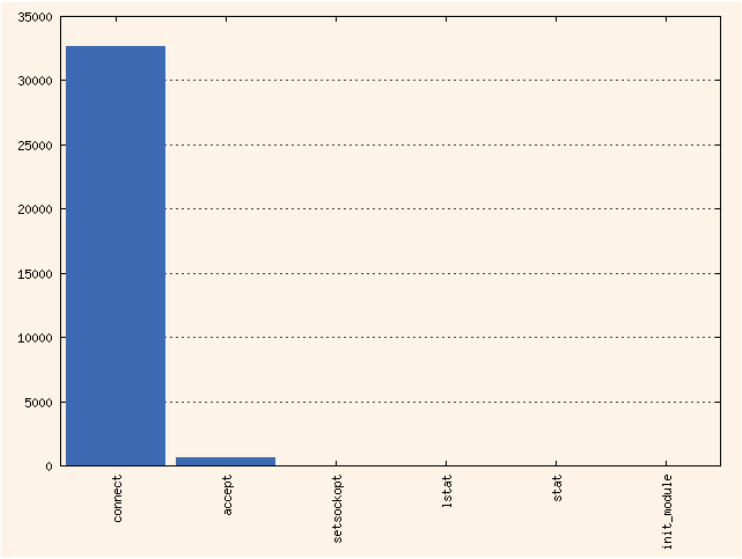

Fig. 5 System Call Statistics Report

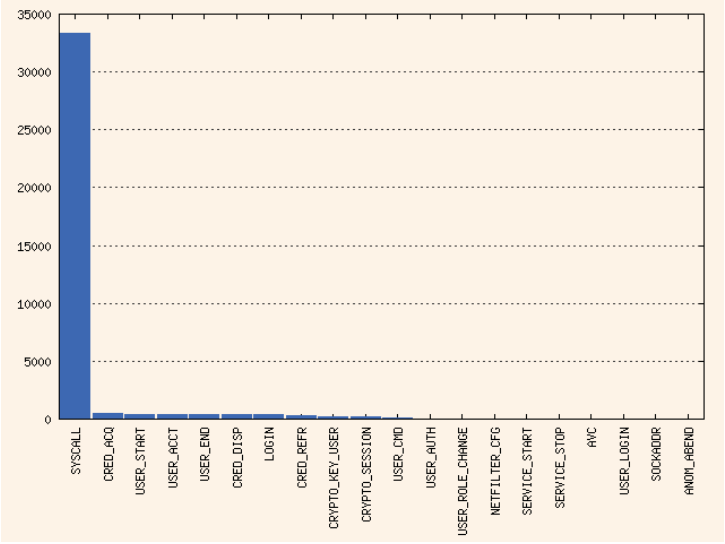

Fig. 6 Event Ranking