SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2015; **8**:3523–3534
Published online 4 June 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1277

RESEARCH ARTICLE

Auditing overhead, auditing adaptation, and benchmark evaluation in Linux

Lei Zeng[1], Yang Xiao[1]* and Hui Chen[2]

[1] Department of Computer Science, The University of Alabama, Tuscaloosa 35487-0290, AL, U.S.A.
[2] Department of Mathematics and Computer Science, Virginia State University, Petersburg 23806, VA, U.S.A.

ABSTRACT

Logging is a critical component of Linux auditing. However, our 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 to 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. We then design an adaptive auditing mechanism that 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. We further adopt LMbench to evaluate the performance of key operations in Linux with compliance to four security standards. Copyright © 2015 John Wiley & Sons, Ltd.

KEYWORDS
logging; overhead; Linux; auditing

***Correspondence**
Yang Xiao, Department of Computer Science, The University of Alabama, 101 Houser Hall, PO Box 870290, Tuscaloosa 35487-0290, AL, U.S.A.
E-mail: yangxiao@ieee.org

1. INTRODUCTION

Auditing in operating systems 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 [1] published by the US Department of Defense in 1985 defines the requirement of logging for auditing purpose. This book 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 level of security requirements while category A has the highest level. 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 subclasses, 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, log-in 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 regard 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 [1–8]. 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 nonvolatile memory.

Linux audit framework [9] helps Linux meet many government and industrial security standards, such as Controlled Access Protection Profile/Evaluation Assurance Level 4+ (CAPP/EAL4+) [10], Labeled Security Protection Profile (LSPP) [11], role-based access control (RBAC) [12], National Industrial Security Program Operating Manual (NISPOM) [13], Federal Information Security Management Act (FISMA) [14], payment card industry (PCI) [15,16], and Director of Central Intelligence Directive 6/3 (DCID 6/3) [17]. 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 certifications 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/high-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 adoption 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. We further adopt LMbench to evaluate the performance of key operations in Linux with compliance to four security standards.

The rest of the paper is organized as follows. In Section 2, we introduce Linux audit framework. Section 3 presents the Linux benchmarks and Linux security standards. In Section 4, we study overhead of Linux logging. We propose an adaptive mechanism and study the performance of the proposed adaptive logging in Section 5. In Section 6, we present an evaluation for Linux benchmarks. Finally, we conclude the paper in Section 7.

2. LINUX AUDIT FRAMEWORK

Linux audit provides users with a way to analyze system activities in great detail [9]. It does not, however, protect users from attacks [9]. Instead, Linux audit helps users to track these issues and take security measures to prevent them [9].

Linux audit framework includes several components: auditd, auditctl, audit rules, aureport, ausearch, audispd, and autrace, as shown in Figure 1 [9].

We explain the functions in Figure 1 as follows [9]:

- 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.

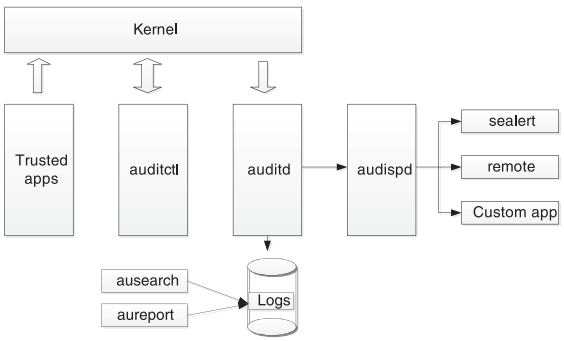

Figure 1. Linux audit framework [9].

Security Comm. Networks 2015; **8**:3523–3534 © 2015 John Wiley & Sons, Ltd.
DOI: 10.1002/sec

An audit event that is written to disk is shown in Figure 2.

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 [9]:

- The type field in the record denotes the type of events recorded.
- The msg field stores a message identification (ID). The arch field refers to the central processing unit (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 that 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 the mode field stores the file access permission in a numerical representation.
- The uid and gid of inode are saved in the ouid and ogid fields, respectively.
- Rdev is not able to be applied for the example record in Figure 2.

```
type=SYSCALL msg=audit(1175176190.105:157): arch=40000003
syscall=5 success=yes
exit=4 a0=bfba161c a1=8000 a2=0 a3=8000 items=1 ppid=4457 pid=4462
auid=0
uid=0 gid=0 euid=0 suid=0 fsuid=0 egid=0 sgid=0 fsgid=0 tty=pts0
comm="less"
exe="/usr/bin/less" subj=unconstrained key="LOG_audit_log"
type=CWD msg=audit(1175176190.105:157): cwd="/tmp"
type=PATH msg=audit(1175176190.105:157): item=0
name="../var/log/audit/audit.log" inode=458325 dev=03:01
mode=0100640 ouid=0
ogid=0 rdev=00:00
```

Figure 2. An audit event record [9].

3. LINUX BENCHMARKS AND SECURITY STANDARDS

3.1. Synthetic benchmarks versus application benchmarks

There are two kinds of benchmarks for operating systems: synthetic and application [18]. In application benchmarks, a common application is chosen to test the system performance. One drawback of application benchmarks is that it is not likely that source Linux codes may be not available for common application benchmarks.

In synthetic benchmarks, a chosen component of the computer system is exercised to the maximum capacity. Whetstone suite is one example of a synthetic benchmark in FORTRAN, and it measures the performance of floating point in a CPU [18]. The main drawback of synthetic benchmarks is that the chosen component and the program to exercise it may not present the real performance of the computer system in realistic situations. For example, the loop in the Whetstone suite is small and may be held in the cache of the CPU instead of the memory, but it may fill floating-point unit all the time to achieve the maximum capacity. Therefore, we need to be careful when understanding the results of benchmarking current microprocessors [19]. Furthermore, synthetic benchmarks should test a particular aspect of the system independently of all other aspects of the system. One common mistake is to average several different synthetic benchmarks to claim a better benchmark [19].

In summary, if we can understand the limitations and purposes of synthetic benchmarks, they may be good choices for benchmarks, particularly when source codes of common application benchmarks are not available for Linux, even though application benchmarks may test the system performance better.

3.2. Linux standard benchmarks

Even though kernel compilation is a usual task in Linux, exercising most of functions of common benchmarks (except floating point) so that it can be a good candidate for individual tests, it cannot be used as a good benchmark for different systems because its performance differs with different configurations of system software and hardware [18,19].

There are many open-source Linux benchmark tools. More information can be found in a website such as OpenBenchmarking.org. Table I presents six well-known and free Linux benchmark tools [20].

3.3. Linux security standards

Linux audit framework helps Linux achieve a number of security standards (CAPP/EAL4+, LSPP, RBAC, NISPOM, FISMA, PCI, and DCID 6/3).

Table I. Six free Linux benchmark tools [20].

Benchmarks for Linux	Explanation
Phoronix Test Suite	Benchmark for comprehensive testing
IOzone	Benchmark for file system
Netperf	Benchmark for network performance
LLCbench	Benchmark for low-level architecture
HardInfo	Benchmark and system profiler
GtkPerf	Benchmark for GTK+ performance

3.3.1. EAL2+.

SUSE Linux Enterprise Server 8 obtained certification with Evaluation Assurance Level 2 (EAL2+) in 2003 [21]. This Linux distribution provided identification and authentication along with basic discretionary access control capabilities [22]. Writing a security target using these capabilities is sufficient enough to certify a system at EAL2+ [23]. EAL2 is useful when developers or users need a low level or a moderate level of security, particularly when legacy systems need security enhancement or when the developers are not available all the time [19].

3.3.2. CAPP at EAL3+ and EAL4+.

CAPP mandates auditing security-relevant events and requires EAL3+ or higher [24]. Therefore, different audit systems in Linux were developed to achieve compliance with CAPP/EAL4+. Linux Audit Subsystem [25] and lightweight audit framework were developed by SUSE and Red Hat [26]. Although the two auditing systems share some common features, they are not compatible [22].

EAL3 allows developers to design better security at the design stage without changing much of the common development practices, and it is good for developers or users with a moderate-level security to deeply analyze Target Of Evaluation (TOE) [19].

3.3.3. EAL4+.

EAL4 is the highest in terms of security levels and is feasible to achieve financial gains for existing software products [19]. It provides the maximum security level with reasonable development practices without requiring special knowledge and skills for developers [19].

EAL4 includes security function analysis, designs of TOE, designs of TOE security policies, TOE testing of security functions, vulnerability analysis, control usage of development environments, and procedures of secure delivery, without tamping the TOE [19].

3.3.4. LSPP.

Labeled Security Protection Profile is a superset of CAPP and mainly mandates that multi-level security (MLS)—enforcement of Bell–LaPadula rules should be implemented [22]. Figure 3 shows an example of MLS. There are four objects: A, B, C, and D, with corresponding security level labeled. There is a process with security labeled as secret. In the MLS model, the process can only read lower security level objects and write higher security level objects. The principle is to ensure that information

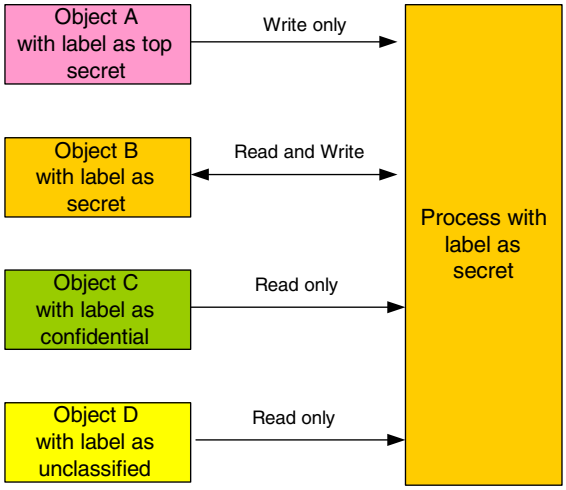

Figure 3. Multi-level security system [41].

with high-level security label cannot flow to entity with low-level security label. Therefore, process cannot write to objects C and D and cannot read object A. LSPP has additional requirements based on CAPP [22]:

- Security audit: adding labels of a new subject and a new object to audit records for future search and sorting based on these new labels.
- User data protection: enforcing MLS policy, adding labels of subject and object, and controlling labeled and unlabeled data import and export.
- Identification and authentication: enforcing MLS policy based on subject labels.
- Security management: restricting MLS on object label changes and initializing static MAC attributes.

3.3.5. RBACPP.

RBAC model is illustrated in Figure 4. In this figure, there is a role that denotes Role 1, and three users that denote User1, User2, and User3. These three users have the same role and thus can only do tasks under Role 1's privileges. In this figure, Role 1 is assigned privileges to do transaction a with object A and transaction b with object B. Another privilege is not assigned to Role 1, and thus, User1, User1, and User3 cannot do transactions other than trans_a and trans_b. RBAC protection profile (RBACPP)

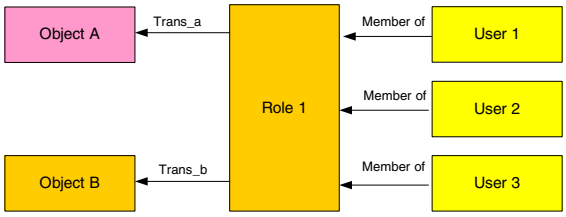

Figure 4. Role-based access control [42].

DOI: 10.1002/sec

has additional requirements based on LSPP pertaining to roles and management of role data [22]:

- Security audit: auditing role-related attributes, such as role and privileges; capable of searching and sorting audited data with role access enabled.
- User data protection: enforcing RBAC policy on subjects and objects with roles controlling permissions.
- Identification and authentication: adding role data to user attribute.
- Security management: restricting RBAC on role management and mandating usage of secure values, RBAC control of data.
- Protection of TOE functions: preserving RBAC database across failures; recovering after severe interruption manually; indicating success or failure of recovery.
- TOE access: restricting users to authorize roles; denying users' log-in with empty role set.

3.3.6. NISPOM.

National Industrial Security Program Operating Manual provides regulations on standard requirements and procedures and mandates the requirements for information system security from data manipulation to logging and auditing when government contractors access classified information [27].

3.3.7. FISMA.

The FISMA of 2002 provides program requirements for federal agencies to satisfying requirements of security information and report [14].

3.3.8. DCID 6/3.

Protecting Sensitive Compartmented Information within Information Systems (DCID 6/3) gives requirements and guidance for protecting sensitive data in the information system. Besides, it mandates technical and administrative requirements in a given environment [28].

4. OVERHEAD OF LINUX AUDIT LOGGING

4.1. Traffic patterns and types of events

Linux distributions have been used as a server operating system for its stability, security, and pricing [29]. There are several types of servers in the general network environment: web server, mail server, file server, application server, catalog server, domain name service (DNS) server, data server, and so on [30]. 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 [31–33]. Besides, Linux distributions play a major role on scientific computing because of their efficiency [34] and stability. Last but not least, Linux can be used in workstation. Current workstations use sophisticated CPU such as Intel Xeon, AMD Opteron, or IBM Power and run Unix/Linux systems to provide reliable workhorse for computing-intensive tasks [35].

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.

4.2. Experimental design and overhead measurement

There are many free Linux benchmark tools available such as Phoronix Test Suite and Netperf [18]. Other benchmark tools are listed in Table I. The drawback is that the benchmark is different from real situations [19,36].

Similar to the logging performance study, we will run a worker program (on the host, or from the network) and 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 $C = (T_{on} - T_{off})/T_{off}$.

4.3. Performance overhead analysis of Linux audit logging

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

Because there are 347 system calls in arch/x86/kernel/syscall_table_32.S for x86 architecture, these primary system calls in [37] and [38] are configured to be audited. The performance overhead is shown in Figure 5. In this figure, the x dimension denotes the action frequency in the worker program from 1, 5, 10 to 20 times/s. Because the action in the worker program can only happen 23.8 times/s, the highest frequency recorded in Figure 5 is 20 times/s. The y dimension denotes the overhead computed using $C = (T_{on} - T_{off})/T_{off}$.

Significant performance overhead is observed when action frequency is 20 times/s. With the action frequency increasing, performance overhead increases as well. Because auditing these system calls incurred system overhead, the

Table II. 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

CPU: central processing unit.

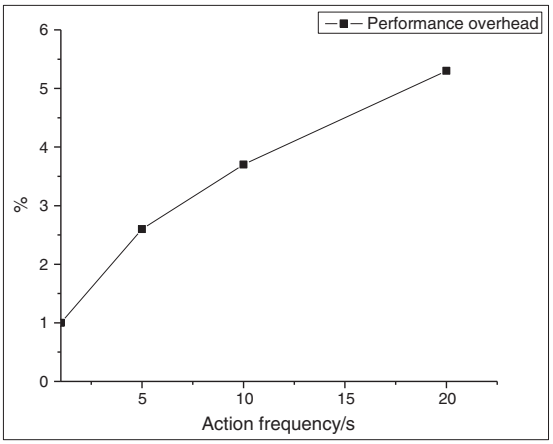

Figure 5. Performance overhead when auditing 347 primary system calls.

performance penalty will increase when the frequency of invoking system calls increases.

5. ADAPTIVE-LOGGING AND PERFORMANCE EVALUATION

5.1. Adaptive auditing mechanism

The study on the problem leads to 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.

The logging system can be both input/output (I/O) and CPU intensive. Therefore, it is better if it has the ability to turn logging on or off on-demand or to adaptively log events. Adaptive logging also indicates that based on either users' requirements of overhead or whether there is a security incident or some other important events accompanied with precursory events, adaptively log more or less events. In other words, if the system detects some precursors that important events may happen, the system can automatically adjust the level of logging or turn the logging on. For example, the system can start to monitor a system call or a group of calls more or less based on either users' requirements or on some precursors. In this paper, we do not define those security incidents or precursory events in particular but provide a methodology/framework that can adopt those easily.

Because a system usually functions normally and the associated logging data are not very useful, this capability ensures that the logging overhead can be limited by turning only course-grained logging on and off when the system is running under normal conditions. However, what constitutes a normal condition is yet to be answered. Note that user requirements regarding the resolution of logging events can vary. This capability also helps the system meet the system requirements and avoid unnecessary logging.

Next, the three goals are (1) to identify critical events specific for Linux server traffic, (2) to estimate the performance overhead introduced by the auditing function, and (3) to analyze the security performance.

Common tasks in the Linux server, such as bash, aureport, crond and yum, and so on, are shown in Figure 6.

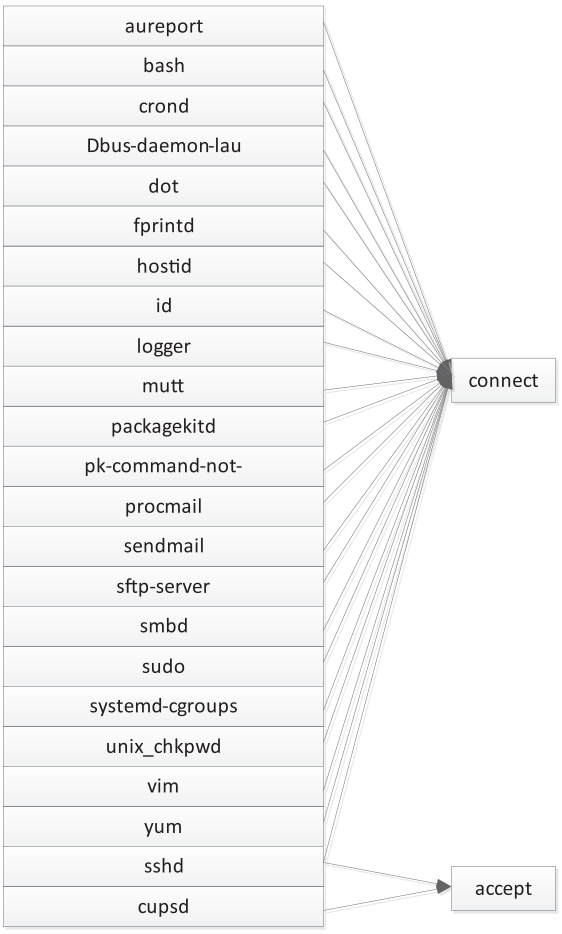

Figure 6. Linux tasks and corresponding system calls.

These tasks will eventually invoke the connect system call or the accept system call, which 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.

5.2. Performance evaluation

This subsection shows that adaptive logging has reduced overhead.

Figures 7–11 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 Figure 7. 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 Figure 7 is a log file owned by ssh utility and does not have a failed access recorded.

The system call statistics report is shown in Figure 8. In this figure, two system calls, connect and accept, are recorded. The connect system call was invoked many times because many of the common tasks of server traffic will invoke this system call.

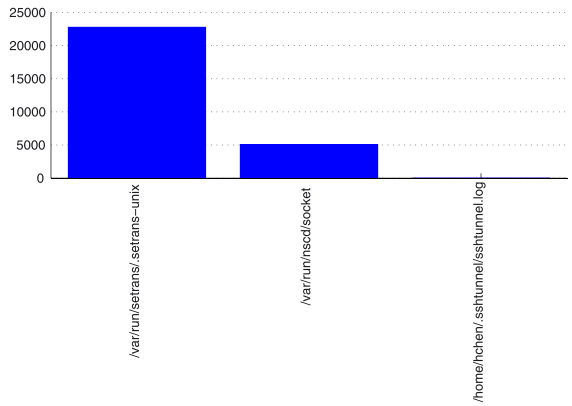

Figure 7. Failed-access statistics report.

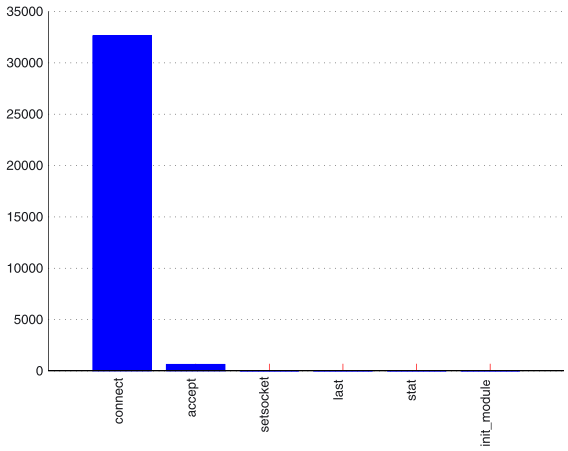

Figure 8. System call statistics report.

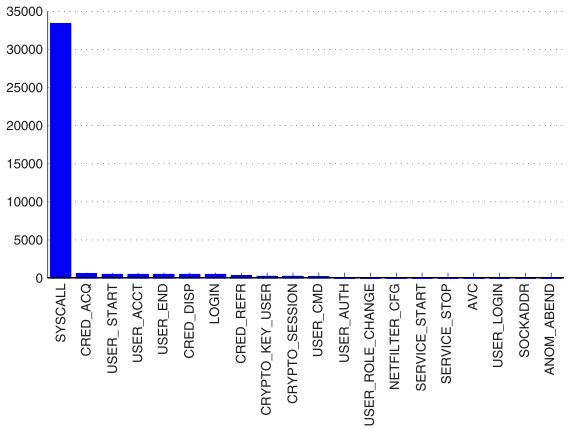

Figure 9. Event ranking.

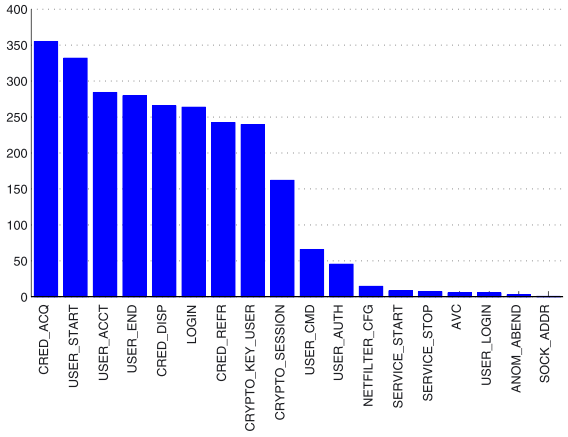

Figure 10. Nonsystem-call event ranking.

Event ranking is shown in Figure 9. In the figure, the system call event ranked first among all events in the server operating system.

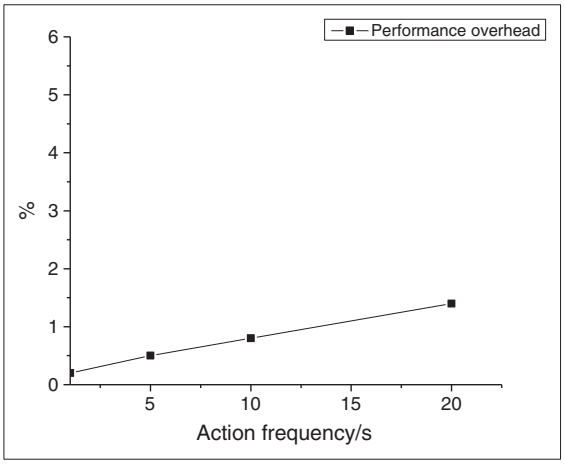

Figure 11. Performance overhead.

Nonsystem-call event ranking is shown in Figure 10. The event CRED_ACQ had the highest rank among all nonsystem call events. For a server operating system, remote access to the server involves obtaining credentials of different objects. The high occurrence of remote access will cause high frequency of CRED_ACQ. Furthermore, user-related events, such as USER_START, USER_ACCT, and USER_END, are common as well.

Performance overhead for adaptive logging is shown in Figure 11. In this figure, the x dimension denotes the action frequency in the worker program from 1, 5, 10 to 20 times/s. Because action in the worker program can only happen 23.8 times/s, the highest frequency recorded in Figure 11 is 20 times/s. The y dimension denotes the overhead computed using $C = (T_{on} - T_{off})/T_{off}$.

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.

5.3. Security evaluation

Linux audit framework helps Linux achieve a number of security standards (CAPP/EAL4+, LSPP, RBAC, NISPOM, FISMA, PCI, and DCID 6/3). A brief analysis shows that the adaptive logging does not lead to any violation of the satisfied standards. Performance overhead is shown in Figure 12. In this figure, the x dimension denotes the action frequency in the worker program from 1, 5 , 10 to 20 times/s. Because action in the worker program can only happen 23.8 times/s, the highest frequency recorded in Figure 12 is 20 times/s. The y dimension denotes the overhead computed using $C = (T_{on} - T_{off})/T_{off}$.

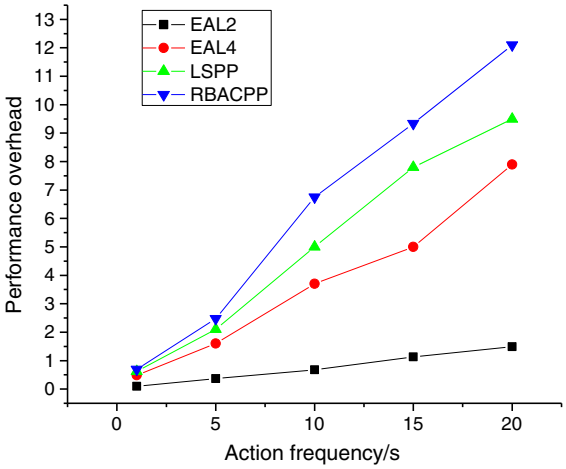

Figure 12. Performance overhead for different security models. EAL2, Evaluation Assurance Level 2; EAl4, Evaluation Assurance Level 4; LSPP, Labeled Security Protection Profile; RBACPP, role-based access control protection profile.

We can observe that RBACPP has the highest performance overhead among all security standards, namely EAL2, EAL 4, LSPP, and RBACPP. The reason is that EAL2 has the minimum requirements and RBACPP is the most restricted security standard, which involves security audit, user data protection, identification and authorization, security management, protection of TOE, and TOE access. We also observed that when action frequency increases, the performance overhead also increases. Because each action, which contains 83 tasks, will trigger the Linux audit function, more actions will cause more system overhead.

6. LINUX BENCHMARK EVALUATION

In this section, a free Linux benchmark tool, LMbench, is used to evaluate Linux audit framework with compliance to the different security standards. LMbench is designed specifically for measuring the performance of key operations in Linux such as file system access and memory access [39].

6.1. Arithmetic operation

Figure 13 shows that for basic arithmetic operations, there is not much difference between these four security standards, namely EAL2, EAL4, LSPP, and RBAC. The reason is that these CPU-bound tasks do not invoke any system calls that we audit.

6.2. Memory and file system operations

Figure 14 shows that no significant system overhead is observed for cache and memory access for all security

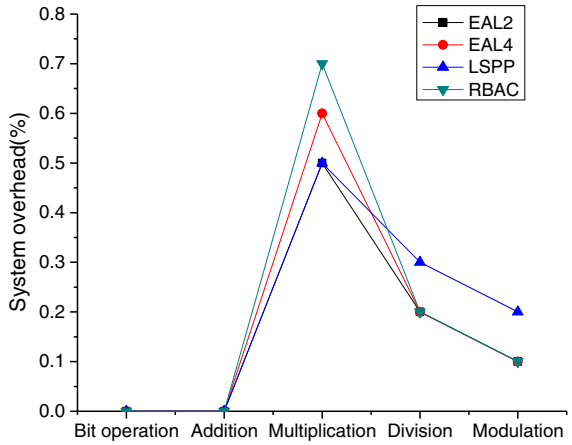

Figure 13. Arithmetic operation benchmarks. EAL2, Evaluation Assurance Level 2; EAl4, Evaluation Assurance Level 4; LSPP, Labeled Security Protection Profile; RBAC, role-based access control.

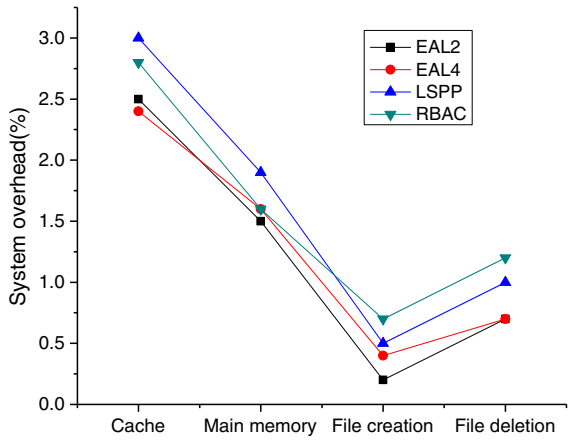

Figure 14. Memory and file system operations benchmarks. EAL2, Evaluation Assurance Level 2; EAl4, Evaluation Assurance Level 4; LSPP, Labeled Security Protection Profile; RBAC, role-based access control.

standards. In the test machine, we have level 1 and level 2 caches. Because access to the memory is implemented through assembly code where no audit function is performed, we should not observe significant system overhead.

On the other hand, file system operations are considered to be sensitive operations and will introduce extensive file system-related audit records. We expect to observe high system overhead. However, no significant system overhead is observed for file system operations such as file creation and file deletion.

6.3. Process-related benchmarks

Context switch is the process of storing and restoring the state of a process in order to resume execution from the same point at a later time [40]. System overhead of process-related benchmarks is shown in Figure 15. We performed benchmarks on 8, 16, and 32 processes and associate 0-KB, 16-KB, and 32-KB data. No significant difference for four security standards is observed for process-related benchmarks.

6.4. Network benchmarks

At last, we measure the performance of inter-process communications. There is not much difference, as shown in Figure 16, between these four security standards for different inter-process communications. RBAC security model has relatively higher system overhead over the three security models. The reason is role-based authentication and authorization are required for RBAC, and these sensitive operations will incur audit records.

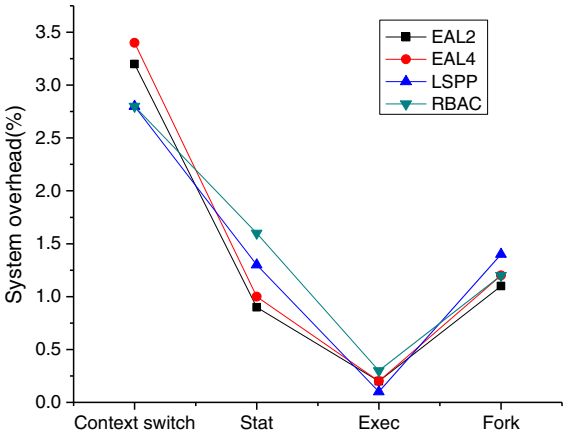

Figure 15. Process-related benchmarks. EAL2, Evaluation Assurance Level 2; EAl4, Evaluation Assurance Level 4; LSPP, Labeled Security Protection Profile; RBAC, role-based access control.

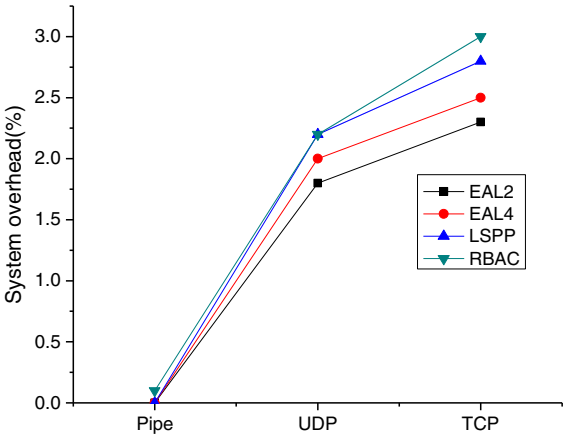

Figure 16. Inter-process communication for the different security standards. EAL2, Evaluation Assurance Level 2; EAl4, Evaluation Assurance Level 4; LSPP, Labeled Security Protection Profile; RBAC, role-based access control.

6.5. Different factors of Linux audit framework

Although we observed that there is a significant performance penalty when auditing is turned on a Linux system, as shown in Figure 5, people may argue that the performance penalty may not be dominated by the Linux auditing alone: the factors that contribute to Linux auditing-performance penalty besides "action frequencies" that are used include multiple sources. Imaging this, the overhead comes from many factors. If a slow hard drive makes logging slow, it is a slow network. If we only store events in memory, then we remove the factors of "hard drive and network". The factors would be I/O speed and memory speed, concurrency control, and so on. Besides the factors that we explored such as actions/second and file versus network, another important factor is "cyclic buffer" size. Perhaps, it can help us know where the overhead comes from if we vary the buffer size. Next, we evaluate different factors of Linux audit framework.

We try to answer the following questions: If we save the logged data on disks, does most of penalty come from disk seek time/read/write time? If so, if we replace disks by networks (we can configure the syslogd to transport audit data to another machine), do we observe less penalty? How about we increase memory buffer?

In order to evaluate the performance penalty from disk seek time/read/write time, two test cases are designed. The first one writes log data to files when necessary, and the second is configured to dispatch log data to the network. Dispatching log data to the network will not incur disk seek time/read/write time on test machine. In Figure 17, the improved system performance is observed when audit daemon does not write log data to files. In conclusion, most of the penalty comes from disk seek time/read/write time.

In Figure 18, three test cases are designed with different buffers in kernel from 320, 640, and 1280. The cyclic

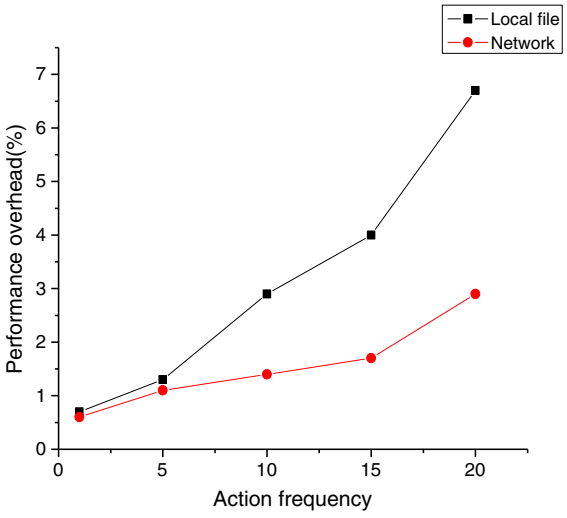

Figure 17. Performance overhead of audit daemon when log data are dispatched to the network.

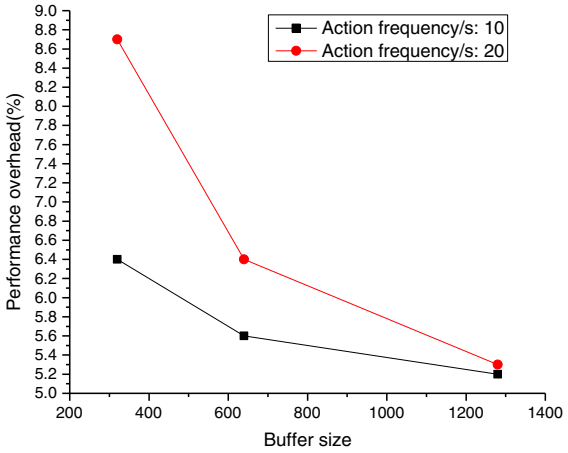

Figure 18. Performance overhead for different buffer sizes.

buffer in the kernel space is a configuration factor in Linux audit system: in /etc/audit/audit.rules (note the -b parameter). We observe that with bigger buffer in kernel, the curve is increasingly flatter and bigger buffer can help improve system performance in stress events because more buffers in kernel can improve system performance in stress events.

7. 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. Finally, we used LMbench to evaluate the performance of key operations in Linux with compliance to the four security standards.

ACKNOWLEDGEMENTS

This paper was supported in part by the National Natural Science Foundation of China (NSFC) under grant 61374200 and the U.S. National Science Foundation (NSF) under grants CNS-0716211, CCF-0829827, CNS-0737325, and CNS-1059265.

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. Xiao Y. Flow-net methodology for accountability in wireless networks. *IEEE Network* 2009; **23**(5): 30–37.
3. Fu B, Xiao Y. 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. Fu B, Xiao Y. Accountability and Q-Accountable logging in wireless networks. *Wireless Personal Communications* 2014; **75**(3):1715–1746.
5. Xiao Y, Yue S, Fu B, Ozdemir S. GlobalView: building global view with log files in a distributed/networked system for accountability. *(Wiley Journal of) Security and Communication Networks* 2014; **7**(12):2564–2586.
6. Xiao Y, Meng K, Takahashi D. Accountability using flow-net: design, implementation, and performance evaluation. *(Wiley Journal of) Security and Communication Networks* 2012; **5**(1):29–49.
7. Meng K, Xiao Y, Vrbsky SV. Building a wireless capturing tool for WiFi. *(Wiley Journal of) Security and Communication Networks* 2009; **2**(6):654–668. doi:10.1002/sec.107.
8. Xiao Y. Accountability for wireless LANs, ad hoc networks, and wireless mesh networks. *IEEE Communication Magazine* 2008; **46**(4):116–126.
9. SUSE Linux Enterprise—the Linux audit framework. Available: http://www.suse.com/documentation/sled10/pdfdoc/audit_sp2/audit_sp2.pdf, May 8, 2008
10. 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 [accessed on July 20, 2012].
11. Information Systems Security Organization. Labeled security protection profile, URL: http://www.commoncriteriaportal.org/files/ppfiles/lspp.pdf, October 8, 1999.
12. Sandhu RS, Coyne EJ. Role-based access control models, URL: http://profsandhu.com/journals/computer/i94rbac(org).pdf, 1996, IEEE.
13. National Industrial Security Program—Operating Manual, available: http://www.ncms-isp.org/NISPOM_200602_with_ISLs.pdf, February 2006.
14. Federal Information Security Management Act of 2002, available: http://en.wikipedia.org/wiki/Federal_Information_Security_Management_Act_of_2002.
15. Payment card industry (PCI). http://en.wikipedia.org/wiki/Payment_card_industry
16. Liu J, Xiao Y, Chen H, Ozdemir S, Dodle S, Singh V. A survey of payment card industry (PCI) data security standard. *IEEE Communications Surveys & Tutorials* 2010; **12**(3): 287–303 Third Quarter.
17. 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
18. Linux Benchmarking HOWTO, available: ftp://ftp.lyx.org/pub/sgml-tools/website/HOWTO/Benchmarking-HOWTO/t152.html also http://www.tldp.org/HOWTO/pdf/Benchmarking-HOWTO.pdf
19. Evaluation assurance levels, available: http://cygnacom.com/labs/cc_assurance_index/CCinHTML/PART3/PART36.HTM
20. 6 best free Linux benchmark tools, http://www.linuxlinks.com/article/2012042806090428/BenchmarkTools.html
21. IBM Corporation and SUSE. IBM and SUSE earned first security certification of Linux, available: http://www-03.ibm.com/press/us/en/pressrelease/5662.wss.
22. Wilson G, Weidner K, Salem L. Extending Linux for multi-level security, available: http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaav/SELinux/lspp-rbac.pdf
23. Atsec Gmbh and IBM Corporation. SUSE Linux Enterprise Server V8 Security Target, V1.6, 2003. Available: http://www.commoncriteriaportal.org/%20public/files/%20epfiles/0216b.pdf
24. ATSEC GMBH AND IBM CORPORATION. SuSE Linux Enterprise Server V 8 with Service Pack 3 Security Target for CAPP Compliance, v2.7. 2003. http://www.commoncriteriaportal.org/public/files/epfiles/0234b.pdf
25. SUSE LINUX AG. Linux audit-subsystem design documentation for Linux kernel 2.6, v0.1, 2004. http://www.uniforum.chi.il.us/slides/HardeningLinux/LAuS-Design.pdf
26. ATSEC GMBH AND IBM CORPORATION. Red Hat Enterprise Linux Version 4 Update 1 Security Target for CAPP Compliance, v2.6. 2005. http://www.commoncriteriaportal.org/public/files/epfiles/STVID10072-ST.pdf
27. Wikipedia—National Industrial Security Program, available: http://en.wikipedia.org/wiki/National_Industrial_Security_Program
28. Protecting sensitive compartmented information within information system manual, available: http://www.fas.org/irp/offdocs/dcid-6-3-manual.pdf
29. Wikipedia—Linux, available: http://en.wikipedia.org/wiki/Linux
30. Wikipedia—server, available: http://en.wikipedia.org/wiki/Server_(computing)
31. Wikipedia—cloud computing, available: http://en.wikipedia.org/wiki/Cloud_computing
32. Xiao Z, Xiao Y. Security and privacy in cloud computing. *IEEE Communications Surveys & Tutorials* 2013; **15**(2): 843–859. Second Quarter
33. Xiao Z, Xiao Y. Achieving Accountable MapReduce in cloud computing. *(Elsevier) Future Generation Computer Systems* 2014; **30**(1):1–13.
34. Saphir B. Linux for scientific computing, available: http://www.lugod.org/presentations/linux4scientificcomputing.pdf
35. Wikipedia—workstation, available: http://en.wikipedia.org/wiki/Workstation

36. Zeng L, Xiao Y, Chen H. Linux auditing: overhead and adaptation, Proceedings of The IEEE International Conference on Communications 2015 (IEEE ICC 2015).
37. He J. Linux System Call Quick Reference, available: http://www.digilife.be/quickreferences/qrc/linux%20system%20call%20quick%20reference.pdf
38. Linux System Call Reference, available: http://syscalls.kernelgrok.com/.
39. Jiang Q. Assessing the performance impact of the Linux kernel audit trail, May 2004, available: http://www.scribd.com/doc/12807832/Assessing-the-Performance-Impact-of-the-Linux-Kernel-Audit-Trail
40. Context switch—Wikipedia, available: http://en.wikipedia.org/wiki/Context_switch
41. Multi-level security, available: http://www.centos.org/docs/5/html/Deployment_Guide-en-US/sec-mls-ov.html
42. Ferraiolo DF, Kuhn DR. Role based access control, 15th National Computer Security Conference, 1992, available: http://arxiv.org/ftp/arxiv/papers/0903/0903.2171.pdf

Security Comm. Networks 2015; **8**:3523–3534 © 2015 John Wiley & Sons, Ltd.
DOI: 10.1002/sec