Secure and Usable Integrity Protection Model for Operating Systems

Chang Eric
Yale University

Abstract

Host compromise is one of the most serious security problems for operating systems today. Existing integrity protection models for operating systems are difficult to use; on the other hand, the most available integrity protection models only provide heuristic approaches without strong guarantees.

This paper presents SecGuard, a secure and high-available integrity protection model for operating systems. To ensure the security of systems, SecGuard provides formal guarantees that operating systems are security under three threats: network-based threat, IPC communication threat, and contaminative file threat. On the other hand, we introduces some novel mechanisms to ensure high-available of the model. For instance, SecGuard leverages the information of the existing discretionary access control mechanism to initialize integrity labels for subjects and objects in the systems. Moreover, we describe the implementation of SecGuard for Linux using Linux Security Modules framework, and show it has low overhead and effectively achieve security and high-availability for operating systems.

1 Introduction

As the scale of global Internet increases, the threat from remote network becomes the norm rather than the exception. For instance, computer worms can propagate by first compromising vulnerable hosts, and then propagate to other hosts. Compromised hosts may be organized under an uniform command and control infrastructure, forming botnets [9], [17]. Botnets can be used for launching attacks such as decoying, spam E-mail and distributed denial of service (DDoS). Most existing designs against such network-based attacks rely on the network level using valuable technologies such as firewalls and network intrusion detection systems. However, in order to address the above problems effectively, we have to seek the key cause of these threats — the vulnerability of end hosts [23, 30, 13, 19, 32, 20, 26, 21, 10]. The study in [12] indicated that two key reasons why there are some vulnerability in the end hosts are: (1) software running on the hosts are buggy, and (2) the discretionary access control mechanism in operating systems is insufficient for defend against network-based attacks [29].

Generally, previous studies on making operating systems more security mainly focus on making use of mandatory access control (MAC). The existing MAC models, e.g., DTE [4, 5], Janus [8], Security Enhanced Linux (SELinux) [3], Apparmor [1, 6], systrace [15], and Linux Intrusion Detection System (LIDS) [2], are very complex to configure and difficult to use. For instance, there are too much different classes of objects in SELinux, and this is very difficult to configure for normal users; moreover, after configuring such MAC models, some existing applications will not be able to run and many common practices for administering the computer system will become impossible. On the other hand, there has also been some work on how to design high-available MAC models, e.g., LOMAC [7] and UMIP [12]. However, these studies only provide heuristic approaches without hard guarantees. Furthermore, these models are evaluated only against synthetic attacks, and they also need to make some strong assumptions. For example, UMIP model allows the remote system administration through secure shell daemon (sshd) to be completely trustworthy (Completely trustworthy means the integrity level of process can not drop). However, in fact, the attackers can successfully exploit bugs in such daemon program, and then take over the system. In summary, it is still an open question how to design a high-available MAC model to protect the integrity of operating systems [24], [25], [31], [28], [27], [16].

Our Goal and Approach: This paper aims to answer the following question:

Is it possible to design a high-available integrity protection model which does not only protect the integrity *of computer systems without complex configuration, but also provides strong provable guarantees?*

The fundamental insight answering this question is that the existing discretionary access control (DAC) information (such as current authority of process and DAC list in the operating system) can be used to generate mandatory integrity protection mechanisms, so this ensures the model high-available (Users need not to face the complex configuration tasks and existing applications can still be used under our model). Furthermore, we provide formal guarantees to proof the security of our model.

Our Contributions and Results: This paper presents SecGuard, a secure and high-available integrity protection model for operating systems. SecGuard targets three threats in systems: network-based threat, IPC communication threat, and contaminative file threat. SecGuard has the following salient features:

- SecGuard secures operating systems from three threats: network-based threat, IPC communication threat, and contaminative file threat.
- SecGuard is a high-available MAC model, and it is easier to be configured and used than the existing MAC models.
- SecGuard provides formal guarantees; therefore, the security of the model can be ensured in theory.
- SecGuard has been implemented as a prototype protection system in Linux, and we present some representative designs and evaluations.

Outline: The rest of this paper is organized as follows. The security targets, assumptions and threat scenarios will be described in Section 2. The details of SecGuard will be discussed in Section 3. Section 4 shows the formal definitions and guarantees of SecGuard. Our implementation of SecGuard and its evaluations will be presented in Section 5. Finally, we conclude in Section 6.

2 Security Targets, Assumptions and Threat Scenarios

Targets of Integrity Protection: SecGuard aims at protecting the integrity of operating systems. However, due to the various concepts in the integrity protection (Many definitions on integrity protection given by security experts are different), we firstly define the targets of integrity protected by SecGuard in order to avoid ambiguity in the following sections.

The study in [11] indicated that the threats of integrity confronted by operating system are mainly from three aspects:

1. The data is modified maliciously by attackers in the information transmission;
2. Due to confronting the hardware errors or application bugs, the data is modified during the information transmission;
3. The data is maliciously modified by malicious code such as *Trojan horse*.

The targets of integrity protection of SecGuard are (1) and (3). Therefore, for SecGuard, there are two cases that system data can be modified: one is the data directly modified by the processes in systems, and another one is malicious code modifies the data owned by process after this process accesses the objects contaminated by the malicious code. Note that the object denotes computer system resources (e.g., file).

Assumptions: We assume that network server and client programs contain bugs and can be exploited if the attacker is able to feed input to them. We assume that users may make careless mistakes in their actions, e.g., downloading a malicious program from the Internet and running it. However, we assume that the attacker does not have physical access to the host to be protected. SecGuard aims at ensuring that under most attacks, the attacker can only get limited privileges (not root information) and cannot compromise the operating system integrity.

Threat Scenarios: Due to aiming at protecting the operating system integrity against network-based attacks, SecGuard will confront threat from network. Moreover, we also need to consider what a process receives Inter-Process Communications (IPC) from another local process, since those IPCs can be used to send free-formed data to make integrity contamination. Finally, we should consider the threat from those contaminative files. In summary, SecGuard needs to consider three threat scenarios:

- *Network-based Threat:* Because the applications of system may contain some bugs, the attackers can make use of the network to plant malicious code into our host. Although the attackers will not launch the active attack, careless users may also download the malicious code into their local hosts from insecure network.
- *IPC Communication Threat:* When two processes communicate with each other, one process will read the IPC object owned by the other process. However, the IPC object may contain the malicious code which can destroy the integrity of systems.
- *Contaminative File Threat:* The most common way to destroy the integrity of system is one particular

process may read the system objects carrying malicious code, and thus the data owned by this process can be modify by the malicious code.

3 SecGuard Model

In this section, we discuss all the details of SecGuard. Because we have implemented the SecGuard model as a prototype protection system for Linux using the Linux Security Module (LSM) framework [22], the descriptions of the SecGuard model in this section is based on our design for Linux. Meanwhile, we believe that our model can be applied to other UNIX variants with minor changes.

3.1 The Integrity Labels of SecGuard

SecGuard assigns subjects with two integrity labels (Normally, subjects denote processes in the system). Two labels are *important integrity label of subject*, $s\_i(s)$, and *current integrity label of subject*, $s\_c(s)$, respectively. Both of $s\_i(s)$ and $s\_c(s)$ have two levels (values): high or low. Meanwhile, SecGuard also assigns objects (Normally, objects denote system resources such as files) with two integrity labels. They are *important integrity label of object*, $o\_i(o)$, and *current integrity label of object*, $o\_c(o)$, respectively. The same as $s\_i(s)$ and $s\_c(s)$, both of $o\_i(o)$ and $o\_c(o)$ have two levels: high or low. Note that we define *important integrity level* as the level of important integrity label, and define *current integrity level* as the level of current integrity label.

Why SecGuard assigns both subjects and objects two integrity labels? The same as existing MAC models, current integrity label is mainly used for the mandatory access control policy of SecGuard. The purpose of introducing important integrity label is in order to assign the identity for both subjects and objects, and this label can be used to record the authorities both of subjects and objects in the system. For instance, when administrator would like to raise the current integrity level for some processes, he needs to refer to the important integrity level of these processes, since we think, for the process whose important integrity level is low, its current integrity level can not be raised (This will be mentioned in Section 3.3). Therefore, we say the important integrity label is the identity for both subjects and objects.

How to initialize the levels of integrity labels for subjects and objects? In SecGuard, only the important integrity levels of root processes (system-level processes) are high; meanwhile, their current integrity levels are high in the startup. Normal processes' important integrity levels should be set low, and their current integrity levels are also low. When a process (subject) is created, it will inherit both the important integrity level and current integrity level from its parent process. In sub-process's life cycle, its important integrity level can not be changed. On the other hand, the current integrity level of subject can be changed dynamically according to the security policy of SecGuard model (The detail mentioned in Section 3.3). We will discuss the initialization scheme for objects in Section 3.2.

Algorithm 1: Initialization Algorithm

```
1  for each object_(i) in DAC do
2      if the other-bits(9-bits) of object_(i) is writable then
3          $object_{(i)}.s\_i \leftarrow low$;
4          $object_{(i)}.s\_c \leftarrow low$;
5          continue;
6      if the group-bits(9-bits) of object_(i) is writable then
7          for each user_(j) in group of object_(i) do
8              if $user_{(j)}.s\_i = low$ && $user_{(j)}.s\_c = low$ then
9                  $object_{(i)}.s\_i \leftarrow low$;
10                 $object_{(i)}.s\_c \leftarrow low$;
11                 break;
12         continue;
13     if the user-bits(9-bits) of object_(i) is writable then
14         $object_{(i)}.s\_i \leftarrow object_{(i)}.owner.s\_i$;
15         $object_{(i)}.s\_c \leftarrow object_{(i)}.owner.s\_c$;
```

3.2 SecGuard Initialization Algorithm

SecGuard proposes a novel *Initialization Algorithm* for objects in the system. The algorithm can utilize the existing DAC information of system to initialize the configuration of integrity level for objects. Note that we only pay attention to the 9-bits mechanism for DAC, and the current DAC enhanced by ACL mechanism is not our consideration, since the information provided by ACL mechanism is not used by Initialization Algorithm.

To elaborate the Initialization Algorithm clearly, we present the meanings of symbols of algorithm in Table 1. The Initialization Algorithm encompasses three key steps to initialize the configuration of integrity level for objects in the system.

Step 1: The algorithm checks the *other-bit* of *9-bits* of each object in the system. For the object whose other-bit is writable, both the important integrity level and current integrity level of the object are set to low. The object whose integrity level does not be changed enters Step2. For example, there are two object O_1 and O_2 in the system. If the other-bit of 9-bits of O_1 is writable, both the

Table 1: The Definitions of Symbols of Initialization Algorithm

Symbols	***Meaning***
DAC	DAC information list of system
$other-bits(9-bits)$	the 7th, 8th, and 9th bits of 9-bits for DAC authority
$group-bits(9-bits)$	the 4th, 5th, and 6th bits of 9-bits for DAC authority
$user-bits(9-bits)$	the 1st, 2nd, and 3rd bits of 9-bits for DAC authority
$object_{(i)}.s\_i$	the important integrity level of $object_{(i)}$
$object_{(i)}.s\_c$	the current integrity level of $object_{(i)}$
$user_{(j)}.s\_i$	the important integrity level of $user_{(j)}$
$user_{(j)}.s\_c$	the current integrity level of $user_{(j)}$
$object_{(i)}.owner.s\_i$	the important integrity level of the user who is the owner of $object_{(i)}$
$object_{(i)}.owner.s\_c$	the current integrity level of the user who is the owner of $object_{(i)}$

important integrity level and current integrity level of O_1 are set to low and low. If the other-bit of 9-bits of O_2 is not writable, O_2 enters Step2.

Step 2: The algorithm searches all users of the *user group* of the each object which enters this step, and if there is a user whose important integrity level and current integrity level are both low, both the important integrity level and current integrity level of this object are set to low. The object whose integrity level does not be set enters Step 3. For example, there is object O in the system, and user U is the user in O's user group. If both the important integrity level and current integrity level of U are low, both the important integrity level and current integrity level of O are set to low; otherwise, O enters Step3.

Step 3: In this step, the algorithm sets both the important integrity level and current integrity level of the object, which enters this step, according to both the important integrity level and current integrity level of the owner of the object. For example, in the Step3, there are object O and its owner user U in the system. If both the important integrity level and current integrity level of U are high and low respectively, both the important integrity level and current integrity level of O are set to high and low respectively.

The above three steps show the executing process of Initialization Algorithm. Algorithm 1 presents the details of Initialization Algorithm.

3.3 Security Policies of SecGuard

The Policies of Access/Read/Write: In SecGuard, when a subject accesses an object or communicates with a subject, the accessed object or subject must be in available state; If a subject can read an object, the current integrity level of the subject must dominate the current integrity level of the object and the important integrity level of the subject must dominate the important integrity level of the object; If a subject can modify an object, the current integrity level of the subject must dominate the important integrity level of the object.

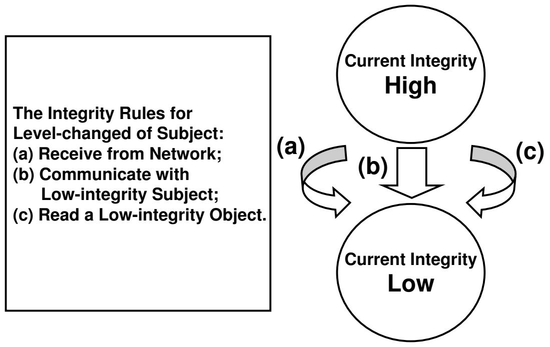
The Integrity Rules for
Level-changed of Subject:
(a) Receive from Network;
(b) Communicate with Low-integrity Subject;
(c) Read a Low-integrity Object.
Current Integrity High
Current Integrity Low
(a)
(b)
(c)

Figure 1: Security Policies of SecGuard

As shown in Fig. 1: (1) When a subject receives the traffic from network, the subject drops its current integrity level to low; (2) After a subject whose current integrity level is high communicates with a subject whose current integrity level is low, the former drops its current integrity level to low; (3) When a subject whose current integrity level is high reads an object whose current integrity level is low, the subject drops its current integrity level to low.

Partially Trusted Subject: To ensure the availability of the SecGuard model, we introduce the concept of partially trusted subject. Both the important integrity level and current integrity level of partially trusted subject are high, and it can keep both the integrity level constant during its access, read or write. In other words, the partially trusted subject can violate the security policies of the SecGuard. Meanwhile, the partially trusted subject can upgrade the current integrity level (from low to high) of subject whose important integrity level is high. The reason of allowing the partially trusted subject to upgrade current integrity level is that, after one subject (e.g., both the important integrity level and current integrity level of the subject are high) drops its integrity level, it may not execute some operations which could be operated in the past by this subject. The conditions of partially trusted subject upgrading the current integrity level of subject are that the important integrity level of the subject upgraded is high, and its current integrity level is low. The others are not in the scope of upgrading.

4 Formal Guarantees for SecGuard

SecGuard provides strong provable guarantees. In this section, we use formal methods to prove the security of SecGuard. Table 2 shows the definitions of constants and the states of SecGuard model, and the following sections present the invariants of SecGuard, the constraints of SecGuard, and proof of SecGuard.

4.1 The Invariants of SecGuard

$SecGuard\_C_1$ (Usable Property): A subject is usable, if it can access other objects or subjects:

$\forall o \in O, \forall s, s_1 \in S, (s,o,\underline{x}) \vee (s,s_1,\underline{x}) \in b \Rightarrow (U_o(o), U_s(s)) \vee (U_s(s_1), U_s(s))$.

Where, $(s,o,\underline{x})$ is a specific state of system. The definition of general state of system, $(V, R \times D, \tau, v_0)$, is in the study [18]. Due to the space constraints, we do not show the details of formal symbols of system (Details shown in the study [18]).

$SecGuard\_C_2$ (Simple Security Property): A subject can read an object, if subject's important integrity level and current integrity level dominate the important integrity level and current integrity level of the object respectively:

$\forall o \in O, \forall s \in S, (s,o,\underline{o}) \in b \Rightarrow s\_i(s) \trianglerighteq o\_i(o) \wedge s\_c(s) \trianglerighteq o\_c(o)$.

$SecGuard\_C_3$ (*-Security Property): The operation (including observe, signal, modify, and copy) between subject and object/subject are defined as follows:

- $(s_1,s_2,\underline{i}) \in b \Rightarrow ((s\_i(s_1) \trianglerighteq s\_i(s_2)) \wedge (s\_c(s_1) \trianglerighteq s\_c(s_2))) \vee ((s\_c(s_1) = s\_c(s_2)) \wedge (s\_i(s_2) \triangleright s\_i(s_1)))$;
- $(s,\underline{o}) \in M(o) \Rightarrow (s\_i(s) \trianglerighteq o\_i(o)) \wedge (s\_c(s) \trianglerighteq o\_c(o))$;
- $(s,\underline{m}) \in M(o) \Rightarrow s\_c(s) \trianglerighteq o\_i(o)$;
- $(s,\underline{c}) \in M(o) \wedge network(o) \Rightarrow s\_c(s) \trianglerighteq o\_i(o)$.

$SecGuard\_C_4$ (Discretionary Security Property): The DAC of SecGuard model is defined as follows:

$(s,o,\underline{x}) \in b \Rightarrow (o,s,\underline{x}) \in M$.

$SecGuard\_C_5$ (Integrity Control Property): In any condition, the integrity control policies of SecGuard are as follows:

- $\forall s \in S, s\_i(s) \trianglerighteq s\_c(s)$;
- $\forall o \in O, o\_i(o) \trianglerighteq o\_c(o)$.

4.2 The Constraints of SecGuard

$SecGuard\_CT_1(IntegrityLevelDynamicRules)$:

1. If $(s,\underline{o}) \in M(o)$ and s is not partially trusted subject, according to $SecGuard\_C_3$, both s and o satisfy: $s\_i(s) \trianglerighteq o\_i(o) \wedge s\_c(s) \trianglerighteq o\_c(o)$. When s observes o, the integrity levels are changed as follows: $b^* = b \cup (s,o,\underline{o}), o\_i^*(o) = o\_i(o), o\_c^*(o) = o\_c(o), s\_i^*(s) = s\_i(s), s\_c^*(s) = o\_c(o)$;
2. If $(s,\underline{c}) \in M(o) \wedge network(o)$ and s is not a partially trusted subject, according to $SecGuard\_C_3$, s and o satisfy: $s\_c(s) \trianglerighteq o\_i(o)$. When s downloads o from network to local, the integrity levels are changed as follows: $b^* = b \cup (s,o,\underline{c}), o\_i^*(o) = o\_i(o), o\_c^*(o) = o\_c(o), s\_i^*(s) = s\_i(s), s\_c^*(s) = o\_c(o)$;
3. If $(s_1,s_2,\underline{i}) \in b$ and s is not a partially trusted subject, according to $SecGuard\_C_3$, s_1 and s_2 satisfy: $((s\_i(s_1) \trianglerighteq s\_i(s_2)) \wedge (s\_c(s_1) \trianglerighteq s\_c(s_2))) \vee ((s\_c(s_1) = s\_c(s_2)) \wedge (s\_i(s_2) \triangleright s\_i(s_1)))$
   (a) If $(s_1,s_2,\underline{i}) \in b \wedge ((s\_i(s_1) \trianglerighteq s\_i(s_2)) \wedge (s\_c(s_1) \trianglerighteq s\_c(s_2)))$, then $b^* = b \cup (s_1,s_2,\underline{i}), s\_i^*(s_1) = s\_i(s_1), s\_i^*(s_2) = s\_i(s_2), s\_c^*(s_1) = s\_c(s_1), s\_c^*(s_2) = s\_c(s_2)$;
   (b) If $(s_1,s_2,\underline{i}) \in b \wedge ((s\_c(s_1) = s\_c(s_2)) \wedge (s\_i(s_2) \triangleright s\_i(s_1)))$, then $b^* = b \cup (s_1,s_2,\underline{i}), s\_i^*(s_1) = s\_i(s_1), s\_i^*(s_2) = s\_i(s_2), s\_c^*(s_1) = s\_c(s_1), s\_c^*(s_2) = s\_c(s_1)$.
4. If $(s,\underline{m}) \in M(o)$, according to $SecGuard\_C_3$, s and o satisfy: $s\_c(s) \trianglerighteq o\_i(o)$, the state of model remains constant.

Because we introduce the concept of partially trusted subject, two integrity levels of the partially trusted subject can remain constant in the course of accessing object of different integrity levels.

$SecGuard\_CT_2(DiscretionaryAccessControl)$: A subject access an object, if it has the discretionary access right of the object:

$\forall s \in S, \forall o \in O, \underline{x} \in OPERATION, (s,o,\underline{x}) \in b^* - b \wedge \neg IPC(o) \Rightarrow (s,\underline{x}) \in M(o)$.

$SecGuard\_CT_3$ (Create and Remove of Subject):

The condition of creating subject is:

$\forall s \in S, U_s^*(s) \wedge \neg U_s(s) \Rightarrow create\_cast(s) = TCB \vee s\_c(create\_cast(s)) \trianglerighteq s\_c(s) \wedge s\_i(create\_cast(s)) \trianglerighteq s\_i(s)$.

The condition of removing subject is:

Table 2: The Definitions of Constants and States

Constant and State	***Meaning***
$SUBJECT$	subject constant
$OBJECT$	object constant
$OPERATION$	operation constant
$LEVEL$	integrity levels constant
$BOOL$	Bool constant
S	the set of subject
O	the set of object
I	the set of integrity labels
$\_ \trianglerighteq \_$	more-than-or-equal relationship on I
$\_ \triangleright \_$	more-than relationship on I
M	the permission of access control
$\underline{\text{o}}$	the operation that a subject observes an object
$\underline{\text{i}}$	the operation that a subject signals to another subject
$\underline{\text{m}}$	the operation that a subject modifies an object
$\underline{\text{c}}$	the operation that a subject copies an object
Predicate and Function	***Meaning***
$create\_cast(o)$,$o \in O$	create object o
	return the subject that creates o
$create\_cast(s)$,$s \in S$	create subject s
	return the subject that creates s
$delete\_cast(o)$,$o \in O$	remove object o
	return the subject that removes o
$delete\_cast(s)$,$s \in S$	remove subject s
	return the subject that removes s
$network(o)$:$O \rightarrow BOOL$	the object o comes from network
$IPC(o)$:$O \rightarrow BOOL$	the object o is IPC object
U_o:$O \rightarrow BOOL$	object o is usable
U_s:$S \rightarrow BOOL$	subject s is usable
$b \subseteq S \times O \times OPERATION$	
$\cup S \times S \times OPERATION$	the condition of access currently

$\forall s \in S, U_s^*(s) \wedge \neg U_s(s) \Rightarrow delete\_cast(s) = TCB \vee s\_c(delete\_cast(s)) \trianglerighteq s\_c(s) \wedge s\_i(delete\_cast(s)) \trianglerighteq s\_i(s)$.

SecGuard_CT$_4$ (Create and Remove of Object): If an object is not IPC, the condition of creating this object is:

$\forall o \in O, \exists s \in S, [(\neg U_o(o) \wedge U_o^*(o)) \vee (U_o(o) \wedge \neg U_o^*(o))] \wedge \neg IPC(o) \Rightarrow s\_i(s) \trianglerighteq o\_c(o)$.

If an object is unusable and it is *IPC*, it can only be created by TCB:

$\forall o \in O, \neg U_o(o) \wedge U_o^*(o) \wedge IPC(o) \Rightarrow delete\_cast(o) = TCB$.

If an object is usable and it is *IPC*, the condition of removing this object should satisfy that the subject which removes this object is *TCB* or the important integrity level of subject which removes this object is not lower than the current integrity level of the object removed:

$\forall o \in O, IPC(o) \wedge \neg U_o(o) \wedge U_o^*(o) \Rightarrow delete\_cast(o) = TCB \vee (\exists s \in S \wedge s\_i(s) \trianglerighteq o\_c(o))$.

Next, we will proof some important theorems for SecGuard model; however, due to the space constraints, here we only present two representative proofs.

4.3 The Proofs of SecGuard

Theorem 1: If o_1 and o_2 are both objects and s is not a partially trusted subject, and $(s, o_1, \underline{o}) \in b, (s, o_2, \underline{m}) \in b$. The invariant and constraint of the model should satisfy: $o\_i(o_1) \trianglerighteq o\_i(o_2)$.

Proof:

$\because (s, o_1, \underline{o}) \in b$

$\therefore s\_c(s) \trianglerighteq o\_c(o_1)$ $(SecGuard\_C_3)$

$\therefore s\_c^*(s) = o\_c(o_1)$ $(SecGuard\_CT_1)$

$\because (s, o_2, \underline{m}) \in b$

$\therefore s\_c^*(s) \trianglerighteq o\_i(o_2)$ $(SecGuard\_C_3)$

$\because s\_c^*(s) = o\_c(o_1)$

$\therefore o\_c(o_1) \trianglerighteq o\_i(o_2)$ $(Transitivity)$

$\therefore o\_i(o_1) \trianglerighteq o\_i(o_2)$ $(SecGuard\_C_5)$

Proof finished

Theorem 2 (Access Stable Proof): For the invariant and constraint of our model, there are: $(\forall s \in S, \forall o \in O, \underline{x} \in \{\underline{o}, \underline{m}\}), (s, o, \underline{x}) \in b \wedge (s, o, \underline{x}) \in b^* \Rightarrow (s, o, \underline{x})$ although under the new state, model satisfies both invariant and constraint. In other words, if an access operation has not been revoked in the new state, the access operation can be executed all the same in the new state.

Proof: We only discuss that s is not a partially trusted subject, since if s is a partially trusted subject, its integrity level will not be changed after access operation; therefore, the partially trusted subject must satisfy Theorem 2. We define that the initial state of current integrity level of subject s is $s\_c^0(s)$, and $s\_c^1(s)$ is the state that after access operation.

(1) $\underline{x} = \underline{o}$:

$\because (s, o, \underline{o}) \in b$

$\therefore s\_c^0(s) \trianglerighteq s\_c^1(s)$ $(SecGuard\_CT_1)$

$\therefore s\_c^0(s) \trianglerighteq o\_c(o)$ $(SecGuard\_C_3)$

$\therefore s\_c^1(s) = o\_c(o)$

The model satisfies *SecGuard_C*$_3$ all the same in the new state.

(2) $\underline{x} = \underline{m}$:

$\because (s, o, \underline{m}) \in b$

$\therefore s\_c^1(s) = s\_c^0(s)$ $(SecGuard\_CT_1)$

$\therefore s\_c^0(s) \trianglerighteq o\_i(o)$ $(SecGuard\_C_3)$

$\therefore s\_c^1(s) \trianglerighteq o\_i(o)$

The model satisfies *SecGuard_C*$_3$ all the same in the new state.

Proof finished

5 The Implementation of SecGuard

We have implemented the SecGuard model as a prototype protection system for Linux, using the Linux Security Module (LSM) framework. We have been using the system involving the prototype in our lab for a few months. Due to the space constraints, we only present some representative designs and evaluations.

Implementation: The basic implementation of SecGuard is as follows. Each process has two integrity labels, and when the one is created, it inherits the important integrity level from its parent process. SecGuard can not restrict the process whose current integrity level is high; however a low current integrity process can not perform any sensitive operation in the system. If a process can send a request, it must be authorized by both the DAC of system and our SecGuard. Due to the space constraints, we cannot present details for the implementation of SecGuard.

Evaluation of High-availability: To evaluate the availability of SecGuard model, we established a server using *Fedora Core 6* with kernel version 2.6.18, and enabled SecGuard as a security module loaded during the system boot. The existing programs of the system have not been affected after our security module loading, and then we installed some commonly used applications and the system can still provide the services to our lab for a few months. SecGuard contributes several features of high availability on the operating system: the novel initialization algorithm, without complex integrity levels, and existing application programs and common practices for using can still be used under SecGuard.

Evaluation of Security: To evaluate the security of SecGuard model, we make use of Linux Rootkit Family (LRK) to attack our system. The LRK is a well-known rootkit of user-mode and it can replace many system programs and introduce some new programs to build backdoors and to hide the adversaries. LRK can be installed

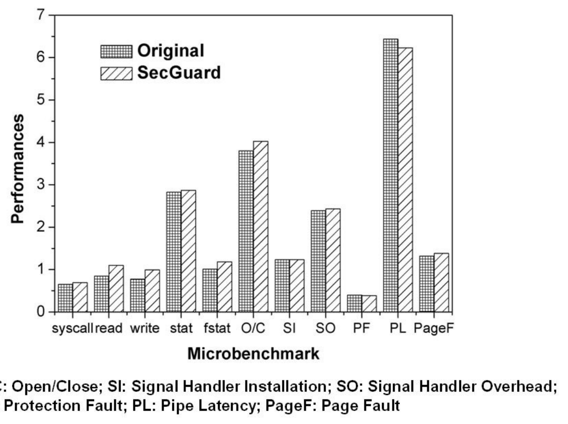

Figure 2: The performance results of lmbench 3 measurements

successfully and replaces the current SSH daemon in the system as soon as SecGuard was closed. Then, we can connect to the server as root with the predefined password. When our SecGuard was enabled, installation is failed and system returns a permitted error. Thus, our system remained security under SecGuard.

Performance: We have conducted benchmarking tests to compare performance overhead incurred by our protection system (the operating system which has installed SecGuard). We use Lmbench 3 benchmark and the Unixbench 4.1 benchmark suites. We believe that these tests were used to determine the performance overhead incurred by the protection system for different process, file, and operations. The experimental results are given in Table 3 and Fig. 2. We compare our result with SELinux [3]. The performance data of SELinux is taken from [14]. For most benchmark results, the percentage overhead is small($\leq 5\%$). The performance of our model is significantly better than the data for SELinux.

6 Conclusion

This paper presents SecGuard, a novel secure and high-available integrity protection model for operating systems. Aiming to three threats in systems: network-based threat, IPC communication threat, and contaminative file threat, SecGuard provides a robust defense for operating systems, and leverages information in the existing discretionary access control mechanism to initialize integrity labels both for the processes and files in the systems. Furthermore, SecGuard provides strong guarantees for the security of the model in theory. Finally, we describe the implementation of the SecGuard for Linux and evaluations.

References

[1] Apparmor application security for linux, http://www.novell.com/linux/security/apparmor/.

[2] LIDS: Linux intrusion detection system, http://www.lids.org/.

[3] NSA: Security enhanced linux, http://www.nsa.gov/selinux/.

[4] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat. A domain and type enforcement UNIX prototype. In *USENIX Security Symposium*, 1995.

[5] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghighat. Practical domain and type enforcement for UNIX. In *IEEE Symp. on Security and Privacy*, 1995.

[6] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. Gligor. Subdomain: Parsimonious server security. In *Conf. on Systems Administration*, 2000.

[7] T. Fraser. LOMAC: Low water-mark integrity protection for COTS environments. In *IEEE Symp. on Security and Privacy*, 2000.

[8] I. Goldberg, D. Wagner, R. Thomas, and E. A. Brewer. A secure environment for untrusted helper applications: Confining the wily hacker. In *USENIX Security Symposium*, 1996.

[9] Nike Gui, Ennan Zhai, Jian-bin Hu, and Zhong Chen. SWORDS: improving sensor networks immunity under worm attacks. In *Web-Age Information Management, 11th International Conference, WAIM 2010, Jiuzhaigou, China, July 15-17, 2010. Proceedings*, pages 86–96, 2010.

[10] Jianchun Jiang, Liping Ding, Ennan Zhai, and Ting Yu. Vrank: A context-aware approach to vulnerability scoring and ranking in SOA. In *Sixth International Conference on Software Security and Reliability, SERE 2012, Gaithersburg, Maryland, USA, 20-22 June 2012*, pages 61–70, 2012.

[11] R. Jueneman. Integrity controls for military and commercial applications. In *Conf. Aerospace Computer Security Applications*, 1988.

[12] N. Li, Z. Mao, and H. Chen. Usable mandatory integrity protection for operating systems. In *IEEE Symp. on Security and Privacy*, 2007.

Table 3: The Performance Results of Unixbench 4.1 Measurements

Benchmark	**Base**	**SecGuard**	**Overhead(%)**	**SELinux(%)**
Dhrystone	335.8	330.1	1.6	
Double-Precision	211.9	211.1	0.3	
Execl Throughput	616.6	605.1	1.8	5
File Copy 1K	474.1	450.2	5.0	5
File Copy 4K	505.1	480.1	4.9	2
Shell Scripts	649.1	630.2	2.9	4
System Call	215.8	215.1	0.2	
Overall	446.6	435.0	3	

[13] Bo Liu, Ennan Zhai, Huiping Sun, Yelu Chen, and Zhong Chen. Filtering spam in social tagging system with dynamic behavior analysis. In *2009 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2009, 20-22 July 2009, Athens, Greece*, pages 95–100, 2009.

[14] P. Loscocco and S. Smalley. Integrating flexible support for security policies into the Linux operating system. In *FREENIX track: USENIX Annual Technical Conference*, 2001.

[15] N. Provos. Improving host security with system call policies. In *USENIX Security Symposium*, 2003.

[16] Mark Santolucito, Ennan Zhai, and Ruzica Piskac. Probabilistic automated language learning for configuration files. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, pages 80–87, 2016.

[17] Cong Sun, Ennan Zhai, Zhong Chen, and Jianfeng Ma. A multi-compositional enforcement on information flow security. In *Information and Communications Security - 13th International Conference, ICICS 2011, Beijing, China, November 23-26, 2011. Proceedings*, pages 345–359, 2011.

[18] Inc. Trusted Information System. Trusted mach mathematical model. In *Technical Report, TIS tmach EDOC-0017-96B, Trusted Information System, Inc.*, 1996.

[19] Yonggang Wang, Ennan Zhai, Cui Cao, Yongqiang Xie, Zhaojun Wang, Jian-bin Hu, and Zhong Chen. Dspam: Defending against spam in tagging systems via users' reliability. In *16th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2010, Shanghai, China, December 8-10, 2010*, pages 139–146, 2010.

[20] Yonggang Wang, Ennan Zhai, Jian-bin Hu, and Zhong Chen. Claper: Recommend classical papers to beginners. In *Seventh International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2010, 10-12 August 2010, Yantai, Shandong, China*, pages 2777–2781, 2010.

[21] Yonggang Wang, Ennan Zhai, Eng Keong Lua, Jian-bin Hu, and Zhong Chen. isac: Intimacy based access control for social network sites. In *9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing, UIC/ATC 2012, Fukuoka, Japan, September 4-7, 2012*, pages 517–524, 2012.

[22] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman. Linux security modules: General security support for the linux kernel. In *USENIX Security Symposium*, 2002.

[23] Ennan Zhai, Ruichuan Chen, Zhuhua Cai, Long Zhang, Eng Keong Lua, Huiping Sun, Sihan Qing, Liyong Tang, and Zhong Chen. Sorcery: Could we make P2P content sharing systems robust to deceivers? In *Proceedings P2P 2009, Ninth International Conference on Peer-to-Peer Computing, 9-11 September 2009, Seattle, Washington, USA*, pages 11–20, 2009.

[24] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. An untold story of redundant clouds: making your service deployment truly reliable. In *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems, HotDep 2013, Farmington, Pennsylvania, USA, November 3, 2013*, pages 3:1–3:6, 2013.

[25] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. Heading off correlated failures through independence-as-a-service. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, pages 317–334, 2014.

[26] Ennan Zhai, Liping Ding, and Sihan Qing. Towards a reliable spam-proof tagging system. In *Fifth International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2011, 27-29 June, 2011, Jeju Island, Korea*, pages 174–181, 2011.

[27] Ennan Zhai, Liang Gu, and Yumei Hai. A risk-evaluation assisted system for service selection. In *2015 IEEE International Conference on Web Services, ICWS 2015, New York, NY, USA, June 27 - July 2, 2015*, pages 671–678, 2015.

[28] Ennan Zhai, Zhenhua Li, Zhenyu Li, Fan Wu, and Guihai Chen. Resisting tag spam by leveraging implicit user behaviors. *PVLDB*, 10(3):241–252, 2016.

[29] Ennan Zhai, Qingni Shen, Yonggang Wang, Tao Yang, Liping Ding, and Sihan Qing. Secguard: Secure and practical integrity protection model for operating systems. In *Web Technologies and Applications - 13th Asia-Pacific Web Conference, APWeb 2011, Beijing, China, April 18-20, 2011. Proceedings*, pages 370–375, 2011.

[30] Ennan Zhai, Huiping Sun, Sihan Qing, and Zhong Chen. Spamclean: Towards spam-free tagging systems. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009, Vancouver, BC, Canada, August 29-31, 2009*, pages 429–435, 2009.

[31] Ennan Zhai, Huiping Sun, Sihan Qing, and Zhong Chen. Sorcery: Overcoming deceptive votes in P2P content sharing systems. *Peer-to-Peer Networking and Applications*, 4(2):178–191, 2011.

[32] Ennan Zhai, David Isaac Wolinsky, Hongda Xiao, Hongqiang Liu, Xueyuan Su, and Bryan Ford. Auditing the Structural Reliability of the Clouds. Technical Report YALEU/DCS/TR-1479, Department of Computer Science, Yale University, 2013. Available at `http://www.cs.yale.edu/homes/zhai-ennan/sra.pdf`.