Automatic Analysis Method for SELinux Security Policy

Gaoshou Zhai and Tong Wu

School of Computer and Information Technology, Beijing Jiaotong University
gszhai@bjtu.edu.cn

Abstract

Configuration of security policies is one of the most important prerequisites for secure and credible running of secure operating systems. Although it is a hard, tedious and complicated task within which errors and bugs are incidental at all time. Accordingly, methods for automatic analysis of SELinux security policies are discussed in this paper. Firstly, security mechanism, security models and policy description language for SELinux are briefly introduced. Then a security analysis model is constructed in order to verify validity and integrity of security policies and all rules for Type Enhancement (TE), Role-Based Access Control (RBAC) are rewritten as formal expressions while all subjects, objects and elements are described as sets and mappings formally. Algorithms for analysis are designed based on such model. Comparing with that in SELinux Access Control (SELAC) model, scope of possible values for role can be reduced and thus a great many invalid security contexts are eliminated in our model. Finally, a prototype is implemented in C language and a security policy configuration case as to an application system called Student-Teacher system is designed to be used to test the prototype. Test results show that the prototype and corresponding methods can verify validity and integrity of policy configuration and are potential to be used to assist people to complete correct and reliable configuration.

Keywords: *Secure Operating Systems; Access Control; SELinux; Security Policy; Analysis Method*

1. Introduction

Mandatory access control (MAC) mechanism is a necessary part of secure operating systems, which are the key foundation of security for information systems [1-5]. SELinux is one of the most excellent MAC mechanisms inside Linux and it is currently implemented as a loadable security application module based on Linux Security Modules (LSM) [6].

SELinux can enforce a policy based on robust mandatory access control and can be used cooperated with discretionary access control inside Linux kernel to implement effective control whenever a subject request to access an object. But it is hard for people to perform security policy configuration correctly and inerrably and such task is both time consuming and tedious. Therefore, it is rather significant to study automatic analysis method about security policy configuration so as to build appropriate computer-aided configuration tools [7].

In this paper, SELinux security mechanism and its policy description language are briefly discussed. Then the analysis model is built up to verify the validity and integrity of SELinux policies and formal representations for both security policy language and policy analysis goals are generated. Finally, a prototype is implemented using C language based on such model and corresponding algorithms while a case of security policy configuration as to an application system called Student-Teacher system is designed to be used to test the prototype.

2. Methodology

2.1. Security mechanism of SELinux

Nowadays, SELinux is implemented based on LSM (refer to Figure 1) and Flask (refer to Figure 2).

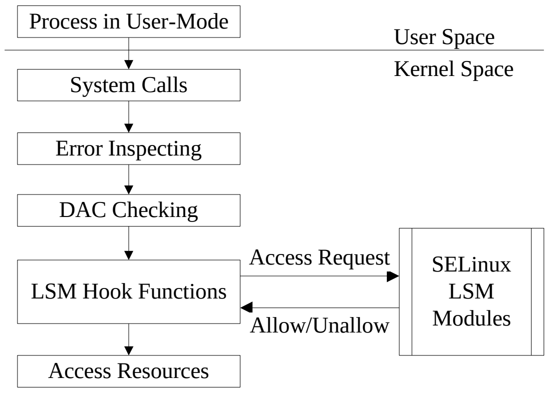

Figure 1. **Framework between LSM and SELinux**

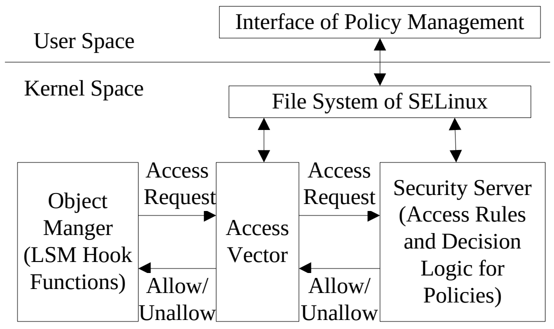
Interface of Policy Management
User Space
Kernel Space
File System of SELinux
Object Manger (LSM Hook Functions)
Access Request
Allow/ Unallow
Access Vector
Access Request
Allow/ Unallow
Security Server (Access Rules and Decision Logic for Policies)

Figure 2. **Flask framework of SELinux**

2.2. Security Policy Description of SELinux

SELinux supports three types of security models including Type Enhancement (TE), Role-Based Access Control (RBAC) and Multi-Level Security (MLS). Among SELinux policies, TE rules account for absolutely majority and RBAC rules account for minority while MLS rules are not optional for default. In addition, two types of logic structural rules, i.e. constraints rules and conditional rules are provided in SELinux policy configuration language. Therefore, rules except that for MLS are focused in this paper.

Policy configuration is much complicated and intractable. A few demonstration polices have been provided by system developers so as to make users' policy design more convenient, among which strict policy and targeted policy are rather widely used. And a new architecture for configuration files and directories as to so-called reference policy is established in order to improve modularization and maintainability [8].

2.3. Analysis Model for SELinux Security Policies

It is taken aim at validity and integrity for policy analysis in this paper, i.e. to make sure that the policy configuration has carried out expected access regulations and to verify that subjects inside Trusted Computing Base (TCB) are prohibited to read wrong information from non-trusted objects while sensible information inside TCB objects are protected from wrongly modified.

Thereafter, all rules for TE and RBAC are rewritten as formal expressions (refer to Table 1) while all subjects, objects and other elements for security policies are marked as sets and mappings formally (refer to Definition 1-10). In addition, Definition 11-24 are used to verify the validity of policies while Definition 25-31 are used to verify the integrity of policies.

Comparing with SELinux Access Control (SELAC) model [9], scope of possible values for *role* can be reduced and thus a great many invalid security contexts are eliminated in our model.

Table 1. Formal Expressions of Policy Rules

Policy rules	Formal expressions
attribute attribute_name;	attribute(attribute_name)
type type_name, a1, a2;	type(type_name, a1, a2)
role role_name **types** { t1 t2 };	role(role_name, t1 ,t2)
user user_name **roles** { r1 r2 };	user(user_name, r1 ,r2)
dominance { **role** r1 { r2 r3 }}	dom(r1, r2, r3)
class class_name { p1 p2 }	class(class_name, p1 , p2)
allow t1 t2:c p;	allow(t1 ,t2, (c, p))
constrain c p expression;	contrain((c, p), expression)
type_transition t1 t2:**process** t3;	type_transtion(t1 ,t2 ,t3)

Definition 1 For an object *a*, if there is a rule like $attribute(a)$, then *a* is judged as an attribute while all attributes are marked as the set $A=\{a \mid \forall a, attribute(a)\}$.

Definition 2 For an object *t*, if there is a rule like $type(t)$ or $type(t, a_1, \cdots, a_n)$ where $a_1, \cdots, a_n \in A$, then *t* is judged as a type while all types are marked as the set $T=\{t \mid \forall t, type(t) \vee type(t, a_1, \cdots, a_n)\}$.

Definition 3 For an object *r*, if there is a rule like $role(r)$ or $role(r, t_1, \cdots, t_n)$ where $t_1, \cdots t_n \in T$, then *r* is judged as a role while all roles are marked as the set $R=\{r \mid \forall r, role(r) \vee role(r, t_1, \cdots, t_n)\}$.

Definition 4 For $r, r_1, \cdots, r_n \in R$, if there is a rule like $dom(r, r_1, \cdots r_n)$, then $r_1, \cdots r_n$ is judged as dominated by r.

Definition 5 For an object *u*, if there is a rule like $user(u)$ or $user(u, r_1, \cdots, r_n)$ where $r_1, \cdots, r_n \in R$, then *u* is judged as a user while all users are marked as the set

$$U=\{u \mid \forall u, user(u) \vee user(u, r_1, \cdots, r_n)\}.$$

Definition 6 For objects *c* and $p_1, \cdots, p_n$, if there is a rule like $class(c)$ or $class(c, p_1, \cdots, p_n)$, then *c* is judged as a class while all classes are marked as the set $C=\{c \mid \forall c, class(c) \vee class(c, p_1, \cdots, p_n)\}$. And $p_1, \cdots, p_n$ and $(c, p_1), \cdots, (c, p_n)$ can be judged according to such rules like $class(c, p_1, \cdots, p_n)$ while all class permissions are marked as the set $Q=\{(c,p) \mid \forall c, class(c, p_1, \cdots, p_n), p \in \{p_1, \cdots p_n\}\}$.

Definition 7 For $\forall t \in T$, the mapping $A: T \mapsto A$ can be defined as $A(t)=\{a \mid type(t, A'), a \in A'\}$ where attribute set $A' \subseteq A$.

Definition 8 For $\forall t \in T$, the mapping $T_a: (A \cup T) \mapsto T$ can be defined as $T_a(t)=t$ while for $\forall a \in A$ it can be defined as $T_a(a)=\{t \mid type(t, A'), a \in A'\}$ where attribute set $A' \subseteq A$.

Definition 9 For $\forall r \in R$, the mapping $T_r: R \mapsto T$ can be defined as

$$T_r(r)=\{t \mid (role(r, T'), t \in T') \vee (dom(r, R'), r' \in R', t \in T(r'))\}.$$

Definition 10 For $\forall u \in U$, the mapping $R: U \mapsto R$ can be defined as

$$R(u)=\{r \mid user(u, R'), r \in R'\}$$ where users set $R' \subseteq R$.

Definition 11 Space for subject security contexts can be defined as the set

$$S=\{(u,r,t) \mid u \in U, r \in R(u), t \in T_r(r)\}.$$

Definition 12 Space for object security contexts (i.e. Whole space for security contexts) can be defined as the set $O=\{(u,r,t) \mid u \in U, r=\text{objetc\_r}, t \in T\} \cup S$ where *object_r* is some user specified for the object.

Definition 13 For $\forall o \in O$ and given that $o=(u,r,t)$, the mapping $Project_t^O: O \mapsto T$, $Project_r^O: O \mapsto R$ and $Project_u^O: O \mapsto U$ can be defined as $Project_t^O(o)=t$, $Project_r^O(o)=r$ and $Project_u^O(o)=u$ respectively.

Definition 14 For $\forall c_0 \in C$, the mapping $Q: C \mapsto Q$ can be defined as

$$Q(c_0) = \{(c_0, p) \mid class(c_0, p_1, \cdots, p_n), p \in \{p_1, \cdots p_n\}\}.$$

Definition 15 For $\forall t_0 \in T$, the mapping $M: T \mapsto (T \times Q)$ can be defined as

$$M(t_0) = \{(t, (c, p)) \mid allow(X_1, X_2, Q'), t_0 \in T_a(X_1), t \in T_a(X_2), (c, p) \in Q'\}$$

where $X_1, X_2 \subseteq (A \cup T)$ and $Q' \subseteq Q(C)$.

Definition 16 For $\forall s \in S$, $\forall o \in O$, $\forall c \in C$ and $\forall (c, p) \in Q(c)$, the mapping $Con: (S \times O \times Q) \mapsto \{true, false\}$ can be defined as follows:

$$Con(s, o, (c, p)) = \begin{cases} false & \text{if} \quad \begin{array}{l} \exists constrain(Q', expression) \\ expression(s, o) = false, (c, p) \in Q' \end{array} \\ true & \text{if} \quad Others \end{cases}$$

where the class permission set $Q' \subseteq Q$ while $expression(s, o)$ is logic expression for subject and object.

Definition 17 For $\forall t_0 \in T$, the mapping $N: T \mapsto T \times Q$ can be defined as

$$N(t_0) = \{(t, (c, p)) \mid auditallow(X_1, X_2, Q'), t_0 \in T_a(X_1), t \in T_a(X_2), (c, p) \in Q'\}$$

where $X_1, X_2 \subseteq (A \cup T)$ and $Q' \subseteq Q(C)$.

Definition 18 For $\forall x \in T \times Q$ and given that $x = (t, (c, p))$, the mapping $Project_q^{T \times Q}: (T \times Q) \mapsto Q$ and $Project_t^{T \times Q}: (T \times Q) \mapsto T$ can be defined as $Project_q^{T \times Q}(x) = (c, p)$ and $Project_t^{T \times Q}(x) = t$ respectively.

Definition 19 For $\forall t_0 \in T$, the mapping $D_p: T \mapsto T \times T$ can be defined as

$$D_p(t_0) = \{(t_e, t_p) \mid type\_transition(t_0, t_e, t_p), t_e, t_p \in T\}.$$

Definition 20 For $\forall t_p \in T$, the mapping $E: T \mapsto T$ can be defined as

$$E(t_p) = \{t_e \mid \exists x \in M(t_p), Project_t^{T \times Q}(x) = t_e, Project_q^{T \times Q}(x) = (file, entrypoint)\}.$$

Definition 21 For $\forall t \in T$, the mapping $T_{allow}: T \mapsto T \times T$ can be defined as follows:

$$\begin{aligned} T_{allow}(t) = \{(t_e, t_p) \in T \times T \mid \exists m, n \in M(t), & Project_t^{T \times Q}(m) = t_p, \\ & Project_q^{T \times Q}(m) = (process, transition), \\ & Project_t^{T \times Q}(n) = t_e, t_e \in E(t), \\ & Project_q^{T \times Q}(n) = (file, execute)\} \end{aligned}$$

Definition 22 For $\forall t_0 \in T$, the mapping $T_{tran}: T \mapsto T$ can be defined as

$$T_{tran}(t_0) = \{t_p \in T \mid \exists t_e \in E(t), (t_e, t_p) \in D_p(t_0) \cap T_{allow}(t_0)\}.$$

Definition 23 For $\forall s \in S$, the mapping $\Sigma: S \mapsto O \times Q$ can be defined as follows:

$$\begin{aligned} \Sigma(s) = \{(o, q) \in O \times Q \mid (Project_t^O(o), q) \in M(Project_t^O(s)), Con(s, o, q) = true, \\ T_{tran}(Project_t^O(s)) = Project_t^O(o) \text{ if } q = (process, transition)\} \end{aligned}$$

Definition 24 For $\forall o \in O$, the mapping $\Pi: O \mapsto S \times Q$ can be defined as follows:

$$\begin{aligned} \Pi(o) = \{(s, q) \in O \times Q \mid (Project_t^O(o), q) \in M(Project_t^O(s)), Con(s, o, q) = true, \\ T_{tran}(Project_t^O(s)) = Project_t^O(o) \text{ if } q = (process, transition)\} \end{aligned}$$

Definition 25 For $\forall o \in O$ and given that $G_{TCB}: T \mapsto \{true, false\}$ is single mapping, the mapping $G_{TCB}: O \mapsto \{true, false\}$ can be defined as follows:.

$$G_{TCB}(o) = \begin{cases} true & \text{if} \quad G_{TCB}(Project_t^O(o)) = true \\ false & \text{if} \quad G_{TCB}(Project_t^O(o)) = false \end{cases}$$

Definition 26 Define the set $T_{TCB} = \{t \in T \mid G_{TCB}(t) = true\}$ and the set

$$O_{TCB} = \{o \in O \mid G_{TCB}(o) = true\}.$$

Definition 27 Define the single mapping $F : Q \mapsto \{read, write, rw, none\}$ where *read*, *write* and *rw* represent reading, writing, reading & writing information flows from subject to object while *none* represents that there is no reading or writing information flows from subject to object.

Definition 28 For $\forall o_0 \in O$, the mapping $I_{direct} : O \mapsto O$ can be defined as follows:

$$\begin{aligned} I_{direct}(o_0) = \{o \in O \mid & (\exists(o,q) \in \Sigma(o_0), F(q) = read \vee F(q) = rw) \\ & \vee (\exists(o,q) \in \Pi(o_0), F(q) = write \vee F(q) = rw)\} \end{aligned}$$

Definition 29 For $\forall o_0 \in O$, the mapping $I_{all} : O \mapsto O$ can be defined as

$I_{all}(o_0) = \{o \mid o \in I_{direct}^{(n)}(o_0), n \to \infty\}$ where $I_{direct}^{(n)}$ represents invoke of I_{direct} for *n* times successively.

Definition 30 For $\forall o_0 \in O_{TCB}$, the mapping $I_{break} : O_{TCB} \mapsto O$ can be defined as

$$I_{break}(o_0) = \{o \in O \mid o \in I_{all}(o_0), o \notin O_{TCB}\}.$$

Definition 31 For $\forall o_0 \in O_{TCB}$, the mapping $I_{real} : O \mapsto O$ can be defined as follows:

$$\begin{aligned} I_{real}(o_0) = \{o \in O \mid o \in I_{direct}(o_0), o \notin O_{TCB}, o \notin Project_o^{O\times Q}(N(Project_t^O(o_0))), \\ o_0 \notin Project_o^{O\times Q}(N(Project_t^O(o)))\} \end{aligned}$$

3. Prototype and Results

Algorithms are designed based on above analysis model and a corresponding prototype is implemented in C language, which is made up of reference policy transformation module, security policy extract module, security policy analysis module and analysis result display module. In addition, a group of security policy modules are designed based on the architecture of reference policy as to so-called student-teacher system and are used to test the prototype. Test results show that the prototype not only can get all objects with corresponding permissions that any subject with specified security context <*user*, *role*, *type*> can access but also can get all subjects with corresponding permissions that any object with specified security context <*user*, *role*, *type*> can be accessed. Moreover, all rules that could potentially influence integrity of subjects and objects can be detected.

4. Summary

In this paper, an improved SELAC model is constructed and corresponding prototype is designed to perform automatic analysis of SELinux policies. And test results are satisfactory.

Nevertheless, some simplified process is done in this paper. For example, Boolean variables and a few special signs and macro blocks are ignored during the analysis. All these details ought to be full considered in the future research. In addition, both method and prototype for analysis must be improved farther for practicability.

Acknowledgements

The authors would like to give grateful thanks to the support of the Fundamental Research Funds for the Central Universities (No.2009JBM019).

References

[1] G. Zhai, J. Zeng, M. Ma and L. Zhang, "Implementation and Automatic Testing for Security Enhancement of Linux Based on Least Privilege", In: Proceedings of the 2nd International Conference on Information Security and Assurance (ISA 2008), pp.181-186. IEEE Computer Society, California **(2008)**.

[2] G. Zhai, J. Zeng, M. Ma and L. Zhang, "Implementation and Automatic Testing for Security Enhancement of Linux Based on Least Privilege", International Journal of Security and Its Applications, vol. 2, no. 3, pp. 93-100 **(2008)**.

[3] G. Zhai and Y. Li, "Analysis and Study of Security Mechanisms inside Linux Kernel", In: Proceedings of 2008 International Conference on Security Technology (SECTECH2008), pp. 58-61. IEEE Computer Society, California **(2008)**.

[4] G. Zhai and Y. Li, "Study and Implementation of SELinux-like Access Control Mechanism Based on Linux", In: T.-k. Kim, T.-h. Kim, A. Kiumi (eds.): SecTech 2008- Advances in Security Technology, CCIS (Communications in Computer and Information Science), vol. 29, pp. 50-66. Springer-Verlag Berlin, Berlin **(2009)**.

[5] G. Zhai, H. Niu, N. Yang, M. Tian, C. Liu and H. Yang, "Security Testing for Operating System and Its System Calls", In: D. Slezak et al. (Eds.): SecTech 2009, CCIS (Communications in Computer and Information Science), vol. 58, pp. 116-123. Springer-Verlag Berlin, Berlin **(2009)**.

[6] S. Smalley, C. Vance and W. Salamon, "Implementing SELinux as a Linux security module", Technical Report 01-043, NAI Labs **(2001)**.

[7] G. Zhai, W. Ma, M. Tian, N. Yang, C. Liu and H. Yang, "Design and Implementation of a Tool for Analyzing SELinux Secure Policy", In: Proceedings of 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (ICIS 2009), pp. 446–451. Association for Computing Machinery (ACM), New York **(2009)**.

[8] F. Mayer, K. MacMillan and D. Caplan, "SELinux By Example: Using Security Enhanced Linux", Prentice Hall **(2006)**.

[9] G. Zanin and L. V. Mancini, "Towards a Formal Model for Security Policies Specification and Validation in the SELinux System", In: Proceedings of the 9th ACM Symposium on Access Control Models and Technologies, pp. 136-145. Association for Computing Machinery (ACM), New York **(2004)**.

Authors

Gaoshou Zhai received the B.Sc., Master and Ph.D. degrees in 1993, 1996 and 2000 respectively. From 2000 to 2002, he was a lecturer in the School of Computer and Information Technology at Beijing Jiaotong University. Since 2002 he has been an associate professor and since 2007 he has been vice director of the Department of Computer Science. From January to May in 2006, he had been to the department of computer science at UIUC as a visiting scholar. His research interests include operating systems, information security, system software design and automatic tools for software engineering, algorithm analysis and design, artificial intelligence and intelligent traffic systems. In these areas he has published more than 40 papers in journals and conference proceedings. He served as program committee member of SERA2009 and invited review specialist for Chinese Science and Technology Papers Online sponsored by Centre for Science and Technology Development, MEPRC. He is also invited to review papers for Journal of Xi'an Jiaotong University, Journal of Beijing Jiaotong University, Journal of Lanzhou Jiaotong University, ICCIT2009 and ICSAI2012. He is a member of ACM and SERSC, a senior member of CCF and IACSIT.