SCIATool: A Tool for Analyzing SELinux Policies Based on Access Control Spaces, Information Flows and CPNs

Gaoshou Zhai[1(✉)], Tao Guo[1,2], and Jie Huang[1]

[1] School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China
{gszhai,lll20437,13120394}@bjtu.edu.cn
[2] Henan Center of Patent Examination Cooperation of the Patent Office SPIO, Henan, China

Abstract. Although security policies configuration is crucial for operating systems to constrain applications' operations and to protect the confidentiality and integrity of sensitive resources inside the systems, it is an intractable work for security administrators to accomplish correctly and consistently solely by hands. Thus policies analysis methods are becoming research hotspots. A great deal of such researches are focused on SELinux, which is a security-enhanced module of open-source and popular Linux. Among various analysis methods for SELinux policies, those based on access control spaces, information flows and colored Petri-nets (CPNs) can be thought as the three most valuable methods and they can be exploited together and complementarily. In this paper, a prototype of SELinux policies Configuration Integrated Analysis Tool, i.e. SCIATool, is designed and implemented by integrating these three methods together. Test results are provided and further researches as to construct a computer-aided configuration tool for SELinux policies are discussed.

Keywords: Security policies configuration · Analysis method · Access control spaces · Information flows · Colored Petri-nets · SELinux

1 Introduction

Security of operating systems is always the research focus in the fields of information security for their irreplaceable position inside the whole information systems. As Linux is becoming popular and powerful SELinux has been embedded into the Linux kernel, they are being research hotspots in domain of security of operating systems.

SELinux can enforce mandatory access control through policies configuration based on TE, RBAC and MLS models [1, 2] to defend against local and remote attacks and to protect systematic integrity and confidentiality and thus make Linux fulfill various security requirements for most situations. However, statements and rules of policies configuration are immense and complex because there are so many programs that can become potential subjects and so many objects including processes, files, devices, sockets and other resources of sorts inside computer systems. Moreover, subjects, objects and relationships among them are complicated and confused. Thereafter, it is difficult and

M. Yung et al. (Eds.): INTRUST 2014, LNCS 9473, pp. 294–309, 2015.
DOI: 10.1007/978-3-319-27998-5_19

error-prone for security administrators to accomplish the correct and consistent configuration manually without auxiliary means.

Computer-aided policies analysis is thus becoming an effective way to provide more helpful information for policies configuration. Such analysis typically includes TCB integrity analysis and validity analysis. Generally, the former is to find all rules that could potentially influence integrity of the initially specified TCB while the latter is to work out authorized or prohibited permissions as for a specified subject and/or a specified object as well as a specified information flow path so as to verify whether the policies configuration satisfies the security targets.

A lot of researches have been done as to policies analysis methods and those based upon access control spaces [3–6], information flow [5–8] and colored Petri-nets (CPNs) [9–12] reflect more practical values. In addition, the last method also depends on information flow query and verification and is essentially an information flow analysis method. Furthermore, all of these methods have their own limitation while more analysis targets or results will be more helpful for policies configuration. In details, the policy analysis method using access control method is convenient for designing security policies while the other two methods can be used to verify whether a special security goal is in practice enforced by corresponding policies configuration. In addition, the latter two methods can provide more detailed security requirements for validation by information flow inquiries. Therefore, they are complementary and can be exploited together.

In this paper, a prototype of SELinux policies Configuration Integrated Analysis Tool, i.e. SCIATool, is designed and implemented in C language by integrating methods based on access control spaces, information flows and colored Petri-nets. The main contributions of this paper are:

- The approach presented in this paper is, to the best of our knowledge, among the first efforts on systematic integration of such three methods for SELinux policies configuration analysis. And SCIATool is the first such prototype implemented independently without other software tools.
- Integrating different methods is always a challenging problem because it is not only to put them together but also to bring them into full play in a redundancy minimized framework. An integrated architecture is put forward for SCIATool so as to make full use of different analysis methods but minimize the redundant design.
- Both the TCB integrity analysis and various validity analyses can be achieved effectively by SCIATool. Various analysis targets such as all rules that could potentially influence integrity of the initially specified TCB, authorized or prohibited permissions as for a specified subject and/or a specified object as well as information flow path in CPN of policy configuration as for a specified information flow path requirement can be worked out, which provides richer analysis results than many available analysis prototypes and will be more helpful for computer-aided policies configuration.

The remaining part of the paper is organized as follows: Sect. 2 introduces SELinux policies configuration, typical analysis methods and main idea about our integrated analysis method; Sect. 3 describes our design and implementation of SCIATool; Sect. 4 tests and evaluates our SCIATool prototype with a SELinux policy configuration illustration as for some student-teacher application security requirements, then compares our research in this paper with related work of others; Sect. 5 summarize the research work in this paper and discusses the limitations of SCIATool and future work as to construct a practical computer-aided configuration tool for SELinux policies.

2 Methodology

2.1 SELinux Policies Configuration

SELinux policies configuration is made up of a series of policy source files. In another word, it is the collection of statements, i.e. rules that determine allowed access for a system, and it defines the roles any SELinux user may assume, the domains a role can access, the types any process can access and how. When a process tries to gain access to a particular object, a security decision has to be made whether the access is allowed or denied depending on the security context (i.e. $<user_i, role_j, type_k>$) of the subject, the security context of the object, and the corresponding policies.

SELinux has combined three different policy models in its policies configuration, where its RBAC model associates users with roles and roles to TE domains that are authorized to specific access permissions. All roles are used to constrain the association of users with the types of processes, except that the dummy role *object_r* is used in security contexts for all object types. The SELinux types are classified based on the functions performed by processes and the operations performed on the different objects. Types in SELinux can be classified into domain types, security types, device types, file types, procfs types, devpts types, nfs types and network types while domain types are special for processes and can be further classified into system domains, user program domains, and user login domains. In general, Security policy description language of SELinux provide users the following top-level components such as Flask definitions, TE and RBAC declarations and rules, user declarations, constraint definitions, and security context specifications for a policy configuration.

2.2 Typical SELinux Policies Analysis Methods

Among analysis methods for SELinux security policies, those based upon access control spaces, information flows and colored Petri-nets are the three most valuable types of methods that can be exploited in practical configuration.

Method Based on Access Control Spaces. The policy analysis method based on access control spaces is put forward by Trent Jaeger et al. and is firstly used in Gokyo [3] to analyze a SELinux policy configuration for ApacheWeb server system and the example policy of the SELinux for Linux 2.4.16. And a formal model called SELAC is developed on this basis by Giorgio Zanin and Luigi Vincenzo Mancini [4] for analyzing an arbitrary security policy configuration for the SELinux system.

Access control space is the core concept for this method, which is defined as the set of all possible permission assignments of a subject (or role) and can be divided into three natural subspaces: the permissible subspace (contains the permission assignments in the current configuration), the prohibited subspace (contains the permission assignments precluded by the constraints) and the unknown subspace (contains the permission assignments that are neither permitted nor prohibited). Ideally, these three subspaces should partition the access control space without intersection and the unknown subspace should be minimal. But in practice, subspaces are not disjoint and the unknown subspace is large. In another word, sometimes the specified space conflicts with the prohibited space and the unknown space. In addition, the permission assignments within the permissible subspace can be divided into two parts: one part is explicitly expressed in the configuration and the other part is not yet specified. The former part constitutes so-called the specified subspace and it contains a subset of the permissible assignments (accordingly named by the obligated subspace) that are obligated required for correct operation of the system.

Once access control spaces are constructed for a given SELinux policies configuration, all possible integrity conflicts embodied by the conflicting subspaces for TCB subjects can then be identified and classified for solving related conflicts. So the method can be used to help security administrator to accomplish custom-made SELinux policies configuration. However, it is focused on single special analysis target and it is not appropriate for analysis on multiple targets.

Method Based on Information Flows. The policy analysis method based on information flows is firstly developed and applied in SELinux policies for the e-commerce processing system by Guttman et al. [7, 8].

Information flow is the key conception for this method. If a subject with security context $<u_1, r_1, t_1>$ can write an object with security context $<u_2, r_2, t_2>$, we say that there is a *write-like* information flow transition from the subject with security context $<u_1, r_1, t_1>$ to the object with security context $<u_2, r_2, t_2>$. Similarly, if a subject with security context $<u_2, r_2, t_2>$ can read an object with security context $<u_1, r_1, t_1>$, we say that there is a *read-like* information flow transition from the object with security context $<u_1, r_1, t_1>$ to the subject with security context $<u_2, r_2, t_2>$. Both cases can be formalized as an information flow from security context $<u_1, r_1, t_1>$ to security context $<u_2, r_2, t_2>$ through event $<c, p>$ where c, p represents corresponding class and permission for the object. Furthermore, if there is an information flow from security context $<u_i, r_i, t_i>$ to security context $<u_{i+1}, r_{i+1}, t_{i+1}>$ through event $<c_i, p_i>$ for all $0 \le i \le n - 1$, it can be concluded that there is an information flow from security context $<u_0, r_0, t_0>$ to security context $<u_n, r_n, t_n>$ through event sequence $\{<c_i, p_i> \mid 0 \le i \le n - 1\}$. In addition, an information flow can be accepted if and only if all entities passed in the flow are trusted.

Once the information flow model for a given SELinux policies configuration is built up, information flow security goal statements for the objectives that SELinux is intended to achieve can be expressed in linear temporal logic and model checking method can be used to determine whether security goals hold in the given system. So the method is focused on the whole information flow path and can be used to validate whether the SELinux policies configuration is fully in accordance with the whole

desired security objectives of the system. At the same time, those expressions or inquiries for security goal statements to be validated are difficult to write and provide and thus its application is restricted to some extent. Moreover, it is not appropriate for designing or developing policies configuration directly.

Method Based on Colored Petri-Nets. The policy analysis method based on colored Petri-nets is firstly developed and applied in SELinux policies for the e-commerce processing system by Chen and Kao [9]. It is also used to model the trusted computing based secure systems [10].

This method is also developed based on information flows model thus it can be viewed as a special policy analysis method based on information flows. But it describes the SELinux policies configuration and security objectives in the way of colored Petri-nets instead of information flow graphs. It is obvious that colored Petri-net is the key conception for this method. Colored Petri-net (CPN) is formulated on the basis of traditional Petri-net concept by introducing color set. Furthermore, colored Petri-net can be formally defined by a tuple $CPN = <\Sigma,P,T,A,N_A,C_P,G_T,E_A,I_P>$ which satisfies the following requirements: (1) Σ is color set, i.e. a finite set of non-empty data types. Different color stands for different place sort, i.e. type place or permission place. (2) P is a finite set of places which are used to describe types and permissions in SELinux. Each place can hold 0, 1 or several token(s). (3) T is a finite set of transitions which are used to describe access relationships between type places and permission places. (4) A is a finite set of directed arcs which are used to link places and transitions and to describe flow directions. (5) N_A: $A \rightarrow P \times T \cup T \times P$ is a node function that associates directed arcs with two nodes (place or transition). (6) C_P: $P \rightarrow \Sigma$ is a color function that associates places with color set. (7) G_T: $T \rightarrow EXP$ is a guard function that associates transitions with expressions such that: $\forall t \in T$, $(type(G_T(t)) = bool) \wedge (type(var(G_T(t))) \subseteq \Sigma)$, where $type(e)$ denotes the data type of an expression e, $type(\{e1, e2, \ldots\})$ denotes the set of data types of expressions $e1$,$e2$, …, $var(e)$ denotes the set of free variables of an expression e, and EXP denotes the set of all expression. G_T is used to describe necessary conditions for information flows. (8) E_A: $A \rightarrow EXP$ is an arc expression function that associates directed arcs with expressions such that: $\forall a \in A$, $(type(E_A(a)) = C_P(p(a))_{MS}) \wedge (type(var(E_A(a))) \subseteq \Sigma)$, where $p(a)$ is the place of $N_A(a)$, and *'t_{MS}'* denotes type ***'multi-set of type t'***. E_A is used to describe the tokens passes the directed arcs and corresponding update modes. (9) I_P: $P \rightarrow EXP$ is an initialization function that associates places with expressions such that: $\forall p \in P$, $type(I_P(p)) = C_P(p)_{MS}$. I_P is used to set initial token values for those places corresponding with source types of inquiry statements.

In addition, two type places are generally linked by one permission place and two transitions. This means that the former type place has the authorization of permission denoted by the permission place against the latter type place. Obviously, it is consistent completely to *allow* statements in SELinux policies configuration. Thereafter, it is mainly by analysis and process of *allow* statements to construct the CPN model for policy analysis.

A token is a dynamic entity in a place and it can move from one place to another place. A transition can be initiated if and only if the value of the token in the place matches the description on the directed arc and thus pass the test of guard function

associated with the transition. During the procedure of analysis as for a given inquiry, a token will record types, permissions and all other information related to inquiry in the information flow path from the place corresponding to source type of the inquiry statement to the current place, called inquiry information flow path. So it can be denoted as a tuple <*bool, queryType, typeList, permissionList*> where *bool* is a Boolean value to store the decision result whether the type in the current place matches the type in the inquiry statement; *queryType* is a char string to record the recent classification label in the inquiry statement during analysis procedure; *typeList* is a char string list to store all types on the inquiry information flow path; *permissionList* is also char string list but it is used to record all permissions on the inquiry information flow path. Values of tokens in places on the inquiry information flow path ought to be updated with the proceeding of analysis according to corresponding different classification labels.

This method has powerful analysis and verification capabilities for SELinux policies configuration. But it has also the disadvantage at difficult inquiry description as well as the information flow analysis method for its essence of information flow.

2.3 Main Idea About an Integrated Analysis Method Based on Access Control Spaces, Information Flows and Colored Petri-Nets

SELinux policies configuration is the basic foundation for systematic security enforcement and it can be viewed as the embodiment of security objectives. Thus the analysis of SELinux policies configuration ought to dedicate to validate if it faithfully supports confidentiality and integrity under mandatory access control, to check if there is any loophole that may impair the security goals, and to help security administrator to make appropriate configuration solution that accords with principles such as least privilege and separation of permission. According to the facts that each method has its own advantages and disadvantages, an integrated analysis method should be developed based upon access control spaces, information flows and colored Petri-nets. Thus different analysis method can be exploited to achieve different analysis goals so as to make full use of respective advantages.

The main idea of the integrated analysis method can be induced as follows:

1. Method based on access control spaces can be used for validity analysis (i.e. to check if policies configuration meets security goals), e.g. to sum up all objects that a specified subject can access and all subjects that can access a specified object and to decide if a specified subject can access a specified object (where subject/object specification can take the way of assigning security context and access can also be assigned as a special access mode), and it is helpful for security administrator to separate permissions, detect configuration bugs (such as undesired authorization or obligations that cannot be fulfilled because of the lack of authorizations) and make complete configuration.
2. Method based on information flows can be used along with access control spaces to analyze integrity and the integrity of trusted computing base (TCB) is the premise and foundation to ensure the security of whole system. By analyzing integrity conflicts between the TCB entities and the non-TCB entities, information can be provided to help security administrators to ameliorate SELinux policies configuration and to optimize and consummate the assignment of TCB entities.

3. Method based on colored Petri-nets can be also used for validity analysis and to find potential problems such as information flow leaks. By elaborate design to make it support both inquiries in positive description and those in negative description as well as inquiries including intermediate types, it will be convenient for security administrators to check completeness and consistency of policies configuration against security objectives.
4. All these methods ought to be implemented in a uniform architecture while modular, simple but effective design principle should be pursued and followed.

3 Prototype Design and Implementation

3.1 Architecture Design

SCIATool takes aims at validity analysis and integrity analysis for a given SELinux policies configuration.

Validity analysis is to make sure that the configuration has put expected access regulations into effect and met corresponding security goals thus inquiries ought to be processed correctly for the following cases. Firstly, if a subject is specified by security context, all objects with corresponding security context and permissions (in the form of <*class_name*, *access_mode*>) that it is authorized to access can be worked out. Secondly, if an object is specified by security context, all subjects with corresponding security context and permissions that it is authorized to be accessed can also be worked out. Thirdly, if a subject is specified by security context, all objects with corresponding security context and permissions that it is prohibited to access can be worked out. Fourthly, if an object is specified by security context, all subjects with corresponding security context and permissions that it is prohibited to be accessed can also be worked out. Fifthly, if a subject and an object are specified in the way of security context respectively, corresponding access relationships (i.e. authorized or prohibited permissions) can be figured out. Finally, if the feature of an information flow path is specified, whether it can be supported by the configuration ought to be analyzed and concluded. Except that last inquiry is processed using colored Petri-nets method, all others are analyzed based on access control spaces.

Integrity analysis in this paper is processed around the TCB. Integrity of the TCB holds if there is no type that can be written by a type outside the TCB and read by a type inside the TCB, except for special cases in which a designated trusted program sanitizes untrusted data when it enters the TCB. Thereafter, integrity analysis is to verify that subjects inside TCB are prohibited to read wrong information from non-trusted objects while sensible information inside TCB objects are protected from wrongly modified. If results show that no non-TCB subject or object can infect any TCB ones, it can be proved that the integrity of TCB is protected by the policy configuration. In fact, it is necessary that information flow from a non-TCB one into a TCB one in some cases. But such information flow ought to be audited, which can be ensured by a different authorization way of **auditallow** statement (opposite to **allow** authorization way). So that any information flow ought to be worked out if it could

influence the integrity of TCB without audit. Integrity analysis is performed based upon access control spaces and information flows.

Accordingly, SCIATool can be divided into following functional parts: (1) a common module that is to extract security elements from source files for SELinux policies configuration and to store them in elaborated data structures in memory; (2) a pair of modules that are to construct security context spaces for subjects and objects respectively; (3) a group of modules that are to construct valid access control spaces for subjects and objects respectively so that analysis of authorization and prohibition for special subject and/or object denoted by security context can be figured out conveniently; (4) a group of modules that are to perform analysis of authorization and prohibition as for inquiry with specified subject and/or object; (5) a pair of modules that are to construct TCB space and to analyze integrity conflicts of TCB on premise of specifying initial TCB entities; (6) a pair of modules that are to construct colored Petri-nets for SELinux policies configuration and to perform inquiry analysis for specified information flow path. The modular architecture of SCIATool can thus be designed and illustrated as Fig. 1.

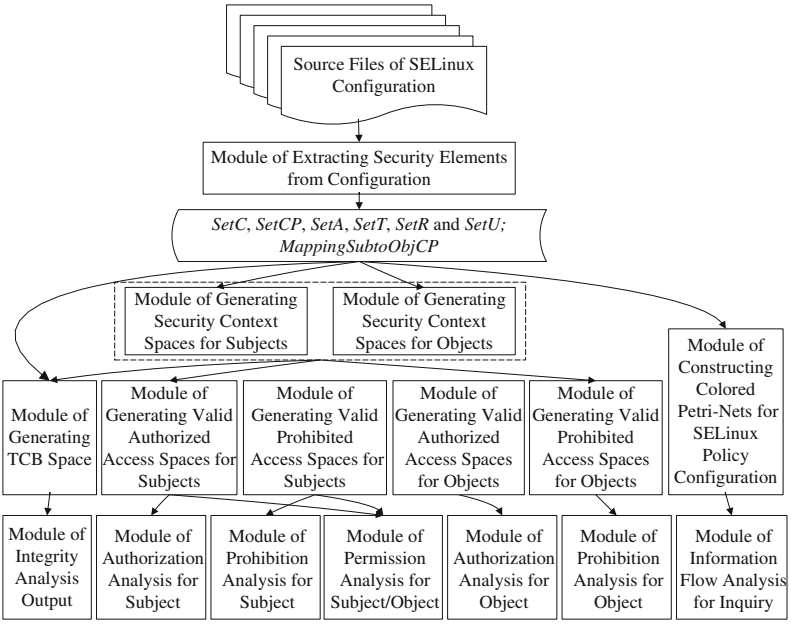

Fig. 1. Architecture of SCIATool

3.2 Main Problems and Solutions About Integration

Because we have already designed and implemented three prototypes based on the former three methods respectively, the prototype of SCIATool is implemented by the way of integrating them together.

The primary problem about integration is to start with clear and relationships among both data structures and modules whether within same prototype or across different prototypes and then to figure out a uniform framework to place those necessary modules at appropriate layer together with corresponding data structures. Just as the architecture of SCIATool we have designed finally (refer to Fig. 1), the module of extracting security elements from configuration is concluded as the fundamental module of SCIATool (thus completeness and correctness of policy information that it extracted will affect all other modules and quality for all kinds of analysis) while modules of generating security context spaces for subjects or objects and that of constructing colored Petri-nets can be placed at the second layer. In addition, all other modules can be placed at a higher layer to perform various practical and flexible analyses.

Another critical problem is that different programming styles and identifier naming habits reflected in different prototypes. So it is a rather difficult work to compose them together, especially when we select the module of extracting security elements from configuration and put it into the final prototype. The module version we have selected is more compatible with other modules placed in the final prototype but another eliminated module version have more strong extracting functions, e.g. it can process policy configuration sources files across different directories which is more closer to real application situations. Therefore, a great deal of work needs to be done to strengthen functions and to improve compatibility as for those modules of the SCIATool prototype.

The third aspect is to enrich practical analysis functions and to provide a more complete analysis result in a more convenient way for security administrators. For example, all sorts of inquiry services are provided including what kind of objects (i.e. objects in what security context) with what permissions a specified subject has been authorized or prohibited, who (i.e. subjects in what security context) is authorized or prohibited to access a specified object with what permissions and what permissions are authorized or prohibited as for a specified subject and a specified object. In addition, both input of formal inquiry statement and wizard-style inquiry input are supported by our prototype of SCIATool to provide security administrator with convenient inquiry input.

3.3 Prototype Implementation

The prototype of SCIATool is developed on Red Hat Enterprise Linux Server release 5.4 while it is written in standard C language and compiled by GCC 4.1.2 20080704 (Red Hat 4.1.2-46). Source codes of the prototype are made up of more than 6000 lines. Compared with some typical SElinux policy analysis tools such as SLAT [7, 8], SEAnalyzer [9] and etc., SCIATool is implemented completely independently and it can be executed and perform analysis tasks without any other available software tools.

4 Test Results and Discussion

The prototype is tested by using a suite of SELinux policies configuration designed for a simplified student-teacher system. And related test cases are devised around validity analysis about authorized or prohibited permissions as for a specified subject or object

in the way of security context, validity analysis as for inquiry of a specified information flow path, and integrity analysis of the TCB.

4.1 Test Results and Analysis

Validity analysis about authorized and prohibited permissions is tested by having security administrator specified any subject and/or object with security context. Test results for validating prohibited permissions by specifying an object with security context <*student_u, student_r, student_t*>, and validating authorized permissions by specifying a subject with security context <*teacher_u, object_r, coursemark_t*> are illustrated in Figs. 2 and 3 respectively.

```
SCIATool – An integrated analysis tool for SELinux policies configuration based on
          access control spaces(ACS), information flows(IF) and colored Petri-nets(CPN)
Copyright@ 2006-2015 Beijing Jiaotong University.

================================[Validity Analysis Based on ACS]================================
Please enter 1~5 to perform corresponding validity analysis based on ACS:
  1. Display all permissions authorized by configuration for specified subject
  2. Display all permissions authorized by configuration to specified object
  3. Display all permissions authorized by configuration for specified subject/object pair
  4. Display all permissions prohibited by configuration for specified subject
  5. Display all permissions prohibited by configuration to specified object
  0. Return
=>4
Input security context of subject as "user role type": student_u student_r student_t

All permissions prohibited by configuration for the specified subject is as follows:
(file, write)   <student_u, object_r, coursepremark_t>
(file, write)   <college_admin_u, object_r, coursepremark_t>
(file, write)   <teacher_u, object_r, coursepremark_t>
(file, write)   <student_u, object_r, coursemark_t>
(file, write)   <college_admin_u, object_r, coursemark_t>
(file, write)   <teacher_u, object_r, coursemark_t>
================================================================================================
```

Fig. 2. Test results for prohibited permissions of subject <*student_u, student_r, student_t*>

```
================================================================================================
Please enter 1~5 to perform corresponding validity analysis based on ACS:
  1. Display all permissions authorized by configuration for specified subject
  2. Display all permissions authorized by configuration to specified object
  3. Display all permissions authorized by configuration for specified subject/object pair
  4. Display all permissions prohibited by configuration for specified subject
  5. Display all permissions prohibited by configuration to specified object
  0. Return
=>2
Input security context of object as "user role type": teacher_u, object_r, coursemark_t

All permissions authorized by configuration to the specified object is as follows:
<college_admin_u, collegeadmin_r, collegeadmin_t>              (file, read)
<student_u, student_r, student_t>              (file, read)
<college_admin_u, collegeadmin_r, accessmark_t>         (file, write)
================================================================================================
```

Fig. 3. Test results for authorized permissions of object <*teacher_u, object_r, coursemark_t*>

By careful analysis and comparison, it is confirmed that test results are consistent with the policy configuration and the configuration faithfully satisfies the original security goals of the system.

Validity analysis about inquiry of an information flow path is focused on ***allow*** rules in the policy configuration. So a series of formal description of inquiry statements are devised for its test (refer to Table 1). Because some security goals of configuration

Table 1. Inquiry statements about security configuration goals

No	Mode	Inquiry statements
1	positive	collegeadmin_t:(RL,-,!()):coursepremark_t
2	positive	collegeadmin_t:(RL,-,!()):coursemark_t
3	positive	collegeadmin_t:(RL,-,!()):coursework_t
4	positive	collegeadmin_t:(RL,-,!()):coursesourse_t
5	positive	collegeadmin_t:(RL,-,!()):coursesourse_t
6	positive	collegeadmin_t:(WL,+,!()):coursemark_t
7	positive	student_t:(RL,-,!()):coursesourse_t
8	positive	student_t:(RL,-,!()):courserecord_t
9	positive	student_t:(RL,-,!()):coursemark_t
10	positive	student_t:(WL,+,!()):coursework_t
11	positive	student_t:(WL,+,!()):courserecord_t
12	positive	teacher_t:(RL,-,!()):coursesourse_t
13	positive	teacher_t:(RL,-,!()):coursework_t
14	positive	teacher_t:(WL,+,!()):coursesourse_t
15	positive	teacher_t:(WL,+,!()):coursepremark_t
16	negative	student_t:(WL,+,!()):coursepremark_t
17	negative	student_t:(WL,+,!()):coursemark_t
18	negative	teacher_t:(WL,+,!(coursepremark_t)):coursemark_t

are not convenient to be described in positive mode (i.e. inquiry described in accordance with security goal), negative mode (i.e. inquiry described opposite to or not in accordance with configuration) is also adopted for some inquiries, e.g. inquiry statements of No. 16–18 in Table 1.

Screenshot about test for inquiry statement No. 11 in Table 1 is illustrated as Fig. 4. Note that the inquiry statement is input by a wizard-style way.

It can be seen from Fig. 4 that a subject with type *student_t* can operate firstly in the way of process transition and turn into a subject with type *accessrecord_t* and then can write the object with type *courserecord_t*.

Inquiry statements in Table 1 have been input into the running prototype and test results show that all the security targets can be met with the policy configuration.

In order to verify whether the prototype can executed effectively as for policy configuration with some bugs, some test cases are also devised and used to test the prototype. For example, it is required that a subject with type *student_t* ought to be authorized to have *read* permission as for an object with type *coursesourse_t*. Thus a new test case for inquiry statement No. 7 in Table 1 is devised for validity analysis based on CPN that the rule statement "*allow student_t coursesourse_t:file{read}*" is deleted from the configuration. Then run the prototype again with the same inquiry statement and test result shows that no information flow can be found and verifies that there is some bug about permission authorization for type *student_t* and *coursesourse_t* in the configuration. Therefore, the prototype is proved to be correct and effective from the negative aspect.

```
SCIATool - An integrated analysis tool for SELinux policies configuration based on
           access control spaces(ACS), information flows(IF) and colored Petri-nets(CPN)
Copyright@ 2006-2015 Beijing Jiaotong University.

============================[Validity Analysis Based on CPN]============================
Please enter 1~2 to perform corresponding validity analysis based on CPN:
  1. Directly input complete query statement to describe access relationship
  2. Input query statement to describe access relationship step by step with wizard
  0. Return
=>2
======[Input query statement to describe access relationship step by step with wizard]======
Source type: student_t
Target type: courserecord_t
Enter "1" for direct access or "0" for indirect access: 0
Enter "1" for readlike access or "2" for writelike access or "0" for readlike/writelike access: 2
Input types not through flow path separated by blank space:
The query statement is as follows:
student_t:(WL,+,!()):courserecord_t
QUERY: Is there any indirect writelike information flow path from student_t to courserecord_t  acc

The query result is YES and the information flow path can be described as follows:
student_t==>transition[process]==>accessrecord_t==>write[file]==>courserecord_t
========================================================================================
```

Fig. 4. Screenshot about test for inquiry statement No. 11

Above results show that the prototype can not only work out all objects (or subjects) with corresponding permissions that any subject (or objects) with specified security context are authorized or prohibited according to the given SELinux policies configuration, but also can it verify correctness of configuration based upon information flow inquiry. In addition, the TCB integrity analysis can be done successfully and all rules that could potentially influence integrity of TCB subjects and objects can be detected.

4.2 Related Work and Discussion

A lot of research work has been done around SELinux policies analysis. As mentioned in the Sect. 2, Gokyo [3] and [13, 14], SELAC [4], SLAT [7, 8] and [15] and SEAnalyzer [9] provide valuable reference for our research in this paper. However, Gokyo is mainly used to check integrity of a proposed trusted computing base (i.e. to identify where untrusted data may enter the TCB) and to resolve constraint conflicts for SELinux that has multiple security goals with obviously different kinds of trust relationship, and it cannot cover all the aspects of policy violations; SELAC is focused on formalization of SELinux configuration language and to model the relationships occurring among sets of configuration rules and verification whether a given subject can access a given object in a given mode as for an arbitrary given security policy configuration; SLAT draws support from the model checker NuSMV for information flow model checking; SEAnalyzer also makes use of a software tool named CPN to aid its analysis procedure. Only one analysis method either based on access control space or based on information flow or based on colored Petri-net is used for policy analysis by each of them, and the analysis must be restricted by limitations of each method.

In addition, logic-programming approach [16, 17], deductive database approach [18], deductive spreadsheets based approach [19], model-based approach [20] and learning-based approach [21] are also used or put forward for analyzing SELinux policies. Specifically, PAL [16] uses SLAT's information flow model and creates a logic program using the XSB logic-programming system to run queries for analyzing SELinux policies; PALMS [17] is implemented in Prolog for policy analysis based on a logical specification for SELinux MLS policies; Lopol [18] normalizes and encodes SELinux policies as logical relations in conjunction with inference rules to perform a number of simple analyses and it can quickly tailor a large default policy (such as strict, targeted, or reference policy) to the specific needs of a system or a class of systems by using a logical rule set as a high-level language; XcelLog [19] is implemented based upon deductive spreadsheets as an add-in to Microsoft Excel and the XSB tabled logic programming system is used as the underlying deductive engine for security policy analysis while SELinux policies in policy.conf format are loaded into XcelLog by using a Perl script to transform the policy into comma-separated-value (.csv) format and then opening the .csv files in Excel; A model-based approach [20] is presented to analyze the dynamic proliferation of access rights in SELinux, which maps SELinux policies to an isomorphic HRU security model whose safety properties can then be analyzed by applying methods and tools available for the analysis of HRU model safety; A learning-based approach [21] is devised to analyze system call logs and to monitor an application's behavior through system calls and an application's policy within SELinux can be improved by reducing the number of Domain-Type associations, i.e. reducing SELinux application access to minimum set of types used by the application.

Compared with above research, SCIATool integrates methods based on access control spaces, information flows and colored Petri-nets organically and realizes the mutual supplement with each other's advantages. Most importantly, SCIATool is implemented independently in C language and its policy analysis doesn't require the help of other available software tools, thus it is easier to be enlarged to construct a well-functioning computer-aided SELinux configuration tool and is convenient to be integrated into other available SELinux configuration tools. However, trends of visualization and engineering [22–30] reflected in recent work to improve SELinux policy configuration ought to be considered and referenced in our future efforts to put SCIATool into practice. Furthermore, SCIATool can be improved by adopting a visualization-based policy analysis framework and it should be implemented as a policy engineering workbench encompassing the automation of engineering steps, prebuilt model patterns, integrated plausibility checks, and model analysis tools.

5 Conclusions

In this paper, we have done some exploratory research and practice for the analysis of SELinux policies configuration by making integrated use of three typical analysis methods based upon access control spaces, information flows and colored Petri-nets. And corresponding prototype, i.e. SCIATool is designed and implemented and tested. To our best of knowledge, SCIATool is the first analysis tool that integrating such three methods together.

By synthesizing complementary advantages of different methods, SCIATool can perform the TCB integrity analysis and various validity analyses effectively, as have been verified by test results. Furthermore, it can not only detect all rules that could potentially influence integrity of the initially specified TCB, but also can it figure out authorized or prohibited permissions as for a specified subject and/or a specified object as well as information flow path in CPN of policy configuration as for a specified information flow path requirement.

Specially, by using inquiry analysis based on colored Petri-nets, a whole or comparatively independent part of security goals can be described as a series of inquiry statement and then be testified by SCIATool. Therefore, SCIATool can not only perform analysis tasks of security goals (i.e. authorized and prohibited permissions) as for a single entity (i.e. subject, object or subject-object pair), but also can it perform analysis tasks of security goals as for a whole system or subsystem by in turn detecting and deciding whether there is an information flow path that satisfies each statement of security goals denoted as the specified inquiry requirement.

SCIATool is implemented completely in popular standard C language and can run independently without the help of any other available software, as make it easy to be enlarged to construct a well-functioning computer-aided SELinux configuration tool and be convenient to be integrated into other available SELinux configuration tools.

Although some non-expert oriented interface design schemes (e.g. support of wizard-style inquiry input and hierarchical display of security elements in policies configuration) have been adopted, a lot of efforts, such as comprehensive support for graphical interface and policy configuration engineering, need to be taken in order to make SCIATool achieve real practicality. At the same time, for complexity and large-scale features that the real configuration of SELinux policies has, performance of analysis and how to improve analysis efficiency ought to be considered in our future work.

Acknowledgements. The research presented in this paper was performed with the support of the Fundamental Research Funds for the Central Universities (No. 2009JBM019). This paper was also supported by the State Scholarship Fund of China Scholarship Council (File No. 201307095025).

References

1. Smalley, S., Vance, C., Salamon, W.: Implementing SELinux as a linux security module. NAI labs report #01-043 (2006)
2. Smalley. S.: Configuring the SELinux policy. NAI Labs Report #02-007 (2005)
3. Jaeger, T., Zhang, X., Edwards, A.: Policy management using access control space. ACM Trans. Inf. Syst. Secur. **6**(3), 327–364 (2003)
4. Zanin, G., Mancini, L.V.: 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)
5. Zhai, Gaoshou, Tong, Wu: Algorithms for automatic analysis of SELinux security policy. Int. J. Secur. Appl. **7**(1), 71–84 (2013)

6. Zhai, Gaoshou, Tong, Wu: Automatic analysis method for SELinux security policy. Int. J. Secur. Appl. **6**(2), 229–234 (2012)
7. Guttman, J.D., Herzog, A.L., Ramsdell, J.D.: Information flow in operating systems: eager formal methods. In: Workshop on Issues in the Theory of Security (WITS 2003). IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS. Warsaw, Poland (2003)
8. Guttman, J.D., Herzog, A.L., Ramsdell, J.D., Skorupka, C.W.: Verifying information flow goals in security-enhanced linux. J. Comput. Secur. **13**, 115–134 (2005)
9. Chen, Y.-M., Kao, Y.-W.: Information flow query and verification for security policy of security-enhanced linux. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., Kawamura, S.-I. (eds.) IWSEC 2006. LNCS, vol. 4266, pp. 389–404. Springer, Heidelberg (2006)
10. Gu, L., Guo, Y., Yang, Y., Bao, F., Mei, H.: Modeling TCG-based secure systems with colored petri nets. In: Chen, L., Yung, M. (eds.) INTRUST 2010. LNCS, vol. 6802, pp. 67–86. Springer, Heidelberg (2011)
11. Ahn, G.J., Xu, W., Zhang, X.: Systematic policy analysis for high-assurance services in SELinux. In: Proceedings of 2008 IEEE Workshop on Policies for Distributed Systems and Networks, pp. 3–10. IEEE Computer Society (2008)
12. Guo, Tao, Zhai, Gaoshou: Automatic analysis of SELinux security policies based on colored petri-net (in Chinese). Inf. Secur. Technol. **4**(11), 35–40 (2013)
13. Jaeger, T., Sailer, R., Zhang, X.: Analyzing integrity protection in the SELinux example policy. In: Proceedings of the 12th USENIX Security Symposium, pp. 59–74. Washington, D.C., USA (2003)
14. Jaeger, T., Sailer, R., Zhang, X.: Resolving constraint conflicts. In: SACMAT 2004, pp. 105–114. Yorktown Heights, New York, USA (2004)
15. Guttman, J.D., Herzog, A.L., Ramsdell, J.D.: SLAT: information flow in security enhanced linux. Included in the SLAT distribution, available from http://www.nsa.gov/SELinux (2003)
16. Sarna-Starosta, B., Stoller, S.D.: Policy analysis for security-enhanced linux. In: Proceedings of the Workshop on Issues in the Theory of Security (WITS 2004), pp. 1–12. IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS. Barcelona, Spain (2004)
17. Hicks, B., Rueda, S., St. Clair, L., Jaeger, T., McDaniel, P.: A logical specification and analysis for SELinux MLS policy. ACM Trans. Inf. Syst. Secur. **13**(3), 26 (2010)
18. Kissinger, A., Hale, J.C.: Lopol: a deductive database approach to policy analysis and rewriting. In: Proceedings of the Second Annual Security-enhanced Linux Symposium. Baltimore, Maryland, USA (2006)
19. Singh, A., Amakrishnan, C.R., Ramakrishnan, I.V.: Security policy analysis using deductive spreadsheets. In: FMSE 2007, pp. 42–50. Fairfax, Virginia, USA (2007)
20. Amthor, P., Kühnhauser, W.E., Pölck, A.: Model-based safety analysis of SELinux security policies. In: 2011 5th International Conference on Network and System Security (NSS), pp. 208–215. IEEE Press, New York (2011)
21. Marouf, S., Phuong, D.M., Shehab, M.: A learning-based approach for SELinux policy optimization with type mining. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW 2010). ACM, New York (2010)
22. Tresys Technology: SETools—policy analysis tools for SELinux. http://oss.tresys.com/projects/setools
23. Wenjuan, X., Shehab, M., Ahn, G.-J.: Visualization-based policy analysis for SELinux: framework and user study. Int. J. Inf. Secur. **12**, 155–171 (2013)

24. Clemente, P., Kaba, B., Rouzaud-Cornabas, J., Alexandre, M., Aujay, G.: SPTrack: visual analysis of information flows within SELinux policies and attack logs. In: Huang, R., Ghorbani, A.A., Pasi, G., Yamaguchi, T., Yen, N.Y., Jin, B. (eds.) AMT 2012. LNCS, vol. 7669, pp. 596–605. Springer, Heidelberg (2012)
25. Marouf, S., Shehab, M.: SEGrapher: visualization-based SELinux policy analysis. In: 2011 4th Symposium on Configuration Analytics and Automation (SAFECONFIG), pp. 1–8. Arlington, VA. IEEE Press, New York (2011)
26. Amthor, P., Kuhnhauser, W.E., Polck, A.: WorSE: a workbench for model-based security engineering. Comput. Secur. **42**, 40–55 (2014)
27. Athey, J., Ashworth, C., Mayer, F., Miner, D.: Towards Intuitive tools for managing SELinux: hiding the details but retaining the power. Tresys Technology. http://www.tresys.com/innovation/papers/Power_of_SELinux.pdf. Accessed 12 March 2007
28. MacMillan, K., Brindle, J., Mayer, F., Caplan, D., Tang, J.: Design and Implementation of the SELinux policy management server. Tresys Technology. http://www.tresys.com/innovation/papers/Design-And-Implementation-of-PMS.pdf. Accessed 1 March 2006
29. Singh, S.: Automatic verification of security policy implementations. Doctoral Dissertation in Computer Science, University of Illinois at Urbana-Champaign (2012)
30. Nakamura, Y., Sameshima, Y., Yamauchi, T.: SELinux security policy configuration system with higher level language. J. Inf. Process. **18**, 201–212 (2010)