A Framework for Security Policy Derivation

Fei Peng[1, a], Tao Zhang[2, b], Weiguang Xu[3, c], Min Zhao[4, d]

[1]College of Command Information System, PLA University of Science and Technology, Nanjing, 210007, China

[2]College of Command Information System, PLA University of Science and Technology, Nanjing, 210007, China

[3]College of Command Information System, PLA University of Science and Technology, Nanjing, 210007, China

[4]College of Command Information System, PLA University of Science and Technology, Nanjing, 210007, China

[a]email:maryfeimm@163.com, [b]email:vigorxu@163.com

Keywords: Security Requirement; Ontology; Security policy; Operating System

Abstract. It is difficult for end users to configure security models in operation system without technical knowledge, since security models are defined by experts who have security experience. Indeed, there is no complete method to translate user security requirements into implementable security policies. In order to answer this demand we present an ontology-based framework for eliciting end users' security requirements toward operation system, deriving security policies and matching appropriate security models for them. In the framework, ontology is employed to standardize knowledge representation, reuse security requirements, reason security policies and describe the ability of security model.

Introduction

In the past two decades, operating system has faced a variety of security challenges. Security issues are concerned by experts, and various kinds of security models are proposed, such as the Bell-LaPadula security policy, Biba, Role-based access controls (RBAC), SELinux, etc. These security models focus on diffident security issues and adapt to diffident security requirements (SR). Most end users without security or technical knowledge have no idea about what these policies are aimed for and how to use them. Even if they know how to use them, configuring the policies one by one is time-consuming and easily leads to mistakes. Therefore, it is important to derive security policies from SR which is considered in terms of the full system. However, there is no complete method to translate users' SR into implementation-level security policies (ISP) directly. This is because researches on SR and security policies lack interaction. Researches on SR focus on eliciting, expressing and analyzing requirement accurately, but pay less attention to how to implement it [1]. On the other side, researches on security policies also do not express in place of requirements, and are complex and concrete. In this paper, we propose a framework for deriving of ISP from end users' SR and attempting to bridge the gap between SR and security policies. There are three main challenges in enforcement: firstly, to help end user express their needs accurately. The second challenge is to translate abstract SR with roles and abstract actions into ISP with instance and operation. The third challenge is to choose diffident security models to satisfy these ISP.

We solve this challenge in this paper. The rest of paper is organized as follows. Related work on security requirement modeling and implementation are introduced in section 2. The framework we proposed is depicted in Section 3. Section 4 gives some conclusions and future work.

Related Work

There are previous works to elicit SR and derive ISP. The model for eliciting SR included KAOS [2], I* [3], Threat Tree [4] and Abuse Case [5]. KAOS and I* are goal-oriented methodologies.

They just express ‘what’, ‘how’, ‘why’, but do not express ‘who’. This leads to a lack of subject for the elicited SR. Threat Tree relies deeply on the users’ security experience, that cause it is difficult for most people to use. Abuse Case is more suitable to be used as a test tool than a high assurance tool. As we said, security model is matched in the last phase of our work. For this we need a new model to elicit SR with main elements like subject, object, action etc. The methods above are not so appropriate for our research. Ontology is well known for a good way to express concept and relation semantically. Some recent researchers consider using ontology for SR elicitation, like O. Daramola [6] use a threat ontology to elicit SR, whose work focus on dealing with textual SR. A. Souag [7] uses ontology to elicit and analyze the resulting SR model elicited by I*, but do not consider the security implementation. Some researches try to translate SR into actual implementable policy like B. Tsoumas [8] uses Nmap to scan network infrastructure asset as the ontology instance, and use IE tools to extract SR from policy documents. Ontology in the research is act as a container of security knowledge and instance. The main difference from our work is that they do not collect SR from user and they deal with network SR.

A Frame for Security Policies Derivation

The main goal of the framework is to translate SR into ISP and match the security model which is in line with ISP. To achieve the goal, we use the reverse engineering idea in our work. We summarize the access control model and produce a unified model, Security Object Model (SOM), which is used to elicit the SR. As shown in figure 1, the framework is divided into three phases. In the first phase, we abstract some basic concepts and relations of access control model, and define rules, and then organize them to establish SOM. After that we use ontology to describe it. Secondly, the ontology-based SOM is used to elicit specified SR and translate them into ISP. Besides, conflict detection for ISP is also completed in this phase. Thirdly, security models such as BLP, Biba, RBAC, SELinux, are also described in the form of ontology. Then, matching appropriate security model with the resulted security policies by an algorithm. In the following section, the framework is introduced in detail.

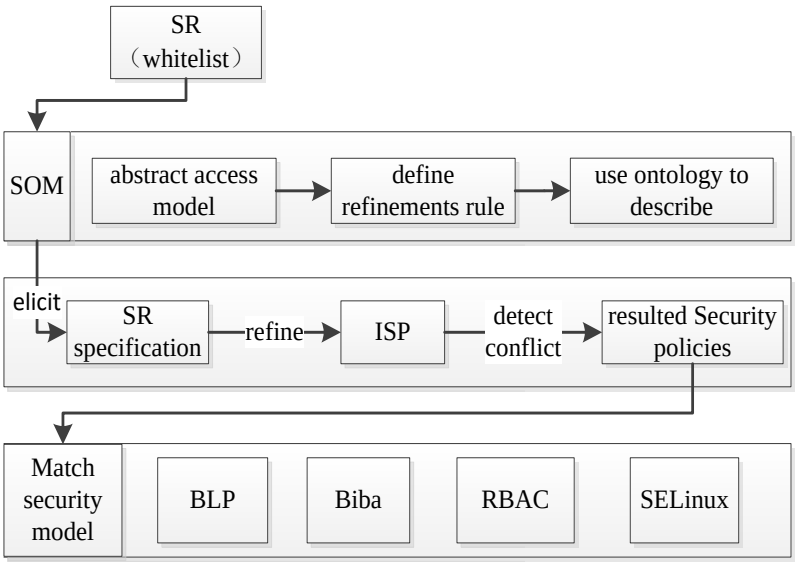

Fig.1. The proposed framework

Ontology-based Security Object Model

SOM is an abstraction of access control model. We describe SOM as a three-tuple: SOM= <*Co*, *In*, *Re*, *Ru* >. Co is the set of concepts of access control model. Typical access control models are always described as allow or not allow subject performing action on objects. According to the analysis of access control model, we define Co as Co=<*security attribute*, *subject*, *object*, *file*, *directory*, *file system*, *file link*, *process*, *action*>. In is a set of instances which includes integrity (I),

availability (A), confidentiality (C), read, write, mount, unmount, etc. *Re* is the set of Relations which are described as $\mathrm{Re} = \{R \mid R \subseteq C_1 \times C_2, C_1 \in C\mathrm{o}, C_2 \in Co\}$. For example, we define that $H\mathrm{as\_path} \subseteq \mathrm{object} \times \mathrm{path}$ indicates object may have path, $Perform \subseteq \mathrm{action} \times \mathrm{object}$ indicates action is performed on object, $Only\_allowpolicy \subseteq subject \times Perform$ indicates subject is only allowed to perform action on object, $Include \subseteq dir \times file$ indicates directory includes files, $Attribute \subseteq object \times \mathrm{sec}urity\_attribute$ indicates object has a security attribute, $A\mathrm{ffect} \subseteq action \times Attribute$ indicates action affect security attribute of object, $O\mathrm{nly\_allow} \subseteq subject \times Attribute$ indicates only allow subject affect the attribute security of object. Ru is the set of rules, which are described as follows:

Security attribute refinement (rule1): We research the control requirements on the system calls in SELinux and classify these security-related system calls by the affect taken on the different security attributes of the different objects. For example, $< read, < file, C >> \in Affect$, $< link, < file, C >> \in Affect$, $< mount, < filesystem, C >> \in Affect$, $< write, < dir, I >> \in Affect$, $< remove, < filelink, A >> \in Affect$, etc. We use these relations to help refining the abstract SR with security attribute of object affected by subject to ISP with actual operation performed on object by subject.

Rule 1:

$$\forall \mathrm{a} \in action, \forall sa \in \mathrm{sec}urity\_attribute, \forall o \in object, \forall s \in subject,$$
$$< o, sa > \in Attribute \wedge < a, < o, sa >> \in Affect \wedge < s, < o, sa >> \in Only\_allow \rightarrow$$
$$< a, o > \in Perform \wedge < s, < a, o >> \in Only\_allowpolicy \quad (1)$$

Example:

$$read \in action, C \in \mathrm{sec}urity\_attribute, file \in object, \forall p1 \in subject,$$
$$< file, C > \in Attribute \wedge < read, < file, C >> \in Affect \wedge < p1, < file, C >> \in Only\_allow \rightarrow$$
$$< read, file > \in Perform \wedge < p1, < read, file >> \in Only\_allowpolicy$$

Inherit from directory (rule 2): The rule indicates security attribute of directory will inherit on the file included in it.

Rule 2:

$$\forall f \in file, \forall d \in dir, \forall s \in subject, \forall sa \in \mathrm{sec}urity\_attribute, < d, sa > \in Attribute \wedge$$
$$< s, < d, sa >> \in Only\_allow \wedge < d, f > \in Include \rightarrow < s, < f, sa >> \in Only\_allow \quad (2)$$

Example:

$$/data/semobile/config \in file, /data/semobile \in dir, jd \in subject, I \in \mathrm{sec}urity\_attribute,$$
$$< /data/semobile, I > \in Attribute \wedge$$
$$< /data/semobile, /data/semobile/config > \in Include \wedge$$
$$< jd, < /data/semobile, I >> \in Only\_allow \rightarrow$$
$$< jd, < /data/semobile/config, I >> \in Only\_allow$$

Object path complement (rule 3): every object must have a path.

Rule 3:

$$\forall o \in object, \exists pa \in path \rightarrow < o, pa > \in Has\_path \quad (3)$$

Example:

$$file1 \in object, /data/config \in path \rightarrow < file1, /data/config > \in Has\_path$$

For the standard expression and reuse of SR, ontology is used to describe SOM and SR. Part of our ontology is shown in figure 2.

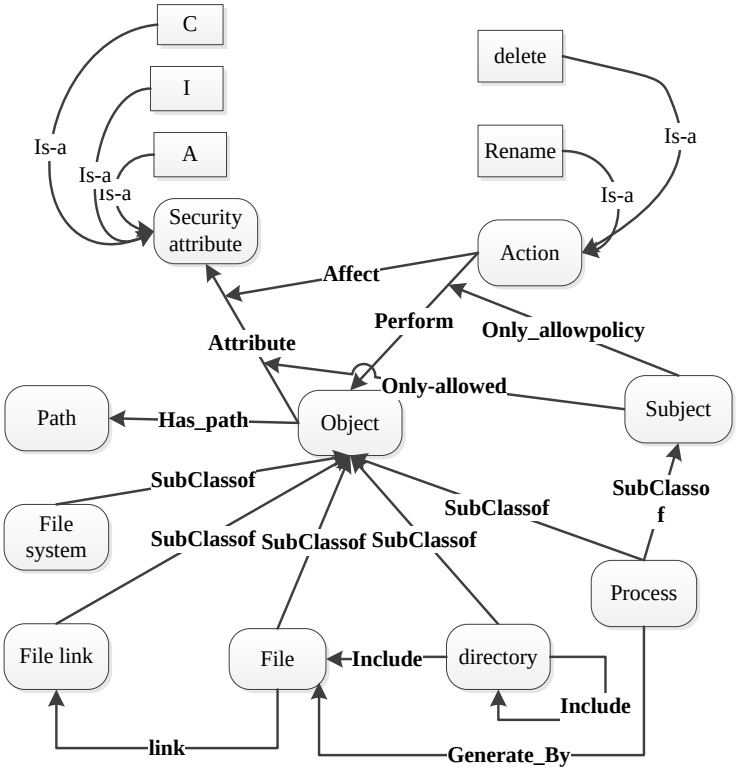

Fig.2. Part of SOM ontology

Security Requirements Elicitation and Refinement

For the sake of helping user to express SR more accurately and simply, we limit end users to express SR in term of whitelist. The whitelist is a list of subjects which have privilege. It expresses which subjects perform action to affect the security attribute of object is legitimate. For example, end users want to protect the confidentiality of file f1 and provide the whitelist as only allowing process p1 to affect the confidentiality of file f1. SR is elicited as follows:

$$C \in \text{security\_attribute},\ f1 \in object,\ p1 \in subject, < f1, C > \in Attribute \wedge$$

$$< p1,\ < f1,\ C >> \in Only\_allow$$

We use rule1 to refine the specified SR to the ISP as follows:

$$read, link \in action, C \in \text{security\_attribute},\ f1 \in object,\ p1 \in subject,$$

$$< f1, C > \in Attribute \wedge < read, < f1, C >> \in Affect \wedge < p1,\ < f1,\ C >> \in Only\_allow$$

$$\rightarrow < p1,\ < f1, read >> \in Only\_allowpolicy,$$

$$< f1, C > \in Attribute \wedge < link, < f1, C >> \in Affect \wedge < p1,\ < f1,\ C >> \in Only\_allow$$

$$\rightarrow < p1,\ < f1, link >> \in Only\_allowpolicy$$

Rule3 is used to complete the path.

$$f1 \in object, \exists /.../\ f1 \in path, < f1, /.../\ f1 > \in Has\_path$$

After the complementation and refinement, Conflict is detected. In our research, the conflict is defined as when a subject is not in the whitelist performs action to affect the security attribute of object.

Security Model Matching

Different security model focuses on different security attributes, for example BLP focuses on confidentiality, Biba focuses on integrity, and SELinux can protect both integrity and confidentiality. The granularity of different security model is also different. RBAC is coarser than other security models with implementable policy. A matching algorithm would be designed to help providing appropriate security models suggestion for user. Ontology in our research is implemented

in OWL. And another algorithm would be proposed to help translate the security policy in OWL into the security model language which can be configured in operating system directly.

Conclusion

In this paper, we present a framework for deriving security policy. To support a more accurate expression, we use ontology to describe and refine SR. A key contribution of the approach is providing a complete method to translate user SR into ISP which can be configured in operation system directly. Some processes of approach in this paper are semi-automatic like establishing ontology and eliciting SR. In the future, we want to advance it to an automatic process.

References

[1] Hadavi M, Hamishagi V, Sangchi H. Security Requirements Engineering; State of the Art and Research Challenges, International MultiConference of Engineers and Computer Scientists, 2008.

[2] Lamsweerde A. Elaborating security requirements by construction of intentional anti-models, 26th International Conference on Software Engineering, 2004.

[3] Liu L, Yu E, Mylopoulos J. Security and privacy requirements analysis within a social setting, 11th IEEE International Requirements Engineering Conference, 2003.

[4] Amoroso E J. Fundamentals of Computer Security. Prentice Hall, 1994.

[5] McDermott J, Fox C. Using abuse case models for security requirements analysis, 15th Computer Security Applications Conference. 1999.

[6] OLAWANDE, GUTTORM, THOMAS. Ontology-based support for security requirements specification process. On the Move to Meaningful Internet Systems: OTM 2012 Workshops[C]. Rome: Springer, 2012.194–206.

[7] Souag A, Salinesi C, Wattiau I, Mouratidis H. Using security and domain ontologies for security requirements analysis, Computer Software and Applications Conference Workshops, 2013.

[8] BILL, PANAGIOTIS, STELIOS, ET al. Security-by-ontology: A knowledge-centric approach. Security and Privacy in Dynamic Environments[C]. Karlstad: Springer, 2006.99–110.