Architecture and Models for Security Policy Verification

I.V. Kotenko, A.V. Tishkov, O.V. Chervatuk[1]

1. Introduction

Policy-based security management of computer networks is one of the most actual directions of research in information security area. At present the IETF [1] recommendations are commonly accepted standard for the architecture of policy-based management systems. According to these recommendations such architecture should contain the centralized repository of policy rules for entire system, thus making the policy available for analysis and verification. This paper elaborates the architecture and models of security policy verification system – SEcurity Checker (SEC) – originally suggested in [2] and implemented corresponding to the IETF recommendations.

In the paper the improved architecture of Security Checker is considered and the mechanisms of operating with the policies of three levels are described: (1) upper-level, that is approximated to the user requirement language, (2) intermediate level, classifying rules according to several categories, and (3) low-level, describing the policy in the format of Common Information Model (CIM). The approach to the design and implementation of SEC kernel is given. An example of authorization policy conflicts emulation and detection is suggested. The relevant works are analyzed.

2. Improved architecture of Security Checker

In SEC the policy description language has three levels: upper, intermediate, and low (fig. 1).

Upper-level language (UL) describes the problem from generalized point of view. Formulations allow mentioning the groups of devices and the types of applications ("subnet S should not be accessible from host H by protocol P"). For specification of upper-level policies a scripting language is used as well as the set of translators from upper (U) level to the intermediate (I) one (UI-translators).

Upper-level rules are translated to the intermediate level (specified in intermediate level language (IL)) into the one of six categories of policy rules: authentication, authorization, filtering, confidentiality, operation rules, and vulnerability assessment rules. *For* each mentioned category an UI-translator is created. UI-translator receives upper-level rule as input and gives XML documents as output. These XML documents are valid according to XML-schema of corresponding category.

One of non-trivial translators from upper level to the intermediate one is UI-translator that defines filtering policy. In this task context the nodes of computer network are divided into two types: filtering and non-filtering (see fig. 2).

When a policy specifying non-filtering node is set, a task of using filtering nodes for granting or denying access to protected node is solved on the graph representing the network topology. This task is solved as a one about minimal

[1] The research is supported by the "Fund for support of national science", grant of Russian Foundation of Basic Research (№ 04-01-00167), grant of the Department for Informational Technologies and Computation Systems of the Russian Academy of Sciences (contract №3.2/03) and partly funded by the EC as part of the POSITIF project (contract IST-2002-002314).

graph cut. Fig. 2 gives an example of creating four filtering rules by UI-translator, when upper level policy requires prohibiting the access between non-filtering nodes.

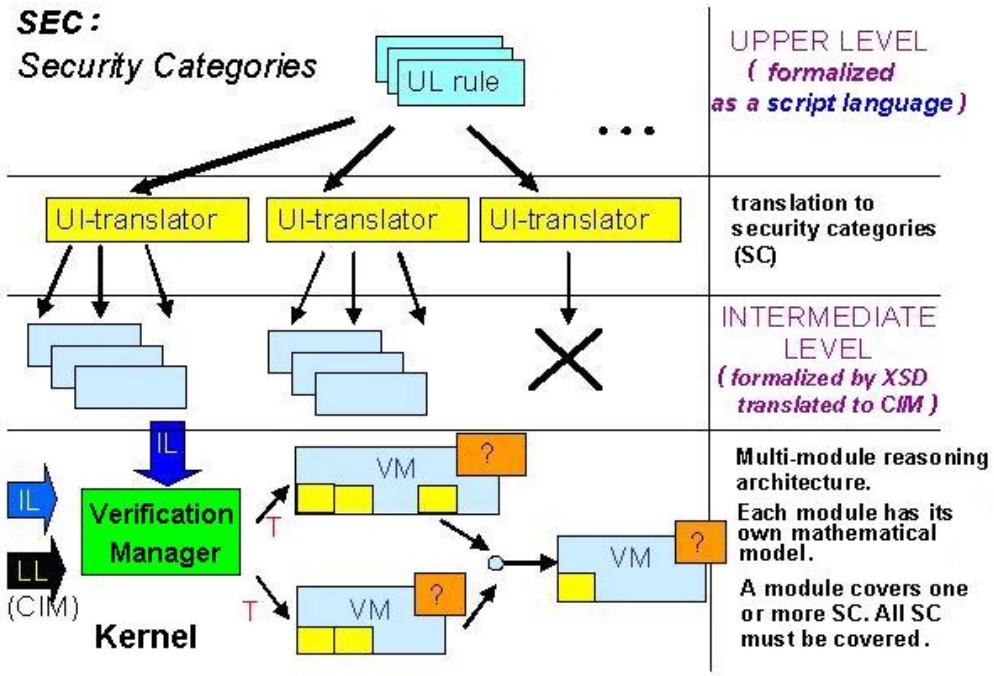

Fig.1. Generalized SEC architecture

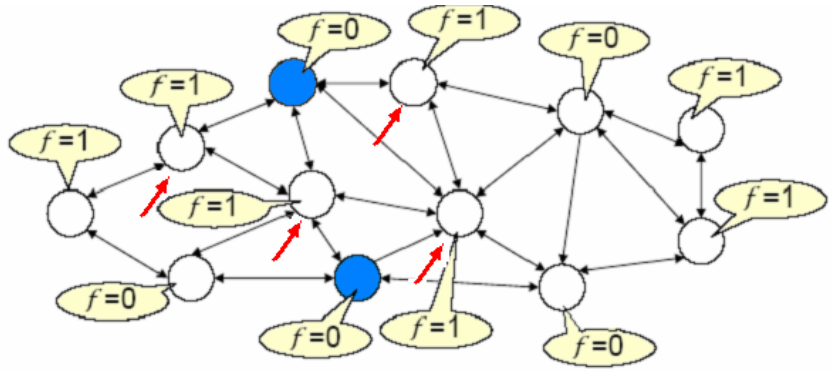

Fig.2. Filtering (f=1) and non-filtering (f=0) nodes

At extending of upper-level language with new constructions, a set of new UI-translators should be uploaded to the system for each category involved into such extending. Only those extensions are allowed, that do not change existing sublanguage. Thus, the SEC architecture is implemented as open for interpreting rules of other languages, such as Ponder [3] and other user-defined languages.

Finally, low-level language (LL) is a translation of intermediate level rules to object-oriented format of Common Information Model (CIM).

Structure of SEC kernel contains two types of basic elements: verification manager and verification module (VM).

Each verification module has its own knowledge base (as axiomatics, temporal logics formulae, action semi-lattices and others) and implements its own algorithm for checking policies consistency and applicability to given system description. Besides that, each module declares security categories with which it works.

Verification manager, getting intermediate and low-level policies as input, calls verification modules in parallel or subsequently. Parallel verification is possible only for modules that do not change the set of rules. Modules, that delete, change or add rules, are launched subsequently, getting at input a policy that is potentially changed by preceding modules. Such algorithm of kernel processing implies iterative calling of modules sequence. Iterations continue until the set of rules stops changing or until stop condition executes, in simplest case — by the explicit limitation of iterations number.

3. Security categories

As it was mentioned above, the intermediate level language is based on XML schemas for six categories of rules.

Authentication rule contains subjects (roles and users), objects (services defined on system description language [1]), actions that can be performed on the services, authentication method and security level, which the rule is associated with. The authentication method is defined by classes which are derived from CIM-class AuthenticationCondition. These classes are as follows: SharedSecretAuthentication, AccountAuthentication, BiometricAuthentication, NetworkingIDAuthentication, PublicPrivateKeyAuthentication, KerberosAuthentication, DocumentAuthentication, PhysicalCredentialAuthentication. All rules are accompanied with security level label. Security system can switch from one security level to another if, for example, the attack is detected.

Authorization rule is formulated as if-then rule. The conditional part contains quantifier-free predicate formula using NOT, AND, and OR logical operations. Atoms are the definitions of subject, object, action, security level, and the condition of system state. System state is described by the current state of services (run, stopped, waiting, busy), the results of authorization and authentication rules enforcement (the subject is authorized/authenticated for performing the action on the object), and user-defined system state conditions. The main used CIM-classes are as follows: Policy, AuthorizedSubject, AuthorizedObject, AuthorizedPriviledge, ComputerSystem, Role, and Identity.

Filtering rule represents commonly used access control list, each row of which consists of source address and port, destination address and port, deny/allow privilege and, additionally, security level. The used CIM-classes are Policy, FilterList, and FilterEntry.

Confidentiality rules are currently considered only for communication security, and define security protocols for data channels. The corresponding XML schema supports SSL or IPSec protocol. The main used CIM-classes are Policy, IPSecRule, and SSLRule.

Operational rules are specified by system state condition and actions, which should be performed on objects when system state matches the condition of a rule. The corresponding XML schema contains components for definition of network services installed on hosts, and actions which can be performed on those services. The CIM-class Policy is used, and three classes are added to CIM policy class hierarchy. They are OperationalRule, StatusCondition и OperationalAction.

Vulnerability assessment rules are created by use of vulnerability database [4]. The rule contains vulnerability ID, reference to exploit, name and version of vulnerable software, information about patch/update that eliminates the vulnerability, and some additional information [5].

4. Kernel implementation

Basic SEC kernel classes are verification manager and verification module.

Verification manager (VerificationManager) gives to verification modules the system specification (in system description language) and fragments of policy specifications, according to security categories, for which the verification module is responsible. Besides that in suggested representation the manager gives out information about verification results, information about contradictions, if they appeared, and achieved security level. This class implements design pattern "singleton" [6], because verification manager should be only one in the system.

UML-representation for verification manager is given in fig. 3. For each public field the existence of set value and get value functions is supposed.

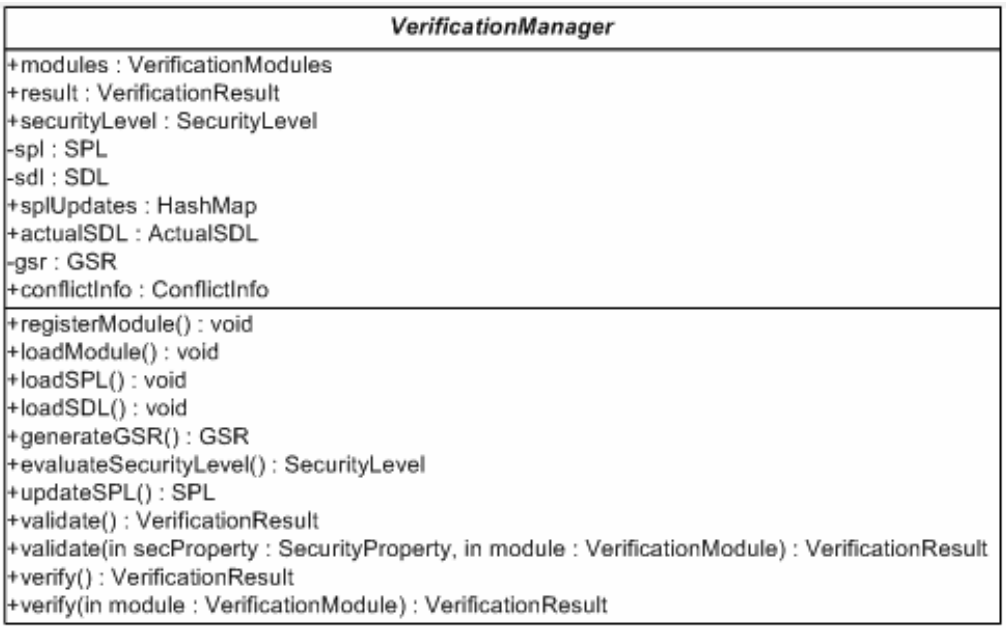

Fig 3: VerificationManager class

In this paper let us consider only several main fields and methods of class VerificationManager:

- Field *HashMap splUpdates* contains references to objects SPLUpdates created in each module. Objects SPLUpdates store list of changes that are necessary to be applied to rules set of policy for resolution of conflicts that were revealed during verification.
- Field *ActualSDL actualSDL* is revised network topology, in which some services are blocked by policies. ActualSDL contains list of blocked services.
- Field *ConflictInfo conflictInfo* contains information about conflicts revealed in the process of validation and verification.
- Method *updateSPL()* implements rules sets changing, proposed by modules.
- Method *validate()* without parameters checks rules for each security category using all registered and loaded modules that are responsible for this security category.
- Method *validate()* with parameters performs detecting and resolving rules conflicts within one security category. Security category and module that performs checking are passed as method parameters.

- Method *verify()* checks consistency of entire rules set and their applicability to the given systems description using special module.

Verification module VerificationModule (fig. 4) performs validation and verification of categories rules SecurityProperty, for which it is responsible and which are listed in corresponding field.

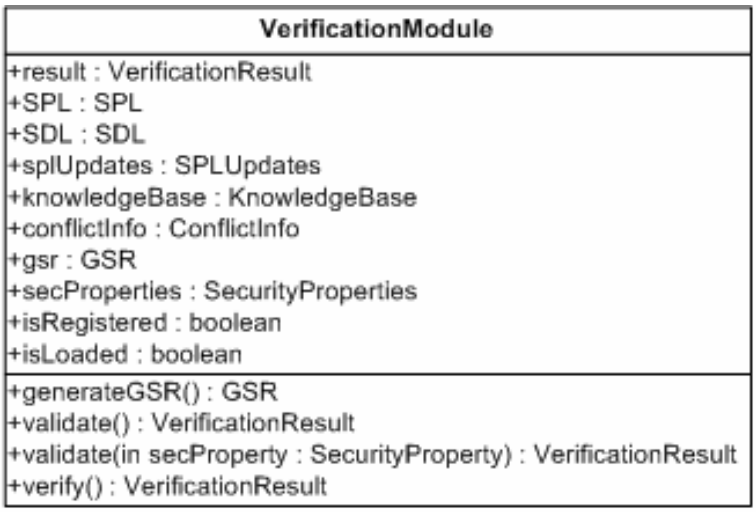

Fig. 4: VerificationModule class

Main methods of class VerificationModule are validate() and verify(). Through these methods class VerificationModule delegates corresponding functionality to class VerificationModule.

5. Example of conflict detection

At current implementation of three verification modules is being done: (1) based on Event Calculus [7], (3) based on Model Checking [8], and (3) by creating the semi-lattices of actions.

Let us describe a simple example of modeling and detection of authorization conflict implemented by SPIN models checker [9].

Authorization conflict appears in the case when one user is attached to two roles R1 and R2 that have contradictory privileges for the same action: for one role there is permission, and for the other there is prohibition.

Key blocks of the program are two processes. The first process appoints and deletes belonging of a user to one of two roles (R1 or R2) at random. The following code corresponds to assigning a user to a role:

```
active proctype userRoleAssignment()
{
…
     :: (r.q<max_q_roles-1)->
          atomic      {
                r.q++;
                if
                ::r.ar[r.q]=R1;
                ::r.ar[r.q]=R2;
                fi
                }
…
```

The second process models print requests, sent by user at random moments. Procedure IsAssigned checks user's belonging to the given role.

The following code, receiving print request, assigns true value to variable deny (if the user at current belongs to role R1), or variable allow (if it belongs to role R2):

```
        ::printer_in?action,rr-> atomic
                        {
                        deny=false;
                        allow=false
                        IsAssigned(rr,R1,R1Res);
                        IsAssigned (rr,R2,R2Res);
                        if
                        ::R1Res->deny=true
                        ::else
                        fi
                        if
                        ::R2->allow=true
                        ::else
                        fi
        …
```

Conflict appearance is in non-fulfillment of the following system state correctness condition: allow and deny cannot be performed simultaneously:

```
        assert((allow && !(deny)) || (!(allow) && deny))
```

6. Relevant works

Many contemporary policy-based security systems are well-matured, but do not involve all the security categories that are presented in this paper and have differing architectures.

Extensible markup language for access control XACML [10] corresponds to SEC authorization policy. Three-level structure of policy specification (rule – policy as set of rules – set of policies) allows to build flexible resolution system using the formalized notion of decision algorithm on the levels of policy and policy set. Unlike the suggested approach, XACML does not have special system specification language, and the specification of network nodes is a part of rules description.

Language Ponder [3] contains the rules of positive and negative authorization, the rules of obligation and delegation. The authors of Ponder suggested several interesting approaches for conflict resolution strategies [11, 12], which are nevertheless too specific to the policies formalism introduced.

Flexible Authorization Framework (FAF) [13, 14] corresponds to access control systems. The FAF advantages are the detailed considering the hierarchies of objects, subjects and privileges on access estimation. The formalism used allows specifying of positive and negative authorization, involves terms of privileges propagation through hierarchies, algorithms and strategies of conflicts resolving on authorization.

There are other approaches representing different techniques for conflicts detecting and resolving in security policies. Here we mark the deontic logics approach [15], dynamic conflict detection with temporal logics [16, 17], as well as one of basic papers on classification of security policies conflicts [18].

Conclusion

This paper proposes the Security Checker architecture for policy-based security management system. Three-level structure for policies definition language is defined: from nearly natural upper-level language to object-

oriented policy representation in CIM format. Security categories are specified, into which policy rules are separated. UML representation of principal classes of SEC kernel is given, the idea for implementation of conflict detecting in authorization policy is demonstrated.

Further work deals with the enhancement of techniques and algorithms of security policy verification and the design of SEC prototype basing on web-services technology.

References

1. IETF Policy Framework (policy) Working Group. http://www.ietf.org/html.charters/policy-charter.html
2. I.V. Kotenko, A.V. Tishkov. Events calculus for specification and verification of security policies for protected computer network. 3rd Russian Conference "Mathematics and Security of Information Technologies". Moscow, MSU, 2004.
3. Ponder: A Policy Language for Distributed Systems Management. Department of Computing, Imperial College. http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml
4. OSVDB: The Open Source Vulnerability Database. http://www.osvdb.org/
5. M. Rohse. Vulnerability naming schemes and description languages: CVE, Bugtraq, AVDL and VulnXML. SANS GSEC PRACTICAL, 2003.
6. M. Grand. Patterns in Java, Volume 1, A Catalog of Reusable Design Patterns Illustrated with UML. John Wiley & Sons, 1998.
7. R.A. Kowalski, M.J. Sergot. A Logic-Based Calculus of Events. New Generation Computing, No 4, 1986.
8. E.M. Clarke, O. Grumberg, D.A. Peled. Model Checking. MIT Press, 1999.
9. G.J. Holzmann. The Spin Model Checker. IEEE Trans. on Software Engineering, Vol.23, No.5, 1997.
10. OASIS: eXtensible Access Control Markup Language (XACML). http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
11. L. Lymberopoulos, E. Lupu, M. Sloman. Ponder Policy Implementation and Validation in a CIM and Differentiated Services Framework. IFIP/IEEE Network Operations and Management Symposium (NOMS 2004), Seoul, Korea, 2004.
12. A. Bandara, E. Lupu, A. Russo. Using Event Calculus to Formalize Policy Specifications and Analysis// IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003.
13. S. Jajodia, P. Samarati, M.L. Sapino, V.S. Subrahmanian. Flexible support for multiple access control policies. ACM Trans. Database Systems, Vol. 26, No.2, 2001.
14. S. Jajodia, P. Samarati, V.S. Subrahmanian. A Logical Language for Expressing Authorizations. IEEE Symposium on Security and Privacy, 1997.
15. L. Cholvy and F. Cuppens. Analysing consistency of security policies. Proceedings of IEEE Symposium on Security and Privacy, 1997.
16. N. Dunlop, J. Indulska, K. Raymond. Methods for Conflict Resolution in Policy-Based Management Systems. Proceedings of the Seventh IEEE International Enterprise Distributed Object Computing Conference (EDOC'03), 2003.
17. N. Dunlop, J. Indulska, K. Raymond. Dynamic Conflict Detection in Policy-Based Management Systems. Proceedings of the Sixth IEEE

International Enterprise Distributed Object Computing Conference (EDOC'02), 2002.
18. E. Lupu, M. Sloman. Conflict Analysis for Management Policies. Fifth IFIP/IEEE International Symposium on Integrated Network Management IM'97, San-Diego, 1997.