Hardening large-scale networks security through a meta-policy framework

Mathieu Blanc
Laurent Oudot
CEA

Patrice Clemente
Pierre Courtieu
Gaétan Hains
Christian Toinard
LIFO

Stéphane Franche
Lionel Vessiller
ENSI-Bourges

ABSTRACT

This paper presents a novel approach where distributed nodes participating to a common infrastructure can modify in a distributed way a Mandatory Access Control policy without any central component. This approach is considered for the security of large shared networks such as securing distributed stations connected to the Internet. The local modification enables a node first to adapt its configuration to the application that has to be deployed on that node, and second to react to specific attacks that are detected locally. Moreover, a local approach provides a better fault tolerance since the policy update does not rely on a central component. The general idea is to have a common shared policy including protection rules plus modification rules. A modification rule enables a node first to modify existing protection rules and second to add new types, roles and users in the system in order to define new rules. A modify rule provides also the ability to suppress types, roles and users from the protection rules. So, our approach is to have a meta-control supporting distributed evolutions of local protection rules. This approach is developed as a joint research project with INRIA and FT R&D, called ACI SATIN, where verification techniques will be proposed to verify that the distributed modifications cannot violate the required security properties.

Keywords: Internet security, access control, security management, peer-to-peer.

1 INTRODUCTION

Different approaches exist to carry out Mandatory Access Control. Discretionary Access Control has shown its limits as malicious hackers usually find their way to root access. The need for stronger control is obvious as now the focus is on Mandatory Access Control. Access control management should not be delegated to end-users but to an independent entity that can be implemented at a kernel level.

There are many models of Access Control, the most generic being the Lampson model [1] of access matrix. The classic Bell - La Padula model [2], was inherited from the military classification model, and implements access control in a hierarchical way. The classic laws of BLP - No Read Up, No Write Down - ensure the non-disclosure. The Biba Integrity model [3] is the dual of BLP and deals with integrity issues, the No Read Down and No Write Up rules ensure that objects and subjects cannot access lower integrity levels in a way that would compromise them. These models have been heavily criticized and have proven to be difficult to apply on standard operating systems.

New Access Control models apply much more to modern operating systems. Role-based Access Control (RBAC) [4] [5] defines roles, with a given set of permissions, i.e actions on the operating system, and associates users to roles. Given a particular role, every access that is not explicitly authorized should be denied. Also, the Domain and Type Enforcement model [6] has been designed to allow the specification of security policies covering many applications. It associates subjects to domains, and objects to types, where domains and types are security equivalence classes. Access Control rules then specify how domains can access types, and also the domains and types for new subjects and objects.

This paper presents a novel approach where distributed nodes can update locally their MAC policies. In contrast with the other solutions, it enables distributed nodes to update accordingly their local policy while satisfying a common meta-policy. This approach can serve for distributed applications relying on a set of Internet nodes. For example, it can help an Internet provider to protect end user station while enabling the client to modify locally its policy without requiring any external control. A central approach would provide poor fault-tolerance since attackers could carry out a denial of service on the server. To resist such attacks, each node has the power to update its policy accordingly while satisfying a meta security properties that the provider defined.

We propose a novel framework where distributed nodes can add, remove or update different attributes of their protection rules while reusing existing shared attributes and rules. The important point is that the framework provides a meta-policy guarantying that the distributed modifications satisfy a control policy telling who has the right to make local modifications and for which attributes and rules. Since the purpose is to be able to guaranty security properties for such distributed evolutions, we present a solution for verifying that the distributed modifications still satisfy some security properties. Typically, we propose to guaranty that there is no information flow for non permitted interactions.

This paper first presents the different solutions that can be compared with our new approach of meta-policy. Second, it presents our motivation for defining a framework for controlling distributed policies. Third, it presents our approach based upon a language for controlling distributed modifications of MAC policies. Fourth, we propose a formal approach to show that forbidden information flows cannot occur for a given meta-policy.

2 RELATED WORKS

Linux Security Modules [7] is a powerful and extensible architecture that adds security hooks to the Linux kernel. LSM does not provide any specific control but allows the implementation of many kinds of access control.

SELinux [8] uses the LSM framework [9] to implement a fine-grained MAC. The configuration grammar provides a very configurable and extensible way to define a security policy, in particular Role-Based Access Control. The Flask architecture [10] allows a strong separation between the security enforcement and the security policy. The Security Server is a kind of local manager that distributes the security policy to the Access Vector Cache for performance improvements.

Despite Flask is not a distributed system, one can imagine extending The system through externalization of the Security Server, which could then distribute the security policy to a collection of distant AVCs. The distributed solution would be similar to the DSI [11] distribution system. The counterpart of the refinement of the SELinux security policy is the administration effort required to defined efficient security rules. The policy language is very expressive but a deep understanding of SELinux and of the applications to secure are required to define relevant security rules. In practice, a hard and time-consuming work is needed to define security rules for a given application.

Grsecurity [12] can restrict the access rights of specific subjects, with an ACL system that is relatively rough compared to SELinux. Nevertheless, it is much easier to define security rules. The real problem is the difficulty to evaluate the expressiveness of the security language. Moreover, the prevention system of Grsecurity addresses a really different purpose. Prevention deals with making really hard the exploitation of buffer overflows, when MAC deals with containment for buffer overflows. MAC also deals with other types of attacks such as social engineering. Actually, the main limitation is the Poor description of the software design.

DSI [11] does not address primarily the completeness of the access control but the capability to distribute a security policy among a group of nodes. The security language is very simple as the access control is limited to a subset of the system calls. In contrast with the three previous solutions, DSI is the only solution dealing with the distribution of a security policy, but it relies on a centralized approach, without advanced recovery solutions. For example, a routing attack could separate nodes from the server. Even in case of server replication, nodes could be prevented from receiving updates for the security policy, thus making the policy distribution ineffective. Moreover, various forms of DoS attacks could also prevent the server from updating the security policy.

Extensions have been proposed to cover organizational features. In OrBAC [13], the basic idea is to provides a set of policies entities (role, activity, context, view, subject, object, action, organization) that enables to adapt a common policy to different organizations using organization model as an extension of role based access control. Inheritance enables to give a sub-role R_2 the same power than a role R_1. A hierarchy of activity and a hierarchy of organization can also be defined. All these hierarchies are based on inheritance. Since all the basic roles, activities and organization are defined globally, there is no facilities to add locally new roles, subject, object, views and rules while satisfying a meta-protection. It is more or less a database approach where all the elements of a hierarchy are globally defined. So, there is no distributed way to manage locally new protection attributes. Finally, there is no tool to verify security properties.

3 META-POLICY FRAMEWORK

3.1 Architecture overview

The architecture of the meta-policy framework is described in Figure 1. Basically, it consist of a policy control component installed on each host of the network. This component is an agent in charge of validating the policy updates against the modification policy.

This agent obtains the modification policy plus a reusable protection policy from a ”meta-policy bus”, i.e. a secure communication channel. It can be an encrypted file on a USB disk, that will then be plugged during the meta-policy framework deployment on each host, as well as a secure network protocol allowing distant configuration of

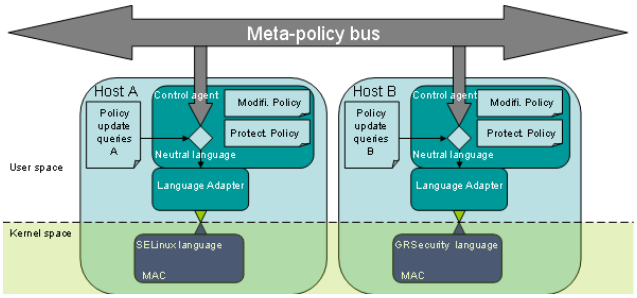

Figure 1: Architecture overview.

the framework. Every host connected to the same meta-policy bus recovers thus the same modification policy and a start-up protection policy. This paper does not detail the meta-policy bus because various communication tools can address the transmission of the meta-policy and the hardest point is how to define a modification policy and how to solve the heterogeneity between different target systems (e.g. SELinux and grsecurity).

The modification policy defines modification rules using a meta-policy language. These modification rules form a protection mechanism for controlling the updates of the protection policy. The main concern is to define a modification language enabling to write modification rules. For simplification, the modification rules can be seen as a meta-protection matrix. The protection policy contains the current protection of the resources that the under-laying MAC target has to apply. In turn, the protection policy is composed of protection rules. The protection policy is expressed into a neutral language in order to solve the heterogeneity between the different under-laying targets. For that purpose, a Language Adapter converts the rules defined with the neutral language into target rules (e.g. SELinux rules or grsecurity rules). The neutral rules can be seen for convenience as a protection matrix.

The different nodes, participating to the same distributed system, share the modification policy. Since the security policies are modified locally without need of any communications with a third party or server, two distant policies can be completely different while satisfying the global security properties that are defined by the modification policy.

Security policy updates queries are used as input to this agent. The latter is then responsible for verifying that these queries are valid accordingly with the modification policy. How security updates are transmitted is outside the scope of this paper and it is not the main concern.

3.2 Reusable protection policy

In a general manner such as SELinux, a protection policy defines a permission between a couple of security context. Each security context has three attributes role, type and user. A type defines a set of given subjects or objects. Different users are considered in the system. These users are MAC users. Specific MAC users can be projected onto classical end-user such as Unix users. Finally, a role enables the same MAC user to play different roles into the system. It provides a concise mean to define protection rules since the same role can be played by different MAC users.

A protection rule gives the authorization for a specific interaction between two security contexts. For example, an interaction for writing on a raw device is allowed between a subject security context (R_1, T_1, U_1) and an object security context (R_2, T_2, U_2). So, a protection rule is expressed with a language such as enable(Interaction Type, Subject Security Context, Object Security Context). That type of language can be easily projected onto a protection matrix such as:

	sc_{O_1}	sc_{O_2}	sc_{O_3}
sc_{S_1}	IT_A, IT_B	IT_C, IT_D, IT_E	...
sc_{S_2}	...	...	...
sc_{S_3}	...	...	...

For compact coding, each element of that matrix contains a list of permitted interactions. In the sequel, we do not consider anymore the under-laying protection language. At the beginning of its lifetime, a node recovers a copy of the reusable protection rules. Then that copy of the protection rules can evolve locally becoming thus a completely new protection rule that could be very different from distant protection rules.

3.3 Modification policy

This type of rules are not available currently within existing MAC or RBAC systems. We propose a protection language for defining the permissions to modify the protection policy. This language enables 1) to modify the reusable protection policy and 2) to add/remove/update security contexts.

3.3.1 Meta-policy rules for policy modification

A rule $enableAddIT(sc, [sc_S, sc_O], PVA)$ allows a given entity sc to add an authorization for a new interaction (IT) between the two target security contexts sc_S and sc_O, in the scope of PVA which is a Permitted Value Array.

The rules $enableRemoveIT$ and $enableModifyIT$ respectively control the functions $removeIT([sc_S, sc_O], PVA)$ and $modifyIT([sc_S, sc_O], PVA)$.

For example, consider sc_{apache} (the security context of the apache process) for which the interaction IT_{accept} (accept connections) is authorized. The rule $enableRemoveIT(sc_{IDS}, [sc_{apache}, sc_*], [accept])$ allows a process with the security context sc_{IDS} to call $removeIT([sc_{apache}, sc_*], [accept])$ and to forbid the acceptance of new connections by apache. This could be useful in case an attack is detected by an intrusion detection tool, to prevent attacks on apache.

3.3.2 Meta-policy rules for type, role and user management

Three rules *enableCreateT*(sc), *enableCreateR*(sc) and *enableCreateU*(sc) allow a given entity *sc* to create either a new type, role or MAC user. For example, the addition of a new application requires the creation of a new type. A user with a security context sc_{S_U} installing this new application can be granted the right to add new types so that it can be integrated in the protection policy, by the way of the installer. A rule such as *enableCreateT*(sc_{S_U}) would be added to the meta-protection.

Also, new roles or MAC users can be created for the cases where specific permissions are needed. For example, a particular system user may need a read access to a log file, typically owned by an admin, so a particular role could be created with that permission so that he can read this file. The system administrator with security context $sc_{S_{ADM}}$ would be given the right to add users and roles, with these rules: *enableCreateR*$(sc_{S_{ADM}})$, *enableCreateU*$(sc_{S_{ADM}})$.

We define three more rules *disableCreateT*(sc), *disableCreateR*(sc) and *disableCreateU*(sc), that are the exact opposite of the create rules. They may be required to disable certain enable rules, for example in case some conditions are not satisfied.

The six rules defined previous apply to these three functions: *createT*$(T, attributes)$ creates a type where *attributes* define its scope; *createR*$(R, types)$ creates a role with a set of associated types; *createU*$(U, roles)$ creates a MAC user with a set of possible roles.

According to the examples given previously, a user installing a new application would call *createT*(T_A) with T_A being the type of the new application. Adding new users would be done by $sc_{S_{ADM}}$ with *createU*(U_1) to create a new user U_1.

Now we define six additional rules, so that types, roles and users can be removed, and so that their properties can be changed.

The rules allowing to remove previously defined types, roles and MAC users are the following: *removeT*(T), *removeR*(R) and *removeU*(U).

The rules allowing to remove previously defined types, roles and MAC users are the following: *changeT*$(T, attributes)$ to modify the scope of the type T; *changeR*$(R, types)$ to modify the set of types associated to the role R; *changeU*$(U, roles)$ to modify the set of possible roles for user U.

3.4 Neutral policy language

The meta-policy framework doesn't rely on a specific MAC mechanism. The fact that different MAC mechanisms are configured in different ways lead to the creation of a neutral policy language, that can express a security policy in a generic way.

The creation of this neutral language has been done by comparing the semantics of the configuration grammars for SELinux and grsecurity. The features included in these grammars have been studied, resulting in a generic language that can be used to write a policy. This policy can then be translated to configure the underlying MAC mechanism.

Here is a partial example policy for the apache program, written using the neutral language (keywords are in french):

```
autoriser /usr/sbin/apache lire   /var/www
autoriser /usr/sbin/apache ecrire /var/log/apache
autoriser /usr/sbin/apache lire   /etc/apache
autoriser /usr/sbin/apache setuid
```

When applied to grsecurity, it would be translated that way:

```
/usr/sbin/apache {
    /var/www r
    /var/log/apache w
    /etc/apache r

    -CAP_ALL
    +CAP_SETUID

    connect {
        disabled
    }

    bind {
        disabled
    }
}
```

Note that connect and bind are disabled here because we didn't specify any rule allowing it. Such rules could be added easily, but here we can see that, by default, an application cannot connect or bind sockets.

3.5 Application to the Internet

Let us give a typical use case in context of an Internet provider. Typically, consumers will have an offer for the Internet connection but also for various services. Security packages become a predominant added-value service. Using our solution, the Internet provider can propose different kinds of security services.

First, the provider can set-up a propagation bus between its proper administration facilities and the various hosts and network components that are available at the customer location. The propagation bus enables to set-up the meta-protection policies. The reusable protection policy provides common protection rules controlling the usage of various resources located at a customer host or network component. For example, rules can be provided to control the usage of a Web server available at a customer host. Thus, the Internet

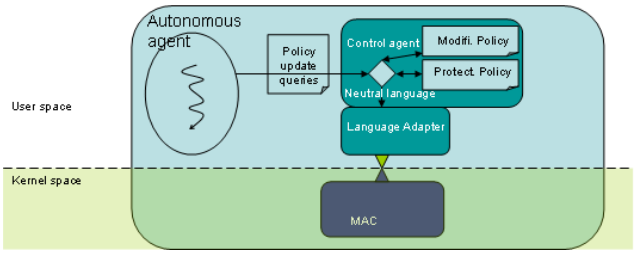

Figure 2: Local intelligence reacting to malicious code.

provider discharge the customer administrator from the job of securing network services. Our solution can be implemented on top of MAC kernel such as SELinux or grsecurity. But it can also the integrated as an extensible Access Control facility on classical Windows station. Our framework provides a generic and extensible solution. It can feet to the MAC approach has a robust security service. But, it can also be useful as a flexible Access Control that extend or wrap classical DAC solutions. Thus the framework is a common environment that can be projected onto various station and network components located at the customer place.

A modification policy enables updates during a disconnection phase. For example, an autonomous agent can learn the standard behavior of the local machine in order to compute updates for the protection policy (see Figure 2). That autonomous agent will submit a policy update query to the control agent in order to modify the protection policy according to the modification policy. One can imagine an autonomous agent react locally to specific attacks. It can be useful to react to some worm that start to close the Internet connection in order to prevent eradication from the network. In that case, the autonomous agent will learn which Security Context normally close the Internet connection and in case of any change, it will remove an irrelevant Security Context from the protection rule authorizing the connection closure. The removed SC has thus high probability to be a malicious code trying to close abnormally the Internet connection.

4 FORMAL MODEL AND VERIFICATION

Fixed protection rules are represented by a protection matrix similar to that of SLAT for SELinux [14]. From this protection matrix, we compute a so-called flow matrix containing information flow data. This transformation is based on an ad hoc interpretation of the operating system's kernel. Of course this is an approximation of the real information flow because the behavior of particular agents is not modelized. However reachability in the corresponding information flow graph (matrix product) provides sufficient information to detect the possible failure of usual security properties.

Dynamic protection rules raise a more difficult verification problem. In particular it is not sufficient to compute information flow from a set of fixed protection matrices. Indeed protection rules may change *at any time* during the information flows. We define the graph nodes as pairs of a security context x with the current protection matrix R. The graph's (forward information flow) edges are the following:

$(x,R) \rightarrow (y,R)$ if $Flow(R)(x,y)$, where $Flow(R)$ is the information flow matrix of R.

$(x,R) \rightarrow (x,R')$ if $Meta(R,R')$ where the $Meta$ predicate implements the static meta-policy rules.

From the point of view of information flow verification, this graph has two more issues compared to the static case:

Firstly, steps of the second kind can occur at any time, there can be arbitrarily many such steps. To deal with this issue, we make the same kind of approximation (at meta protection level) as in verification of static policies: everything than can be allowed is effectively allowed. More formally, we will consider the following information flow graph, where R is not anymore in the nodes:

$x \rightarrow_{\max} y \; if Flow(R_{\max})(x,y)$

Where $R_{\max}$ contains all allowable information flow:

$p \in R_{\max}(x,y)$ iff $(\exists R \in reachable(R_{init},Meta), R(x,y))$

where $reachable(R_{init},Meta)$ is the set of reachable protection matrices from the initial one following the static rules of $Meta$. This set is not trivial to characterize because of the possibility to add new security contexts dynamically. Is is easy to see that $\rightarrow_{\max}$ is such that $(x,R_{init}) \rightarrow^* (x',R') \Rightarrow x \rightarrow^*_{\max} x'$. So verification on this relation is correct with respect to $\rightarrow$.

The second issue is that steps of the second kind may create (infinitely many) new security contexts. This makes difficult the computation of $reachable(R_{init},Meta)$ and $R_{\max}$ in general. One important point is that security contexts having the same permissions and meta permissions will generate the same transitions in $\rightarrow_{\max}$. In other words (meta-)permissions define equivalence classes over security contexts. Therefore we focus on the $Flow(R_{\max})$ over equivalence classes, which again is not computable in general. In practice however it will be finite or regular, because there is a finite number of meta protection rules and therefore the set of reachable Rs is finite or regular depending on the expressive power of the meta-protection language.

We plan to solve the reachability problem either by theorem proving or application of well chosen reachability algorithms over the relation $\rightarrow_{\max}$ over equivalence classes. A preliminary computation of $Flow(R_{\max})(x,y)$ over equivalence classes is necessary.

5 FUTURE WORKS

Currently, the language to express the meta-protection rules as been designed. The development is still to be completed. Also, different communication systems are still under evaluation to carry out the meta-policy transmission for the meta-policy. For the moment the meta-protection must be copied manually on each machine.

Regarding the verification system, we still have to define an underlying communicating process model for the protocols implemented by the kernel for every interaction type. This will allow 1) the mechanical derivation of the information flow matrix from the protection matrix and 2) define a matrix of interaction protocols with appropriate composition on which the reflexive-transitive closure would express concrete scenarios for information flow.

REFERENCES

[1] B. Lampson, "Protection," in *Proceedings of the 5th Annual Princeton Conference on Information Sciences and Systems*, (Princeton University), pp. 437–443, 1971.

[2] D. E. Bell and L. J. La Padula, "Secure Computer Systems: Mathematical Foundations and Model," Technical Report M74-244, The MITRE Corporation, Bedford, MA, May 1973.

[3] K. J. Biba, "Integrity Considerations for Secure Computer Systems," Technical Report MTR 3153, The MITRE Corporation, Apr. 1977.

[4] D. Ferraiolo and R. Kuhn, "Role-Based Access Controls," in *15th NIST-NCSC National Computer Security Conference*, pp. 554–563, 1992.

[5] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[6] L. Badger, D. F. Sterne, D. L. Sherman, and K. M. Walker, "A Domain and Type Enforcement UNIX Prototype," in *Proceedings of the Fifth USENIX UNIX Security Symposium*, (Salt Lake City, Utah, USA), pp. 127–140, June 1995.

[7] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel," in *Proceedings of the 11th USENIX Security Symposium*, (San Francisco, CA, USA), Aug. 2002. `http://lsm.immunix.org/`.

[8] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," in *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01)*, USENIX, June 2001.

[9] S. Smalley, C. Vance, and W. Salamon, "Implementing SELinux as a Linux Security Module," tech. rep., NSA, May 2002.

[10] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau, "The Flask Security Architecture: System Support for Diverse Security Policies," in *Proceedings of the 8th USENIX Security Symposium*, (Washington, D.C., USA), pp. 123–129, Aug. 1999. `http://www.cs.utah.edu/flux/fluke/html/flask.html`.

[11] M. Pourzandi, A. Apvrille, E. Gingras, A. Medenou, and D. Gordon, "Distributed Access Control for Carrier Class Clusters," in *Proceedings of the Parallel and Distributed Processing Techniques and Applications (PDPTA '03) Conference*, (Las Vegas, Nevada, USA), June 2003.

[12] B. Spengler, "Detection, Prevention, and Containment: A Study of grsecurity," in *Libre Software Meeting 2002 (LSM2002)*, (Bordeaux, France), July 2002. `http://www.grsecurity.net/papers.php`.

[13] A. A. E. Kalam, R. El-Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin, "Organization based access control," in *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'2003)*, (Lake Como, Italie), pp. 120–131, June 2003.

[14] J. Guttman, A. Herzog, and J. Ramsdell, "Information flow in operating systems: Eager formal methods," in *Workshop on Issues in the Theory of Security (WITS)*, 2003.