ASPF: A Policy Administration Framework for Self-Protection of Large-Scale Systems

Ruan He Marc Lacoste
Orange Labs
Security and Trusted Transactions Dept.
{ruan.he,marc.lacoste}@orange-ftgroup.com

Jean Leneutre
Telecom ParisTech
Network, Mobility and Security Dept.
jean.leneutre@telecom-paristech.com

***Abstract*—Despite its potential to tackle many security challenges of large-scale systems such as pervasive networks, self-managed protection has been little explored. This paper addresses the problem from a policy management perspective by presenting a policy-driven framework for self-protection of pervasive systems called ASPF (Autonomic Security Policy Framework). Enforced authorization policies in a device are adapted according to the security context, both at the network and device levels. ASPF describes how an autonomic security manager may control OS-level authorization mechanisms supporting multiple classes of policies. Evaluation of an ASPF implementation shows that the framework enables effective self-protection of pervasive systems. ASPF is also applicable for autonomic security management of other types of large-scale infrastructures such as cloud environments.**

***Keywords*-Autonomic Computing, Self-Protection, Policy Management, Authorization, Pervasive Networks.**

I. INTRODUCTION

Advances in pervasive networking are rapidly taking us to the final frontier in security, revealing a whole new landscape of threats. In open and dynamic environments, malicious nodes may enter a network undetected, and various malwares may invisibly install themselves on a device. When roaming between heterogeneous networks, each with its own protection requirements, a device may also take advantage of security policy conflicts to gain unauthorized privileges. In embedded settings including limited and often unstable computing and networking resources, denial of service attacks are easier, with little lightweight security countermeasures. Finally, these decentralized, large-scale systems make end-to-end security supervision difficult. Administration by hand is clearly impossible, with the risk of some sub-system security policies not being up-to-date. These threats may only be mitigated with mechanisms highly adaptable to execution conditions and security requirements (e.g., supporting multiple authorization policies), with limited overhead. Above all, protection mechanisms should be self-managed [1], following the autonomic approach to security introduced by IBM [2], which defines a *self-protecting system* as a system that "can anticipate, detect, identify and protect [itself] against threats." [3].

To realize context-aware autonomic adaptations, the policy-driven paradigm has successfully demonstrated its flexibility and generality [4]: system functionalities are governed by a set of policies. As the context changes, other policies may be selected to activate within the system functions better adapted to its new environment. Unfortunately, this type of design was little applied to self-protection of pervasive systems.

In this paper, we validate the viability of this approach by presenting a policy-driven security management framework called *ASPF (Autonomic Security Policy Framework)*. ASPF describes the design of an autonomic security manager for pervasive systems. The framework is built on an earlier implemented OS security architecture called *Virtual Security Kernel (VSK)* [5]–[7] that specifies the managed security mechanisms. VSK implements kernel-level policy-neutral authorization, and supports dynamic policy reconfiguration, but without describing any control strategy of adaptation.

The original features of this framework are the following:

- ASPF enables the selection of the most appropriate authorization policy to be enforced in the device in order to match the estimated risk level of the current environment. Two levels of adaptation are possible, policies being tuned (or generated) according to the security context of the network and of the device.
- Policies are specified in an XACML extension for the attribute-based model of access control [8], which provides a fairly generic manner to describe permissions in open systems.
- An authorization architecture is also defined to refine the ASPF models, and is implemented above the VSK authorization mechanisms.

Performance, resilience, and security evaluation results show that the combined ASPF and VSK frameworks enable to achieve effective self-protection (Section IX-B evaluates the autonomic maturity level achieved with ASPF regarding security mechanisms). Moreover, ASPF is generic enough to be applied to other types of large-scale infrastructures such as cloud computing environments by defining the proper framework refinement.

This paper is organized as follows. After reviewing related work (Section II), we introduce briefly our self-protection architecture (Section III). We then describe the ASPF design principles (Section IV), policy model (Section V), framework (Section VI) and authorization architecture (Section VII). We present an ASPF implementation over the VSK mechanisms (Section VIII), and some evaluation results (Section IX). We finally show how ASPF may be refined for self-protection in cloud environments (Section X).

II. Related Work

Self-protection has so far been explored very little. While quite an early idea [2], it was discussed at the level of principles with few frameworks available, mainly for enterprise information systems [9], [10]. To orchestrate the components needed for autonomic security management, a policy-driven design [4], [10] seems promising, since the approach has been successfully applied to other self-* properties: indeed, several generic policy management frameworks [11]–[13] have been proposed to automate device and network reconfigurations to respond to context changes. Unfortunately, these frameworks hardly considered security. Notable exceptions are [13] for large organizations and [14] for pervasive systems which supports authorization and obligation policies. But with those frameworks, it remains unclear how to specify and federate authorization policies described in different security models to overcome heterogeneity of network security policies.

Three main elements seem to be missing: (1) descriptions of self-protection strategies; (2) specifications of security policies; and (3) authorization mechanisms supporting multiple policies and/or their reconfiguration. A promising approach for (1) is based on Domain-Specific Languages (DSLs) [15], but does not yet address security. For (2) and (3), one main challenge is the great diversity of access control models [16] proposed to describe policies. Policy-neutral access control (PNAC) languages [17] allow supporting several models, but lack real enforcement mechanisms. On the other end, several PNAC frameworks have been proposed [18], [19] but without generic enough specification languages. An interesting mid-term is described in [20] which combines a highly expressive security model (ABAC) [8], [21], a PNAC language (XACML) [22], and an authorization architecture. However, self-management of policies is not described. Further work is therefore needed.

III. Self-Protection Architecture

We now provide some background on the solution we explored for self-protection of pervasive networks [5]–[7].

We consider a pervasive system to be organized into a flat number of *clusters*, each containing a set of *nodes*. Nodes may join or leave a cluster dynamically. A *cluster* enforces a *cluster-level authorization policy*, applicable to nodes in the cluster. Nodes have various resource limitations, ranging from sensors to laptops, and enforce different *node-level authorization policies*. We now focus on a single cluster, but the approach can be generalized to any number of clusters.

For self-protection, we consider a security architecture divided into 3 abstract layers (see Figure 1). For each node, an *execution space* provides a running environment for application- or system-level services, encapsulated and manipulated as components. Node security management is performed in a *(security) control plane* using the VSK component. It oversees the execution space, both in terms of application-specific customizations and of enforcement of authorizations to access resources. Finally, a distributed *autonomic plane* supervises the VSK authorization policies in each node and performs the necessary adaptations at the cluster and node levels using several feedback loops. This paper provides an answer on how to design that layer using the ASPF framework.

VSK implements the managed OS-level security mechanisms in a node. It consists of a *Virtual Kernel (VK)* and an *Access Control Monitor* (ACM). The VK allows to reconfigure the *execution space* by providing run-time management functionalities over components and their bindings. It also efficiently controls access to the *execution space* resources, playing the role of an enforcement point for ACM decisions. Otherwise, the VK remains hidden in the background to minimize interactions with the *execution space* for performance optimization. The ACM is a decision engine allowing run-time selection of multiple authorization policies described in different access control models. Its design is compliant with the ABAC vision of access control [8], [21]: security attributes and permissions are separately managed (by an *Attribute Manager* for subject-attribute mappings, and by a *Rule Manager* for attribute-permissions assignments), and may be dynamically updated. More details regarding the VSK design may be found in [7].

We now present ASPF, a policy management framework which realizes an autonomic security manager above the VSK. The general idea is to adapt system security functionalities to the environment by context-aware change (tuning and/or generation) of authorization policies, such as adapting the policy strength to the ambient risk level.

The following sections describe the ASPF design principles, policy model (which specifies how to represent the managed authorization policies), security management framework, and resulting authorization architecture.

IV. ASPF Design Principles

We now describe the requirements for the framework.

A. Design Requirements

Several requirements should be met for such a self-protection framework.

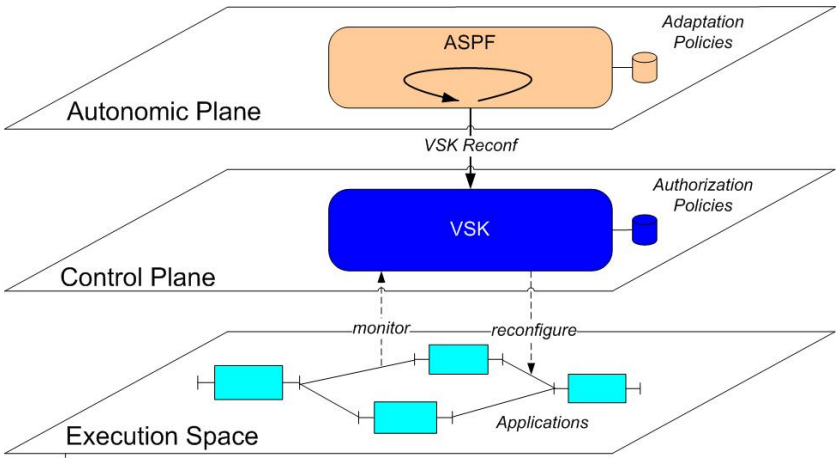

Figure 1. A 3-Level Architecture for Self-Protection.

1) Policy neutrality: Different types of authorization policies may be enforced in clusters and nodes. *Policy-neutrality* is thus mandatory to account for heterogeneous security domains by supporting several classes of authorization policies. Moreover, policies should be able to be *reconfigured dynamically* (including between different classes) when nodes move between security domains, or when the context changes.

2) Scalability: Pervasive systems are highly open and dynamic: nodes can enter and leave a network at run-time. The numbers of connected nodes may thus vary greatly in time, scaling network capacity both up and down, while the infrastructure remains unchanged. Scalability is thus a major challenge for the underlying protection framework, which should support both small- and large-scale systems.

3) Consistency: At the device level, a single system component (e.g., the security kernel [23]) usually controls all access to resources and enforces authorization policies. However, at the network level, each node still applies its own policy, but some nodes may share resources. The lack of a centralized module for enforcement of authorizations may lead to inconsistent network security policies. A solution for policy administration is thus required to guarantee consistency of distributed authorization policies.

4) User-friendly administration: Pervasive systems become increasingly complex, involving multiple users with different roles. Thus, the issue of system administration with minimal human intervention cannot be ignored. A security policy management framework should therefore simplify administration tasks and make system modifications transparent to users.

5) Context-awareness: Openness and dynamicity of pervasive networks induce rapid changes in the system context, calling for context-aware administration and protection. For instance, node availability may affect access privileges, as in ASRBAC authorization policies [24]. A node part of some clusters may have specific types of permissions that cannot be assigned to nodes in other clusters. Node migration between clusters may thus require update of access privileges. The management framework should thus select security functions based on evolution of the context.

6) Other Requirements: The security framework should also take into account requirements such as unified modeling of heterogeneous nodes, efficient protection mechanisms compatible with embedded constraints, or collaboration of decentralized security infrastructures.

B. ASPF Overview

Administration of authorization policies includes creation, deletion, and maintenance of access attributes and rules, and management of run-time constraints. To achieve this goal, ASPF applies the *autonomic approach* to make systems self-protected. Moreover, ASPF is *policy-driven*, i.e., the security behavior of the system is entirely governed by policies. The main distinguishing features of the framework are the following:

1) Policy-based management of authorization: The policy-driven approach is well adapted for administration of systems in open and dynamic environments: evolutions only trigger updates of applied policies, without changing the enforcement mechanisms. In our case, we use authorization policies to control protection. ASPF enables to modify, deploy, and enforce them through out the whole system.

2) Attribute-based authorization enforcement: Attribute-based access control [8], [20] is more suitable for open environments than traditional identity-based authorization: pervasive devices are not known by their exact names but by a dynamic set of *attributes*. This paradigm presents benefits in terms of expressivity and flexibility: it enables to support a large set of existing authorization policies, making policy-neutrality possible without developing a fully-fledged specific architecture. Separation of attributes from permissions also improves flexibility for dynamic policy reconfiguration.

3) Decentralized validation of authorizations: A scalable distributed system avoids using a central authority for validating authorizations. Our framework is based on a hybrid architecture using the concepts of *cluster* and *node*. Each node enforces a local authorization policy. Authorization policies of nodes inside a cluster are centrally controlled by a *cluster authority* which guarantees policy consistency between nodes. Policy synchronization between cluster authorities may be either centralized or decentralized. This architecture allows decentralized enforcement of authorization policies, while maintaining an efficient central control of policy deployment.

4) Integration of self-protection control loops: To satisfy the context-awareness requirement, ASPF regulates security using several self-protection feedback loops to select the authorization policy best fitting the system security context.

5) Self-configuration control loops for policy deployment: To guarantee consistency of decentralized policies, and facilitate system administration, self-configuration control loops allow the system to configure itself with minimal human intervention. Modification of chosen authorization policies will thus be automatically propagated through the whole network to guarantee consistent policy deployment.

V. ASPF Policy Model

In a pervasive system, different classes of authorization policies may be enforced: for instance, policies specified in the Domain and Type Enforcement (DTE) [25], Multiple Level Security (MLS) [26], or Role-Based Access Control (RBAC) [27] models.

ASPF allows to express those different models by describing authorization policies using the ABMAC (Attribute-Based Multi-Policy Access Control) model [20]. In this model, distinguishing features of system elements (subjects, objects, environment...) are described by attributes on which access decisions are based. Access attributes include principal identities, group membership, roles, security clearances, labels, or any other authorization information. Attributes are clearly separated from access rules, enabling independent modifications, e.g., activate/deactivate a role depending on location without reloading a full authorization policy.

```
<?xml version="1.0"?>
<DTEPolicy>
  <Rule>
    <Target>
      <SubjectAtt... name="domain">Trusted</SubjectAtt...>
      <ObjectAtt... name="type">Private</ObjectAtt...>
      <ActionAtt... name="operation">write</ActionAtt...>
    </Target>
    <Effect>grant</Effect>
  </Rule>
  ...
</DTEPolicy>
```

The ASPF policy model is shown in Figure 2. An ASPF policy consists of an *attribute map* and a set of *rules*. The map links system elements to their attributes. Elements may be subjects, objects, actions, or context data. Examples of

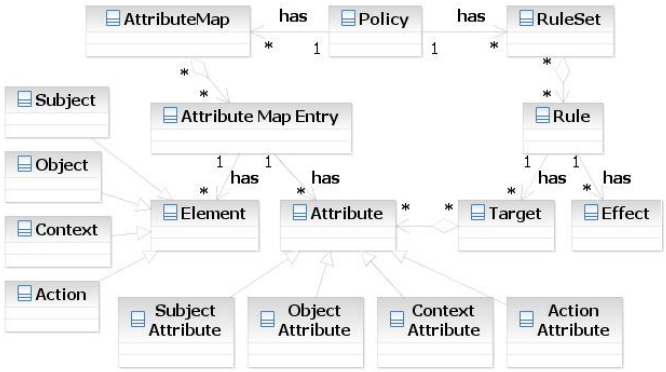

Figure 2. The ASPF Policy Model.

corresponding attributes include security domains, resource types, read/write operations, or location/time information. A rule contains a target, described by several attributes and an effect (allow/deny). A sample DTE policy is shown above, granting write authorizations to private resources for subjects in a trusted domain.

The result is a quite expressive model, while still remaining policy-neutral: as for XACML [22], specific authorization policies may be supported by refining the model through profiles. For instance, DTE, MLS, and RBAC policies are simply specified by defining the right types of attributes (domains, types, labels, roles...). Similarly, context-aware or history-based policies may be defined by adding specific context or history attributes.

As a drawback the processing of XACML policies may induce a performance overhead when dealing with policies with a large number of rules. This issue can be tackled by adopting an XACML policy optimization approach such as proposed in [28].

VI. ASPF Design

ASPF is a security management framework that governs authorization policies enforced by underlying VSK mechanisms. This section presents the ASPF design, based on several types of models.

A. Overall Design

The ASPF design is organized into three models:

- A *core model* describes system resources, security, and autonomic functionalities.
- An *extended model* refines the security and autonomic models for each type of resource.
- An *implementation model* describes the realization of the extended model, organizing functionalities into components to be implemented.

Those models are defined in the three steps shown in Figure 3. The core model consists of a *resource model*, a

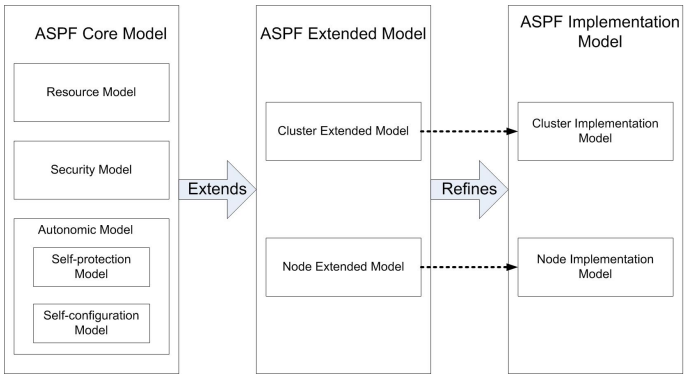

Figure 3. ASPF Overall Design.

security model and an *autonomic model*. These models are then refined into the *extended model* which involves a *cluster extended model* and a *node extended model* for cluster and node resources. Finally, these two models are refined into the corresponding implementation models.

B. ASPF Core Model

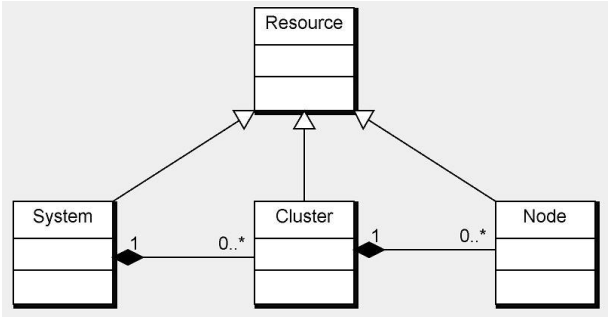

Figure 4. The Resource Model.

1) Resource Model: The *resource model* describes the structural organization of the system. The main concepts are those of *System*, *Cluster*, and *Node*, as shown in Figure 4:

- A *Resource* is the top-level concept which may be extended if the framework needs to be refined. It serves as coupling point with other models to describe different system functionalities.
- The *System* class represents the overall system to be protected (i.e., the pervasive network). It is organized into clusters.
- A *Cluster* is a coarse-grained structural unit including a set of nodes which collaborate to achieve some tasks, e.g., to provide a given service.
- A *Node* is the minimal structural unit. In pervasive networks, it represents a mobile device able to perform several functions and communicate with other nodes.

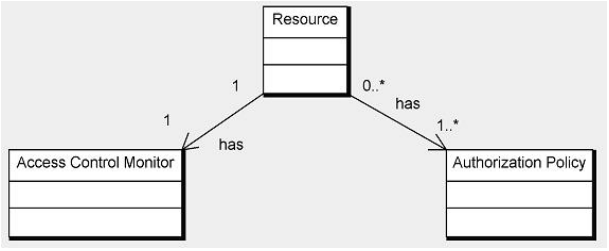

Figure 5. The Security Model.

2) Security Model: The *security model* specifies the authorization functionality to control access to *Resources*. The main concepts are those of *Access Control Monitor (ACM)* and *Authorization Policy* as shown in Figure 5:

- The *ACM* is a reference monitor which controls all access requests to resources.
- The *Authorization Policy* expresses conditions under which authorizations are granted or denied. It is specified according to the policy model previously described.

3) Autonomic Model: The *autonomic model* specifies how self-configuration and self-protection are achieved in the system. The *self-protection model* adapts authorization policies according to evolution of the context. The *self-configuration model* customizes authorization policies according to resource types, user preferences, or administrator-defined security governance strategies.

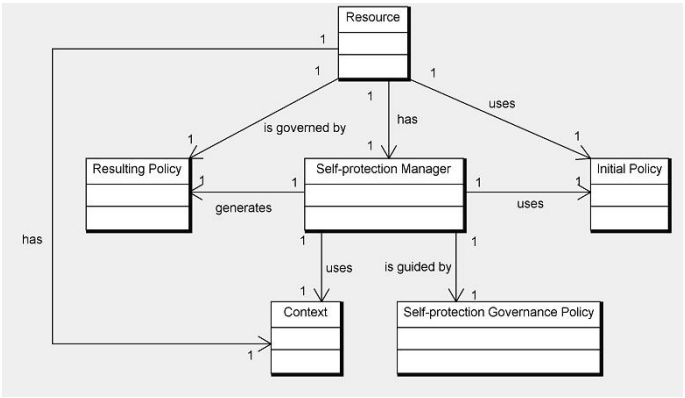

Figure 6. The Self-Protection Model.

The *self-protection model* describes how adaptations (selection of adequate security counter-measures) are launched at run-time, driven by evolution of the security context. Adaptations are performed both at the cluster level and at the node level. The main element of the model are the following, as shown in Figure 6:

- The *Self-Protection Manager* controls and orchestrates all activities related to self-protection. Its main role is to monitor the context and update authorization policies.
- The *Self-Protection Governance Policy* captures the administration strategy for self-protection. It drives

decision-making, specifying how to select the right authorization policy according to context information.
- The *Context* captures all information about the system environment which may influence such decisions.
- The *Initial Policy* is the current authorization policy, input for the context-aware security adaptation process.
- The *Resulting Policy* is the authorization policy output of the security adaptation process. This policy is generated by the *Self-Protection Manager*.

Once a *Resulting Policy* has been generated, this new policy should be propagated through the whole network for enforcement. As ASPF targets large-scale distributed systems, global policies should be translated into local ones to be enforced by each node. The *self-configuration model* specifies this translation process. The main element of the model are the following, as shown in Figure 7:

- The *Self-Configuration Manager* is the component in charge of the self-configuration process. It generates a *Resulting Policy* based on an *Initial Policy* according to a *Self-Configuration Governance Policy*.
- The *Self-Configuration Governance Policy* contains the guidelines for the translation process. It may be specified by condition-action rules.
- The *Initial Policy* is policy output of the self-protection model, input for the self-configuration process. It typically represents the new network security policy.
- The *Resulting Policy* is a policy derived from the *Initial Policy*, customized for each resource. It typically represents the new node security policy, adapted to the node-specific setting, e.g., by filtering all network access control rules not involving directly that node to comply with node computational limitations.

C. ASPF Extended Model

The role of the ASPF core model is to describe the security framework independently from the type of large-scale system. However, to be useful in practice, the framework must be described in a concrete setting. This is the purpose of the *extended model* which specifies the security framework for a specific type of large-scale system such as pervasive networking or cloud computing infrastructures. We now present an extended model for the pervasive setting which was the core focus of our study. However, another extension for cloud environments is detailed in Section X.

As pervasive systems are modeled as clusters and nodes, two extended models are defined to describe self-management of security at the cluster and node levels.

1) Cluster Extended Model: The main elements of the model are shown in Figure 8.

- The *Cluster Self-Protection Manager* captures the overall intelligence for self-protection of a cluster, coordinating the different necessary components.
- The *Cluster Context* class captures information about the context of a cluster. It may be specified using a more detailed context model such as DEN-ng [29] describing multiple dimensions of context.
- The *Cluster Self-Protection Governance Policy* captures the strategy to select the most adequate security function based on the cluster context.
- The *Cluster Initial Authorization Policy* is the starting point for the security adaptation process. It may be initially one of a set of predefined policies.
- The *Cluster Resulting Authorization Policy* is the result of the security adaptation process, and is generated by the *Cluster Self-Protection Manager* according to the current cluster status. That policy will then be applied to all nodes of the cluster.

The ASPF modular design into several models makes it more easy to select only the features necessary for the considered setting: compared to the core model, the cluster extended model only integrates the self-protection model. Authorization and self-configuration are left aside since: (1) policy enforcement is performed directly in the nodes; and (2) policy propagation towards nodes will be specified in the node extended model.

2) Node Extended Model: The main elements of the model are shown in Figure 9.

- The *Node Self-Configuration Manager* coordinates the components for self-configuration at the node level, i.e., to propagate adaptations decided at the cluster level. Such operations will be performed according to the *Node Self-Configuration Governance Policy*.
- The *Node Self-Protection Manager* orchestrates the components for self-protection of a node. Such operations will be performed according to the *Node Self-Protection Governance Policy* which describes reactions (i.e., authorization policies) to apply in security-sensitive situations, based on the *Node Context*.
- The *Node Resulting Authorization Policy* is the final output of the ASPF framework: after the adaptation process, both at the cluster and node levels, this policy will be installed inside the node for access control enforcement by the *Node Access Control Monitor*.

Overall, at the node level, self-management of security is a combination of self-configuration and self-protection: the result of the security adaptation process at the cluster level (*Cluster Resulting Authorization Policy*) is transformed into a *Node Resulting Authorization Policy* (self-configuration). Updates on the *Node Resulting Authorization Policy* will also be performed based on the node context (self-protection).

D. ASPF Implementation Model

The previous models are now refined at the implementation level, different implementation architectures being possible. In the sequel, we present an implementation model which fulfils the requirements presented in Section IV-A.

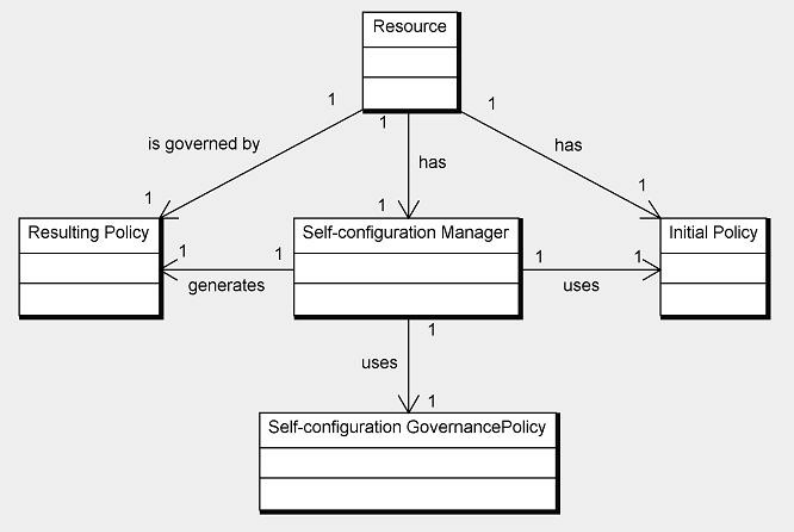

Figure 7. The Self-configuration Model.

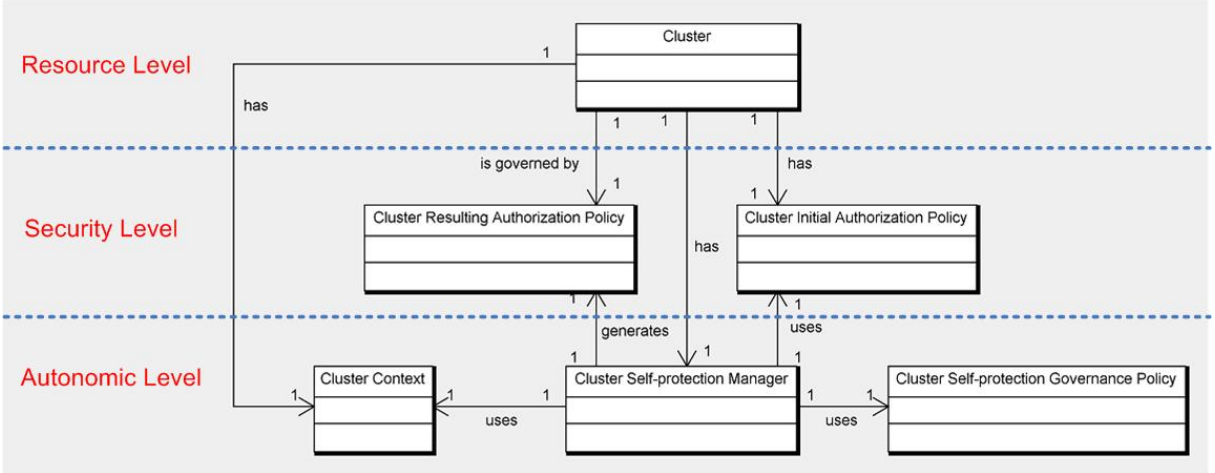

Figure 8. A Cluster Extended Model.

1) Cluster Implementation Model: The elements of the model are shown in Figure 10.

- The *Cluster Authority* component implements the *Cluster Self-protection Manager* class. It coordinates all self-protection tasks in the cluster.
- The *Cluster Context Monitor* provides a representation of the cluster security context. It aggregates low-level inputs from different sources (system/network monitoring probes, sensors,...), relying on context management infrastructures or intrusion detection systems.
- The *Cluster Authorization Policy Repository* contains a set of initial cluster authorization policies to enforce protection within different potential situations.
- The *Cluster Governance Policy Engine* generates security adaptation strategies to tune authorization policies to the environment, e.g., use DTE (resp. MLS) policies in a friendly (resp. hostile) setting. It may also define new policies to cope with unknown situations.
- The *Cluster Resulting Authorization Policy* is the output of the cluster-level security adaptation process.

2) Node Implementation Model: The main elements of the model are shown in Figure 11.

- The *Self-Configuration Manager* and *Self-Protection Manager* functionalities are implemented by two components, the *Node Authority* and the *Node Adapter*. The *Node Authority* typically resides on a server at the cluster-level, while the *Node Adapter* is a component local to each node. The *Node Authority* is the main control point to administer security configurations and customize authorization policies. The *Node Adapter* is a local security controller in the node with two roles. It is a proxy for the remote *Node Authority*, executing its

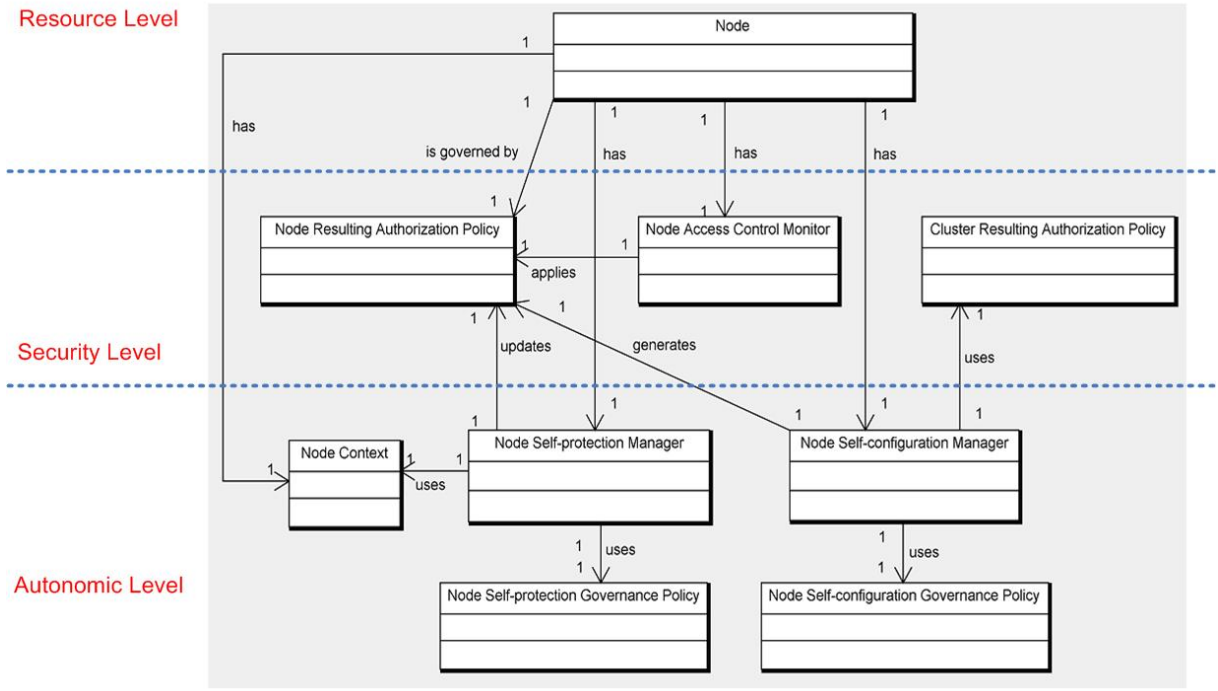

Figure 9. The Node Extended Model.

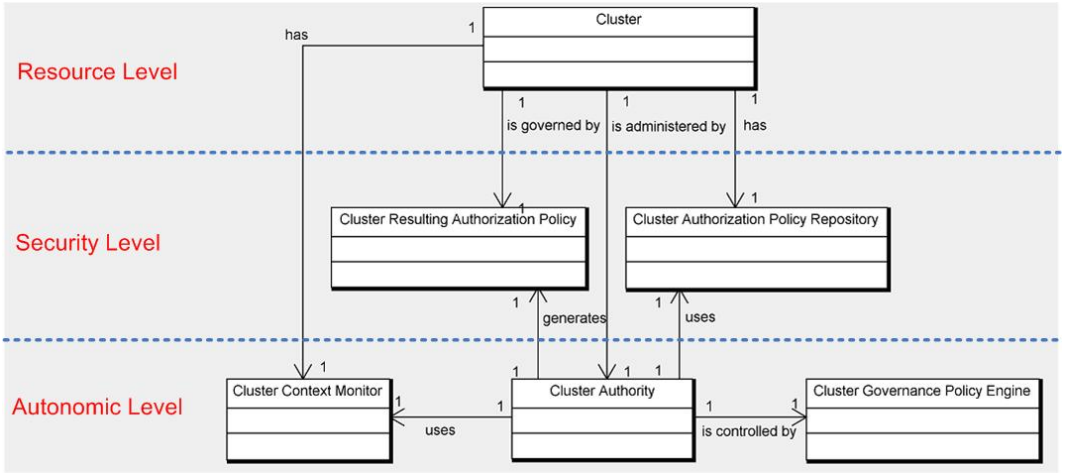

Figure 10. The Cluster Implementation Model.

decisions and installing in the node authorization policies customized at the other endpoint. It also realizes node-level self-protection to adapt node authorization policies based on the node context.

- The *Node Profile* refines the *Node Self-Configuration Governance Policy* by describing the node capabilities (CPU, memory, storage...). As a cluster might contain many nodes, a large part of cluster policy rules might not be relevant for each node and should be filtered. In our design, node-level self-configuration is viewed as filtering the cluster authorization policy according to constraints described in this profile.
- Other components such as the *Node Context Monitor* or the *Node Governance Policy Engine* play the same roles as on the cluster side, but for the node setting.
- The *Node Resulting Authorization Policy* is the final output of the node-level security adaptation process. The corresponding access control rules may then be enforced in the node with a lightweight authorization overhead thanks to the underlying VSK OS.

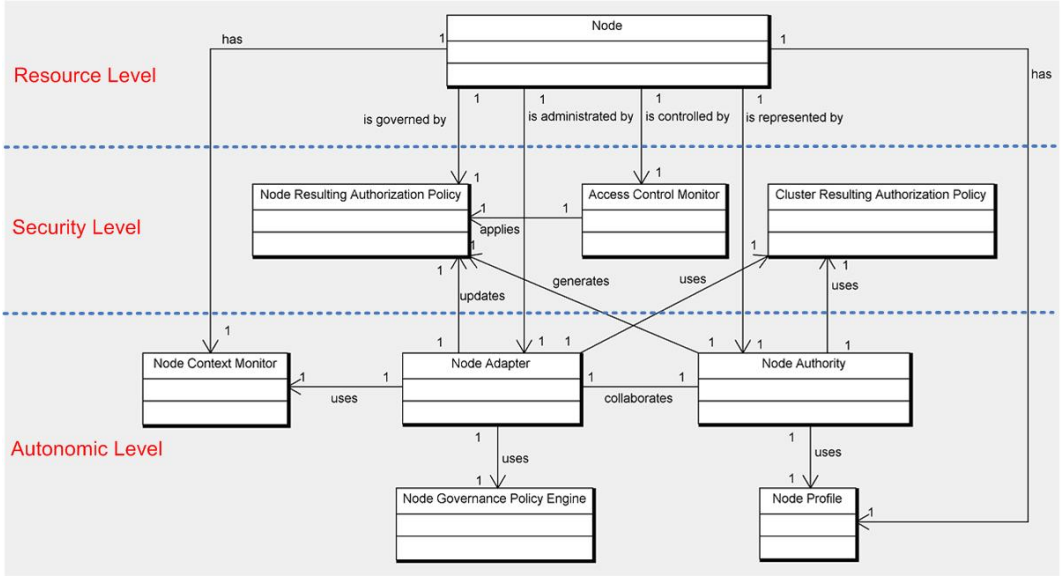

Figure 11. The Node Implementation Model.

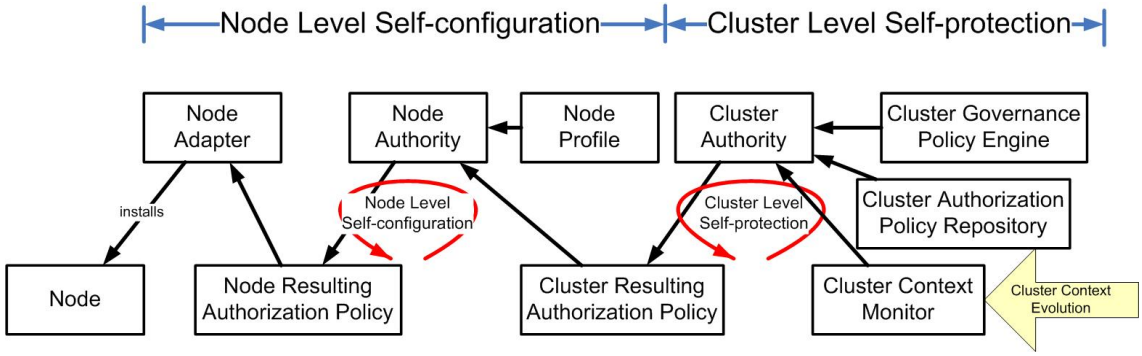

Figure 12. Cluster-Level Self-Protection Loop

E. A Double Control Loop for Self-Protection

ASPF regulates security at two levels, using separate feedback loops, both at the cluster and node levels. The previous implementation components interact as follows.

1) Cluster-Level Self-Protection: This loop shown in Figure 12 aims to mitigate threats against a cluster. The *Cluster Context Monitor* aggregates security-relevant events to reach a representation of the cluster security context. It notifies the *Cluster Authority* in case the context changes. The *Cluster Authority* then updates the *Cluster Authorization Policy*, according to the strategy specified in the *Cluster Self-Protection Governance Policy*. This operation may be performed by selecting a predefined stronger/weaker policy from the *Cluster Authorization Policy Repository*.

Nodes have severe resource limitations, for instance in terms of computing capabilities or power consumption. Execution must therefore be optimized. The chosen cluster policy is further interpreted by each node according to its specificities (CPU, memory, battery, etc.) captured in the *Node Profile*, generating a new node authorization policy (*Node Resulting Authorization Policy*). Policy rules not relevant for each node are notably discarded. This policy is then loaded into the node authorization sub-system for enforcement. This customization improves efficiency and scalability. It also makes the system more manageable by reducing the number of authorization rules.

2) Node-Level Self-protection: A simpler loop also operates at the node level to defeat attacks on a single node as shown in Figure 13). Based on information about the node security context (captured by the *Node Context Monitor*), this loop tunes security attributes such as assigning a different role to a subject in order to reduce his privileges in a hostile environment, without modifying the rest of the node authorization policy. For instance, when a node is attacked, the security level of a highly sensitive resource could be increased from *Confidential* to *Top Secret* to minimize possibilities to access the resource.

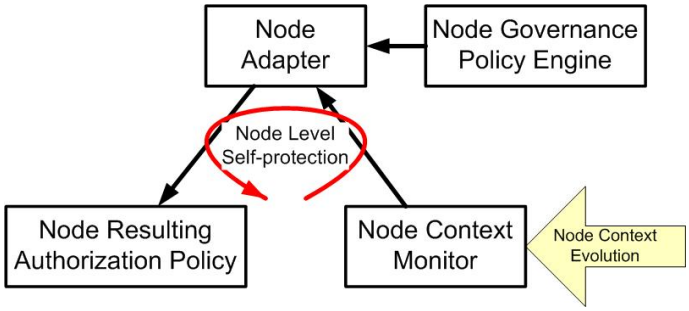

Figure 13. Node-Level Self-Protection Loop

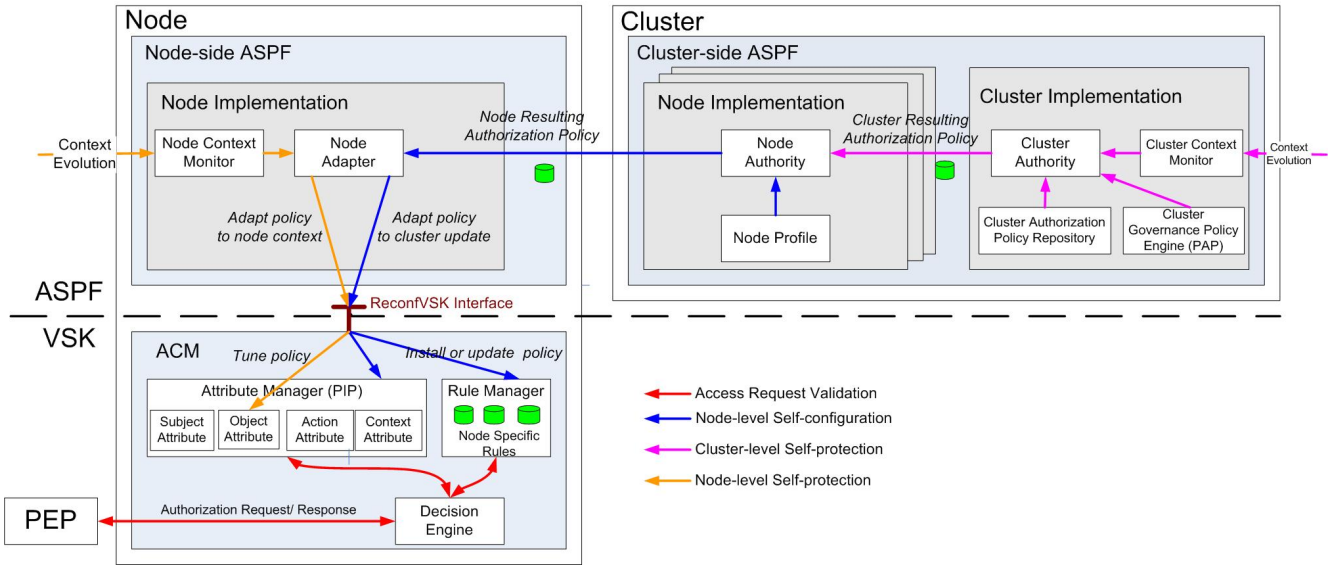

Figure 14. Authorization Architecture

VII. Authorization Architecture

The ASPF authorization architecture integrates the self-configuration and self-protection models into the XACML authorization framework (see Figure 14). XACML defines 4 main components for policy enforcement (PEP), decision-making (PDP), administration (PAP), and management of attributes (PIP) [22]. In the VSK architecture: the PEP is the *Virtual Kernel (VK)* component, which enforces authorizations on execution resources; the PDP is the *Decision Engine* component; the PIP is the *Attribute Manager (AM)* component that provides additional information for access validation. The authorization policy is stored in the *Rule Manager (RM)* component.

Access requests to resources (located in the execution space) are forwarded to the Decision Engine and transformed to an ABAC-compliant request. Attributes are fetched from the Attribute Manager, and the request is validated against the authorization policy. The decision is then enforced by the VK by reconfiguring the execution space to establish access to requested resources.

ASPF may be seen as an enhanced PDP. Pure decision-making is extended with autonomic capabilities to generate or tune the security policies contained in the VSK ACM based on policy sets written by a cluster network administrator.

Figure 14 shows how ASPF realizes the two self-protection control loops described in Section VI-E. The cluster-level self-protection model together with the node-level self-configuration model achieve a global control loop which updates both rules and attributes of authorization policies according to cluster context and node profile information. The node-level self-protection loop tunes security attributes based on node context information. The overall architecture not only performs access control enforcement and decision-making. It also improves management of authorization policies, notably by enabling context-aware adaptations thanks to autonomic features.

VIII. Implementation

A first prototype implementation was realized including a set of devices running a kernel composed of VSK and of a node-side ASPF component (implemented using the THINK [30] component-based OS framework), and a cluster authority server (implemented in Java). DTE, MLS and RBAC security policies are currently supported, with 10 subjects (threads) and 60 objects (system calls) to model a typical embedded OS environment, and 3 security levels for the cluster security context. Cluster policies are filtered according to active subjects or objects described in the node profile. The resulting attribute mappings and rules are then loaded inside the VSK via a dedicated reconfiguration interface `ReconfVSK` allowing to change dynamically security attributes and policy rules.

IX. Evaluation

The self-protection capabilities of the framework (ASPF+VSK) were evaluated in terms of overall response time and resiliency to attacks. A qualitative assessment of the security of the framework is also given. All measurements were performed on a 2.7GHz DELL OptiPlex 740 desktop PC with Linux/Ubuntu 9.04 and 1GB of RAM, on which are run the cluster authority server and node simulations.

A. End-to-End Response Time

We measure the overall latency to complete a full self-protection loop for adaptations at the cluster and node levels. Evaluation results for each step of the loop are shown in Figures 15a and 15b for different types of security policies.

In the first benchmark, detection of an attack on a cluster of 100 nodes in a steady state is simulated by direct update of the cluster security context. In practice, this step would be performed by an Intrusion Detection System (IDS) such as Snort, with 1ms as typical order of magnitude for attack detection and countermeasure initiation. The next steps are generation of a node-specific policy (given times are averaged on the number of nodes), invoking the node VSK to load the policy, kernel reconfiguration with the new policy, and return to the steady state. The overall latency averaged over different security models is 33.92ms.

In the second benchmark, attacks are detected by a node context handler. The next steps include invoking the VSK, tuning security attributes to adapt to the new security context, and returning to a steady state. The measured overall latency for this adaptation loop is 1.15ms.

Overall, the adaptation response times seem reasonable, since the time between two policy reconfigurations is typically from a few seconds to one minute, for instance when switching between wireless networks in different locations. As expected, node-level adaptations are much lighter than cluster-level reconfigurations. This is in part due to the ABAC approach: the same authorization rules may be applied, only attributes values being tuned. For highly dynamic environments, this design makes self-protection more lightweight, allowing to follow small variations of the context, without regenerating a full policy.

B. Resilience

To measure the effectiveness of self-protection using ASPF, we use the methodology for benchmarking self-* capabilities of autonomic systems proposed in [31] based on injection of disturbances (see Figure 16a). The idea, coming from dependability benchmarks, is to introduce in the System Under Test (SUT) disturbances in the form of attacks or faults, and to measure the impact on the performance workload. This type of benchmark, already used to assess self-healing abilities, measures how well the SUT adapts to the injected changes in terms of speed of recovery, impact on performance, etc.

In our case, the SUT is the set VSK+ASPF on which is applied a workload to validate access requests from the execution space. We measure the impact on throughput (number of requests per second validated by the VSK, averaged over a sliding sampling time window τ) of updating security policies to respond to injected attacks. An attack from a malicious node is simulated by directly changing the cluster security context at the beginning of an injection slot, and waiting from the SUT to come back to a steady state. The results are shown in Figure 16b for $\tau = 1ms$ and $\tau = 0.16ms$, which is about the latency value for an end-to-end reconfiguration. The decrease in throughput due to security adaptations depends on the sampling slot value: 89% for $\tau = 0.16ms$ (worst case), but only 15% for $\tau = 1ms$ (standard situation). These results show that the system is able to protect itself effectively with a reasonable performance cost. The recovery time is almost immediate for $\tau = 0.16ms$, and about 2ms for $\tau = 1ms$. Thus, the system is able to complete successfully its reconfiguration in times which are largely acceptable. These metrics tend to show that ASPF provides self-protection with minimal impact on system resources.

C. Security Evaluation

Evaluating the quality of the autonomic response is harder: does the system remain secure after a security reconfiguration? To avoid rogue third parties to directly update node authorization policies inside the VSK, a single reconfiguration interface (`ReconfVSK`) is introduced as unique entry point to control the VSK. This interface remains internal to a node, to avoid policy update requests coming from the network aiming to lower node security settings. ASPF behaves as a distributed security management plane which guarantees complete mediation over this interface: all authorization policy modifications may only be issued by the *Node Adapter*, *Node Authority*, and *Cluster Authority*

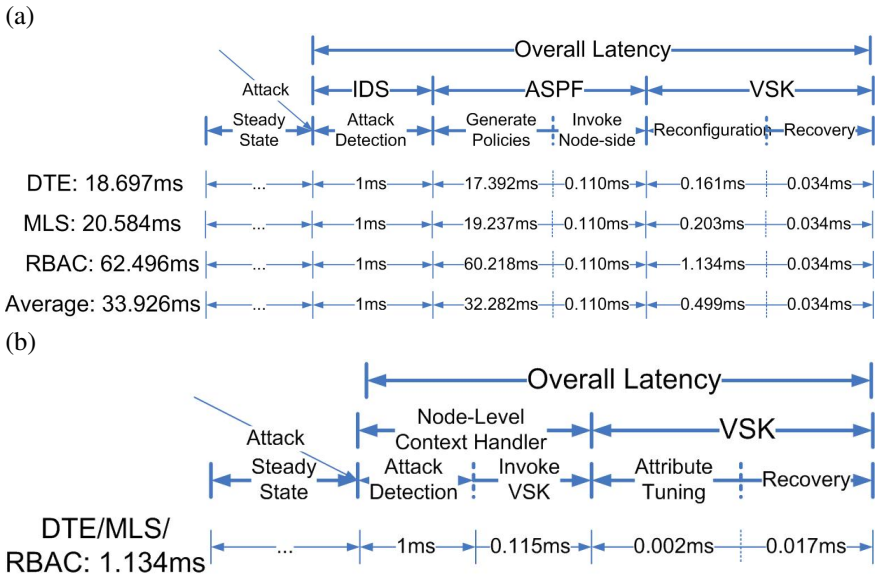

Figure 15. Self-Protection Latencies: (a) Cluster-Level; (b) Node-Level.

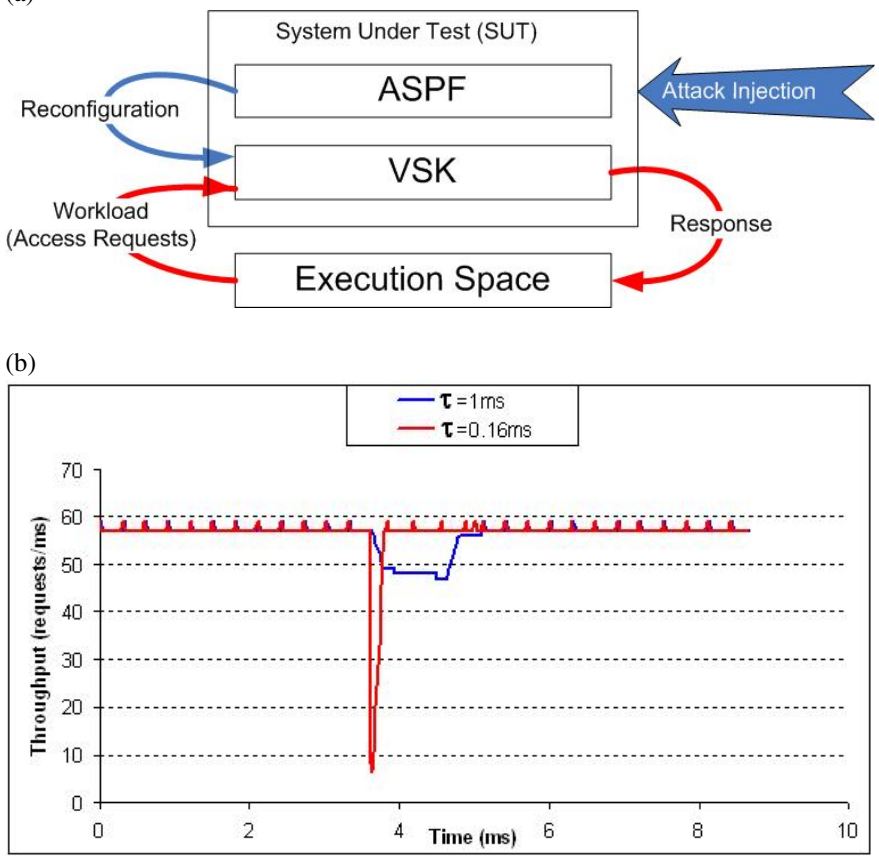

Figure 16. Benchmarking Self-Protection Capabilities: (a) Principle; (b) Results.

components along a trusted path. The link between node-side and cluster-side ASPF components is also assumed to be a secure, authenticated channel to avoid man-in-the-middle attacks or rogue cluster authorities. Finally, an MMU-like hardware mechanism in the node prevents circumventing the *Node Adapter* component. These features qualify ASPF as a strongly protected management plane over VSK authorization mechanisms.

The underlying VSK infrastructure which serves as foundation for ASPF was also evaluated in terms of security. Three main threats were identified:

1) An application-level component gains illegitimate access rights through an existing binding. For bindings, the current THINK framework does not distinguish between *read* and *write* actions, i.e., a binding for a *read* access action could be used to perform a *write* invocation. Thus, a malicious component could perform a privilege escalation through such a binding, as there is currently no checking mechanism in THINK to prevent it.
2) An application-level component illegally accesses another such component by bypassing VSK protection. The VSK limits itself to checking and validating access requests based on presented attributes. However, an illegal access bypassing the kernel remains possible, as in THINK an invocation may directly access to a physical address without any control.
3) An application-level component illegally accesses the VSK. This threat is an extension of the previous one: since VSK is also built on the THINK framework, access to the kernel by directly jumping to a physical address may be possible.

The first threat may be mitigated by extending the definition of interfaces of the THINK framework with access action types. During compilation, checking may be included to determine if method invocations match authorized access types. The second and third threats correspond to bypass attacks. A MMU-based hardware mechanism is usually used to avoid circumventing reference monitors. Such mechanisms may be used to prevent bypass of VSK authorization checks. One MMU solution for component-based OSes was implemented in *CRACKER* [19]: the MMU organizes components into different memory pages according to their security level, and performs additional checking for inter-page invocations. For some hardware platforms like AVR or ARM which do not support MMU, a tool was proposed for code checking which replaces memory access by a pointer to a manager for security policy validation [32]. We believe that isolation between application-level components and the VSK may be achieved through these two categories of solutions.

X. Applying ASPF To Cloud Infrastructures

We now further validate the framework design by showing through a short case study that ASPF is generic enough to be applicable to other types of large-scale systems than simply pervasive networks. In the sequel, we focus on cloud computing infrastructures. We first recall some of the main security issues of those environments (Section X-A), highlighting the need for self-protection mechanisms. We then present the targetted self-protection scenarios (Section X-B). We finally show how the ASPF core model (Section X-C), extended model (Section X-D), and authorization architecture (Section X-E) may be refined to realize and coordinate several self-protection loops in a cloud setting.

A. Towards Self-Protecting Clouds

Cloud computing raises many security challenges [33], notably due to vulnerabilities introduced by virtualization of computing resources, and unclear effectiveness of traditional security architectures in fully virtualized networks. One of the main issues is how to guarantee strong resource isolation, both on the computing and networking side in a multi-tenant environment.

Few solutions are available, usually addressing only one of the two aspects [34], [35]. The extremely short response times required to activate system defenses efficiently, and the impossibility of manual security maintenance call for a flexible, dynamic, and automated security management of cloud infrastructures, which is clearly lacking today. A framework enabling self-protection of a cloud infrastructure could provide answers to some of those challenges, making ASPF an interesting candidate to reach this objective.

In the cloud, virtualization has two facets:

- *Computing resources* are abstracted away from the hardware in the form of *virtual machines (VMs)* isolated by a hypervisor on each server of a data center. Threats come at two levels of granularity: at the host level, through weaknesses either in the VM (guest OS) or the hypervisor; and at the cloud-level, mainly in the form of network-level attacks found in traditional security environments (e.g., DDoS). An autonomous security management framework for the cloud should thus put in place self-protection loops at each of those two levels.
- *Network resources* (routers, firewalls,...) themselves become virtualized, e.g., as virtual appliances. Network zones where traffic could be separated physically or logically using VLANs or VPNs are replaced by *logical security domains* which may have variable boundaries. It is thus critical to be able to manage security autonomously in such "islands". The security management framework should thus also provide self-protection abilities in logical security domains, called *VSBs (Virtual Security Domains)* in the sequel.

B. Cloud Self-Protection Scenario

We explore the realization of *adaptable quarantine zones*: a number of VMs considered as compromised are isolated from the data center temporarily. Confinement may be lifted when the risk has decreased, and the VMs not considered hostile any more.

We assume that on each physical machine of the data center is installed a firewall component which allows to control strictly communications between VMs: an authorization policy specifies which interactions are allowed/forbidden. This virtual firewall may for instance be located in the

domain 0 of a Xen hypervisor. Additional firewalls may also be placed at the cloud level to control inter-machine communications. The authorization policy is reconfigurable dynamically according to the estimated level of risk. Self-protection of the virtualized infrastructure then consists in adapting this set of policies according to the execution context of the data center, more or less hostile. Depending on alerts generated from an IDS (local or distributed in the data center), the most adequate authorization policy is autonomously selected, and installed in the different firewalls to realize hardened control over VM communications, and enforce the quarantine zone (see Figure 17).

In what follows, the quarantine zone is implemented at three levels of granularity: (1) within in physical server (*machine-level self-protection*); (2) within a VSB (*logical self-protection*); and (3) at the cloud level (*system-level self-protection*). The next sections describe how the ASPF core and extended models may be refined to realize those 3 self-protection loops.

C. ASPF Core Model

1) Resource Model: This model describes the organization of a cloud infrastructure (see Figure 18). As for the pervasive case, entities derive from a generic *Resource* class.

- The *System* class represents the overall cloud infrastructure to be protected, physically composed of a set of machines and logically divided into several *VSBs*. Both physical and logical isolation are realized through *Authorization Policies*.
- A *Machine* is a server in the data center. It hosts several *Virtual Machines (VMs)*, isolated by an hypervisor, which may create, destroy, or migrate VMs on demand.
- The VM is the first-class architectural component of the cloud. It runs a guest OS on top of the hypervisor, which manages VM resources.
- The VSB is a logical unit of VM isolation, e.g. to compartimentalize different services. VMs belonging to a VSB may be distributed on several machines. VSBs may be strictly isolated between each other using network-level mechanisms.
- A *Local VSB* contains all VMs of a VSB which reside on a given machine. It realizes local isolation from VMs of other VSBs in the machine. VM isolation at the VSB level is achieved by collaboration between all the corresponding Local VSBs.

2) Security Model: As for the pervasive case, access to resources is controlled by authorization policies. However, in the cloud, the security model features several types of policies since the resource model is richer (see Figure 19).

- The *System Authorization Policy* contains all access permissions to cloud resources. It will be enforced by the *System ACM* component at the cloud level.
- The *VSB Authorization Policy* contains access permissions in the scope of a VSB: it controls VM access at a logical level (the VSB security domain), regardless of the VM physical location. If we assume that access between two VMs belonging to different VSBs is always denied (strict isolation between VSBs), the System Authorization Policy may be viewed as the collection of VSB Authorization Policies. Policies in each VSB may be specified in different authorization models (e.g., DTE, MLS, or RBAC), as each VSB is a security island where policies may be administrated in a specific manner.
- The *Local VSB Authorization Policy* is the projection of the VSB Authorization Policy inside a machine, and thus corresponds to two types of situations: VMs are co-located on the same machine; or VMs reside in different machines. In the former situation, access may be directly validated by at the machine-level. The latter calls for inter-machine collaboration.
- The *Machine Authorization Policy* is the collection of Local VSB Authorization Policies for all Local VSBs in the machine. Due to possible heterogeneity of authorization models between VSBs, in the general case, the Machine Authorization Policy will be a set of Local VSB Authorization Policies specified in different models. This policy will be enforced by the *Machine ACM* component residing on each machine.

In our cloud model, to control inter-VM communications, policy enforcement is performed both at the machine level and the system level. We describe next a simple solution, other alternatives being possible.

If the VMs reside on the same machine, the Machine ACM applies the Machine Authorization Policy to validate the request. Since by default the VMs reside in the same VSB, validation is straightforward by enforcing the corresponding Local VSB Authorization Policy. However, since Local VSB Authorization Policies may be described in different models, a policy-neutral solution is required for access control enforcement at the machine level. Using ABAC for policy specification allows to achieve that goal as in the pervasive case.

If the VMs reside in different machines, the Machine ACM of the requesting VM checks in its Machine Authorization Policy whether this VM has permission to access an external machine. Control is then transferred to the System ACM which checks in the System Authorization Policy whether inter-machine communication to the target VM is allowed. Finally, the Machine ACM of the target VM checks that requests to this VM coming from a remote machine are allowed. Such a three-step validation of requests allows authorization to be more efficient and scalable (local policies do not deal with inter-machine communications) and to check consistency of distributed policies at the system level.

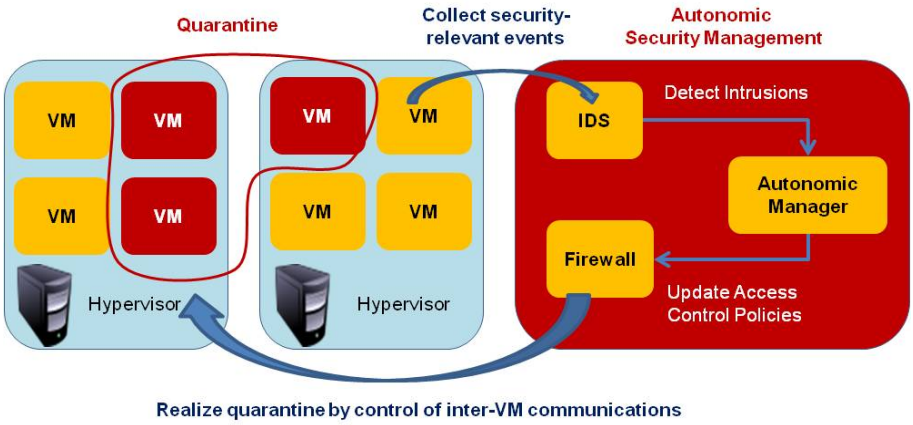

Figure 17. An Adaptable Quarantine Zone.

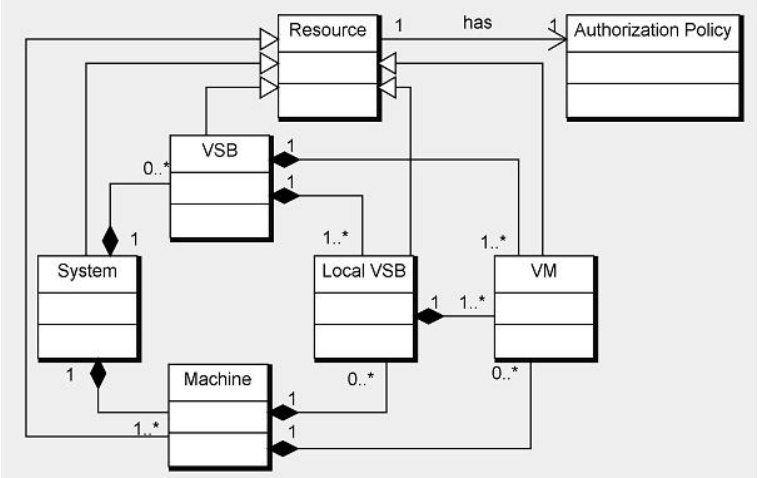

Figure 18. Cloud Resource Model.

D. Extended Models

The extended models describe the realization of several self-protection loops at different levels of granularity in the cloud, to address threats targeted at a machine, a logical security domain (i.e., a VSB), or the cloud itself by updating the corresponding authorization policies.

1) Machine Extended Model: If a malicious VM compromises the hypervisor [36], [37], the threat may spread to all the VMs residing on the machine, which may need to be confined. Defeating such attacks is the objective of this self-protection loop (Figure 20).

When an attack is detected by the *Machine Context* monitor, the *Machine Self-Protection Manager* applies a *Machine Self-Protection Governance Policy* to adapt the Machine Authorization Policy to the current situation, policy which will be propagated to the authorization policies of each Local VSB on the machine. At the same time, the manager collaborates with the *System Self-Protection Manager* to determine whether further counter-measures should be triggered at the cloud level.

2) VSB Extended Model: This self-protection loop (Figure 21) addresses a wider scope: it aims to defeat attacks which have spread into a logical security domain, e.g., by isolating compromised VMs. The VSB Authorization Policy is updated to fit the evolving *VSB Security Context* – those modifications are propagated to the System Authorization Policy to maintain policy consistency. A self-configuration loop is then launched to refine this policy into corresponding Local VSB Authorization Policies – the modifications being propagated to the Machine Authorization Policies.

3) System Extended Model: Two events may launch the system self-protection loop (Figure 22): detection of a cloud-level attack through *System Context* monitoring; or a request from a Machine Self-Protection Manager for increased counter-measures, faced with an anomaly which cannot be handled at the machine level alone. Regarding self-protection, the *System Self-Protection Manager* tunes the System Authorization Policy following the run-time

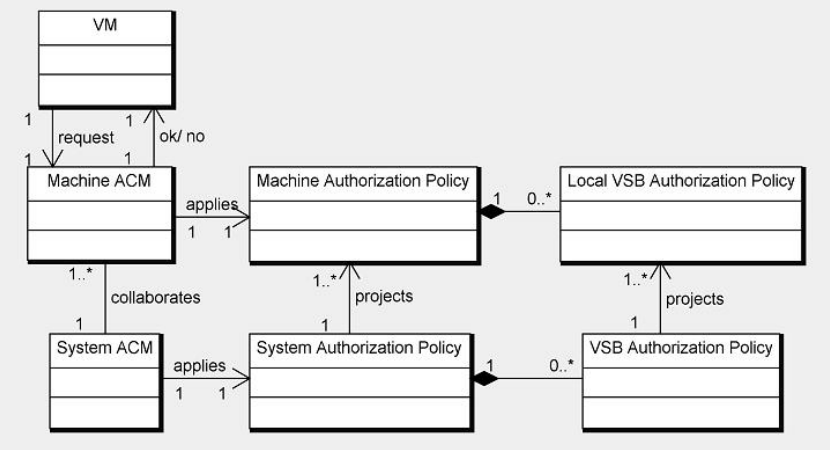

Figure 19. Cloud Security Model.

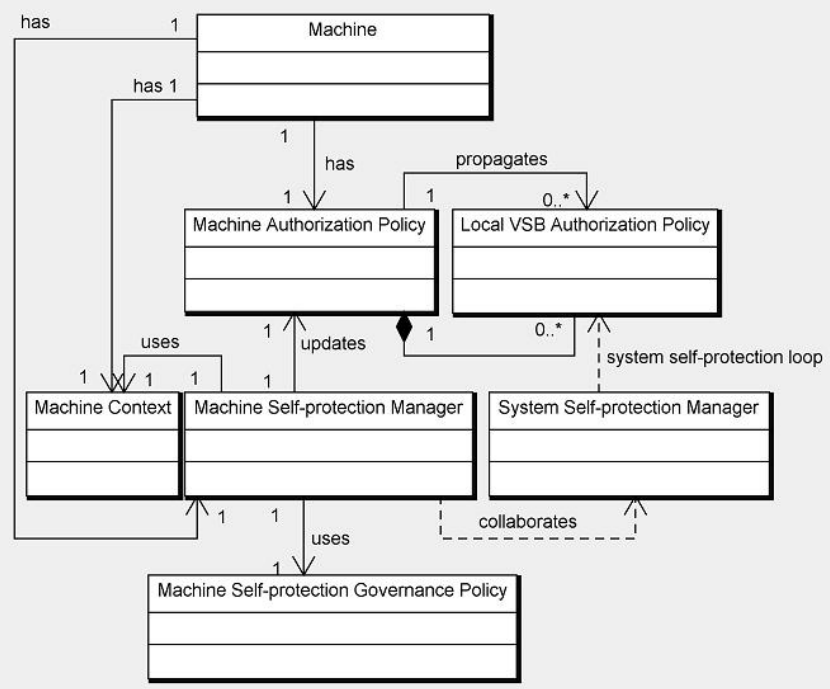

Figure 20. Machine Extended Model.

adaptation strategy defined in the *System Self-Protection Governance Policy*. This update is propagated towards the relevant VSB Authorization Policies. As in a pervasive case, on each machine, a self-configuration mechanism then translates each VSB Authorization Policy into a Local VSB Authorization Policy, finally updating the Machine Authorization Policy.

E. Authorization Architecture

An authorization architecture called SECloud was defined to implement the previous self-protection models. SECloud refines the ASPF authorization architecture. As shown in Figure 23, authorization validation is the result of a collaboration between System and Machine ACMs. SECloud consists of a number of server-side components installed in the cloud service provider network to control System, VSB, and local VSB functionalities, while the machine-side components essentially apply authorization policy adaptation decisions taken at the other end-point, and control access among local VMs. Such an architecture is currently under implementation.

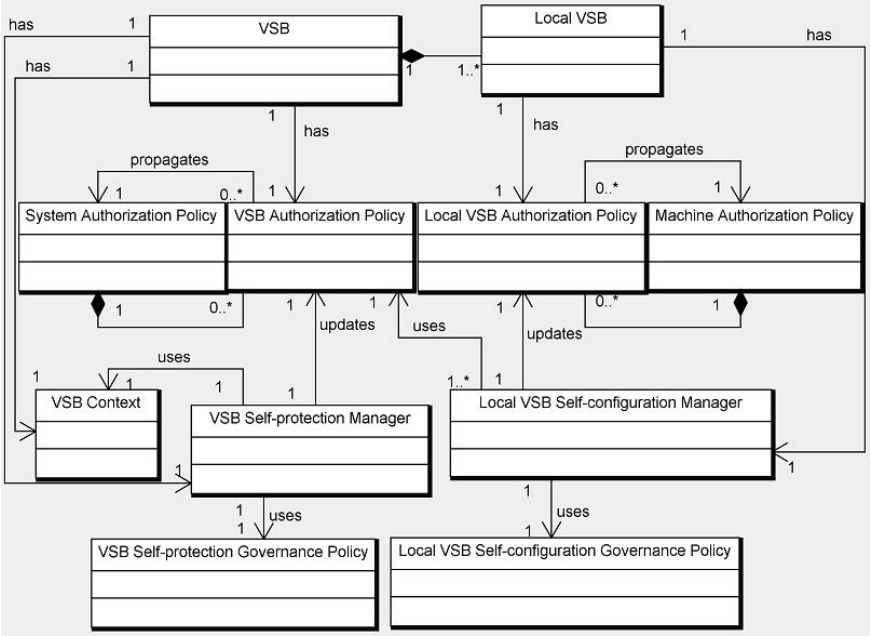

Figure 21. VSB Extended Model.

XI. Conclusion

This paper presented ASPF, a policy-based security management framework illustrating the design of an autonomic security manager to control OS-level authorization mechanisms in a pervasive device. ASPF implements several self-protection loops, authorization policies being adapted according to security context variation at both the network and device levels. Policies are described with an attribute-based extension of XACML to support policies specified in multiple authorization models. Performance, resilience, and security evaluations show that, together with VSK, ASPF provides strong and yet tuneable security while still achieving good performance, making it suitable for self-protection of pervasive systems. ASPF is also applicable to other types of large-scale systems such as cloud computing environments.

Current work focuses on the definition of the security adaptation strategy. We are currently investigating the approach where autonomic management strategies are specified using domain-specific languages (DSLs) [38]. Current ASPF adaptation strategies are purely action-based. However, higher-level strategies using objective or utility function policies are also desirable [39]. By enabling the specification of governance strategies with richer types of policies, the DSL approach should allow describing self-managed security at different levels of granularity which can be refined (e.g., with notions of policy continuum [40]), and thus evolve towards greater autonomic maturity in the corresponding systems.

Acknowledgments

This work has been funded by the ANR SelfXL project.

References

[1] R. He, M. Lacoste, and J. Leneutre, “A Policy Management Framework for Self-Protection of Pervasive Systems,” in *International Conference on Autonomic and Autonomous Systems (ICAS)*, 2010.

[2] D. Chess, C. Palmer, and S. White, “Security in an Autonomic Computing Environment,” *IBM Systems Journal*, vol. 42, no. 1, pp. 107–118, 2003.

[3] IBM, “An Architectural Blueprint for Autonomic Computing,” 2006, Autonomic Computing White Paper.

[4] J. Strassner, *Policy-Based Network Management: Solutions for the Next Generation*. Morgan Kaufman, 2003.

[5] R. He and M. Lacoste, “Applying Component-Based Design to Self-Protection of Ubiquitous Systems,” in *3rd ACM workshop on Software Engineering for Pervasive Services (SEPS)*, 2008.

[6] R. He, M. Lacoste, and J. Leneutre, “An OS Architecture for Device Self-Protection,” in *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2009.

[7] ——, “Virtual Security Kernel: A Component-Based OS Architecture for Self-Protection,” in *3rd IEEE International Symposium on Trust, Security and Privacy for Emerging Applications (TSP)*, 2010.

[8] L. Wang, D. Wijesekera, and S. Jajodia, “A Logic-Based Framework for Attribute-Based Access Control,” in *ACM Workshop on Formal Methods in Security Engineering*, 2004.

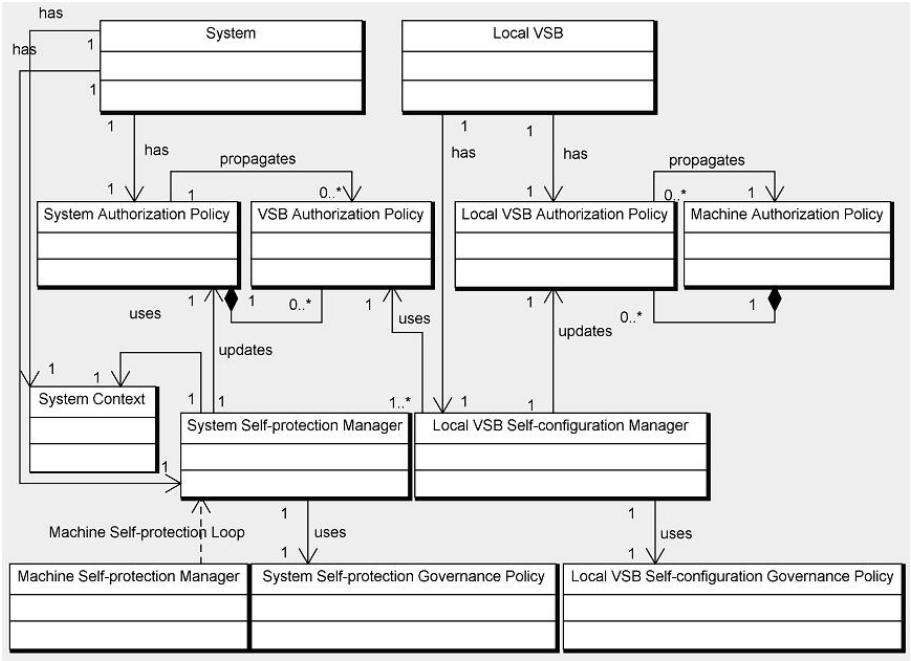

Figure 22. System Extended Model.

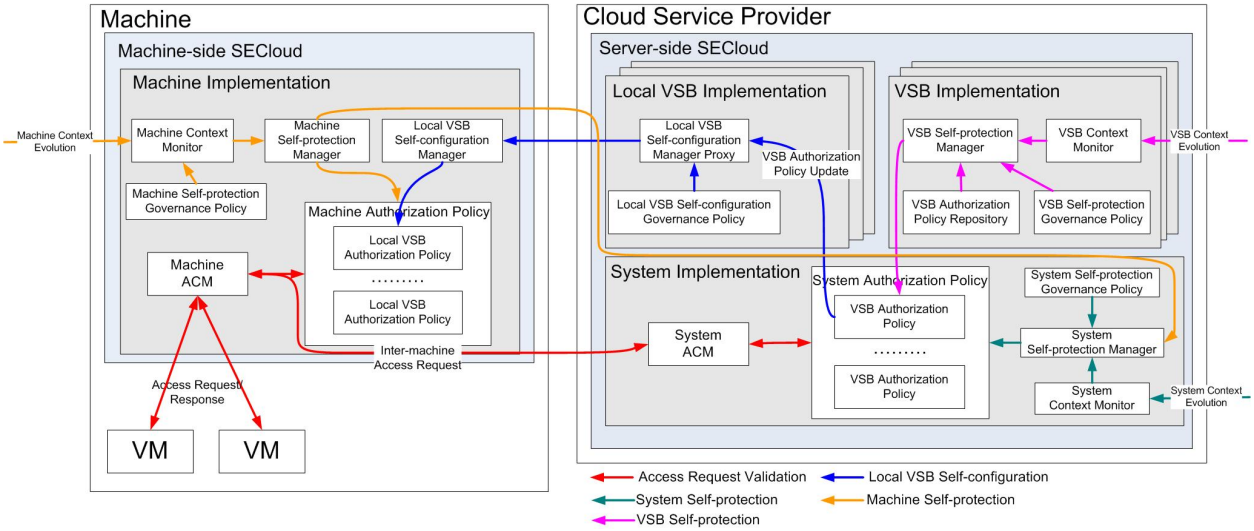

Figure 23. The SECloud Authorization Architecture.

[9] B. Claudel, N. De Palma, R. Lachaize, and D. Hagimont, "Self-Protection for Distributed Component-Based Applications," in *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, 2007.

[10] J. Agosta et al., "Towards Autonomic Enterprise Security: Self-Defending Platforms, Distributed Detection, and Adaptive Feedback," *Intel Technology Journal*, vol. 10, no. 4, 2006.

[11] D. Agrawal, K.-W. Lee, and J. Lobo, "Policy-Based Management of Networked Computing Systems," *IEEE Communications Magazine*, vol. 43, no. 10, pp. 69–75, 2005.

[12] J. Strassner, N. Agoulmine, and E. Lehtihet, "FOCALE: A Novel Autonomic Networking Architecture," in *Latin American Autonomic Computing Symposium (LAACS)*, 2006.

[13] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language," in *International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2001.

[14] K. Twidle, N. Dulay, E. Lupu, and M. Sloman, "Ponder2: A Policy System for Autonomous Pervasive Environments," in *International Conference on Autonomic and Autonomous Systems (ICAS)*, 2009.

[15] L. Broto, D. Hagimont, P. Stolf, N. De Palma, and S. Temate, "Autonomic Management Policy Specification in Tune," in *Symposium on Applied Computing (SAC)*, 2008.

[16] NIST, "A Survey of Access Control Models," in *NIST Privilege (Access) Management Workshop*, 2009.

[17] S. De Capitani di Vimercati, S. Foresti, P. Samarati, and S. Jajodia, "Access Control Policies and Languages," *International Journal of Computational Science and Engineering*, vol. 3, no. 2, pp. 94–102, 2007.

[18] P. Loscocco and S. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," in *USENIX Annual Technical Conference*, 2001.

[19] M. Lacoste, T. Jarboui, and R. He, "A Component-Based Policy-Neutral Architecture for Kernel-Level Access Control," *Annals of Telecommunications*, vol. 64, no. 1-2, pp. 121–146, 2009.

[20] B. Lang, I. Foster, F. Siebenlist, R. Ananthakrishnan, and T. Freeman, "A Flexible Attribute-Based Access Control Method for Grid Computing," *Journal of Grid Computing*, vol. 7, no. 2, pp. 169–180, 2009.

[21] E. Damiani, S. Di Vimercati, and P. Samarati, "New Paradigms for Access Control in Open Environments," in *International Symposium on Signal Processing and Information*, 2005.

[22] OASIS, "eXtensible Access Control Markup Language (XACML)," 2010, http://www.oasis-open.org/.

[23] J. Ames, S. R., M. Gasser, and R. R. Schell, "Security Kernel Design and Implementation: An Introduction," *Computer*, vol. 16, no. 7, pp. 14–22, 1983.

[24] M. Aljnidi and J. Leneutre, "ASRBAC: A Security Administration Model for Mobile Autonomic Networks (MAutoNets)," in *4th International Workshop on Data Privacy Management (DPM) and Second International Workshop on Autonomous Spontaneous Security (SETOP)*, 2009.

[25] L. Badger, D. Sterne, D. Sherman, K. Walker, and S. Haghighat, "Practical Domain and Type Enforcement for UNIX," in *IEEE Symposium on Security and Privacy*, 1995.

[26] D. Bell and L. La Padula, "Secure Computer System: Unified Exposition and Multics Interpretation," MITRE Corporation, Bedford, MA, Tech. Rep. MTR-2997, 1975.

[27] D. Ferraiolo, R.Sandhu, S.Gavrila, D. Kuhn, and R.Chandramouli, "Proposed NIST Standard for Role-Based Access Control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 4, no. 3, pp. 224–274, 2001.

[28] S. Marouf, M. Shehab, A. Squicciarini, and S. Sundareswaran, "Adaptive Reordering & Clustering Based Framework for Efficient XACML Policy Evaluation," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, 2010.

[29] M. Serrano, S. van der Meer, J. Strassner, S. Paoli, A. Kerr, and C. Storni, "Trust and Reputation Policy-Based Mechanisms for Self-Protection in Autonomic Communications," in *International Conference on Autonomic and Trusted Computing (ATC)*, 2009.

[30] M. Anne, R. He, T. Jarboui, M. Lacoste, O. Lobry, G. Lorant, M. Louvel, J. Navas, V. Olive, J. Polakovic, M. Poulhiès, J. Pulou, S. Seyvoz, J. Tous, and T. Watteyne, "Think: View-Based Support of Non-Functional Properties in Embedded Systems," in *IEEE International Conference on Embedded Software and Systems (ICESS)*, 2009.

[31] A. Brown and C. Redlin, "Measuring the Effectiveness of Self-Healing Autonomic Systems," in *International Conference on Autonomic Computing (ICAC)*, 2005.

[32] C. Rippert, "Protection in Flexible Operating System Architectures," *Operating Systems Review*, vol. 37, no. 4, pp. 8–18, 2003.

[33] Cloud Security Alliance, "Top Threats To Cloud Computing," 2010, http://www.cloudsecurityalliance.org/topthreats.html.

[34] S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan, "TVDc: Managing Security in the Trusted Virtual Datacenter," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 1, pp. 40–47, 2008.

[35] P. Ruth, J. Rhee, D. Xu, R. Kennell, and S. Goasguen, "Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure," in *IEEE International Conference on Autonomic Computing (ICAC)*, 2006.

[36] J. Rutkowska and R. Wojtczuk, "The Qubes OS Architecture," Invisible Things Lab, Tech. Rep., 2010.

[37] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds," in *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[38] R. He, M. Lacoste, J. Pulou, and J. Leneutre, "A DSL for Specifying Autonomic Security Management Strategies," in *Third IEEE International Workshop on Autonomous and Spontaneous Security (SETOP)*, 2010.

[39] J. Kephart and W. Walsh, "An Artificial Intelligence Perspective on Autonomic Computing Policies," in *IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2004.

[40] J. Strassner, J. de Souza, D. Raymer, S. Samudrala, S. Davy, and K. Barrett, "The Design of a New Policy Model to Support Ontology-Driven Reasoning for Autonomic Networking," in *Latin American Network Operations and Management Symposium (LANOMS)*, 2007.