Enhancing Cloud Security with Context-aware Usage Control Policies

Christian Jung, Andreas Eitel, Reinhard Schwarz

Fraunhofer Institute for Experimental
Software Engineering IESE
Kaiserslautern, Germany
{christian.jung, andreas.eitel, reinhard.schwarz}@iese.fraunhofer.de

Abstract: Cloud environments strongly rely on virtualization infrastructure that provides virtual resources by abstracting from the physical hardware. Thus, cloud providers can cost-efficiently share physical hardware among multiple tenants, and a single virtual resource may span multiple physical resources at different geo-locations. From a tenant's perspective, the uncertainty about location and context of virtual resources is a potential security threat. For instance, tenants may want to enforce geo-fencing to prevent their applications and data from migrating to undesirable jurisdictions, untrusted co-tenants, or dubious locations. They may also want to ensure that certain virtual resources share (or expressly do not share) a common physical resource, for example, to improve fault tolerance or performance. To tackle these problems, we suggest a flexible policy decision and enforcement framework for enabling usage control in cloud environments. In support of this framework, we collect additional information from the cloud environment to enforce context-aware and therefore more fine-grained usage control policies. Our solution offers flexible controls for secure and resilient cloud management. The paper presents our policy enforcement framework IND^{2}UCE and its extension to enable context-ware policy enforcement on an exemplary cloud infrastructure using VMware products.

1 Introduction

The Cloud offers organizations the chance to move their IT services to foreign operated cloud environments. Due to the economy of scale, IT outsourcing is often more cost-efficient and more flexible than in-house service provisioning, and professional cloud operation can even provide more security and resilience.

However, for being competitive, the cloud providers will run their systems close to the tolerable load limit. To reach their goal, they may share each of their physical resources among multiple customers, even among direct competitors or tenants of varying trustworthiness. In general, sensitive data are mixed with nonsensitive data, and critical services are running together with noncritical services. This unpredictability of runtime context raises security concerns among potential cloud users.

To explore the cloud users' security needs and reservations, we invited about 60 different companies to participate in a survey on cloud security and critical infrastructure IT [RS14]. Participants stated cost reduction, scalability, elasticity and especially availability, reliability, and resilience as the main advantages of cloud infrastructures for their organizations. However, the majority of the respondents claimed security and privacy as well as loss of control over key IT systems and the infrastructure itself as their main concerns regarding service deployment in cloud infrastructures. Our survey endorses results from previous surveys, for instance, performed by ENISA [CH09] or the TClouds project [BCH13].

Seemingly, the multi-tenancy problem, but also not knowing where data is really stored, prevents security and privacy aware enterprises from shifting their business software into cloud environments. To lower the barriers for cloud adoption, we need to put cloud users back into control by providing them security policies and corresponding mechanisms to express and enforce their protection needs for sensitive applications and data.

Contribution. We present an approach for security policy enforcement across multiple cloud abstraction layers. To this end, we adapted our policy enforcement framework IND^{2}UCE to the VMware virtualization environment, and we provided several enforcement components for VMware, such as policy enforcement and execution points as well as a policy information point. These components interact with the virtualization environment by retrieving events and by triggering detective, preventive, or corrective actions within the cloud environment when predefined conditions occur. Based on this framework, we can formulate usage control policies that are monitored and enforced in a VMware cloud.

2 Related Work

In [HKK12], Hamlen et al. propose a framework that contains different modules for policy decision and enforcement, such as a reasoning module. They propose to have control over data by replacing specific instruction (e.g., read method invocations) by surrounding the code with their own check (i.e., intercepting the method invocation and asking the reasoning module for a decision) before further processing.

The approach by Hamlen et al. is very similar to our IND^{2}UCE framework. However, we are focusing on controlling the management capabilities of the cloud environment with focus on the infrastructure level rather than the applications running in the cloud. Our future research will address security policies linking service level and infrastructure level. Furthermore, we provide a mechanism for context-aware enforcement of security policies.

Goyal et al. [GM09] present a “policy-based event-driven services-oriented architecture”. They state that policies are a set of rules that control behavior and usage of services. The services are managed by hierarchically distributed controllers named “mediators”.

Their architecture is focusing on the service layer, more specifically, on the enterprise service bus (ESB). The relation to the cloud is given, as such services may run within the cloud, but is not cloud-specific. Hence, our approach could be complementary to the approach presented by Goyal et al.

For KAoS [IHM13] was originally designed as an agent platform, but eventually became a set of platform-independent services for defining policies for the security, predictability, and controlability of both agents and traditional distributed systems. For policy enforcement, the KAoS Policy Services Architecture is provided. However, KAoS does not specifically address cloud environments.

VMware and other vendors developing virtualization software for cloud environments offer powerful management capabilities to their users (including security policies). However, they are focusing on their technologies and are often limited to specific system events. In our approach, policies can be defined for all available events in the system. Moreover, contextual information can be specified flexibly and can go beyond the information provided by the cloud infrastructure (e.g., geolocation or co-location of services).

3 Usage Control

Usage control extends access control by adding restrictions to future data usage once data access has been granted [PS04, PHB06, PS02]. Typical technologies for implementing enforcement mechanisms for usage control include ad-hoc solutions, runtime verification and complex event processing mechanisms.

3.1 The IND^{2}UCE Framework

The IND^{2}UCE framework ("Integrated Distributed Data Usage Control Enforcement"), illustrated in Figure 1, enforces usage control policies at different abstraction layers of a system. The framework is related to XACML [OAS10], a standard describing declarative access control policies and a processing model for it.

PDP	Policy Decision Point
PEP	Policy Enforcement Point
PXP	Policy Execution Point
PIP	Policy Information Point
PRP	Policy Retrieval Point
PMP	Policy Management Point
PAP	Policy Administration Point

Table 1: IND^{2}UCE Framework Component Abbreviations

The central component of the IND^{2}UCE framework is the PDP (cf. Table 1 for a list of abbreviations), a generic reasoning component. Each policy loaded into the PDP is represented by a state machine controlling policy evaluation. If external information or entities are referenced in the policy, the PDP will contact other framework components, such as PIPs for context information retrieval or PXPs for executing policy directives. Depending on the events that affect or are affected by the usage control policy, the PDP subscribes to one or more PEPs.

The role of the PEPs is to intercept the policy-relevant events in the system and to enforce the decisions of the PDP on them. A PEPs submits intercepted events to the PDP, where they are analyzed according to the specified policies. In contrast to the PDP, the PEP is a technology-dependent component, which is integrated into the software or system to interact with events.

Basically a distinction is made between preventive and detective enforcement. The latter does not prevent policy violations, but merely detects them to ensure accountability, whereas the former alternatives actually enforce policy compliance. For preventive enforcement, the PEPs are intercepting and blocking events; corrective actions or notifications for detective enforcement are performed by PXPs. In contrast to the PEP, the PXP is directly triggered from the PDP and not by a system event. Hence, a PXP cannot intercept or interact with events, it only executes additional actions, such as sending notifications, on behalf of the PDP. Enforcement is explained in more detail in Section 3.2.

In the XACML OASIS Standard [OAS13], the PIP is described as a "system entity that acts as a source of attribute values". Missing attribute values are added by a so-called context handler component, which can happen with or without request by the PDP.

Accordingly, our PIP component offers additional information for the decision making in the PDP and has to be integrated into the system being observed. PIP attributes may, for example, refer to contextual or information flow tracking data, such as geo-location or network connectivity. Contrary to the XACML framework, our PIP is directly connected to the PDP without having a context handler in between, so that the PDP can configure the PIP according to policy information needs. After configuration, the PIP collects and maintains all required information autonomously.

We differentiate between two components that handle the specification and management of usage control policies — the PAP and the PMP. The PAP provides a human-computer interface to specify usage control needs as policies. It transforms those demands into a machine-readable format.

The PMP is responsible for managing all IND^{2}UCE components. It is the first component to be available in the system. All other components have to register at the PMP and have to name their available connection interfaces. If a PDP wants to subscribe to specific events, the first step is to ask the PMP for an appropriate PEP for these events. Moreover, the PMP is also responsible for deployment and revocation of usage control policies.

The PRP provides a secure policy storage, protecting them against malicious modification. The PRP is only accessed by the PMP for policy management and by the PDP for policy retrieval.

3.2 IND^{2}UCE Enforcement Mechanisms

Policy enforcement is based on an event-condition-action (ECA) paradigm, which is reflected by the XML structure of the policy specifications. An event is an observable system

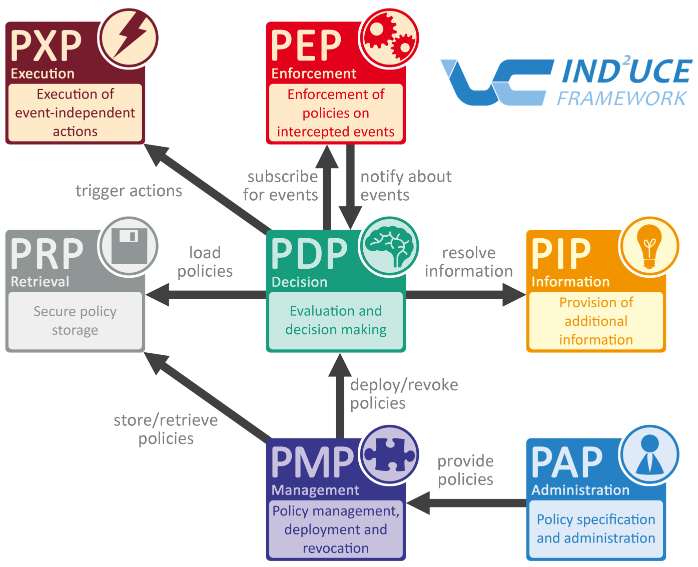

Figure 1: IND^{2}UCE Framework

activity happening at a given instant of time. Whenever an event is intercepted by a PEP, it is forwarded to the PDP. If the event matches the specified event description in a policy mechanism, the condition part of the mechanism is evaluated. If the condition evaluates to *true*, the instruction specified in the action part is executed.

In preventive policy mechanisms, the intercepted system event is caught before it is executed in the observed system. IND^{2}UCE offers four enforcement types for preventive mechanisms. To enforce, for example, the policy "no personal data is allowed to leave the system" on event "send message", there are the following options:

- *inhibition:* The event (i.e., sending the message) is rejected or simply dropped.
- *modification:* The event content is modified (e.g., person-related message fields are erased) before the event is permitted.
- *delay:* The event is delayed until a compensatory action has been executed (e.g., manual permission is requested before sending the message).
- *execution:* The event is permitted, but an additional action is performed (e.g., sending a notification to a privacy monitor).

Obviously, there is a fifth implicit enforcement type *allow*, which is the standard case. In the example, an *allow* decision from the PDP instructs the PEP to permit the message exchange.

The desired behavior is specified in the policy's authorization part. The behavior can also depend on the successful execution of an additional actions, so-called execution actions, such as writing a log entry or sending a notification email. These additional actions are executed by an appropriate PXP component.

Contrary to preventive policy mechanisms, detective policy mechanisms contain only execution actions but cannot prevent undesired system events. Such a behavior may be useful for compensating undesired but uncritical system actions or for a billing system just sending notifications that a particular action has been executed.

Trigger Events For preventive policy mechanisms, the specification of a trigger event is mandatory, whereas for detective ones it is optional. If the trigger event is not specified, the condition is evaluated at every timestep with a granularity specified by the policy author (e.g., once every 30 seconds).

Conditions The condition part of a policy mechanism can simply be *true*, or it can be expressed in past temporal logic. We are using the Obligation Specification Language (OSL) [HPB^{+}07] for expressing restrictions in the condition. OSL provides propositional, cardinal, and temporal operators. The condition part can also refer to additional attributes from PIPs, such as contextual or information flow tracking information. Regarding cloud environments, for instance, we can retrieve information on resource load of the virtual machines from a PIP that can be included in the decision making process.

Authorization and Execution Actions The action section of a policy mechanism may consist of several blocks: the authorization action and optionally several execution action blocks. For preventive mechanisms, an authorization block is mandatory in the policy, as the PEP needs a decision how to proceed with the intercepted event. If the condition evaluates to *true*, the specified authorization is returned to the PEP that intercepted the event to allow, modify, delay, or completely inhibit the execution of the desired action.

3.3 Context-specific Extensions to the IND^{2}UCE Framework

As mentioned before, the condition part can contain OSL operators as well as references to additional information sources (i.e., PIPs). Hence, the framework may be enriched with components for additional information retrieval. Such a component could be a *Context-PIP* that collects, aggregates, and evaluates arbitrary information.

To provide separation of concerns and to decouple the Context-PIP's tasks from one another, Context-PIPs are designed as layered components (see Figure 2). A fully developed Context-PIP implementation usually contains three layers: The *sensor layer* provides several sensor implementations that collect raw context data from the system under observation. The *data layer* persistently stores all collected sensor data in a suitable format, and it schedules the appropriate sensors as required by the configuration. The *evaluation layer* evaluates the aggregated sensor data. To this end, it contains multiple *evaluators*, which we will be described in Section 4. The evaluation layer consumes a context description as

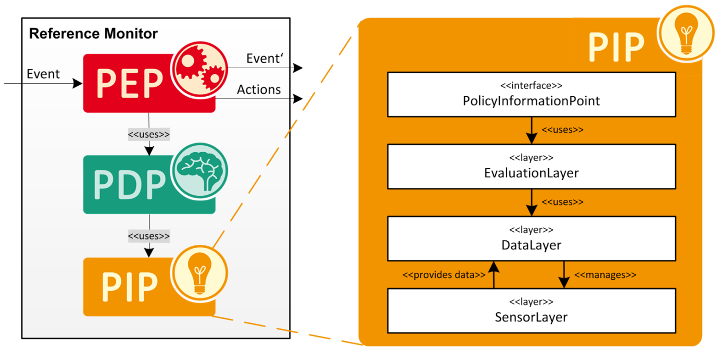

Figure 2: Context-PIP Layered Architecture

configuration, derives a context expression, and provides a unique identifier for referencing the context information. This reference is used by the PDP component for resolving context-dependent policy attributes.

Not all Context-PIP implementations have to provide all these three layers. In the simplest case, they have to provide only the evaluation layer, which is the entry point for configuring contexts and retrieving their evaluation result.

4 Policy Enforcement for VMware

We have chosen the VMware vSphere Hypervisor for implementing our policy enforcement. VMware is one of the major hypervisor vendors. However, VMware software is closed source, which limits our modification possibilities for hooking into the hypervisor. Our prototypical VMware cloud includes two physical hosts running VMware vSphere Hypervisor ESXi. A VMware vCenter Server manages the physical hosts as well as the virtual resources (e.g., network, CPU, memory). Virtualization operators can use either the client software VMware vSphere Client or a web interface for system management. The VMware vSphere Client uses a SOAP interface for interacting with the server. The web interface runs directly on the server using proprietary interfaces.

4.1 Hooking into VMware

We identified three viable solutions for connecting the IND^{2}UCE components with the VMware environment, as explained in more detail in the following.

4.1.1 Man in the Middle Approach

One approach for hooking into the VMware management communication is to insert a proxy component between vSphere Client and vCenter Server. This approach would require to implement our own SOAP interface that replaces the existing SOAP interface provided by VMware and to integrate our enforcement components directly in the code of the proxy implementation in order to control the event flow between client and server.

A clear advantage is that the proxy can intercept and then inhibit, modify, or delay any event before it will be performed on the VMware vCenter Server. This would enable us to implement both preventive and detective policy mechanisms, which offers comprehensive control over the virtualization infrastructure.

However, there are also some disadvantages. First, we would have to reimplement all SOAP interfaces, which is a tedious task. Second, the man in the middle approach affects the security of SSL/TLS in the system. The connection between client and server is protected using SSL/TLS, so we need to establish our own SSL/TLS connection between proxy and client and a second connection between proxy and server. Hence, we would need to provide our own X.509 certificates for the proxy, but deploying such digital certificates to our enforcement framework might be an undesirable modification for a cloud operator. Third, there might be events that are rejected by the system due to insufficient access privileges. Thus, we would have to check whether a specific user has the specific privileges by trying the event on the system, or we would have to implement the permission checks by our own (e.g., by querying the vCenter Server and also the Single-Sign-On server of VMware). To our knowledge, trying whether an event would succeed is not possible, and rebuilding the permission checking would again cause significant implementation overhead. Therefore, both solutions seem to be not very attractive. Moreover, a user can easily bypass the proxy by connecting directly to the VMware environment web interface.

4.1.2 vSphere Client Plugin

Another approach is to implement a vSphere Client plugin. This plugin could replace critical functions in the user interface with our IND^{2}UCE-enabled implementations. In this approach, the enforcement components run within the plugin.

Similar to a proxy, the plugin can intercept all events before they are performed on the server. Hence, a plugin approach enables both preventive and detective enforcement mechanisms.

However, this solution has similar drawbacks as the proxy approach. First, to intercept all possible user interactions, we need to rebuild the entire control center user interface. Second, user roles and privileges need to be handled within our enforcement framework. Third, from a security perspective, a client-based solution is undesirable as we lack control over the client configuration, so users can disable or circumvent it easily or simply connect to the original, unmodified web interface.

4.1.3 SOAP Interface vCenter Server

To avoid these drawbacks, a third option is to build a client software that connects to the SOAP interface of the vCenter Server and that is able to read system events and perform actions on the virtualization cluster. Such a software needs to retrieve events regularly from the management interface.

Such an approach is unaffected by VMware changes (except changes regarding the SOAP interface itself), and it does not depend on any other components or systems interacting with the vCenter Server. Hence, it will not disturb any other systems interacting with the cluster. Furthermore, instead of reimplementing all interfaces or rebuilding the control center user interface, we only have to implement suitable functions for intercepting and manipulating VMware events and for performing actions on the system. In particular, we do not have to take care of the user roles and privileges, as we are only observing events that already happened. The solution does not depend on the origin of the user or system interaction that it controls (thus, it cannot be circumvented), as long as events are produced.

However, the main disadvantage of this approach is the missing preventive enforcement. We can only monitor events that already happened and can perform some compensatory or corrective actions, but we cannot prevent the events from happening.

4.2 Implementation

Our current implementation is based on the latter approach. Although we loose preventive enforcement, it is not possible to circumvent or bypass our solution by simply using another interface or by accessing the ESX hosts directly, as all events will eventually be collected in the vCenter Server. Furthermore, IND^{2}UCE instrumentation can be transparently enabled and disabled without negatively affecting other components or systems interacting with the virtualization environment.

Our solution is based on Java and we implemented a PEP and a PXP for intercepting events and executing actions as well as a PIP component for retrieving additional information from the vCenter Server. All components run on a separate machine and connect to the vCenter Server via the SOAP interface. The components are using the VI Java API [Jin14].

The PEP implementation contains two main components: the EventCollector and the EventConverter. The *EventCollector* connects to the vCenter Server to retrieve events from the event history. This is done by polling for new events once a second. New events are forwarded to the EventConverter. We implemented several standard converters for various data types that could be part of a VMware event (e.g., BooleanConverter, IntegerConverter, CalendarConverter, EntityConverter). Thus, we are able to process arbitrary events as long as they introduce no new data fields in the VMware event.

Overall there are about 700 event types in VMware, and we tested 230 specific event types. We assume we can support nearly all 700 event types with the chosen approach. Events may affect various cloud aspects. For example, they can refer to virtual machines (e.g., migration, life-cycle, or power-cycle events) as well as the entire cluster (e.g., life-cycle, resources, or high availability services). There are six event categories: cluster, hosts, virtual machines, data stores, networking, and roles/permissions. All these event types and their specific parameters can be used to define the trigger event in our policy specification.

The PXP component can perform specific actions on the VMware cluster. For our prototype, we implemented several actions in the categories virtual machines, cluster, and roles/permissions. Hence, we are able to control the power-cycle (e.g., PowerOn/Off, Reset, Suspend, Standby, Shutdown, Reboot) and the life-cycle (e.g., Reconfig, Relocate, Migrate, Clone, CreateSnapshot) of virtual machines running in the virtualization cluster. These actions and their parameters can be specified in the action part of the mechanisms.

Our PIP component retrieves additional contextual information from the virtualization cluster during the decision making of the policy mechanism. *Evaluators* retrieve and aggregate information from the vCenter Server. Due to their generic implementation, they can be used to evaluate any kind of data from the vCenter Server. For example, they can evaluate configuration details of a virtual machine or a physical host as well as run-time information, such as resource load or storage usage. Evaluation results are associated with *expressions*. We differentiate between logic, arithmetic, and comparison expressions. Expressions and evaluators may be nested and combined to model “contexts of contexts”, and we can perform calculations and comparisons.

5 Policy-Enhanced Cloud Security

The following scenario may serve as an example for policy-enabled security enhancement in the cloud. Assume a tenant runs two critical infrastructure services on different virtual machines (VMs) of a cloud datacenter. Due to resilience requirements, the tenant requires that these critical services must not share the same physical resources. If the tenant or the cloud infrastructure operator starts migrating VMs to the same physical host, both critical services would run on the same hardware, which would violate the tenant’s security constraint. Using IND^{2}UCE, we can prevent co-location by specifying a corresponding anti-affinity policy. The policy specifies that migrating critical VMs to the same physical host is automatically resolved, for example, by migrating one critical service away.

Although VMware offers affinity and anti-affinity policies, their virtualization management allows policy violation. In contrast, the IND^{2}UCE framework enforces policy compliance. Moreover, our context-aware extension allows smarter corrective mechanisms taking into account the overall load situation and preventing life-locks (e.g., ping-pong migrations). In addition, IND^{2}UCE not only provides VMware-specific actions, such as creating recovery points (VM snapshots), but can also interact with external components, such as a hardware firewall.

6 Conclusion and Future Work

In response to the emerging need for improving security in cloud environments, we adapted our policy enforcement framework IND^{2}UCE to offer usage control capabilities to VMware. We discussed different options how a policy enforcement and execution point can interact with VMware. With the chosen strategy of our research prototype, we cannot prevent any system event, but we can only react after the event has already happened. Therefore, we will explore other possibilities to interact with the cloud environment to enable preventive policy mechanisms, at least for some critical events. On the other hand, our enforcement components are only interacting with the management interfaces of the cloud environment, which minimizes interference and reduces maintenance effort of the IND^{2}UCE instrumentation.

We have begun to implement policy enforcement framework and hooks to interact with a cloud storage environment, based on HBase and Hadoop. Further research will investigate how policy enforcement components, more specifically PEPs and PXPs, can also be implemented for higher layers of cloud environments (e.g., for the service layer) and how policy interaction between infrastructure and service layer can further improve security. For instance, an application running in the service layer could produce events that trigger the creation of snapshots of the virtual machines or create new instances in the virtualization layer. We have to investigate how a secure communication can happen between applications running in the service layer and our policy decision and enforcement framework placed in the management network of the cloud environment.

To explore context-aware policy enforcement, we developed a component for retrieving contextual information from the cloud environment. Based on context information such as geo-location or physical co-location, we were able to enforce affinity, anti-affinity, and geo-fencing policies that can be used to improve resilience and data privacy, but also performance of cloud applications. Further work will research what kind of information is most useful for improving cloud security and how such information can be modeled in a user-friendly manner.

Currently, our machine-readable security policies written in XML are usually created with a simple text editor. This process is error-prone and can only be used by experts that know the available events and system actions that can be used within the policy description. Hence, we started to research approaches on how to build user-friendly specification interfaces that allow even unskilled users to specify their security demands. The use of different usability pattern and how different user groups with varying skill levels and expertise perceive their support will be studied further.

Finally, we will investigate policy deployment and redeployment to the cloud in a secure manner to realize, for instance, distributed policy decision and enforcement. We also need to distinguish between different policy scopes: server-attached policies (e.g., exclusive resource usage), service-attached policies (e.g., service-logic and behavior-dependent policies), and request-attached policies (e.g., dynamic adaptation depending on policies attached to data in the request).

Acknowledgment The research presented in this paper has been funded by the European Commission in the context of the Research Framework Program Seven (FP7) project SECCRIT (Grant Agreement No. 312758).

References

[BCH13] Roland Burger, Christian Cachin, and Elmar Husmann. Cloud, Trust, Privacy – Trustworthy cloud computing whitepaper. Whitepaper, TClouds Project, 2013.

[CH09] Daniele Catteddu and Giles Hogben. An SME perspective on Cloud Computing. Survey, ENISA, Nov 2009.

[GM09] P. Goyal and Rao Mikkilineni. Policy-Based Event-Driven Services-Oriented Architecture for Cloud Services Operation & Management. In Cloud Computing, 2009. CLOUD '09. IEEE International Conference on, pages 135–138, Sept 2009.

[HKK12] Kevin W. Hamlen, Lalana Kagal, and Murat Kantarcioglu. Policy Enforcement Framework for Cloud Data Management. IEEE Data Eng. Bull., 35(4):39–45, 2012.

[HPB$^+$07] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter. A Policy Language for Distributed Usage Control. In Joachim Biskup and Javier Lpez, editors, Computer Security – ESORICS 2007, volume 4734 of Lecture Notes in Computer Science, pages 531–546. Springer Berlin Heidelberg, 2007.

[IHM13] KAoS Policy Services Framework: User Guide. Technical report, Institute for Human & Machine Cognition (ihmc), Jan 2013.

[Jin14] Steve Jin. VMware VI (vSphere) Java API. `http://vijava.sourceforge.net/`, April 2014.

[OAS10] OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0 - Committee Specification 01. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf`, August 2010.

[OAS13] OASIS. eXtensible Access Control Markup Language (XACML) Version 3.0 - OASIS Standard. `http://http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf`, January 2013.

[PHB06] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. Commun. ACM, 49(9):39–44, 2006.

[PS02] Jaehong Park and Ravi Sandhu. Towards Usage Control Models: Beyond Traditional Access Control. In Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, SACMAT '02, pages 57–64, New York, NY, USA, 2002. ACM.

[PS04] J. Park and R. Sandhu. The UCON ABC usage control model. ACM Trans. Inf. Syst. Secur., 7(1):128–174, 2004.

[RS14] Manuel Rudolph and Reinhard Schwarz. SECCRIT Online Survey on Cloud Security for Critical Infrastructure IT. Survey, Fraunhofer IESE, March 2014.