Policy-based Interaction Model for Detection and Prediction of Cloud Security Breaches

Sara Farahmandian
Faculty of Engineering and IT, University of Technology Sydney, Broadway, Ultimo, Sydney, Australia

Doan B. Hoang
Faculty of Engineering and IT, University of Technology Sydney, Broadway, Ultimo, Sydney, Australia

Abstract: The ever-increasing number and gravity of cyberattacks against the cloud's assets, together with the introduction of new technologies, have brought about many severe cloud security issues. The main challenge is finding effective mechanisms for constructing dynamic isolation boundaries for securing cloud assets at different cloud infrastructure levels. Our security architecture tackles these issues by introducing a policy-driven interaction model. The model is governed by cloud system security policies and constrained by cloud interacting entities' locations and levels. Security policies are used to construct security boundaries between cloud objects at their interaction level. The novel interaction model relies on its unique parameters to develop an agile detection and prediction mechanism of security threats against cloud resources. The proposed policy-based interaction model and its interaction security algorithms are developed to protect cloud resources. The model deals with external and internal interactions among entities representing diverse participating elements of different complexity levels in a cloud environment. We build a security controller and simulate various scenarios for testing the proposed interaction model and security algorithms.

Keywords: Cloud infrastructure, security policies, security isolation, interaction, security boundaries.

Introduction

Security issues in a virtual cloud environment are more complex and challenging than in traditional infrastructures since resources are both virtualised and shared among numerous users. As a result, virtual boundaries among components or participants are not well defined and often undefined, and hence they not visible or controllable by the providers. In a multi-tenant cloud architecture, isolations are a crucial concept for both security and infrastructure management. They should be considered at functional entity levels and appropriate abstraction levels of the infrastructure. Physical isolation is relatively simple in traditional environments, as the boundaries between physical elements are well-defined and visible. The situation is not clear-cut in virtual environments unless one can keep track of all perimeters of all virtual objects created. Defining object boundaries is extremely difficult because virtual objects are dynamic in both characteristics and functionality. The task is resource-expensive due to the sheer number of virtual objects and the complexity of their dynamics. Building security boundaries is critical not only for recognising security violations but also in creating security solutions.

Practically, a security breach is defined in terms of the policies that define the interactions related to the breach. An event is considered a security breach either when it violates a defined security policy or violates the Confidentiality, Integrity, and Availability of security principles that could have been avoided if a relevant security policy had been in place. According to Kosiur (2001), a policy (or policy rule) is a simple declarative statement associating a policy object with a value and a policy rule. In general, a policy is not easy to work with as, at one extreme, a policy applies to the overall behaviour of a complex organisation (or entity) and, at the other extreme, it applies to a particular action on an element of the organisation, or a specific firewall rule on a network connection. To work with policy, one needs to clearly define the appropriate context in both scope and level; otherwise, it is not very useful or realisable.

In this paper, the policy context is on the interaction between entities with a defined set of interaction parameters. Security breaches primarily result from violations of the rule of interaction (or policy that governs the interaction) between objects when they interact. Unless one has a formal interaction model between objects, it is difficult to detect, predict, or prevent security incidents. The policy-based interaction model defines a security breach as when a security policy is violated over an interaction parameter. It has been recognised that security policies play a crucial role in all secured systems because they define what constitutes a security breach. In other words, security policies define the rules for secure interaction between objects of an environment. Security policies define the desired behaviour of the heterogenous application, systems, networks, and any type of object within the system.

Policies are complex in terms of definition and implementation in a distributed cloud infrastructure where resources are shared and dynamically changed. Different policies are often constructed for different architectural levels of a system, together with enforcement mechanisms. The ever-increasing number of virtual functions and the dynamic nature of cloud resources introduce more complexity in defining and enforcing security policies. Enforcing security policies at the interaction level enables system agility in detecting security breaches in cloud infrastructure. The policy-based interaction model is appropriate to impose and enforce dynamic, secure interactions among entities.

In this paper, we construct security boundaries dynamically at the interaction level between entities using the security policies or rules over a proposed interaction model parameter and the constraints imposed on the interacting entities. The construction of security boundaries in a cloud system is related to the characteristics of the interacting entities in the environment and the policies and constraints that govern their interaction. Our design focuses on building a robust, dynamic, and automated security boundary to protect cloud assets relying on a solid and innovative interaction model and security policy expressions that govern the interactions. A security boundary is thus a function or an expression that defines valid interactions among cloud entities.

The paper focuses on constructing security boundaries according to the interaction model and its parameters, object constraints, and dynamic security rules related to interaction parameters. We introduce a policy-driven interaction model that governs the relationship among entities in the cloud environment and develops algorithms to detect and predict security breaches. The interaction model is defined by parameters that control activities among components or entities in a cloud system. The model provides a framework for incorporating system security policies and entity constraints in constructing interaction boundaries and defining a security dictionary of expected/unexpected behaviour of cloud entities while accessing resources in the cloud environment. The main contributions of this research are:

- We propose a novel policy-driven interaction model that governs the interactions among entities in a cloud environment. According to our best knowledge, this is the first approach to use interaction parameters for building dynamic and automated security boundaries.
- We deploy an automatic detection and prediction algorithm called interaction security violation detection and prediction (ISVDP) to identify security breaches related to interaction parameters. The algorithm also maps out possible future attacks based on expected violations of the currently defined interaction parameters.
- We evaluate the proposed model and algorithms by implementing and simulating various interaction scenarios among cloud entities.

The paper is organised as follows. We first describe related work. We then briefly introduce the cloud object model and components used for the interaction model. After that, we describe the proposed general interaction model and its parameters. Building on the general interaction model, we then describe the security policy-based interaction model. We can then introduce our ISVDP algorithms, which we evaluate by simulating various interaction scenarios. Finally, we provide a conclusion.

Related Work

This section describes related work on cloud security isolation methods and security policy enforcement methods.

Cloud resource isolation mechanisms

Mavridis & Karatza (2019) proposed a multi-tenant isolation solution using VMs as the boundary of security whereby applications run within containers on top of these virtual machines. To improve the security of running applications as containers in the cloud, running one container per VM was suggested. However, the drawback of such a system is its performance.

SilverLine (Mundada, Ramachandran, & Feamster, 2011) was proposed for enhancing data and network isolation for cloud tenant services. The model concentrated on providing isolation via OS-level and virtual instances. The method only focused on providing data and network isolation at the tenant level.

A mechanism known as Secure Logical Isolation for Multi-tenancy (SLIM) was introduced by Factor *et al.* (2013) as an end-to-end approach to providing isolation among tenant resources. The model only considered tenant-level isolation. Pfeiffer *et al.* (2019) proposed a method for solid tenant separation in cloud platforms by isolating components at the network level. It focused mainly on tenant separation via physical and cryptographic separation for large infrastructures.

Hoang & Farahmandian (2017) provided a classification of isolation techniques within a cloud infrastructure and proposed isolation solutions using existing technologies. Del Piccolo *et al.* (2016) conducted a research survey on network isolation solutions for multi-tenant data centres for isolating cloud services. It emphasised the main challenges related to isolation in a multi-tenant environment.

Chen *et al.* (2016) proposed a Highly Scalable Isolation Architecture for Virtualized Layer-2 Data Centre Networks. It used Software-Defined Networking (SDN) technology to provide isolation for a layer-2 data centre at the network level. The model only provided isolation at the network level. BlueShield was another method to provide isolation and security in a multi-tenant cloud infrastructure focusing only on network security (Barjatiya & Saripalli, 2012).

Security policy enforcement mechanisms

Karmakar *et al.* (2016) proposed a policy-based security architecture to secure SDN domains. The paper defined different modules within a proposed application to determine security policies related to packets. Wang *et al.* (2015) introduced a policy space analysis and focused on addressing network security policy enforcement issues on middle boxes.

Varadharajan *et al.* (2018) proposed a policy-based security architecture to secure inter- and intra-domain communication using software-defined networks between different hosts across multiple domains. Basile *et al.* (2019) introduced an approach for automatic enforcement of security policies in network function virtualisation according to dynamic network changes. It deployed virtual security functions for security policy reinforcement and introduced a security awareness manager in the orchestrator.

A cyberspace-oriented access control model (CoAC) was proposed to provide access to sensitive data (Li *et al.*, 2018). The method considered operations as a combination of many atomic processes and defined a CoAC policy that permits access only if a particular operation's security risk is below a defined threshold.

Access control policy enforcement methods

Cai *et al.* (2018) reviewed existing access models and policies among different application scenarios focusing on cloud and user requirements. Damiani *et al.* (2007) proposed a geographical Role-Based Access Control. It relied on role-based mechanisms and defined constraints according to user location and position.

Rajkumar & Sandhu (2016) proposed POSTER for enhancing administrative role-based access control. It has integrated obligations via an administrative model by defining three main obligatory actions. However, the model only focused on administrative actions within the system. In 2016, Tarkhanov (2016) addressed the access control difficulties related to objects and linked object states.

The majority of existing research efforts on cloud security focus on users/intruders, traffic monitoring, and computing entities. Detecting and predicting security breaches based on object interaction and system policy have not received much attention. A thorough search of relevant literature yielded the conclusion that this research is the first to define and use interaction parameters to construct dynamic security boundaries and detect and predict security violations.

Cloud Object Model used for Interaction Model

This section describes the cloud object model and the required components to be used in our interaction model. Traditional physical security mechanisms are ineffective in dealing with threats and activities from virtual systems and virtual resource components (Jararweh *et al.*, 2016), as virtual boundaries between virtual components are not often well defined. The infrastructure desires a logically centralised security controller with visibility of security boundaries within different layers. For this purpose, a security model was proposed as a dynamic, intelligent, and automated security service/model to tackle the mentioned challenges in a multi-tenant cloud infrastructure (Farahmandian & Hoang, 2017). It provides a security model, and a security service called the Software-Defined Security Service (SDS_2) that applies to the object-oriented entities of a cloud environment, the interaction among them, and security policies that govern the interaction. The SDS_2 provides a security architecture to protect cloud assets with policies mapped to the cloud, tenant, and resource security policy levels.

The main idea of our centralized model centres around the interaction between constrained entities and is governed by system security policies for the detection and prediction of security breaches. An interaction can be defined as a *relation* between the objects during a specific time slot. To define the security boundaries in terms of interaction, the system requires a sustainable object model. In our security architecture, we define an object as *a component or a sub-component, both virtual and physical, that participates in a cloud environment and can access/be accessed by other objects according to their properties, security constraints, and system policies*. An object has a number of attributes, some are common among all objects (generic), and some represent specific constraints and characteristics of the object (specific). Each attribute defines some properties, and hence together they constitute a boundary for an object relative to other objects in its environment. An object can be simple or complex. A complex object includes nested attributes and may consist of a set of sub-objects. An object can be internal or external to a cloud, depending on its role/interaction.

It should be noted that the policy level is related to the role of an object, and location is associated with the logical or physical location of an object. The security model defines three main objects associated with the cloud, tenant, and resource security domains. Corresponding objects are Cloud Object, Tenant Object, and Resource Objects, which include Compute Object, Storage Object, Network Object, App Object, and User Object. The *cloud domain,* where cloud objects reside, classifies all the data, resources, and interactions at the cloud level while ignoring information related to lower domains like Tenant and Resource. At this level, the main parameters include cloud security policies (SPs), which govern interaction policies among objects at the cloud level; and data and resources policies, which concentrate only on cloud resource level (tenant, cloud-compute, -net, -storage resources). The *tenant domain* only reflects attributes and parameters related to the tenant objects in the cloud domain. The focus is only on the tenants' structure and their parameters and resources. The *resource domain* concentrates on the base underlying physical/virtual resources within the cloud system, as distinct from resources at the cloud and the tenant domains. They provide detailed information related to each resource object. Resource objects are defined similarly to cloud objects but for objects in the resource domain.

A *role (Rl)* assigns some responsibility to an object and the necessary authorities or privileges to discharge its duty. The role is often not static and may change as circumstances demand. A role may be simple or complex, assigned to an individual or a group of objects. A role is often associated with different layers of the architecture of a cloud system. It should be noted that 'role' is best defined using formal logic that entails complex rules to deal with dynamicity and multiple inheritances. In this paper, we avoid the complexity by simply equating a role with a hierarchical level in our defined cloud security architecture. Its attributes are defined explicitly when the role is assigned to a cloud object.

We define an *entity (E)* as an integrated object consisting of the object's role and object structure. The entity is a key concept in our structure to detect and predict security breaches in cloud infrastructure. The role assigned to the object will be considered based on object level and position within the system extracted from defined object parameters. An object may be assigned a role or group of roles activated at a different system level. Objects may assume more than one role with different levels of authority in different domains. We use E as the main component within the interaction model.

Interaction Model

In this section, we introduce our interaction model and its parameters. Security will always be a concern when entities start interacting with each other and with the infrastructure. In general, an interaction is *an act of performing an action by an object on another. A natural disaster can also be considered a special interaction between an external object on a set of objects*. An action always entails some effects or consequences. Potential security violations may occur when an interaction occurs against policies governing the relationship between two or more parties.

Consequently, interactions play a central role in security incidents in a system. The main focus of the Software-Defined Security Service (SDS$_2$) is on the protection of a cloud system by anticipating possible security breaches and preventing them from happening. The SDS$_2$ proposes a novel interaction model that defines exceptional interaction parameters to detect and predict security violations. The following sub-sections describe a detailed structure of each parameter. The scheme centres around a new model of interaction, entities connected to a cloud system, and security policies governing the system.

Table 1. Summary of Notations

Notation	Description
$\boldsymbol{E_i}$	Denotes entity i
$\boldsymbol{M}$	Denotes the interaction mode
$\boldsymbol{m_i}$	A set of mode relation values of the interaction mode
$\boldsymbol{d_n}$	A set of action direction values of the interaction mode
R	Denotes the positional interaction relationship
Rl	Refers to a set of roles of an object
$\boldsymbol{r_n}$	A set of positional relation values of R
T	Refers to interaction time consisting of $t_s(start\ time)$, $t_e(end\ time)$, $t_d(duration\ time)$, and α $(interaction\ state)$
$\boldsymbol{A}$	Represents a set of all possible actions
P	Refers to system security policy
C	Refers to entities' constraints
$\mathbf{S_k}$	Refers to set of security policies on k
$\boldsymbol{L^k}$	Refers to location-based security policy of interaction k
$\boldsymbol{t^k}$	Refers to validate time for an interaction k
M	Set of permissible parameter values for interaction $I^k_{p,c}$
V	Set of non-permissible parameter values for $I^k_{p,c}$
I	Refers to an interaction

In its general sense, an interaction takes place between simple or complex entities in a defined environment such as a cloud system. Figure 1 shows simple and complex interactions between simple and complex entities. We propose an interaction model for characterising a relationship between objects. The interaction model describes how objects interact with one another; it characterises the *modes* of interaction, the *roles* of interacting entities, the *actions* one can perform against others, and the *time* of the interaction. To capture the essentials of an interaction, we define an *object model of interaction* with four parameters or variables: mode (M), positional relationship (R), action (A), and time (t). Each parameter may take on a range of values. The range is determined or constrained by a) the interaction environment such as organisational policies; b) participating entities of the interaction in terms of their nature, properties, capabilities, and constraints; c) roles of the participating entities such as their relative positional relationship; and d) the time of the interaction. These parameters will be defined later in this section.

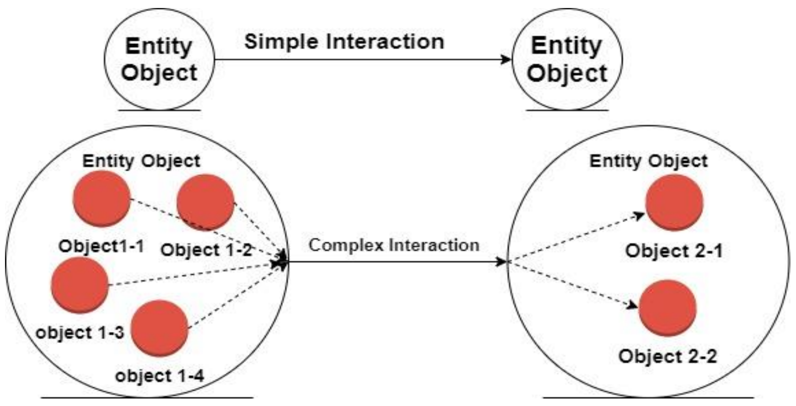


Figure 1. Interaction Types

With these descriptions of the interaction object, we will be interested in the following operations:

- We want to initialise an interaction, allowing default values for all parameters without any constraints.
- We want to know what actions are possible and what are not, due to the constrained nature of the entities involved in the interaction.
- We want to know if the interaction is permissible under a set of governing system policies.

Specifically, we can define several base operations on an interaction between entities:

- Initialise (I): Initialise I with default parameters M, R, A and t
- Mode ($I^k_{E_iE_j}$): Return all the possible modes between E_i and E_j for interaction k
- Relate ($I^k_{E_iE_j}$): Return all possible positional relations between E_i and E_j for interaction k
- Action ($I^k_{E_iE_j}$): Return all possible actions between E_i and E_j for interaction k
- State ($I^k_{E_iE_j}$, S_k): Return all allowed M, R, A and t once the constraints for participating entities and security policies (S_k) have been applied to the interaction.

Additional operations involving entities, their constraints, and system policies relevant to security violation and detection will be described in Security Policy-based Interaction Model below.

Interaction mode

Interaction mode (M) determines both the mode relationship (m) between objects, such as one to one, one to many, etc., and the action direction (d) of the interaction from one object to another, such as one way, both ways, etc. M consists of two parts: the first part refers to the mode relation (m_i), and the second refers to the action direction (d_n). So, M is defined as a set of pairs consisting of m_i and d_n: $M = m_i \times d_n$ where m_i refers to a set of possible relations between entities. $m_i \in \{m_1, m_2, m_3, m_4, m_5, m_6\}$. The values of m signify the following: $m_1 := 1{:}1 (one{:}one)$; $m_2 := 1{:}m\ (one{:}many)$, $m_3 := m{:}1 (many{:}1), m_4 := m{:}m\ (many{:}many),\ m_5 := 1{:}0\ (isolated), m_6 := 0{:}0\ (no\ relation)$.

An interaction's action direction may take on one of three possible values, d_1 d_2, and d_3. Specifically, $d_1 = 1 :=\rightarrow (left\ to\ right), d_2 = 2 :=\leftarrow (right\ to\ left), d_3 = 3 :=\leftrightarrow (two\ way)$.

Interaction positional relationship

An object within a system or an organisation exists at a position either defined by its role within the organisation or the layer or the domain within the system architecture. In an interaction, the role of an entity and its standing relative to the role of the other entities is important, as this may dictate whether the interaction is legitimate. For this reason, we consider a positional interaction relationship (R) as the relative positional relationship between the entities of an interaction. The positional relationship determines the validity of an interaction action through defined rules, roles, layers, and policies associated with an entity's interaction.

For example, a security policy may specify that only objects in the same domain or at the same level may interact. Interaction level is entangled with the role-based level assigned to each domain in the design. Each level entails classified security policies associated with object roles that determine a set of authorised actions. As "roles" may be of a complex nature with inheritance and may change during an entity's lifetime, we simply restrict and associate roles with three positional interaction relationships in any interaction between objects to three different security isolation layers of the security architecture: Cloud, Tenant, and Resource.

In this design, *R* denotes the positional interaction relationship according to entities' relation during an active interaction. R ∈ {r_1, r_2, r_3}, where r_1 is mapped to **down,** representing the interaction between objects from a high layer to a lower layer; r_2 is mapped to **up**, representing interaction from a lower layer to a higher layer; r_3 is mapped to **equal,** representing the interaction between objects in the same layer. Knowledge about positional relationships among objects helps to define the nature of an interaction and the security policy decision.

Interaction time

Interaction time (t) refers to the valid time for an interaction to take place in the system. The interaction time can be specified either by its start time and its end time (t_s, t_e) or by start time and duration (t_s, t_d). There may be cases where the start time of an interaction is known but its end time may be indeterminate depending on some environmental conditions. For such cases, t_d is replaced by the *interaction state* (α) to indicate if the interaction is still ongoing (**on**) or has stopped (**off**). Interaction time can thus be specified by $t = \{(t_s, t_e)\ or\ (t_s, t_d)\ or\ (t_s, \alpha)\}$

Interaction action

An interaction is meaningful if it conveys a particular set of actions. A security breach occurs when objects perform an action that violates their permissible interactions. An action is defined as a possible set of actions over an interaction between system objects by virtue of their specific relationship connected to the system. An action is a set of possible activities that an event may trigger; however, the set of possible actions is often limited by the nature and constraints on object/entities involved and security policy rules governing them and their interactions. Let $\boldsymbol{A}$ represent a set of possible actions that are chosen based on the types of objects found in a cloud environment. For our cloud security model, we studied cloud objects and established the set $\boldsymbol{A}$ of actions as follows: 'read', 'write', 'modify', 'create', 'delete', 'execute', 'migrate', 'suspend', 'enable', 'disable', 'reset', 'lock', 'activation', 'unlock', 'clear'. Clearly, an object cannot perform all actions, as they are subject to system policies and object constraints. Table 2 describes the meaning of actions in $\boldsymbol{A}$.

Table 2. Action Description

Action	Description
Read (Re)	Permission to read the data on another Object
Write (W)	Permission to write data onto another Object
Modify (Md)	Permission to change (Write and Delete) existing data on another Object
Create (Cr)	The right to create instances of another Object
Delete (D)	The right to remove instances of another Object
Execute (Ex)	The rights to run an instance of another Object
Migrate (Mi)	The rights to re-map an instance of another Object
Suspend (Sp)	The rights to pause an instance of another Object
Enable (En)	The rights to run or power up another Object
Disable (Di)	The rights to power down another Object
Reset (Rt)	The rights to delete metadata and reboot instances of another Object
Lock (Lk)	The rights to deny user access to another Object
Unlock (U)	The rights to permit user access to another Object
Activate (Av)	The rights to make another Object available to a User
Clear (Cl)	The rights to remove user data from another Object

Security Policy-Based Interaction Model

This section describes our policy-based interaction model and how we use security policies at the interaction level to detect and predict security breaches. In our design, security policies are mapped to rules that determine the interaction parameters between entities. The proposed policy-based interaction model constructs dynamic security boundaries formed by legitimate interaction parameters according to security rules extracted from the governing security policies. Our model focuses on security policies at the interaction level between entities through a set of interaction parameters. The complex structure of cloud infrastructure and the shared and dynamic nature of their resources demand robust security policy enforcement. This requires a clear definition of a boundary between violated and non-violated policies. Applying security policies at the interaction level allows a system to make visible previously undefined virtual boundaries between engaged entities through their interaction parameters. In the following, we describe our policy-based interaction model and its required components.

A policy can be defined as "an aggregation of policy rules", where policy rules are used to construct sets of conditions consistent with the set permissible actions (Stone, Lundy & Xie, 2001). Policy rules are often derived from human language statements extracted from service level agreements (SLAs) between users and service providers. NIST (2015) defines security policies as *"Aggregate of directives, regulations, rules, and practices that prescribe how an organisation manages, protects, and distributes information"*. In our design, security policies address rules and conditions that establish valid interactions between entities in a cloud environment. In the SDS$_2$ architecture, we define a security policy (SP) as *a directive that governs the interaction among simple/complex entities through specific constraints applied to the entities, their location, and their interaction parameters*. Security constraints extracted from security policies determine the validity of a set of actions taking place during an interaction. Figure 2 illustrates the relationship among these components.

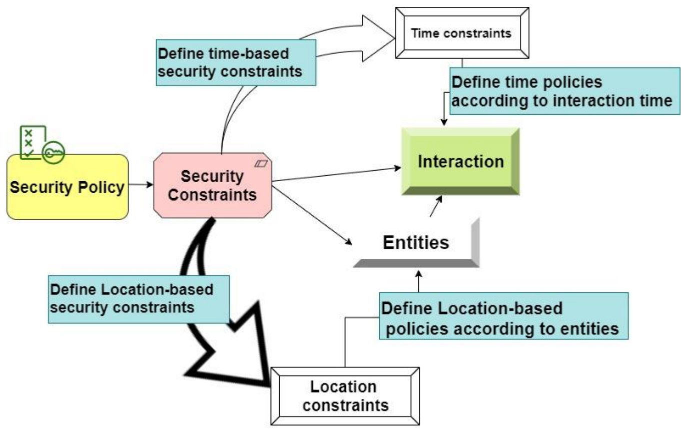

Time constraints
Define time-based security constraints
Define time policies according to interaction time
Interaction
Security Policy
Security Constraints
Entities
Define Location-based security constraints
Define Location-based policies according to entities
Location constraints

Figure 2. Security policy and its components

As discussed, system security policies, when applied to an interaction between the initiator (E_i) and the target entity (E_j), determine sets of parameters (described in the cloud object model) that are secure (valid or permissible) for the interaction. We define a *Security Policy-Based Interaction* model, as shown in Figure 3. Security policies govern the validity of the parameters of the interaction. Together with the system security policies (P), security constraints (C) on entities further limit the interaction in time, isolation level, and location, as defined by legitimate interaction parameters. The SDS$_2$ architecture logically divides cloud infrastructure into three main security isolation levels (SILs) or boundaries for the Cloud, Tenant, and Resource cloud domains. Recently, Yin *et al.* (2018)

introduced a security service framework with three security layers according to security domain divisions; however, the system only focused on divisions related to tenant resources and VMs in building isolation layers. We map security domains into security isolation levels that isolate each domain's entities according to their security policy levels and the entities' locations. Figure 4 shows these isolation levels.

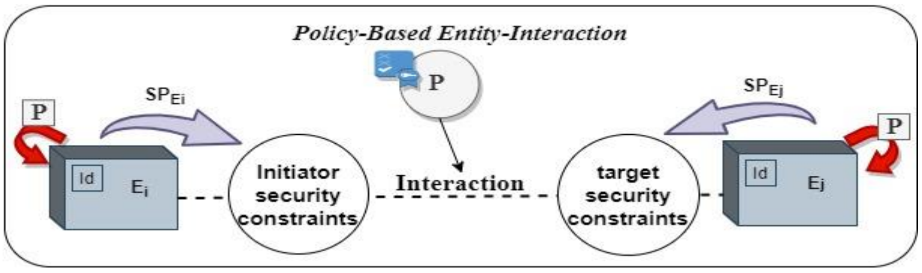

Policy-Based Entity-Interaction
P
SP_{Ei}
SP_{Ej}
P
P
Id
E_i
Initiator security constraints
Interaction
target security constraints
Id
Ej

Figure 3. Security Policy-Based Interaction Model

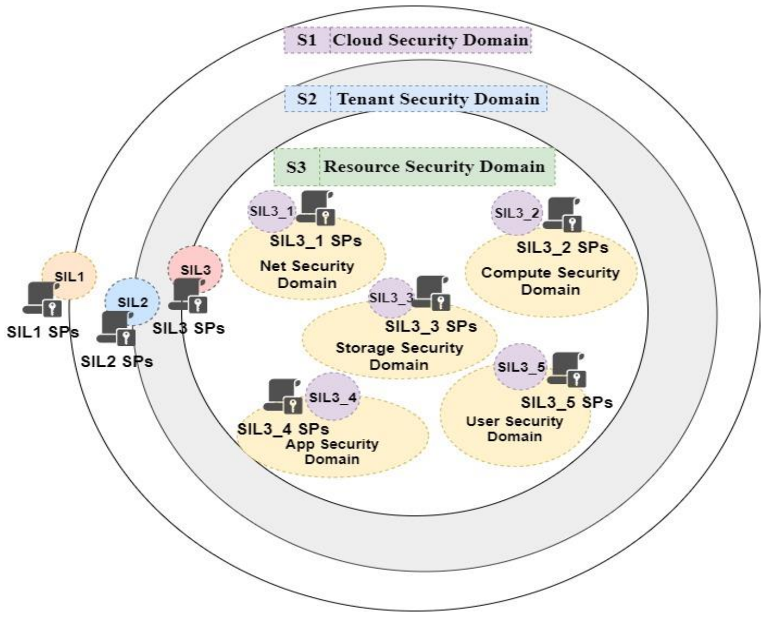


Figure 4. Security Isolation levels

Security policy in our context covers 4 aspects: system interaction policy; time-based security policy; dynamic location-based security policy; and entity-specific constraint policy. System interaction security policies are organisational sets of security policies that dictate allowable object interactions, as specified by valid parameters of an interaction. Time-based security policies dictate the valid time or time duration of an interaction. These policies are often specified at runtime because they are needed when dynamic operational circumstances demand. Location-based security policies are required to deal with dynamic aspects of a cloud entity, such as changes in responsibility and logical/physical zone placement over time. Entity-specific constraint policies deal with the specific nature and properties of an entity. Some entities may not perform some activities because they do not possess the capability, while others are capable but their actions are constrained by relevant policies when they were instantiated. With these definitions, the set of security policies (S_k) relevant to an interaction I^k between E_i and E_j may be expressed by the following equation:

$$S_k\left(I^k,(E_i,E_j)\right)=P_{E_iE_j}\left(L^k(E_i,E_j),t^k\right)=\ P_{E_iE_j}^{L^k,t^k}\qquad where\ i,j\in N\ and\ i\neq j.$$

The notations are defined in Table 3. P denotes the system policy governing the entities, location, and time; L denotes location-based policies for each entity. If E_i, E_j are placed in the same zone and same group zone policy, the location policies are the same for both. The security policy-based interaction model concentrates on two main policy concepts: general policies and local policies. General policies apply to all requests within the system, and local policies apply separately to each entity and their interactions within the system according to their location and assigned constraints. Both sets of policies are stored in separate security databases. Security policies are extracted during an interaction, and rules and constraints are assigned and applied to the interaction over the valid interaction time duration.

Interaction Security Violation Detection and Prediction Algorithm (ISVDP)

With the introduction of the formal model of an interaction and its relationship with security policy, we propose an interaction security violation detection and prediction (ISVDP) algorithm. The ISVDP operates over the SDS$_2$ cloud infrastructure with three levels of security isolation. The algorithm automatically detects and predicts security breaches in relation to a requested interaction according to valid/invalid interaction parameters. The main parameters of ISVDP include:

- Initiator entity: an entity that initiates a relationship with another entity and establishes an interaction;
- Target entity (or Reactor): the entity of an interaction on which the initiator intends to perform certain actions;
- Entity's constraints: the constraints extracted from local policies related to both initiator's and target's role, type, and their intrinsic properties;
- A complete set of system security policies defined over the SDS$_2$ cloud and its isolation levels: Cloud, Tenant, and Resources;
- A requested interaction between the initiator and the target entities (for violation detection).

In ISVDP, a constraint is represented as "a security statement which defines a set of conditions that limits the scope and the property of an interaction between an initiator and its target entity". High-level security policies are written in human-language policies, which will be translated using a policy-translator within the SDS$_2$ controller. Armed with the translated security policies, a security controller determines the validity of an interaction between entities based on their defined interaction parameters. The detection and the prediction algorithms form two fundamental components of the ISVDP model. Both of them share and are built upon the initial three processing stages, as shown in Figure 5 for a specific interaction k. We define the required notations in Table 3. We define the basic set of operations on an interaction object with these notations in Table 4.

Table 3. Required Notations

Notation	Meaning	Detailed expression
c_{jv}	Set of constraints associated with entity j	
E	An entity composed of role and object i	E= E_i^{jk}
I	An interaction object	
I_{init}^k	Interaction object k initialised with default parameters (unconstrained)	$I_{init}^k(*,*)$
I_C^k	Interaction object k with object constraints applied	$I_C^k(E_i, E_j)$ or $I_C^k\left(E_i(c_{iu}), E_j(c_{jv})\right)$
I_P^k	Interaction object k with system policies applied	$I_P^k(E_i, E_j, S)$
$I_{P,C}^k$	Interaction object k with both system policies and object constraints applied	$I_{P,C}^k(E_i(c_{iu}), E_j(c_{jv}), S)$
I_{req}^k	Interaction object k with parameters derived from an interaction request	$I_{req}^k(E_i, E_j)$
Sk	Interaction k policies derived from the system policies	$S_k = P_{E_iE_j}^{L^k,t^k}$

Table 4. Operations Defined on an Interaction Object

Operation	Meaning	Detailed expression
Initialise (I)	Initialise I with default parameters M, R, A and t	
Mode (I^k)	Return possible modes between Ei and Ej for interaction k	$Mode\ of\ (I_C^k\ or\ I_P^k or\ I_{P,C}^k)$
Relate (I^k)	Return possible positional relations between Ei and Ej for interaction k	$Relate\ of\ (I_C^k\ or\ I_P^k or\ I_{P,C}^k)$
Action (I^k)	Return possible actions between Ei and Ej for interaction k	$Action\ of\ (I_C^k\ or\ I_P^k or\ I_{P,C}^k)$
Const (I^k)	Return possible interaction parameters after applying constraints on interaction k	$Const\ on\ (I_{init}^k)$
State (I^k)	Return all states of interaction k between Ei and Ej	$State\ of\ (I_C^k\ or\ I_P^k or\ I_{P,C}^k)$
State (I^k, req)	Return all states of the interaction, as required by the request	$State\ of\ (I_{req}^k)$
Policy (L, E_i)	Returns the set of system policies applied to entity i location	
Policy (I^k, req)	Return the set of system policies applied to interaction k	$Policy\ on\ (Ik\ or\ I_{req}^k)$
Compare (I_m, I_n)	Compare the states of interaction m and interaction n, return differences in M, R, A, and t	
Opposite (I^k)	Returns set of possible violated mode parameters extracted from valid interactions	

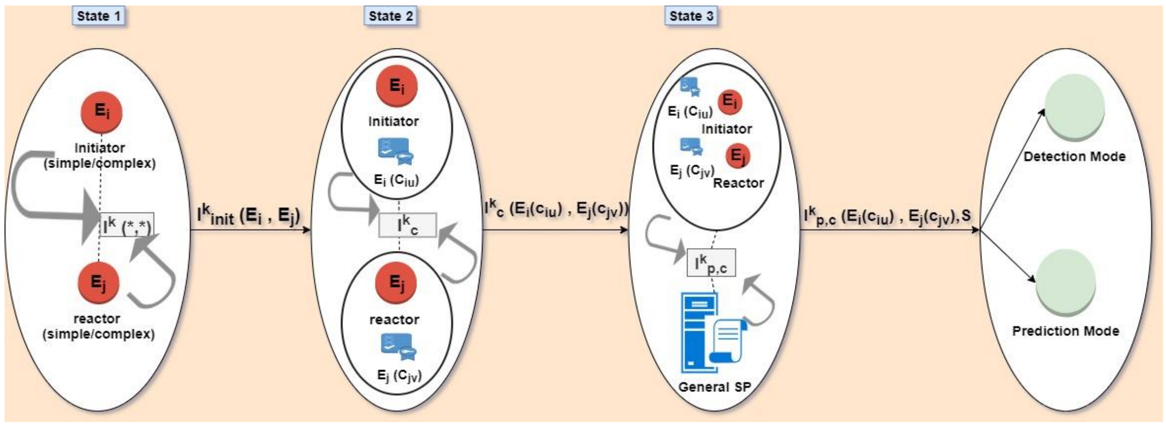


Figure 5. ISVDP stages

Stage 1: Initialising the interaction. At this initial stage, the objects involved in the interaction are made available with their security-rated properties. The interaction template is initialised with no constraints on interaction parameters. The requested interaction is also made available. The result of this stage is the object $I_{init}^{k}(*,*)$. The algorithm intelligently identifies all involved entities and components in this stage. Additionally, the algorithm detects the interaction type and parameters. Dynamically it can change interaction parameters according to the location and nature of entities and create initial interaction parameters between two entities.

Stage 2: Application of entity constraints over the interaction k. At this stage, the interaction parameters are modified according to the properties and constraints of the entities involved. The result of this stage is the object $I_c^k\left(E_i(c_{iu}), E_j\left(c_{jv}\right)\right)$.

Stage 3: Application of the policy over the interaction k. At this stage, the parameters of interaction k will be modified by the constraints derived from the system policies that apply to interaction k. The result of this stage is the object $I_{p,c}^{k}\left(E_i(c_{iu}), E_j\left(c_{jv}\right), P\right)$. The policy-driven interaction algorithm encompassing stages 1, 2, and 3 is shown in Algorithm 1.

Algorithm 1. Policy-driven interaction (PdI ())

```
Input: Ei, Ej, SDS2 cloud objects' DB, System Policy statement (P)
Output: $I_{p,c}^{k}$ (M', R', A', t')
1: while request is valid do
    2: for Ei, Ej do
      3: Intialize (Ik) //get interaction parameters for $E_i, E_j$ without applying constraints and set $I_{init}^{k}$
      4: if $I_{init}^{k} \neq$ Null
           Const ($I_{init}^{k}$)  //get interaction parameters by applying constraints on $I_{init}^{k}$ and set $I_C^k$
         5: $I_C^k$ = State ($I_C^k$) // return parameters after applying constrains
        6:  Policy (Ik) // get system policies (P) applied to the Ik
        7: $I_{p,c}^{k}$ = State (Ik) // return parameters after applying policy system
      8: end if;
    9: end for;
10: end while;
```

Interaction Security Violation Detection

Consider an interaction within a cloud system. The detection algorithm determines if the interaction is safe or violates the system's security policy or, specifically, if a security breach has occurred. With the global knowledge of the cloud environment and the interactions among entities, the security controller intelligently schedules the execution of the Interaction Security Violation Detection (ISVD) algorithm on suspicious circumstances, on a specific request or triggered events, or on a regular basis. The algorithm considers each interaction parameter under consideration to discover if any inconsistency has occurred relative to the security policies, and hence the interaction parameters dynamically applicable to the interaction. The module goes through the three fundamental stages as described above and proceeds to stages 4d, 5d, 6d and 7d for violation detection as follows.

Stage 4d: The requested interaction policy level is analysed according to defined security isolation levels explained for the security policy-based interaction model *(Domain* (I^k_{req})*).*

Stage 5d: The interaction under consideration between the specified objects is analysed, resulting in a set of interaction statuses required by the request: $I^k_{req}(E_i, E_j)$.

Stage 6d: The algorithm intelligently detects each object interaction parameter rule based on security domain and location Domain, $(I^k_{req}(E_i, E_j))$ and Loc (E_i, E_j).

Stage 7d: By analysing $I^k_{p,c}\ (E_i(c_{iu}), E_j(c_{jv}), P)$, and $I^k_{req}(E_i, E_j)\ interaction\ parameters$, the algorithm determines if requested actions are within the set of actions allowable by the policies and constraints imposed on the entities of the interaction.

The algorithm returns the validation status of the interaction: either Safe or Violate. "Safe" means that the requested interaction does not violate any policy related to any interaction parameter and is not a security breach. "Violate" means that the requested interaction violates one of the parameters (M, R, A, t) or location of the allowed security policy that governs the interaction. The algorithm returns whenever a violation of an interaction parameter is detected. However, in cases where policies governing the interaction parameter are undefined (either due to an oversight or situations not yet encountered), it will decide if there is a possibility to partially accept the interaction and initiate an alert for the decisionmaker to create a new policy to cover the newly discovered situation.

Figure 6 shows the decision process of the ISVD algorithm which determines interaction states using the ISVD algorithm. We use (M', R', A', t') to denote State $(I^k_{p,c})$ and (M'', R'', A'', t'') to denote State (I^k_{req}). In the detection process, all system policies, including location, and entity constraints are applied to the interaction k to obtain all the allowable parameters of the interaction. Figure 6 shows the detection approach in determining the validation status of the requested interaction k. The detection algorithm will stop the process on discovering the first interaction parameter violation and activate a security alarm within the security controller. Algorithm 2 describes the ISVD detection algorithm, which analyses the I^k_{req}, extracting number and types of involved entities during the requested interaction. Figure 6 demonstrates the state that an interaction will be considered to be in after using the ISVD algorithm. The results are based on two main conditions defined as equal (=) (where each specific interaction parameter of $I^k_{p,c}$ is equal to its I^k_{req} interaction parameter) and not equal (!) (where interaction parameters of $I^k_{p,c}$ are not the same as I^k_{req}).

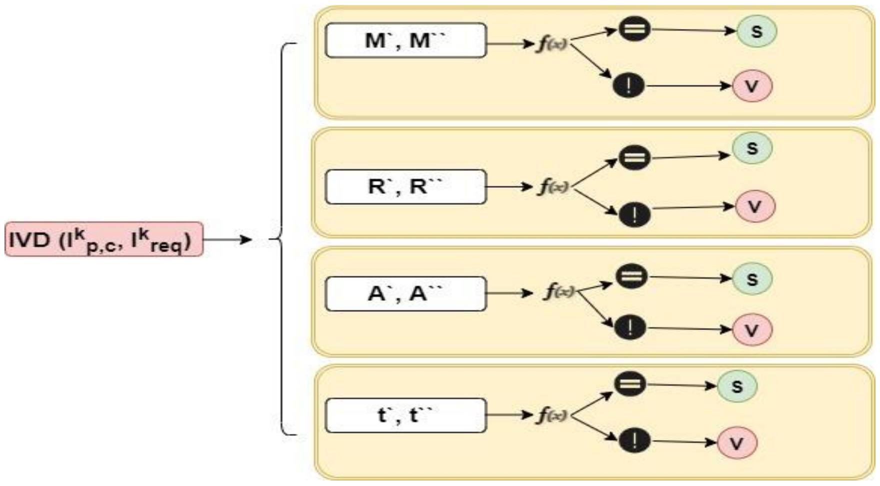


Figure 6. ISVD algorithm

Algorithm 2. Interaction Security Violation Detection (ISVD ())

```
Input: $I^k_{req}$(requested interaction), $I^k_{p,c}$ ,
Output: Safe | Violate
1: for received $I^k_{req}$ do
  2: if Domain ($I^k_{req}$)==True then
 3: Policy (L, E) //Returns set of system policies on the current location of entities during the initiation of $I^k_{req}$ and set L
   4: If L == True (location is verified) then
     5: PdI () //call the algorithm 1 to get $I^k_{p,c}$
        $I^k_{P,C}$ = State ($I^k_{P,C}$)
     6: $I^k_{req}$ = State (Ik, req) // get requested interaction parameters
   7: for $I^k_{req}$ and $I^k_{P,C}$ do
      8: diff := Compare ($I^k_{P,C}$, $I^k_{req}$) // returns difference (diff) parameters between $I^k_{P,C}$ and $I^k_{req}$
      9: P := Policy (Ik, req) // returns system policies applied to interaction parameters
      10: if diff satisfies P then
         11: state ( $I^k_{req}$) is safe
         12: else state ( $I^k_{req}$) is Violate //rise security violation alarm to security controller, isolate the interaction
      13:   end if;
   14:  end for;
  15:  end if;
 16: end if
17: end for;
```

Interaction Security Violation Prediction

In this section, we describe the prediction algorithm and its functionality. The Interaction Security Violation Prediction (ISVP) algorithm enables interaction violation predictability based on permissible values of the parameters of the interaction. The algorithm is different from the detection algorithm in that it determines all "Safe" interactions and all potential "Violate" interactions under the system security policies and constraints imposed on the interaction parameters between given entities. The prediction algorithm automatically discovers the probability of a possible future violation according to the current state of validation interaction parameters. For each interaction parameter, it discovers upcoming violation values. It analyses the state of entities' interaction and predicts future violations according to unacceptable interaction parameters within the system. The prediction algorithm considers each interaction parameter and determines the invalid parameter values through prediction approaches. The prediction algorithm proceeds through the three stages of Algorithm 1 and proceeds through stages 4p and 5p, as shown in Algorithm 3.

Stage 4p: The stage outputs all possible "Safe" interaction parameters between the given entities considering all constraints and security policies.

Stage 5p: The outputs are all potential "Violate" interactions between the given entities.

This is done by inspecting each parameter (M, R, A, t) and applying security constraints on each parameter. If M_s (safe parameters defined for M during k interaction) is the allowed set of safe modes, then $M_v = M - M_s$ is the set of violate modes (M is all possible values). Similarly, R_s and R_v are the set of allowed relational positions and violate relational positions, respectively; A_s and A_v are the set of allowed actions and violate actions, respectively. Similar notations are used for time and the location. The results allow the system to predict possible security breaches if interaction parameter conditions are not met. These conditions display the predicted violations in terms of interaction parameters.

Ideally, all possible violations relative to the current interaction can be discovered/predicted; however, if all the interaction parameters are allowed to vary independently of one another, the analysis can be computationally expensive and not practicable. Realistically, we may want to address and predict most likely violations. We thus restrict ourselves to simple situations where one parameter varies at a time, just to illustrate the prediction process. In the predicting state, the system anticipates all possible different situations that current interaction parameters between defined entities can face. For instance, if the valid actions between two objects are defined as "read", all other possible actions can be considered violations of interaction parameters considering the object nature and constraints. So, the system can stop the violation using its stored predicted violation parameters rather than going through lower layers and nested policy discovery. In the presented prediction algorithm, all opposite interaction parameters against validation parameters are considered potential interaction parameter violations. The security controller runs the ISVDP algorithm to discover the probability of future attacks according to each interaction parameter for an interaction, say *k*. It is an intelligent mechanism that focuses on interaction parameters and their possible forthcoming violation during an interaction.

Algorithm 3 Interaction Security Violation Prediction (ISVP ())

```
Input: $I^k_{p,c}$
Output: V set of possible potential violate interaction parameters
1: PdI () // set $I^k_{p,c}$
2: for $I^k_{p,c}$ do
  3: while $T_M$ = Mode ($I^k_{p,c}$) do  // get all possible sets of M extracted from $I^k_{p,c}$ from safe mode
       $V_M$= opposite ($T_M$) // set of possible violated mode parameters extracted from valid ($T_M$)
  4: end while;
  5: while $T_M$ = Relate ($I^k_{p,c}$) do  // get all possible sets of R extracted from $I^k_{p,c}$ from safe mode
       $V_R$ = opposite ($T_R$)// sets of possible violated R from $I^k_{p,c}$ from safe mode
  6: end while;
  7: while $T_A$ = Action ($I^k_{p,c}$) do  // get all possible sets of A extracted from $I^k_{p,c}$ from safe mode
       $V_A$ = opposite ($T_v$)  // sets of possible violated A from $I^k_{p,c}$ from safe mode
  8: end while;
9: V = ($V_M$, $V_R$, $V_A$)  // set of predicted and possible violation interaction parameters according   $I^k_{p,c}$
10:  end for;
```

Interaction Scenarios and Results

In this section, we demonstrate our policy-driven security scheme by using a security controller in verifying allowable interactions and detecting policy violations between entities in a cloud infrastructure based on our proposed model of interaction. We built the security controller from scratch in Java language and run our ISVDP algorithm in an Ubuntu machine with 16 GB RAM, Intel® Core (TM) i7-7600U CPU. We set different scenarios according to various interaction types, and analyse the results to evaluate the proposed interaction model and its components for each case. We simulate the interaction between different types of objects within the system to detect and predict security violations according to our ISVDP model. We consider the CloudSimSDN-NFV framework (Son, He & Buyya, 2019) to simulate the cloud infrastructure and build our security controller and its ISVDP algorithm. Figure 7 demonstrates the implementation process.

Scenario 1: User interaction. In this scenario, the security controller (SC) receives interactions triggered by a user. The SC identifies the user and interprets the request for an interaction. According to the user level and rights, the security controller determines the security policies related to the user and involved objects. The requested interaction is sent to the interaction security domain controller to extract security policies and interaction parameters. The security controller then initiates a virtual security function (VSF) designed to monitor the interaction based on the received validated interaction, entities' policies, and constraints. The analyser function is responsible for running the ISVDP algorithm to detect and predict security violations.

Scenario 2: Specific requested interaction. In this scenario, a specific interaction runs within the system. The specific interaction is considered as a request to monitor a specific interaction being performed by the security controller. This scenario occurs when the security controller decides to monitor an interaction between specific entities within the system. The security controller triggers an interaction to be monitored among specific entities. It will happen mainly in two sub-scenarios: 1) randomly monitor an entity based on its statistics received from its virtual security functions; 2) activate a scheduled monitoring of a sensitive entity within the system in specific time slots.

Scenario 3: Triggered interaction. The security controller activates a virtual security function to monitor a triggered interaction. This scenario occurs when an abnormal interaction is triggered between entities within the system. The security controller initiates and commands reports from relevant virtual security functions over suspicious entities and then executes the ISVDP algorithm to assess the situation.

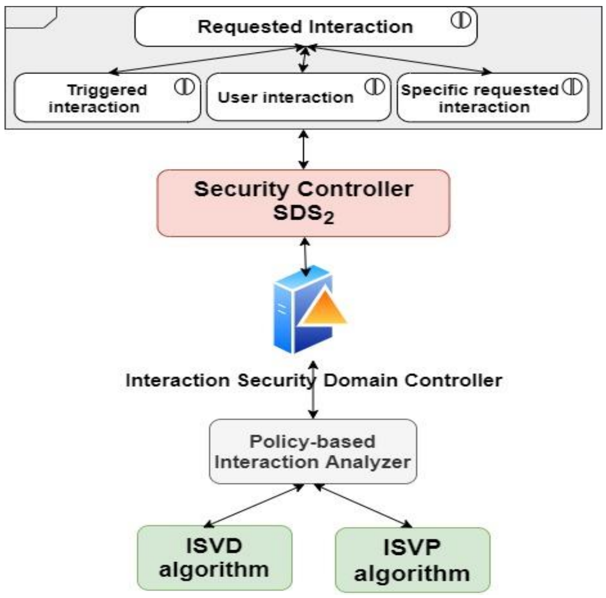
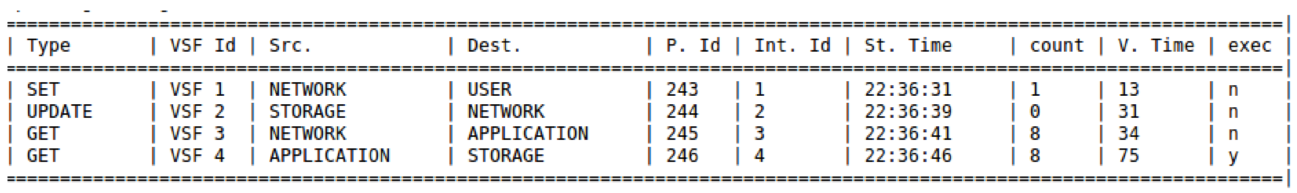


Figure 7. Implementation process

The functions within the security controller perform the ISVDP algorithms and produce the results. As demonstrated in Figure 8, the security controller analyses the requested interaction and involved objects. Security policies and objects' constraints are extracted according to object security isolation layers and entities' location. In this paper, however, we mainly concentrate on the interaction between simple objects and their interactions. Figure 5 shows the flow process of our ISVDP algorithms. In the first step, the security controller creates entities within the cloud system by substantiating the identified objects together with their defined role. In Figure 8 below, the security controller, after receiving the interaction, triggers and assigns a specific VSF to monitor each interaction. It extracts required policies for each interaction and assigns a unique policy ID (P.Id). It labels each interaction based on specific interaction ID (Int.Id) and extract validate time (V. Time) for each interaction as well. Each VSF lifecycle starts according to triggered requested interaction.



```
====================================================================================================|
| Type       | VSF Id | Src.            | Dest.           | P. Id | Int. Id | St. Time     | count | V. Time | exec |
====================================================================================================|
| SET        | VSF 1  | NETWORK         | USER            | 243   | 1       | 22:36:31     | 1     | 13      | n    |
| UPDATE     | VSF 2  | STORAGE         | NETWORK         | 244   | 2       | 22:36:39     | 0     | 31      | n    |
| GET        | VSF 3  | NETWORK         | APPLICATION     | 245   | 3       | 22:36:41     | 8     | 34      | n    |
| GET        | VSF 4  | APPLICATION     | STORAGE         | 246   | 4       | 22:36:46     | 8     | 75      | y    |
====================================================================================================|
```

Figure 8. Extracting involved objects and assigning a monitoring security function to each interaction

We consider cloud objects of different types and determine possible allowable interactions. For each scenario, objects can be at the same or different access levels. System policies are applied to achieve validate interaction parameters once an interaction is triggered. For simplicity, interaction time is assumed valid throughout the whole time under consideration. We detail below an interactive case study between two entities to describe the process of discovering and validating the interaction between the two entities. In the following case, we describe how a policy-based interaction analyser will extract required interaction parameters to be sent to the assigned security function.

VM-Storage interaction: interaction between a virtual machine and a storage entity. In the first step, the program at the controller level creates participating entities (if they have not existed yet) based on the information stored in the security database, SecDB. Then, the security controller analyzes the interaction and extracts policies applied to the requested interaction between the two entities according to each entity and their level within the cloud infrastructure.

As depicted in Table 5, the second and third columns show values of the VM and the storage entities' interaction parameters after their constraints have been applied. The fourth column shows the interpretation of the system policy on the objects' interaction parameters. The last column shows all possible interactions between the two entities as determined by the allowable interaction parameters after all constraints and system policies are considered. The results indicate that the only allowable actions are Re, W in an allowable pair of (m_1, d_2) mode of interaction between the VM and the cloud system's storage entities. In our program, we consider *t* as an acceptable duration time over which an interaction can take place. For violation detection, the security controller calls Algorithm 2. It analyses the incoming request and extracts the required parameters and call $State\ \ (I^k_{req})$. During this phase, the requested interaction statement requests the *removal of a file from the storage object requested by the virtual machine at the same level.*

Table 5. Collected data from the controller for VM-Storage interaction

	I (VM)	I(Storage)	SysPolicy	$I^k_{p,c}$
M (m/d)	(m1, d1)	(m1, d1)	(m1, d2)	(m1, d2)
r	Cloud	Cloud	cloud	cloud
A	Re, W, D	Re, W	Re, W, Cr, D	Re, W
t	600 ms	600 ms	300 ms	300 ms

The program translates the incoming request, which detects the *delete* violation as a delete action against $I^k_{p,c}$. It raises a security alarm, indicating a violation by the requested interaction. For violation prediction against possible attacks, the system will call Algorithm 3 to predict possible violations against the parameters of $I^{k}_{p,c}$. The system calculates possible violation parameters relative to allowable $I^{k}_{p,c}$ parameters. In this case, interaction actions except for Re, W are considered as action violations. More importantly, this algorithm can enumerate all possible interaction violations between two entities (those not allowable by $I^{k}_{p,c}$) by systematically going through the mode, the positional relationship, the action, and the interaction's time parameters. As an example, if we keep all parameters except the mode parameters fixed, we can declare that other modes except m_1 and d_3 are potential (or predicted) violations. Similarly, the system considers any positional relation except *cloud* as a security breach and stores the data. An insider/outsider request that involves any of the predicted violation parameters will be investigated in anticipation of potential security breaches: *ISVP* $(I^{k}_{p,c}) \rightarrow V\,(V_M, V_R, V_A, V_t)$.

We executed various tests according to various scenarios to show expected results. Table 6 reveals some result samples that the security controller captured by performing many cases. The results are simulation results we captured by running various simulated cases to test our security algorithm. Table 6 demonstrates extracted parameters for each scenario through running our security algorithm.

In the table *Int* reveals validated parameters expected after running the PdI () algorithm. After running the ISVD () algorithm, it shows the results using *Act* parameter (s: safe, v: violate). We monitored our security controller performance according to the number of interactions triggered within the system from any resources, the detection processing time, and the time until the system detects the status of the requested interaction. As explained before, each requested interaction monitors by a specific VSF with unique identity (VSF Id) to be distinguished by the security controller. Figure 9 illustrates the SDS_2 performance using detection processing time in the face of different interactions. In the figure, the average processing time increased as the number of interactions received by the security controller increased.

Table 6. Expected results of the simulated scenarios

VSF ID	Src.	Res.	Int. Init	Int.				Act.	P. Id	exec
				M	**R**	**A**	**t**			
VSF 6	VM	Storage	SC	(m_1, d_3)	Cloud	Re, W	3000 ms	s	3	Y
VSF 3	User	Storage	SC	(m_4, d_3)	Tenant	Re, W	900 ms	v	3	N
VSF 2	Storage	APP	UR	(m_1, d_1)	Cloud	Md	10500 ms	v	22	Y
VSF 9	User	App	AT	(m_4, d_2)	Resource	Md	1000 ms	v	23	Y
VSF 7	VM	Storage	SC	(m_2, d_2)	Tenant	Re, W	800 ms	s	30	Y
VSF 11	App	Storage	SC	(m_5, d_2)	Resource	Re, W	600 ms	v	13	N
VSF 8	Net	VM	UR	(m_4, d_1)	Tenant	Ex, Re	600 ms	s	19	N
VSF 22	Storage	VM	AT	(m_1, d_2)	Cloud	Re	300 ms	v	22	N

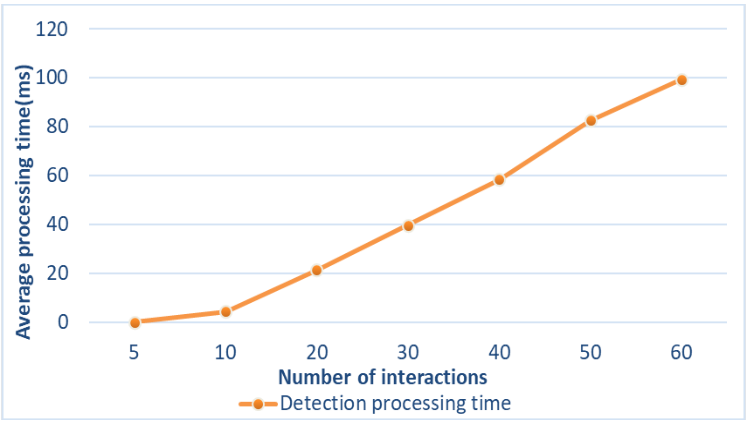


Figure 9. Performance monitoring according to interaction detection processing time

In this scenario, the security controller only considers simple interactions between two entities, while each interaction is initiated with pre-defined conditions, to test the SDS_2 security controller performance according to simple interaction types with pre-defined security policies.

Conclusion

This paper has taken a novel approach with the proposed Policy-based Interaction Model, to provide isolation within the cloud infrastructure. The proposed model introduced a dynamic construction of security boundaries based on our constructed interaction model and its parameters. To secure cloud resources, an intelligent security algorithm has been developed to provide proactive detection and prediction in relation to the interaction parameters. Security policy rules pertaining to entities and their location are further applied to the interaction parameters to determine the overall validity of the participating entities' interaction.

The policy-driven interaction model is, to the best of our knowledge, the first in a new direction for combatting security incidents systematically. A possible next step is to deploy the proposed SDS_2 in a real cloud scenario to detect and predict cloud security violations using new technologies, Software-defined Networking and Network Function Virtualisation.

References

Barjatiya, S., & Saripalli, P. (2012). Blueshield: A layer 2 appliance for enhanced isolation and security hardening among multi-tenant cloud workloads. Paper presented at the Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing.

Basile, C., Valenza, F., Lioy, A., Lopez, D. R., & Perales, A. P. (2019). Adding Support for Automatic Enforcement of Security Policies in NFV Networks. *IEEE/ACM Transactions on Networking, 27*(2), 707-720. http://doi.org/10.1109/TNET.2019.2895278

Cai, F., Zhu, N., He, J., Mu, P., Li, W., & Yu, Y. (2018). Survey of access control models and technologies for cloud computing. *Cluster Computing*, *22*, 6111–6122. https://doi.org /10.1007/s10586-018-1850-7

Chen, C., Li, D., Li, J., & Zhu, K. (2016). SVDC: A Highly Scalable Isolation Architecture for Virtualized Layer-2 Data Center Networks. *IEEE Transactions on Cloud Computing, 6*(4), 1178-1190. http://doi.org/10.1109/TCC.2016.2586047

Damiani, M. L., Bertino, E., Catania, B., & Perlasca, P. (2007). GEO-RBAC: a spatially aware RBAC. *ACM Transactions on Information and System Security (TISSEC), 10*(1), 2.

Del Piccolo, V., Amamou, A., Haddadou, K., & Pujolle, G. (2016). A survey of network isolation solutions for multi-tenant data centers. *IEEE Communications Surveys & Tutorials, 18*(4), 2787-2821. https://doi.org/10.1109/COMST.2016.2556979

Factor, M., Hadas, D., Harnama, A., Har'El, N., Kolodner, H., Kurmus, A., Shulman-Peleg, A., & Sorniotti, A. (2013). Secure logical isolation for multi-tenancy in cloud storage. Paper presented at the 2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST). https://doi.org/10.1109/MSST.2013.6558424

Farahmandian, S., & Hoang, D. B. (2017). SDS 2: A novel software-defined security service for protecting cloud computing infrastructure. Paper presented at the 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA). https://doi.org/10.1109/NCA.2017.8171388

Hoang, D. B., & Farahmandian, S. (2017). Security of Software-Defined Infrastructures with SDN, NFV, and Cloud Computing Technologies. In *Guide to Security in SDN and NFV* (pp. 3-32): Springer.

Jararweh, Y., Al-Ayyoub, M., Darabseh, A., Benkhelifa, E., Vouk, M., & Rindos, A. (2016). Software defined cloud: Survey, system and evaluation. *Future Generation Computer Systems, 58*, 56-74. https://doi.org/10.1016/j.future.2015.10.015

Karmakar, K. K., Varadharajan, V., Tupakula, U., & Hitchens, M. (2016). Policy based security architecture for software defined networks. Paper presented at the Proceedings of the 31st Annual ACM Symposium on Applied Computing. https://doi.org/10.1145/2851613.2851728

Kosiur, D. (2001). *Understanding policy-based networking* (Vol. 20): John Wiley & Sons.

Li, F., Li, Z., Han, W., Wu, T., Chen, L., Guo, Y., & Chen, J. (2018). Cyberspace-Oriented Access Control: A Cyberspace Characteristics-Based Model and its Policies. *IEEE Internet of Things Journal, 6*(2), 1471-1483. https://10.1109/JIOT.2018.2839065

Mavridis, I., & Karatza, H. (2019). Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing. *Future Generation Computer Systems, 94*, 674-696. https://doi.org/10.1016/j.future.2018.12.035

Mundada, Y., Ramachandran, A., & Feamster, N. (2011). SilverLine: Data and Network Isolation for Cloud Services, HotCloud 2011, Portland, OR, USA. Available at https://static.usenix.org/event/hotcloud11/tech/final_files/Mundada6-1-11.pdf

NIST [National Institute of Standards and Technology]. (2015). Information security policy. Committee on National Security Systems Instruction, CNSSI 4009, Glossary. Revised April 6, 2015. US Department of Commerce.

Pfeiffer, M., Rossberg, M., Buttgereit, S., & Schaefer, G. (2019). Strong Tenant Separation in Cloud Computing Platforms. Paper presented at the Proceedings of the 14th International Conference on Availability, Reliability and Security. https://doi.org/10.1145/3339252.3339262

Rajkumar, P.V., & Sandhu, R. (2016). POSTER: security enhanced administrative role based access control models. Paper presented at the Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. https://doi.org/10.1145/2976749.2989068

Son, J., He, T., & Buyya, R. (2019). CloudSimSDN-NFV: Modeling and simulation of network function virtualization and service function chaining in edge computing environments. *Software: Practice and Experience*. https://doi.org/10.1002/spe.2755

Stone, G. N., Lundy, B., & Xie, G. G. (2001). Network policy languages: a survey and a new approach. *IEEE network, 15*(1), 10-21. https://doi.org/10.1109/65.898818

Tarkhanov, I. (2016). Extension of access control policy in secure role-based workflow model. Paper presented at the 2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT). https://doi.org/10.1109/ICAICT.2016.7991691

Varadharajan, V., Karmakar, K., Tupakula, U., & Hitchens, M. (2018). A policy-based security architecture for software-defined networks. *IEEE Transactions on Information Forensics and Security, 14*(4), 897-912. https://doi.org/10.1109/TIFS.2018.2868220

Wang, X., Shi, W., Xiang, Y., & Li, J. (2015). Efficient network security policy enforcement with policy space analysis. *IEEE/ACM Transactions on Networking, 24*(5), 2926-2938. https://doi.org/10.1109/TNET.2015.2502402

Yin, X., Chen, X., Chen, L., Shao, G., Li, H., & Tao, S. (2018). Research of Security as a Service for VMs in IaaS Platform. *IEEE Access, 6*, 29158-29172. https://doi.org/10.1109/ACCESS.2018.2837039